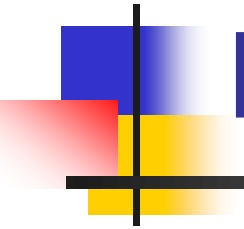
Deep Learning Using TensorFlow

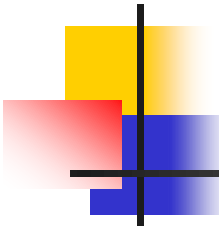


Dr. Ash Pahwa

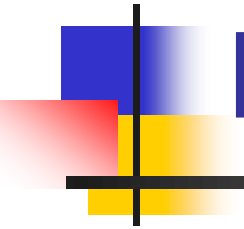
Lesson 4: Building Neural Network Models

Lesson 4.1: Building Neural Networks Models Using TensorFlow

Outline



- Example#1: Simple Neural Network
  - Load Libraries and Enter Training Data
  - Specify Constants: Learning Rate + Epoch
  - Build the Neural Network Model
  - Compute Layer#2 + Output
  - Define Cost & Optimization Functions
  - Initialize All Variables
  - Train the Model
- Example#2: Neural Network Using Iris Dataset
  - Load Libraries
  - Read Dataset
  - Encode the response categorical variable
  - Split data into Training and Testing
  - Build the Neural Network Model + Compute Output
  - Define Cost & Optimization Functions
  - Initialize All Variables & Train the Model



Example#1:

Software: TensorFlow

Dataset: XOR Data

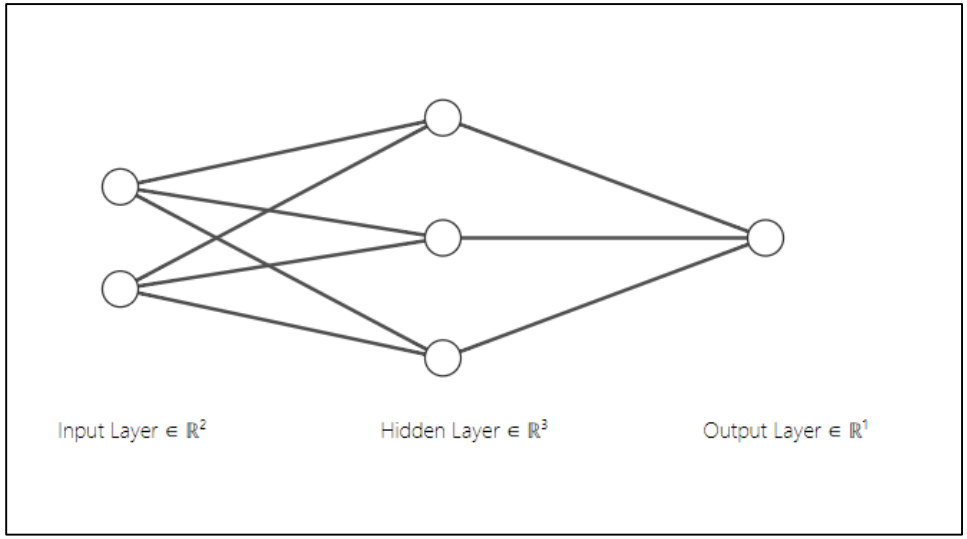
Procedure to Build Neural Network

- 1. Load Libraries and Enter Training Data
- 2. Specify Constants: Learning Rate + Epoch
- 3. Create the Neural Network Model
- 4. Compute Layer#2 + Output
- 5. Define Cost & Optimization Functions
- 6. Initialize All Variables
- 7. Train the Model

Example#1: Neural Network

- Input Neurons = 2
- Hidden Layer#1 Neurons = 3
- Output Neurons = 1

Input 1	Input 2	Output
0	0	0
1	0	1
0	1	1
1	1	0



1. Load Libraries + Enter Training Data

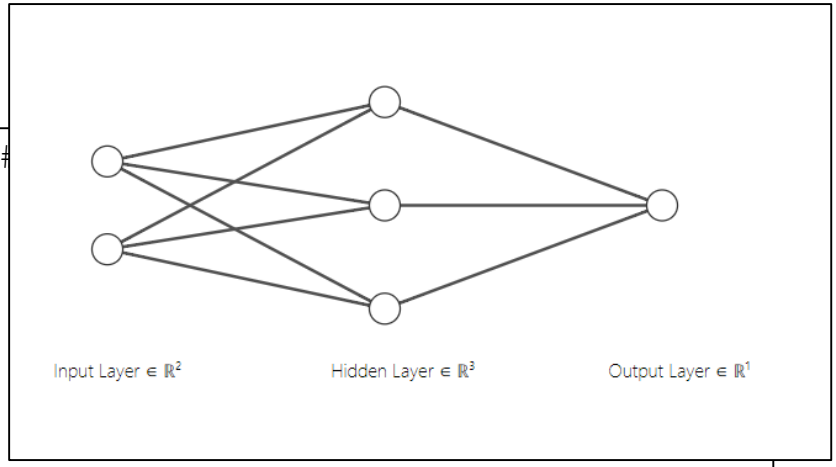
Input 1	Input 2	Output
0	0	0
1	0	1
0	1	1
1	1	0

```
################################################
# 1. Load the libraries + Training Data
import tensorflow as tf
import numpy as np

x_data = np.array([
         [0,0],[1,0],[0,1],[1,1]
         ])
x_data
Out[13]:
array([[0, 0],
       [1, 0],
       [0, 1],
       [1, 1]])

y_data = np.array([
         [0],[1],[1],[0]
        ])
y_data
Out[15]:
array([[0],
       [1],
       [1],
       [0]])
```

2. Specify Constants Learning Rate + Epochs

```
#####################################################
# 2. Enter Constants
#

learning_rate = 0.1

epochs = 10000
```

3. Build the Neural Network Model




```
##################################################
# 3. Build Neural Network: Define Weights and Bias
#
n_input = 2
n_hidden = 3
n_output = 1

X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)

W1 = tf.Variable(tf.random_uniform([n_input, n_hidden], -1.0, 1.0))
W1
Out[21]: <tf.Variable 'Variable:0' shape=(2, 3) dtype=float32_ref>

W2 = tf.Variable(tf.random_uniform([n_hidden, n_output], -1.0, 1.0))
W2
Out[23]: <tf.Variable 'Variable_1:0' shape=(3, 1) dtype=float32_ref>

b1 = tf.Variable(tf.zeros([n_hidden]), name='Bias1')
b1
Out[25]: <tf.Variable 'Bias1:0' shape=(3,) dtype=float32_ref>

b2 = tf.Variable(tf.zeros([n_output]), name='Bias2')
b2
Out[27]: <tf.Variable 'Bias2:0' shape=(1,) dtype=float32_ref>
```

	Variable	Tensor Shape
1	W1	2, 3
2	W2	3, 1
3	b1	3,
4	b2	1,

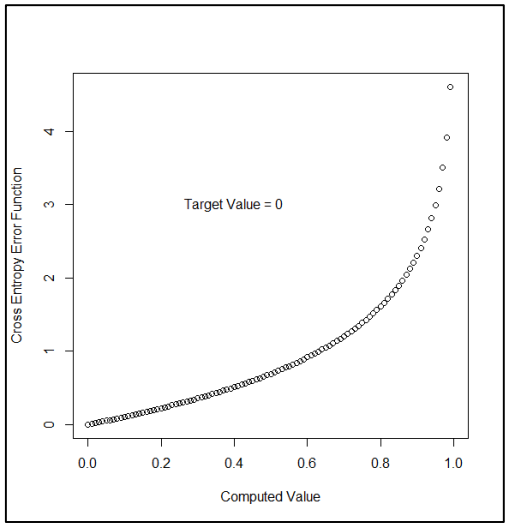
4. Compute Layer#2 + Output

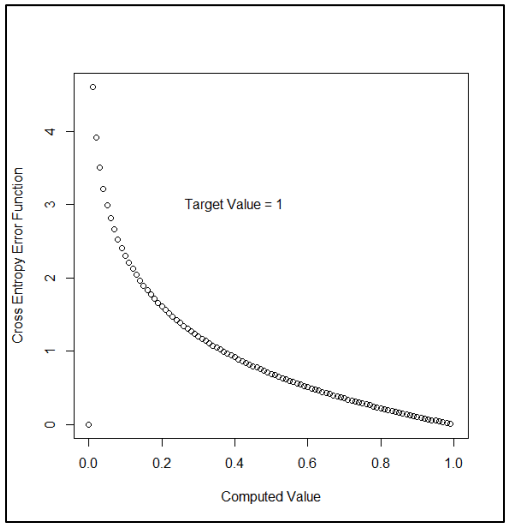
```
#########################################################
# 4. Compute Layer#2 + Computed Output
#

L2 = tf.sigmoid(tf.matmul(X,W1) + b1)

compOutput = tf.sigmoid(tf.matmul(L2,W2) + b2)
```

Cost Function
Cross Entropy Cost Function

- For Linear regression model – where response value is numerical
  - $Cost\ Function = \sum(y_{observed} - y_{Computed})^2$
- For models - where the response variable is categorical (0 or 1)
  - Cross Entropy Cost Function
  - $Cost\ Function = -\sum(target * \log(compValue) + (1 - target) * \log(1 - compValue))$

	Target	Computed Value	Cross Entropy Error
1	0	0.9	2.3
2	0	0.5	0.69
3	0	0.1	0.11
4	0	0.0	0
5	1	1	0
6	1	0.9	0.11
7	1	0.5	0.69
8	1	0.1	2.30





5. Define Cost & Optimization Functions

- Cost Function
  - Cross Entropy Cost Function
- Gradient Descent
  - Feed "Learning Rate" as a parameter to the optimization function

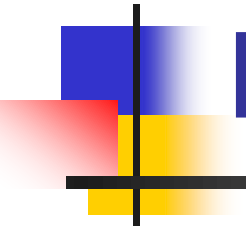
```
#########################################################
# 5. Define Cost and Optimization Function
# Cost = Cross Entropy Cost Function
# Optimization = Gradient Descent
#
cost = tf.reduce_mean(-Y*tf.log(compOutput) - (1-Y)*tf.log(1-compOutput))

optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

6. Initialize All Variables

```
#########################################################
# 6. Initialize All Variables
#

init = tf.global_variables_initializer()
```

7. Train the Model

```
with tf.Session() as session:
        session.run(init)

        for step in range(epochs):
            session.run(optimizer, feed_dict={X: x_data, Y: y_data})

            if step % 1000 == 0:
                print (session.run(cost, feed_dict={X: x_data, Y: y_data}))


0.741868
0.689807
0.652354
0.303535
0.100745
0.0535813
0.0355308
0.0263162
0.0207975
0.017146
```



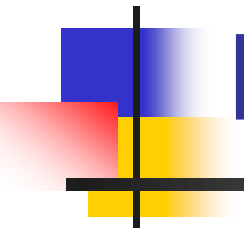
Example#2

Software: TensorFlow

Dataset: Iris

Procedure to Build Neural Network

- 1. Load Libraries
- 2. Read Dataset
- 3. Encode the response categorical variable
- 4. Split data into Training and Testing
- 5. Build the Neural Network Model + Compute Output
- 6. Define Cost & Optimization Functions
- 7. Initialize All Variables & Train the Model



Create the Training/Testing Dataset

Dataset: Iris

	A	B	C	D	E
1	SepalLength	SepalWidth	PetalLength	PetalWidth	Name
2	5.1	3.5	1.4	0.2	setosa
3	4.9	3	1.4	0.2	setosa
4	4.7	3.2	1.3	0.2	setosa
5	4.6	3.1	1.5	0.2	setosa
6	5	3.6	1.4	0.2	setosa
7	5.4	3.9	1.7	0.4	setosa
8	4.6	3.4	1.4	0.3	setosa
9	5	3.4	1.5	0.2	setosa
10	4.4	2.9	1.4	0.2	setosa
11	4.9	3.1	1.5	0.1	setosa
12	5.4	3.7	1.5	0.2	setosa
13	4.8	3.4	1.6	0.2	setosa
14	4.8	3	1.4	0.1	setosa
15	4.3	3	1.1	0.1	setosa
16	5.8	4	1.2	0.2	setosa
17	5.7	4.4	1.5	0.4	setosa
18	5.4	3.9	1.3	0.4	setosa
19	5.1	3.5	1.4	0.3	setosa
20	5.7	3.8	1.7	0.3	setosa

1. Load the Libraries

```
import tensorflow as tf

import numpy as np

from sklearn import datasets

from sklearn.model_selection import train_test_split

RANDOM_SEED = 42

tf.set_random_seed(RANDOM_SEED)
```

2. Read the Dataset

```
################################################
# Read the Dataset
#
iris    = datasets.load_iris()
features = iris["data"]
type(features)
Out[65]: numpy.ndarray

features[:5,:]
Out[66]:
array([[ 5.1,  3.5,  1.4,  0.2],
       [ 4.9,  3. ,  1.4,  0.2],
       [ 4.7,  3.2,  1.3,  0.2],
       [ 4.6,  3.1,  1.5,  0.2],
       [ 5. ,  3.6,  1.4,  0.2]])

labels = iris["target"]
labels
Out[68]:
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

Response variable "target" is "categorical"
Values: 0 ,1, 2

0: Setosa
1: Versicolor
2: Virginica

3. Encode the Response Categorical Variable

- Since the "response variable" is categorical
  - Create 3 output variables

```
#####################################################
# Encode the response variable
#

one_hot = np.zeros(shape=(len(labels), 3))

# for classification we need the labels to be in
one hot vectors

for i in range(0, len(labels)):
    one_hot[i, labels[i]] = 1
```

Encoded Response Variable

Setosa

```
one_hot[0:25,:]
Out[89]:
array([[ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 1.,  0.,  0.]])
```

Versicolor

```
one_hot[50:75,:]
Out[90]:
array([[ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.]])
```

Virginica

```
one_hot[100:125,:]
Out[91]:
array([[ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  1.]])
```

4. Split the Data into Training and Testing

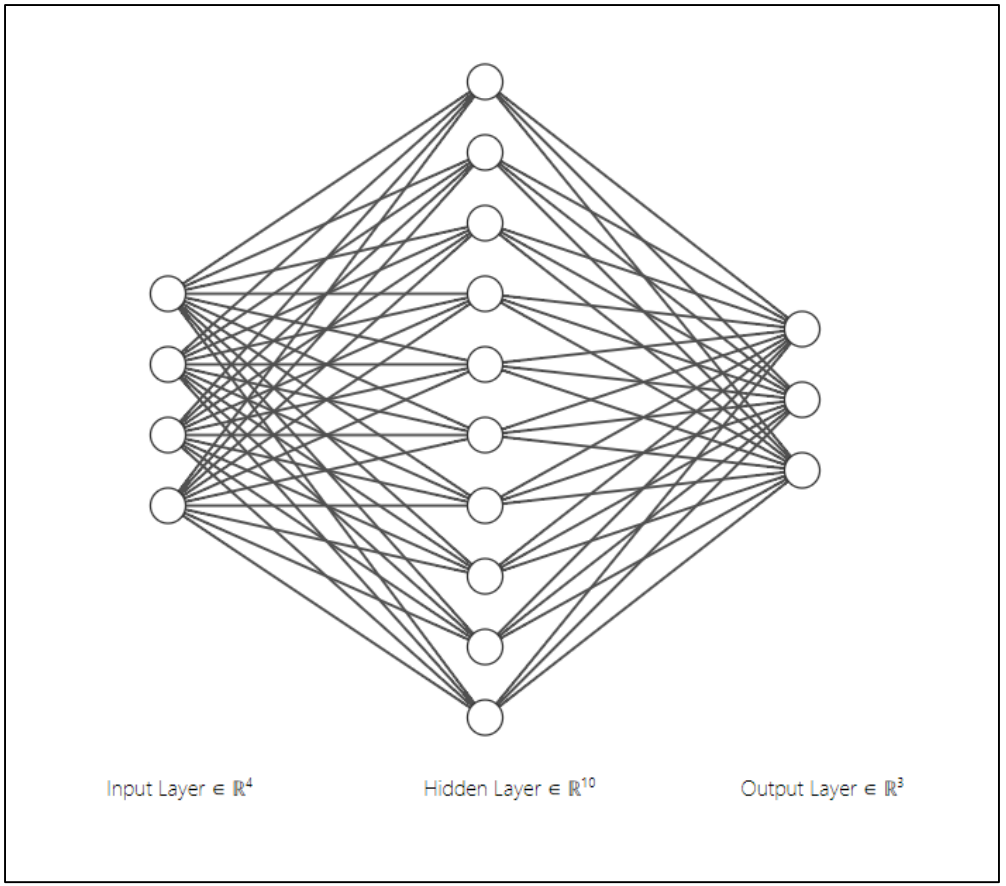
```
# Split dataset into training + testing (33%)
#
train_feats, test_feats, train_lab, test_lab = train_test_split(features, one_hot,
test_size=0.33, random_state=RANDOM_SEED)

# Training data
train_feats.shape
Out[137]: (100, 4)

train_feats[:5,:]
Out[138]:
array([[ 5.7,  2.9,  4.2,  1.3],
       [ 7.6,  3. ,  6.6,  2.1],
       [ 5.6,  3. ,  4.5,  1.5],
       [ 5.1,  3.5,  1.4,  0.2],
       [ 7.7,  2.8,  6.7,  2. ]])

train_lab.shape
Out[139]: (100, 3)

train_lab[:5,:]
Out[140]:
array([[ 0.,  1.,  0.],
       [ 0.,  0.,  1.],
       [ 0.,  1.,  0.],
       [ 1.,  0.,  0.],
       [ 0.,  0.,  1.]])
```

```
# Testing data
test_feats.shape
Out[142]: (50, 4)

test_feats[:5,:]
Out[143]:
array([[ 6.1,  2.8,  4.7,  1.2],
       [ 5.7,  3.8,  1.7,  0.3],
       [ 7.7,  2.6,  6.9,  2.3],
       [ 6. ,  2.9,  4.5,  1.5],
       [ 6.8,  2.8,  4.8,  1.4]])

test_lab.shape
Out[144]: (50, 3)

test_lab[:5,:]
Out[145]:
array([[ 0.,  1.,  0.],
       [ 1.,  0.,  0.],
       [ 0.,  0.,  1.],
       [ 0.,  1.,  0.],
       [ 0.,  1.,  0.]])
```

5. Create the Neural Network Model

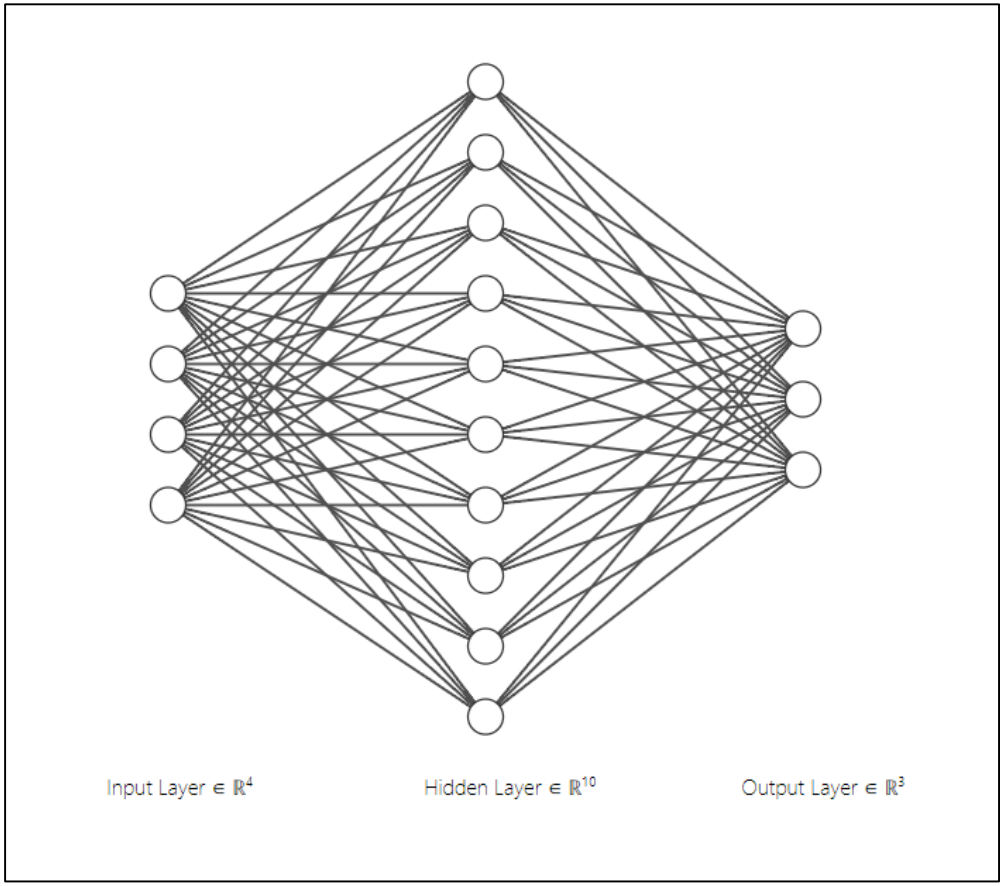
Neural Network Architecture

- Input Neurons = 4
  - Sepal Length
  - Sepal Width
  - Petal Length
  - Petal Width
- Hidden Layer = 10
- Output Layer = 3
  - Setosa
  - Versicolor
  - Virginica



Neural Network Architecture TensorFlow Code

```
####################################
# Create the Neural Network Model

feat_shape = train_feats.shape[1]
feat_shape
Out[50]: 4

hidden_nodes = 10

out_shape = train_lab.shape[1]
out_shape
Out[53]: 3
```

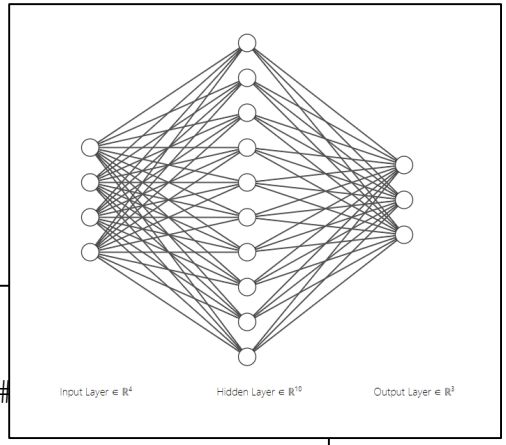


Axis in Python

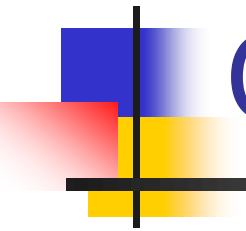
```
a = np.array( [ (1,2,3),(3,4,5)])

print(a)
[[1 2 3]
 [3 4 5]]

print(a.sum(axis=0))
[4 6 8]

print(a.sum(axis=1))
[ 6 12]
```

Build the Model

feat_shape=4, hidden_nodes=10, out_shape=3




```
inputs = tf.placeholder("float", shape=[None, feat_shape])
outputs = tf.placeholder("float", shape=[None, out_shape])
###########################################################
W1 = tf.get_variable(name="W1",
                     shape=[feat_shape, hidden_nodes],
                     initializer=tf.contrib.layers.xavier_initializer())

b1 = tf.get_variable(name="b1",
                     shape=[hidden_nodes],
                     initializer=tf.constant_initializer(0.0))

H1 = tf.matmul(inputs, W1) + b1
H1 = tf.nn.relu(H1)

###########################################################
W2 = tf.get_variable(name="W2",
                    shape=[hidden_nodes, out_shape],
                    initializer=tf.contrib.layers.xavier_initializer())

b2 = tf.get_variable(name="b2",
                     shape=[out_shape],
                     initializer=tf.constant_initializer(0.0))

pred_tensor = tf.matmul(H1, W2) + b2
predict = tf.argmax(pred_tensor, axis=1)
```

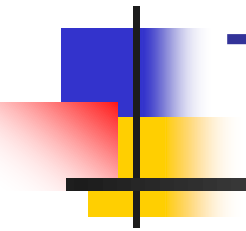


6. Define Cost & Optimization Functions

Define Cost and Optimization Function

- Cost Function
  - Cross Entropy Cost Function
- Gradient Descent
  - Feed "Learning Rate" as a parameter to the optimization function

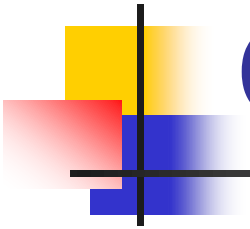
```
cost    = tf.reduce_mean
(tf.nn.softmax_cross_entropy_with_logits_v2(labels=outputs, logits=pred_tensor))

updates = tf.train.GradientDescentOptimizer(0.01).minimize(cost)
```



7. Initialize All Variables and Train the Model

Initialize and Train the Model

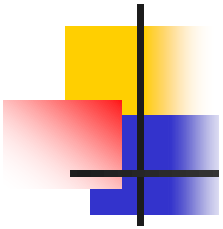
```
with tf.Session(graph = graph) as sess:
    init = tf.global_variables_initializer()
    sess.run(init)

    for epoch in range(100):
        # Train with each example
        for i in range(len(train_feats)):
            op, cst = sess.run([updates, cost],
                    feed_dict={inputs: train_feats[i: i + 1], outputs: train_lab[i: i + 1]})

        test_accuracy  = np.mean(np.argmax(test_lab, axis=1) ==
                             sess.run(predict,
                    feed_dict={inputs: test_feats, outputs: test_lab}))

        if (epoch % 10) == 0:
            print("Epoch: %d, acc: %.2f, cost: %.5f"
              % (epoch, test_accuracy, cst))
```



Output

```
Epoch: 0, acc: 0.32, cost: 0.62991
Epoch: 10, acc: 0.98, cost: 0.16621
Epoch: 20, acc: 0.98, cost: 0.02675
Epoch: 30, acc: 0.98, cost: 0.00887
Epoch: 40, acc: 0.98, cost: 0.00344
Epoch: 50, acc: 0.98, cost: 0.00150
Epoch: 60, acc: 0.98, cost: 0.00073
Epoch: 70, acc: 0.98, cost: 0.00040
Epoch: 80, acc: 0.98, cost: 0.00025
Epoch: 90, acc: 0.98, cost: 0.00018
```

Summary



- Example#1: Simple Neural Network
  - Load Libraries and Enter Training Data
  - Specify Constants: Learning Rate + Epoch
  - Build the Neural Network Model
  - Compute Layer#2 + Output
  - Define Cost & Optimization Functions
  - Initialize All Variables
  - Train the Model
- Example#2: Neural Network Using Iris Dataset
  - Load Libraries
  - Read Dataset
  - Encode the response categorical variable
  - Split data into Training and Testing
  - Build the Neural Network Model + Compute Output
  - Define Cost & Optimization Functions
  - Initialize All Variables & Train the Model