TensorFlow Introduction

Deep Learning

TensorFlow

- TensorFlow is an open source framework for developing deep networks.
- It was originally developed by the Google Brain group starting in 2011 as DistBelief,
- It was further developed into TensorFlow and released as open source software in November of 2015.
- It is written in Python, C++ and cuda.

Keras

- There is more than one API (Application Programming Interface) for TensorFlow.
- We will focus on Keras.
- Keras was originally developed by François Chollet to act as a frontend for multiple frameworks.
- Chollet later joined Google, and Keras became the central API for TensorFlow.

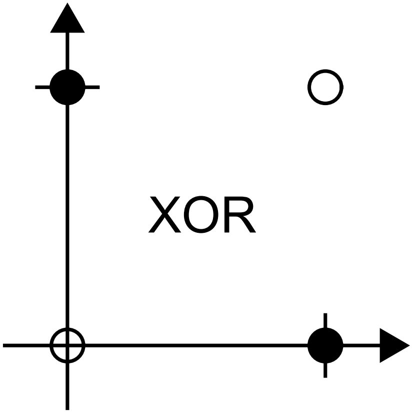
Model training steps

- Load the data.
- Construct the network.
- Train the network.
- Analyze the results.

Network Inputs

- Each element of a network input is called a feature.
- The data set will consist of Q samples of inputs.
- If each input is a vector (tabular) with R features, then the data set is a (Q, R), or (samples, features), NumPy tensor.
- If the input is a time series, the input is a 3D tensor of the form (samples, timesteps, features).
- For images, the 4D input tensor form is (samples, height, width, channels), where channels are usually colors.
- For videos, the 5D input tensor form is (samples, timesteps, height, width, channels).
- For Keras, the network outputs are also NumPy tensors.

XOR test problem

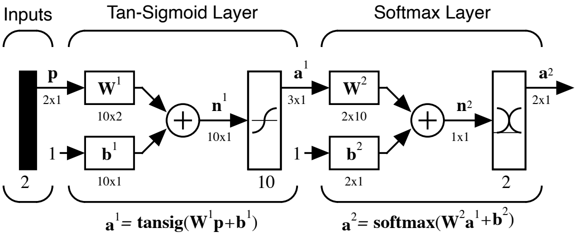


Generate XOR data

```
from tensorflow.keras.utils import
    ↪ to_categorical
p = np.array([[0, 0], [0, 1], [1, 0], [1,
    ↪ 1]])
t = np.array([0, 1, 1, 0])
t = to_categorical(t)
print(t)
```

```
[[1. 0.]
 [0. 1.]
 [0. 1.]
 [1. 0.]]
```

Constructing the model with Sequential



```
from tensorflow.keras import models
from tensorflow.keras import layers
model = models.Sequential()
model.add(layers.Dense(10, activation='
    ↪ tanh', input_shape=(2,)))
model.add(layers.Dense(2, activation='
    ↪ softmax'))
```

Constructing the model with the Functional API

```
p = layers.Input(shape=(2,))
a1 = layers.Dense(10, activation='tanh')(p)
a2 = layers.Dense(2, activation='softmax')(a1)
model = models.Model(inputs=p, outputs=a2)
```

Constructing the model using the model subclass method

```
class Twolayer(models.Model):
    def __init__(self):
        super(Twolayer,self).__init__()
        self.dense1 = layers.Dense(10,
           ↪ activation='tanh')
        self.dense2 = layers.Dense(2,
           ↪ activation='softmax')
    def call(self, inputs):
        x = self.dense1(inputs)
        output = self.dense2(x)
        return output

model = Twolayer()
```

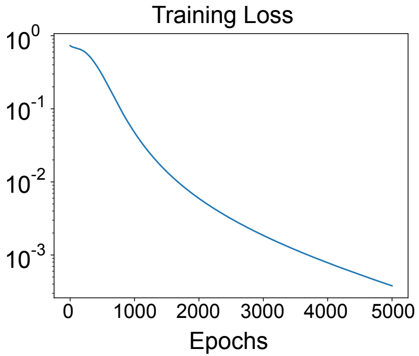
Training the network

```
model.compile(optimizer='adam',loss='
    ↪ categorical_crossentropy')
```

```
history = model.fit(p,t,epochs=5000,
    ↪ batch_size=4)
```

```
import matplotlib.pyplot as plt
history_dict = history.history
loss_values = history_dict['loss']
epochs = history.epoch
plt.semilogy(epochs, loss_values)
plt.title('Training␣Loss')
plt.xlabel('Epochs')
plt.show()
```

Convergence plot



Predict method for inference

```
print(model.predict(p))
```

```
[[9.9955982e-01 4.4020030e-04]
 [5.6742248e-04 9.9943250e-01]
 [4.5553621e-04 9.9954444e-01]
 [9.9940550e-01 5.9446518e-04]]
```

Advanced Data Loading

- Before network training comes Extract, Transform and Load (ETL).
- First, the data are taken from one or multiple files, which may be distributed across multiple machines.
- Next, the data is transformed (normalizing, augmenting by rotating or scaling images, adding noise, etc.)
- Finally, the data is loaded into the training process, often in minibatches.

Data generators

- The first argument to the `fit` method can be a data generator.
- A data generator has many advantages:
    - The data set may be too large to fit into memory.
    - Minibatches can be used.
    - Distribute computation across multiple GPUs.
    - Modify the data during training (shuffle or augment).

TensorFlow Dataset

- `tf.data.Dataset` can create an input pipeline.
- Can be passed to the fit method instead of a data generator.

```
import pandas as pd
sample_df = pd.read_csv('SampleDF.csv')
```

```
P = np.array(sample_df['FVC'])
T = np.array(sample_df['Percent'])
```

```
from tensorflow.data import Dataset
dataset = Dataset.from_tensor_slices((P, T
    ↪ ))
```

Iterate on a Datasest

- The Dataset is an iterable, like a data generator.
- We can access the elements with a for loop

```
for feat, targ in dataset.take(5):
  print ('Features:␣{},␣Target:␣{}'.format
      ↪ (feat, targ))
```

```
Features: 2972, Target: 81.8281938325991
Features: 2253, Target: 59.622102254684
Features: 1648, Target: 68.1160618335125
Features: 969, Target: 49.07571537097999
Features: 2885, Target: 98.66621067031471
```

Batching with a Dataset

```
dataset = dataset.batch(5)
for feat, targ in dataset.take(5):
  print ('Features:␣{},␣Target:␣{}'.format(feat, targ
      ↪ ))
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Features: [2972 2253 1648  969 2885], Target:
    ↪ [81.82819383 59.62210225 68.11606183
    ↪ 49.07571537 98.66621067]
Features: [3045 4791 3171 3350 2833], Target: [
    ↪ 76.91724765 153.14537783  92.15880028
    ↪ 83.59952086  77.21029107]
Features: [4029 3410 3346 4251 1383], Target:
    ↪ [100.26378658  88.15925543  86.50465357
    ↪ 118.74301676  60.20634713]
Features: [3255 2220 1845 2756 1389], Target:
    ↪ [84.27402651 96.92630108 67.90577843
    ↪ 82.55451713 56.68924986]
Features: [2416 2917 3303 2327 4574], Target: [
    ↪ 71.57246119  66.70172871 115.39267747
    ↪ 60.56741281 109.13342241]
```