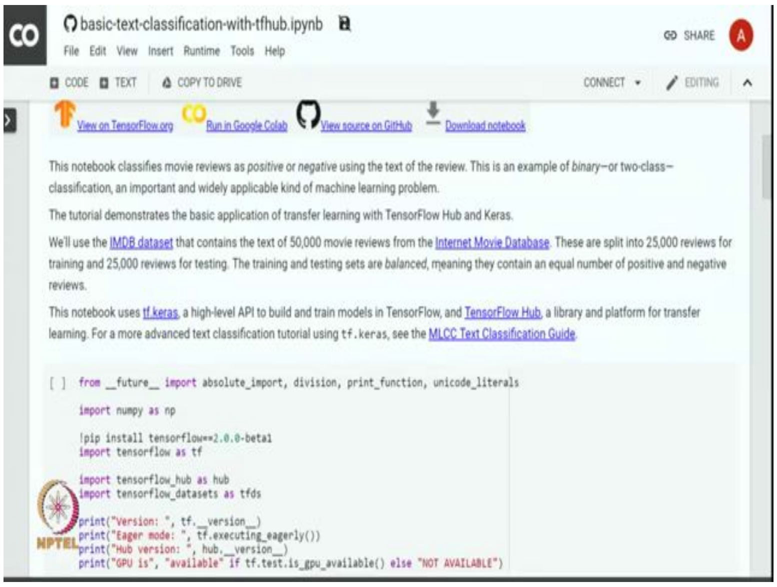
Practical Machine Learning with TensorFlow
Dr. Ashish Tendulkar
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 17
Text Classification

So far in this course we use TensorFlow API to build models for image classification and regression. In this module, we will demonstrate how to use TensorFlow API to build model for text data. In this module, we will use TensorFlow API to classify the movie reviews into positive or negative reviews based on internet movie review data set.

(Refer Slide Time: 00:52)



This notebook classifies movie reviews as *positive* or *negative* using the text of the review. This is an example of *binary*—or two-class—classification, an important and widely applicable kind of machine learning problem.

The tutorial demonstrates the basic application of transfer learning with TensorFlow Hub and Keras.

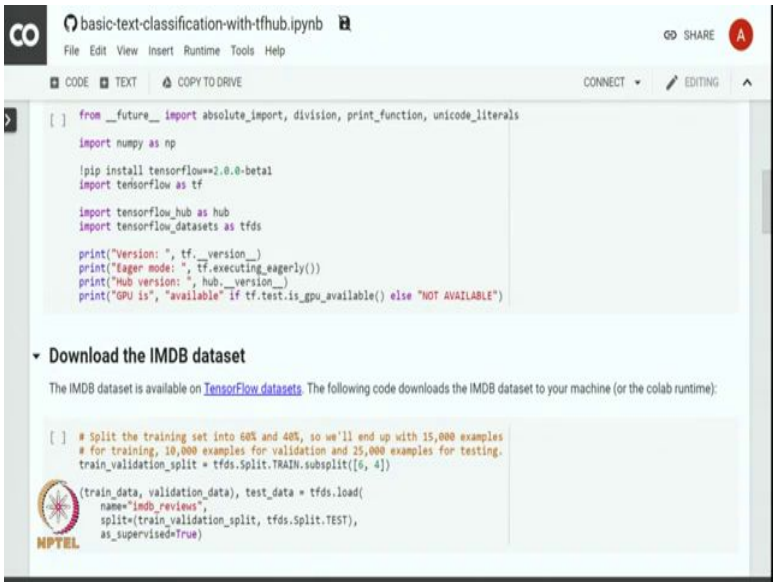
We'll use the IMDB dataset that contains the text of 50,000 movie reviews from the Internet Movie Database. These are split into 25,000 reviews for training and 25,000 reviews for testing. The training and testing sets are *balanced*, meaning they contain an equal number of positive and negative reviews.

This notebook uses tf.keras, a high-level API to build and train models in TensorFlow, and TensorFlow Hub, a library and platform for transfer learning. For a more advanced text classification tutorial using `tf.keras`, see the MLCC Text Classification Guide.

```
from __future__ import absolute_import, division, print_function, unicode_literals

import numpy as np

!pip install tensorflow==2.0.0-beta1
import tensorflow as tf

import tensorflow_hub as hub
import tensorflow_datasets as tfds

print("Version: ", tf.__version__)
print("Eager mode: ", tf.executing_eagerly())
print("Hub version: ", hub.__version__)
print("GPU is", "available" if tf.test.is_gpu_available() else "NOT AVAILABLE")
```

So, we have 50000 movie reviews in total, we use 25000 reviews for training and the remaining 25000 are used for testing. In this example, we will use transfer learning with TensorFlow hub and Keras.

(Refer Slide Time: 01:11)


basic-text-classification-with-tfhub.ipynb
File Edit View Insert Runtime Tools Help
CODE TEXT COPY TO DRIVE
SHARE
CONNECT
EDITING

```
from __future__ import absolute_import, division, print_function, unicode_literals

import numpy as np

!pip install tensorflow==2.0.0-beta1
import tensorflow as tf

import tensorflow_hub as hub
import tensorflow_datasets as tfds

print("Version: ", tf.__version__)
print("Eager mode: ", tf.executing_eagerly())
print("Hub version: ", hub.__version__)
print("GPU is", "available" if tf.test.is_gpu_available() else "NOT AVAILABLE")
```

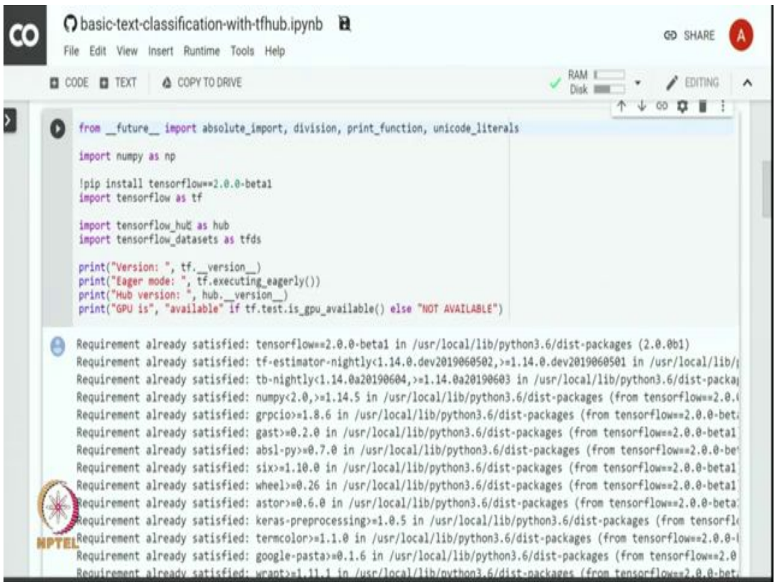
Download the IMDB dataset

The IMDB dataset is available on TensorFlow datasets. The following code downloads the IMDB dataset to your machine (or the colab runtime):

```
# Split the training set into 60% and 40%, so we'll end up with 15,000 examples
# for training, 10,000 examples for validation and 25,000 examples for testing.
train_validation_split = tfds.Split.TRAIN.subsplit([6, 4])

(train_data, validation_data), test_data = tfds.load(
    name="imdb_reviews",
    split=(train_validation_split, tfds.Split.TEST),
    as_supervised=True)
```

Let us install all the required libraries and tensorflow version 2.0, we use numpy for data manipulation and we use tensorflow hub for pre-trained models.

(Refer Slide Time: 01:30)

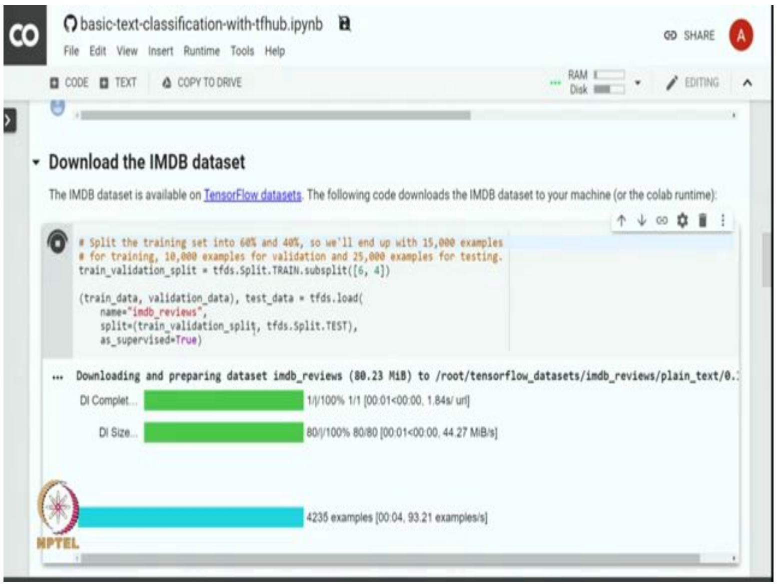

basic-text-classification-with-tfhub.ipynb
File Edit View Insert Runtime Tools Help
CODE TEXT COPY TO DRIVE
RAM
Disk
SHARE
EDITING

```
from __future__ import absolute_import, division, print_function, unicode_literals

import numpy as np

!pip install tensorflow==2.0.0-beta1
import tensorflow as tf

import tensorflow_hub as hub
import tensorflow_datasets as tfds

print("Version: ", tf.__version__)
print("Eager mode: ", tf.executing_eagerly())
print("Hub version: ", hub.__version__)
print("GPU is", "available" if tf.test.is_gpu_available() else "NOT AVAILABLE")
```

```
Requirement already satisfied: tensorflow==2.0.0-beta1 in /usr/local/lib/python3.6/dist-packages (2.0.0b1)
Requirement already satisfied: tf-estimator-nightly<1.14.0.dev2019060502,>=1.14.0.dev2019060501 in /usr/local/lib/
Requirement already satisfied: tb-nightly<1.14.0a20190604,>=1.14.0a20190603 in /usr/local/lib/python3.6/dist-packa
Requirement already satisfied: numpy<2.0,>=1.14.5 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.
Requirement already satisfied: grpcio>=1.8.6 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-bet
Requirement already satisfied: gast>=0.2.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-beta1
Requirement already satisfied: absl-py>=0.7.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-be
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-beta1
Requirement already satisfied: wheel>=0.26 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-beta1
Requirement already satisfied: astor>=0.6.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-beta
Requirement already satisfied: keras-preprocessing>=1.0.5 in /usr/local/lib/python3.6/dist-packages (from tensorfl
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-
Requirement already satisfied: google-pasta>=0.1.6 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0
Requirement already satisfied: wrapt>=1.11.1 in /usr/local/lib/python3.6/dist-packages (from tensorflow==2.0.0-bet
```

One of the beauties of neural networks is that we can use neural network models trained on one particular task for some other task. For example, a model trained on image can be used to perform classification of some other images. So, this is called as transfer learning. The model that was previously trained on some data set is called a pre-trained model and it is used as a black box in some other model.

(Refer Slide Time: 02:09)


basic-text-classification-with-tfhub.ipynb
File Edit View Insert Runtime Tools Help
CODE TEXT COPY TO DRIVE
RAM Disk
EDITING
SHARE

Download the IMDB dataset

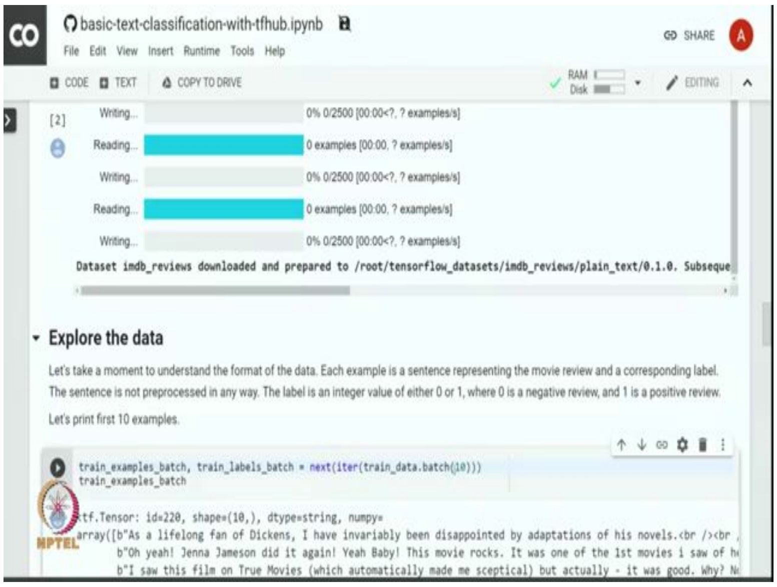
The IMDB dataset is available on TensorFlow datasets. The following code downloads the IMDB dataset to your machine (or the colab runtime):

```
# Split the training set into 60% and 40%, so we'll end up with 15,000 examples
# for training, 10,000 examples for validation and 25,000 examples for testing.
train_validation_split = tfds.Split.TRAIN.subsplit([6, 4])

(train_data, validation_data), test_data = tfds.load(
    name="imdb_reviews",
    split=(train_validation_split, tfds.Split.TEST),
    as_supervised=True)
```

```
Downloading and preparing dataset imdb_reviews (80.23 MiB) to /root/tensorflow_datasets/imdb_reviews/plain_text/0.
Dl Complet...  1/|/100% 1/1 [00:01<00:00, 1.84s/ url]
Dl Size...     80/|/100% 80/80 [00:01<00:00, 44.27 MiB/s]

               4235 examples [00:04, 93.21 examples/s]
```

We can see that TensorFlow 2.0 is now installed. Now, we will download the IMDB dataset and split that into training and validation split of 60 to 40 percent, 60 percent training 40 percent into validation. We are using TensorFlow datasets load method to load the IMDB reviews from the internet.

(Refer Slide Time: 02:34)


basic-text-classification-with-tfhub.ipynb
File Edit View Insert Runtime Tools Help
SHARE
CODE TEXT COPY TO DRIVE
RAM Disk
EDITING

```
[2]   Writing...   0% 0/2500 [00:00<?, ? examples/s]
      Reading...   0 examples [00:00, ? examples/s]
      Writing...   0% 0/2500 [00:00<?, ? examples/s]
      Reading...   0 examples [00:00, ? examples/s]
      Writing...   0% 0/2500 [00:00<?, ? examples/s]
Dataset imdb_reviews downloaded and prepared to /root/tensorflow_datasets/imdb_reviews/plain_text/0.1.0. Subseque
```

Explore the data

Let's take a moment to understand the format of the data. Each example is a sentence representing the movie review and a corresponding label. The sentence is not preprocessed in any way. The label is an integer value of either 0 or 1, where 0 is a negative review, and 1 is a positive review.

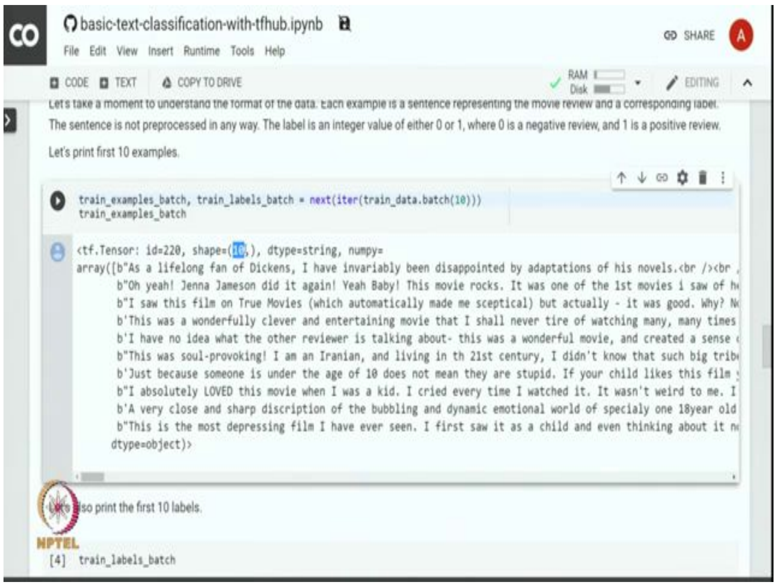
Let's print first 10 examples.

```
train_examples_batch, train_labels_batch = next(iter(train_data.batch(10)))
train_examples_batch
```

```
<tf.Tensor: id=220, shape=(10,), dtype=string, numpy=
array([b"As a lifelong fan of Dickens, I have invariably been disappointed by adaptations of his novels.<br /><br ,
       b"Oh yeah! Jenna Jameson did it again! Yeah Baby! This movie rocks. It was one of the 1st movies i saw of h
       b"I saw this film on True Movies (which automatically made me sceptical) but actually - it was good. Why? N
```

Let us print the first 10 examples to see how the data looks like. We do that with the batch function.

(Refer Slide Time: 02:43)


basic-text-classification-with-tfhub.ipynb
File Edit View Insert Runtime Tools Help
SHARE
CODE TEXT COPY TO DRIVE
RAM Disk
EDITING

Let's take a moment to understand the format of the data. Each example is a sentence representing the movie review and a corresponding label. The sentence is not preprocessed in any way. The label is an integer value of either 0 or 1, where 0 is a negative review, and 1 is a positive review.

Let's print first 10 examples.

```
train_examples_batch, train_labels_batch = next(iter(train_data.batch(10)))
train_examples_batch
```

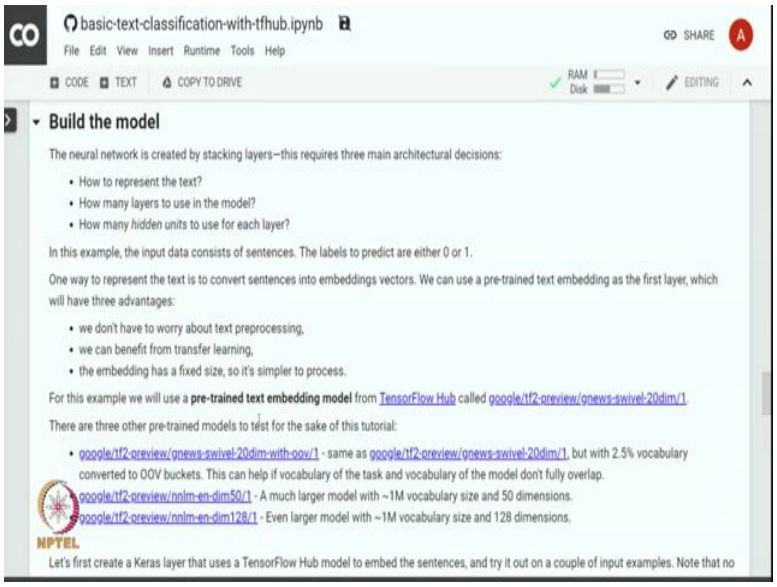
```
<tf.Tensor: id=220, shape=(10,), dtype=string, numpy=
array([b"As a lifelong fan of Dickens, I have invariably been disappointed by adaptations of his novels.<br /><br ,
       b"Oh yeah! Jenna Jameson did it again! Yeah Baby! This movie rocks. It was one of the 1st movies i saw of h
       b"I saw this film on True Movies (which automatically made me sceptical) but actually - it was good. Why? N
       b'This was a wonderfully clever and entertaining movie that I shall never tire of watching many, many times
       b'I have no idea what the other reviewer is talking about- this was a wonderful movie, and created a sense
       b"This was soul-provoking! I am an Iranian, and living in th 21st century, I didn't know that such big trib
       b'Just because someone is under the age of 10 does not mean they are stupid. If your child likes this film
       b"I absolutely LOVED this movie when I was a kid. I cried every time I watched it. It wasn't weird to me. I
       b'A very close and sharp discription of the bubbling and dynamic emotional world of specialy one 18year old
       b"This is the most depressing film I have ever seen. I first saw it as a child and even thinking about it n
      dtype=object)>
```

lso print the first 10 labels.

```
[4]  train_labels_batch
```

So, you can see that each review is on a single line and so you can see that there are 10 reviews over here, each review is in a single line and there are labels in the label batch. So, you can see that most of the reviews in the first 10 reviews are positive and except for a couple of them which are negative. This is also 1D tensor with shape 10.

(Refer Slide Time: 03:22)


basic-text-classification-with-tfhub.ipynb
File Edit View Insert Runtime Tools Help
SHARE
CODE TEXT COPY TO DRIVE
RAM Disk
EDITING
NPTEL

Build the model

The neural network is created by stacking layers—this requires three main architectural decisions:

- How to represent the text?
- How many layers to use in the model?
- How many *hidden units* to use for each layer?

In this example, the input data consists of sentences. The labels to predict are either 0 or 1.

One way to represent the text is to convert sentences into embeddings vectors. We can use a pre-trained text embedding as the first layer, which will have three advantages:

- we don't have to worry about text preprocessing,
- we can benefit from transfer learning,
- the embedding has a fixed size, so it's simpler to process.

For this example we will use a **pre-trained text embedding model** from TensorFlow Hub called google/tf2-preview/gnews-swivel-20dim/1.

There are three other pre-trained models to test for the sake of this tutorial:

- google/tf2-preview/gnews-swivel-20dim-with-oov/1 - same as google/tf2-preview/gnews-swivel-20dim/1, but with 2.5% vocabulary converted to OOV buckets. This can help if vocabulary of the task and vocabulary of the model don't fully overlap.
- google/tf2-preview/nnlm-en-dim50/1 - A much larger model with ~1M vocabulary size and 50 dimensions.
- google/tf2-preview/nnlm-en-dim128/1 - Even larger model with ~1M vocabulary size and 128 dimensions.

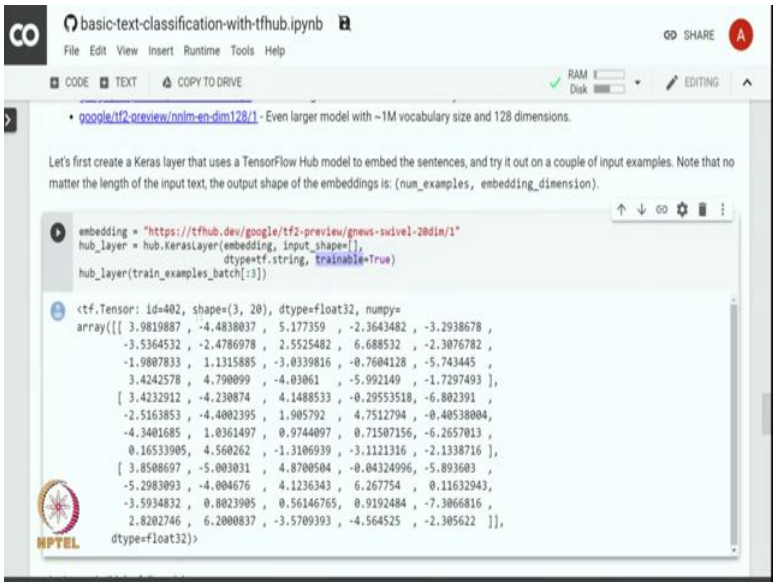
Let's first create a Keras layer that uses a TensorFlow Hub model to embed the sentences, and try it out on a couple of input examples. Note that no

Now, that we have explored the data the next task is to build a neural network model. There are 3 main decisions when we decide to build a neural network model. The first decision is to figure out how to represent a text, then how many layers should we use in the model and how many hidden units should be should we use in each of the layer. In this example, the input data consist of sentences and the output label output label is binary which is either 0 or 1, 0 represents the review is negative and 1 represents that the review is positive.

So, one way to represent a text is to convert the sentences into embeddings vector. This is where we can use some of the pre-trained text embeddings at the as the first layer. This has got multiple advantages. We do not have to worry about text processing. We can benefit from transfer learning and embedding has a fixed size. So, it is simple to process.

So, we will use a pre trained text embedding model from TensorFlow hub. Let us look at what TensorFlow hub is. TensorFlow hub has a number of reusable neural network models that can be used as block black box models in other applications. We will use, we will use a text embedding model based on google news which embeds a given sentence in a 20 dimensional vector.

(Refer Slide Time: 05:02)


basic-text-classification-with-tfhub.ipynb
File Edit View Insert Runtime Tools Help
CODE TEXT COPY TO DRIVE
RAM Disk
EDITING
SHARE

- google/tf2-preview/nnlm-en-dim128/1 - Even larger model with ~1M vocabulary size and 128 dimensions.

Let's first create a Keras layer that uses a TensorFlow Hub model to embed the sentences, and try it out on a couple of input examples. Note that no matter the length of the input text, the output shape of the embeddings is: (num_examples, embedding_dimension).

```
embedding = "https://tfhub.dev/google/tf2-preview/gnews-swivel-20dim/1"
hub_layer = hub.KerasLayer(embedding, input_shape=[],
                           dtype=tf.string, trainable=True)
hub_layer(train_examples_batch[:3])
```

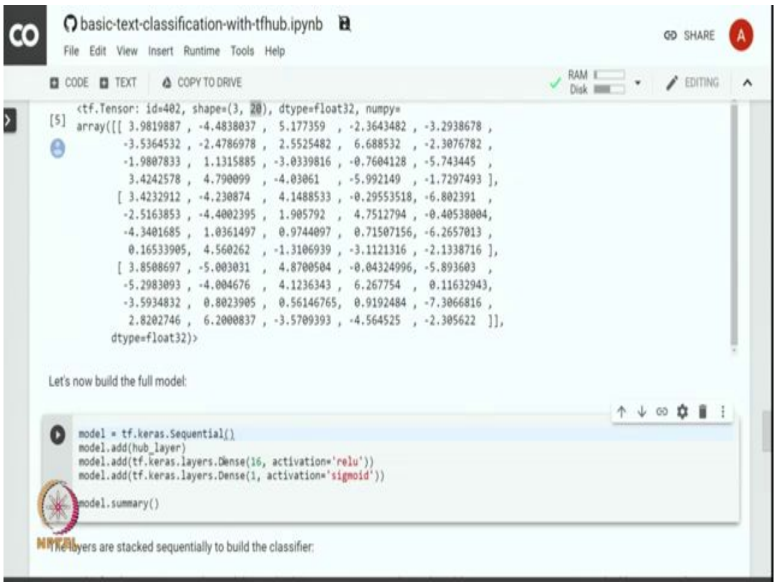
```
<tf.Tensor: id=402, shape=(3, 20), dtype=float32, numpy=
array([[ 3.9819887 , -4.4838037 ,  5.177359  , -2.3643482 , -3.2938678 ,
        -3.5364532 , -2.4786978 ,  2.5525482 ,  6.688532  , -2.3076782 ,
        -1.9807833 ,  1.1315885 , -3.0339816 , -0.7604128 , -5.743445  ,
         3.4242578 ,  4.790099  , -4.03061   , -5.992149  , -1.7297493 ],
       [ 3.4232912 , -4.230874  ,  4.1488533 , -0.29553518, -6.802391  ,
        -2.5163853 , -4.4002395 ,  1.905792  ,  4.7512794 , -0.40538004,
        -4.3401685 ,  1.0361497 ,  0.9744097 ,  0.71507156, -6.2657013 ,
         0.16533905,  4.560262  , -1.3106939 , -3.1121316 , -2.1338716 ],
       [ 3.8508697 , -5.003031  ,  4.8700504 , -0.04324996, -5.893603  ,
        -5.2983093 , -4.004676  ,  4.1236343 ,  6.267754  ,  0.11632943,
        -3.5934832 ,  0.8023905 ,  0.56146765,  0.9192484 , -7.3066816 ,
         2.8202746 ,  6.2000837 , -3.5709393 , -4.564525  , -2.305622  ]],
      dtype=float32)>
```

There are other embedding models that are also available on the TensorFlow hub, but for this particular exercise we will use a google news based text embedding model, we will use google news based text embedding model. Let us create, let us first create a Keras layer that uses TensorFlow hub model to embed the sentences. So, we can define that using hub.KerasLayer, we specify the model that we are using for embedding by a URL of that particular model.

We specify the input shape, we also specify the data type that is string and we specify whether the model is trainable or not. So, in this case you want to retrain the model that is why we set trainable to be true. And what we will do is we will take first 3 examples and see what happens when we pass these examples through the hub layer.

So, you can see that as we pass these 3 examples through the hub layer we get a tensor which is a 2D tensor which has got 3 examples and each example is represented by a 20 dimensional vector. Each vector is a real number either positive or negative and each entry in the vector is a 32 bit floating point number. Let us build a full model and let us see how this particular hub layer fits inside the full model.

(Refer Slide Time: 06:37)



```
[5] <tf.Tensor: id=402, shape=(3, 20), dtype=float32, numpy=
    array([[ 3.9819887 , -4.4838037 ,  5.177359  , -2.3643482 , -3.2938678 ,
            -3.5364532 , -2.4786978 ,  2.5525482 ,  6.688532  , -2.3076782 ,
            -1.9807833 ,  1.1315885 , -3.0339816 , -0.7604128 , -5.743445  ,
             3.4242578 ,  4.790099  , -4.03061   , -5.992149  , -1.7297493 ],
           [ 3.4232912 , -4.230874  ,  4.1488533 , -0.29553518, -6.802391  ,
            -2.5163853 , -4.4002395 ,  1.905792  ,  4.7512794 , -0.40538004,
            -4.3401685 ,  1.0361497 ,  0.9744097 ,  0.71507156, -6.2657013 ,
             0.16533905,  4.560262  , -1.3106939 , -3.1121316 , -2.1338716 ],
           [ 3.8508697 , -5.003031  ,  4.8700504 , -0.04324996, -5.893603  ,
            -5.2983093 , -4.004676  ,  4.1236343 ,  6.267754  ,  0.11632943,
            -3.5934832 ,  0.8023905 ,  0.56146765,  0.9192484 , -7.3066816 ,
             2.8202746 ,  6.2000837 , -3.5709393 , -4.564525  , -2.305622  ]],
          dtype=float32)>
```

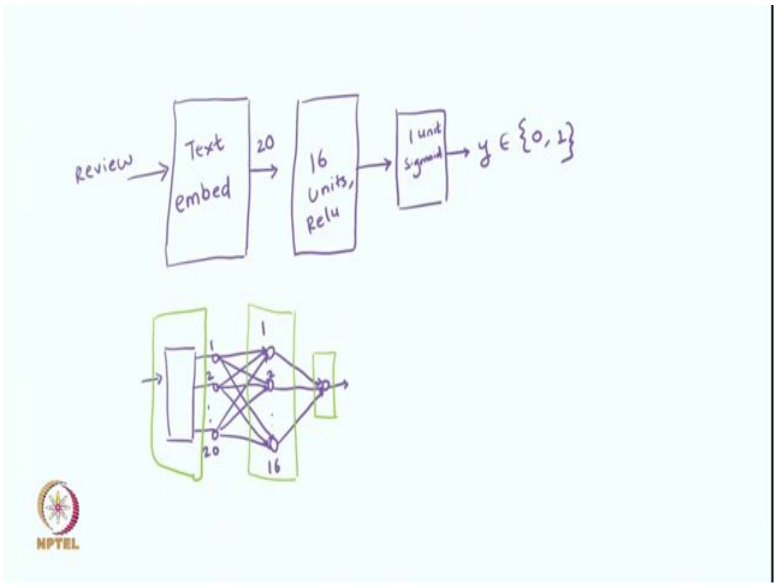
Let's now build the full model:

```
model = tf.keras.Sequential()
model.add(hub_layer)
model.add(tf.keras.layers.Dense(16, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

model.summary()
```

The layers are stacked sequentially to build the classifier:

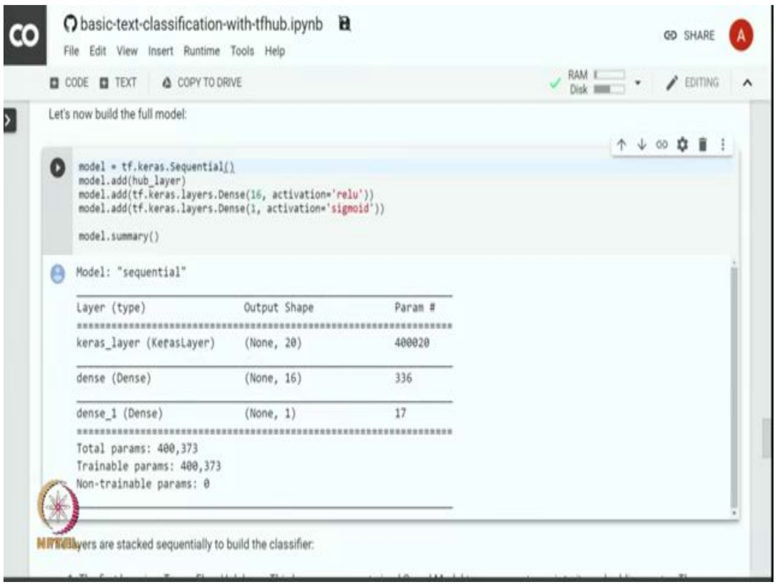
Now, you are quite familiar with the sequential models in Keras. So, we will add the hub layer as the first layer to convert the sentence into the desired embedding. Later we will take the output of this particular embedding and give it to the second layer which is a hidden layer with 16 units which uses relu as an activation function. Finally, we have an output layer which is a dense layer with a single unit because we have binary classification problem here and it uses activation as sigmoid. So, let us quickly look at the architecture of this model.

(Refer Slide Time: 07:27)



The text embedding model takes a review as input and it gives us for every review, 20 numbers.. We send these 20 numbers to a hidden layer with 16 units and it uses relu as an activation function and then it goes to a single unit output layer with sigmoid activation which gives us y which gives us either 0 or 1.

(Refer Slide Time: 09:29)


basic-text-classification-with-tfhub.ipynb
File Edit View Insert Runtime Tools Help
CODE TEXT COPY TO DRIVE
SHARE
RAM Disk
EDITING

Let's now build the full model:

```
model = tf.keras.Sequential()
model.add(hub_layer)
model.add(tf.keras.layers.Dense(16, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

model.summary()
```

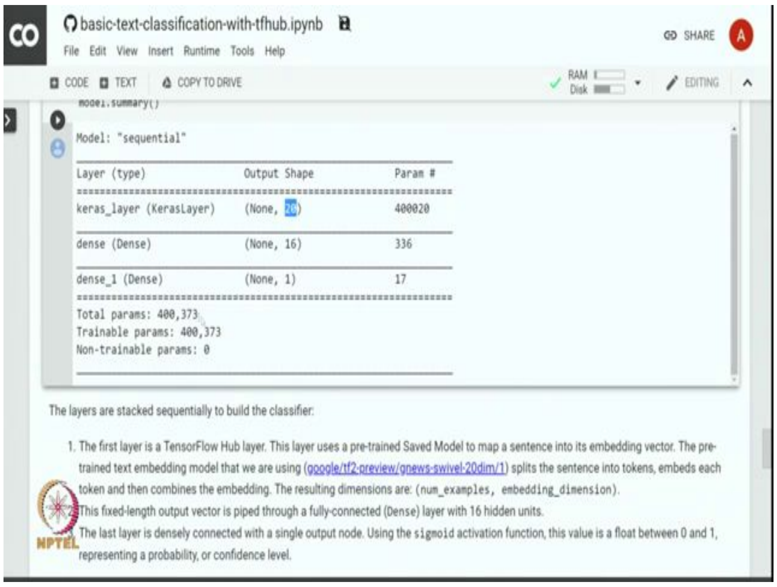
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer (KerasLayer)     (None, 20)                400020
_________________________________________________________________
dense (Dense)                (None, 16)                336
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 17
=================================================================
Total params: 400,373
Trainable params: 400,373
Non-trainable params: 0
_________________________________________________________________
```

...yers are stacked sequentially to build the classifier:

Now, that we have built a classifier let us run this and see the summary of the model. So, we can see that there is a keras layer which is an embedding layer which outputs 20 numbers which go to the dense layer, which outputs 16 numbers and then we have an output layer which outputs a single number which is the prediction. And the number of parameters in the keras layer are about 400k, then 336 parameter in the first in the hidden layer and 17 parameters in the dense layer.

We can clearly see that number of parameters for the output layer is 17 because there are 16 inputs and 1 bias term. In the same manner, you can see that 336 comes from 16 into 20 plus 16 bias units corresponding to each of the units in the hidden layer. And then keras layer has 400 k parameters. So, each of these units in keras layer has 20001 parameters per unit.

(Refer Slide Time: 10:31)


basic-text-classification-with-tfhub.ipynb
File Edit View Insert Runtime Tools Help
CODE TEXT COPY TO DRIVE
SHARE
RAM Disk
EDITING

```
model.summary()
```

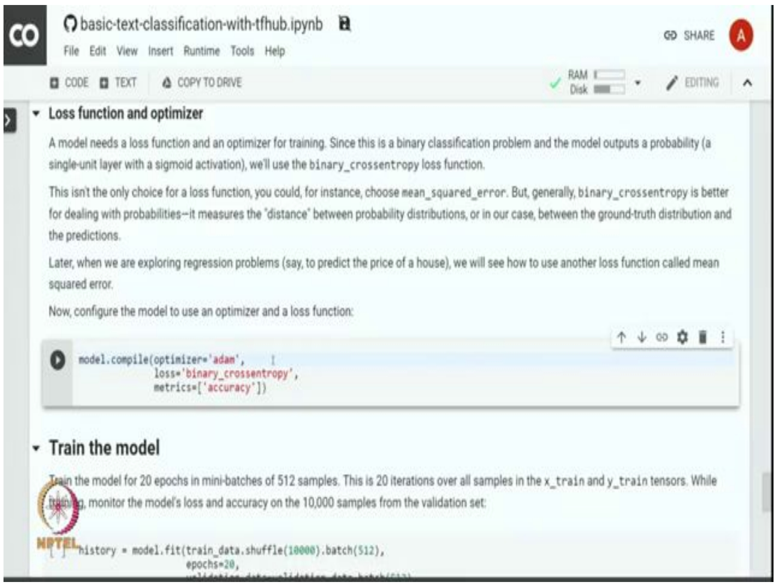
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer (KerasLayer)     (None, 20)                400020
_________________________________________________________________
dense (Dense)                (None, 16)                336
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 17
=================================================================
Total params: 400,373
Trainable params: 400,373
Non-trainable params: 0
_________________________________________________________________
```

The layers are stacked sequentially to build the classifier:

1. The first layer is a TensorFlow Hub layer. This layer uses a pre-trained Saved Model to map a sentence into its embedding vector. The pre-trained text embedding model that we are using (google/tf2-preview/gnews-swivel-20dim/1) splits the sentence into tokens, embeds each token and then combines the embedding. The resulting dimensions are: `(num_examples, embedding_dimension)`.
2. This fixed-length output vector is piped through a fully-connected (`Dense`) layer with 16 hidden units.
3. The last layer is densely connected with a single output node. Using the `sigmoid` activation function, this value is a float between 0 and 1, representing a probability, or confidence level.

Out of this 20001 parameters there is a parameter each for a word in the vocabulary and an additional parameter is used for out of the vocabulary words. So, that makes it about 400 k parameters in the keras layer. So, we have 400373 total parameters to train.

(Refer Slide Time: 11:04)


basic-text-classification-with-tfhub.ipynb
File Edit View Insert Runtime Tools Help
CODE TEXT COPY TO DRIVE
SHARE
RAM Disk
EDITING

Loss function and optimizer

A model needs a loss function and an optimizer for training. Since this is a binary classification problem and the model outputs a probability (a single-unit layer with a sigmoid activation), we'll use the `binary_crossentropy` loss function.

This isn't the only choice for a loss function, you could, for instance, choose `mean_squared_error`. But, generally, `binary_crossentropy` is better for dealing with probabilities—it measures the "distance" between probability distributions, or in our case, between the ground-truth distribution and the predictions.

Later, when we are exploring regression problems (say, to predict the price of a house), we will see how to use another loss function called mean squared error.

Now, configure the model to use an optimizer and a loss function:

```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

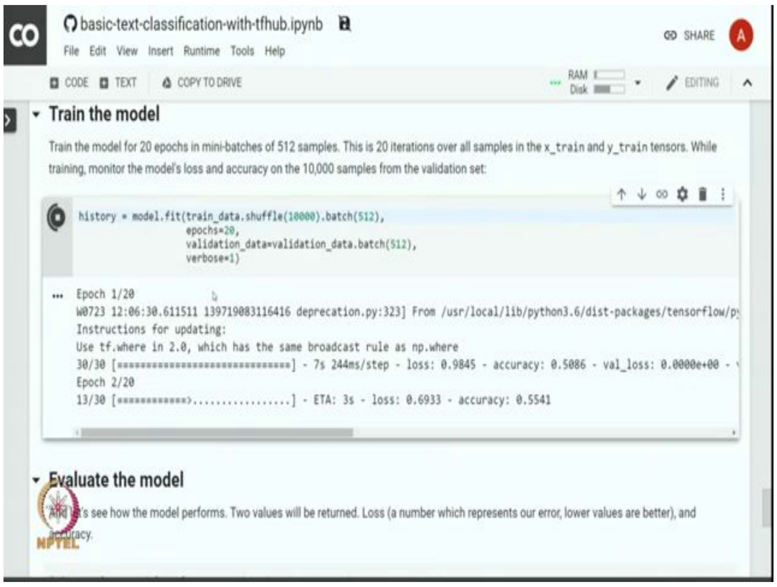
Train the model

Train the model for 20 epochs in mini-batches of 512 samples. This is 20 iterations over all samples in the `x_train` and `y_train` tensors. While training, monitor the model's loss and accuracy on the 10,000 samples from the validation set:

```
history = model.fit(train_data.shuffle(10000).batch(512),
                    epochs=20,
```

Now that the model is defined, let us compile the model. We use adam as an optimizer, we use binary cross entropy loss because we have a binary classification problem to solve here and we track accuracy as a metric.

(Refer Slide Time: 11:13)


basic-text-classification-with-tfhub.ipynb
File Edit View Insert Runtime Tools Help
SHARE
CODE TEXT COPY TO DRIVE
RAM
Disk
EDITING

Train the model

Train the model for 20 epochs in mini-batches of 512 samples. This is 20 iterations over all samples in the x_train and y_train tensors. While training, monitor the model's loss and accuracy on the 10,000 samples from the validation set:

```
history = model.fit(train_data.shuffle(10000).batch(512),
                    epochs=20,
                    validation_data=validation_data.batch(512),
                    verbose=1)
```

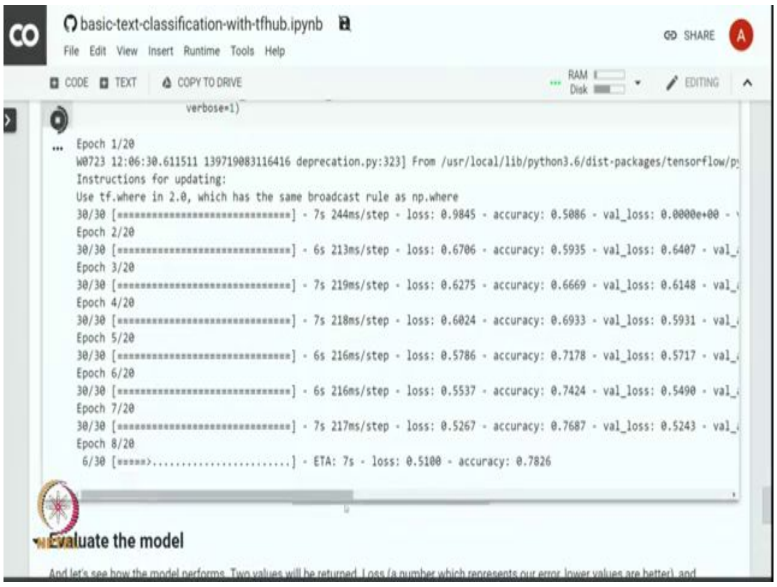
```
Epoch 1/20
W0723 12:06:30.611511 139719083116416 deprecation.py:323] From /usr/local/lib/python3.6/dist-packages/tensorflow/p
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
30/30 [==============================] - 7s 244ms/step - loss: 0.9845 - accuracy: 0.5086 - val_loss: 0.0000e+00 -
Epoch 2/20
13/30 [============>.................] - ETA: 3s - loss: 0.6933 - accuracy: 0.5541
```

Evaluate the model

And let's see how the model performs. Two values will be returned. Loss (a number which represents our error, lower values are better), and accuracy.

We will batch the training data into a mini batches of size 512 samples and we run the training loop for 20 epochs. We store the output of the training loop in the history variable.

(Refer Slide Time: 11:37)


basic-text-classification-with-tfhub.ipynb
File Edit View Insert Runtime Tools Help
SHARE
CODE TEXT COPY TO DRIVE
RAM
Disk
EDITING

```
                    verbose=1)
```

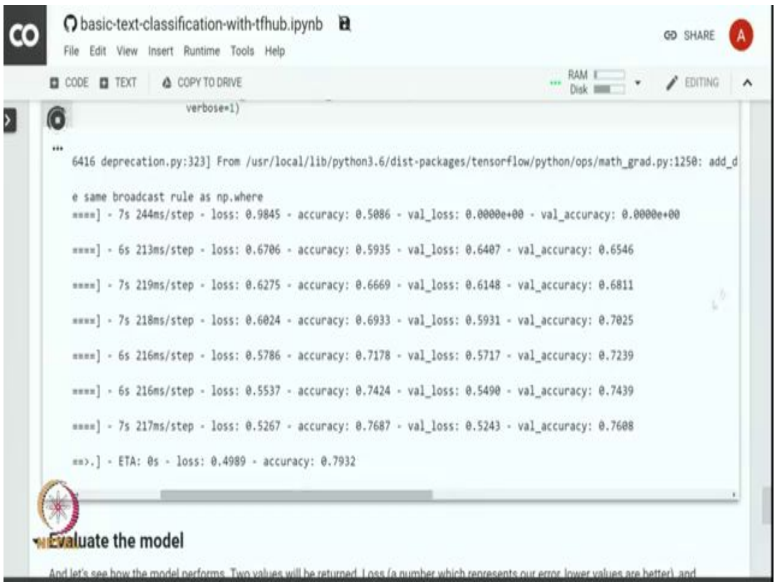
```
Epoch 1/20
W0723 12:06:30.611511 139719083116416 deprecation.py:323] From /usr/local/lib/python3.6/dist-packages/tensorflow/p
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
30/30 [==============================] - 7s 244ms/step - loss: 0.9845 - accuracy: 0.5086 - val_loss: 0.0000e+00 -
Epoch 2/20
30/30 [==============================] - 6s 213ms/step - loss: 0.6706 - accuracy: 0.5935 - val_loss: 0.6407 - val_
Epoch 3/20
30/30 [==============================] - 7s 219ms/step - loss: 0.6275 - accuracy: 0.6669 - val_loss: 0.6148 - val_
Epoch 4/20
30/30 [==============================] - 7s 218ms/step - loss: 0.6024 - accuracy: 0.6933 - val_loss: 0.5931 - val_
Epoch 5/20
30/30 [==============================] - 6s 216ms/step - loss: 0.5786 - accuracy: 0.7178 - val_loss: 0.5717 - val_
Epoch 6/20
30/30 [==============================] - 6s 216ms/step - loss: 0.5537 - accuracy: 0.7424 - val_loss: 0.5490 - val_
Epoch 7/20
30/30 [==============================] - 7s 217ms/step - loss: 0.5267 - accuracy: 0.7687 - val_loss: 0.5243 - val_
Epoch 8/20
 6/30 [=====>........................] - ETA: 7s - loss: 0.5100 - accuracy: 0.7826
```

Evaluate the model

And let's see how the model performs. Two values will be returned. Loss (a number which represents our error, lower values are better), and

Since, we stored it in history variable we can use it later to plot learning curves or any such statistics around the training loop. You can see that the training loss is going down after every epoch and the accuracy is going up. Also keep an eye on the validation accuracy and see what is happening to the validation accuracy.

(Refer Slide Time: 12:01)


basic-text-classification-with-tfhub.ipynb
File Edit View Insert Runtime Tools Help
SHARE
CODE TEXT COPY TO DRIVE
RAM Disk
EDITING

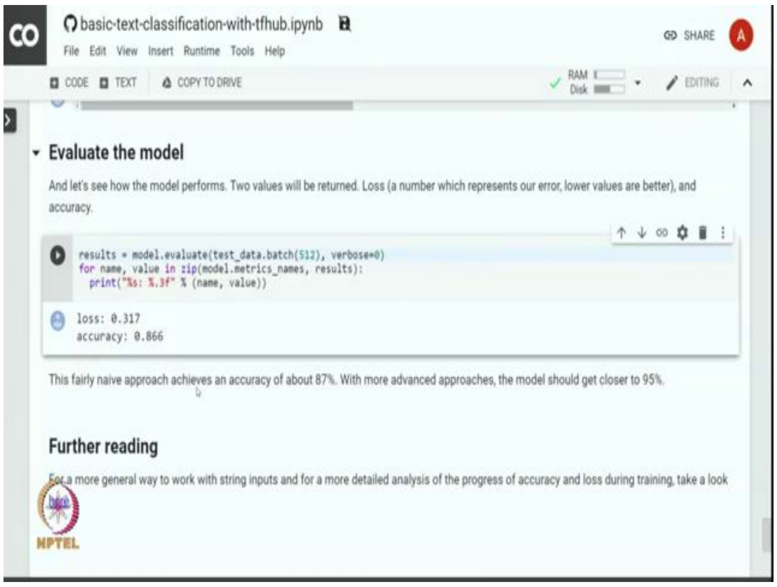
```
                    verbose=1)

6416 deprecation.py:323] From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/math_grad.py:1250: add_d

e same broadcast rule as np.where
====] - 7s 244ms/step - loss: 0.9845 - accuracy: 0.5086 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00

====] - 6s 213ms/step - loss: 0.6706 - accuracy: 0.5935 - val_loss: 0.6407 - val_accuracy: 0.6546

====] - 7s 219ms/step - loss: 0.6275 - accuracy: 0.6669 - val_loss: 0.6148 - val_accuracy: 0.6811

====] - 7s 218ms/step - loss: 0.6024 - accuracy: 0.6933 - val_loss: 0.5931 - val_accuracy: 0.7025

====] - 6s 216ms/step - loss: 0.5786 - accuracy: 0.7178 - val_loss: 0.5717 - val_accuracy: 0.7239

====] - 6s 216ms/step - loss: 0.5537 - accuracy: 0.7424 - val_loss: 0.5490 - val_accuracy: 0.7439

====] - 7s 217ms/step - loss: 0.5267 - accuracy: 0.7687 - val_loss: 0.5243 - val_accuracy: 0.7608

==>.] - ETA: 0s - loss: 0.4989 - accuracy: 0.7932
```

Evaluate the model

And let's see how the model performs. Two values will be returned. Loss (a number which represents our error, lower values are better), and

So, validation accuracy also seems to be going up with each epoch. And training and validation accuracies seem to be quite close. After 30 epoch the training accuracy has crossed 93 percent and validation accuracy has crossed 87 percent.

(Refer Slide Time: 12:19)


basic-text-classification-with-tfhub.ipynb
File Edit View Insert Runtime Tools Help
CODE TEXT COPY TO DRIVE
RAM Disk
EDITING
SHARE

Evaluate the model

And let's see how the model performs. Two values will be returned. Loss (a number which represents our error, lower values are better), and accuracy.

```
results = model.evaluate(test_data.batch(512), verbose=0)
for name, value in zip(model.metrics_names, results):
  print("%s: %.3f" % (name, value))
```

```
loss: 0.317
accuracy: 0.866
```

This fairly naive approach achieves an accuracy of about 87%. With more advanced approaches, the model should get closer to 95%.

Further reading

For a more general way to work with string inputs and for a more detailed analysis of the progress of accuracy and loss during training, take a look here.

Let us evaluate the model performance on an unseen data set. We copy the result of the evaluation in the results variable. So, here on the test data we get an accuracy above 86 percent it is very close to 87 percent. So, this fairly naive approach achieves an accuracy of about 87 percent.

In this module, we built a text classifier using tensorflow API. We used transfer learning based on models in tf hub. Hope you enjoyed learning these concepts.