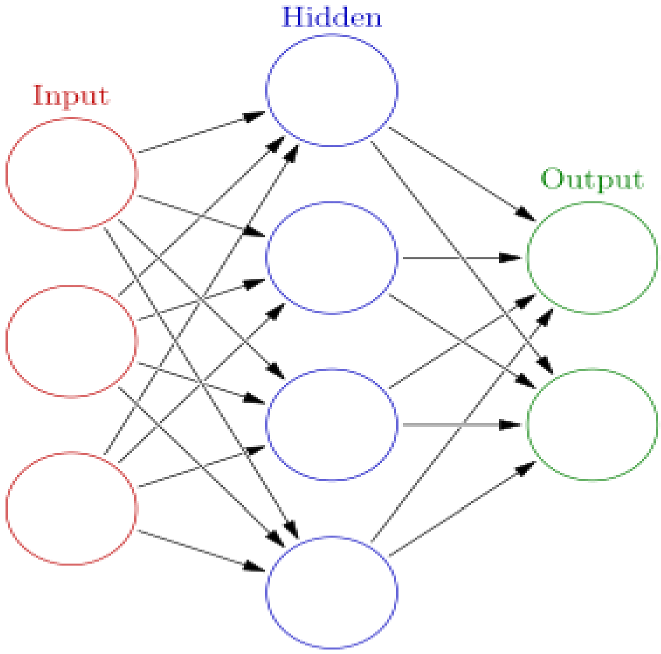
Implementing a one-layer Neural Network

We will illustrate how to create a one hidden layer NN

We will use the iris data for this exercise

We will build a one-hidden layer neural network to predict the fourth attribute, Petal Width from the other three (Sepal length, Sepal width, Petal length).



In [1]:

```
"""
Implementing a one-layer Neural Network
We will illustrate how to create a one hidden layer NN
We will use the iris data for this exercise
We will build a one-hidden layer neural network
 to predict the fourth attribute, Petal Width from
 the other three (Sepal length, Sepal width, Petal length).
"""

import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf1
import tensorflow.compat.v1 as tf
from sklearn import datasets
```

```
tf.disable_v2_behavior()
from tensorflow.python.framework import ops
ops.reset_default_graph()
```

In [2]:

```
ops.reset_default_graph()
```

In [3]:

```
iris = datasets.load_iris()
x_vals = np.array([x[0:3] for x in iris.data])
y_vals = np.array([x[3] for x in iris.data])
```

In [4]:

```
# Create graph session
sess = tf.Session()
```

In [5]:

```
# make results reproducible
seed = 2
tf.set_random_seed(seed)
np.random.seed(seed)
```

In [6]:

```
# Split data into train/test = 80%/20%
train_indices = np.random.choice(len(x_vals), round(len(x_vals)*0.8), repla
ce=False)
test_indices = np.array(list(set(range(len(x_vals))) - set(train_indices)))
x_vals_train = x_vals[train_indices]
x_vals_test = x_vals[test_indices]
y_vals_train = y_vals[train_indices]
y_vals_test = y_vals[test_indices]
```

In [7]:

```
# Normalize by column (min-max norm)
def normalize_cols(m):
    col_max = m.max(axis=0)
    col_min = m.min(axis=0)
    return (m-col_min) / (col_max - col_min)

x_vals_train = np.nan_to_num(normalize_cols(x_vals_train))
x_vals_test = np.nan_to_num(normalize_cols(x_vals_test))
```

In [8]:

```
# Declare batch size
batch_size = 50

# Initialize placeholders
x_data = tf.placeholder(shape=[None, 3], dtype=tf.float32)
y_target = tf.placeholder(shape=[None, 1], dtype=tf.float32)
```

In [9]:

```
# Create variables for both NN layers
hidden_layer_nodes = 10
A1 = tf.Variable(tf.random_normal(shape=[3,hidden_layer_nodes])) # inputs -
> hidden nodes
b1 = tf.Variable(tf.random_normal(shape=[hidden_layer_nodes]))   # one bias
es for each hidden node
A2 = tf.Variable(tf.random_normal(shape=[hidden_layer_nodes,1])) # hidden i
nputs -> 1 output
b2 = tf.Variable(tf.random_normal(shape=[1]))   # 1 bias for the output


# Declare model operations
hidden_output = tf.nn.relu(tf.add(tf.matmul(x_data, A1), b1))
final_output = tf.nn.relu(tf.add(tf.matmul(hidden_output, A2), b2))

# Declare loss function (MSE)
loss = tf.reduce_mean(tf.square(y_target - final_output))

# Declare optimizer
my_opt = tf.train.GradientDescentOptimizer(0.005)
train_step = my_opt.minimize(loss)
```

In [10]:

```
# Initialize variables
init = tf.global_variables_initializer()
sess.run(init)

# Training loop
loss_vec = []
test_loss = []
for i in range(500):
    rand_index = np.random.choice(len(x_vals_train), size=batch_size)
    rand_x = x_vals_train[rand_index]
    rand_y = np.transpose([y_vals_train[rand_index]])
    sess.run(train_step, feed_dict={x_data: rand_x, y_target: rand_y})
```

```
    temp_loss = sess.run(loss, feed_dict={x_data: rand_x, y_target: rand_y}
)
    loss_vec.append(np.sqrt(temp_loss))

    test_temp_loss = sess.run(loss, feed_dict={x_data: x_vals_test, y_targe
t: np.transpose([y_vals_test])})
    test_loss.append(np.sqrt(test_temp_loss))
    if (i+1)%50==0:
        print('Generation: ' + str(i+1) + '. Loss = ' + str(temp_loss))
```