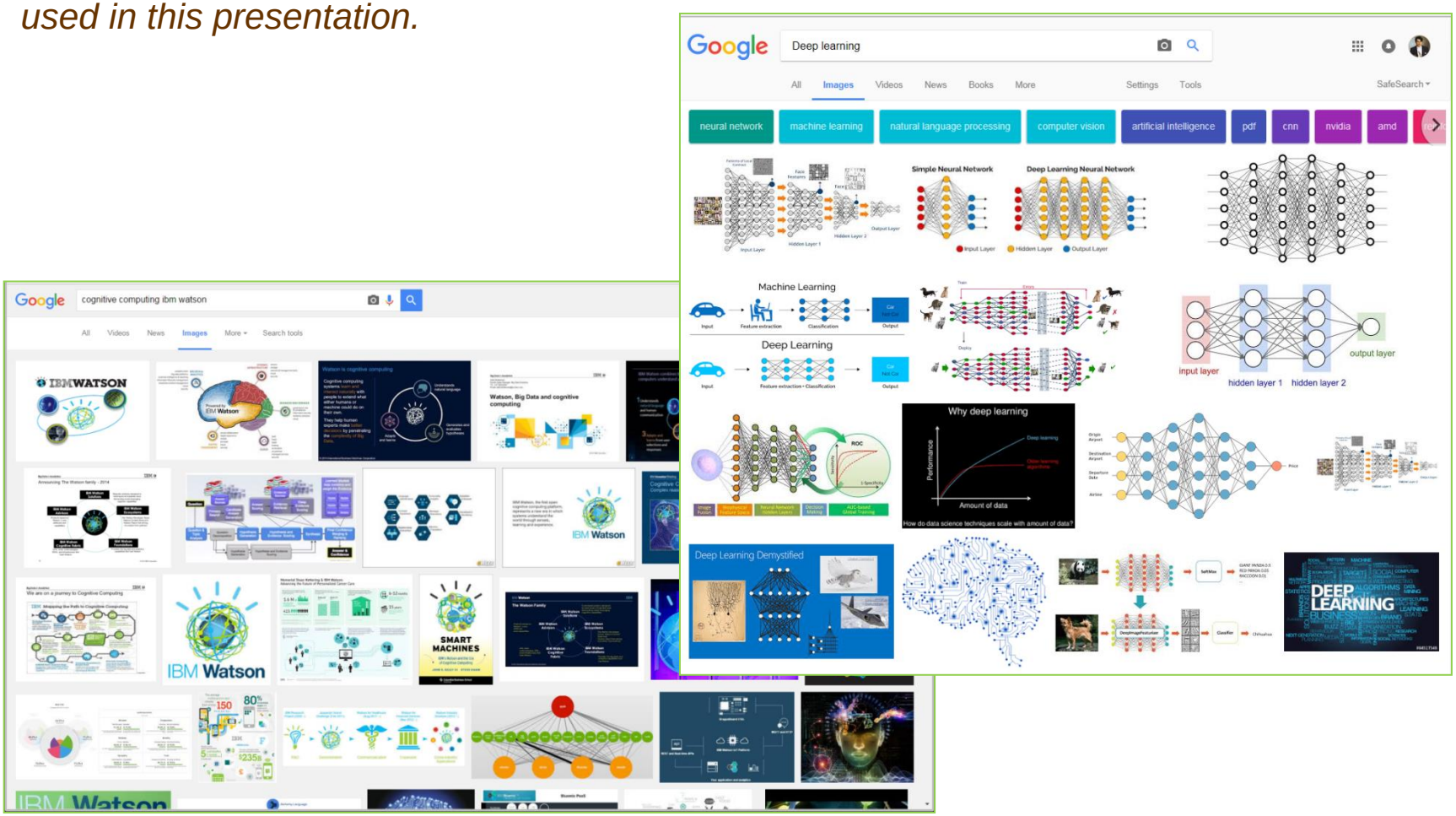
Lecture 1: Introduction to Neural Networks with TensorFlow *(Linear Regression)*

TIES4911 Deep-Learning for Cognitive Computing for Developers
Spring 2024

by:
Dr. Oleksiy Khriyenko
IT Faculty
University of Jyväskylä

Acknowledgement

I am grateful to all the creators/owners of the images that I found from Google and have used in this presentation.



TensorFlow

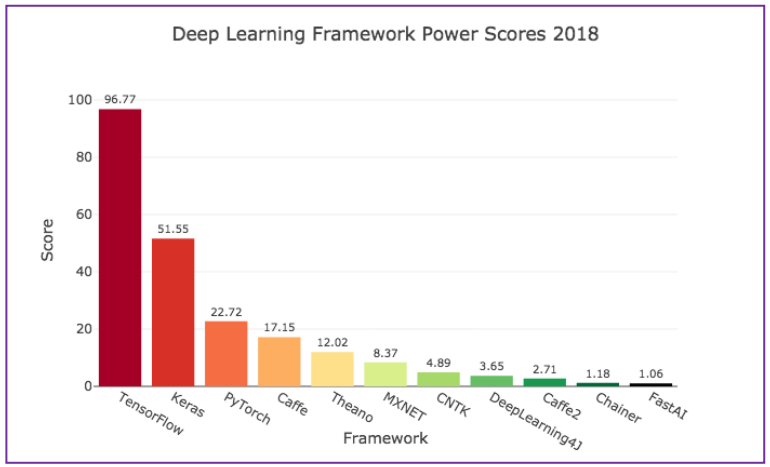
TensorFlow™ *is an open source software library for numerical computation using data flow graphs (originally developed by the Google Brain Team).* https://www.tensorflow.org/

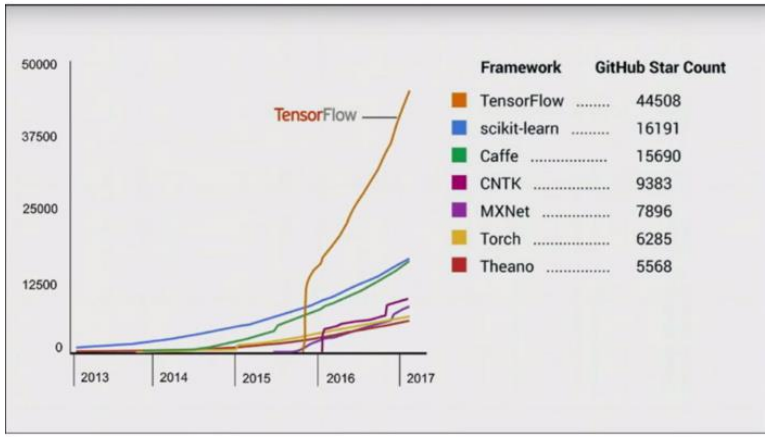
- *Python API*
- *flexible architecture allows deploying computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API*
- *could be used on various platforms from Raspberry Pi, Android, Windows, iOS, Linux to server farms*
- *Many big companies like DeepMind, Uber, AirBnB, or Dropbox have all decided to leverage this framework.*
- *is supported by large community, therefore, a lot of documentation, guidelines and tutorials are freely available in the web.*

TensorBoard *is a suite of web applications for inspecting and understanding your TensorFlow runs and graphs. Use TensorBoard to:*

- *visualize TensorFlow graph*
- *plot quantitative metrics about the execution of your graph*
- *show additional data like images that pass through it*

https://www.tensorflow.org/tensorboard/get_started
https://github.com/tensorflow/tensorboard





Some basics…

CS230 tutorial: TensorFlow for Deep Learning Research
https://cs230.stanford.edu/blog/tensorflow/

Video tutorial:

- https://www.youtube.com/watch?v=QPDsEtUK_D4
- https://www.youtube.com/watch?v=g-EvyKpZjmQ
- https://www.youtube.com/watch?v=Se9ByBnKb0o&list=PLXO45tsB95cJHXaDKpbwr5fC_CCYylw1f&index=1

To become familiar with TensorFlow, check the official low level tutorial:
https://www.tensorflow.org/guide/basics

Relevant links:
http://adventuresinmachinelearning.com/python-tensorflow-tutorial/
https://medium.com/@ODSC/5-deep-learning-frameworks-to-consider-for-2020-183e6c12cba9

TensorFlow

- *Install TensorFlow following the instructions (https://www.tensorflow.org/install/)*

You may install with native pip, or via Anaconda environment. There are might be some problems with Anaconda installation (https://github.com/ContinuumIO/anaconda-issues/issues/2533), probably caused by running antivirus or other protection software.

Check installation running the code:

```
python
>>> import tensorflow as tf
>>> print("TensorFlow version:", tf.__version__)
>>> hello = tf.constant('Hello, TensorFlow!')
>>> print(hello.numpy())
```

```
TensorFlow version: x.x.x
'Hello, TensorFlow!'
```

- *Docker image of Tensorflow*
  - *Install **Docker** on your platform of choice (https://www.docker.com/community-edition#/download). For Windows, if you have version different from Windows 10 Professional or Enterprise 64-bit, then get Docker Toolbox (https://docs.docker.com/toolbox/toolbox_install_windows/)*
  - *Install **Docker image for Tensorflow***

```
Docker run   -it -p 8888:8888   - -name   myTF   tensorflow/tensorflow
```

Optionally you may mount locla folder on a host machine to the Ddocker container image. Since VM Virtual Box by default associated with c:/Users/ path, create the locla directory somewhere there.

```
Docker run   -it -p 8888:8888 -v /c/Users/…/local:/notebooks/local   - -name   myTF   tensorflow/tensorflow
```

  - *Go to the localhost in your browser (port:8888) and provide corresponding token.*

Be aware, you are not using a localhost of hosting machine… You have to check the IP of the Docker you run (check the IP when you start the Docker) (e.g. 192.168.99.100). In this case, your open the browser on http://192.168.99.100:8888/.

- *Use Jupyter Notebook - an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. (https://jupyter.org/)*
- *Use Google Colaboratory: Google's free cloud service for AI developers using Keras, Tensorflow and PyTorch. Supports use of the free Tesla K80 GPU (https://colab.research.google.com/).*

Relevant tutorial: https://colab.research.google.com/github/cs231n/cs231n.github.io/blob/master/python-colab.ipynb
https://medium.com/deep-learning-turkey/google-colab-free-gpu-tutorial-e113627b9f5d
https://www.kdnuggets.com/2018/02/essential-google-colaboratory-tips-tricks.html

- *Take a look at Interactive Python tutorials: https://www.learnpython.org/*
https://www.datacamp.com/courses/tech:python

TensorFlow

Tensor in a deep learning framework *is the data structure used by machine learning systems. It is a container for numerical data that could be present by three main attributes:*

- *rank refers to the tensor's number of axes.*
- *shape refers to the number of dimensions along each axis. E.g. (2,2) or (2,5,10), etc.*
- *data type refers to the type of data contained in tensor (e.g. float32, float64, uint8, int32, int64, etc.).*

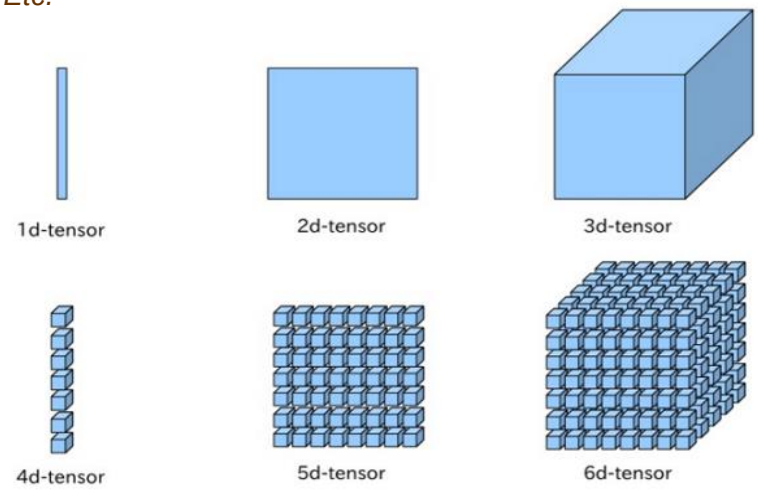
Some common tensor representations:

- *Vectors: 1D — (features)*
- *Sequences: 2D — (timesteps, features)*
- *Images: 3D — (height, width, channels)*
- *Videos: 4D — (frames, height, width, channels)*

Corresponding batches of e.g. images or videos will be present by 1 more dimensional tensors:

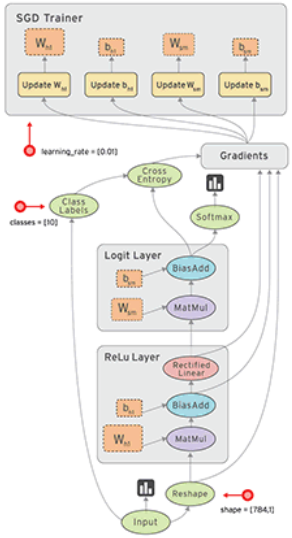
- *4D for images (samples, height, width, channels)*
- *5D for videos (samples, frames, height, width, channels)*

Tensor could be considered as n-dimensional matrix:
0-d tensor: scalar (number)
1-d tensor: vector
2-d tensor: matrix
Etc.



Relevant links:
https://medium.com/datadriveninvestor/what-is-the-tensor-in-deep-learning-77c2af7224a1

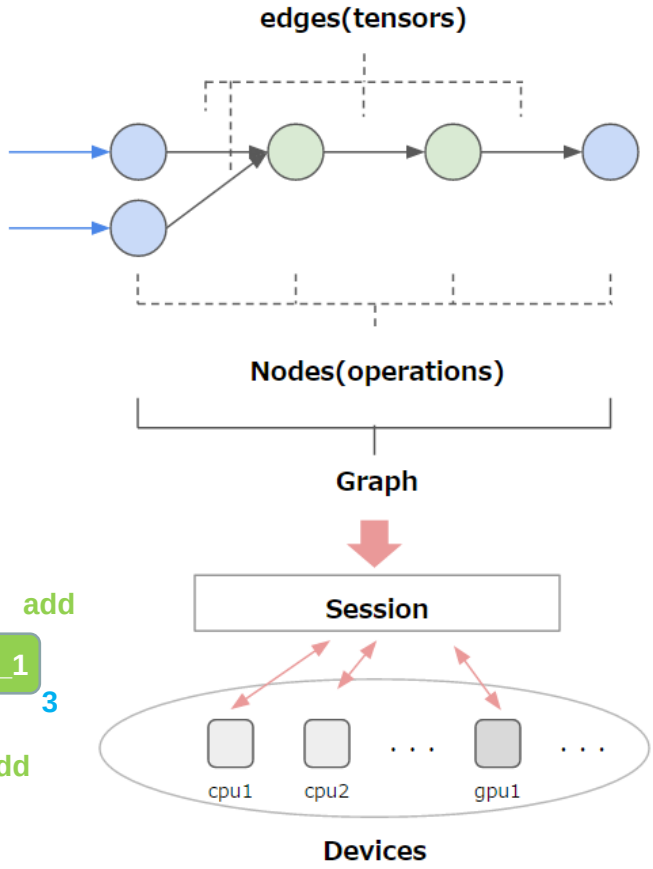
TensorFlow



Computational graph of TensorFlow

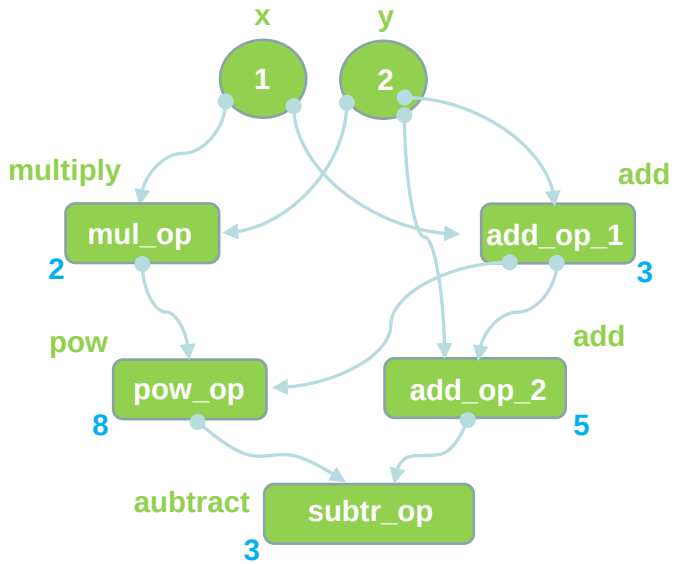
- *nodes in the graph represent mathematical operations*
- *graph edges represent the multidimensional data arrays (tensors) communicated between the nodes*
- *session executes the graph*

Splitting up complex expression into a set of smaller simple computations, we enabling parallel computation based on CPU or GPU that gives us significant gain in performance…



$z = (x * y) \wedge (x + y) - ((x + y) + y)$

a = (x * y)
b = (x + y)
c = a ^ b
d = b + y
z = c - d



TensorFlow 1.x vs 2.x

Tip: Ways to know you're using a 1.x tutorial

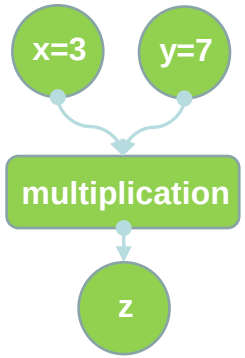
- `tf.enable_eager_execution()`
- `session.run`
- `tf.placeholder`
- `feed_dict`

Relevant links:

https://www.tensorflow.org/guide/migrate/tf1_vs_tf2
https://www.youtube.com/watch?v=lEljKc9ZtU8&list=PLOU2XLYxmsILVTiOlMJdo7RQS55jYhsMi&index=82&t=0s
https://mlfromscratch.com/tensorflow-2/#/

TensorFlow

Simple example that multiplies two numbers:



1. *Import tensorflow library*
2. *Build the graph*
3. *Create a session to execute the graph*
4. *Run the session*

```
import tensorflow as tf

x = tf.constant (3)
y = tf.constant (7)
z = tf.multiply(x,y)
sess = tf.Session()
result = sess.run(z)

print('Result: %r' %result)
sess.close()
```

```
Result: 21
```

TensorFlow 1.0

1. *Import tensorflow library*
2. *Build the graph*

```
import tensorflow as tf

x = tf.constant(3)
y = tf.constant(7)
z = tf.multiply(x,y)
tf.print(z, output_stream=sys.stderr,sep=',')
print(z)
print('Result: {}'.format(z))
```

```
21
tf.Tensor(21, shape=(), dtype=int32)
Result: 21
```

TensorFlow 2.0

TensorFlow 1.x vs 2.x

Placeholder vs Function …

*Use **Functions**, not **Sessions**. A session.run call is almost like a function call: you specify the inputs and the function to be called, and you get back a set of outputs. In TF2, you can decorate a Python function using tf.function to mark it for JIT compilation so that TensorFlow runs it as a single graph (Functions 2.0 RFC).*

```
# TF1.x
outputs = session.run(f(placeholder), feed_dict={placeholder: input})
# TF2
outputs = f(input)
```

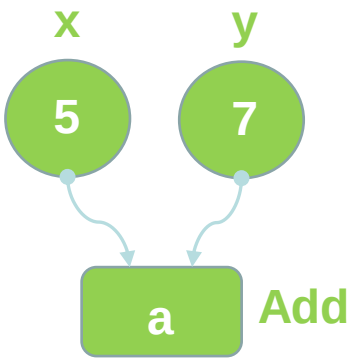

```
import tensorflow as tf

# a placeholder of type float 32-bit, shape is a vector of 5 elements
a = tf.placeholder(tf.float32, shape=[5])
# a constant of type float 32-bit, shape is a vector of 5 elements
b = tf.constant([1, 2, 3, 4, 5], tf.float32)

c = a + b # short for tf.add(a, b)

with tf.Session() as sess:
    print (sess.run(c, {a: [1, 1, 1, 1, 1]}))
```


```
import tensorflow as tf

# a constant of type float 32-bit, shape is a vector of 5 elements
b = tf.constant([1, 2, 3, 4, 5], tf.float32)

@tf.function
def add(a):
    return a + b

c = add([1, 1, 1, 1, 1])
print(c.numpy())
```

Relevant links:
https://www.tensorflow.org/guide/migrate/tf1_vs_tf2

Data Flow Graphs

```
import tensorflow as tf

a = tf.add(5, 7)
print (a)
```



Result:

```
Tensor("Add:0", shape=(), dtype=int32)
```

- *TF automatically names the nodes when you don't explicitly name them. Therefore x=5, y=7.*
- *The result is not "12" as could be expected since we did not actually execute our computational graph yet…*

tf.Session

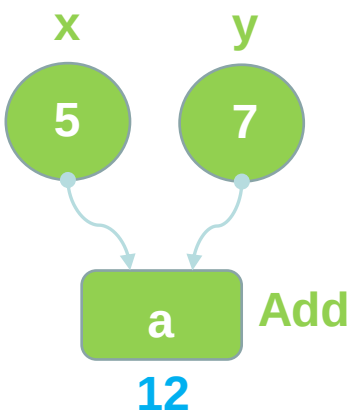
*A **Session** object encapsulates the environment in which Operation objects are executed, and Tensor objects are evaluated.*

- *Create a session, assigning it to variable **sess** to be called later*
- *Evaluate the graph within the session to fetch the values*

```
import tensorflow as tf

a = tf.add(5, 7)
sess = tf.Session()
print (sess.run(a))
sess.close()
```

Result:

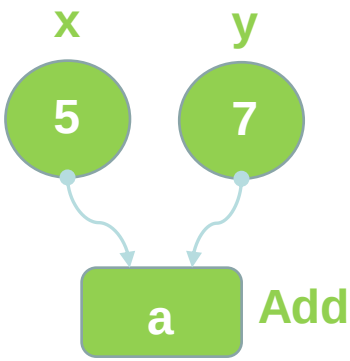
```
12
```



- *You may use alternative session handling*

```
import tensorflow as tf

a = tf.add(5, 7)
with tf.Session() as sess:
    print (sess.run(a))
```

Data Flow Graphs




```
import tensorflow as tf

a = tf.add(5, 7)
print (a)
```

Result: *tf.Tensor(12, shape=(), dtype=int32)*

```
print(a.numpy())
```

Result: *12*

Will be deprecated in the future…

TensorFlow 1.0

```
import tensorflow as tf
tf.compat.v1.disable_v2_behavior()

a = tf.add(5, 7)
print (a)
```

Result: *Tensor("Add_2:0", shape=(), dtype=int32)*

```
sess = tf.compat.v1.Session()
print(sess.run(a))
sess.close()
```

Result: *12*

```
import tensorflow as tf
tf.compat.v1.disable_eager_execution()

a = tf.add(5, 7)
print (a)
```

Result: *Tensor("Add_2:0", shape=(), dtype=int32)*

```
sess = tf.Session()
print(sess.run(a))
sess.close()
```

Result: *12*

Subgraphs

```
import tensorflow as tf

x = 1
y = 2
add_op = tf.add(x, y)
mul_op = tf.multiply(x, y)
add_op_2 = tf.add(add_op, y)
pow_op = tf.pow(mul_op, add_op)

with tf.Session() as sess:
    res = sess.run(pow_op)
    print ('Result: %r' %res)
```

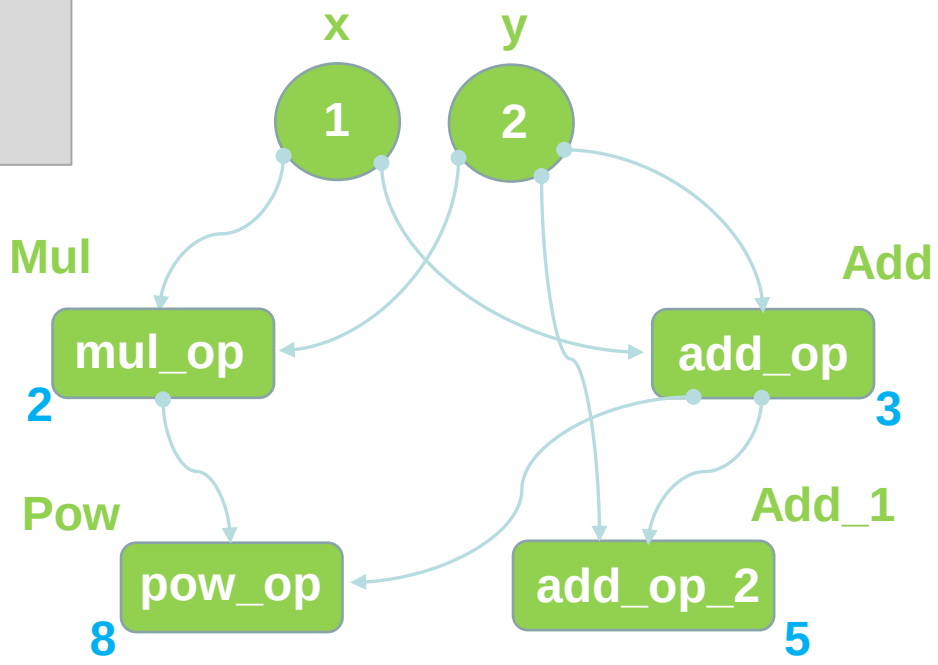
Result:

```
Result: 8
```

*Since we only need to know the value of **pow_op** operation, session would not compute values of any unnecessary for this goal operations (e.g. add_op_2)*

***Simply**:*

1. *Import tensorflow library*
2. *Build the graph*
3. *Create a session to execute the graph*
4. *Run the session*



Subgraphs

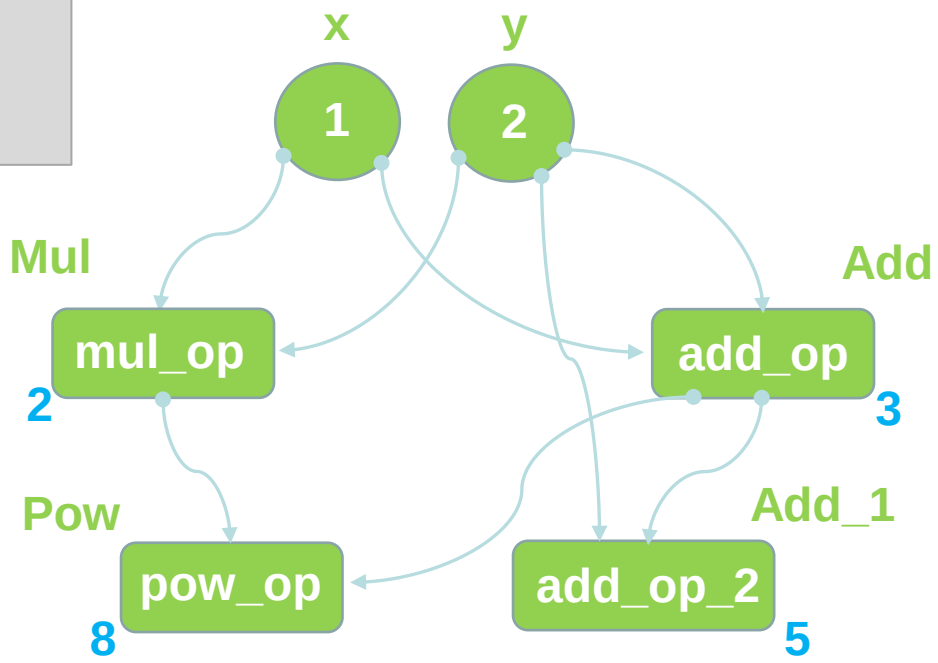
```
import tensorflow as tf

x = 1
y = 2
add_op = tf.add(x, y)
mul_op = tf.multiply(x, y)
add_op_2 = tf.add(add_op, y)
pow_op = tf.pow(mul_op, add_op)

with tf.Session() as sess:
    res1, res2 = sess.run([pow_op, add_op_2])
    print ('Results: {}, {}'.format(res1,res2))
```

Result:

```
Results: 8, 5
```

If you need values of more than one variable, pass all of them in fetches…

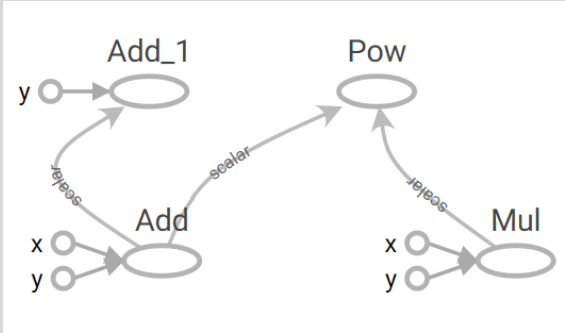
```
tf.Session.run(
    fetches,
    feed_dict=None,
    options=None,
    run_metadata=None
)
```



TensorBoard

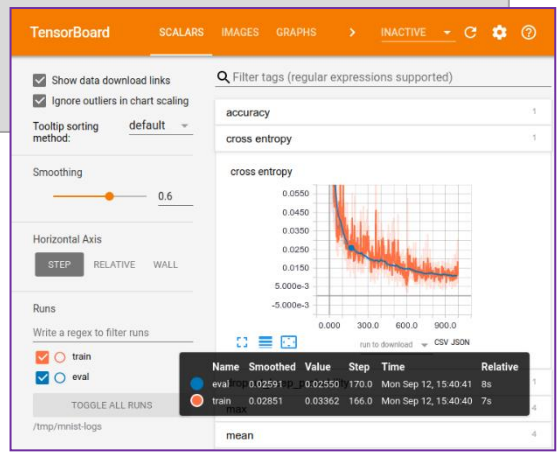
TensorFlow 1.0

```
import tensorflow as tf

x = 1
y = 2
add_op = tf.add(x, y)
mul_op = tf.multiply(x, y)
add_op_2 = tf.add(add_op, y)
pow_op = tf.pow(mul_op, add_op)

with tf.Session() as sess:
    # to use TensorBoard, create the summary writer after graph definition and before running the session
    writer = tf.summary.FileWriter('./graphs', sess.graph) # location of the event files
    res1, res2 = sess.run([pow_op, add_op_2])
    print ('Results: {}, {}'.format(res1,res2))
    writer.close() # close the writer when you're done with it
```



From the terminal:

- *Run you python file: python <filename>.py*
- *Run TensorBoard: tensorboard --logdir="./graphs" --port 6006 ,*
  or py -m tensorboard --logdir "./graphs" --port 6006
- *Open your browser and go to: http://localhost:6006/*



From Notebooks:

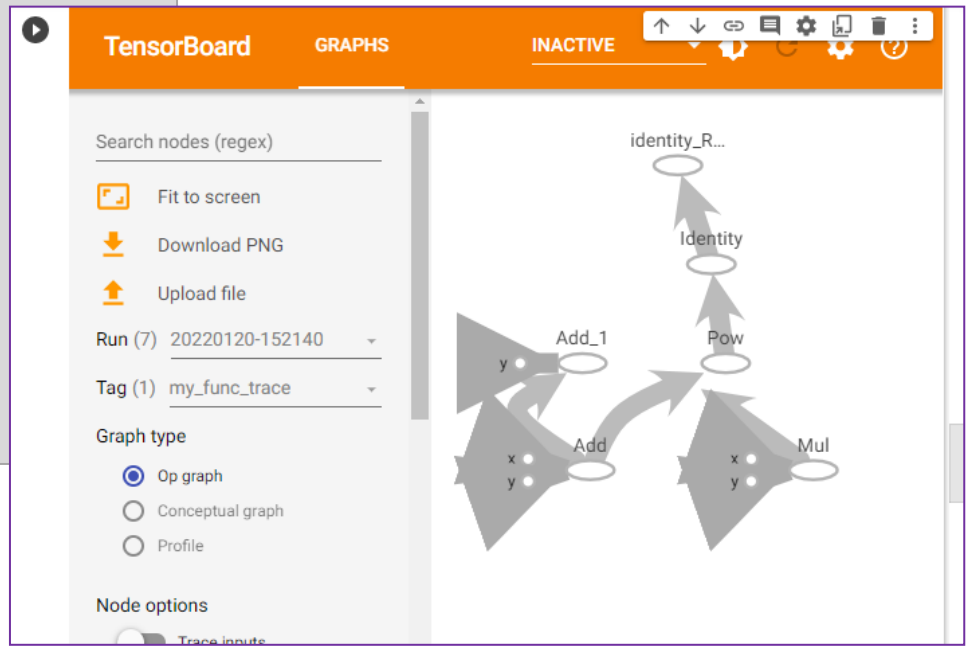
```
# Load the TensorBoard notebook extension
%load_ext tensorboard
...
%tensorboard --logdir './graphs'
```

https://colab.research.google.com/github/tensorflow/tensorboard/blob/master/docs/tensorboard_in_notebooks.ipynb#scrollTo=TB0wBWfcVqHz

TensorBoard

```
import tensorflow as tf
import tensorboard
%load_ext tensorboard
from datetime import datetime

# Set up logging.
stamp = datetime.now().strftime("%Y%m%d-%H%M%S")
logdir = 'logs/func/%s' % stamp  # name of this `run`
writer = tf.summary.create_file_writer(logdir)
# Start tracing and store it in `tf.summary`
tf.summary.trace_on(graph=True, profiler=False)

@tf.function
def pow_op(x,y):
  add_op = tf.add(x,y)
  mul_op = tf.multiply(x,y)
  add_op_2 = tf.add(add_op,y)
  pow_op = tf.pow(mul_op, add_op)
  return pow_op

print(pow_op(1,2))
print(pow_op(1,2).numpy())
```

```
with writer.as_default():
  tf.summary.trace_export(
      name="my_func_trace",  # name of tag
      step=0,
      profiler_outdir=logdir)

%tensorboard --logdir logs/func
```




```
tf.Tensor(8, shape=(), dtype=int32)
8
```

Relevant links:
https://www.tensorflow.org/tensorboard/graphs

Constants

tf.constant

creates a constant tensor...

```
import tensorflow as tf

a = tf.constant(1)
b = tf.constant(2, name=”b”)
res = tf.add(a, b, name=”myAdd”)
```

```
…
a = tf.constant([3,5], name=”a”)
b = tf.constant([[3,0],[1,5]], name=”b”)
…
```

```
tf.constant(
    value,
    dtype=None,
    shape=None,
    name='Const',
    verify_shape=False
)
```

Tensors with a specific values…

tf.zeros

creates a tensor of shape and all zeros (when ran in session)

more compact than other constants in the graph def, resulting → faster startup

```
tf.zeros([2, 3], tf.int32)  # [[0, 0, 0], [0, 0, 0]]
```

```
tf.zeros(
    shape,
    dtype=tf.float32,
    name=None
)
```

tf.zeros_like

creates a tensor of shape and type (unless type is specified) as the temp_tensor but all elements are zeros.

```
temp_tensor = tf.constant([[1, 2, 3], [4, 5, 6]])
tf.zeros_like(temp_tensor)  # [[0, 0, 0], [0, 0, 0]]
```

```
tf.zeros_like(
    tensor,
    dtype=None,
    name=None,
    optimize=True
)
```

Similarly: tf.ones *and* tf.ones_like

tf.fill

creates a tensor filled with a scalar value.

```
# Output tensor has shape [2, 3].
tf.fill([2, 3], 9) # [[9, 9, 9],[9, 9, 9]]
```

```
tf.fill(
    dims,
    value,
    name=None
)
```

Randomly generated constants…

tf.random_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None, name=None)

tf.truncated_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None, name=None)

tf.random_uniform(shape, minval=0, maxval=None, dtype=tf.float32, seed=None, name=None)

tf.random_shuffle(value, seed=None, name=None)

tf.random_crop(value, size, seed=None, name=None)

tf.multinomial(logits, num_samples, seed=None, name=None)

tf.random_gamma(shape, alpha, beta=None, dtype=tf.float32, seed=None, name=None)

tf.set_random_seed(seed)

sets the graph-level random seed…

Constants vs Variables

- *Constants are stored in the graph definition, making loading graphs expensive when constants are big.*
- *Only use constants for primitive types.*

```
import tensorflow as tf

a_1 = tf.constant([3.0, 5.0], name="my_const")
with tf.Session() as sess:
    print (sess.graph.as_graph_def())
```

Use variables or readers for more data that requires more memory…

```
# create variable var1 with scalar value
var1 = tf.Variable(7, name="scalar")

# create variable var2 as a vector
var2 = tf.Variable([3, 5], name="vector")

# create variable var3 as a 2x2 matrix
var3 = tf.Variable([[0, 1], [2, 3]], name="matrix")

# create variable W as 784 x 10 tensor, filled with zeros
W = tf.Variable(tf.zeros([784,10]))
```

Variables

tf.Variable

- *is a class. Therefore, it is not **tf.variable** similarly to **tf.constant** which is an op.*
- *tf.Variable holds several ops:*

```
x = tf.Variable(...)

x.initializer # init op
x.value() # read op
x.assign(...) # write op
x.assign_add(...) # add more
x.assign_sub(...) # subtract
```

Variables should be initialized…

- *all variables at once*

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

- *only a subset of variables*

```
init_v1_v2 = tf.variables_initializer([v1, v2], name="init_v1_v2")
with tf.Session() as sess:
    sess.run(init_v1_v2)
```

- *a single variable*

```
W = tf.Variable(tf.zeros([784,10]))
with tf.Session() as sess:
    sess.run(W.initializer)
```

Relevant links:
https://www.tensorflow.org/guide/variable
https://www.tensorflow.org/guide/migrate/tf1_vs_tf2

Variables

Eval() a variable

```
import tensorflow as tf

var1 = tf.Variable(7, name="scalar")
with tf.Session() as sess:
    sess.run(var1.initializer)
    print (var1)  # >> <tf.Variable 'scalar:0' shape=() dtype=int32_ref>
```

```
import tensorflow as tf

var1 = tf.Variable(7, name="scalar")
with tf.Session() as sess:
    sess.run(var1.initializer)
    print (var1.eval())  # >> 7
```

tf.Variable.assign()

```
var1 = tf.Variable(1)
var1.assign(10)
with tf.Session() as sess:
    sess.run(var1.initializer)
    print (var1.eval()) # >> 1
```

- *.assign() doesn't actually assign the value to variable, but just creates an assign op that should be run to take effect.*

```
var1 = tf.Variable(1)
assign_op = var1.assign(10)
with tf.Session() as sess:
    sess.run(assign_op)
    print (var1.eval()) # >> 10
```

- *assign_op does variable initialization for you. So, no need for initialization…*

Variables

assign_add() and *assign_sub()*

```
import tensorflow as tf

var1 = tf.Variable(7, name="scalar")
with tf.Session() as sess:
    sess.run(var1.initializer)
    # increment by 10
    sess.run(var1.assign_add(10)) # >> 17
    # decrement by 2
    sess.run(var1.assign_sub(2)) # >> 15
```

Each session maintains its own copy of variable…

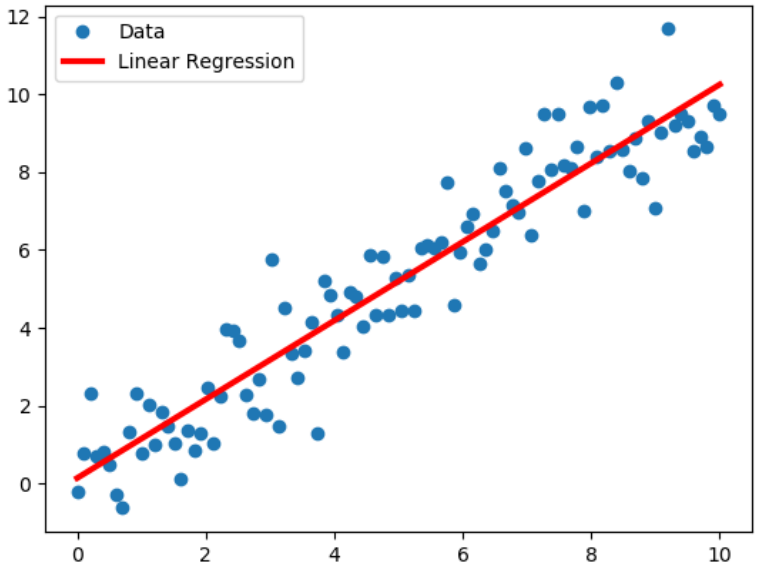
```
import tensorflow as tf

var1 = tf.Variable(7, name="scalar")
sess1 = tf.Session()
sess2 = tf.Session()
sess1.run(var1.initializer)
sess2.run(var1.initializer)
print (sess1.run(var1.assign_add(10))) # >> 17
print (sess2.run(var1.assign_sub(2))) # >> 5
sess1.close()
sess2.close()
```

Variables

```
import tensorflow as tf

v = tf.Variable(1.)
print(v)
print(v.numpy())

v.assign(2.)
print(v.numpy())

v.assign_add(2.)
print(v.numpy())

v.assign_sub(1.)
print(v.numpy())
```

Result:

```
<tf.Variable 'Variable:0' shape=() dtype=float32, numpy=1.0>
1.0
2.0
4.0
3.0
```

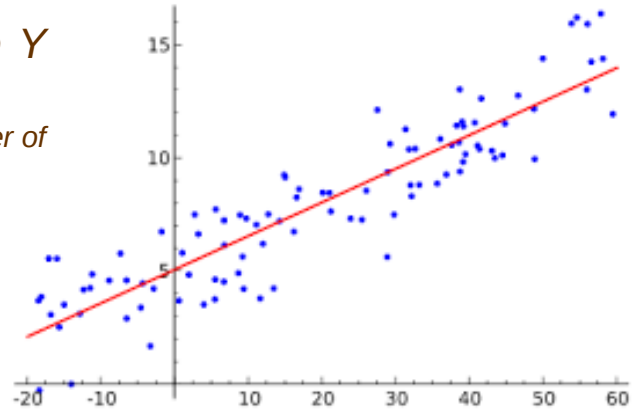


Linear Regression

Linear Regression

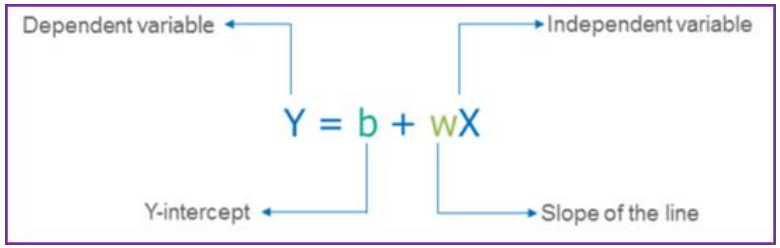
Model relationship between a scalar dependent variable Y and independent variables X.

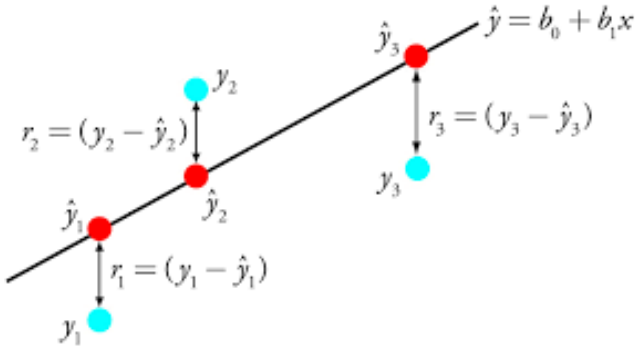
(e.g. dependence of salary level from age, or amount of theft incidents from number of fire cases, or house price from distance to city center, etc.)



Model: *Multiply each **feature** by a **weight** and add them up. Add an **intercept** to get the final estimate.*

$y = w_0x_0 + w_1x_1 + \cdots + w_nx_n + b$



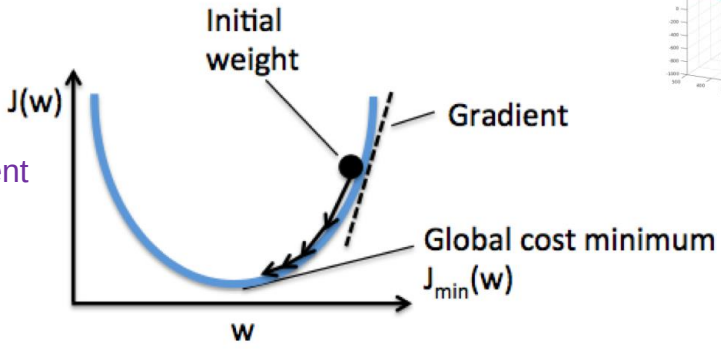


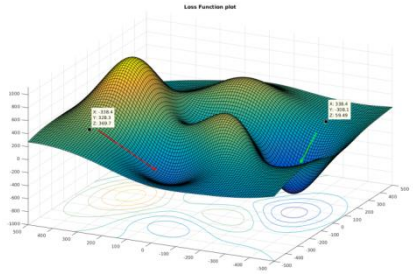
Cost function: $(Y - Y\_predicted)^2$

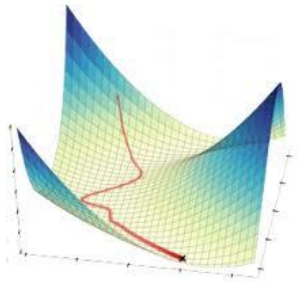
Linear Regression

***Optimization**: Holding X and Y constant, adjust **parameters** (weights and intercept) to minimize **cost**.*

Gradient descent:
https://en.wikipedia.org/wiki/Gradient_descent

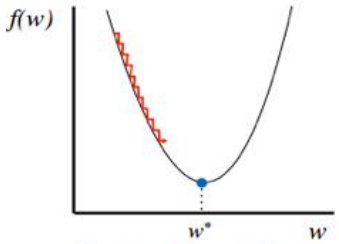






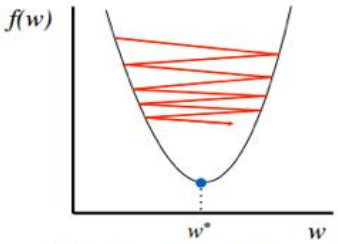
Hyper-parameter Learning rate allows us to manipulate with our learning steps…

Undershoot



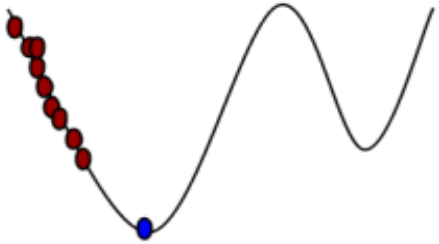
Too small: converge very slowly

Overshoot



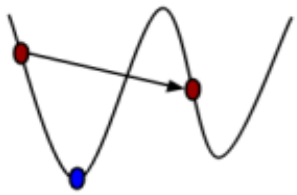
Too big: overshoot and even diverge

Undershoot



very small learning rate needs lots of steps

Overshoot

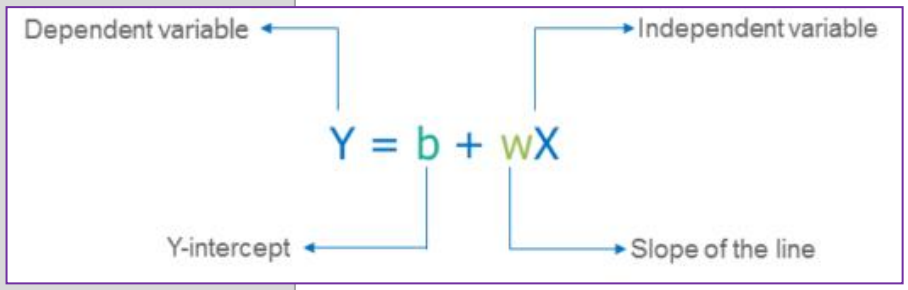


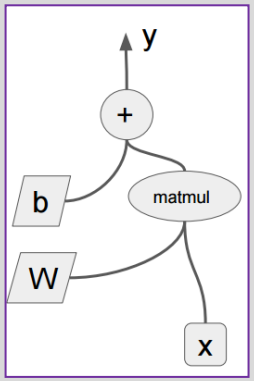
too big learning rate: missed the minimum

Linear Regression

Simple Linear Regressor

```
import tensorflow as tf

# Model parameters
W = tf.Variable([.3], dtype=tf.float32)
b = tf.Variable([-.3], dtype=tf.float32)
# Model input and output
x = tf.placeholder(tf.float32)
linear_model = W*x + b
y = tf.placeholder(tf.float32)
# loss
loss = tf.reduce_sum(tf.square(linear_model - y)) # sum of the squares
# optimizer
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
# training data
x_train = [1, 2, 3, 4]
y_train = [0, -1, -2, -3]
# training loop
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init) # reset values to wrong
for i in range(1000):
    sess.run(train, {x: x_train, y: y_train})
# evaluate training accuracy
curr_W, curr_b, curr_loss = sess.run([W, b, loss], {x: x_train, y: y_train})
print("W: %s b: %s loss: %s"%(curr_W, curr_b, curr_loss))
# predict new values
x_predict = [-1, 0, 1, 2]
predicted_values = [(W*x + b).eval(session=sess) for x in x_predict]
print(predicted_values)
```





List of optimizers in TF:

tf.train.GradientDescentOptimizer

tf.train.AdagradOptimizer

tf.train.MomentumOptimizer

tf.train.AdamOptimizer

tf.train.ProximalGradientDescentOptimizer

tf.train.ProximalAdagradOptimizer

tf.train.RMSPropOptimizer

…

Linear Regression generalized

TensorFlow 1.0

$$y = w_0x_0 + w_1x_1 + \cdots + w_nx_n + b$$

```
…
N = 3          # is number of variables
# Placeholders
X = tf.placeholder(tf.float32, [None, N])
Y = tf.placeholder(tf.float32, [None, 1])
# Parameters/Variables
W = tf.get_variable(”weights”, [N, 1],
          initializer=tf.random_normal_initializer())
b = tf.get_variable(”intercept”, [1],
          initializer=tf.constant_initializer(0))
learning_rate = 0.01
NUM_EPOCHS = 10
BATCH_SIZE = 50
X_train, Y_train, X_test, Y_test = getData() # function should be implemented
# Model
Y_model = tf.matmul(X, W) + b
# Cost function (loss)
loss = tf.reduce_mean(tf.square(Y_model - Y)) # mean(avarage) of the squares
# Optimization
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)
# ----------------------
with tf.Session() as sess:
    # initialize variables
    sess.run(tf.global_variables_initializer())
    # Train
    for _ in range(NUM_EPOCHS):
        for X_batch, Y_batch in get_minibatches(X_train, Y_train, BATCH_SIZE): # function should be implemented
            sess.run(optimizer, feed_dict={X: X_batch, Y: Y_batch})
    # Test
    Y_predicted = sess.run(Y_model, feed_dict = {X: X_test})
    squared_err = tf.reduce_mean(tf.square(Y_test - Y_predicted))
```

Google Colaboratory

Access files from Google Drive…

```
[1]  from google.colab import drive
     drive.mount("/content/gdrive")
     Mounted at /content/gdrive

     base_path = '/content/gdrive/My Drive/TIES4911_2022'
```

*/content/gdrive/My Drive/**< path to your file >***

Linear Regression

Dataset:
https://github.com/Avik-Jain/100-Days-Of-ML-Code/blob/master/datasets/studentscores.csv

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow as tf
import matplotlib.pyplot as plt
%matplotlib notebook      # use "%matplotlib inline" instead when do in Google Colab)

# Linear regressor
class SimpleLinearRegression:
    def __init__(self, initializer='random'):
        if initializer=='ones':
            self.var = 1.
        elif initializer=='zeros':
            self.var = 0.
        elif initializer=='random':
            self.var = tf.random.uniform(shape=[], minval=0., maxval=1.)

        self.m = tf.Variable(1., shape=tf.TensorShape(None))
        self.b = tf.Variable(self.var)

    def predict(self, x):
        return tf.reduce_sum(self.m * x, 1) + self.b

    def mse(self, true, predicted):
        return tf.reduce_mean(tf.square(true-predicted))
```

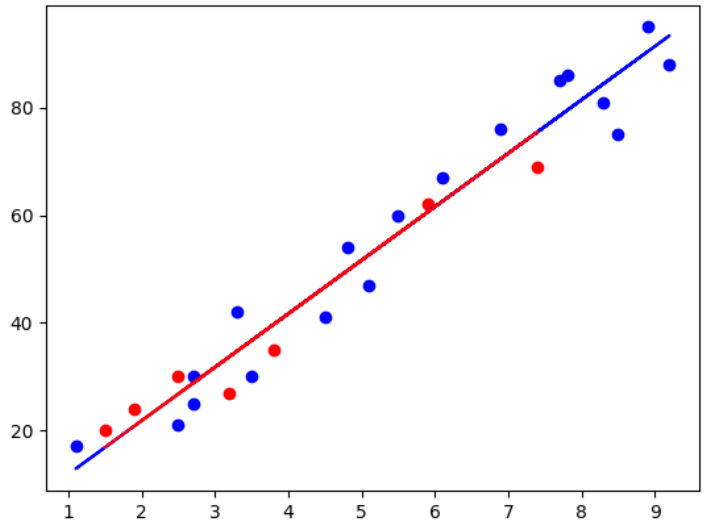
```
    def update(self, X, y, learning_rate):
        with tf.GradientTape(persistent=True) as g:
            loss = self.mse(y, self.predict(X))

        print("Loss: ", loss)

        dy_dm = g.gradient(loss, self.m)
        dy_db = g.gradient(loss, self.b)

        self.m.assign_sub(learning_rate * dy_dm)
        self.b.assign_sub(learning_rate * dy_db)

    def train(self, X, y, learning_rate=0.01, epochs=5):

        if len(X.shape)==1:
            X=tf.reshape(X,[X.shape[0],1])

        self.m.assign([self.var]*X.shape[-1])

        for i in range(epochs):
            print("Epoch: ", i)

            self.update(X, y, learning_rate)
```

Source: https://towardsdatascience.com/linear-regression-from-scratch-with-tensorflow-2-part-1-3e2443804df0

Linear Regression

```
# Data Preprocessing
dataset = pd.read_csv('studentscores.csv')
X = dataset.iloc[ : , : 1 ].values
Y = dataset.iloc[ : , 1 ].values

# split the data
x_train, x_test, y_train, y_test  = train_test_split( X, Y, test_size = 1/4,
random_state = 0)

# standardize the data
mean_label = y_train.mean(axis=0)
std_label = y_train.std(axis=0)
mean_feat = x_train.mean(axis=0)
std_feat = x_train.std(axis=0)
x_train_norm = (x_train-mean_feat)/std_feat
y_train_norm = (y_train-mean_label)/std_label

# Create and train a regressor
linear_model = SimpleLinearRegression('zeros')
linear_model.train(x_train_norm, y_train_norm, learning_rate=0.01,
epochs=10000)

# standardize
x_test_norm = (x_test-mean_feat)/std_feat
# prediction on test data
test_pred = linear_model.predict(x_test_norm)
# reverse standardization
test_pred *= std_label
test_pred += mean_label
```

```
# prediction on train data
train_pred = linear_model.predict(x_train_norm)
# reverse standardization
train_pred *= std_label
train_pred += mean_label

#Visualization of Training results
plt.scatter(x_train , y_train, color = 'blue')
plt.plot(x_train , train_pred, color ='blue')

#Visualization of Test results
plt.scatter(x_test , y_test, color = 'red')
plt.plot(x_test , test_pred, color ='red')
```



Source: https://towardsdatascience.com/linear-regression-from-scratch-with-tensorflow-2-part-1-3e2443804df0

scikit-learn
machine learning in Python

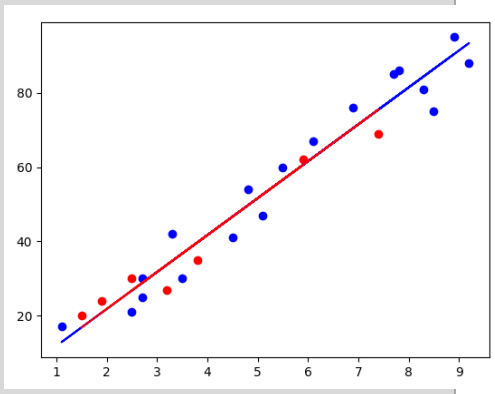
scikit-learn *- Machine Learning in Python*

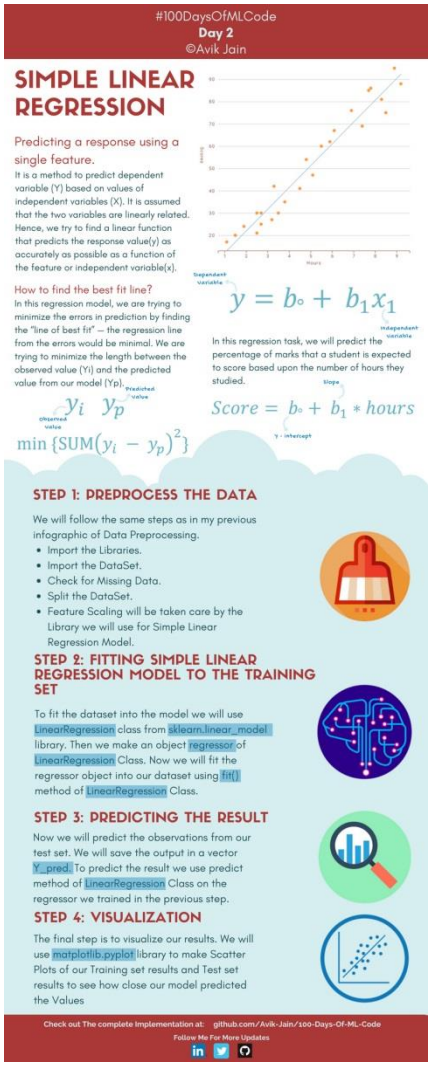
https://scikit-learn.org/stable/

Dataset:

https://github.com/Avik-Jain/100-Days-Of-ML-Code/blob/master/datasets/studentscores.csv

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# from sklearn.cross_validation import train_test_split   ### deprecated in new sklearn version
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Data Preprocessing
dataset = pd.read_csv('studentscores.csv')
X = dataset.iloc[ : , : 1 ].values
Y = dataset.iloc[ : , 1 ].values
X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size = 1/4, random_state = 0)

# Fitting Simple Linear Regression Model to the training set
regressor = LinearRegression()
regressor = regressor.fit(X_train, Y_train)

# Predicting the Result
Y_pred = regressor.predict(X_test)

#Visualization of Training results
plt.scatter(X_train , Y_train, color = 'red')
plt.plot(X_train , regressor.predict(X_train), color ='blue')

#Visualization of Test results
plt.scatter(X_test , Y_test, color = 'red')
plt.plot(X_test , regressor.predict(X_test), color ='blue')
```





https://github.com/Avik-Jain/100-Days-Of-ML-Code/blob/master/Code/Day2_Simple_Linear_Regression.md

Linear Regression

Boston house price *- The dataset describes 13 numerical properties of houses in Boston suburbs and is concerned with modeling the price of houses in those suburbs in thousands of dollars.*

Dataset: https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.data *(save the content as housing.csv file)*

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow as tf
import matplotlib.pyplot as plt
%matplotlib notebook     # use ”%matplotlib inline” instead when do in Google Colab)

# Linear regressor
class SimpleLinearRegression:
    def __init__(self, initializer='random'):
        if initializer=='ones':
            self.var = 1.
        elif initializer=='zeros':
            self.var = 0.
        elif initializer=='random':
            self.var = tf.random.uniform(shape=[], minval=0., maxval=1.)

        self.m = tf.Variable(1., shape=tf.TensorShape(None))
        self.b = tf.Variable(self.var)

    def predict(self, x):
        return tf.reduce_sum(self.m * x, 1) + self.b

    def mse(self, true, predicted):
        return tf.reduce_mean(tf.square(true-predicted))
```

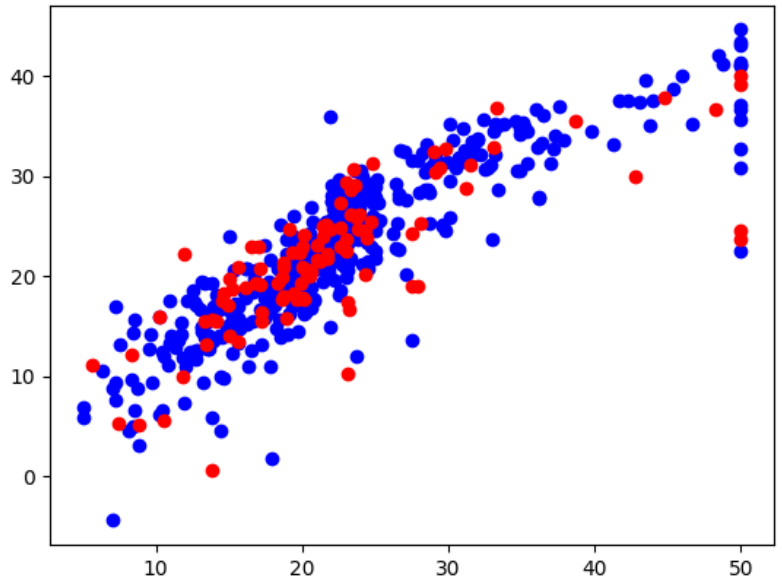
```
    def update(self, X, y, learning_rate):
        with tf.GradientTape(persistent=True) as g:
            loss = self.mse(y, self.predict(X))

        print("Loss: ", loss)

        dy_dm = g.gradient(loss, self.m)
        dy_db = g.gradient(loss, self.b)

        self.m.assign_sub(learning_rate * dy_dm)
        self.b.assign_sub(learning_rate * dy_db)

    def train(self, X, y, learning_rate=0.01, epochs=5):

        if len(X.shape)==1:
            X=tf.reshape(X,[X.shape[0],1])

        self.m.assign([self.var]*X.shape[-1])

        for i in range(epochs):
            print("Epoch: ", i)

            self.update(X, y, learning_rate)
```

Source: https://towardsdatascience.com/linear-regression-from-scratch-with-tensorflow-2-part-1-3e2443804df0

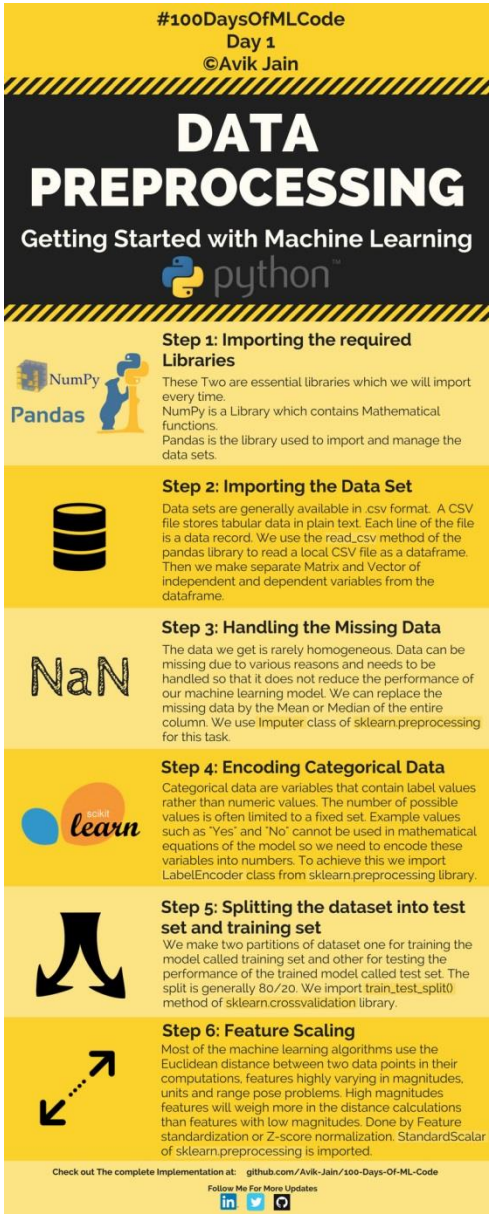
Linear Regression

```
# data preprocessing
dataframe = pd.read_csv("housing.csv", delim_whitespace=True,
header=None)
dataset = dataframe.values
X = dataset[:,0:13]
Y = dataset[:,13]

# split the data
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2,
random_state = 0)

# standardize the data
mean_label = y_train.mean(axis=0)
std_label = y_train.std(axis=0)
mean_feat = x_train.mean(axis=0)
std_feat = x_train.std(axis=0)
x_train_norm = (x_train-mean_feat)/std_feat
y_train_norm = (y_train-mean_label)/std_label
# Create and train a regressor
linear_model = SimpleLinearRegression('zeros')
linear_model.train(x_train_norm, y_train_norm, learning_rate=0.01,
epochs=10000)

# standardize
x_test_norm = (x_test-mean_feat)/std_feat
# prediction on test data
test_pred = linear_model.predict(x_test_norm)
# reverse standardization
test_pred *= std_label
test_pred += mean_label
```

```
# prediction on train data
train_pred = linear_model.predict(x_train_norm)
# reverse standardization
train_pred *= std_label
train_pred += mean_label

# visualize prediction…
plt.scatter(y_train , train_pred, color = 'blue')
plt.scatter(y_test , test_pred, color = 'red')
```



Source: https://towardsdatascience.com/linear-regression-from-scratch-with-tensorflow-2-part-1-3e2443804df0



Data PreProcessing

Dataset:
https://github.com/Avik-Jain/100-Days-Of-ML-Code/blob/master/datasets/Data.csv

```
# import the libraries
import numpy as np
import pandas as pd
#from sklearn.preprocessing import Imputer ### deprecated in new sklearn version
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
# from sklearn.cross_validation import train_test_split   ### deprecated in new sklearn version
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler  # see also MinMaxScaler
# importing dataset
dataset = pd.read_csv('Data.csv')
X = dataset.iloc[ : , :-1].values
Y = dataset.iloc[ : , 3].values
# handling the missing data
imputer = SimpleImputer(missing_values = np.nan, strategy = "mean")
imputer = imputer.fit(X[ : , 1:3])
X[ : , 1:3] = imputer.transform(X[ : , 1:3])
# encoding categorical data
labelencoder_X = LabelEncoder()
X[ : , 0] = labelencoder_X.fit_transform(X[ : , 0])
ct = ColumnTransformer([("Country", OneHotEncoder(), [1])], remainder = 'passthrough')
X = ct.fit_transform(X).toarray()
labelencoder_Y = LabelEncoder()
Y = labelencoder_Y.fit_transform(Y)
# splitting the datasets into training and test sets
X_train, X_test, Y_train, Y_test = train_test_split( X , Y , test_size = 0.2, random_state = 0)
# feature scaling
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.fit_transform(X_test)
# use <scaler>.inverse_transform() for backward transformation of the predicted results (if needed)
```

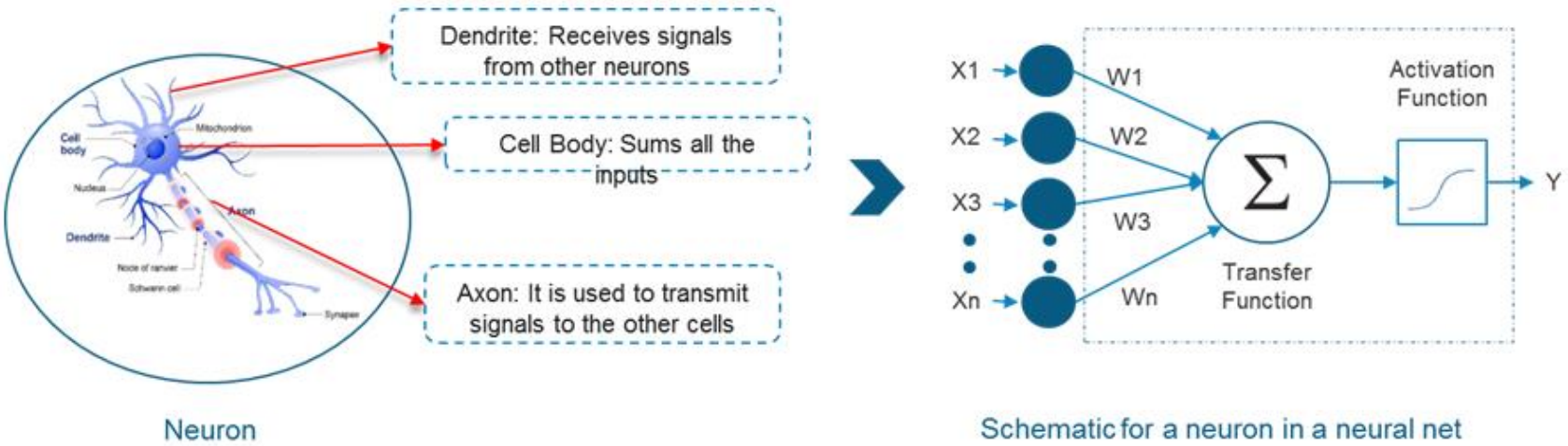
https://github.com/Avik-Jain/100-Days-Of-ML-Code/blob/master/Code/Day%201_Data%20PreProcessing.md

Linear Regression ? Neural Network

Linear Regression is a simplest Neural Network…

Aiming at re-engineering a human brain, ***Deep Learning studies*** *the basic unit of a brain - a brain cell or a* ***neuron****…*

Perceptron – is an artificial neuron developed in analogy to a biological neuron.



Neuron

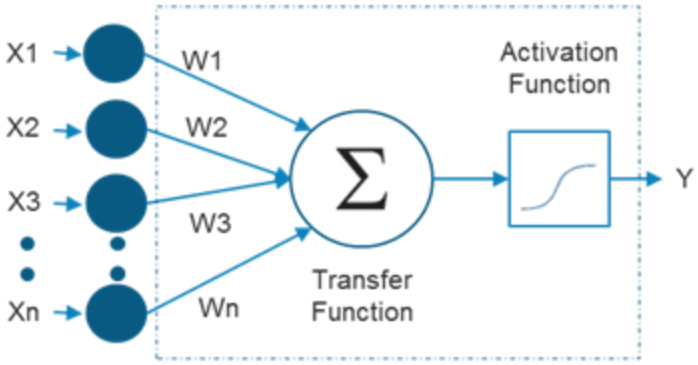
Schematic for a neuron in a neural net

Relevant links:
https://www.edureka.co/blog/deep-learning-tutorial

Activation function

Been fed with a lot of information simultaneously our brain tries to distinguish and classify the information between useful and not-so-useful…

Activation functions help the network do the segregation. They basically decide whether a neuron should be activated or not. They help the network use the useful information and suppress the irrelevant data points.



- The neurons do a linear transformation on the input by the weights and biases.
- The non linear transformation is done by the activation function.

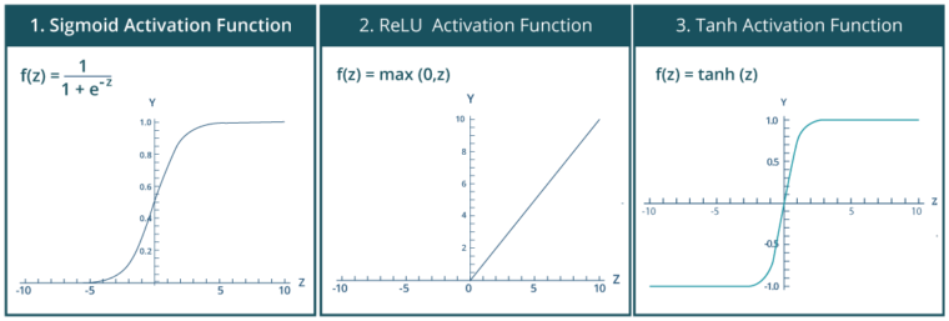
$$Y = \text{Activation}(\Sigma(weight * input) + bias)$$

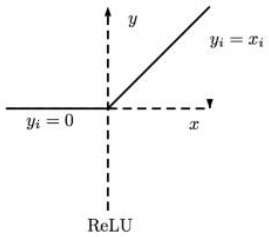
- Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases. Without the differentiable non linear function, this would not be possible.

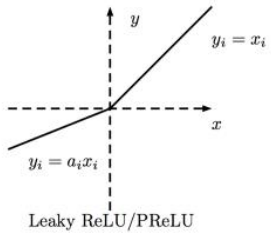
Relevant links:
https://en.wikipedia.org/wiki/Activation_function
http://dataaspirant.com/2017/03/07/difference-between-softmax-function-and-sigmoid-function/
https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/

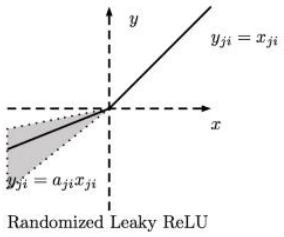
Activation function

- ***Binary Step*** *and* ***Linear Functions***
  - *the derivatives are "0" and "constant", meaning that every time we do a back propagation, the gradient would be the same.*
- ***Sigmoid*** *or* ***Logistic***
  - *isn't zero centered that makes optimization harder*
  - *vanishing gradient problem*
- ***Tanh*** *– Hyperbolic tangent*
  - *vanishing gradient problem*
- ***ReLU*** *– Rectified linear units*
  - *it should only be used within Hidden layers (output layer should use* ***Softmax*** *function for a Classification and simply use a* ***linear*** *function for a regression problem).*
  - *some gradients can be fragile during training and can die resulting in Dead Neurons*
- ***Leaky ReLU*** *(****Parameteric PReLU****,* ***Randomized RReLU****)*
  - *it should only be used within Hidden layers*
- ***Maxout*** *(forms both ReLu and Leaky ReLu)*









The ***Softmax*** *function is a sigmoid type of function but is handy to handle multiple classes classification problems. Sigmoid helps to classify between two classes, Softmax helps to attain the probabilities to define the class of each input.*

Use *ReLu* (applied to the hidden layers only) and, if the Model suffers form dead neurons during training, use *Leaky ReLu* or *Maxout* function.

Relevant links:

https://en.wikipedia.org/wiki/Activation_function
http://dataaspirant.com/2017/03/07/difference-between-softmax-function-and-sigmoid-function/
https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/
https://www.youtube.com/watch?v=-7scQpJT7uo
https://www.youtube.com/watch?v=s-V7gKrsels

Keras

Keras *is a high-level neural networks API, written in Python and capable of running on top of* ***TensorFlow***, ***CNTK***, *or* ***Theano***. *(https://keras.io/) (https://keras.io/getting_started/intro_to_keras_for_engineers/) (https://www.tensorflow.org/guide/keras/sequential_model)*

It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research.

- *Allows for easy and fast prototyping (through user friendliness, modularity, and extensibility).*
- *Supports both convolutional networks and recurrent networks, as well as combinations of the two.*
- *Runs seamlessly on CPU and GPU.*

Keras is compatible with: Python 2.x-3.x.

The core data structure of Keras is a ***model****, a way to organize layers. The simplest type of model is the Sequential model, a linear stack of layers. For more complex architectures, you should use the Keras Functional API, which allows to build arbitrary graphs of layers. Finally, you may use fully-customizable model subclassing to implement own custom forward-pass of the model…*

Example:

```
# example of a model definition with the sequential api
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense
# define the model
model = Sequential()
model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=1))
…
```

```
# example of a model definition with the functional api
from tensorflow.keras import Model
from tensorflow.keras import Input
from tensorflow.keras.layers import Dense
# define the layers
x_in = Input(shape=(100,))
x = Dense(64)(x_in)
x_out = Dense(1)(x)
# define the model
model = Model(inputs=x_in, outputs=x_out)
…
```

Relevant links:

https://www.tensorflow.org/guide/keras/sequential_model

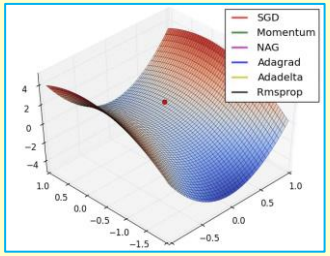
https://github.com/keras-team/keras

https://www.pyimagesearch.com/2019/10/28/3-ways-to-create-a-keras-model-with-tensorflow-2-0-sequential-functional-and-model-subclassing/

Layer
- *Dense*
- *Convolutional: Conv1D, Conv2D, Conv3D*
- *Pooling: …*
- *Recurrent: RNN, GRU, LSTM*
- *Flatten*
- *Activation*
- *Dropout*

Losses
- *mean_squared_error*
- *mean_squared_error(y_true, y_pred)*
- *mean_absolute_error(y_true, y_pred)*
- *mean_absolute_percentage_error(y_true, y_pred)*
- *mean_squared_logarithmic_error(y_true, y_pred)*
- *categorical_crossentropy(y_true, y_pred)*

Optimizers
- *SGD()*
- *RMSprop()*
- *Adagrad()*
- *Adadelta()*
- *Adam()*



Metrics
- *accuracy*
- *binary_accuracy*
- *binary_accuracy(y_true, y_pred)*
- *categorical_accuracy(y_true, y_pred)*
- *top_k_categorical_accuracy(y_true, y_pred, k=5)*

Activations
- *softmax*
- *relu*
- *tanh*
- *sigmoid*
- *hard_sigmoid*
- *Linear*

Keras

Model compilation

- ***Loss function.*** *This is the objective that the model will try to minimize. It can be the string identifier of an existing loss function (such as categorical_crossentropy or mse), or it can be an objective function.*
- ***Optimizer.*** *This could be the string identifier of an existing optimizer (such as rmsprop or adagrad), or an instance of the Optimizer class.*
- ***List of metrics***

```
model.compile(loss='mean_squared_error', optimizer='sgd',
              metrics=[metrics.mae, metrics.categorical_accuracy])
```

```
model.compile(loss='categorical_crossentropy', optimizer='sgd',
              metrics=['accuracy'])
```

```
model.compile(loss=keras.losses.categorical_crossentropy,
        optimizer=keras.optimizers.SGD(lr=0.01, momentum=0.9, nesterov=True),
        metrics=[keras.metrics.SparseCategoricalAccuracy(name="acc")])
```

Model training

*Keras models are trained on Numpy arrays of input data and labels, typically using **fit** function.*

```
history = model.fit(X_train, Y_train, batch_size=128, epochs=8)
```

Model evaluation

```
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)
```

Model prediction

```
prediction = model.predict(x_test, batch_size=128)
```

Regression problem

Boston house price *- The dataset describes 13 numerical properties of houses in Boston suburbs and is concerned with modeling the price of houses in those suburbs in thousands of dollars.*

Dataset: https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.data *(save the content as housing.csv file)*

```
import pandas as pd
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split

# data preprocessing
dataframe = pd.read_csv("housing.csv", delim_whitespace=True, header=None)
dataset = dataframe.values
X = dataset[:,0:13]
Y = dataset[:,13]
# create model
model = Sequential()
model.add(Dense(1, input_dim=13, activation='relu'))
# you may play with different network architectures (adding new layers, and changing their amount of nodes)

# compile model
model.compile(loss='mean_squared_error', optimizer='adam', metrics = ['accuracy'])
# split the data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 0)
# fit the model
model.fit(X_train, Y_train, epochs=100, verbose=0)
# evaluate the model
mse_value, acc = model.evaluate(X_test, Y_test, verbose=0)
print('Evaluation MSE: '+str(mse_value))
# predict the value
Y_pred = model.predict(X_test[0].reshape(1, -1))
print(Y_pred[0][0])
```

Source: https://machinelearningmastery.com/regression-tutorial-keras-deep-learning-library-python/
https://www.tensorflow.org/tutorials/keras/regression

Public datasets…

- ***Kaggle*** *Datasets (https://www.kaggle.com/datasets)*
- ***Hugging Face*** *Datasets (https://huggingface.co/docs/hub/datasets-overview)*
- ***UCI Machine Learning Repository*** *Datasets (https://archive.ics.uci.edu/)*
- ***Google Dataset Search*** *(https://datasetsearch.research.google.com/)*

More datasets:

- http://promise.site.uottawa.ca/SERepository/datasets-page.html

Relevant links:

- https://medium.com/datadriveninvestor/the-50-best-public-datasets-for-machine-learning-d80e9f030279
- https://en.wikipedia.org/wiki/List_of_datasets_for_machine_learning_research