European IT Certification Curriculum
Self-Learning Preparatory Materials

EITC/AI/TFF

TensorFlow Fundamentals

This document constitutes European IT Certification curriculum self-learning preparatory material for the EITC/AI/TFF TensorFlow Fundamentals programme.

This self-learning preparatory material covers requirements of the corresponding EITC certification programme examination. It is intended to facilitate certification programme's participant learning and preparation towards the EITC/AI/TFF TensorFlow Fundamentals programme examination. The knowledge contained within the material is sufficient to pass the corresponding EITC certification examination in regard to relevant curriculum parts. The document specifies the knowledge and skills that participants of the EITC/AI/TFF TensorFlow Fundamentals certification programme should have in order to attain the corresponding EITC certificate.

Disclaimer

This document has been automatically generated and published based on the most recent updates of the EITC/AI/TFF TensorFlow Fundamentals certification programme curriculum as published on its relevant webpage, accessible at:

https://eitca.org/certification/eitc-ai-tff-tensorflow-fundamentals/

As such, despite every effort to make it complete and corresponding with the current EITC curriculum it may contain inaccuracies and incomplete sections, subject to ongoing updates and corrections directly on the EITC webpage. No warranty is given by EITCI as a publisher in regard to completeness of the information contained within the document and neither shall EITCI be responsible or liable for any errors, omissions, inaccuracies, losses or damages whatsoever arising by virtue of such information or any instructions or advice contained within this publication. Changes in the document may be made by EITCI at its own discretion and at any time without notice, to maintain relevance of the self-learning material with the most current EITC curriculum. The self-learning preparatory material is provided by EITCI free of charge and does not constitute the paid certification service, the costs of which cover examination, certification and verification procedures, as well as related infrastructures.

TABLE OF CONTENTS

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: INTRODUCTION TO TENSORFLOW
TOPIC: FUNDAMENTALS OF MACHINE LEARNING

Artificial Intelligence - TensorFlow Fundamentals - Introduction to TensorFlow - Fundamentals of machine learning

Artificial Intelligence (AI) and machine learning have gained significant attention in recent months. You may have seen inspiring videos showcasing the capabilities of AI machine learning. But what exactly is AI? In this educational material, we will delve into the world of AI and explore what it entails to write code for machine learning.

To begin, let's consider a simple example of creating a game of Rock, Paper, Scissors. While this game is easy for humans to learn, programming a computer to recognize the different hand gestures can be challenging. The diversity in hand types, skin color, and variations in how people form the scissors gesture make it a complex task. Traditional programming would require thousands of lines of code to achieve this. However, machine learning offers an alternative approach.

In traditional programming, data from a webcam, for instance, is processed using predefined rules expressed in a programming language. These rules form the bulk of the code and determine the output based on the input data. Machine learning, on the other hand, flips this paradigm. Instead of explicitly defining the rules, we provide the computer with answers and let it figure out the rules from the data.

Machine learning involves training a computer to recognize patterns in data. By showing the computer lots of pictures of rocks, paper, and scissors, we can teach it to identify these objects by finding the patterns that match them. This process of training a computer to recognize patterns is the essence of machine learning.

Before diving into complex tasks like recognizing rock, paper, and scissors, let's start with a simpler example. Consider a set of numbers with a relationship between the X and Y values. By observing the pattern, we can deduce the relationship as Y = 2X - 1. This principle of pattern recognition forms the basis of all machine learning.

To illustrate this concept, we provide a code snippet that creates a machine-learned model for matching these numbers. The code defines a neural network model using TensorFlow's Keras library. This model consists of a single layer with a single neuron. The input to the neural network is the X value, and the network predicts the corresponding Y value. The model is then compiled using a loss function and an optimizer, which are crucial components of machine learning.

During training, the model makes initial guesses about the relationship between the numbers, such as Y = 5X + 5. The loss function quantifies the accuracy of these guesses, allowing the model to learn from its mistakes and improve its predictions over time.

This example demonstrates the fundamentals of machine learning using TensorFlow. By training models to recognize patterns in data, we can enable computers to perform tasks that mimic human intelligence. As you progress in your learning journey, you will explore more advanced concepts and applications of machine learning.

In the context of machine learning, TensorFlow is a powerful open-source library that allows us to build and train artificial neural networks. One fundamental concept in TensorFlow is the optimization process, which involves iteratively adjusting the parameters of a model to minimize the difference between its predictions and the actual data.

To illustrate this concept, let's consider a simple example. Imagine we have a set of data points, each consisting of an input value (X) and an output value (Y). Our goal is to find a mathematical formula that can accurately predict the output value for any given input value. In other words, we want to find a function that maps X to Y.

To achieve this, we can create a neural network model using TensorFlow. Initially, the model will make random guesses for the formula. Then, it will use an optimizer function to generate a new guess based on the difference

between the model's predictions and the actual data. By repeating this process multiple times, the model gradually improves its guess until it reaches a more accurate formula.

In our example, the data is represented as an array of Xs and Ys. The process of matching the input and output values is performed by the "fit" method of the model. By calling this method and specifying the number of iterations (in this case, 500), we train the model to find the best formula that fits the given data.

After training the model, we can use it to predict the output value (Y) for a given input value (X). However, it's important to note that the model's predictions may not always be exact. In our example, if we try to predict the output value for X=10, we might expect the answer to be 19. However, due to the limited training data, the model's prediction may be slightly different, such as 18.9998. This discrepancy occurs because the model has only been trained on a small number of data points, and its ability to generalize to new values is limited.

In machine learning, it is common to encounter situations where the model's predictions are not exact but rather close approximations. This is because the model is making probabilistic predictions based on the available training data. While there is a high probability that the relationship between X and Y is a straight line, we cannot be certain. Therefore, the model's prediction reflects this uncertainty by providing a value very close to the expected result.

To further explore and experiment with this concept, you can try running the provided code using the link provided in the description. This will allow you to see firsthand how the model's predictions vary for different input values.

TensorFlow provides a powerful framework for building and training machine learning models. By iteratively optimizing the model's parameters, we can improve its predictions and find the best formula that fits the given data. While the model's predictions may not always be exact, they are close approximations that take into account the inherent uncertainty in the data.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: INTRODUCTION TO TENSORFLOW
TOPIC: BASIC COMPUTER VISION WITH ML

In this didactic material, we will explore the fundamentals of TensorFlow, specifically focusing on basic computer vision with machine learning. We will learn how to train a computer to recognize different objects using the Fashion MNIST dataset.

Machine learning is a field of study that involves teaching computers to learn from data and make predictions or decisions without being explicitly programmed. In the previous episode, we saw a simple example of machine learning, where a computer learned to match numbers to each other through trial and error using Python code.

In this episode, we will take the concept of machine learning further by teaching a computer how to see and recognize different objects. For example, we will look at pictures and determine how many shoes are present. While it may seem easy for us to identify shoes, it becomes challenging to explain the concept to someone who has never seen shoes before.

To overcome this challenge, we can train a computer using a dataset called Fashion MNIST. This dataset consists of 70,000 images in 10 different categories, including shoes. By showing the computer thousands of images of shoes, it will learn to recognize the common features that make a shoe a shoe.

The images in the Fashion MNIST dataset are small, only 28x28 pixels. Despite their size, they still contain enough information for a computer to recognize different items of clothing. For example, even in a small image, we can still identify a shoe.

To train the computer to recognize items of clothing based on the Fashion MNIST dataset, we will use code that is similar to what we used in the previous video. TensorFlow provides a convenient way to load the Fashion MNIST dataset into our code. The dataset consists of 60,000 training images and 10,000 test images.

Each image in the dataset is associated with a label, which represents the class of clothing it belongs to. For example, the label "09" indicates an ankle boot. Using numbers as labels instead of text helps computers process the data more efficiently and avoids bias towards any specific language.

Before we dive into the code, let's explore the input and output values of our neural network. Our neural network is more complex than the one we discussed in the previous episode. The input layer has a shape of 28x28, which matches the size of our images. The output layer has 10 nodes, representing the 10 different items of clothing in our dataset.

To understand the purpose of the number 128 in our code, let's think of it as 128 functions, each with its own parameters. These functions take in the pixels of the shoe image and output a value. The goal is for the combination of these functions to output the correct label for the shoe image. The computer will adjust the parameters of these functions to optimize the results and extend this learning to all the other items of clothing in the dataset.

In machine learning, we use an optimizer function and a loss function. The optimizer function helps update the parameters of the functions to improve the results, while the loss function measures how good or bad the predictions were compared to the actual labels.

Additionally, we have activation functions in our neural network. The first activation function, called "relu" or rectified linear unit, filters out values that are less than or equal to zero. The second activation function, called "softmax," selects the largest value from a set of numbers.

By training our neural network using the Fashion MNIST dataset and optimizing the parameters of the functions, we aim to teach the computer to recognize different items of clothing.

This episode introduces the concept of training a computer to recognize objects using machine learning and TensorFlow. We explored the Fashion MNIST dataset, which contains images of different clothing items. By training a neural network with this dataset, we can teach the computer to identify various items of clothing. The

code used in this episode is similar to what we learned in the previous video, showcasing the power and versatility of TensorFlow's programming API.

In the previous material, we discussed the concept of softmax, which is a function used in machine learning to assign probabilities to different classes. Instead of searching for the largest probability, softmax sets the highest probability to 1 and the rest to 0. This simplifies the process of finding the most likely class.

Training a model is a straightforward process. We fit the training images to the training labels. In this case, we will train the model for just 5 epochs. It's important to note that there are additional images and labels that we didn't use for training. These unseen images can be used to evaluate the performance of our model. We can pass these test images to the evaluate method to assess how well our model performs.

To make predictions on new images, we can use the model.predict function. This allows us to obtain predictions from our trained model. By following these steps, we can teach a computer how to see and recognize images.

If you would like to try this out for yourself, you can access the notebook provided in the description (https://colab.research.google.com/github/lmoroney/mlday-tokyo/blob/master/Lab2-Computer-Vision.ipynb).

One limitation of the current approach is that it assumes the images are always 28x28 grayscale, with the item of clothing centered. However, in real-world scenarios, we often encounter normal photographs with various contents and compositions. To address this, we can use convolutional neural networks (CNNs) and the process of spotting features. CNNs are a powerful tool for image recognition and can handle more complex scenarios where the subject is not centered or is part of a larger scene.

In the next material, we will delve into the topic of convolutional neural networks and explore how they can be used to overcome the limitations we discussed. Make sure to stay tuned and hit the subscribe button!

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: INTRODUCTION TO TENSORFLOW
TOPIC: INTRODUCING CONVOLUTIONAL NEURAL NETWORKS

In this material, we will introduce the concept of convolutional neural networks (CNNs) in the context of artificial intelligence and TensorFlow. CNNs are a type of deep neural network that can overcome limitations in basic computer vision tasks.

Previously, we learned about basic computer vision using a deep neural network that matched pixels of an image to a label. However, this approach had limitations. The image had to have the subject centered and be the only thing in the image. To overcome these limitations, we use convolutional neural networks.

The idea behind a convolutional neural network is to filter the images before training the deep neural network. By filtering the images, we can bring forward the features within the images and use them to identify objects. A filter in a CNN is simply a set of multipliers. Each pixel in the image is multiplied by its respective filter value and the neighboring pixels are also multiplied accordingly. The results are then summed up to obtain the new value for the pixel.

Filters in a CNN can be learned over time. As the image is fed into the convolutional layer, a number of randomly initialized filters pass over the image. The results are then fed into the next layer, where matching is performed by the neural network. Over time, the filters that give the best matches will be learned, and this process is called feature extraction.

Pooling is another important concept in CNNs. It groups up the pixels in the image and filters them down to a subset. For example, max pooling groups the image into sets of 2x2 pixels and simply picks the largest value. This reduces the size of the image while maintaining the important features.

The combination of filtering and pooling allows CNNs to extract and enhance features in an image. The filters are learned and can be specific to certain objects or patterns. By stacking multiple convolutional layers, we can break down the image and learn from very abstract features.

To build a convolutional neural network in TensorFlow, we can use similar code to what we used before. The main difference is the addition of a convolutional layer on top of the flattened input. This layer takes the input and generates multiple filters that are multiplied across the image. Each epoch, the network figures out which filters give the best signals to match the images to their labels.

In the next video, we will explore more complex images and see how CNNs can be applied. But before that, we encourage you to try out the provided notebook to see convolutions in action. The link to the notebook can be found in the description below.

Thank you for reading and don't forget to hit that subscribe button to stay updated on our series.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: INTRODUCTION TO TENSORFLOW
TOPIC: BUILDING AN IMAGE CLASSIFIER

In this didactic material, we will explore the process of building an image classifier using TensorFlow. We will start by revisiting the problem of recognizing hands of different shapes, sizes, ethnicities, decorations, and more, as discussed in the first episode of this series. We will then learn how to train a neural network to solve this problem.

To begin, we need a dataset of rock, paper, and scissors poses. We can download a zip file containing the training set and another file containing the testing and validation set. Using the zip file library in Python, we can extract the images to a temporary directory. The images are organized into folders based on their categories, which will serve as labels for our training data.

Next, we create an image data generator that generates training data from the downloaded directory. This generator will automatically label the images based on their parent directory, saving us the effort of manually creating labels. We can create a similar generator for the test set as well.

Now let's define our neural network. The network architecture is similar to what we saw in the previous video, but with additional layers. Since the images are more complex and larger than before, our input size is now 150x150 pixels. The output layer consists of three neurons, corresponding to the three classes: rock, paper, and scissors. The code for the neural network definition is provided in the material.

After defining the network, we compile it using the appropriate code. We can then train the model using the model.fit function. It's worth noting that we don't need to provide labels explicitly because we are using the image data generator, which infers the labels from the parent directories of the training and validation datasets.

During training, we may encounter a phenomenon called overfitting, where the model becomes very good at recognizing what it has seen before but struggles to generalize to new data. To mitigate overfitting, we can employ techniques like image augmentation. The material includes code for image augmentation, which you can try out yourself to observe its impact on overfitting.

Once the model is trained, we can use the model.predict function to make predictions on new images. The code provided in the material demonstrates how to reformat an image to the appropriate size and obtain a prediction from the model. Several examples of successful predictions are included in the material.

To further explore and experiment with the image classifier, a link to a notebook containing the code is provided in the description of the material. By following the instructions in the notebook, you can train your own neural network to recognize rock, paper, and scissors images.

This series of videos has introduced the concept of machine learning and demonstrated its application in computer vision. By building an image classifier using TensorFlow, we have gained insight into the programming paradigm of machine learning and set ourselves on the path to becoming Artificial Intelligence Engineers.

Artificial Intelligence (AI) has revolutionized various fields, including computer vision. One powerful tool for implementing AI in computer vision tasks is TensorFlow, an open-source machine learning framework developed by Google. In this didactic material, we will introduce TensorFlow and explore the process of building an image classifier using this framework.

TensorFlow is widely used in the AI community due to its flexibility and scalability. It allows developers to create and train deep neural networks efficiently. These networks can learn from large datasets and make predictions on new, unseen data. TensorFlow provides a high-level API called Keras, which simplifies the process of building and training neural networks.

To build an image classifier using TensorFlow, we need to follow a series of steps. First, we need a labeled dataset of images. This dataset should contain images belonging to different classes or categories. For example, if we want to build a classifier to distinguish between cats and dogs, we would need a dataset of cat images and a dataset of dog images.

Once we have our dataset, we can start building our image classifier. We begin by importing the necessary libraries, including TensorFlow and Keras. Next, we define the architecture of our neural network. This architecture consists of layers of interconnected nodes, also known as neurons. Each neuron performs a mathematical operation on the input data and passes the result to the next layer.

In our image classifier, the input layer receives the image data, which is typically represented as a matrix of pixel values. The subsequent layers perform operations such as convolution, pooling, and activation functions to extract meaningful features from the images. These features are then fed into a fully connected layer, which maps them to the corresponding classes.

After defining the architecture, we compile the model by specifying the loss function, optimizer, and evaluation metrics. The loss function measures the discrepancy between the predicted outputs and the true labels. The optimizer adjusts the neural network's parameters to minimize this discrepancy during training. The evaluation metrics provide insights into the model's performance.

Once the model is compiled, we can train it using our labeled dataset. During training, the model iteratively adjusts its parameters based on the provided examples. This process involves forward propagation, where the model makes predictions on the training data, and backward propagation, where the model updates its parameters based on the calculated errors.

After training the model, we can evaluate its performance on a separate test dataset. This evaluation helps us assess how well the model generalizes to unseen data. We can also use the trained model to make predictions on new images.

Building an image classifier using TensorFlow opens up a wide range of possibilities. It allows us to solve complex computer vision problems, such as object recognition, image segmentation, and even more advanced tasks like image generation. TensorFlow's versatility and extensive community support make it an invaluable tool for AI practitioners.

TensorFlow is a powerful framework for implementing AI in computer vision tasks. By following a series of steps, we can build an image classifier using TensorFlow and train it to recognize different classes of images. This process involves defining the neural network architecture, compiling the model, training it on labeled data, and evaluating its performance. TensorFlow's flexibility and scalability make it a popular choice among AI researchers and developers.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: NEURAL STRUCTURED LEARNING WITH TENSORFLOW
TOPIC: NEURAL STRUCTURED LEARNING FRAMEWORK OVERVIEW

Neural networks have become a powerful tool in machine learning, with applications in computer vision, language understanding, and classification. In this educational material, we will introduce you to a new learning framework called neural structured learning. This framework allows neural networks to learn with structured signals, improving model quality and robustness.

The neural structured learning framework is designed to address the challenge of incorporating structured information into neural networks. Consider the task of classifying an image as either a cat or a dog. The image is fed into the neural network, activating neurons layer by layer, leading to a classification decision. However, what if there are other similar images related to the input image? In this case, there is a structure, such as a graph, representing the similarity among these images. The neural structured learning framework aims to leverage this structure to improve the learning process.

The framework optimizes both the features of the training samples and the structured signals among the samples to enhance the neural network's performance. There are two types of input for the neural network: the features of a training sample (e.g., the pixels of an image) and the structure (e.g., the graph representing similarity). Both the features and the structure are fed into the neural network for training.

To utilize the structure in training, the framework augments each training sample with information from its neighbors in the given structure. This augmentation creates a new training batch that includes both the original samples and their neighbors. The neural network then processes both the training sample and its neighbors, generating embedding representations for each. The embedding representation captures the essence of the sample and its neighbors.

The difference between the embedding representation of a sample and its neighbors is calculated and added to the final loss as a regularization term. This regularization term encourages the neural network to preserve the similarity between a sample and its neighbors, maintaining the local structure. By leveraging these structured signals, neural networks can learn from unlabeled data and become more robust.

In addition to this overview, we provide hands-on tutorials that guide you step by step in using the neural structured learning framework. These tutorials cover various applications, including language understanding. In the next part of this material, we will apply the framework to a language understanding problem, specifically classifying the topic of a document. You can find the tutorial in the description below, along with more information on getting started with neural structured learning.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: NEURAL STRUCTURED LEARNING WITH TENSORFLOW
TOPIC: TRAINING WITH NATURAL GRAPHS

Neural Structured Learning is a new learning paradigm that enhances model accuracy and robustness. In this episode, we will explore how Neural Structured Learning can be used to train neural networks with natural graphs.

A natural graph is a set of data points that have an inherent relationship with each other. This relationship can vary based on the context. Examples of natural graphs include social networks, the World Wide Web, and data used for machine learning tasks. For instance, user behavior can be modeled as a co-occurrence graph, while articles or documents with references or citations can be modeled as a citation graph. In natural language applications, a text graph can represent entities and their relationships.

To train a neural network using natural graphs, consider the task of document classification. Often, there are many documents to classify, but only a few have labels. Neural Structured Learning utilizes citation information from the natural graph to leverage both labeled and unlabeled examples. If one paper cites another paper, it is likely that both papers share the same label. This relational information helps compensate for the lack of labels in the training data.

Building a Neural Structured Learning model for document classification involves several steps. First, the training data needs to be augmented to include graph neighbors. This is done by combining the input citation graph and document features to create an augmented training dataset. The pack neighbors API in Neural Structured Learning facilitates this process, allowing you to specify the number of neighbors for augmentation.

Next, a base module needs to be defined. This can be any type of Keras model, such as a sequential model, a functional API-based model, or a subclass model. Once the base model is defined, a graph regularization configuration object is created to specify hyperparameters. In this example, three neighbors are used for graph regularization. The base model is then wrapped with the graph regularization wrapper class, creating a new graph Keras model with a training loss that includes a graph regularization term. The graph Keras model can be compiled, trained, and evaluated like any other Keras model.

Creating a graph Keras model is straightforward and requires just a few extra lines of code. A Colab-based tutorial demonstrating document classification with Neural Structured Learning is available on the website.

Neural Structured Learning enables the use of natural graphs for document classification and other machine learning tasks. In the next material, we will explore how graph regularization can be applied when the input data does not form a natural graph.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: NEURAL STRUCTURED LEARNING WITH TENSORFLOW
TOPIC: TRAINING WITH SYNTHESIZED GRAPHS

Neural Structured Learning is a powerful technique that can be used in machine learning tasks where the input data does not form a natural graph. In such cases, a graph can be synthesized from the input data by defining a similarity metric and converting raw instances to their embeddings or dense representations. This can be done using pretrained embedding models like those on TensorFlow Hub. Once the embeddings are obtained, a similarity function such as cosine similarity can be used to compare pairs of embeddings, and if the similarity score is above a certain threshold, an edge is added to the resulting graph. This process is repeated for the entire dataset to build the graph.

Once the graph is built, applying neural structured learning is straightforward. For example, let's consider the task of sentiment classification using the IMDb dataset, which contains movie reviews. The code to build a neural structured learning model for this task involves several steps. First, the IMDb dataset is loaded. Then, the raw text in the movie reviews is converted to embeddings using a specific embedding model, such as swivel embeddings. The embeddings are then used to build the graph using the build graph API provided by neural structured learning. The graph is created with a similarity threshold, which determines the edges that are included in the graph.

Next, the features of interest for the model are defined, and the features are combined with the graph using the partNeighbours API in neural structured learning. This step augments the training data by incorporating information from the graph. Once the augmented training data is obtained, a graph regularized model is created. This involves defining a base model, which can be any type of Keras model, and a graph regularization configuration object that specifies hyperparameters such as the number of neighbors to consider for graph regularization. The base model is then wrapped with the graph regularization wrapper class, resulting in a new graph Keras model that includes a graph regularization term.

The final steps involve compiling, training, and evaluating the graph regularized model. It is important to note that neural structured learning also supports estimators, not just Keras models. The code example provided in the transcript is available as a colab-based tutorial on the website.

Neural structured learning allows us to handle machine learning tasks where the input data does not form a natural graph. By synthesizing a graph from the input data using similarity metrics and embeddings, we can apply neural structured learning techniques to improve model performance. This approach is applicable to various types of input data, including text, images, and videos.

In the next video, you will learn about another aspect of neural structured learning called adversarial learning, which can enhance a model's robustness to adversarial attacks. So stay tuned for more exciting content on neural structured learning!

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: NEURAL STRUCTURED LEARNING WITH TENSORFLOW
TOPIC: ADVERSARIAL LEARNING FOR IMAGE CLASSIFICATION

Welcome to the fourth episode of the Neural Structure Learning series. In this material, we will discuss learning with implicit structured signals constructed from adversarial learning. Neural structure learning is a framework that optimizes sample features and structured signals to improve neural networks. To illustrate this concept, let's consider an example of classifying an image as a cat or a dog. In reality, there are other similar images that form a structure representing the similarity among them. Neural structure learning optimizes both the sample features and the structured signals to enhance the neural network's performance.

But what if there is no explicit structure available to train the neural network? One approach is to dynamically construct the structure by generating adversarial neighbors. An adversarial neighbor is a modified version of the original sample designed to mislead the neural network into incorrect classification. To generate adversarial neighbors, we apply carefully designed perturbations, typically based on the reverse gradient direction, to the original sample. These perturbations are imperceptible to human eyes but confuse the neural network, resulting in incorrect classification.

To construct the structure, we connect the sample with its adversarial neighbor. This structure can then be utilized in the neural structure learning framework. The connection between the sample and its adversarial neighbor informs the neural network that they are similar despite the small perturbation. This helps the neural network maintain the similarity between the sample and its neighbor, improving its performance.

In TensorFlow, there are libraries and functions available to generate adversarial neighbors. Additionally, Keras APIs are provided to enable easy-to-use end-to-end training with adversarial learning. If you are interested in exploring the details of this library and APIs, please visit our website.

To demonstrate the application of adversarial learning in computer vision, let's consider a task of training a neural network to recognize handwritten digits. In the code example provided, we load the MNIST dataset containing images of handwritten digits and their corresponding labels. The features of each image are normalized to a range of 0 to 1. We then build a neural network using Keras APIs, which are supported by the neural structure learning framework. This framework enables adversarial learning by invoking the relevant APIs. Various hyperparameters can be configured, such as the multiplier applied to the adversarial regularization. Different values for these hyperparameters are provided based on empirical knowledge of their effectiveness. After configuring the neural network, we follow the standard Keras workflow of compiling, fitting, and evaluating the model.

With the APIs from the neural structure learning framework, we can enable adversarial learning with just a few lines of code. This allows us to improve the neural network's performance by incorporating adversarial neighbors into the learning process.

Adversarial learning is a powerful technique in neural structure learning that enhances the performance of neural networks by generating adversarial neighbors and incorporating them into the learning process. By connecting samples with their adversarial neighbors, the neural network learns to maintain similarity and improve classification accuracy.

In the presented material, two models were evaluated on their ability to classify images correctly. The first image was correctly recognized as a nine by both models. The second image, however, was an adversarial image designed to mislead the models. The baseline model incorrectly identified it as a five, while the model with adversarial learning successfully recognized it as a six. Similarly, the third image was also an adversarial image. The baseline model misclassified it as an eight, while the model with adversarial learning correctly classified it as a three.

This experiment demonstrates that adversarial learning can enhance the robustness of neural networks against small but misleading perturbations. The process of constructing the structure involves generating adversarial neighbors. In this video, we introduced the concept of adversarial learning and provided a code example using the API from the neural structure learning framework to enable adversarial learning.

For more detailed information and a step-by-step tutorial on the example discussed, please refer to the video description below. Additionally, a collab tutorial is available via the provided link. If you found this material informative, consider subscribing to this channel for more educational content.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: NATURAL LANGUAGE PROCESSING WITH TENSORFLOW
TOPIC: TOKENIZATION

In this didactic material, we will discuss the fundamental concept of tokenization in natural language processing using TensorFlow. Tokenization is the process of representing words in a way that a computer can process them, with the goal of training a neural network to understand their meaning.

To begin, let's consider the word "listen." This word is made up of a sequence of letters, which can be represented by numbers using an encoding scheme such as ASCII. However, the word "silent" has the same letters in a different order, making it difficult to understand the sentiment of a word based solely on its letters.

Instead of encoding letters, it may be easier to encode words themselves. For example, in the sentence "I love my dog," we can assign the word "I" the number 1, and the sentence as a whole would be represented as 1, 2, 3, 4. If we take another sentence, such as "I love my cat," we can encode it as 1, 2, 3, 5, where "I love my" has already been assigned numbers and we only need to encode the word "cat." By comparing the encoded sequences of the two sentences, we can observe a similarity between them, as they both express love for a pet.

Now, let's explore how we can implement tokenization using TensorFlow. There is an API available for tokenization, and we will demonstrate how to use it with Python. Here is an example of code that tokenizes sentences:

```
1. from tensorflow.keras.preprocessing.text import Tokenizer
2.
3. sentences = ["I love my dog", "I love my cat"]
4.
5. tokenizer = Tokenizer(num_words=100)
6. tokenizer.fit_on_texts(sentences)
7.
8. word_index = tokenizer.word_index
9. print(word_index)
```

In the code above, we import the necessary tokenizer API from TensorFlow Keras. We then create an instance of the tokenizer object, specifying the maximum number of words to keep (in this case, 100). The tokenizer is then fitted on the provided sentences, and we can access the word index property to obtain a dictionary where the keys are words and the values are their corresponding tokens.

The tokenizer is also intelligent enough to handle exceptions. For example, if we add a third sentence like "I love my dog!" where "dog" is followed by an exclamation mark, the tokenizer will recognize that it should not create a new token for "dog exclamation," but rather treat it as the existing token for "dog." Additionally, it will create a new token for the word "you" if it appears in the text.

If you would like to try out the code yourself, you can find it in the provided Colab notebook. By tokenizing words and sentences, you have taken an important step in preparing your data for processing by a neural network. In the next episode, we will explore the tools available in TensorFlow for managing the sequencing of numbers to represent sentences. Don't forget to subscribe for more educational material.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: NATURAL LANGUAGE PROCESSING WITH TENSORFLOW
TOPIC: SEQUENCING - TURNING SENTENCES INTO DATA

Welcome to this didactic material on turning sentences into data using TensorFlow for Natural Language Processing. In the previous material, you learned about tokenizing words using TensorFlow's tools. In this material, we will take it a step further by creating sequences of numbers from sentences and processing them to prepare for teaching neural networks.

To begin, let's recap the process of tokenizing words. We use a tokenizer to convert words into numeric tokens. This allows us to represent sentences as sequences of tokens. In this material, we will add a new sentence to our set of texts to demonstrate how to handle sequences of different lengths.

The tokenizer provides a method called "texts_to_sequences" which performs most of the work for us. It creates sequences of tokens representing each sentence. The resulting sequences can be seen in the output. For example, the first sequence is [4, 2, 1, 3], representing the tokens for "I", "love", "my", and "dog" in that order.

However, there is a challenge when dealing with unseen words in the text. If the neural network needs to classify texts that contain words not present in the word index, it can confuse the tokenizer. To handle this, we can use the "OOV" (Out Of Vocabulary) token property. By setting it as something unexpected, such as "<OOV>", the tokenizer will create a token for it and replace unrecognized words with the "OOV" token.

Using the "OOV" token helps maintain the sequence length, but it may still result in some loss of meaning. To address this, we can use padding. Padding ensures that all sequences have the same length by adding zeros at the beginning or end of the sequence. The "pad_sequences" function from the preprocessing module can be used for this purpose.

In the provided code, we import "pad_sequences" and pass our sequences to it. The function pads the sequences with zeros to match the length of the longest sequence. The output shows the word index, the initial sequences, and the padded sequences. The padded sequences have zeros added at the front to make them the same length as the longest sequence.

If you prefer the zeros to be added at the end of the sequence, you can set the "padding" parameter to "post". Additionally, if you don't want the padded sequences to have the same length as the longest sequence, you can specify the desired length.

This material has covered the process of turning sentences into data for Natural Language Processing using TensorFlow. We learned about tokenizing words, creating sequences of tokens, handling unseen words using the "OOV" token, and padding sequences to ensure uniform length.

When working with text data in Natural Language Processing (NLP), it is important to preprocess the text before feeding it into a neural network. One common preprocessing step is tokenization, which involves breaking down the text into individual words or tokens. In this video, we will learn how to tokenize text using TensorFlow.

To begin, we can use the TensorFlow Tokenizer API to tokenize our text. One important parameter to consider is `maxlen`, which determines the maximum length of each sequence. If a sentence is longer than the specified `maxlen`, we have two options for handling it. We can either truncate the sentence by chopping off words at the end (post truncation) or from the beginning (pre-truncation).

Post truncation involves removing words from the end of the sentence, while pre-truncation involves removing words from the beginning. By specifying the desired truncation method, we can ensure that our sequences have a consistent length.

In the next video, we will explore how to train a neural network using the tokenized text data. We will work with a dataset that contains sentences classified as sarcastic or not sarcastic. Our goal will be to determine if a given sentence contains sarcasm. Stay tuned for the next video to learn more!

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: NATURAL LANGUAGE PROCESSING WITH TENSORFLOW
TOPIC: TRAINING A MODEL TO RECOGNIZE SENTIMENT IN TEXT

Artificial Intelligence - TensorFlow Fundamentals - Natural Language Processing with TensorFlow - Training a model to recognize sentiment in text

In this educational material, we will explore how to train a model to recognize sentiment in text using TensorFlow. Sentiment analysis is the process of determining the sentiment or emotion expressed in a piece of text, such as whether it is positive, negative, or neutral. This can be a valuable tool in various applications, such as understanding customer feedback, analyzing social media sentiment, or automated content moderation.

To begin, we will use a dataset of headlines categorized as sarcastic or not. The dataset, provided by Rishabh Misra on Kaggle, consists of headlines and corresponding labels indicating whether they are sarcastic or not. We will focus on the headline text for our analysis.

The first step is to preprocess the data. We will tokenize the text, which involves converting each word into a numeric value. This allows us to represent the text in a format that can be understood by a machine learning model. We will use the TensorFlow tokenizer to accomplish this. By fitting the tokenizer on the headline text, we create tokens for each word in the corpus. These tokens are then used to convert the sentences into sequences of tokens.

Next, we need to ensure that all sequences have the same length. We achieve this by padding the sequences with zeros or truncating them if necessary. This step is important because machine learning models require input data of consistent shape and size. The padded sequences ensure that all input data has the same length, regardless of the original length of the headline.

Once we have preprocessed the data, we need to split it into training and testing sets. This allows us to evaluate the performance of our model on unseen data. We can easily accomplish this in Python by slicing the sequences and labels into separate training and testing sets. It is important to note that the tokenizer should only be fit on the training data to ensure that the model does not have access to the test data during training.

At this point, we have transformed the text into numerical sequences, but how do we extract meaning from these numbers? This is where word embeddings come into play. Word embeddings are dense vector representations of words that capture semantic relationships between words. By representing words as vectors in a high-dimensional space, we can measure the similarity or difference between words based on their positions in this space.

For sentiment analysis, we can plot sentiments on an x- and y-axis, with positive sentiments in one direction and negative sentiments in the opposite direction. Words with neutral sentiments would fall somewhere in between. By mapping words to their corresponding vectors, we can determine the sentiment of a piece of text based on the direction of the vector in the embedding space.

This is just a brief overview of the process of training a model to recognize sentiment in text using TensorFlow. The complete code and step-by-step instructions can be found in the material. By following these steps, you will be able to preprocess text data, tokenize it, pad the sequences, split the data into training and testing sets, and utilize word embeddings to extract sentiment information.

In natural language processing, one important task is sentiment analysis, which involves determining the sentiment or emotion expressed in a piece of text. In this context, we can use a technique called embedding to represent words as vectors in a multi-dimensional space. By training a neural network on labeled data, we can learn the directions in this space that correspond to different sentiments.

The process begins by plotting words labeled with sentiments, such as sarcastic and not sarcastic, in multiple dimensions. As we train the network, it learns what these directions should look like. Words that only appear in sarcastic sentences will have a strong component in the sarcastic direction, while others will have one in the not-sarcastic direction. As more sentences are loaded into the network for training, these directions can change.

Once the network is fully trained, we can input a set of words and have the network look up the vectors for these words, sum them up, and provide an idea of the sentiment. For example, if we input the phrase "not bad, a bit meh," the resulting vector would have coordinates of 0.7 on the y-axis and 0.1 on the x-axis, indicating a slightly positive sentiment.

To implement this concept, we can use a neural network with an embedding layer, where the direction of each word is learned over multiple epochs. After the embedding layer, we perform global average pooling, which involves adding up the vectors. The pooled vectors are then fed into a deep neural network. Training the model is as simple as using the model.fit function with the training data and labels, and specifying the validation data.

In an example implementation, the model achieved 99% accuracy on the training data and 81% to 82% accuracy on the test data, which consists of words the network has never seen before. This demonstrates the effectiveness of the sentiment analysis model.

To use the model for sentiment analysis on new sentences, we can tokenize the sentences using a tokenizer created earlier. This ensures that the words have the same tokens as the training set. The tokenized sequences are then padded to match the dimensions of the training set and use the same padding type. Finally, we can predict the sentiment on the padded set.

In the example provided, the first sentence had a predicted sentiment of 0.91, indicating a high probability of sarcasm. The second sentence had a predicted sentiment of 5 times 10 to the minus 6, indicating an extremely low chance of sarcasm.

All the code necessary to implement this sentiment analysis model is available in a runnable Colab notebook, which can be accessed through a provided URL. This allows users to try it out for themselves and build their own text classification models.

By using embedding and training a neural network, we can create a text classification model for sentiment analysis. This model is capable of accurately predicting the sentiment expressed in a piece of text, even on data it has never seen before.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: NATURAL LANGUAGE PROCESSING WITH TENSORFLOW
TOPIC: ML WITH RECURRENT NEURAL NETWORKS

In the previous material, you learned about tokenizing text and using sequences of tokens to train a neural network for sentiment classification. Now, let's explore the concept of generating text using neural networks.

To generate text, we need to consider the order of words in a sentence. This is where recurrent neural networks (RNNs) come into play. Unlike traditional neural networks, RNNs take the sequence of data into account when learning. In sentiment classification, the order of words doesn't matter because the sentiment is determined by the overall vector representation of the sentence. However, for text generation, the order of words is crucial.

To understand RNNs, let's look at the Fibonacci sequence as an example. The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones. We can represent the sequence using variables, such as n0, n1, and so on. The rule that defines the sequence is that any number in the sequence is the sum of the two numbers before it.

In a computation graph, we can visualize the Fibonacci sequence as a series of additions. Each number is contextualized into every other number in the sequence. This concept of recurrence, where a value can persist throughout the series, forms the basis of recurrent neural networks.

A recurrent neuron in an RNN takes an input value and produces an output value. Additionally, it generates a feed-forward value that gets passed to the next neuron in the sequence. By connecting multiple recurrent neurons together, we create a recurrent neural network. Each neuron takes the output and feed-forward value from the previous neuron as input and produces a new output.

The sequence of inputs and outputs in an RNN encodes the sequence information, similar to the Fibonacci sequence. However, it's important to note that the impact of an input weakens as the context spreads. For example, the first word in a sentence has little impact on the last word. This limitation can be useful for predicting text where the relevant context is close by, but it becomes challenging for longer sentences.

Recurrent neural networks (RNNs) are a type of neural network that takes the sequence of data into account when learning. They are particularly useful for tasks like text generation, where the order of words matters. RNNs encode sequence information by connecting recurrent neurons, allowing values to persist throughout the series.

In the study of Natural Language Processing (NLP), it is important to understand how words and phrases are predicted based on their context. In a recent discussion, the topic of language prediction was explored, specifically in the context of the word "Gaelic" being predicted from the word "Ireland."

While one might initially assume that the prediction would be "Irish," it is actually "Gaelic." The reason for this lies in the fact that the word "Ireland" is the key factor in determining the correct prediction. If we were to solely rely on words in close proximity to the desired word, we would miss this crucial information and end up with an inaccurate prediction.

To overcome this limitation, a recurrent neural network (RNN) with a longer short-term memory is employed. This type of network is known as a long short-term memory (LSTM) network. The LSTM network allows for the retention of information from further back in the sentence, enabling more accurate predictions.

In the next material, the concept of LSTM networks will be further explored. Be sure to stay tuned and subscribe for more insightful episodes on "Coding TensorFlow at Home."

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: NATURAL LANGUAGE PROCESSING WITH TENSORFLOW
TOPIC: LONG SHORT-TERM MEMORY FOR NLP

In this material, we will explore how to manage context in language across longer sentences using Long Short Term Memory (LSTM) in Natural Language Processing (NLP) with TensorFlow. Understanding the impact of words early in a sentence on the meaning and semantics of the end of the sentence is crucial. For example, if we want to predict the next word in a sentence like "today has a beautiful blue _____," we can easily predict that the next word is "sky" because we have a lot of context close to the word, especially the word "blue." However, predicting the missing word in a sentence like "I lived in Ireland, so I learned how to speak _____" is more challenging. The correct answer is "Gaelic," not "Irish," but the keyword that determines this answer is "Ireland," which is far back in the sentence.

Recurrent Neural Networks (RNNs) can struggle with capturing long-distance dependencies in language. The traditional RNN architecture can pass context to the next timestamp, but over a long distance, this context can become diluted, making it difficult to see how meanings in faraway words influence the overall meaning of a sentence. This is where the LSTM architecture comes in. LSTM introduces a cell state that can be maintained across many timestamps, allowing it to bring meaning from the beginning of the sentence to bear. It can learn that "Ireland" denotes "Gaelic" as the language. Moreover, LSTM can also be bi-directional, meaning that later words in the sentence can provide context to earlier ones, improving the accuracy of understanding the sentence's semantics.

To implement LSTM in TensorFlow, we can define an LSTM-style layer with a specific number of hidden nodes, which also determines the output space dimensionality. If we want the LSTM to be bi-directional, we can wrap the layer in a bi-directional wrapper. This allows the LSTM to analyze the sentence both forwards and backwards, learn the best parameters for each direction, and then merge them. However, it's important to note that bi-directional LSTMs may not always be the best choice for every scenario, so experimentation is recommended.

LSTMs can have a large number of parameters, as indicated by the model summary. For example, a bi-directional LSTM layer with 64 hidden nodes in each direction results in a total of 128 parameters. Additionally, we can stack multiple LSTM layers, similar to dense layers, where the outputs of one layer are fed into the next. In such cases, it is important to set the "return_sequences" parameter to true for all layers that are feeding another layer.

We have explored the concept of LSTM in NLP with TensorFlow for managing context in language across longer sentences. LSTM's ability to maintain a context (cell state) across many timestamps allows it to capture the meaning of words that are far apart in a sentence. It can also be bi-directional, enabling later words to provide context to earlier ones. Implementing LSTM in TensorFlow involves defining an LSTM-style layer, optionally wrapping it in a bi-directional wrapper, and stacking multiple LSTM layers if necessary.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: NATURAL LANGUAGE PROCESSING WITH TENSORFLOW
TOPIC: TRAINING AI TO CREATE POETRY

In this didactic material, we will explore the topic of Natural Language Processing (NLP) using TensorFlow. Specifically, we will focus on training an Artificial Intelligence (AI) model to create poetry using the lyrics of traditional Irish songs.

To begin, we have a corpus of text which includes the lyrics of various Irish songs. For simplicity, we will start with a single song called "Lanigan's Ball." The lyrics of this song are stored as a single string with newline characters indicating new lines.

The first step is to tokenize the lyrics and split them into sentences. This will form our corpus of text. We will use a tokenizer to convert each word in the lyrics into a numerical index. Additionally, we will add an out-of-vocabulary token to the word index to account for unseen words.

Next, we will convert the sentences into training data. For each line in the corpus, we will create a list of tokens using the text-to-sequence conversion. We will then generate n-grams from these tokens. An n-gram is a sequence of n words. By creating n-grams, we can train a model to predict the next word based on the previous n-1 words. This will allow us to generate new poetry.

To prepare the n-grams for training, we will pad them with zeros to ensure they have the same length. This will create a set of input sequences, where each sequence represents a line of the song. We can use everything but the last word in each sequence as our input (X), and the last word as our output (Y).

To train the model, we need to one-hot encode the output labels (Y). This means converting each label into a binary vector where only the index corresponding to the label is 1, and the rest are 0. This allows us to predict the most likely next word in the sequence given the current set of words.

Finally, we can use the Keras library to achieve this one-hot encoding. By training the model on the encoded data, we can generate new poetry based on the patterns learned from the lyrics of traditional Irish songs.

We have learned how to tokenize and sequence text, prepare it for training, and generate new poetry using TensorFlow and NLP techniques. By training an AI model on the lyrics of traditional Irish songs, we can create our own poetry.

In Natural Language Processing (NLP), one approach to training AI models involves using TensorFlow, a popular open-source machine learning framework. In this process, we can train AI to create poetry by feeding it sequences of words and predicting the next word in the sequence.

To begin, we need to prepare our data by assigning labels to our input sequences. Each input sequence is represented as a series of features, where each feature is a word in the sequence. The labels are the next word in each sequence. We can represent the features as a binary vector, where each word in the sequence is assigned a unique index. The index of the current word is set to 1, while all other indices are set to 0.

With our features and labels prepared, we can train a neural network using TensorFlow. The model architecture we will use is a simple one, but it can be optimized and improved upon. The model starts with a sequential layer, followed by an embedding layer. The embedding layer is used to represent the words in our corpus as dense vectors. The number of dimensions in the embedding layer can be adjusted based on the variation of words in the corpus. In this case, we set it to 240.

Next, we add a single LSTM (Long Short-Term Memory) layer, which is a type of recurrent neural network that can capture sequential information. We make the LSTM layer bi-directional to improve its ability to understand the context of the input sequence. Finally, we add a dense layer with the total number of words as the output. Since our labels are one-hot encoded, we want the output to be representative of this.

To train the model, we need to define a loss function and an optimizer. Since our problem is categorical with multiple classes, we use a categorical loss function such as categorical cross entropy. Once the loss function

and optimizer are defined, we can fit the features (X's) to the labels (Y's) and start training the model.

During training, the initial accuracy may be low, but it will improve over time as the model learns to match the input sequences to the corresponding labels. Once training is complete, we will have a model that can take a sequence of words as input and predict the next word in the sequence.

To generate poetry, we can seed the model with a sequence of words and use it to predict the next word. We can then add the predicted word to the sequence and repeat the process to generate more words. The accuracy of the model in predicting the next word will depend on the complexity of the input sequence. With the simple model architecture described above, we can achieve an accuracy of around 70 to 75%.

Using TensorFlow and NLP techniques, we can train AI models to generate poetry. By providing a sequence of words as input, the model can predict the next word in the sequence, allowing us to generate new poetry. Experimenting with different model architectures and training times can lead to further improvements in the accuracy and quality of the generated poetry.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: PROGRAMMING TENSORFLOW
TOPIC: INTRODUCTION TO TENSORFLOW CODING

Welcome to the world of coding with TensorFlow! In this educational material, we will explore the different aspects of TensorFlow from a coding perspective. Our aim is to equip you with the knowledge and skills to effectively utilize TensorFlow for machine learning and artificial intelligence applications. Throughout this material, we will provide concrete and hands-on examples to enhance your understanding.

One key component of TensorFlow that we will focus on is TensorFlow Lite. TensorFlow Lite is a lightweight solution specifically designed for mobile and embedded devices. It allows you to deploy TensorFlow models on resource-constrained platforms. This is particularly useful when you want to run machine learning and AI models on devices with limited computational power.

To illustrate the capabilities of TensorFlow Lite, let's consider an example application. Imagine an app that utilizes the camera feed of a mobile device. Using a pre-trained MobileNet model, this app can classify the dominant objects in the images captured by the camera. TensorFlow Lite enables the efficient execution of this model on the device, making real-time image classification possible.

We are committed to providing you with valuable content, and we would love to hear your suggestions and requests. Please feel free to contact us and let us know the topics and concepts you would like to explore further. Don't forget to subscribe to our material to stay updated and ensure you don't miss any of our future episodes.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: PROGRAMMING TENSORFLOW
TOPIC: INTRODUCING TENSORFLOW LITE

TensorFlow Lite is a lightweight solution provided by TensorFlow for running machine learning models on mobile and embedded devices. It allows you to execute machine learning tasks on mobile devices with low latency, eliminating the need for round trips to a server. TensorFlow Lite is currently supported on Android and iOS platforms.

Before using TensorFlow Lite, you need to have a trained model. This model is created by training a high-powered machine with a set of data. Once the training is complete, you will have a model file and a set of associated checkpoints. These checkpoints represent the state of the variables at different iterations of the learning process.

The model file can be in different formats, all based on the concept of protocol buffers (protobuf). Protocol buffers define data structures that can be used to load, save, and access data in a simple way. One of the formats is a graph definition file (graph def) with either a .pb or .pbtxt extension. The graph def file contains a description of the model's graph, including the operations performed by the model. Another format is the checkpoint file, which contains serialized variables from the TensorFlow graph. It provides information about the variable values at different points in the learning process. Additionally, there is a frozen graph file (frozen graph) that combines the variables from the latest checkpoint file with the graph and turns them into constants.

To use TensorFlow Lite, you need to convert the frozen graph into a TensorFlow Lite model. This conversion is done using the TensorFlow Optimizing Converter tool (TOCO). The resulting TensorFlow Lite model is optimized for mobile and embedded devices.

If you have access to the code, you can directly create a TensorFlow Lite model during the training process. This allows you to skip the step of converting a frozen graph into a TensorFlow Lite model. The code snippet provided demonstrates how to convert an image tensor from the session's graph def into a TensorFlow Lite object.

It is important to note that TensorFlow Lite is currently in Developer Preview, and not all operations supported by TensorFlow are yet handled by TensorFlow Lite. However, the TensorFlow team is continuously working on improving TensorFlow Lite to make it compatible with more operations.

In terms of compatibility, TensorFlow Lite supports popular public models that are commonly used for various scenarios. Two of the tested and compatible models are Inception v3 and MobileNets. Inception v3 is a model that has been validated with the popular ImageNet dataset and is commonly used as a benchmark for image classification tasks. MobileNets, on the other hand, are a set of models designed to be mobile-friendly with lower power requirements. While they may not be as accurate as Inception, they are suitable for mobile applications.

TensorFlow Lite is a lightweight solution for running machine learning models on mobile and embedded devices. It allows you to take advantage of machine learning capabilities on mobile devices without the need for server round trips. TensorFlow Lite supports various model formats and provides tools for converting models into TensorFlow Lite format. It is compatible with popular models like Inception v3 and MobileNets.

In this didactic material, we will explore the concept of image classification using TensorFlow and specifically focus on the use of MobileNet models. Image classification is a fundamental task in the field of artificial intelligence, and TensorFlow provides a powerful framework to accomplish this.

MobileNet is a pre-trained deep learning model that has been specifically designed for mobile and embedded devices. It is capable of accurately classifying images by identifying the dominant objects or subjects within them. By utilizing MobileNet, we can easily integrate image classification capabilities into our own applications.

To get started, you can download the code and MobileNet models from the TensorFlow for Poets Code Labs. These resources will serve as the foundation for our image classification project. The code labs are divided into two parts. In Part 1, you will learn about the basics of MobileNet image classification. In Part 2, you will discover how to deploy the model to a mobile device using TensorFlow Lite.

By following the code labs and understanding the underlying principles, you will gain a solid understanding of how to implement image classification using TensorFlow. Additionally, the material mentions that there are more videos available that delve into the details of building an application similar to the one demonstrated. These videos cover image classification on both Android and iOS platforms.

To stay updated with the latest content, it is recommended to subscribe to the channel providing these educational materials. Subscribing ensures that you receive notifications when new videos are released, allowing you to deepen your knowledge and expand your capabilities in TensorFlow.

If you have any questions or need further clarification, feel free to leave your queries in the comments section. The creator of the material will be able to provide additional guidance and support.

Now, let's dive into the exciting world of coding with TensorFlow and explore the possibilities of image classification!

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: PROGRAMMING TENSORFLOW
TOPIC: TENSORFLOW LITE FOR ANDROID

TensorFlow Lite is a lightweight solution developed by TensorFlow for running machine learning models on mobile and embedded devices. It allows you to run models on mobile devices with low latency, without the need for a round trip to a server. TensorFlow Lite is currently supported on Android and iOS devices. On Android devices, it can use the Android Neural Networks API for hardware acceleration if available, otherwise it will use the CPU.

To use TensorFlow Lite in your Android app, you need to include the TensorFlow Lite libraries in your project. This can be done by editing your build.gradle file. Once the libraries are included and synced, you can import a TensorFlow interpreter in your code. The interpreter loads the model and allows you to run it by providing a set of inputs. TensorFlow Lite will then execute the model and provide the outputs.

To load the model, you can store it in your app's assets folder and read it directly from there. After loading the model, you can instantiate an interpreter and load the model into it. The interpreter will then be ready to run the model.

In the provided example, the app uses a MobileNet model, which is a small, low latency, and low-power model designed for various use cases such as object detection, face attributes, fine-grained classification, and landmark recognition. Pre-trained MobileNet models are available, including one for image classification. The model is provided as a TensorFlow Lite file (.tflite) and a labels file describing the labels that the model is trained for.

In the app, frames from the camera are converted into images, which are then used as inputs to the model. The model outputs values that correspond to labels and their probabilities. The app selects the top three labels with the highest probabilities and displays them in the user interface.

To convert the camera frames into inputs that the model can read, the app converts the received bitmap image data into a byte buffer. This buffer is then passed to the TensorFlow Lite interpreter as input. The interpreter processes the input and provides the output, which includes the labels and their probabilities.

It is important to note that TensorFlow Lite is still in the Developer Preview stage and may not support all operations of TensorFlow. It is recommended to use pre-built models for now to avoid encountering issues with unsupported operations.

TensorFlow Lite is a lightweight solution for running machine learning models on mobile and embedded devices. It allows you to take advantage of machine learning capabilities without the need for a server round trip. By following the provided steps, you can include TensorFlow Lite in your Android app and run pre-trained models for various use cases.

TensorFlow Lite for Android is an exciting technology that allows you to load your models onto an Android device and execute them using onboard hardware. This opens up a world of possibilities beyond just image recognition in a video stream. While TensorFlow Lite is currently in Developer Preview, it is constantly being updated to support more operations.

If you want to learn more about TensorFlow Lite and how to retrain the models for specific scenarios, you can check out the TensorFlow for Poet code labs on the Google developer site. These code labs will guide you through the process of tailoring the models to your needs.

TensorFlow Lite for Android is a powerful tool that enables you to bring the power of TensorFlow to mobile devices. With its ability to leverage onboard hardware, you can create innovative applications that take advantage of the capabilities of Android devices. So, start exploring TensorFlow Lite and unleash your creativity!

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: PROGRAMMING TENSORFLOW
TOPIC: TENSORFLOW LITE FOR IOS

TensorFlow Lite is a lightweight solution for running machine learning models on mobile and embedded devices. It allows you to run models with low latency on devices like smartphones and tablets, without the need for a server connection. TensorFlow Lite is currently supported on both Android and iOS platforms.

In this tutorial, we will focus on using TensorFlow Lite with iOS. We will explore an iOS app that utilizes TensorFlow Lite to classify objects in real-time. The app uses a pre-trained model called MobileNet, which is a small, low-latency, and low-power model designed for various use cases such as object detection, face attributes, fine-grain classification, and landmark recognition.

To get started, you will need to download the MobileNet model file and its corresponding labels file. These files are available for download and can be found in the TensorFlow Lite documentation. Once downloaded, you will unzip the files to obtain a .tflite file describing the model and a labels file that provides the names of the objects the model has been trained to recognize.

Before we proceed with coding the app, you will need to install Xcode, which is the integrated development environment (IDE) for iOS app development. Xcode can be downloaded from the App Store. Additionally, you will need to install Homebrew, a package manager for macOS, which will be used to install necessary dependencies.

Once you have Xcode and Homebrew installed, you can proceed to build the TensorFlow Lite library for iOS. The library can be built from the open-source code available. There is a script provided that automates the building process, but it may take some time to complete. Once the building process is finished, you can find the source code for the sample app in the examples/iOS/camera directory.

After obtaining the source code, navigate to the project directory and run "pod install" to install the required dependencies. Once the dependencies are installed, you can open the project in Xcode. In the project navigator, you will find a data folder that contains the .tflite file for the model and a .hex file for the labels. You can replace these files with the ones you downloaded earlier or keep them as they are.

To load the model and labels in the app, open the ViewController.m file. At the top of the file, you will find constants specifying the names of the model and labels files. Make sure these constants match the names of the files you have in the data folder.

In the ViewController.m file, you will also find code that initializes an interpreter object from the TensorFlow Lite API. The interpreter takes the captured frames from the camera as input and outputs a set of probabilities for each possible label. These probabilities indicate the likelihood of the model recognizing an object as a particular label.

Once you have made the necessary changes to the code, you can run the app on an iOS device. Please note that the app does not work in the simulator; you will need a physical iOS device to test it.

By following these steps, you can utilize TensorFlow Lite to run machine learning models on iOS devices. This allows you to perform tasks such as object classification and detection directly on your mobile device, without the need for a server connection.

TensorFlow Lite for iOS is an exciting technology that allows you to load your models onto a device and execute them, taking advantage of the device's onboard hardware. One example of using TensorFlow Lite is image recognition in a video stream. By using the TensorFlow Lite API, you can analyze the video stream and identify objects in real-time.

In the provided example, the speaker demonstrates how to use TensorFlow Lite for iOS to perform image recognition on a coffee mug. The speaker uses a helper function called "get top n" to retrieve the top labels and their associated probabilities from the model's output. By pointing the camera at the coffee mug, the interpreter returns a label with a 74 percent probability that it is a coffee mug.

It's important to note that the labels are indexed, and the speaker explains that the list of labels starts at one when printed by Xcode, but in memory, it starts at zero. Therefore, the fifth label in the list corresponds to the coffee mug. This example showcases the functionality of TensorFlow Lite for iOS in recognizing objects in real-time.

However, it's worth mentioning that TensorFlow Lite is currently in Developer Preview, which means there may be some restrictions and unsupported TensorFlow APIs. The TensorFlow team is continuously updating and improving the APIs to provide a better experience for developers.

If you are interested in learning more about TensorFlow Lite and how to retrain the model used in the example for specific scenarios, the speaker recommends checking out the TensorFlow for Poets code labs on the Google Developer site. These code labs provide detailed instructions on how to customize and tailor models to suit your needs.

TensorFlow Lite for iOS is a powerful tool that enables developers to deploy and execute models on mobile devices. By leveraging onboard hardware, developers can perform tasks such as image recognition in real-time. Although TensorFlow Lite is in Developer Preview, it offers exciting possibilities for mobile application development.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW.JS
TOPIC: TENSORFLOW.JS IN YOUR BROWSER

Artificial Intelligence - TensorFlow.js in your browser

Welcome to this educational material on using JavaScript for machine learning in the browser with TensorFlow.js. TensorFlow.js is a JavaScript library that allows you to train and deploy machine learning models directly in the browser and on Node.js. In this material, we will explore how to build and train a simple model that runs entirely in the browser.

To get started, let's create a basic web page with a single empty div element. Next, we need to add the TensorFlow.js libraries to our page. This can be done by inserting a script tag with the latest version of TensorFlow.js. Make sure to include the script loader in the head tag of your HTML document.

Now that we have set up our page for TensorFlow.js, let's dive into a simple but powerful example of how it works. The goal of machine learning is to train a model using input data, which can then be used to predict output data for future inputs. In our example, we have a set of data points that exhibit a linear relationship.

To train a model on this data, we need to define the model and compile it. In TensorFlow.js, we can create a model using the `tf.sequential` function, which allows us to stack layers sequentially. In our case, we only have one layer and one node, but this is the simplest way to define a neural network. We then add a dense layer, where all nodes in each layer are connected to each other.

After defining the model, we compile it by specifying parameters such as the loss function and optimizer. The loss function measures the error between predicted and actual values, and the optimizer determines how the model adjusts its parameters during training. In our example, we use mean squared error as the loss function and stochastic gradient descent as the optimizer.

Next, we define the input (x) and output (y) values for the linear relationship. These values are represented as tensors, which are multi-dimensional arrays in TensorFlow.js. We create tensors for both x and y values using the `tf.tensor2d` function.

Finally, we train the model by calling the `fit` method for a fixed number of iterations, known as epochs. In our example, we train the model for 250 iterations. Once the model is trained, we can use it to make predictions. For example, if we want to predict the output (y) for a given input (x=5), we use the `predict` method of the model.

TensorFlow.js allows us to build and train machine learning models directly in the browser. We can define models, compile them with specific parameters, and train them using input data. Once trained, these models can be used to make predictions for new input values.

In this didactic material, we will explore the concept of using TensorFlow.js in the browser to create and train a neural network for predicting a linear relationship. TensorFlow.js is a JavaScript library that allows us to build and train machine learning models directly in the browser.

To begin, we pass an input tensor, which is a single value in a one by one array, to the neural network. TensorFlow then processes this input and provides us with a predicted value. For example, when we input the value 10, the neural network predicts a value close to 19. However, the initial prediction may not be accurate, as it depends on the training of the network.

To improve the accuracy of the neural network, we can train it for more epochs. An epoch refers to one complete pass through the training data. By training for more epochs, we give the network more time to correct its errors and improve its predictions. For instance, training the network for 500 epochs resulted in a more accurate prediction of 38.9 when the input value was 39.

It is important to note that refreshing the page may cause the predicted value to change slightly due to retraining the neural network. However, after retraining, the predicted value remains consistent. This

demonstrates the ability to train and retrain the network to achieve more accurate predictions.

By utilizing TensorFlow.js in the browser, we can create and train neural networks for various tasks, such as predicting linear relationships. The process involves passing input tensors to the network and receiving predicted values in return. By training the network for more epochs, we can improve the accuracy of the predictions. This opens up possibilities for implementing machine learning models directly in web applications.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW.JS
TOPIC: PREPARING DATASET FOR MACHINE LEARNING

In this material, we will explore the process of preparing a dataset for machine learning using TensorFlow.js. Specifically, we will focus on a classification problem using the well-known iris dataset. By the end of this material, you will understand how to shape your data and get it ready for training a machine learning model.

To begin, let's briefly recap the importance of shaping data in TensorFlow. Shaping data is a crucial step in the data science process, as it allows us to structure our data in a way that can be effectively used for training a machine learning model.

Previously, in a basic machine learning scenario, we dealt with a linear relationship between data points. However, in more complex scenarios, such as classification problems, we need to consider multiple items of data and their relationships to determine the classification of a given entity.

For example, consider an email classification problem, where we want to determine if an email is spam or not based on various features like the sender, keywords, and pictures. Traditional if-then type code is not suitable for such scenarios, which is where machine learning comes into play.

In our case, we will use the iris dataset, which consists of measurements from 150 different flower samples. These measurements include petal length, petal width, sepal length, and sepal width. Each measurement is associated with one of three types of iris flowers.

To train a neural network with this dataset, we need to shape the data appropriately. Typically, the data is provided in the form of comma-separated values. Each entry in the dataset contains four measurements and a value (0, 1, or 2) representing the category of the flower.

To prepare the data for training, we split it into arrays based on the flower classes. By iterating through the dataset, we create separate arrays for each class, containing the measurements and corresponding values.

Once we have these arrays, we can convert them into tensors. Tensors are the fundamental data structure used in TensorFlow.js. In our case, we create four sets of tensors: X for training, X for testing, Y for training, and Y for testing.

To determine the split between training and testing data, we use a parameter called "test split." In this example, we set it to 0.2, meaning 80% of the data will be used for training, and 20% will be used for testing.

The key function in this process is "convert to tensors." This function takes the data, targets, and split as inputs and converts them into tensors, splitting them into training and test sets accordingly.

By following this process, we can effectively shape our dataset and get it ready for training a machine learning model. Additionally, we can evaluate the model's performance by comparing the predicted values with the actual values from the test set.

Shaping the dataset is a crucial step in preparing it for machine learning. By understanding how to structure the data and use tensors effectively, we can train and evaluate machine learning models accurately.

When working with machine learning models, it is essential to prepare the dataset properly to ensure efficient training. In this didactic material, we will explore the steps involved in preparing a dataset for machine learning using TensorFlow.js.

The first step in dataset preparation is encoding categorical data. In this example, we have three categories of flowers labeled as 0, 1, and 2. To encode this categorical data, we convert each category into an array representation. For example, the flower labeled as 0 will be encoded as [1, 0, 0], the flower labeled as 1 will be encoded as [0, 1, 0], and so on. This encoding allows the data to map directly to the output neurons of the model.

Next, we need to slice the data into four arrays based on a specified test split size. This step helps in dividing the dataset into training and testing subsets. The test split determines the proportion of data that will be used for testing the model's performance.

Finally, we want to convert the 2D arrays into a clean and linear set of tensors that can be fed into the training process. This is achieved using the TensorFlow `concat` function along axis 0. By concatenating the arrays along the specified axis, we obtain a one-dimensional tensor that is suitable for training. This step reduces the complexity of the data and improves training speed and accuracy.

Here is an example code snippet that demonstrates the dataset preparation process:

```
function prepareDataset(flowers, labels, testSplit) {
  // Perform one-hot encoding of the labels
  const encodedLabels = labels.map(label => {
    const encoded = [0, 0, 0];
    encoded[label] = 1;
    return encoded;
  });

  // Slice the data into training and testing arrays
  const testSize = Math.floor(flowers.length * testSplit);
  const trainSize = flowers.length - testSize;

  const xTrain = flowers.slice(0, trainSize);
  const yTrain = encodedLabels.slice(0, trainSize);
  const xTest = flowers.slice(trainSize);
  const yTest = encodedLabels.slice(trainSize);

  // Concatenate the arrays into tensors
  const xTrainConcat = tf.concat(xTrain, 0);
  const yTrainConcat = tf.concat(yTrain, 0);
  const xTestConcat = tf.concat(xTest, 0);
  const yTestConcat = tf.concat(yTest, 0);

  return [xTrainConcat, yTrainConcat, xTestConcat, yTestConcat];
}

// Example usage
const [xTrain, yTrain, xTest, yTest] = prepareDataset(flowers, labels, 0.2);
console.log('xTrain:', xTrain);
console.log('xTest:', xTest);
```

By following these steps, you have successfully pre-processed the raw data into tensors that are suitable for efficient training. Preparing the dataset correctly is a crucial aspect of designing any machine learning system.

In the next material, we will explore how to train a neural network using this pre-processed data and discuss the design considerations for the network.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW.JS
TOPIC: BUILDING A NEURAL NETWORK TO PERFORM CLASSIFICATION

In this didactic material, we will explore the process of building a neural network using TensorFlow.js to perform classification tasks. TensorFlow.js is a JavaScript library that allows us to train and deploy machine learning models directly in the browser and on Node.js.

In the previous episodes, we covered the basics of getting started with TensorFlow.js in the browser. We learned how to build a simple model that fits values to a line using a small training set. We also discussed the importance of data preparation for training, where we converted a raw CSV dataset into tensors for both the feature and label data.

Now that our data is ready, let's dive into building a neural network that can classify future data. The goal is to create a model that can identify the type of iris flower based on unknown measurements. This scenario represents the fundamental concept of machine learning, where we learn to infer desired results from existing data without explicitly programming rules.

To start, we will define an asynchronous function called "do iris" that will be called at the end of our JavaScript code block. In the previous episode, we created the "iris.js" file, which contained the data and preprocessing code. We used a function called "get iris data" to split the data into training and test sets. By setting the split parameter to 0.8, we allocated 80% of the data for training and 20% for testing.

Next, we will create a model by calling a function called "train model" and passing the training and test data as parameters. Before we proceed, let's make the function asynchronous so that we can await its return. We will create a sequential neural network, similar to what we did in the previous episode.

Now, let's set up some values for the learning rate and the number of epochs (iterations) for our machine learning process. These values will be constants that we can tweak later. The learning rate is used to define the optimizer, and in this case, we will use the Adam optimizer. Adam is an optimization algorithm introduced in 2015 as an improvement over stochastic gradient descent.

With our model set up, we can now define its architecture. In this example, we will use a neural network with two layers. The first layer will have ten neurons and will be activated by a sigmoid function. The sigmoid function provides an output between 0 and 1, which is ideal for classification tasks. The second output layer will have three units, representing the three types of iris flowers. Its activation function will be softmax, which normalizes the input values to ensure they add up to 1. This allows us to interpret the outputs as likelihoods for each flower type.

Once our model architecture is defined, we can compile it with the Adam optimizer, a categorical cross-entropy loss function, and the desired metric. The categorical cross-entropy loss function is suitable for classification tasks like ours. It calculates the loss between the predicted and actual labels.

Now that our model is ready, it's time to train it. We will use the "model.fit" method and pass it our training and validation data. We will also specify the number of epochs we want to train for. During training, we can track the progress by using the "on epoch end" callback, which allows us to print the current loss value after each epoch.

By running the training process, we will observe the loss value diminishing epoch by epoch, indicating that our model is learning and improving its performance.

We have covered the process of building a neural network using TensorFlow.js to perform classification tasks. We learned how to prepare data, define the model architecture, compile the model, and train it using the fit method. This knowledge forms the foundation for more advanced machine learning applications.

In this material, we will discuss how to build a neural network using TensorFlow.js to perform classification. We will start by training the model and then move on to using it for predictions.

To train the model, we need to create a tensor with input values that match those of the real data. After training

the model, we will have a model that can classify the input data. However, since we have only trained the model for a short period of time, there may be some errors. We will learn how to fix these errors later.

Now, let's focus on using the trained model for predictions. We can pass the tensor to the model and get a prediction back. The prediction will consist of three values, which determine the likelihood of each flower matching the tensor. In this case, it seems that number two is the closest match to being the winner.

To make the prediction even clearer, we can use the Arg max function to polarize the values. This effectively sets the likelihood for flower zero and one to zero, and flower two to one. Think of this as similar to writing if-then statements to compare values and find the biggest one. This approach is much easier than dealing with a large number of values and writing a lot of code.

If you want to test your model against a test set to see how many predictions it gets right, you can use the provided code. For each input in the test set, get the prediction from the model and compare it against the real value. If they are the same, the prediction is correct. If they are different, the prediction is incorrect. If you get a high error rate, you can adjust the number of epochs and the learning rate and try again.

This concludes our discussion on building a neural network to perform classification using TensorFlow.js. If you have followed along with the previous episodes, you have taken your first steps into machine learning in the browser with JavaScript. If you prefer to use JavaScript with Node.js instead, you can learn more about that and everything JavaScript-related on the official TensorFlow website.

If you have any questions, please leave them in the comments below. Don't forget to subscribe for more great TensorFlow content.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW.JS
TOPIC: USING TENSORFLOW TO CLASSIFY CLOTHING IMAGES

In this didactic material, we will explore the fundamentals of using TensorFlow.js to classify clothing images. TensorFlow.js is a powerful library that allows us to build and train machine learning models directly in the browser. Specifically, we will be using TensorFlow.js to create a deep neural network model that can classify images of different types of clothing.

To get started, we need to locate the code that we will be using. You can find the code by following the instructions provided in the description of this material. Once you have found the code, we can proceed with the rest of the steps.

The dataset we will be using is called Fashion MNIST. It contains 28 by 28 pixel images of various types of clothing, such as t-shirts, tops, sandals, and ankle boots. This dataset is widely used in the machine learning community for classification tasks.

Our neural network model will take the 28 by 28 pixel images as input. If we flatten the image, we will have an array of 784 input values. The first hidden layer of our model will consist of 128 neurons, where each neuron will receive input from all the pixels in the image. Finally, we have the output layer, which will give us ten values representing the probability that the image belongs to a specific class. These output values will be probabilities, meaning that if we sum all of them, the result will be one.

Now, let's move on to the code. The code can be executed directly from the browser using Collab, a virtual execution environment running in Google Cloud. Make sure you are logged in with your Google account before proceeding.

The first step in the code is to import the necessary libraries. Then, we load the Fashion MNIST dataset using a convenience function in Keras. This function will give us two lists: one for training the model and another for testing its accuracy. The dataset has ten classes, each represented by a number. We create a list mapping these numbers to textual descriptions of the classes.

Next, we explore the dataset by examining its shape. The training dataset contains 60,000 images, each of size 28 by 28 pixels. Similarly, the test dataset contains 10,000 images. We can also plot and visualize some of the images in the dataset.

Before training the model, we need to normalize the pixel values. Instead of having integer values between 0 and 255, we convert them to float values between 0 and 1. This normalization step is important for improving the performance of the model.

Finally, we define the neural network model using the Sequential API. The layers are processed in the order they are declared. In our case, we have a flattened input layer, followed by two dense layers. The first dense layer receives input from all the pixels in the image. We apply a nonlinear activation function (ReLU) to the results. The final output layer consists of ten classes, and we use the softmax activation function to create a probability distribution that sums to one.

To train the model, we provide the training images and labels, and specify the number of epochs. One epoch represents a full iteration over the entire dataset. By training the model, we aim to teach it to accurately classify the different types of clothing.

This didactic material has provided an overview of using TensorFlow.js to classify clothing images. We have learned about the Fashion MNIST dataset, the structure of the neural network model, and the steps involved in training the model. By following the provided code, you can start building your own image classification models using TensorFlow.js.

The training data set consists of 60,000 examples, totaling 300,000 images used to train the model. After training, the model's accuracy on the test data set is evaluated. The model performs well, considering its simplicity. It can now be used for predictions. The prediction for the first image is a probability distribution

indicating that it belongs to class number 9, which represents an ankle boot. The correct label for the first image confirms that the model made the correct prediction. Further predictions are made, displaying both the predicted value and the correct labels alongside the images. The model continues to perform well.

To demonstrate the prediction process, the first image from the test data set is selected. It has a resolution of 28 by 28 pixels. An initial dimension is added to the image because the predict call requires a list of images. The predict call is made, and the model predicts that the image belongs to the class representing an ankle boot. Finally, the highest index from the probability distribution list, index number 9, is selected.

This concludes the demonstration of using TensorFlow to classify clothing images. I hope you found this material informative. For more content on machine learning and TensorFlow, remember to subscribe to the TensorFlow channel. Now it's your turn to go out there and create some great models. Don't forget to share your experiences with us!

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TEXT CLASSIFICATION WITH TENSORFLOW
TOPIC: PREPARING DATA FOR MACHINE LEARNING

Text Classification with TensorFlow: Preparing Data for Machine Learning

In this didactic material, we will discuss text classification, which is the process of training a neural network to predict or classify text data. Before we dive into coding, let's understand some unique challenges and the steps involved in preparing the data for machine learning.

Neural networks typically work with numbers, not text. Therefore, the first step in text classification is to convert words into numerical representations. In this case, we will be learning from movie reviews to determine if they are positive or negative. To achieve this, we need to convert the words into numbers that represent them.

To begin, we will check the licenses and import the necessary libraries. We will be using TensorFlow, NumPy, and Keras. After importing the libraries, we will download the IMDB dataset, which is included with Keras. This dataset has already converted the words into integers and sorted them into a dictionary. The top 10,000 words used across all the reviews are included in this dataset.

Once we have loaded the dataset, we will have our training data and labels, as well as our test data and labels. The labels are simple, with zero representing a negative review and one representing a positive review. The training data consists of a set of numbers that are indexes into the array of words. Each review starts with a 1, indicating the start of the review. The subsequent numbers represent the words in the review.

To decode the review and understand the words, we can use a handy-dandy decoding function. The values 0 to 3 are reserved, with 1 representing the start of the review and 0 used for padding. Padding is important because it ensures that all the training data is of the same length, making it easier to train a neural network. If a review is longer than the specified length, it will be trimmed, and if it is shorter, it will be padded with zeros.

To accomplish this, we can use the preprocessing APIs provided by Keras. By setting the desired length to 256 words, we can ensure that all reviews are of the same length. If padding is required, it will be done using the pad character, which is represented by 0. After preprocessing, all the reviews will be 256 words long.

Once the data is prepared, we are ready to move on to the next step, which is designing a neural network to accept this data and train a model to determine the sentiment of movie reviews. This will be covered in the next episode.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TEXT CLASSIFICATION WITH TENSORFLOW
TOPIC: DESIGNING A NEURAL NETWORK

In this didactic material, we will explore the concept of designing a neural network for text classification using TensorFlow. We will focus on the use of embeddings to convert words into vectors in a multi-dimensional space, allowing us to derive sentiment from text.

To begin, we have already pre-processed the data, converting it into a numeric format suitable for training a neural network. However, in order to design a neural network that can learn from this data, we need to use embeddings. An embedding is a representation of a word as a vector in a multi-dimensional space. The idea is that words with similar sentiment will have similar directions in this space.

To better understand embeddings, let's consider a simplified example. Imagine we are fans of Regency era romances, like those written by Jane Austen. We can take characters from her novel "Pride and Prejudice" and plot them on a 2D chart. One axis represents gender, derived from their title, and the other axis represents their position in society, estimated based on their title. For example, Mr. Collins, a male character, would be plotted in blue. Mr. Darcy, another male character, would be plotted in red. By adding Lady Catherine de Bourg, a female character, we can see that she is plotted in orange, indicating her gender and nobility. From these vectors, we can gain some understanding of these characters and their relationships.

This process of converting words into vectors is called embedding, and there are various algorithms available in TensorFlow and Keras to handle this. Using an embedding layer, we can automatically determine the appropriate axes for a plot like this and sort words into vectors based on their sentiment.

Now, let's see how to implement this in TensorFlow. In the provided material, you can observe the construction of the model using Keras. The first layer is an embedding layer that takes the 10,000 words in our dataset and converts them into 16-dimensional vectors. These vectors are then flattened into a 1-dimensional vector and passed into a dense layer with 16 nodes. The output of this layer is then fed into a final dense layer with 1 node, which uses a sigmoid activation function to produce a value between 0 and 1. In our case, a value of 1 indicates a positive review, while a value of 0 indicates a negative review.

After constructing the model, we compile it by specifying an optimizer and a loss function. In this example, we use the Adam optimizer and the binary cross-entropy loss function, as we are dealing with binary classification. It is common practice to hold off on testing against the test data until we have a finished training model. Therefore, a portion of the training data, around 10,000 records, is set aside for testing and validation.

Once the model is trained and the loss values are satisfactory, we can evaluate the model against the test set to measure its accuracy. In the provided material, you can see that the model achieves approximately 87.5% accuracy on the test set.

The rest of the notebook focuses on plotting the loss function to check for overfitting, but we will not delve into that in this didactic material.

To demonstrate the model's performance with a new review, two additional reviews are created. The first review consists of random words, while the second review consists solely of the word "brilliant." These reviews are then evaluated using the model's predict function. The results show that the random review scores 0.34, as expected, while the biased review consisting of the word "brilliant" scores significantly higher.

This didactic material has provided an overview of designing a neural network for text classification using TensorFlow. We have explored the concept of embeddings and how they can be used to convert words into vectors, enabling sentiment analysis. The material also demonstrates the construction and evaluation of a neural network model for text classification.

In the previous material, we learned about building a text sentiment classification using TensorFlow. We saw the steps involved in this process and how to implement them. To further explore this topic, please refer to the workbook provided in the description below, where you can practice these steps on your own.

In the next material of this series, we will shift our focus to regression using TensorFlow. Regression is a technique used to predict continuous numerical values based on input data. We will learn how to implement regression models using TensorFlow and understand the concepts behind it.

Make sure to subscribe to our channel for more educational materials on TensorFlow and artificial intelligence. Stay tuned for the upcoming material on regression with TensorFlow.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: OVERFITTING AND UNDERFITTING PROBLEMS
TOPIC: SOLVING MODEL'S OVERFITTING AND UNDERFITTING PROBLEMS - PART 1

In this didactic material, we will discuss the concept of overfitting and underfitting in machine learning models. Overfitting and underfitting are common problems encountered when training models. In this first part, we will focus on understanding these problems and how to solve them.

Overfitting occurs when a model becomes too specialized in solving the training data and performs poorly when tested on new data. It can be visualized as a situation where the training loss continues to decrease, but the validation loss starts to increase. This happens because the model memorizes the answers in the training data and fails to generalize to new data.

On the other hand, underfitting occurs when a model is too simple and does not have enough variables to solve the training data. In this case, a more advanced model with more parameters and variables would perform better.

To illustrate these concepts, we will use the IMDB movie reviews dataset. In a previous episode, we classified these reviews using text classification techniques. In that episode, we observed that the loss initially decreased for both the training and validation datasets. However, after a certain point, the training loss continued to decrease while the validation loss started to increase. This is a clear sign of overfitting.

To prepare the input data for our model, we need to transform the movie reviews into a multi-hot encoded array. In this encoding, each word index present in the review is assigned a value of one, while all other indexes are set to zero. This encoding helps the model understand which words are present in a review.

In the code provided below, we import the necessary libraries, check the TensorFlow version, and load the IMDB dataset. The dataset consists of reviews represented as sets of numbers, where each number corresponds to the ID of a word. The dataset is sorted, meaning that the most common word is assigned ID number one, the second most common word is assigned ID number two, and so on. By loading 10,000 words, we load the 10,000 most common words across all the reviews.

To create the multi-hot encoded array, we use the code snippet provided. This array represents the presence or absence of words in each review. The x-axis represents the word ID, and the y-axis represents the hot encoding. As shown in the example, most words have low IDs since they are the most commonly used words in the reviews.

Before concluding this episode, we have a little homework for you. In the code, you will find a function that maps numbers to words. Your task is to figure out how these numbers correspond to specific words. You can find hints in the function and the code from the text classification episode.

That's it for this episode. In the next part, we will explore more techniques to solve the overfitting and underfitting problems. Now it's your turn to apply what you've learned and create some great models. See you in the next episode!

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: OVERFITTING AND UNDERFITTING PROBLEMS
TOPIC: SOLVING MODEL'S OVERFITTING AND UNDERFITTING PROBLEMS - PART 2

Overfitting and underfitting are common problems in machine learning models, including those built using TensorFlow. In this didactic material, we will explore these problems and discuss how to solve them.

To start, let's briefly review the concepts of overfitting and underfitting. Overfitting occurs when a model performs well on the training data but fails to generalize to new, unseen data. This happens when the model learns to fit the noise or random fluctuations in the training data instead of the underlying patterns. On the other hand, underfitting occurs when a model is too simple and fails to capture the complexity of the data, resulting in poor performance on both the training and test data.

In the previous episode, we looked at the multi-hot encoding of an input string using the sentence "the small cat." We then introduced three different models to demonstrate overfitting. The first model, called the baseline model, consisted of three dense layers with 16 neurons each and a classification layer using sigmoid. The second model, the small model, was a simplified version of the baseline model with only four neurons. Finally, the third model, the bigger model, had 512 neurons for the first two layers, similar to the baseline model.

We trained and tested these models using TensorFlow. When comparing the training loss, we observed that the baseline and bigger models quickly decreased their loss, while the small model took longer to converge. However, the interesting part was evaluating the models on the test dataset. Here, we noticed that as the models had more features, their loss increased. This is a clear example of overfitting, where the models memorize the training data but fail to generalize to new data.

To address overfitting, we can use regularization and dropout techniques. Regularization involves forcing the weights of the model to be as small as possible, preventing it from learning specialized things about the training data. This can be achieved by using the kernel regularizer parameter when defining the model. By applying regularization, we observed that the model performed better on the test dataset compared to the previous baseline model.

Another technique to combat overfitting is dropout. Dropout randomly sets a fraction of the layer's features to zero during training. This helps to prevent the model from relying too heavily on any single feature, reducing overfitting. By adding a dropout probability of 50% to our layers during training, we saw an improvement in the overfitting problem.

If you want to delve deeper into overfitting and underfitting, we recommend watching the generalization video from the machine learning crash course, which provides more information on these topics.

Overfitting and underfitting are common challenges in machine learning models. By applying techniques like regularization and dropout, we can mitigate these problems and improve the model's performance on unseen data. Remember to experiment with different approaches and find the best solution for your specific problem.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: ADVANCING IN TENSORFLOW
TOPIC: SAVING AND LOADING MODELS

Saving and loading models is an important aspect of working with TensorFlow. When training models of significant complexity, the training process can take a long time, sometimes even days or weeks. If the training process is interrupted, all the model weights and values will be lost, and the training will have to start from the beginning. To avoid this, it is crucial to save the model at regular intervals.

Saving the model allows you to resume training from the point where it was saved. This is particularly useful when training models that require a significant amount of time to converge. Additionally, saving the model enables you to transfer it to another computer and continue training there.

To save a model in TensorFlow, you can use the ModelCheckpoint callback. This callback allows you to specify the path where the model should be saved, as well as other options such as saving only the weights and enabling debug output during the saving process. By calling the fit method and providing the ModelCheckpoint callback, the model will be saved once every epoch.

When the model is saved, three files are created. The cp.ckpt.data file contains all the weight values, the cp.ckpt.index file specifies which partition file contains which weights, and the checkpoint file is a text file that points to the latest model.

To load a saved model, you can use the load_weights method and provide the checkpoint path. This will initialize the model with the previous training state, allowing you to continue training from where it was saved.

There are additional options for saving and loading models in TensorFlow. For example, you can specify the period parameter when creating the ModelCheckpoint object to save a new model every certain number of epochs. You can also use the tf.train.latest_checkpoint function to automatically find the latest saved model.

Another way to save models is by calling the save method on the model itself. This will create an HDF5-formatted file that includes not only the weights but also the model's configuration and the state of the optimizer. To load a model saved using this method, you can use the keras.models.load_model function.

Saving and loading models in TensorFlow is essential for resuming training from a specific point and transferring models to other computers. By using the ModelCheckpoint callback or the save method, you can easily save and load models, ensuring that your training progress is not lost.

TensorFlow, a popular open-source machine learning framework, includes a crucial file format known as SavedModel. This format enables the seamless exchange of models across various components of TensorFlow, such as TensorFlow Python, TensorFlow.js, and TensorFlow Lite. The versatility of SavedModel facilitates collaboration and interoperability within the TensorFlow ecosystem.

SavedModel is particularly significant because it allows users to save and load models, making it easier to reuse and share them across different projects and platforms. By utilizing this file format, developers can seamlessly transfer models between TensorFlow versions and even across different programming languages.

One of the notable features of SavedModel is its compatibility with Keras, a high-level neural networks API in TensorFlow. The TensorFlow team is actively working on integrating SavedModel support into Keras, ensuring a streamlined experience for users. This integration will enhance the convenience and accessibility of SavedModel, enabling Keras users to leverage its capabilities effortlessly.

To learn more about SavedModel and its integration with Keras, you can refer to the provided links. These resources offer in-depth information and insights into the usage and benefits of SavedModel.

SavedModel is a vital file format in TensorFlow that facilitates the exchange of models between various TensorFlow components. Its compatibility with Keras enhances its usability, enabling users to save, load, and share models seamlessly. By leveraging SavedModel, developers can efficiently collaborate, reuse, and deploy machine learning models across different platforms and programming languages.

Now it's your turn to explore the possibilities of SavedModel and create remarkable models. Don't forget to share your experiences and discoveries with the TensorFlow community.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: ADVANCING IN TENSORFLOW
TOPIC: TENSORFLOW LITE, EXPERIMENTAL GPU DELEGATE

Running inference on machine learning models on mobile devices can be demanding due to limited processing power and considerations like battery life. To address this, TensorFlow Lite has introduced a developer preview of a GPU back end. This back end utilizes OpenGL ES 3.1 compute shaders on Android and Metal compute shaders on iOS. While a full open-source release is planned for later in 2019, developers can currently try out the pre-compiled binary preview.

Performance tests have shown that the GPU back end can be two to seven times faster than a floating-point CPU implementation. The inference time is represented in orange for GPU and gray for CPU. The speed-up is most significant for complex models that are better suited for GPU utilization. However, smaller and simpler models may not see as much benefit due to the time cost of transferring data into GPU memory.

To get started with the GPU delegate, developers can try the demo apps provided by TensorFlow Lite. There are tutorial links and screen casts available for Android and iOS. For Android, an Android archive is provided that includes TensorFlow Lite with the GPU back end. On iOS and C++, developers can use modify graph with delegate after creating their model. All necessary code is included in the sample app.

It is important to note that not all operations are currently supported by the GPU back end. Models that use only the supported ops will run fastest, while others will automatically fall back to the CPU. The TensorFlow.org documentation provides more information on GPU, including a deep dive into how it works, optimizations, performance best practices, and more.

TensorFlow is continuously working on adding more optimizations, performance improvements, and API updates. Developers are encouraged to provide feedback on the GitHub page and YouTube channel. Any questions or comments can be left in the comments section.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW IN GOOGLE COLABORATORY
TOPIC: GETTING STARTED WITH GOOGLE COLABORATORY

Google Colab is an executable document that allows you to write, run, and share code within Google Drive. It is similar to the popular Jupyter project, where Colab can be thought of as a Jupyter notebook stored in Google Drive. A notebook document consists of cells, which can contain code, text, images, and more. The notebook is connected to a cloud-based runtime, which means that Python code can be executed without any setup required on your own machine.

The code cells in Colab are executed using the cloud-based runtime, providing a rich and interactive coding experience. You can use any of the functionality that Python offers. For example, you can define variables, loop through ranges of numbers, and print the square of each number.

Executing cells in Colab is convenient, as you can use the Shift-Enter shortcut instead of the Play button. The outputs of the cells are not limited to simple text. They can contain dynamic and rich outputs. For instance, you can search Colab's built-in library of code snippets and insert code to create interactive data visualizations. Colab supports several third-party visualization libraries, such as Altair.

Colab notebooks can be shared like Google Docs. To provide a narrative around the code you've executed, you can use text cells formatted using Markdown. Markdown is a plain text document format that allows you to add headings, paragraphs, lists, and even mathematical formulas.

Sharing your notebooks with others can be done through Google Drive sharing or by exporting the notebook to GitHub. The notebooks are stored in the standard Jupyter Notebook format, allowing them to be viewed and executed in Jupyter Notebook, JupyterLab, and other compatible frameworks.

The convenience of sharing notebooks means that you can find and explore many interesting notebooks around the web. One useful collection is the Seedbank project at research.google.com/seedbank. It provides various notebooks, including the Neural Style Transfer seed, which demonstrates how to use deep learning to transfer styles between images.

To learn more about Colab, you can visit colab.research.google.com and find the Welcome notebook. This notebook contains links to tutorials and other information about Jupyter and Colab notebooks. Additionally, there are more videos in this series that explore Colab in more depth.

In the next video, Lawrence will explore how to install TensorFlow using Colab and how to use different runtimes to access resources like the GPU.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW IN GOOGLE COLABORATORY
TOPIC: GETTING STARTED WITH TENSORFLOW IN GOOGLE COLABORATORY

In this didactic material, we will explore how to get started with TensorFlow in Google Colaboratory. TensorFlow is a popular open-source machine learning framework that allows you to build and train neural networks. Google Colaboratory, or Colab for short, is a free cloud-based platform that provides a Python development environment with GPU support, making it ideal for running TensorFlow code without the need for any local installation.

To begin, you can create a new Colab notebook directly from Google Drive. Simply click on "New" and select "Colaboratory" from the More menu. Once the Colab is created, you will have a code cell ready for you to start coding. You can type Python code directly into the cell.

To install TensorFlow in Colab, you can use the PIP command. However, since PIP is a command line tool and not a code command, you need to prefix it with an exclamation mark (!) for Colab to understand it. In a new code cell, you can use the command "!pip install tensorflow" to install TensorFlow. Colab will then download and install the TensorFlow library if it is not already installed. You can check if the installation was successful by printing out the installed version using the code "import tensorflow as tf; print(tf.__version__)". This will display the version of TensorFlow that is installed. In the provided example, version 1.12 is installed.

Additionally, TensorFlow comes in different versions, including a GPU version that allows you to leverage the power of a GPU for faster training. To install the GPU flavor of TensorFlow in a fresh notebook, you need to ensure that your runtime has a GPU. You can do this by resetting all runtimes and then setting the runtime type to be GPU-based. Once the GPU-based runtime is allocated and launched, you can install the GPU version of TensorFlow using the command "!pip install tensorflow-GPU". Again, you can verify the installed version of TensorFlow using the same code as before.

In the next video of this series, you will learn how to train a neural network using Keras, a high-level API built on top of TensorFlow, to perform classification of breast cancer data. This will allow you to apply the concepts and techniques learned in TensorFlow to real-world problems.

By following these steps, you can easily get started with TensorFlow in Google Colaboratory and begin your journey into the world of artificial intelligence and machine learning.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW IN GOOGLE COLABORATORY
TOPIC: BUILDING A DEEP NEURAL NETWORK WITH TENSORFLOW IN COLAB

Welcome to part three of this series on using Google Colab to build and train neural networks. In the previous material, we learned how to install TensorFlow with Colab. In this material, we will focus on using TensorFlow to build a neural network for breast cancer classification, which can be done entirely in the browser using Colab.

The data used for training this neural network comes from the Diagnostic Wisconsin Breast Cancer Database, which contains nearly 600 samples of data. Each sample represents a cell biopsy, with 30 features extracted per cell. For the purpose of this exercise, the data has been pre-processed into several CSV files, allowing us to focus solely on the neural network itself.

To begin, we need to upload the CSV files. Colab offers a convenient way to load external data. We can use the following code to load the CSVs into panda dataframes:

```
1. # Code for uploading CSV files
```

Next, we will use Keras in the sequential API to create the neural network. Since each cell has 30 features, our input dimension will be 30. The network will consist of layers with sizes 16, 8, 6, and 1, respectively. The final layer will be activated by a Sigmoid function, which will push the output towards either 1 or 0, as we are classifying two features.

The network requires a loss function and an optimizer to be defined. The loss function measures how well the network performs during training, while the optimizer tries to improve on the network's performance. The training process consists of 100 steps, with each step iterating over the data. The training itself takes place in the Fit function, where the network makes a guess at the relationship between the input and the output, measures its performance using the loss function, and updates its guess using the optimizer. You can easily modify the number of training iterations to explore different results.

After training, the network achieves a loss of 0.0595, indicating an accuracy of approximately 94%. We can now test the network with data that it hasn't seen before, specifically the x-test data. This will give us a set of predicted y values, which represent probabilities. To obtain binary predictions (0 or 1), we can use the following code:

```
1. # Code for obtaining binary predictions
```

Finally, we can compare the predicted values for the test set against the actual known values using the following code:

```
1. # Code for comparing predicted values with actual values
```

In this case, the test set contains 114 values, and the network achieves 100% accuracy.

Now, a question for you: why do you think the network achieves 100% accuracy on the test set, even though its overall accuracy during training was approximately 94%? Post your answers in the comments below.

That concludes this material. In the next material of this series, we will learn about using different runtimes and processors, as well as how to utilize GPUs and TPUs directly in your browser. Don't forget to subscribe to stay updated. Thank you.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW IN GOOGLE COLABORATORY
TOPIC: HOW TO TAKE ADVANTAGE OF GPUS AND TPUS FOR YOUR ML PROJECT

Machine learning techniques such as Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs) have shown great promise in various applications, including image classification, scene reconstruction, and speech recognition. To efficiently train these models on large datasets, machine learning engineers often rely on specialized hardware like Graphics Processing Units (GPUs) or Tensor Processing Units (TPUs). GPUs and TPUs serve as accelerators for parallelizable operations within the models, enabling faster and more efficient training.

Google Colab provides a platform for developing deep learning models using GPUs and TPUs at no cost. To utilize these resources, simply change the runtime type in Google Colab by selecting "Runtime," then "Change Runtime Type," and choosing either GPU or TPU. For this example, we will focus on GPU.

To confirm that TensorFlow can access the GPU, run the command "device_name=tf.test.gpu_device_name()" in a Colab notebook. The output will display the device's location, typically at slot 0. This verification ensures that TensorFlow can leverage the GPU for accelerated computations.

To observe the speed-up provided by GPUs compared to CPUs, we can use a basic Keras model. By executing the model, we find that it completes training in approximately 43 seconds, whereas the same task would take around 69 seconds on a CPU. This demonstrates a significant boost in speed, approximately a third of the original time.

For further information about the hardware in use, you can execute two commands in any Colab notebook. These commands provide details about the CPU, RAM, and GPU configurations, allowing you to understand the resources available for your machine learning project.

Moving on to TPUs, we can explore a more interesting example. By changing the runtime type to TPU and executing the necessary steps, we can predict Shakespearean text using TPUs and Keras. In the Colab notebook, a two-layer forward LSTM model is built, and the Keras model is converted to its TPU equivalent using the standard methods of Fit, Predict, and Evaluate. The notebook includes steps for downloading data, building a data generator with TF logging, checking the size of the input array from Project Gutenberg, and constructing the model.

Training this model will take some time due to the extensive nature of Shakespeare's corpus. After cycling through ten epochs and achieving satisfactory accuracy, predictions can be made using the trained model. While the generated text may not be perfect, it exhibits characteristics of a traditional script. By adding more layers, nodes, and clusters of TPUs, you can further improve accuracy and generate more Shakespeare-like plays.

This material highlights how to accelerate machine learning projects using GPUs and TPUs in Google Colab. The next video will guide you through upgrading existing TensorFlow code to TensorFlow 2.0. Stay tuned by subscribing to the TensorFlow YouTube channel. Meanwhile, continue building exciting projects with TensorFlow and share them on Twitter using the hashtag #PoweredbyTF. We look forward to seeing your creations.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW IN GOOGLE COLABORATORY
TOPIC: UPGRADE YOUR EXISTING CODE FOR TENSORFLOW 2.0

TensorFlow 2.0 is a new version of TensorFlow that focuses on usability, developer productivity, and simple, intuitive APIs. It combines the best features of Keras and Eager Execution, while still providing the low-level control that users expect from the original TensorFlow. With TensorFlow 2.0, all the components have been packaged together into a comprehensive platform that supports machine learning workflows through training and deployment.

One of the key changes in TensorFlow 2.0 is the introduction of new APIs and the modification of existing ones. To help with the transition, TensorFlow provides the TF upgrade V2 tool, which converts existing TensorFlow 1.12 Python scripts to TensorFlow 2.0 preview scripts.

To use the TF upgrade V2 script, you need to install the TF nightly preview using the PIP command. Once installed, you can use the script by prefacing your command with an exclamation point. For example, you can specify an input file and an output file, and then run the script. The output code will show all the conversions that have taken place due to the upgrade script. You can also check the report file to ensure that the original script has been modified correctly.

It is important to note that you should not manually update parts of your code before running the script. Functions that have had reordered arguments, such as TF.argmax or TF.batch_to_space, can cause the script to add keyword arguments incorrectly. Instead of reordering arguments, the script adds keyword arguments to functions that have had their arguments reordered.

However, it is worth mentioning that the conversion process may not be able to upgrade all functions. One example is the TF.nn.conv2d function, which no longer takes the "use_cudnn_on_gpu" argument. If the script detects this, it will report it in the standard output and in the report file. In such cases, you will need to fix it manually by updating the code.

By following these steps, you can successfully upgrade your legacy TensorFlow code to TensorFlow 2.0 using the TF 2.0 upgrade script. If you encounter any issues during the conversion process, you can file an issue on GitHub for assistance. Additionally, if you have any feedback or suggestions for TensorFlow 2.0, you can send an email to testing@TensorFlow.org.

We hope you find this upgrade process helpful and encourage you to explore the new features and improvements in TensorFlow 2.0. Happy engineering!

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW IN GOOGLE COLABORATORY
TOPIC: USING TENSORFLOW TO SOLVE REGRESSION PROBLEMS

Today, we will be learning how to use TensorFlow to solve regression problems. In a regression problem, we aim to predict a single numerical value, such as a price or a probability. This is different from a classification problem, where we try to determine the class or group an example belongs to.

Before we dive into the details, let's briefly discuss the difference between regression and classification. Regression and classification are two of the most common problems solved with machine learning. In regression, our model predicts a numerical value, while in classification, our model assigns examples to different classes or groups.

In this tutorial, we will focus on training a model to predict the miles per gallon of cars from the 1970s. We will use data such as weight, number of cylinders, and horsepower to make these predictions. Since miles per gallon is a single number, regression is the appropriate approach for this problem.

To get started, we will be using Keras, a high-level API for deep learning that is user-friendly and powerful. We will also need to install seaborn, a data visualization library, as it is not included by default in Google Colaboratory. Once installed, we will import pandas, TensorFlow, and Keras.

Next, we will retrieve a dataset from the University of California at Irvine, which provides a repository of public domain datasets. Keras makes it easy for us to download and access this dataset. We will then assign names to the columns for better readability.

Data cleaning is an important step in any machine learning project. We will check for unknown values in our dataset and drop any rows that contain them. Additionally, we will handle the Origin column, which is categorical, by converting it into one-hot columns. This allows us to represent each category as a separate binary column.

To evaluate the performance of our model, we need to split our data into training and test sets. We will keep 80% of the data for training and reserve the remaining 20% for testing. This ensures that our model can generalize well to unseen data.

Before we proceed, let's take a closer look at the data. We will use seaborn's pair plot utility, which provides a visual representation of the joint distributions of our features. This plot allows us to observe any relationships between the features.

Furthermore, we will examine summary statistics of our features, including the mean, standard deviation, minimum, quartiles, and maximum values. It is important to note that the ranges of these values can vary significantly, which can impact the performance of our model.

We have learned how to use TensorFlow to solve regression problems. We discussed the difference between regression and classification and applied regression techniques to predict the miles per gallon of cars from the 1970s. By using Keras and data visualization libraries like seaborn, we were able to preprocess the data, split it into training and test sets, and analyze the relationships between the features.

When training a machine learning model, it is important to ensure that the model does not receive the correct answers as labels in the training or test data. To achieve this, we need to split off the labels from our data sets.

Another important step is to normalize the features of our data. This involves scaling the values so that they fall between 0 and 1. One way to achieve this is by using a z-score, which involves subtracting the mean and dividing by the standard deviation.

To build our model, we can use Keras sequential, which provides a fully connected model. In this case, we will have three dense layers. The first two layers will have a relu activation function, while the last layer will have a linear activation function, which is suitable for regression models.

We also need to specify an optimizer, which in this case will be RMSprop, and a loss function, which will be mean squared error. Additionally, we can define metrics to evaluate the performance of our model, such as mean squared error and mean absolute error.

After creating our model, we can examine its summary using the summary() function in Keras. This will provide information about the layers and the number of trainable parameters.

Before training the model, it is a good practice to test it to ensure it produces results without errors. Although the results will be meaningless at this stage, it helps to verify that the model is functioning correctly.

To train the model, we can specify the number of epochs, which represents the number of passes through the training data. During training, we can print a dot for each epoch to keep track of the progress. Additionally, we can split off a portion of the data as a validation set to evaluate the model's performance during training.

Once training is complete, we can examine the training results, including the loss, mean absolute error, and mean squared error. However, it is important to note that if the loss and validation loss are increasing, it indicates overfitting, which means the model is not generalizing well.

To address overfitting, we can use a technique called early stopping. This involves stopping the training process when the model stops improving. We specify a metric to monitor, such as validation loss, and a patience parameter to determine how long to wait before stopping.

After implementing early stopping, we can train the model again and observe the learning curves. Ideally, both the training and validation loss should decrease together, indicating that the model is not overfitting.

Finally, we can evaluate the overall performance of our model by looking at metrics such as mean absolute error. In this case, the model has an error of 1.8 miles per gallon, which may be considered good depending on the context.

With our trained model, we can now make predictions and plot them to visualize their accuracy. While the predictions may not be perfect, they are generally close to the expected values.

TensorFlow provides powerful tools for solving regression problems. By following the steps outlined above, we can train a model, address overfitting using early stopping, and make accurate predictions.

In this material, we will summarize the key concepts covered in the previous material. We will discuss error analysis, mean squared error loss, metrics for evaluating model performance, data normalization, and early stopping.

To begin, let's focus on error analysis. In machine learning, it is crucial to understand the errors made by our models. One way to visualize and analyze errors is by using a histogram. By plotting the errors, we can gain insights into the distribution and identify any patterns or deviations from the expected Gaussian distribution.

Next, we will delve into mean squared error loss. This is a commonly used loss function in regression problems. The mean squared error measures the average squared difference between the predicted and actual values. By minimizing this loss, we aim to improve the accuracy of our model.

To evaluate the performance of our model, we rely on metrics. These metrics provide us with quantitative measures of how well our model is training. By analyzing metrics such as accuracy, precision, recall, and F1 score, we can assess the effectiveness of our model and make informed decisions for further improvement.

Data normalization is another important step in preparing our data. Normalization ensures that all features have a similar scale, preventing certain features from dominating the learning process. By scaling our data, we can enhance the performance and convergence of our models.

Lastly, we will discuss early stopping. Overfitting is a common challenge in machine learning, where the model performs well on the training data but fails to generalize to new, unseen data. Early stopping is a technique that helps us address overfitting by monitoring the model's performance on a validation set. When the model's performance starts to deteriorate, we stop the training process to prevent overfitting.

To summarize, in this material, we have covered error analysis using histograms, mean squared error loss, metrics for evaluating model performance, data normalization, and early stopping as a solution to overfitting. These concepts are fundamental in using TensorFlow to solve regression problems.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW 2.0
TOPIC: INTRODUCTION TO TENSORFLOW 2.0

At Google, we have released TensorFlow 2.0, an easy-to-use and powerful framework for training, deploying, managing, and scaling machine-learned models. This release has been driven by feedback from developers, enterprises, and researchers who have expressed the need for a flexible and efficient framework that supports deployment to any platform.

TensorFlow 2.0 provides a comprehensive ecosystem of tools for developers, enterprises, and researchers to push the boundaries of machine learning and build scalable ML-powered applications. With the integration of Keras into TensorFlow, eager execution by default, and an emphasis on Pythonic function execution, TensorFlow 2.0 aims to provide a familiar experience for Python developers.

Despite the focus on simpler APIs, TensorFlow 2.0 does not compromise on flexibility. The framework now offers a more complete low-level API, allowing users to export all internal operations and provide inheritable interfaces for crucial concepts such as variables and checkpoints. This enables advanced customizations without the need to rebuild TensorFlow.

TensorFlow 2.0 adopts the Saved model file format, which allows models to be run on various runtimes, including the Cloud, Web, Browser, Node.js, Mobile, and embedded systems. This flexibility enables deployment to different platforms using TensorFlow Extended, TensorFlow Lite for mobile and embedded systems, and TensorFlow.js for running models in the browser and on Node.js.

For distributed training, TensorFlow 2.0 introduces the distribution strategy API, which minimizes code changes and delivers excellent out-of-the-box performance. It supports distributed training with Keras's model.fit as well as with custom training loops. Multi-GPU support is available, and TensorRT can be used for fast inference on GPUs.

To simplify data access, TensorFlow 2.0 expands TensorFlow datasets, providing a standard interface for various types of datasets, including images, text, and video. While the traditional session-based programming model is still supported, we recommend using regular Python with eager execution. The tf.function decorator can convert Python code into graphs, which can then be executed remotely, serialized, and optimized for performance. Autograph, built into tf.function, allows for the direct conversion of regular Python control flow into TensorFlow control flow.

For users familiar with TensorFlow 1.x, we have published a migration guide to help transition to TensorFlow 2.0. The guide includes an automatic conversion script to facilitate the process.

To learn how to build applications using TensorFlow 2.0, we recommend referring to the effective 2.0 guide and trying out our online courses developed in collaboration with deeplearning.ai and Udacity.

For more information about TensorFlow 2.0, including how to download and get started with coding machine-learned applications, please visit the official TensorFlow site at tensorflow.org.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW HIGH-LEVEL APIS
TOPIC: LOADING DATA

TensorFlow high-level APIs provide a convenient and efficient way to develop machine learning models. In this series, we will explore the key steps involved in developing a machine learning model using TensorFlow and its high-level APIs. In this particular video, we will focus on loading and preparing data for machine learning.

To begin, we will use the covertype dataset from the US Forestry Service and Colorado State University. This dataset contains approximately 500,000 rows of geophysical data collected from specific regions in National Forest areas. Our objective is to predict the soil type based on the features present in the dataset.

The dataset consists of various types of features, including real values such as elevation, slope, and aspect, as well as categorical values that assign integers to soil types and wilderness area names. Some of the features have been binned into an 8-bit scale.

Before we proceed with data processing, it is recommended to enable eager execution when prototyping a new model in TensorFlow. Enabling eager execution allows immediate execution of TensorFlow operations, providing flexibility for experimentation and iteration. To enable eager execution, simply add a single line of code after importing TensorFlow.

Next, we will load the dataset from a CSV file using TensorFlow's CSV dataset. The dataset contains 55 columns of integers, which will be processed and consumed for training. TensorFlow's dataset is similar to NumPy arrays or Pandas DataFrames, but it is specifically designed for processing and consuming data for training machine learning models.

Once the data is loaded, we can inspect the structure of the dataset. In its current form, each row is represented as a tuple of 55 integer tensors. However, we want the data to reflect the known structure of the dataset, with features and labels properly separated and grouped.

To achieve this, we can define a function that will be applied to each row of the dataset. This function will parse the row and return the desired set of features and a class label. Additionally, this function can be used to perform special transformations, such as image processing or adding random noise, efficiently using TensorFlow's dataset capabilities.

In our case, we will primarily handle transformations using feature columns, which will be explained further in the series. The parsing function's main objective is to correctly separate and group the columns of features.

By following these steps, we can effectively load and prepare data for machine learning using TensorFlow's high-level APIs. This sets the foundation for further stages in the machine learning model development process, such as model architecture prototyping, training, evaluation, and deployment.

In TensorFlow, when working with data sets, it is important to preprocess the data to make it suitable for machine learning models. One common preprocessing step is loading and transforming the data into a format that can be easily fed into the model. In this transcript, we will discuss how to load and process data using TensorFlow high-level APIs.

The first step in loading data is to understand the structure of the data set. In this example, the data set contains both categorical and numerical features. The soil type, for instance, is a categorical feature that is one-hot encoded, meaning it is represented as a binary vector. To simplify the representation, the soil-type tensor is combined into a single length-40 tensor. This allows us to treat soil type as a single feature rather than 40 independent features.

Next, the soil-type tensor is combined with other features, which are spread out over 55 columns in the original data set. To ensure that all the necessary values are included, the tuple of incoming values is spliced. The resulting values are then zipped up with human-readable column names to create a dictionary of features. This dictionary can be further processed later.

Additionally, the one-hot-encoded wilderness area class is converted into a class label that ranges from 0 to 3. While leaving it as one-hot encoded is possible, converting it into a class label is more suitable for certain model architectures or loss calculations.

Once the features and labels are obtained for each row, they are mapped using a function and then batched into sets of 64 examples. TensorFlow data sets provide built-in performance optimizations for mapping and batching, which helps remove I/O bottlenecks.

After the data is processed, it is important to check its structure. In this case, the parsed dictionaries of integers with human-readable names are batched. For example, a feature that is a single number becomes a length-64 tensor, while the conversion of soil type results in a tensor with a shape of 64 by 40. The class labels are also represented as a single tensor with category indices.

To summarize, in this transcript, we learned how to load and preprocess data using TensorFlow high-level APIs. We combined categorical features, such as soil type, into a single tensor and zipped up the features with human-readable column names to create a dictionary. We also converted the one-hot-encoded wilderness area class into a class label. Finally, we mapped the data row-wise and batched it into sets of 64 examples, taking advantage of TensorFlow's built-in performance optimizations.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW HIGH-LEVEL APIS
TOPIC: GOING DEEP ON DATA AND FEATURES

In this didactic material, we will explore the topic of TensorFlow high-level APIs, specifically focusing on going deep on data and features. We will learn how to prepare data for machine learning using TensorFlow's high-level APIs, including the use of feature columns and handling categorical data.

To begin, let's discuss the importance of preparing data for machine learning. In many real-world applications, data is structured and represents vocabularies or human concepts that need to be transformed before they can be used in machine learning models. TensorFlow provides feature columns as a powerful tool for this purpose.

A feature column in TensorFlow is a configuration class that tells our model how to transform raw data so that it matches the expectations of machine learning models. Feature columns are particularly useful when working with structured data that is not already numeric. They allow us to convert categorical or non-numeric data into a format that can be used by machine learning models.

For example, let's consider the Covertype dataset, which contains geophysical data collected from different regions in National Forest areas. One of the features in this dataset is the type of tree in each region, represented by an integer between 1 and 7. To transform this categorical data into a format suitable for machine learning, we can define a feature column that represents this category. This feature column will instruct the model to expect categorical IDs less than 8.

In addition to defining feature columns, we also need to configure how we want to transform our categorical data for use in a model that expects continuous data. Using feature columns, we can easily build a set of instructions that convert the categories into an embedding column. This conversion can be done manually, but using feature columns has the advantage of making the transformations part of the model's graph, allowing them to be exported with the saved model.

In TensorFlow, we can define feature columns for each of our features. Numeric data can be represented using a simple numeric column. For data that is spread out over a vector, such as soil type data, we can use numeric feature columns with a shape argument to capture the relationship between the data points.

Once we have defined all our feature columns, they become the first layer of our model using a feature layer. This layer acts like any other Keras layer and takes in the raw data, including categorical indices, and transforms it into the appropriate representations that our neural network expects. The feature layer also handles the creation and training of embedding columns for categorical data.

By using feature columns, we can handle data transformations batch by batch in TensorFlow, eliminating the need for a separate pipeline to do feature transformations in memory. TensorFlow provides a wide range of feature columns and even allows us to combine individual columns into more complex representations of the data that our model can learn.

To summarize, in this didactic material, we have learned about the importance of preparing data for machine learning and how feature columns in TensorFlow's high-level APIs can help us with this task. We have seen how feature columns can be used to transform categorical data into numeric representations and how they become the first layer of our model. By using feature columns, we can handle data transformations efficiently and effectively.

In the previous material, we learned about the data transformation process in the context of TensorFlow high-level APIs. Now, we will explore how to feed the transformed data into the model and train it. This is part three of our series on TensorFlow fundamentals.

To begin, we need to understand the importance of data in training a model. Data is the foundation upon which machine learning models are built. It contains the information that the model uses to make predictions or classifications. However, raw data is often not suitable for direct consumption by the model. It requires preprocessing, cleaning, and transformation to be in a format that the model can understand.

In part two of this series, we discussed various techniques for data transformation, such as normalization, one-hot encoding, and feature scaling. These techniques help to preprocess the data and make it compatible with the model's requirements. Once the data is transformed, we can proceed to feed it into the model.

Feeding the transformed data into the model involves integrating it into the model's architecture. This step is crucial as it allows the model to learn from the data and adjust its internal parameters accordingly. The transformed data serves as the input to the model, and the model processes this input to make predictions or classifications.

In part three of this series, we will delve deeper into the process of feeding data into the model. We will explore how to choose appropriate loss functions and optimizers. Loss functions measure the discrepancy between the model's predictions and the actual values, providing feedback on how well the model is performing. Optimizers, on the other hand, adjust the model's internal parameters based on the feedback from the loss function, aiming to minimize the loss and improve the model's accuracy.

By understanding how to feed data into the model and optimize its performance, we can enhance the effectiveness of our machine learning models. This knowledge is crucial for building robust and accurate AI systems.

To learn more about adding data and training models using TensorFlow high-level APIs, stay tuned for the next part of this series. Remember to subscribe to the TensorFlow YouTube channel for updates on future materials.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW HIGH-LEVEL APIS
TOPIC: BUILDING AND REFINING YOUR MODELS

TensorFlow Fundamentals - TensorFlow high-level APIs - Building and refining your models

In this didactic material, we will explore the process of building and refining models using TensorFlow's high-level APIs. We will start by discussing the architecture of a simple sequential model and how to compile it with the desired optimizer, loss, and metrics. Then, we will delve into the training process, including the use of hardware accelerators and distributed training. Next, we will cover the evaluation of the model on validation data and the importance of using the same processing procedure for both training and test data. Finally, we will touch on the deployment of the model using TensorFlow's model saving format and the possibility of further improving the model's accuracy.

To begin, let's define the architecture of our model. We will use a simple sequential model that consists of modular keras layers. The output of each layer is connected to the input of the next layer. The first layer performs the necessary data transformations, followed by densely connected layers. The final layer outputs the class predictions for the wilderness areas of interest. It's important to note that at this stage, we have only established the architecture of the model and have not yet connected it to any data.

Once the layer architecture is defined, we can compile the model. This step involves adding the optimizer, loss function, and metrics that we are interested in. TensorFlow offers a variety of optimizers and loss functions to choose from, providing flexibility in model optimization.

After compiling the model, we can proceed to the training phase. In real-world scenarios, where large datasets are involved, leveraging hardware accelerators such as GPUs or TPUs is recommended. Additionally, distributing the training process across multiple GPUs or nodes can further enhance performance. TensorFlow provides distribution strategies for this purpose.

Moving on to evaluation, we first need to load the validation data. It's crucial to use the same processing procedure for the test data as we did for the training data to ensure consistent and repeatable results. We can define a function that handles the data transformations and use it for both training and test data. By calling the evaluate method of our model with the validation data, we can obtain the loss and accuracy metrics on the test data.

Once the model is validated on independent held-out data processed in the same way as the training data, we can consider deploying the model. TensorFlow offers tooling for real-world deployment, such as the TensorFlow Extended (TFX) library. Additionally, TensorFlow provides a model saving format that works with other TensorFlow products like TensorFlow Serving and TensorFlow.js. This saved model includes all the weights, variables, and the graph used for training, evaluation, and prediction. It can be loaded back into Python for retraining or reuse.

At this point, we have covered all the critical stages of building a model in TensorFlow. However, if we want to improve the model's accuracy, there are several avenues to explore. We could collect more data, change the data processing procedure, modify the model architecture, experiment with different optimizers and loss functions, or tweak hyperparameters.

To illustrate one possible approach, we will explore TensorFlow's canned estimators. These are built-in implementations of more complex models that may not fit neatly into a layer-based architecture. One example is the DNN linear combined classifier, also known as the wide and deep model. By configuring this estimator with the same feature columns, we can leverage the research and development that went into creating this model structure. It combines traditional linear learning with deep learning, providing a powerful tool for improving model accuracy.

This didactic material has provided an overview of the process of building and refining models using TensorFlow's high-level APIs. We have covered the steps of defining the model architecture, compiling the model with desired optimization parameters, training the model with hardware acceleration and distributed strategies, evaluating the model on validation data, and deploying the model using TensorFlow's model saving

format. We have also discussed the possibility of further improving model accuracy through various means, including the use of canned estimators.

To build and refine our models in TensorFlow, we can use the high-level APIs provided by the framework. In this didactic material, we will focus on the process of building a wide and deep model using these high-level APIs.

First, we need to feed our categorical data directly into the linear half of the model. For the numerical data, we will configure a deep neural network (DNN) with two dense layers. This combination of linear and deep components allows us to capture both simple and complex patterns in our data.

To train the wide and deep model, we can follow a similar process as with our previous models. However, it's important to note that the canned estimator in TensorFlow expects an input function instead of a dataset directly. Estimators have their own sessions and graphs, allowing them to build and replicate graphs as needed during distribution.

In this case, our input function provides instructions for obtaining and processing the dataset, producing the tensors we need. The estimator will call this function in its own graph and session when necessary. We can use a lambda function to wrap our data loading function, preconfiguring the file names so that the estimator can call it at runtime to retrieve the appropriate features and labels.

We can also use a similar strategy to evaluate our model using test data. By running the training for a specified number of epochs, we can obtain a trained model that can be compared to our previous models.

It's worth mentioning that the wide and deep model is just one of the canned estimators available in TensorFlow. There are other canned estimators for tasks such as boosting trees, time series analysis, and more.

When working with estimators, we need to specify the shape and type of tensor to expect at inference time. To do this, we define an input receiver function. This function builds the tensor shapes that we expect when serving the model. Although it may sound confusing, TensorFlow provides a convenience function that simplifies this process. We can use this function by providing the shapes of the tensors we want to run inference on. In eager mode, we can grab a row from our dataset to determine the expected tensor shapes.

It's important to note that in real-world scenarios, the inference data may require additional processing, such as parsing from a live request string. The input receiver function is where we can encode this logic.

Once we have the input receiver function, we can use it to generate a saved model. This saved model can be used in TensorFlow Serving, TensorFlow Hub, and other deployment scenarios, similar to how we used it with Keras.

By using the high-level APIs in TensorFlow, we can easily build and refine our models. The wide and deep model, in particular, allows us to combine linear and deep components to capture both simple and complex patterns in our data. With the canned estimators and input receiver functions, we can train, evaluate, and serve our models efficiently.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW EXTENDED (TFX)
TOPIC: ML ENGINEERING FOR PRODUCTION ML DEPLOYMENTS WITH TFX

ML Engineering for Production ML Deployments with TFX

In this didactic material, we will explore ML engineering for production ML deployments using TensorFlow Extended (TFX). TFX is an open-source platform developed by Google specifically designed for production ML scenarios. This platform addresses the challenges faced when dealing with changing ground truth and data, particularly in real-time scenarios.

Production ML can be categorized into three types based on the rate of change in ground truth and data. In easy problems, where the ground truth and data change slowly (e.g., recognizing cats and dogs), model retraining is usually driven by model improvements, better data, or software/system changes. Labeling is relatively easy in these cases. However, in domains where the ground truth and data change over weeks, model retraining needs to be driven by declining model performance along with improvements. Labeling becomes more challenging in these domains, requiring direct feedback from systems or crowd-based labeling. The most difficult scenarios occur when ground truth and data change rapidly, such as predicting markets or real-time scenarios. In these cases, model retraining and declining model performance go hand-in-hand, and labeling becomes a significant challenge.

To address these challenges, Google developed TFX to cater to their numerous applications that utilize ML in production and mission-critical environments. TFX offers a set of libraries and components that enable the creation of ML pipelines for training and inference. These components can be used individually or combined to form a complete TFX pipeline.

A TFX component represents a task performed in an ML pipeline. Each component has a configuration and takes input artifacts from metadata, except for the initial component that ingests the original dataset. The component performs its task and produces a new artifact, which is then stored back into metadata. Components communicate with each other through input and output channels, with data flowing through the pipeline based on these dependencies.

Metadata plays a crucial role in TFX pipelines, as it allows for tracing the lineage of artifacts, comparing previous training runs, and reusing previously computed outputs. With metadata, one can easily track the data used to train a model, compare different training runs, and avoid re-running components if the input remains unchanged.

TFX leverages Apache Beam to distribute processing over a distributed cluster, enabling efficient and scalable ML engineering for production ML deployments.

TFX provides a comprehensive platform for ML engineering in production scenarios. It addresses the challenges posed by changing ground truth and data, offers a set of libraries and components for building ML pipelines, and utilizes metadata to enable lineage tracing, comparison of training runs, and output reuse.

Apache Beam is a framework that supports distributed clusters, such as Apache Spark, Apache Flink, and Google Cloud DataFlow. It also provides support for multiple programming languages through software development kits (SDKs). In the context of Apache Beam, an SDK is used to define a Beam pipeline, which is then translated to the native API of the target cluster. This allows for the execution of the pipeline on the chosen cluster, producing the desired result.

Now let's discuss the standard components of TensorFlow Extended (TFX), which is a framework for building production-ready machine learning (ML) pipelines. These components can be used as building blocks for ML development, and it's also possible to create custom components.

The first standard component is ExampleGen, which is responsible for ingesting data. It takes the data, performs ingestion operations such as splitting, and saves the processed data as TensorFlow (TF) examples in a TF record file.

Next, StatisticsGen calculates statistics about the data by making a full pass over it. This is useful for understanding the data and detecting any changes when working with new datasets.

SchemaGen calculates a schema for the data, determining the types of each feature and the valid categories for categorical variables.

Example Validator takes the results from StatisticsGen and SchemaGen and checks for problems in the data. It looks for examples that have the wrong type for a particular feature or categorical values that shouldn't be present.

Transform is where feature engineering is performed. It converts the desired feature engineering operations into an input graph that becomes part of the ML model.

Trainer takes the input graph and the model defined using TensorFlow and trains the model.

Evaluator performs a deep analysis of the trained model's performance. It examines individual slices of the data to understand how the model's performance varies across different parts of the dataset.

InfraValidator ensures that the model can be deployed to the serving infrastructure. It checks if the model meets the necessary requirements for deployment.

If both Evaluator and InfraValidator confirm that the model can be deployed and that it outperforms the existing production model, Pusher is responsible for pushing the model to one of the deployment targets, such as TensorFlow Serving, TensorFlow Lite, TensorFlow JS, or TensorFlow Hub.

TFX also supports mobile and IoT deployments through TF Lite. TF Lite support in TFX enables the training, evaluation, validation, and deployment of TF Lite models from TFX pipelines. To use TF Lite in TFX, two changes need to be made: invoking the TF Lite rewriter within Trainer and evaluating the model using the TF Lite model for evaluation.

In addition to the standard components, TFX allows the creation of custom components. There are three ways to create custom components: customizing the executor of an existing component, using a Python function with a decorator and annotations for the arguments, or extending existing component classes.

TFX is also supported on Google Cloud using the Google Cloud AI Platform Pipelines. This allows for the execution of TFX pipelines on Google Kubernetes Engine (GKE) within Kubeflow Pipelines, leveraging the Cloud-managed services available on Google Cloud.

In terms of development, there are three ways to run a TFX pipeline. One approach is to create a complete TFX pipeline in a notebook, such as a Colab notebook. Another approach is to use the TFX CLI, which provides a command-line interface for managing and running TFX pipelines. Finally, TFX can also be integrated with development environments like JupyterLab, allowing for interactive development and debugging of TFX pipelines.

The Chicago Taxi example is a practical demonstration of using TensorFlow Extended (TFX) for ML engineering in production ML deployments. In this example, we will go through the process of creating an interactive context, running components, exploring artifacts, and adding components to the pipeline.

To start, we have a CSV file containing the data for the Chicago Taxi example. The first step is to create the interactive context, which allows us to orchestrate the pipeline by stepping through notebook cells. This context is especially useful for development purposes.

Next, we run the first component, ExampleGen, which ingests the data and splits it. After running each component, we can use the Artifact Explorer to explore the imported artifacts and the resulting outputs. In the case of ExampleGen, we can see the inputs (the CSV file) and the outputs (the split dataset).

Moving forward, we take a Pythonic approach to look at the artifacts and examine individual examples that were ingested. Then, we run the second component, StatisticsGen, which calculates statistics across the dataset. The Artifact Explorer for StatisticsGen allows us to explore the statistics and features of the dataset, including

identifying missing values and valid categorical features.

The next component is SchemaGen, which infers the schema (types) of the features in the dataset. While SchemaGen does its best to infer the correct schema, adjustments can be made if necessary. It is important to review and validate the inferred schema for accuracy.

At this point, we transition to the next style of development, which involves running TFX locally on a desktop or virtual desktop. We use JupyterLab with a notebook and an IDE (such as VS Code) for this purpose. We start by setting up paths, checking the TFX version, and configuring the project directory. Then, we copy the template files into the project directory using the TFX CLI.

It is worth noting that since we are not running in Colab, TFX is already installed in the virtual environment, so we skip the installation step. We change into the project directory and browse the files to ensure everything is in place. Running unit tests on the files is also recommended.

After copying the template into the project directory, we create a pipeline for the Beam DAG runner. At this stage, we are only creating the pipeline and not running it yet. We can verify the pipeline's existence and then proceed to run it using the TFX CLI. The ExampleGen component, which reads from the CSV file, is the only component in the pipeline at this point.

To add more components to the pipeline, we can use an IDE like VS Code. By editing the pipeline.py file, we can easily add components to the pipeline structure. In this example, we are working with the Chicago Taxi template.

This concludes the overview of the Chicago Taxi example and the process of using TFX for ML engineering in production ML deployments. By following these steps, you can effectively develop and deploy ML models using TFX.

TFX (TensorFlow Extended) is a comprehensive platform that provides a set of tools and components for building and deploying machine learning (ML) models in production. In this didactic material, we will explore the fundamentals of TFX and its usage in ML engineering for production ML deployments.

One of the key components of TFX is ExampleGen, which is already part of the pipeline. However, there are other components that need to be added, such as StatisticsGen, SchemaGen, ExampleValidator, and Transform. These components are responsible for generating statistics, validating data, and performing feature engineering.

To add these components to the pipeline, uncomment the relevant code in the configuration file. In the Transform component, there is a user-defined function called preprocessing function, which is used for feature engineering. This function is passed to the Transform component, which then applies the preprocessing or feature engineering steps to the dataset.

To debug and run the pipeline, a Beam DAG runner is used. By setting breakpoints in the code, you can inspect variables and analyze the features of the dataset. This allows for IDE-style development, where you can use notebooks or an IDE to interactively work with the pipeline.

Another approach to running TFX pipelines is by utilizing Google Cloud and Cloud AI Platform Pipelines. This involves using Kubeflow Pipelines as the underlying framework. With this approach, you can create and manage pipelines using a JupyterLab instance and a template similar to the Beam Orchestrator. The CLI is used to create, update, and run pipelines.

Within the Google Cloud environment, you have access to various tools and features. You can visualize pipelines and experiments, examine artifacts, and explore the lineage of artifacts to understand how data flows through the pipeline. These tools provide a comprehensive view of the pipeline and facilitate monitoring and analysis.

TFX is a flexible platform that caters to different layers of ML engineering, including orchestration and metadata. It offers standard components for common production needs, and you can also create custom components to meet specific requirements. TFX has been widely adopted by Google and is recommended for putting ML applications into production.

To learn more about TFX, you can refer to the TFX website and explore the provided links. Additionally, a blog post explaining the history of TFX is available for further reading. If you are interested in deploying ML models in production, TFX is a powerful tool worth considering.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW EXTENDED (TFX)
TOPIC: WHAT EXACTLY IS TFX

TensorFlow Extended (TFX) is a framework developed by Google to help put machine learning models into production. It is designed to address the challenges of building production pipelines and making ML models available to the world. TFX is the default framework for the majority of Google's ML production solutions and has also had a deep impact on partner companies like Twitter, Airbnb, and PayPal.

When developing an ML application, there are several factors to consider. First, it is important to gather labeled data for supervised learning and ensure that the dataset covers a wide range of possible inputs. Dimensionality reduction and maximizing the predictive information in the feature set are also crucial. Fairness and avoiding biases in the application are important considerations. Additionally, rare conditions should be taken into account, especially in fields like healthcare where predictions for rare but important situations may be required. Lastly, it is necessary to plan for the lifecycle management of the data as the solution evolves over time.

In addition to the ML-specific considerations, putting a software application into production requires meeting the requirements of any production software, such as scalability, consistency, modularity, testability, safety, and security. These challenges must be addressed alongside the ML-specific challenges.

TFX allows developers to create production ML pipelines that incorporate the requirements and best practices of production software deployments. The pipeline starts with data ingestion and progresses through data validation, feature engineering, training, evaluation, and serving. TFX provides libraries for each phase of the ML pipeline, including TensorFlow Data Validation, TensorFlow Transform, and TensorFlow Model Analysis. These libraries, along with the TFX pipeline components, enable the creation of customized components.

To manage and optimize pipelines, TFX includes horizontal layers for pipeline storage, configuration, and orchestration. These layers are essential for effectively managing and running applications on the pipelines.

In the next episode, the functioning of TFX pipelines will be discussed in more detail. For further information on TFX, visit tensorflow.org/tfx.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW EXTENDED (TFX)
TOPIC: TFX PIPELINES

TensorFlow Extended (TFX) is a powerful tool that helps in putting machine-learning models into production. In this didactic material, we will explore TFX pipelines and their components, understand their structure, and learn about the role of orchestration in managing these pipelines.

TFX pipelines are built as a sequence of components, each performing a specific task. These components are organized into directed acyclic graphs (DAGs). A TFX component consists of three main parts: a driver, an executor, and a publisher. While the driver and publisher are mostly boilerplate code, the executor is where customization and code insertion take place.

The driver is responsible for coordinating job execution and feeding data to the executor. The publisher updates the metadata store with the results generated by the executor. The executor is where the actual work is done for each component.

To configure a component in TFX, Python is used. Input data for the component is obtained from the metadata store, and the results are written back to the metadata store. As data flows through the pipeline, components read metadata produced by earlier components and write metadata that will be used by downstream components.

Orchestration plays a crucial role in managing TFX pipelines. Task-aware architectures are sufficient if the goal is to kick off the next stage of the pipeline as soon as the previous component finishes. However, for more powerful and efficient pipelines, a task- and data-aware architecture is recommended. This architecture stores all the artifacts of every component over multiple executions, enabling a range of advanced functionalities.

TFX implements a task- and data-aware pipeline architecture, which we will discuss in detail in the next episode. An orchestrator is required to define the sequence of components in the pipeline and manage their execution. TFX provides support for Apache Airflow and Kubeflow as default orchestrators, but it also allows the use of other orchestrators if needed.

In the next episode, we will delve into the role of metadata and how it enhances the capabilities of TFX pipelines. For more information on TFX, please visit tensorflow.org/tfx.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW EXTENDED (TFX)
TOPIC: METADATA

TensorFlow Extended (TFX) is an open-source framework that helps in putting machine learning models into production. It implements a metadata store using ML metadata, which is stored in a relational database. The metadata store stores artifacts, which include trained models, training data, and evaluation results. The data itself is stored outside the database, but the properties and location of the data object are kept in the metadata store.

TFX also keeps execution records for every component each time it is run. This is important because ML pipelines are often run frequently over a long lifetime as new data comes in or as conditions change. Keeping a history of these executions allows for optimization and debugging of the pipeline. Additionally, TFX includes the lineage or provenance of the data objects as they flow through the pipeline. This allows for tracking the origins and results of running components, which is crucial for understanding the impact of changes in data and code.

Having a lineage or provenance of data artifacts enables tracing forward and backward in the pipeline. This is useful for understanding what data was used to train a model or the impact of feature engineering on evaluation metrics. In some cases, this ability to trace data origins and results may be a regulatory or legal requirement. It is also important to note that production solutions are not one-time things. They need to be maintained and evaluated over time as new data is incorporated and models are retrained.

TFX allows for making pipelines more efficient by only rerunning components when necessary and using a warm start to continue training. If a model has already been trained for a day and needs further training, starting from where it left off instead of starting from scratch saves time. Similarly, if the input or code of a component has not changed, the pipeline can reuse the previous result from cache instead of rerunning the component. This is especially beneficial for data pre-processing, which can be expensive in terms of time and resources.

With TFX and ML metadata, reusing components and results is simplified, and the user does not have to manually select which components to run. This not only saves processing time but also provides a simpler run pipeline interface. In the next episode, orchestration and the standard components of TFX will be discussed in detail.

For more information on TFX, visit tensorflow.org/tfx.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW EXTENDED (TFX)
TOPIC: DISTRIBUTED PROCESSING AND COMPONENTS

TensorFlow Extended (TFX) is a framework that helps in putting machine learning models into production. In this episode, we will discuss Distributed Processing and Components.

To handle distributed processing of large amounts of data, especially compute-intensive data like ML workloads, a distributed processing pipeline framework is required. TFX utilizes Apache Beam, which is a unified programming model that can run on various execution engines such as Apache Spark, Apache Flink, and Google Cloud Dataflow. This allows users to use the distributed processing framework of their choice, rather than being limited to a specific one.

TFX components run on top of Apache Beam, enabling users to leverage their existing distributed processing framework or choose a new one. Beam Python can currently run on Flink, Spark, and Dataflow runners, with the addition of new runners in progress. It also includes a local runner, allowing users to run a TFX pipeline on their local system for development purposes.

For example, the Transform component uses Beam to perform feature-engineering transformations like creating a vocabulary or doing PCA. This can be executed on a Flink or Spark cluster, on the Google Cloud using Dataflow, or on a local system. The Trainer component, on the other hand, primarily utilizes TensorFlow for training the model. TFX currently supports 1.X models and TF Estimators.

Some components, like the Pusher component, only require Python to perform their tasks. When all the components are combined and managed by an orchestrator, the pipeline consists of data ingestion on the left and pushing saved models to deployment targets on the right. These deployment targets can include TensorFlow Hub, TensorFlow.js for JavaScript environments, TensorFlow Lite for native mobile applications, and TensorFlow Serving for server farms.

Let's take a closer look at each of the TFX components. First, the input data is ingested using ExampleGen, which runs on Beam. It reads the data, splits it into training and evaluation sets, and formats it as TF examples. The configuration for ExampleGen is simple and requires just two lines of Python code.

The next component, StatisticsGen, performs a full pass over the data using Beam and calculates descriptive statistics for each feature. It leverages the TensorFlow Data Validation library, which includes visualization tools for data exploration and understanding. These tools can be run in a Jupyter notebook, allowing users to identify any data issues.

SchemaGen, another TFX component, uses the TensorFlow Data Validation library to infer the types and range of categories for each feature based on the statistics generated by StatisticsGen. The schema can be adjusted as needed, such as adding new categories that are expected to be present.

ExampleValidator takes the statistics from StatisticsGen and the schema from SchemaGen (or user curation) to identify problems in the data. It looks for anomalies, missing values, or values that do not match the schema, and produces a report of its findings. Since new data is constantly being ingested, it is important to be aware of any problems that arise.

Transform is a more complex component that requires additional configuration and code. It uses Beam for feature engineering, applying transformations to improve the performance of the model. This can include creating vocabularies, bucketizing values, or running PCA on the input data. The code written for Transform depends on the specific feature engineering requirements of the model and dataset.

TFX's Distributed Processing and Components leverage Apache Beam to provide a flexible and scalable framework for putting machine learning models into production. The components, such as ExampleGen, StatisticsGen, SchemaGen, ExampleValidator, and Transform, each play a crucial role in the data ingestion, analysis, and feature engineering stages of the pipeline.

Transform is a crucial component in the TFX (TensorFlow Extended) framework that performs data

preprocessing and feature engineering tasks. It takes in your data and applies various transformations to prepare it for model training and serving.

When running Transform, it will process your data for one full epoch, creating two types of results. For features that require constant calculations such as median or standard deviation, Transform will output a constant value. For features that require different calculations for each example, such as normalization, Transform will output TensorFlow operations. These constants and operations are then used to create a hermetic TensorFlow graph, which contains all the necessary information for applying the transformations.

One of the major advantages of using Transform is that it ensures consistency between the training and serving environments, eliminating training/serving skew. In some cases, when moving a model from the training environment to the serving environment, the feature engineering may not be the same, resulting in discrepancies. Transform solves this issue by using the exact same code for feature engineering, regardless of where the model is run.

Once the Transform stage is complete, the next step is to train the model. The Trainer component takes in the Transform graph, data from Transform, and the schema from SchemaGen, and trains the model using your modeling code. The Trainer component saves two different types of SavedModels: a normal SavedModel for deployment to production, and an EvalSavedModel for analyzing the model's performance.

During the training process, you can monitor and analyze the progress using TensorBoard, which provides visualizations and comparisons of different model-training runs. This is made possible by the ML-Metadata store, which is a key component of TFX.

After training, the Evaluator component analyzes the performance of the model using the EvalSavedModel and the original input data. It goes beyond just looking at the overall results and dives deeper into individual slices of the dataset. This is important because each user's experience with the model depends on their individual data points. The Evaluator component ensures that the model performs well not only on the entire dataset but also on specific data points.

Once the model has been evaluated, the ModelValidator component compares it to the existing model in production using criteria defined by the user. If the new model meets the defined criteria, the Pusher component pushes it to the deployment targets, which could be TensorFlow Lite for mobile applications, TensorFlow.js for JavaScript environments, TensorFlow Serving for server farms, or a combination of these.

The TFX framework, with its components like Transform, Trainer, Evaluator, ModelValidator, and Pusher, enables the development and deployment of machine learning models in a consistent and efficient manner. It provides a seamless pipeline for data preprocessing, model training, evaluation, and deployment. By following this pipeline, you can ensure that your models perform well and meet the criteria for production deployment.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW EXTENDED (TFX)
TOPIC: MODEL UNDERSTANDING AND BUSINESS REALITY

TensorFlow Extended (TFX) is a powerful tool that helps put machine learning models into production. In this final episode of our series on real-world machine learning and production, we will explore how model understanding is crucial for achieving business goals.

TFX, along with TensorFlow model analysis, allows for in-depth analysis of a model's performance. To illustrate the importance of this, let's consider an example of an online retailer selling shoes. They are using a model to predict click-through rates and determine how much inventory to order for each product. Everything seems to be going well until they notice a decline in the model's performance specifically for men's dress shoes.

Now, the retailer faces a dilemma: how much inventory should they order for men's dress shoes? This is a critical decision, especially if these shoes are high-end and represent a significant portion of their business. This is where deep analysis of the model's performance becomes essential, not just once, but on an ongoing basis. TFX enables the creation of pipelines that facilitate continuous and thorough analysis of the model's performance.

It's important to note that overall model performance is not the only factor to consider. Mispredictions on different parts of the data can have varying costs for the business. The available data is rarely the ideal data, and model objectives like AUC serve as proxies for actual business objectives, such as determining inventory levels. Additionally, the real world is dynamic, with data and business conditions constantly changing. Therefore, it is crucial to continuously monitor and analyze how the model reacts to these changes.

To better understand this concept, let's introduce the ML Insights Triangle. When there is a problem with a model's performance for a business, it often indicates a violation of an assumption. The triangle represents three potential assumptions that could be violated:

1. Changes in business realities: Any alterations in the business, such as new suppliers, released products, or shifts in customer behavior, can impact model performance.

2. Issues with data quality: Problems with data can arise from various sources, such as faulty sensors, unreliable service endpoints, broken software updates, or inadequate feature sets for current business conditions.

3. Model-related problems: Sometimes, the issue lies within the model itself. It may require architectural changes, ensemble creation with a rules-based system, or hyperparameter retuning.

When faced with performance issues, the investigation should start with the data. If the data is flawed, nothing else will be right. TFX offers tools and processes to investigate data quality within pipelines. Components like StatisticsGen, SchemaGen, and ExampleValidator, along with TensorFlow Data Validation (TFDV), provide capabilities for identifying outliers, missing values, and changes in feature distributions over time. TFDV also offers visualization tools to compare current data with past data, aiding in the exploration of data patterns.

Additionally, it is crucial to examine specific combinations of features and regions of the loss surface where data may be sparse. Ensuring coverage of the feature space is vital for model performance, and it will evolve as the data changes. In regions where coverage is sparse, additional data collection may be necessary, requiring the creation of new features or the elimination of ineffective ones. Changes in business conditions often drive these shifts.

Another aspect of investigating model performance involves deep analysis. TFX provides tools and processes to conduct in-depth evaluations of a model's performance, but these details are beyond the scope of this material.

TFX and model understanding play a crucial role in putting machine learning models into production. By continuously analyzing a model's performance, businesses can make informed decisions that align with their objectives. TFX's pipelines and built-in tools enable the investigation of data quality and model performance, ensuring that the model remains robust in the face of ever-changing business and data conditions.

To gain deeper insights into the performance of your machine learning model, it is crucial to analyze its behavior on different subsets of your data. TensorFlow Extended (TFX) provides tools like the Evaluator component and TensorFlow Model Analysis (TFMA) to assist you in this process. By examining not only the overall metrics but also the model's performance on individual slices of your data, you can better understand how your model interacts with different parts of your dataset.

When considering which slices to analyze, think about combinations of features and regions of your loss surface that define distinct aspects of your data. It is important to explore edge cases, corner cases, and critical but rare situations. This approach allows you to grasp the nuances of your data and how it relates to your business objectives. TFMA empowers you to explore and evolve your understanding of your data through these tools.

Another valuable tool provided by TFX is the "what-if" tool. This tool enables you to experiment with your data and model by performing what-if scenarios. By making changes to the input data, you can observe how your model responds and gain a better understanding of its behavior. Although the results displayed by the "what-if" tool are not exact due to the use of data samples, they provide approximate insights that can guide your analysis in the right direction. This tool is compatible with both TensorBoard and Jupyter Notebooks and can pull in data from MO Metadata, allowing you to compare current results with past performance.

It is important to recognize that no model is 100% accurate at all times. What truly matters is the impact of mispredictions on your business. To assess the cost of mispredictions, it is essential to combine your model's performance with your business data. By calculating the financial implications of inaccuracies in your model's objectives, which serve as proxies for your business objectives, you can determine the true cost to your organization. This analysis helps you differentiate between minor issues and critical problems that require immediate attention.

TFX offers a comprehensive framework for managing your machine learning models, ML applications, and ultimately your business. Originally developed and used by Google and Alphabet companies for their production ML systems, TFX is now available for everyone to leverage. To learn more about TFX, visit the official website at tensorflow.org/tfx. You can also explore the TFX repositories on GitHub for additional resources. Feel free to leave comments and engage with the community. Thank you for your attention.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW APPLICATIONS
TOPIC: AIR COGNIZER PREDICTING AIR QUALITY WITH ML

Air pollution is a significant issue in Delhi, causing numerous problems for its residents. To address this problem, a group of engineering students developed a mobile application called Air Cognizer. The goal of this application is to make people aware of the air quality around them by allowing users to click a photo of the sky region and obtain information about the air quality in their vicinity.

To accomplish this, the students utilized TensorFlow, a popular machine learning framework. TensorFlow enabled them to perform on-device computing, making the application fast and portable. The first step in developing the application was to capture images and extract relevant features that could be used to determine the Air Quality Index (AQI) level of the atmosphere.

The students collected a dataset of approximately 5,000 images from various locations in Delhi, which they used to train their models. Air Cognizer consists of three models. The first model is an image classifier that determines if the image contains sky or not. The second model is a meteorological pattern database model. The third model is customized for each user and utilizes image parameters to predict the air quality index.

To ensure the application's efficiency and usability, the students utilized TensorFlow Lite. This technology helped compress the size of the models and deploy them on the device. With a binary size of only 10 to 20 kb, Air Cognizer occupies minimal space on the device, consumes very little battery life, and operates quickly.

The students received positive feedback from their first user, who reported that the application worked well. This feedback boosted their confidence in the project and reaffirmed their belief that they were on the right track.

By providing citizens with knowledge about the air quality around them, engineers can contribute to solving the problem of air pollution. Armed with this information, individuals can take appropriate actions to protect their health and the environment.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW APPLICATIONS
TOPIC: HELPING DOCTORS WITHOUT BORDERS STAFF PRESCRIBE ANTIBIOTICS FOR INFECTIONS

Medecins Sans Frontieres (Doctors Without Borders) is a global organization that provides medical care in 70 countries. In their work, they often encounter patients infected with multi-drug resistant bacteria, resulting in the wrong antibiotics being administered. Antibiotic resistance is a growing concern, and if not addressed, it could lead to 10 million deaths per year by 2050.

To effectively treat bacterial infections, it is crucial to identify the specific type of bacteria and determine the most effective antibiotic. However, in many countries where Medecins Sans Frontieres operates, there is a shortage of microbiologists and resources to interpret the test results accurately.

To address this issue, a team developed an application using TensorFlow, computer vision, and machine learning. This application aims to assist lab technicians in interpreting diagnosis test results using their mobile phones. The app utilizes uploaded images of Petri dishes to detect interactions between bacteria and antibiotics.

It is important to note that the application is not meant to replace lab technicians but rather support them in their diagnostic process. The goal is to make the diagnostic test more accessible, affordable, and efficient worldwide.

To train the model, the team used Keras and 15,000 anonymized pictures of diagnosis tests. Surprisingly, they were able to train the model within a few days, thanks to the expressive API provided by TensorFlow. The application is deployed using TensorFlow Lite, enabling offline use on various mobile devices in all Medecins Sans Frontieres clinics.

The team has successfully created a prototype of the application and is excited about its potential impact. They believe that this application can be a game-changer, benefiting millions of people worldwide. By providing accurate and timely diagnosis, the app can help doctors prescribe the most suitable antibiotics, such as in the case of Anwar, a 10-year-old with a highly resistant infection.

The use of artificial intelligence, specifically TensorFlow, in healthcare applications like this one has the potential to revolutionize medical care. By leveraging computer vision and machine learning, lab technicians can receive support in interpreting diagnosis test results, ultimately improving patient outcomes.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW APPLICATIONS
TOPIC: HELPING DOCTORS DETECT RESPIRATORY DISEASES USING MACHINE LEARNING

Sound analysis is a valuable tool in various fields, including music identification and animal classification based on their sounds. In the medical field, physiological sounds play a crucial role. However, the traditional stethoscope, which has remained unchanged for almost two centuries, limits doctors' ability to hear specific frequencies accurately. This outdated method often leads to misdiagnoses. To address this issue, our mission is to leverage machine learning and TensorFlow to revolutionize the diagnosis and treatment of respiratory diseases in low-resource areas like sub-Saharan Africa.

Introducing the Tambua app, a powerful screening tool that enables doctors to make quick decisions. The core technology behind the app aims to mimic the human auditory system. When a patient visits the doctor, lung sounds, symptoms, risk factors, and vital signs are collected. The Tambua app combines this information and provides the doctor with a probability of the patient having a specific respiratory disease.

TensorFlow plays a crucial role in the development and deployment of our machine learning model. By using spectograms, we convert sound data from digital stethoscopes into a visual format that the computer can analyze effectively. We have collaborated with multiple clinics and pathologists, collecting data from 621 patients to train and evaluate our machine learning model.

Once trained and evaluated, our machine learning model is deployed on the Tambua app. TensorFlow Lite enables us to perform inference on mobile devices without requiring a connection to the cloud. This allows doctors to use the Tambua app offline, even in remote areas with limited internet access. Currently, 216 healthcare facilities, including rural clinics, are utilizing the Tambua app and devices.

Medical professionals have expressed their enthusiasm for the Tambua app. By incorporating sound analysis and patient history, doctors can avoid unnecessary procedures like X-rays. Misdiagnosis, a significant problem leading to deaths in Kenya, is being addressed by Tambua. The app provides insights that doctors may have missed using traditional methods.

Respiratory diseases, such as pneumonia, asthma, COPD, and pulmonary tuberculosis, contribute to the deaths of over 2.5 million people annually. By harnessing the power of machine learning, we believe that these diseases can be better managed and treated.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW APPLICATIONS
TOPIC: UTILIZING DEEP LEARNING TO PREDICT EXTREME WEATHER

Extreme weather events, such as heavy rainfall, flooding, and forest fires, are becoming more frequent and severe. Predicting these events accurately is a major challenge we face today. To address this challenge, we have access to vast amounts of climate data, consisting of 100 terabytes every day from satellites, observations, and models. However, analyzing this big data quickly and accurately requires fast and efficient methods.

This is where deep learning, a subfield of artificial intelligence, comes into play. Deep learning is well-suited for problems in climate science due to its ability to handle large amounts of data. Many researchers, including those at NERSC (National Energy Research Scientific Computing Center), use TensorFlow, a popular deep learning framework, to develop models for climate prediction.

In a climate project, TensorFlow was used to create a deep learning model. The researchers started with segmentation models, which have proven successful in satellite imagery segmentation tasks. They then enhanced these models using TensorFlow until they found a set of models that performed well for the specific task at hand.

However, due to the volume and complexity of the climate data, training the models required significant computational resources. In fact, the network used for this project required 14 teraflops of computing power. Training such models on a regular workstation would take months. To tackle these challenges, researchers require access to the largest computational resources available, such as the Summit supercomputer. This state-of-the-art supercomputer is a million times faster than a common laptop and can provide 3.3 exaflops of computing power.

The scalability of TensorFlow was also tested in this project. The researchers were pleasantly surprised by how well it scaled. They were able to run the AI application on thousands of nodes, achieving impressive performance. This was the first time an AI application was run at such a massive scale.

By combining traditional high-performance computing (HPC) with AI, researchers can address complex problems that were previously unimaginable. This includes areas such as fusion reactor research, understanding diseases like Alzheimer's and cancer, and making advancements in genetics, neuroscience, cosmology, and high-energy physics.

The utilization of deep learning, specifically through TensorFlow, is revolutionizing the field of climate science. It enables researchers to analyze vast amounts of climate data quickly and accurately, leading to improved predictions of extreme weather events. Moreover, the integration of AI with traditional HPC allows for the exploration of new frontiers in various scientific disciplines.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW APPLICATIONS
TOPIC: HELPING PALEOGRAPHERS TRANSCRIBE MEDIEVAL TEXT WITH ML

Deciphering and transcribing ancient manuscripts is a time-consuming task that traditionally required a large team of paleographers. However, machine learning has revolutionized this process, making it faster and more efficient. By leveraging the power of TensorFlow, a popular open-source machine learning framework, researchers have been able to develop models that can help paleographers transcribe medieval texts.

Before diving into the machine learning aspect, it is important to note the challenge of collecting data for training these models. Unlike modern images of dogs and cats, there is a scarcity of images of ancient manuscripts available on the internet. To overcome this, the researchers built a custom web application for crowdsourcing data collection. They engaged high school students in this endeavor, enabling them to contribute to the dataset.

In terms of machine learning, the researchers found TensorFlow and its user-friendly interface, Keras, to be the ideal tools for their project. They experimented with various models, starting with binary classification using fully connected networks. Eventually, they settled on a convolutional neural network (CNN) for multiclass classification. CNNs are particularly effective in image recognition tasks, making them well-suited for analyzing individual characters in medieval texts.

The results of their efforts have been impressive. The researchers achieved an average accuracy of 95% in recognizing single characters. This breakthrough will have a significant impact, as it will drastically reduce the time required to transcribe historical information. In a short period, a vast amount of previously inaccessible knowledge will become available.

For Elena Nieddu, one of the researchers involved in the project, solving problems through machine learning is not only a professional pursuit but also a source of personal satisfaction. She sees it as a game against herself, constantly striving to improve and achieve better results.

The application of machine learning, specifically TensorFlow, has revolutionized the process of transcribing medieval texts. Through the use of custom web applications, crowdsourcing, and advanced models like CNNs, researchers have made significant progress in automating this labor-intensive task. The impact of this technology is profound, as it will make a wealth of historical information more accessible than ever before.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW APPLICATIONS
TOPIC: AIRBNB USING ML CATEGORIZE ITS LISTING PHOTOS

Airbnb, an online marketplace, has a vast collection of images of homes, with over 5 million different homes in 81,000 cities, resulting in hundreds of millions of photos. These images play a crucial role in influencing guests' decisions when selecting a home. However, hosts often focus on taking multiple pictures of a single room and neglect to capture images of other rooms. Additionally, the captions provided by hosts are often inaccurate.

To address this challenge, Airbnb turned to machine learning, specifically TensorFlow, to identify and present the content of these images accurately on the site. The primary obstacle was the massive scale of the task, with upwards of half a billion images to process, which would have taken months using traditional methods. By utilizing TensorFlow, Airbnb was able to expedite the process and develop a reasonable model within days.

Airbnb's machine learning platform, called Bighead, played a crucial role in this project. Bighead was designed to be framework-agnostic, allowing the team to leverage TensorFlow for training the model. Furthermore, Bighead facilitated the model lifecycle, feature management, and TensorFlow Serving for serving the model results.

In terms of the model architecture, the team conducted research and determined that ResNet 50, a state-of-the-art performing model, would be suitable for the task. TensorFlow's cross APIs, serving capabilities, and distributed GPU computations were employed to create a pipeline that could efficiently process hundreds of millions of images.

The ultimate goal of this project was to classify images accurately to ensure that guests' initial set of photos showcased the most appealing aspects of a home, such as the living room, bedroom, and swimming pool, rather than focusing solely on less desirable areas like the garage or bathroom. The potential future applications of this technology include the detection of different objects within homes. For example, if users search for specific amenity types on the website, the system can prioritize and display relevant listings.

Machine learning plays a significant role in various aspects of Airbnb's operations, including search ranking, pricing, and predictive booking. As the company continues to develop and implement new models, the number of machine learning models is expected to grow significantly, further enhancing the guest experience and enabling better business decisions.

Airbnb successfully utilized TensorFlow to categorize its extensive collection of listing photos. By employing machine learning at scale, the company can present users with a diverse set of appealing images and potentially expand the application of this technology to detect various objects within homes.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW APPLICATIONS
TOPIC: USING MACHINE LEARNING TO TACKLE CROP DISEASE

It is disheartening to witness the devastation caused by fall armyworm infestations in crop fields. This problem is not limited to Uganda or Africa alone, and if left unaddressed, it could lead to widespread hunger. Farmers who have been affected by this pest have suffered significant losses.

Nazirini Siraji, a software developer, recognized the challenges faced by farmers in Uganda and decided to utilize her skills to help them. At a Google Study Jam, she and her team taught themselves TensorFlow, a powerful machine learning framework. They developed an Android app that utilizes an open-source API to enable farmers to detect infestations early, surpassing the limitations of human observation.

Nazirini's belief in completing what she starts stems from her mother's influence. Despite encountering individuals who discourage women from pursuing careers in technology, she remains determined to prove them wrong. She is committed to making a difference and leveraging technology to empower farmers.

The app developed by Nazirini and her team can identify fall armyworm infestations in real-time, providing farmers with immediate information. Additionally, the app suggests appropriate treatments based on the pest's lifecycle, ultimately saving crops and reducing the need for excessive pesticide use. However, their biggest challenge lies in spreading awareness and ensuring that farmers are aware of the potential benefits of using the app.

This project is just the beginning for Nazirini and her team. They believe that machine learning can revolutionize various sectors, including health and education. While eliminating fall armyworm infestations is currently a difficult task, they remain optimistic that with collective effort and collaboration, it can be achieved.

The impact they have already witnessed by working together as a community is encouraging. Farming plays a vital role in Ugandan culture, and Nazirini takes pride in contributing to the preservation of this essential aspect of life.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW APPLICATIONS
TOPIC: AI HELPING TO PREDICT FLOODS

Floods have become the most common and deadly natural disaster in the world, and many countries lack effective early warning systems and alerts. In India alone, 20 percent of flood fatalities occur. The ability to provide reliable information during a crisis is crucial. Governments, the United Nations, and NGOs have already made significant efforts in this area, and now we are working to build on that work and assist them in their goals.

In India, the government has thousands of people measuring water levels in rivers across the country every hour using stream gauges. While this helps determine if a river will overflow and flood, it doesn't provide information on which specific areas will be affected. This lack of specific information can lead to delayed warnings, leaving people with little time to respond.

Reducing response time is crucial in disaster management. Technological advancements can greatly improve the speed at which warnings are spread. Flood forecasting has been an exploratory project aimed at providing accurate and timely information to those in danger. The main question was whether we could gather enough information to make accurate forecasts that would truly make a difference.

To provide real-time forecasts, we rely on collaboration with the government. By collecting thousands of satellite images, we build a digital model of the terrain. Based on this model, we generate hundreds of thousands of simulations to predict how the river might behave. We then combine these simulations with the measurements provided by the government to produce accurate forecasts.

These forecasts can be sent to individuals using various platforms, such as Search, Maps, and Android notifications. By utilizing these technologies, we can provide alerts with over 90 percent accuracy, giving people more time to prepare and evacuate if necessary. The lead time, or the time between receiving a forecast and the occurrence of the flood, is incredibly important in saving lives.

Collaboration with those who have directly experienced severe floods is crucial in understanding their needs and developing effective solutions. This global collaboration allows us to use technology to make a profound difference in people's lives. Our hope for the future is to give people a few more days of warning before a flood occurs, and to use AI to scale this capability and provide accurate forecasts anywhere in the world.

The Crisis Response and Research team is also exploring the use of AI to provide earlier warnings for other natural disasters, such as fires and earthquake aftershocks. By leveraging technology and AI, we can continue to improve our ability to provide timely and accurate information to those in need.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW APPLICATIONS
TOPIC: POSITIVE CURRENT

In 2014, the city of Flint, Michigan switched their water source from Lake Huron to the Flint River, resulting in lead contamination in the drinking water. This issue particularly affected children, who should not have to drink unclean water. Gitanjali Rao, a scientist and inventor, decided to take action to address this problem.

Rao wanted to create a lead in water detection tool, as the current test for lead took up to two weeks and was expensive and laborious. She came across a new technology that used carbon nanotube sensors and decided to expand on this idea to detect lead in drinking water. Initially unsure if her idea would work, Rao sought guidance and a lab to conduct her experiments.

Denver Water Company recognized Rao's impressive skills and passion, and they decided to partner with her to provide safe drinking water. When they offered her lab space, Rao was ecstatic and knew that if she succeeded, she could help many residents of Flint.

Rao named her device Tethys, after the Greek goddess of fresh water. After going through various versions, Tethys became a 3D printed, fully wired device. To test for lead in water using Tethys, a disposable lead sensor cartridge treated with chloride ions is attached. If the water contains lead, it sticks to the chloride ions, causing resistance to the flow of current. The more resistance, the higher the lead concentration.

To display the results, Rao created an app using Android App Maker. By connecting Tethys to a phone via Bluetooth, users can see if their water is safe, slightly contaminated, or critical.

Rao's device is just one part of the solution to the larger problem in Flint. However, by enabling people to test their water themselves, it empowers them to take action. Rao's dedication and determination serve as an inspiration to others, challenging societal expectations and paving the way for future generations.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW APPLICATIONS
TOPIC: DANIEL AND THE SEA OF SOUND

This exhibit showcases a mobile soundscape, where soundwaves are explored as vibrations in the air. The curator of the exhibit, Daniel, shares his personal connection to sound and music, recalling childhood memories of his parents playing guitar and the impact it had on him. Growing up, Daniel's curiosity and inquisitiveness often got him into trouble, but it also fueled his desire to understand how things work.

Despite not being a good student in high school, Daniel decided to attend community college after graduating. It was there that he discovered his passion for engineering through math classes. Unlike most students who simply wanted equations to calculate numbers, Daniel sought to truly understand the concepts. This led him to delve into physics and the mathematical representation of soundwaves, opening up a whole new world for him.

Daniel's journey took an exciting turn when Cabrillo College organized a symposium connecting students with research institutions. At the symposium, Daniel became intrigued by the work of Danelle and John, who were exploring audio and its connection to music. He realized that his musical background could be valuable in understanding the way sound works.

The scientists at MBARI (Monterey Bay Aquarium Research Institute) were working on a project to listen to and count blue whales using audio recordings from the ocean. However, the sheer amount of data was overwhelming for human analysis. This is where Daniel's expertise in engineering and sound came into play.

Using TensorFlow, a machine-learning tool, Daniel developed software to automatically analyze the recorded audio and identify the calls of blue whales. By converting the sounds into spectrograms, which are visual representations of sound, the team was able to distill the massive amount of data into meaningful patterns. This enabled them to gain insights into the whales' behavior and ecology, ultimately contributing to conservation efforts.

Daniel's journey from the artistry of sound to the science of sound highlights the interconnectedness of these two realms. Music and communication, whether among humans or whales, share a common source: sound energy varying in frequency through time. This realization deepened Daniel's understanding of the power of music as a means of communication.

Despite his initial doubts about his capabilities, Daniel's passion for learning and his ability to overcome challenges have propelled him forward. He acknowledges the difficulties he faces, particularly with severe anxiety, but he sees these struggles as opportunities to view the world differently. Through his work with sound and engineering, Daniel has discovered that even in the darkest times, amazing things can emerge.

Daniel's journey from a curious child to an engineer working with sound and machine learning demonstrates the power of following one's passion and embracing challenges. By combining his musical background with scientific exploration, Daniel has made significant contributions to the understanding of marine life and the conservation of our environment.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW APPLICATIONS
TOPIC: BENEATH THE CANOPY

In the realm of artificial intelligence, there exists a fascinating application known as TensorFlow. This powerful tool allows us to delve into the depths of the forest, exploring its mysteries and safeguarding its inhabitants. Imagine taking an old cell phone and placing it high up in the trees, acting as a vigilant listener to the sounds of the forest. Its purpose? To detect any signs of danger and aid in the preservation of this precious ecosystem.

The TembÃ© people serve as an inspiring example of a community dedicated to protecting their forest. With their exceptional organization and education, they fearlessly embrace new technologies and collaborate with others who share their mission. They understand that the true heroes in these situations are the individuals on the ground, those who possess the knowledge and determination to combat deforestation. However, technology can significantly enhance their efforts, making their work safer and more effective.

The use of old cell phones, often considered obsolete, proves to be a game-changer. These seemingly insignificant devices are, in reality, powerful computers capable of connecting to networks, recording sounds, and performing complex processing tasks. Of course, challenges arise when it comes to powering these devices and ensuring they can pick up sounds from a considerable distance. Nevertheless, such obstacles can be overcome with standard electronics.

The TembÃ© community relies on a limited number of rangers and warriors, approximately 30 in total, to patrol and safeguard their vast forested area. This presents a significant challenge, given the enormity of the land they are responsible for protecting. However, by employing the ability to listen to different parts of the forest continuously, 24/7, the efficiency and safety of their operations are greatly enhanced. This capability becomes a critical tool in their ongoing battle against deforestation.

The intricacies of the forest's soundscape pose a challenge for human detection. The noise, complexity, and distance make it nearly impossible for an individual to identify the sound of a chainsaw from a kilometer away. Enter TensorFlow, an open-source machine learning tool that works wonders in this context. It is capable of detecting the sounds of logging trucks, birds, animals, and even chainsaws within the forest. What is truly astounding is that TensorFlow can uncover sounds that are imperceptible to the human ear. It promptly sends real-time alerts to the rangers, guards, and chiefs, empowering them to respond swiftly and effectively.

For the TembÃ© people, their connection to the forest is rooted in their memories and the places they call home. By utilizing technology like TensorFlow, they can protect and preserve their cherished environment, ensuring that future generations can experience the same sense of belonging and connection.

TensorFlow's application beneath the canopy of the forest exemplifies the immense potential of artificial intelligence in safeguarding our natural world. By repurposing old cell phones and harnessing the power of machine learning, the TembÃ© people and others like them can combat deforestation more effectively and efficiently. With this remarkable tool at their disposal, they can listen to the forest's secrets, detect threats, and take swift action to protect their beloved home.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW APPLICATIONS
TOPIC: USING MACHINE LEARNING TO PREDICT WILDFIRES

Machine learning has proven to be a powerful tool in various fields, and one such application is the prediction of wildfires. In this context, Sanjana Shah and Aditya Shah have developed a solution using machine learning to predict wildfires. Their innovative device, called the Smart Wildfire Sensor, combines weather data with real-time fuel classification to accurately assess the risk of a wildfire occurring.

Traditionally, officials, including firefighters, would manually measure biomass in the field to estimate the potential for wildfires. However, this approach is time-consuming and costly. The Smart Wildfire Sensor aims to streamline this process by providing an automated prediction system.

What sets their device apart is its utilization of TensorFlow, Google's machine learning algorithm. By leveraging TensorFlow, the Smart Wildfire Sensor can predict the appearance of fuel in a forest scene without the need for physical presence. This approach significantly improves efficiency and reduces the resources required for wildfire prediction.

Sanjana and Aditya achieved an impressive 89% accuracy rate with their device. This level of accuracy demonstrates the potential of machine learning algorithms in predicting wildfires. By accurately assessing the risk beforehand, officials can take proactive measures to prevent and mitigate wildfires, ensuring the safety of both people and natural resources.

The Smart Wildfire Sensor developed by Sanjana Shah and Aditya Shah utilizes machine learning, specifically TensorFlow, to predict wildfires by combining weather data and real-time fuel classification. With its high accuracy rate, this device has the potential to revolutionize wildfire prediction and prevention efforts, enabling officials to respond promptly and effectively.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW APPLICATIONS
TOPIC: TRACKING ASTEROIDS WITH MACHINE LEARNING

Artificial Intelligence (AI) has revolutionized various fields, including space exploration and risk assessment. In the context of tracking asteroids, machine learning techniques powered by TensorFlow, an open-source framework developed by Google, are employed to predict the likelihood of asteroids colliding with Earth.

Our planet is surrounded by thousands of asteroids and comets known as Near Earth Objects (NEOs). To prevent potential catastrophic events, NASA initiated a challenge to classify these NEOs accurately. TensorFlow, with its powerful capabilities, plays a crucial role in this endeavor.

One of the applications utilizing TensorFlow is Deep Asteroid, a program designed by Gema Parreno. Deep Asteroid utilizes machine learning algorithms to process vast amounts of data and classify NEOs. By employing a multi-layered approach, Deep Asteroid enhances the accuracy of classification, providing scientists with refined results and valuable insights.

The more layers Deep Asteroid incorporates, the more refined and informed the results become. This process adds knowledge and aids scientists in better understanding the characteristics and trajectories of asteroids. By leveraging TensorFlow's capabilities, Deep Asteroid contributes to our understanding of the potential risks posed by asteroids and helps in devising strategies to mitigate them.

It is important to note that the probability of an asteroid colliding with Earth is extremely low. Therefore, thanks to the efforts of scientists like Gema Parreno and the utilization of machine learning techniques, we can rest assured that our planet is safe from such threats, at least for the time being.

TensorFlow, an AI framework, is instrumental in tracking asteroids and predicting potential collisions with Earth. Deep Asteroid, a program developed using TensorFlow, employs machine learning algorithms to process vast amounts of data and classify NEOs accurately. By refining the classification process through the use of multiple layers, Deep Asteroid enhances our understanding of asteroids and contributes to the prevention of catastrophic events.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW APPLICATIONS
TOPIC: IDENTIFYING POTHOLES ON LOS ANGELES ROADS WITH ML

Artificial Intelligence - TensorFlow Fundamentals - TensorFlow Applications - Identifying potholes on Los Angeles roads with ML

Identifying potholes on the streets of Los Angeles is a challenging task due to the vast road network and the time-consuming manual inspection process. In an effort to address this issue, Alejandra Vasquez and Ericson Hernandez, while studying at LMU, utilized machine learning techniques to develop a faster and more efficient method for identifying potholes throughout the city.

To begin their project, Vasquez and Hernandez recognized the need for data. They equipped a car with a camera and drove around Los Angeles, capturing footage of various roads and freeways. This footage served as the foundation for their machine learning model.

The duo leveraged TensorFlow, an open-source machine learning tool developed by Google, to train their model. TensorFlow allowed them to analyze the captured footage and develop a model capable of accurately identifying not only potholes but also road cracks and other anomalies with a high rate of accuracy.

By employing machine learning, Vasquez and Hernandez have provided a solution that significantly reduces the time spent on identifying potholes. This means that construction workers can devote more time to fixing the identified issues rather than searching for them manually or relying on tips from the public.

The application of TensorFlow in this project showcases the power of artificial intelligence and its potential to revolutionize various industries. By harnessing the capabilities of machine learning, cities like Los Angeles can benefit from efficient and effective infrastructure maintenance, ultimately enhancing the safety and quality of their road networks.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW APPLICATIONS
TOPIC: DANCE LIKE, AN APP THAT HELPS USERS LEARN HOW TO DANCE USING MACHINE LEARNING

Dance Like is an app that utilizes TensorFlow, an artificial intelligence framework, to help users learn how to dance using their mobile phones. By leveraging the power of TensorFlow, the app can analyze body pose through the smartphone camera, making it a powerful tool for dance instruction.

One of the key features of Dance Like is its implementation of an advanced model for pose segmentation. Developed by a team at Google, this model was converted into TensorFlow Lite, allowing for direct usage within the app. This enables the app to run AI and machine learning models that detect body parts, a computationally expensive process that requires the use of on-device GPUs. The TensorFlow library plays a crucial role in leveraging the device's computing resources, providing users with a seamless and high-quality experience.

Teaching dance is just one application of Dance Like and TensorFlow. Any activity involving movement can benefit from this technology. By empowering individuals with expertise to teach others, artificial intelligence acts as a facilitator, bridging the gap between knowledge and learners. This combination of human skill and AI has the potential to be truly transformative.

Dance Like is an innovative app that utilizes TensorFlow to help users learn how to dance. By analyzing body pose through the smartphone camera, leveraging on-device GPUs, and empowering skilled individuals to teach others, Dance Like demonstrates the potential of artificial intelligence in enhancing learning experiences.

EITC/AI/TFF TENSORFLOW FUNDAMENTALS DIDACTIC MATERIALS
LESSON: TENSORFLOW APPLICATIONS
TOPIC: HOW MACHINE LEARNING IS BEING USED TO HELP SAVE THE WORLD'S BEES

Bees play a crucial role in the world's ecosystem, particularly in the pollination of plants, including the fruits and vegetables that we consume daily. However, bees and other insects are facing a global decline, and the reasons behind this decline remain unknown. To address this issue, researchers are utilizing machine learning techniques to gain a deeper understanding of bee behavior and their relationship with their environment.

One such initiative involves the development of a hive monitor equipped with a camera that tracks the activities of bees as they enter and exit the hive. By observing the number of bees leaving the hive and comparing it to the number returning, researchers can identify potential issues within the local ecosystem. If a significant discrepancy exists, it suggests a problem that needs to be addressed.

To analyze the vast amount of data collected from the hive monitor, researchers employ TensorFlow, an open-source machine learning framework developed by Google. TensorFlow allows them to train models that can examine the video footage and extract valuable insights about bee behavior and patterns.

The data collected through the hive monitor and analyzed using TensorFlow is then shared with experts in the field. These experts can utilize the information to make informed decisions regarding various aspects of land management, such as determining the optimal timing for lawn mowing or identifying suitable locations for planting trees and flowers. By incorporating this knowledge, experts can create environments that support and sustain bee populations, thereby helping to preserve the delicate balance of our ecosystem.

While the hive monitor and machine learning techniques represent significant advancements, it is essential to recognize that human intervention is still required. The technology serves as a tool to gather and analyze data, but it is up to us to take action based on the insights gained. By prioritizing nature and utilizing the information provided by these technologies, we can work together to safeguard the future of bees and the vital role they play in our world.