Neural Networks

Prof. Gheith Abandah

Reference: *Hands-On Machine Learning with Scikit-Learn and TensorFlow* by Aurélien Géron (O'Reilly). Copyright 2017 Aurélien Géron, 978-1-491-96229-9.

Introduction

- YouTube Video: *But what *is* a Neural Network?* from 3Blue1Brown

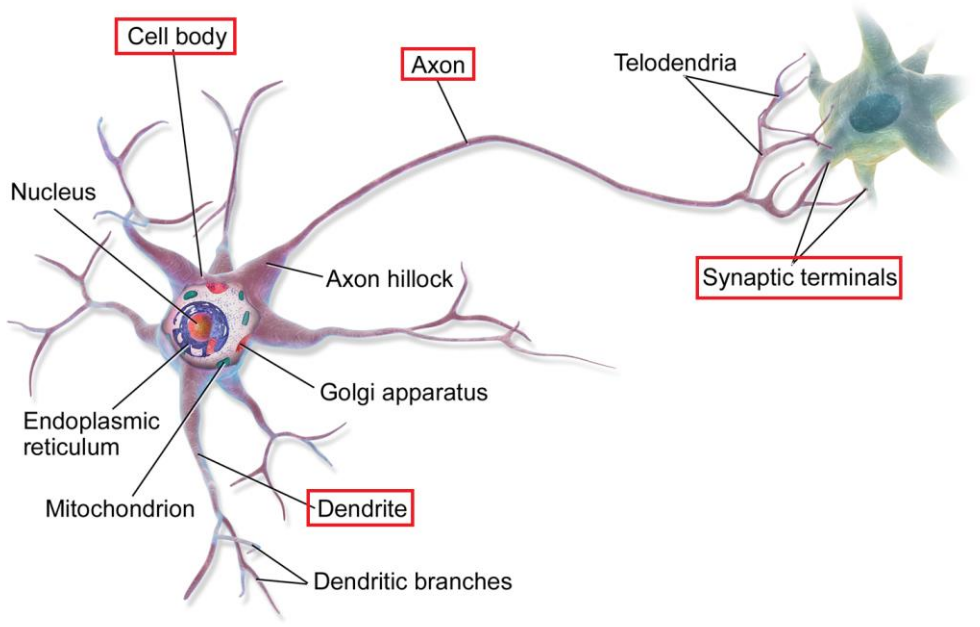
https://youtu.be/aircAruvnKk

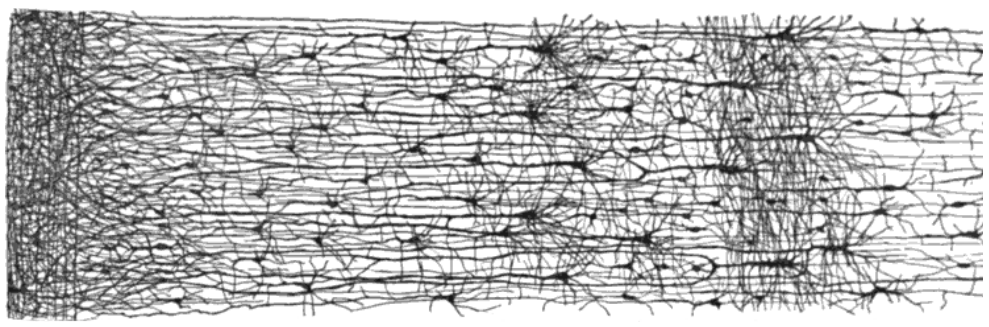
Outline

1. Introduction
2. The perceptron
3. Multi-layer perceptron
4. TensorFlow’s high-level API
5. DNN using plain TensorFlow
6. Fine-tuning neural network hyperparameters
7. Exercises

1. Introduction

- *Artificial neural networks* (ANNs) are inspired by the brain's architecture.
- First suggested in 1943. Is now flourishing due to the availability of:
  - Data
  - Computing power
  - Better algorithms

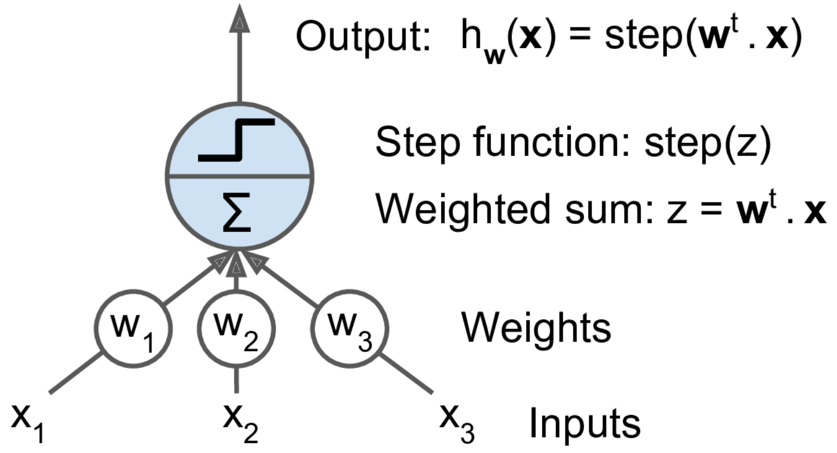




2. The Perceptron

- The *Perceptron* is a simple ANN, invented in 1957 and can perform linear binary classification or regression.

Linear threshold unit (LTU)

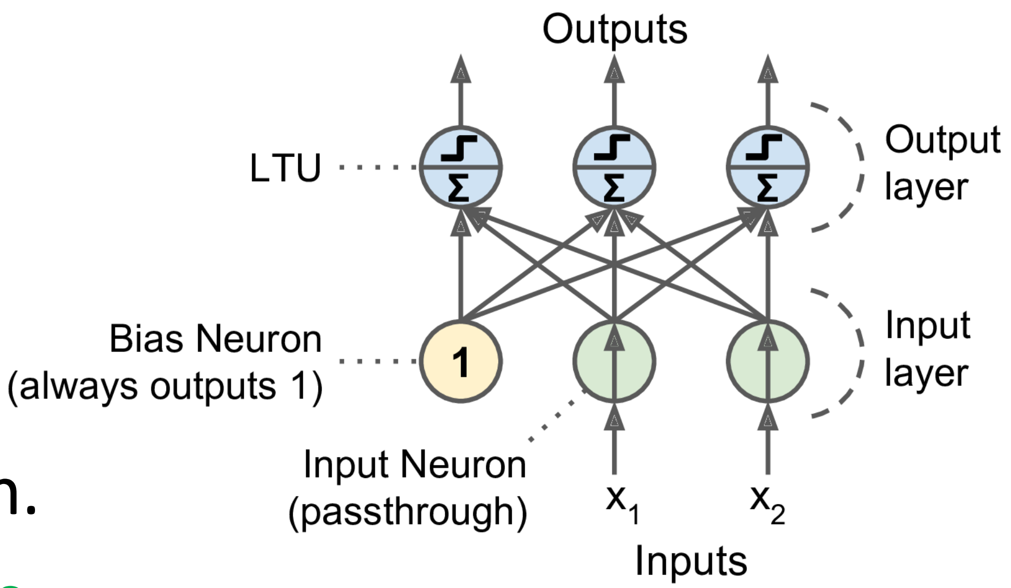

Output: $h_{\mathbf{w}}(\mathbf{x}) = \text{step}(\mathbf{w}^t . \mathbf{x})$
Step function: step(z)
Weighted sum: $z = \mathbf{w}^t . \mathbf{x}$
Σ
w_1
w_2
w_3
Weights
x_1
x_2
x_3
Inputs

- Common step functions:

$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \qquad \text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

2. The Perceptron

- The Perceptron has an *input layer* with *bias* and *output layer*.
- With multiple output nodes, it can perform multiclass classification.
- Hebbian learning "Cells that fire together, wire together."



$$\boxed{w_{i,j}^{(\text{next step})} = w_{i,j} + \eta\left(y_j - \hat{y}_j\right)x_i}$$

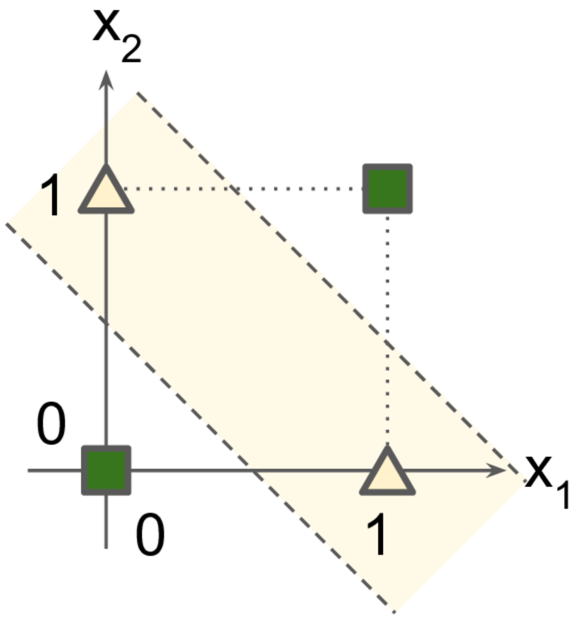
2. The Perceptron

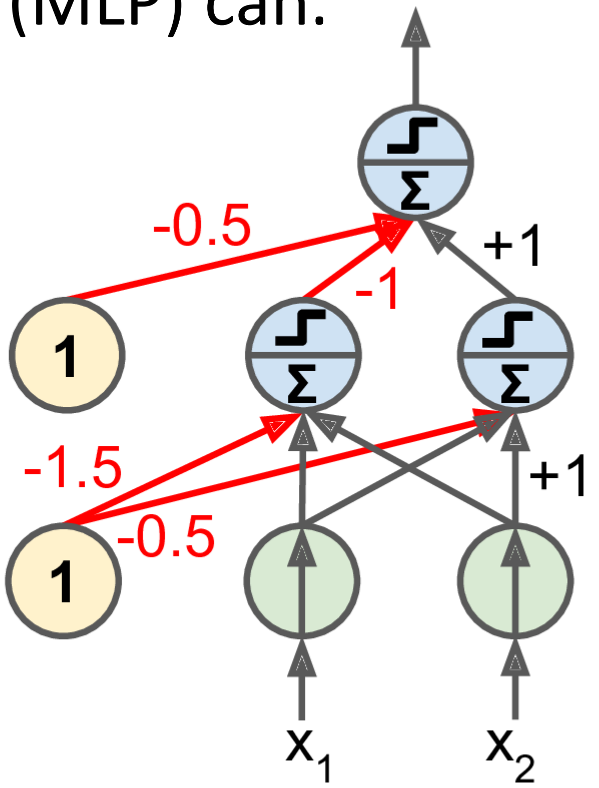
- Scikit-Learn provides a perceptron class.

```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron

iris = load_iris()
X = iris.data[:, (2, 3)]  # petal length, petal width
y = (iris.target == 0).astype(np.int)  # Iris Setosa?
per_clf = Perceptron(random_state=42)
per_clf.fit(X, y)

y_pred = per_clf.predict([[2, 0.5]])
```

2. The Perceptron

- The perceptron cannot solve non-linear problems such as the XOR problem.
- The Multi-Layer Perceptron (MLP) can.



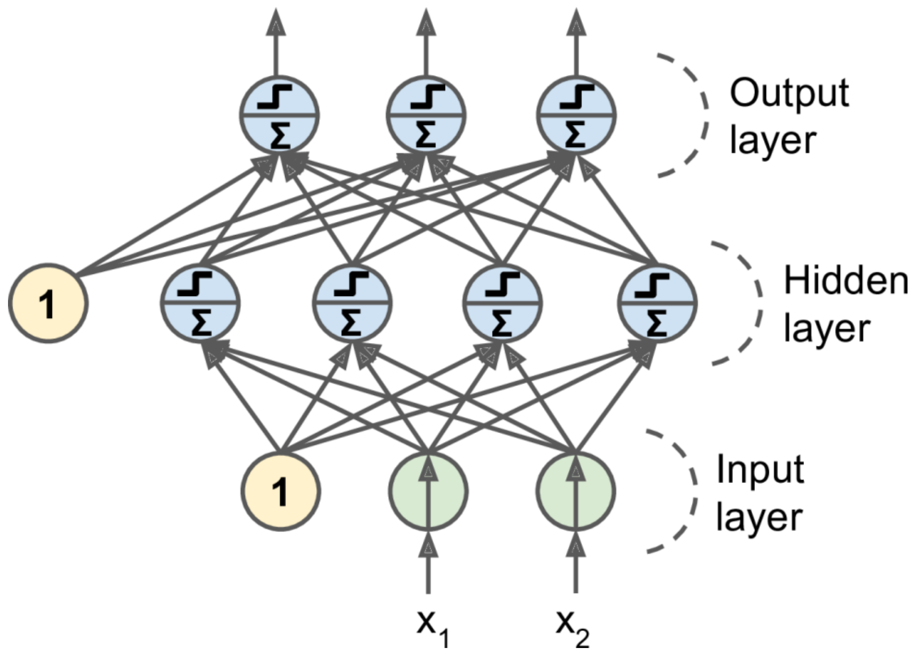


Outline

1. Introduction
2. The perceptron
3. Multi-layer perceptron
4. TensorFlow’s high-level API
5. DNN using plain TensorFlow
6. Fine-tuning neural network hyperparameters
7. Exercises

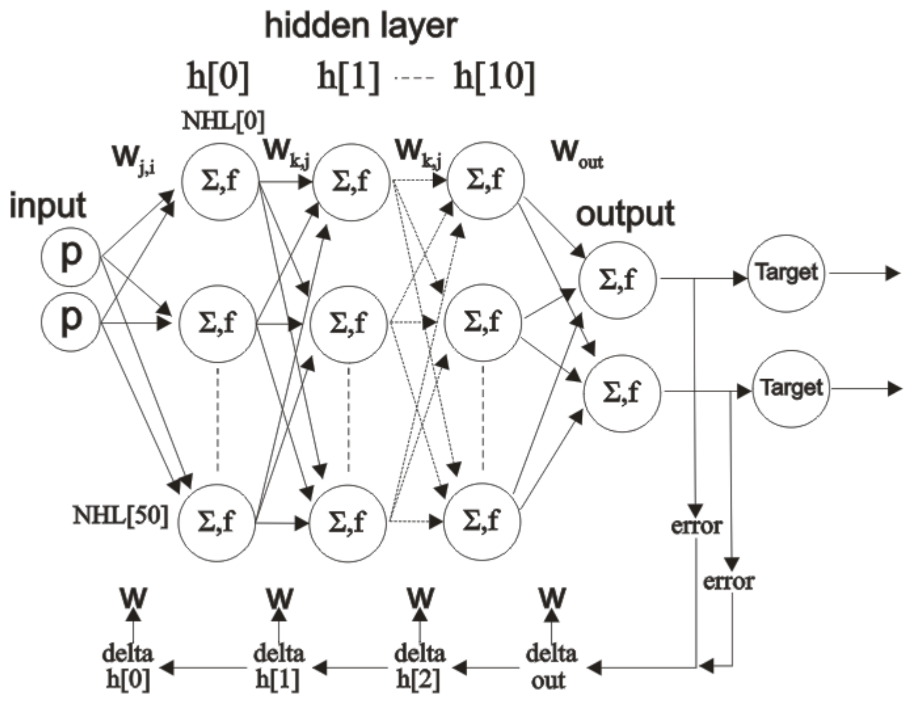
3. Multi-Layer Perceptron (MLP)

- An MLP is composed of a (pass-through) input layer, one or more layers of LTUs, called *hidden layers*, and a final layer of LTUs called the output layer.
- When an ANN has two or more hidden layers, it is called a *deep neural network* (DNN).



3. Multi-Layer Perceptron (MLP)

- Trained using the *backpropagation training algorithm*.
    - For each training instance the algorithm first makes a prediction (*forward pass*), measures the error,
    - then goes through each layer in reverse to measure the error contribution from each connection (*reverse pass*),
    - and finally slightly tweaks the connection weights to reduce the error (*Gradient Descent step*).



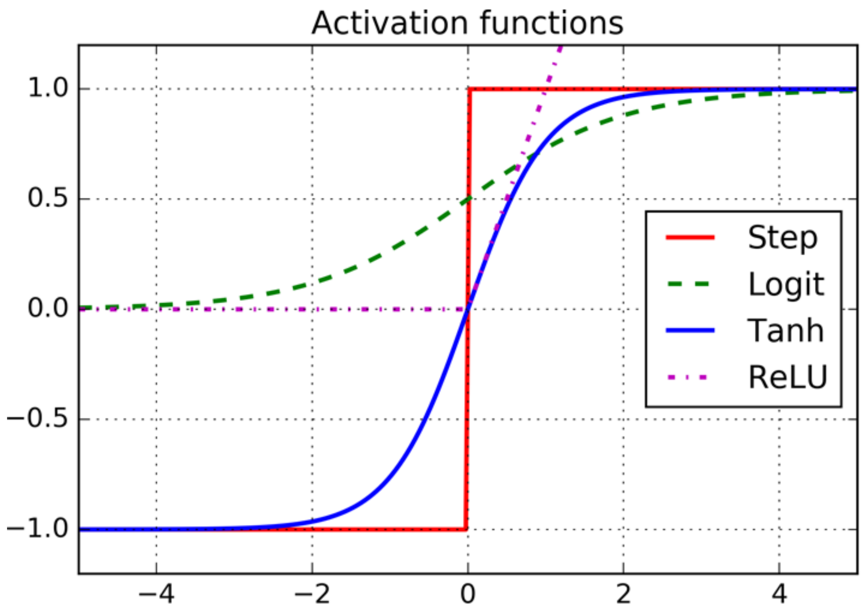
3. Multi-Layer Perceptron (MLP)

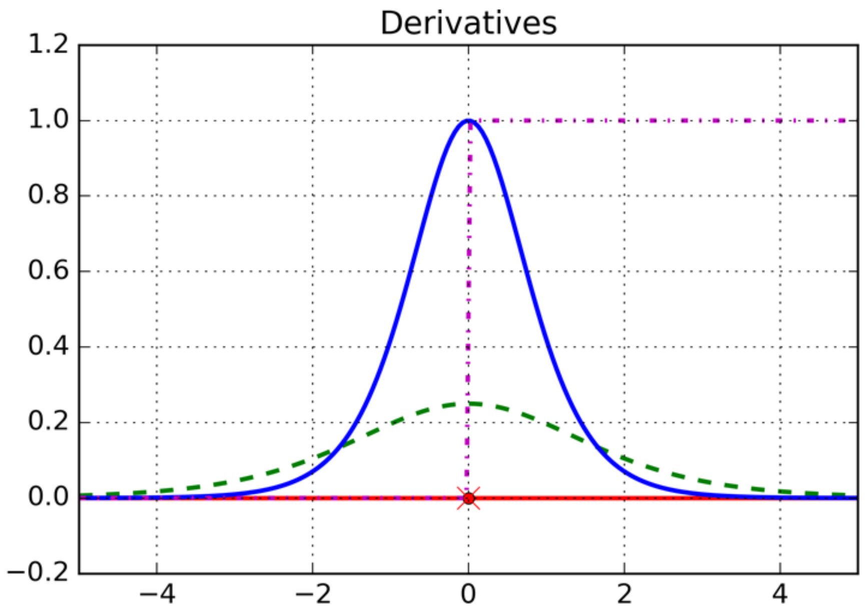
- **Common activation functions**: logistic, hyperbolic tangent, and rectified linear unit.

$$\sigma(z) = 1 \,/\, (1 + \exp(-z))$$

$$\tanh(z) = 2\sigma(2z) - 1$$

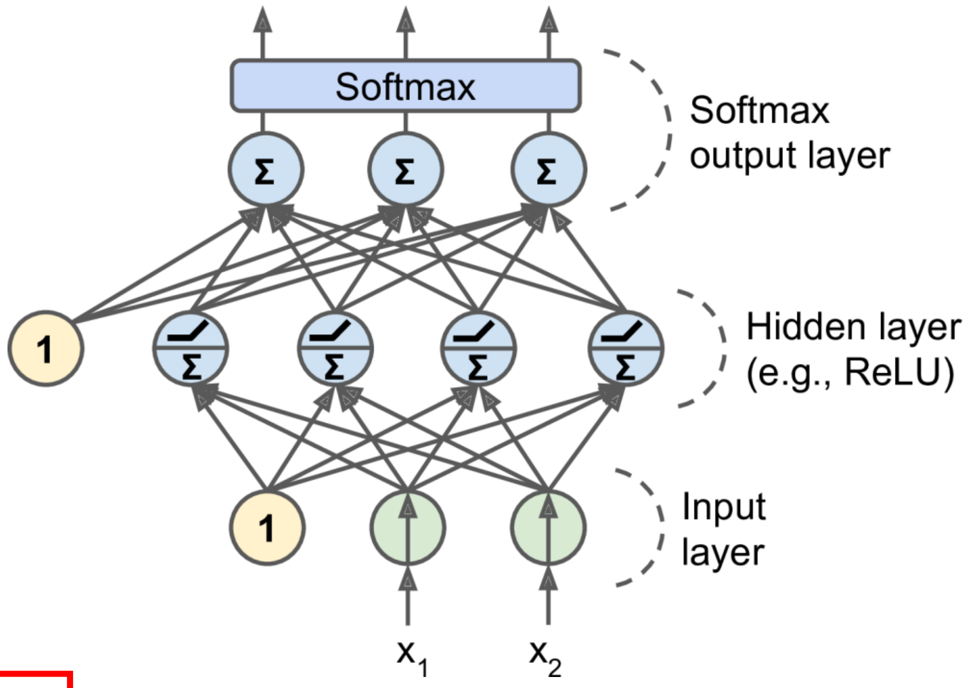
$$\mathrm{ReLU}(z) = \max(0, z)$$





3. Multi-Layer Perceptron (MLP)

- For classification, the output layer uses the *softmax function*.
- The output of each neuron corresponds to the estimated probability of the corresponding class.



$$\boxed{\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp\left(s_k(\mathbf{x})\right)}{\sum_{j=1}^{K} \exp\left(s_j(\mathbf{x})\right)}}$$

$$\boxed{\hat{y} = \underset{k}{\operatorname{argmax}}\ \sigma(\mathbf{s}(\mathbf{x}))_k}$$

Outline

1. Introduction
2. The perceptron
3. Multi-layer perceptron
4. TensorFlow’s high-level API
5. DNN using plain TensorFlow
6. Fine-tuning neural network hyperparameters
7. Exercises

4. TensorFlow's High-Level API

Creates FeatureColumn objects

```
import tensorflow as tf

feature_columns = tf.contrib.learn.infer_real_valued_columns_from_input(X_train)
dnn_clf = tf.contrib.learn.DNNClassifier(hidden_units=[300, 100], n_classes=10,
                                         feature_columns=feature_columns)
dnn_clf.fit(x=X_train, y=y_train, batch_size=50, steps=40000)
```

The new API is tf.estimator.DNNClassifier

```
>>> dnn_clf.evaluate(X_test, y_test)
{'accuracy': 0.98180002, 'global_step': 40000, 'loss': 0.073678359}
```

Outline

1. Introduction
2. The perceptron
3. Multi-layer perceptron
4. TensorFlow’s high-level API
5. DNN using plain TensorFlow
6. Fine-tuning neural network hyperparameters
7. Exercises

5. DNN Using Plain TensorFlow

- **Construction Phase**

1. Define the parameters and place holder nodes.

```
import tensorflow as tf

n_inputs = 28*28  # MNIST
n_hidden1 = 300
n_hidden2 = 100
n_outputs = 10

X = tf.placeholder(tf.float32, shape=(None, n_inputs), name="X")
y = tf.placeholder(tf.int64, shape=(None), name="y")
```

5. DNN Using Plain TensorFlow

2. Construct the two hidden layers and the output layer using the fully_connected function. ReLU activation function is used by default.

```
from tensorflow.contrib.layers import fully_connected

with tf.name_scope("dnn"):
    hidden1 = fully_connected(X, n_hidden1, scope="hidden1")
    hidden2 = fully_connected(hidden1, n_hidden2, scope="hidden2")
    logits = fully_connected(hidden2, n_outputs, scope="outputs",
                             activation_fn=None)
```

5. DNN Using Plain TensorFlow

3. Define loss function and training operation.

```
with tf.name_scope("loss"):
    xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(
                    labels=y, logits=logits)
    loss = tf.reduce_mean(xentropy, name="loss")

learning_rate = 0.01
with tf.name_scope("train"):
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    training_op = optimizer.minimize(loss)

with tf.name_scope("eval"):
    correct = tf.nn.in_top_k(logits, y, 1)
    accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))
```

$$\boxed{J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)}$$

Mean that targets are in top 1 output.

5. DNN Using Plain TensorFlow

- **Execution Phase**

1. Initialize variables, construct a saver object, and define parameters.

```
init = tf.global_variables_initializer()
saver = tf.train.Saver()

from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/")

n_epochs = 400
batch_size = 50
```

5. DNN Using Plain TensorFlow

2. Train the model.

```
with tf.Session() as sess:
    init.run()
    for epoch in range(n_epochs):
        for iteration in range(mnist.train.num_examples // batch_size):
            X_batch, y_batch = mnist.train.next_batch(batch_size)
            sess.run(training_op, feed_dict={X: X_batch, y: y_batch})
        acc_train = accuracy.eval(feed_dict={X: X_batch, y: y_batch})
        acc_test = accuracy.eval(feed_dict={X: mnist.test.images,
                                            y: mnist.test.labels})
        print(epoch, "Train accuracy:", acc_train, "Test accuracy:", acc_test)

    save_path = saver.save(sess, "./my_model_final.ckpt")
```

5. DNN Using Plain TensorFlow

- **Using the Neural Network**

```
with tf.Session() as sess:
    saver.restore(sess, "./my_model_final.ckpt")
    X_new_scaled = [...]  # some new images (scaled from 0 to 1)
    Z = logits.eval(feed_dict={X: X_new_scaled})
    y_pred = np.argmax(Z, axis=1)
```

6. Fine-Tuning Neural Network Hyperparameters

- Number of Hidden Layers
- Number of Neurons per Hidden Layer
- Activation Functions

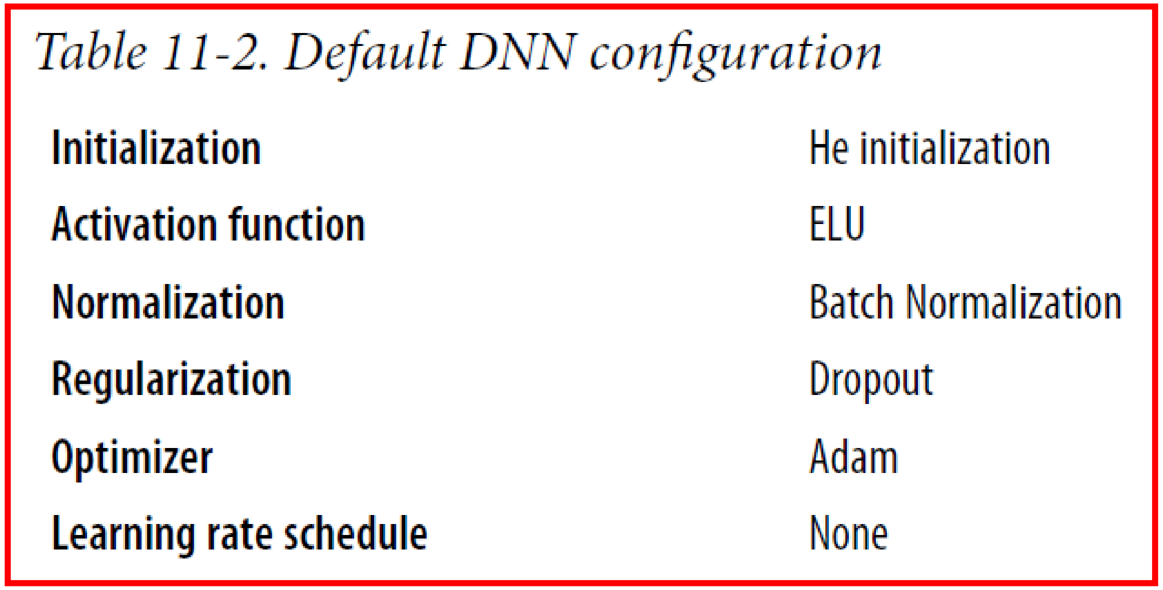


Table 11-2. Default DNN configuration

Initialization	He initialization
Activation function	ELU
Normalization	Batch Normalization
Regularization	Dropout
Optimizer	Adam
Learning rate schedule	None

Summary

1. Introduction
2. The perceptron
3. Multi-layer perceptron
4. TensorFlow’s high-level API
5. DNN using plain TensorFlow
6. Fine-tuning neural network hyperparameters
7. Exercises

Exercises

From Chapter 10, solve exercises:

- 5
- 6
- 9