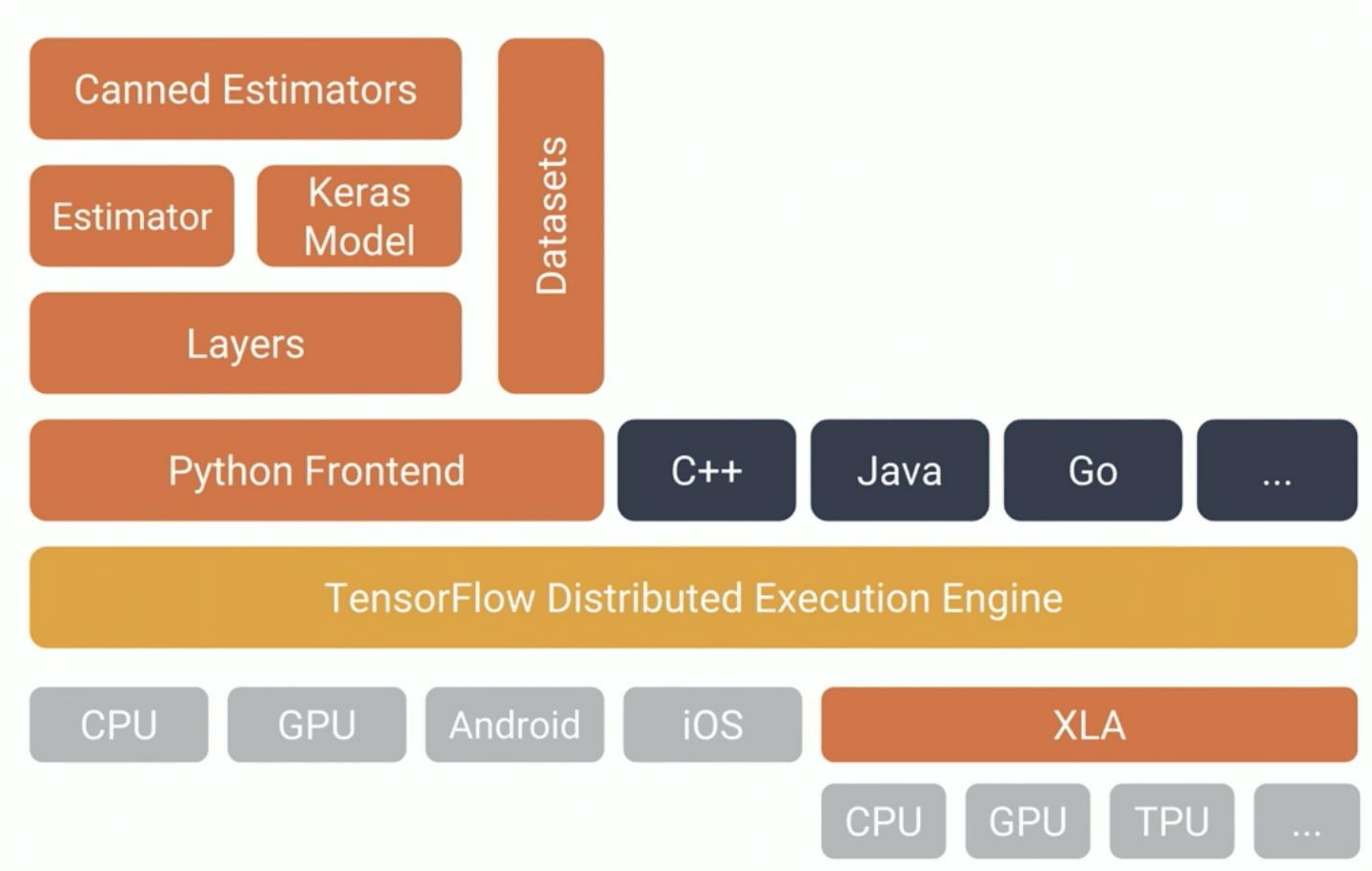
Introduction to TensorFlow

Lecture 17 for 14-763/18-763

Guannan Qu

Nov 4, 2024

TensorFlow Architecture



	Data	**ML Models**
PyTorch	torch.tensor Dataset/DataLoader	nn.Module for building NN models backward() for differentiation
TensorFlow™	**Tensor/Dataset** tf.Tensor tf.data API	**tf.keras High-Level API** keras.Sequential(), keras.Model for building models keras.Model.fit() for training **TensorFlow Core** tf.Module for building models tf.GradientTape() for automatic differentiation

TensorFlow Architecture

Additional tools

TensorBoard
Tool for Data visualization and tuning

tf.distribute
Allows training across multiple GPUs/CPUs on multiple machines

Deployment
TFX for production-level ML
TensorFlow.js for JavaScript Deployment
TensorFlowLite for Mobile/Edge

Basics

Tensor/Dataset

tf.data API

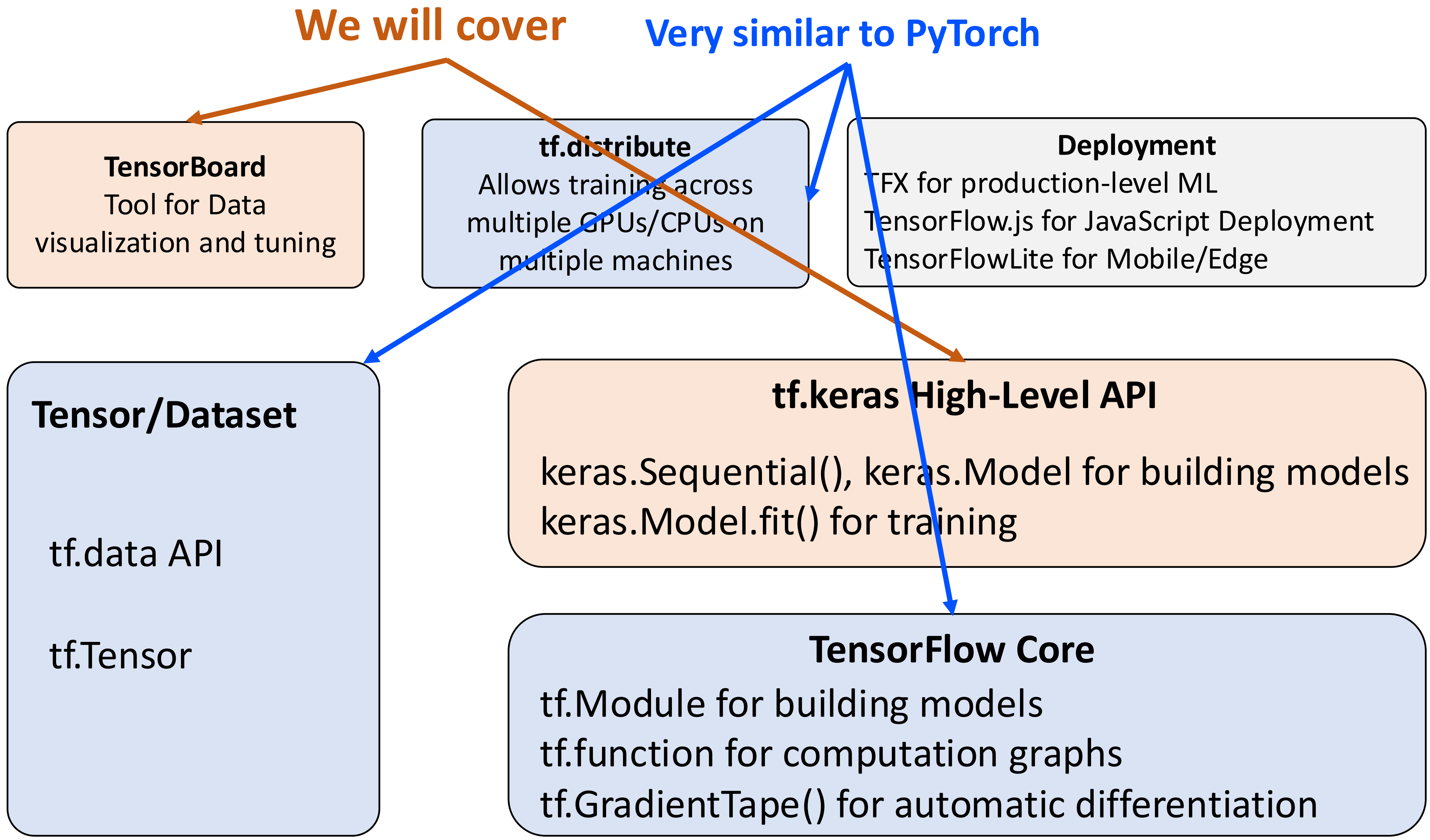
tf.Tensor

tf.keras High-Level API

keras.Sequential(), keras.Model for building models
keras.Model.fit() for training

TensorFlow Core

tf.Module for building models
tf.GradientTape() for automatic differentiation


We will cover
Very similar to PyTorch
TensorBoard
Tool for Data visualization and tuning
tf.distribute
Allows training across multiple GPUs/CPUs on multiple machines
Deployment
TFX for production-level ML
TensorFlow.js for JavaScript Deployment
TensorFlowLite for Mobile/Edge
Tensor/Dataset
tf.data API
tf.Tensor
tf.keras High-Level API
keras.Sequential(), keras.Model for building models
keras.Model.fit() for training
TensorFlow Core
tf.Module for building models
tf.function for computation graphs
tf.GradientTape() for automatic differentiation

TensorBoard
Tool for Data visualization and tuning

tf.distribute
Allows training across multiple GPUs/CPUs on multiple machines

Deployment
TFX for production-level ML
TensorFlow.js for JavaScript Deployment
TensorFlowLite for Mobile/Edge

Tensor/Dataset

tf.data API

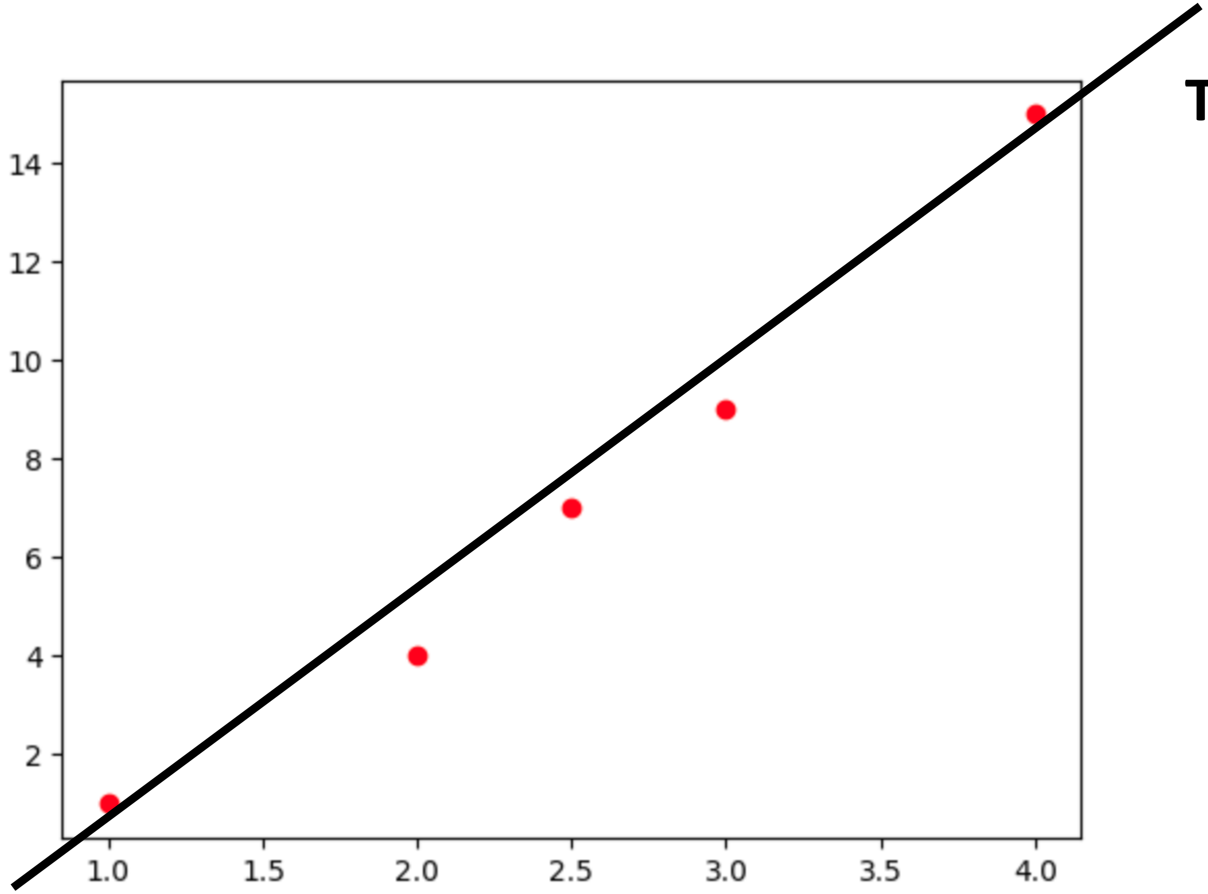
tf.Tensor

tf.keras High-Level API

keras.Sequential(), keras.Model for building models
keras.Model.fit() for training

TensorFlow Core
tf.Module for building models
tf.function for computation graphs
tf.GradientTape() for automatic differentiation

Keras: Linear Regression



Trained (or fitted) linear model

Keras: Linear Regression

How to create a model with tf.keras?

```
model = keras.Sequential()
model.add(keras.layers.Dense(1))
```

This layer is a simple linear function with **output dimension 1**

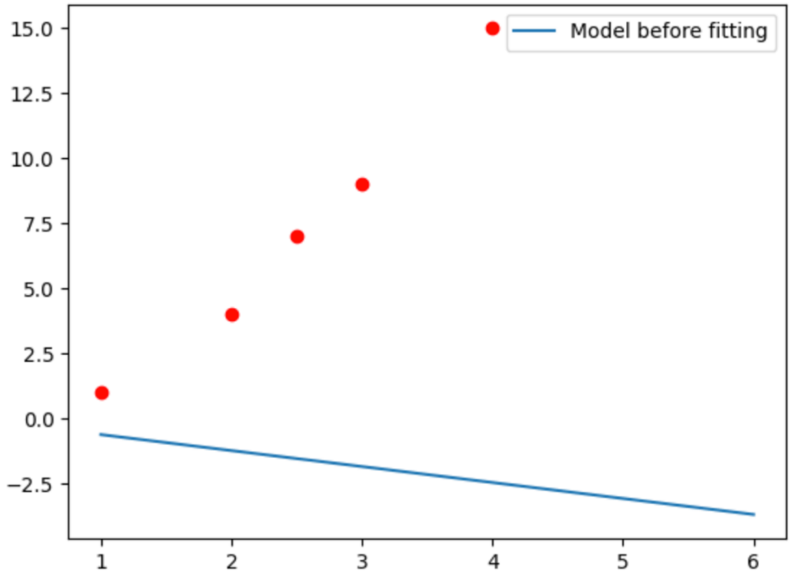
No need to specify input dimension

Input/Feature: x → $y = ax + b$ → Output/Target y

Keras: Linear Regression

How does the model look like? (Note: this model is untrained yet)

```
x_mesh = np.linspace(1,6,100) # generate 100 input values between 1 and 6
x_mesh = tf.constant( x_mesh[:,np.newaxis])
y_pred_mesh = model(x_mesh) # get the output of our model
plt.plot(x, y, 'ro')
plt.plot(x_mesh, y_pred_mesh , label = "Model before fitting")
plt.legend()
```



Keras: Linear Regression

How does the model look like? (Note: this model is untrained yet)

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (100, 1)                  2

=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

Keras: Linear Regression

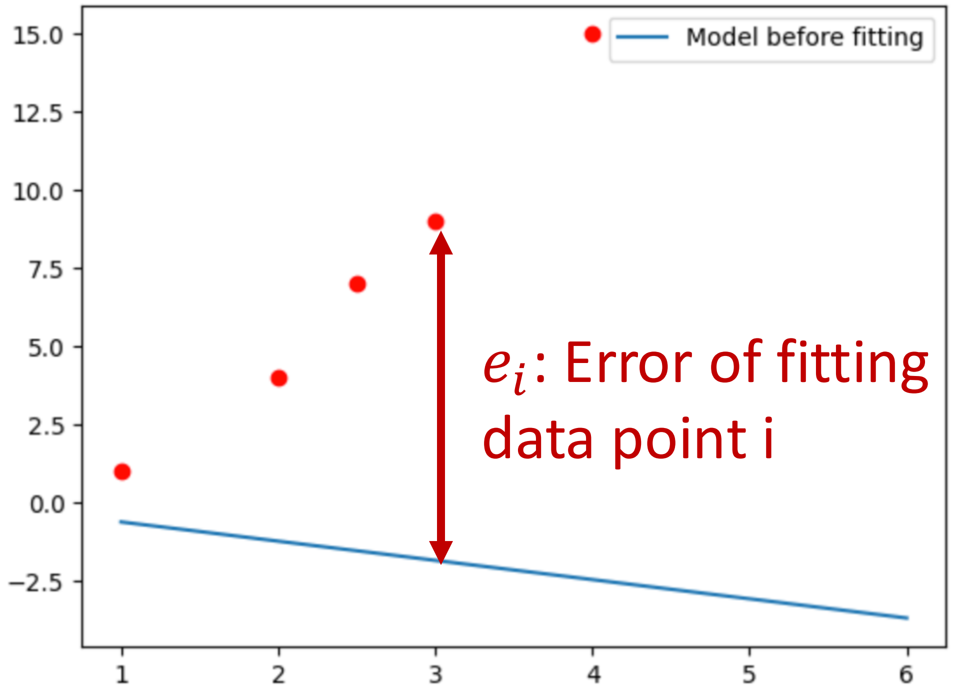
How to train the model?

```
model.compile(optimizer = keras.optimizers.SGD(), loss = keras.losses.MeanSquaredError())
model.fit(x,y,epochs=30)
```

Measures how well the model fits the data

MeanSquaredError =

$$\frac{1}{\#number\ of\ samples}\sum_i (e_i)^2$$



Keras: Linear Regression

How to train the model?

```
model.compile(optimizer = keras.optimizers.SGD(), loss = keras.losses.MeanSquaredError())
model.fit(x,y,epochs=30)
```

Choosing the algorithm to find the model weights that minimize the loss.
SGD, Adam are popular choices, which is similar to PyTorch

Keras: Linear Regression

How to train the model?

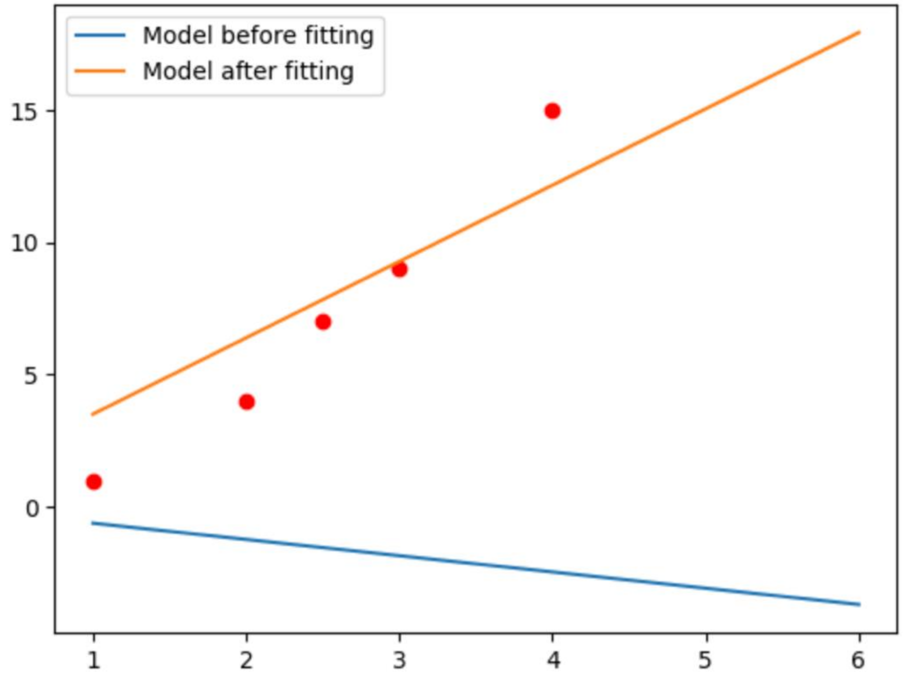
```
model.compile(optimizer = keras.optimizers.SGD(), loss = keras.losses.MeanSquaredError())
model.fit(x,y,epochs=30)
```

Epochs means how many iterations we go through the data (similar to pytorch)

Keras: Linear Regression

How does the trained model look like?

```
y_pred_mesh_afterfitting = model(x_mesh)
plt.plot(x, y, 'ro')
plt.plot(x_mesh, y_pred_mesh , label = "Model before fitting")
plt.plot(x_mesh, y_pred_mesh_afterfitting , label = "Model after fitting")
plt.legend()
```



How to build neural networks?

This is how we build a linear regression model

```
model = keras.Sequential()
model.add(keras.layers.Dense(1))
```

For each hidden layer, specify the width and the activation

To build a neural network, just add more layers

```
model = keras.Sequential()

model.add(keras.layers.Dense(20,activation='relu'))
model.add(keras.layers.Dense(20,activation='relu'))
model.add(keras.layers.Dense(20,activation='relu'))

model.add(keras.layers.Dense(1))
```

Add layers one-by-one

The output dimension

Keras: Simple Neural Network

Equivalent way to create the same neural network

```
model2 = keras.Sequential(layers=[keras.layers.Dense(20,activation='relu'),
                                  keras.layers.Dense(20,activation='relu'),
                                  keras.layers.Dense(20,activation='relu'),
                                  keras.layers.Dense(1)])
```

Keras: Simple Neural Network

Another equivalent way to create the same neural network

```
input = keras.Input(shape = (1))

intermediate_1 = keras.layers.Dense(20,activation='relu')(input)
intermediate_2 = keras.layers.Dense(20,activation='relu')(intermediate_1)
intermediate_3 = keras.layers.Dense(20,activation='relu')(intermediate_2)

output = keras.layers.Dense(1)(intermediate_3)

model3 = keras.Model(inputs = input,outputs = output)
```

Neural Network Classification

I have received many questions regarding
- what should be the input to the cross entropy loss
- what should be the final layer (linear or softmax) of NN.

This can be confusion as PyTorcha and tf.keras has different conventions

I would like to clarify this by first presenting the “mathematical” way to compute cross entropy (which is unambiguous), and then present what is the convention for each platform (pytorch/tensorflow).

Neural Network Classification

Consider a classification problem with C classes $1,2,\ldots C$

Input $\xrightarrow{\text{linear/relu layers}}$ $Score_1$, $Score_2$, …, $Score_C$ $\xrightarrow{\text{SoftMax}}$ $Prob_1$, $Prob_2$, …, $Prob_C$ $\xrightarrow{\text{CrossEntropy}}$ Loss

$Score_i$ is unnormalized (can take values from $-\infty$ to $+\infty$)

The larger $Score_i$, the higher the odds of class i

Softmax Function

$$Prob_i = \frac{\exp(Score_i)}{\exp(Score_1) + \exp(Score_2) + \cdots + \exp(Score_C)}$$

$Prob_i$ is “normalized”, i.e. it must lie between 0,1

Neural Network Classification

Consider a classification problem with C classes $1,2, \dots C$

$$\text{Input} \xrightarrow{\text{linear/relu layers}} \begin{matrix} Score_1 \\ Score_2 \\ \dots \\ Score_C \end{matrix} \xrightarrow{\text{SoftMax}} \begin{matrix} Prob_1 \\ Prob_2 \\ \dots \\ Prob_C \end{matrix} \xrightarrow{\text{CrossEntropy}} \text{Loss}$$

Suppose the true label is $y \in \{1,2, \dots, C\}$
CrossEntropy for this data point is

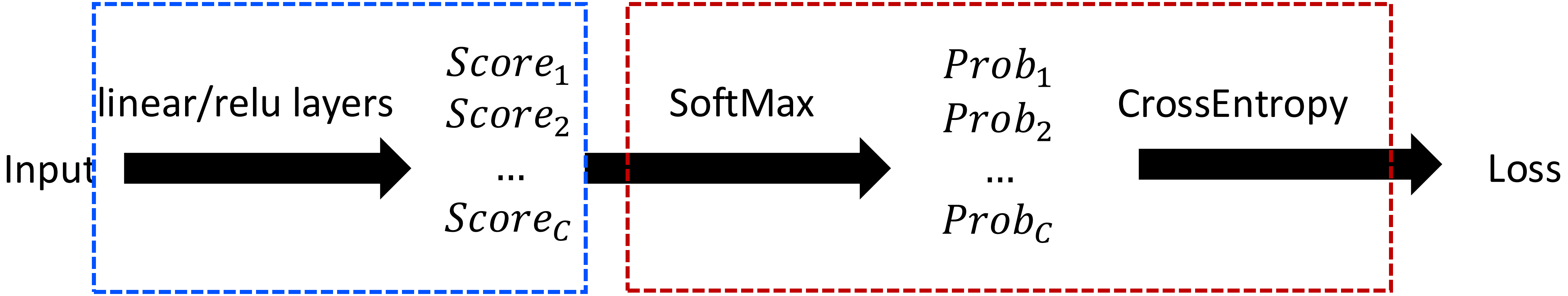
$$-\log Prob_y \;=\begin{cases} 0 & \text{If } Prob_y = 1 \\ +\infty & \text{If } Prob_y = 0 \end{cases}$$

where $Prob_y$ is the probability of the correct class

Minimizing cross entropy encourages predicting the true label with larger prob.

Neural Network Classification

PyTorch Convention



So in PyTorch, neural networks typically don't include softmax as final layer. Typically, linear is the final layer.

PyTorch nn.CrossEntropyLoss()
Includes BOTH Softmax AND CrossEntropy

Neural Network Classification

TensorFlow Keras Convention?

$$\text{Input} \xrightarrow{\text{linear/relu layers}} \begin{matrix} Score_1 \\ Score_2 \\ \ldots \\ Score_C \end{matrix} \xrightarrow{\text{SoftMax}} \begin{matrix} Prob_1 \\ Prob_2 \\ \ldots \\ Prob_C \end{matrix} \xrightarrow{\text{CrossEntropy}} \text{Loss}$$

Let’s use NSL-KDD as example

Neural Network Classification

Which means your NN should NOT include softmax as final layer

If From_logits = true, the SparseCategoricalCrossentorpy() will INCLUDE SOFTMAX

Input $\xrightarrow{\text{linear/relu layers}}$ $Score_1$, $Score_2$, …, $Score_C$ $\xrightarrow{\text{SoftMax}}$ $Prob_1$, $Prob_2$, …, $Prob_C$ $\xrightarrow{\text{CrossEntropy}}$ Loss

```
model_multiclass = keras.Sequential( [keras.layers.Dense(10,activation='relu'),
                                      keras.layers.Dense(10,activation='relu'),
                                      keras.layers.Dense(10,activation='relu'),
                                      keras.layers.Dense(10,activation='relu') ,
                                      keras.layers.Dense(2)] )

model_multiclass.compile(optimizer = 'sgd',
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
```

NN output are two scores, one for normal and one for attack

From_logits = true assumes NN output is score

Neural Network Classification

Which means your NN SHOULD INCLUDE softmax

If From_logits = false, the SparseCategoricalCrossentorpy() will NOT INCLUDE SOFTMAX

Input → linear/relu layers → $Score_1$, $Score_2$, …, $Score_C$ → SoftMax → $Prob_1$, $Prob_2$, …, $Prob_C$ → CrossEntropy → Loss

```
model2 = keras.Sequential( [keras.layers.Dense(10,activation='relu'),
                            keras.layers.Dense(10,activation='relu'),
                            keras.layers.Dense(10,activation='relu'),
                            keras.layers.Dense(10,activation='relu') ,
                            keras.layers.Dense(2,activation='softmax')] )
```

NN output are two probabilities, one for normal and one for attack

From_logits = false assumes NN output is probability

```
model2.compile(optimizer = 'sgd',loss=keras.losses.SparseCategoricalCrossentropy(from_logits=False))
```

Neural Network Classification

Here we use the SparseCategoricalCrossentropy as the loss

- **"Sparse" refers to the fact the true label is integer values**

Can also use CategoricalCrossentropy

- **In this case, the true label should be onehot encoded**

```
model_multiclass.compile(optimizer = 'sgd',
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
```

Summary So Far

- Should set from_logits = True if NN output is score (last layer linear)
- Should set from_logits = False if NN output is probability (last layer softmax)
- NN output dimension should be the same as the number of classes

Converting DF to tf.Tensor

```
to_array = udf(lambda v: v.toArray().tolist(), ArrayType(FloatType()))

nslkdd_df_train = nslkdd_df
nslkdd_df_validate,nslkdd_df_test = nslkdd_df_test.randomSplit([0.5,0.5])

nslkdd_df_train_pandas = nslkdd_df_train.withColumn('features', to_array('features')).toPandas()
nslkdd_df_validate_pandas = nslkdd_df_validate.withColumn('features', to_array('features')).toPandas()
nslkdd_df_test_pandas = nslkdd_df_test.withColumn('features', to_array('features')).toPandas()
```

```
x_train = tf.constant(np.array(nslkdd_df_train_pandas['features'].values.tolist()))
y_train = tf.constant(np.array(nslkdd_df_train_pandas['outcome'].values.tolist()))

x_validate = tf.constant(np.array(nslkdd_df_validate_pandas['features'].values.tolist()))
y_validate = tf.constant(np.array(nslkdd_df_validate_pandas['outcome'].values.tolist()))


x_test = tf.constant(np.array(nslkdd_df_test_pandas['features'].values.tolist()))
y_test = tf.constant(np.array(nslkdd_df_test_pandas['outcome'].values.tolist()))
```

Training for NSL-KDD

```
model = keras.Sequential( [keras.layers.Dense(10,activation='relu'),
                           keras.layers.Dense(10,activation='relu'),
                           keras.layers.Dense(10,activation='relu'),
                           keras.layers.Dense(10,activation='relu') ,
                           keras.layers.Dense(2)] )
```

```
model.compile(optimizer = keras.optimizers.SGD(learning_rate=0.02),
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=[keras.metrics.SparseCategoricalAccuracy()])

model.fit(x_train,y_train, epochs = 5,batch_size = 64, validation_data=(x_validate,y_validate),verbose = 2)
```

Fit the keras Model

```
Epoch 1/5
1969/1969 - 1s - loss: 0.1268 - sparse_categorical_accuracy: 0.9649 - val_loss: 0.7386 - val_sparse_categorical_accuracy: 0.7687 - 699ms/epoch - 355us/step
Epoch 2/5
1969/1969 - 1s - loss: 0.0681 - sparse_categorical_accuracy: 0.9810 - val_loss: 0.8911 - val_sparse_categorical_accuracy: 0.7553 - 502ms/epoch - 255us/step
Epoch 3/5
1969/1969 - 1s - loss: 0.0605 - sparse_categorical_accuracy: 0.9824 - val_loss: 1.0256 - val_sparse_categorical_accuracy: 0.7542 - 506ms/epoch - 257us/step
Epoch 4/5
1969/1969 - 1s - loss: 0.0517 - sparse_categorical_accuracy: 0.9833 - val_loss: 1.3345 - val_sparse_categorical_accuracy: 0.7714 - 508ms/epoch - 258us/step
Epoch 5/5
1969/1969 - 0s - loss: 0.0389 - sparse_categorical_accuracy: 0.9868 - val_loss: 1.4130 - val_sparse_categorical_accuracy: 0.7617 - 497ms/epoch - 253us/step
```

Evaluate it on the test data

```
model.evaluate(x_test,y_test, verbose = 2)
```

✓ 0.1s

```
179/179 - 0s - loss: 1.3673 - sparse_categorical_accuracy: 0.7626 - 93ms/epoch - 518us/step
```

Up Next

TensorBoard
Tool for Data visualization and tuning

tf.distribute
Allows training across multiple GPUs/CPUs on multiple machines

Deployment
TFX for production-level ML
TensorFlow.js for JavaScript Deployment
TensorFlowLite for Mobile/Edge

Tensor/Dataset

tf.data API

tf.Tensor

tf.keras High-Level API

keras.Sequential(), keras.Model for building models
keras.Model.fit() for training

TensorFlow Core

tf.Module for building models
tf.function for computation graphs
tf.GradientTape() for automatic differentiation

TensorBoard

- TensorBoard is an interactive interface that allows you to
  - Track “scalar” metrics, like train/validation loss, auc, accuracy, across different epochs
  - Visualize the structure of neural network
  - Hyper-Parameter Tuning

Let’s try TensorBoard

```
import datetime

model = keras.Sequential( [keras.layers.Dense(10,activation='relu'),
    keras.layers.Dense(10,activation='relu'),
    keras.layers.Dense(10,activation='relu'),
    keras.layers.Dense(10,activation='relu') ,
    keras.layers.Dense(2)] )
model.compile(loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=[keras.metrics.SparseCategoricalAccuracy(name='Accuracy')])
log_dir = "logs14763/myfirstlog/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
model.fit(x=x_train, y=y_train,
            epochs=20, verbose = 2,
            validation_data=(x_validate, y_validate),
            callbacks=[tensorboard_callback])
```

Create a callback object and specify log directory

Pass the callback to Model.fit(), which will run the callback to write all necessary info to the log directory

How to launch TensorBoard?

Launch TensorBoard within notebook

```
%load_ext tensorboard
%tensorboard --logdir logs14763/myfirstlog/
```

Launch TensorBoard in terminal

- Use the following command: tensorboard --logdir logs14763/myfirstlog/
- The terminal will then prompt a URL, typically http://localhost:6006
- Use your browser to enter that URL

Hyper Parameter Tuning with TensorBoard

- We have kept using a Neural Network with 3 hidden layers, each with 20 neurons.
- There is no fixed rule in how we should choose these numbers, and let’s tune it!

Hyper Parameter Tuning with TensorBoard

```
from tensorboard.plugins.hparams import api as hp

HP_WIDTH = hp.HParam('NN_width', hp.Discrete([20,30]))
HP_DEPTH = hp.HParam('NN_depth', hp.Discrete([4,6]))

with tf.summary.create_file_writer('logs14763/hparam_tuning').as_default():
  hp.hparams_config(
    hparams=[HP_WIDTH, HP_DEPTH],
    metrics=[hp.Metric('Accuracy')],
  )
```

Create two hyper parameters and pick several values

Configure the hyper-parameter panel - two hyperparameters to tune, with the metric

Create NN model with given depth and width

```python
def train_test_model(hparams,logdir):
    model = keras.Sequential()
    for _ in range(hparams[HP_DEPTH]):
        model.add(keras.layers.Dense(hparams[HP_WIDTH],activation='relu'))
    model.add(keras.layers.Dense(2))
    model.compile(
        optimizer=keras.optimizers.SGD(),
        loss = keras.losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=[keras.metrics.SparseCategoricalAccuracy(name="Accuracy_epochs")])

    history = model.fit(x_train, y_train, epochs=5, verbose = 2,
    callbacks=[tf.keras.callbacks.TensorBoard(log_dir=logdir, histogram_freq=1)],
    validation_data = (x_validate, y_validate))

    accuracy = np.max(history.history["val_Accuracy_epochs"])
    return accuracy
```

Calculate the largest accuracy across different epochs

```python
for hp_width in HP_WIDTH.domain.values:
    for hp_depth in (HP_DEPTH.domain.values):
        hparams = {
            HP_WIDTH: hp_width,
            HP_DEPTH: hp_depth,
        }
        run_name = f"run-WIDTH{int(hparams[HP_WIDTH])}-DEPTH{hparams[HP_DEPTH]}"
        print('--- Starting trial: %s' % run_name)
        print({h.name: hparams[h] for h in hparams})

        run_dir = 'logs14763/hparam_tuning/' + run_name
        accuracy = train_test_model(hparams,run_dir)

        with tf.summary.create_file_writer(run_dir).as_default():
            hp.hparams(hparams)  # record the values used in this trial
            tf.summary.scalar("Accuracy", accuracy, step=1)
```

Go through all combinations of hyper-parameters

Train our model and get Accuracy

Record the hyper-parameter value, Accuracy to the log directory

Hyper Parameter Tuning with TensorBoard