# Introduction to Neural Networks

**LAB # 11 & 12**

**Fall 2021**

**Data Analytics Lab**

Submitted by: **Shah Raza**

Registration No.: **18PWCSE1658**

Class Section: **B**

"On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work."

Student Signature: ______________

Submitted to:

**Engr. Mian Ibad Ali Shah**

9 March 2022

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

**Introduction to Neural Network:**

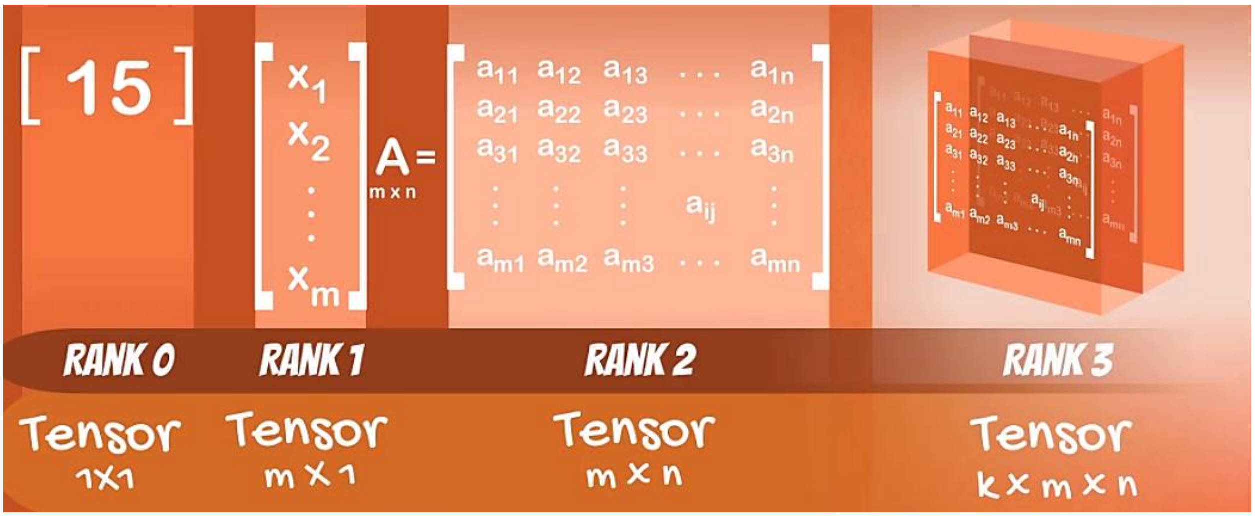
Before that, let's study a little mathematics (Linear Algebra)

Scalar (0D)

Vector (1D)

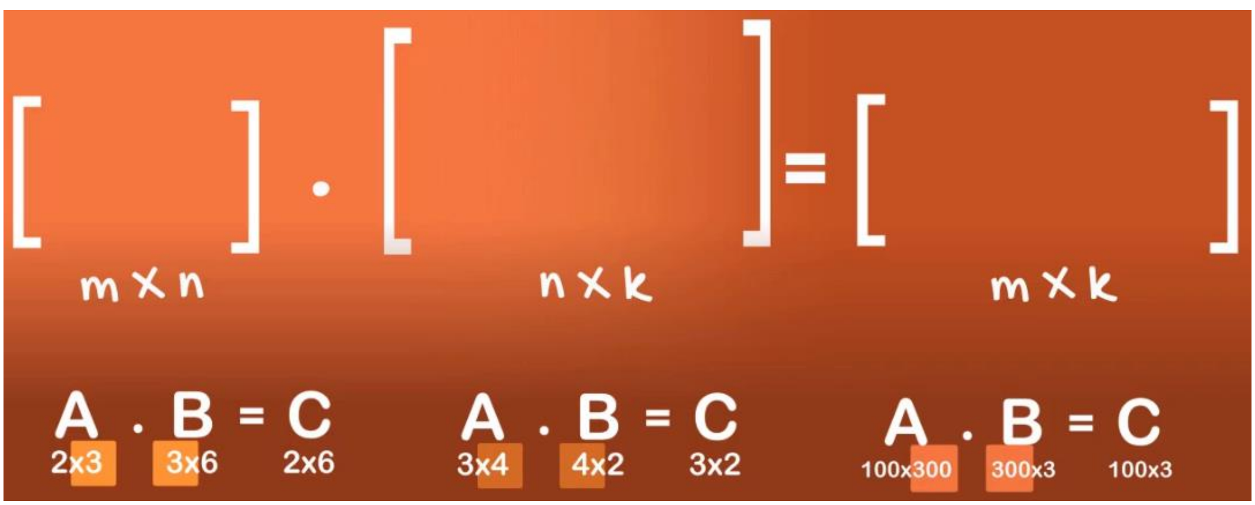
Matrix (2D) -> combination of vectors

Tensor -> collection of matrices

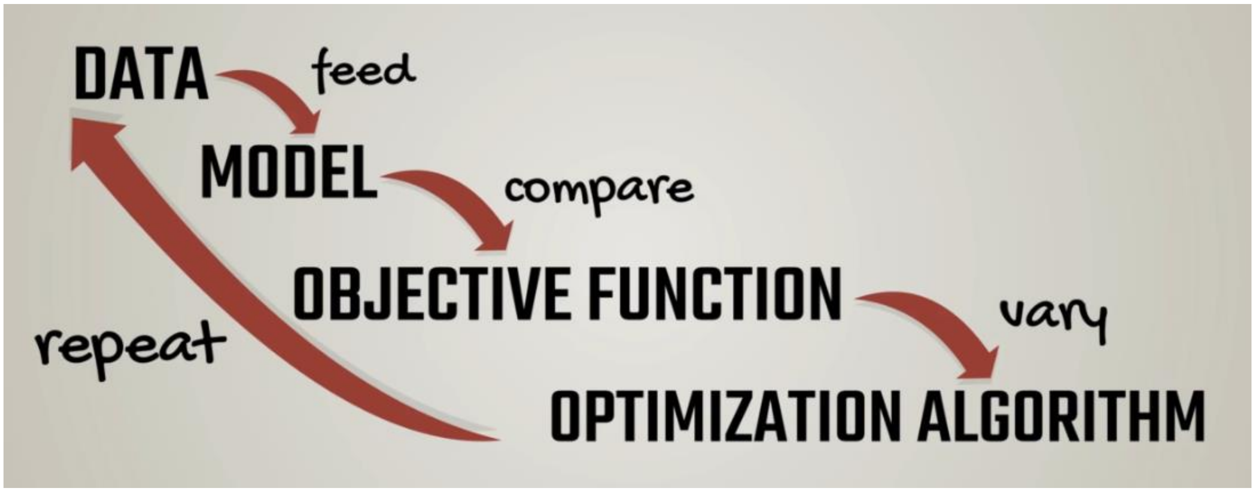



Transposing an array (vector, matrix or tensor) -> x.T in python

Dot product (scalar product) -> np.dot(x,y)



Ingredients of an Algorithm:




Training the model:

Coffee machine making coffee

Self-driving cars

**Types of Machine Learning:**

Supervised

Unsupervised

Reinforcement Learning

**What is a model?**

Example: Linear Regression

$Y=xw+b$

**Objective Function:**

Objective function is the measure used to evaluate how well the model's outputs match the desired correct values.

It has two types:

- Loss Function (Example: Supervised Learning) aka cost function
    - For example: Regression cost function: L2 norm.
    - Classification: Cross entropy
- Reward Function (Example: Reinforcement Learning)

Optimization Algorithm: Example, Gradient Descent

**Task 1:**

### Choosing the objective function and the optimization method

In [27]:
```
# Again, we use a loss function. mean_squared_error is the scaled L2-norm (per observation)
mean_loss = tf.losses.mean_squared_error(labels=targets, predictions=outputs) / 2.
# Note that there also exists a function tf.nn.l2_loss.

# Instead of implementing Gradient Descent on our own, in TensorFlow we can simply state
# "Minimize the mean loss by using Gradient Descent with a given learning rate"
# Simple!
optimize = tf.train.GradientDescentOptimizer(learning_rate=0.05).minimize(mean_loss)
```

### Prepare for execution

In [28]:
```
# So far we've defined the placeholders, variables, the loss function and the optimization method.
# The actual training happens inside sessions.
sess = tf.InteractiveSession()
```

### Initializing variables

In [29]:
```
# Before we start training, we need to initialize our variables: the weights and biases.
initializer = tf.global_variables_initializer()

sess.run(initializer)
```

Activate Windows
Go to Settings to activat

### Learning

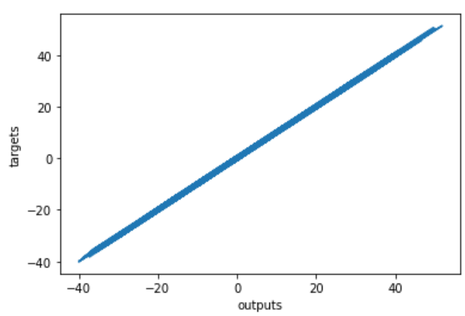
In [31]:
```python
# As in the previous lab, we train for a set number (100) of iterations over the dataset
for i in range(100):
    # sess.run is the session's function to actually do something, anything.
    # Above, we used it to initialize the variables.
    # Here, we use it to feed the training data to the computational graph, defined by the feed_dict parameter
    # So the line of code means: "Run the optimize and mean_loss operations by filling the placeholder
    # objects with data from the feed_dict parameter".
    _, curr_loss = sess.run([optimize, mean_loss],
        feed_dict={inputs: training_data['inputs'], targets: training_data['targets']})

    print(curr_loss)
```

```
224.36153
96.79172
45.56499
24.258543
14.921057
10.505043
8.188638
6.8132415
5.8881536
5.197546
4.6420927
4.1735177
3.7667549
3.4077315
3.087803
2.801146
2.5434911
```

Activate Windows
Go to Settings to activate W

### Plotting the data

In [32]:
```python
out = sess.run([outputs],
               feed_dict={inputs: training_data['inputs']})
plt.plot(np.squeeze(out), np.squeeze(training_data['targets']))
plt.xlabel('outputs')
plt.ylabel('targets')
plt.show()
```




In [ ]:

Activate Windows
Go to Settings to activate

## Task 2:

# Simple Linear Regression. Minimal example

### Import the relevant libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

### Generate random input data to train on

In [2]:
```
observations = 1000
xs = np.random.uniform(low=-10, high=10, size=(observations,1))
zs = np.random.uniform(-10, 10, (observations,1))

inputs = np.column_stack((xs,zs))

print (inputs.shape)
```

```
(1000, 2)
```

### Generate the targets we will aim at

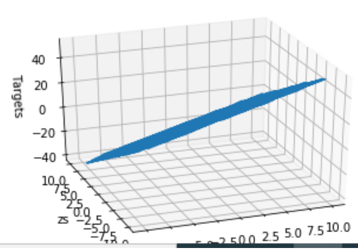
In [3]:
```
# We want to make a function and see if the algorithm has learned it.
# We add a small random noise to the function i.e. f(x,z) = 2x - 3z + 5 + <small noise> as in real life datasets
noise = np.random.uniform(-1, 1, (observations,1))
```

Activate Windows
Go to Settings to activate W

### Plot the training data

The point is to see that there is a strong trend that our model should learn to reproduce.

In [7]:
```
targets = targets.reshape(observations,)

fig = plt.figure()

ax = fig.add_subplot(111, projection='3d')

ax.plot(xs, zs, targets)

ax.set_xlabel('xs')
ax.set_ylabel('zs')
ax.set_zlabel('Targets')
ax.view_init(azim=250)
plt.show()
targets = targets.reshape(observations,1)
```




Activate Windows
Go to Settings to activate W

### Set a learning rate

In [9]:
```
# Play around with learning rate.
learning_rate = 0.05
```

### Train the model

In [14]:
```
for i in range (1000):

    outputs = np.dot(inputs,weights) + biases
    deltas = outputs - targets

    loss = np.sum(deltas ** 2) / 2 / observations

    print (loss)

    deltas_scaled = deltas / observations

    # Finally, apply the gradient descent update rules
    # The weights are 2x1, learning rate is 1x1 (scalar), inputs are 1000x2, and deltas_scaled are 1000x1
    # Transpose the inputs so that we get an allowed operation.
    weights = weights - learning_rate * np.dot(inputs.T,deltas_scaled)
    biases = biases - learning_rate * np.sum(deltas_scaled)
```

```
0.1682932421214414
0.16813935858891543
0.1679915681123002
0.16784962943556467
0.16771331085526878
```

Activate Windows
Go to Settings to activat

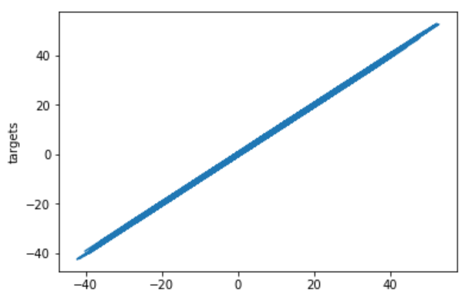
In [15]:
```
print (weights, biases)
```

```
[[ 2.00372083]
 [-3.00100363]] [5.00604098]
```

### Plot last outputs vs targets

Since they are the last ones at the end of the training, they represent the final model accuracy.
The closer this plot is to a 45 degree line, the closer target and output values are.

In [16]:
```
plt.plot(outputs,targets)
plt.xlabel('outputs')
plt.ylabel('targets')
plt.show()
```




Activate Windows
Go to Settings to activate