UNIVERSITY OF JYVÄSKYLÄ

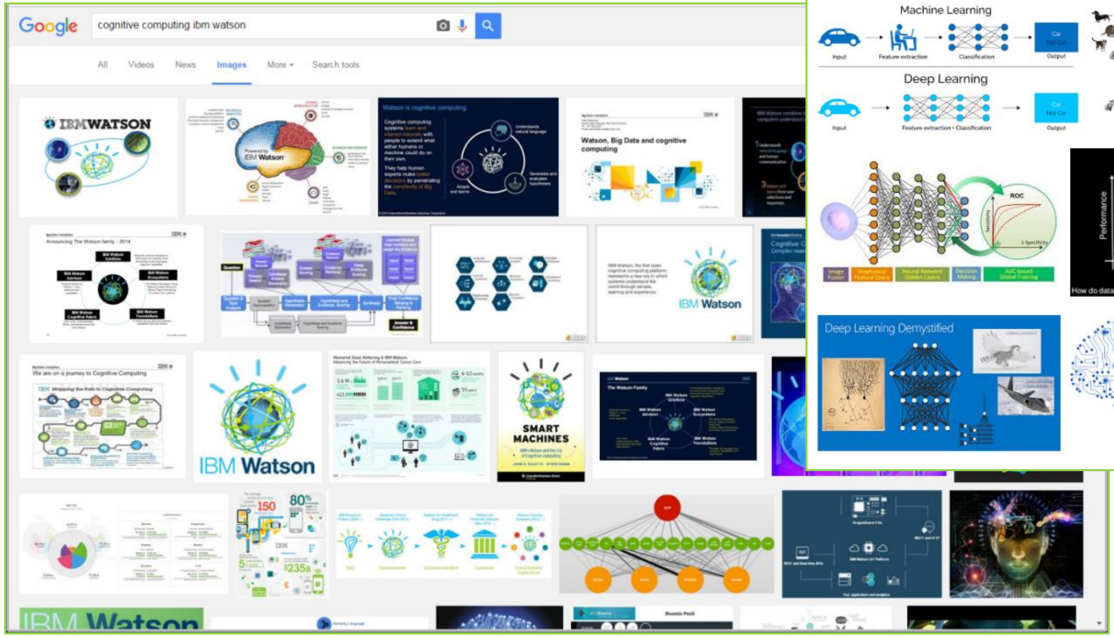
Lecture 2: Introduction to Neural Networks with TensorFlow *(Classification)*

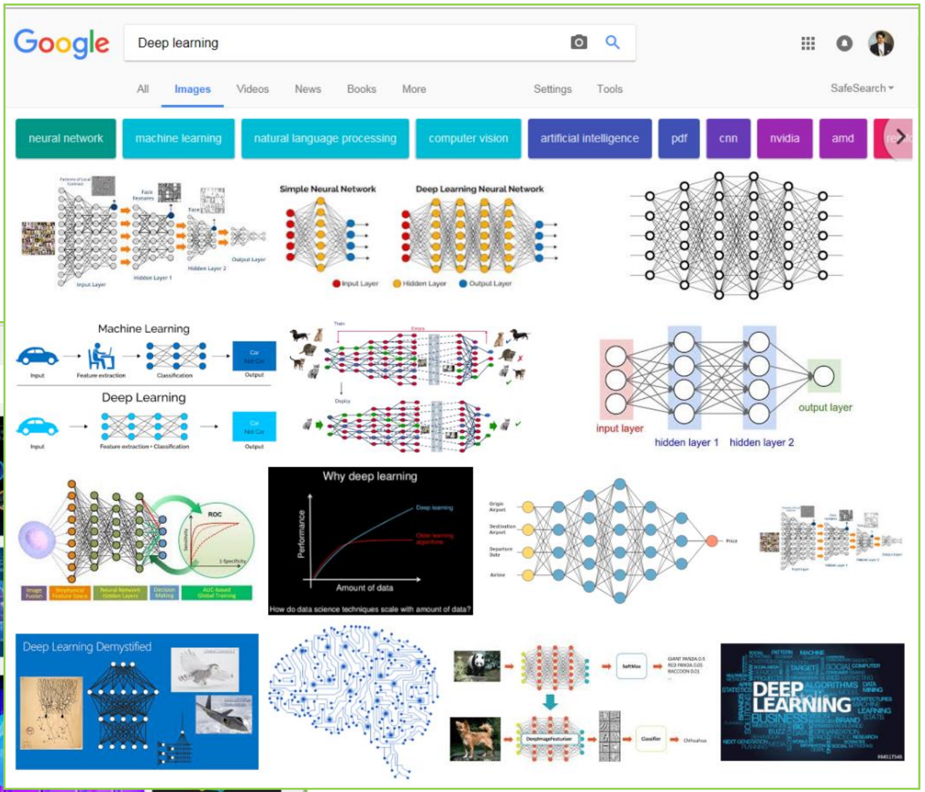
TIES4911 Deep-Learning for Cognitive Computing for Developers
Spring 2023

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.



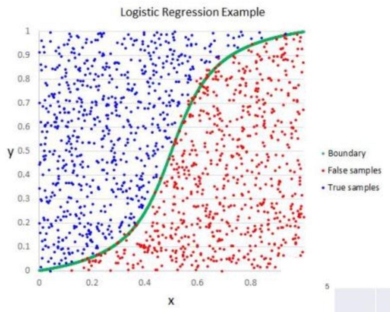


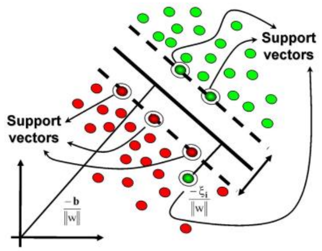
Classification

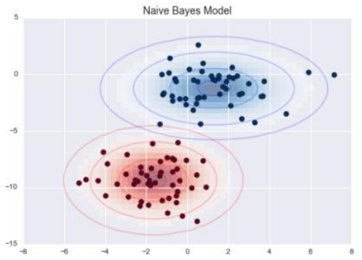
Classification – is the process of categorizing a group of objects…

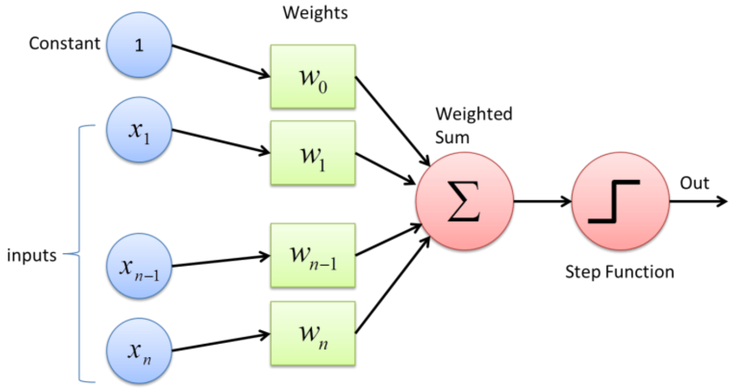
Existing classifiers are:
- Logistic Regression,
- Support Vector Machines (SVM),
- Decision Tree,
- Naive Bayes,
- Neural Networks,
- …

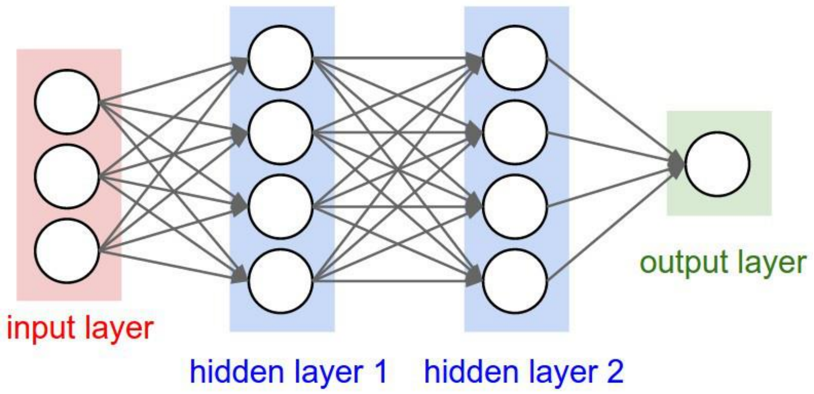
In comparison to other classifiers, when the patterns get complex, neural nets start to outperform all of their competition. The main **advantage of DNN is automated feature extraction** (those that a not specified/set explicitly).

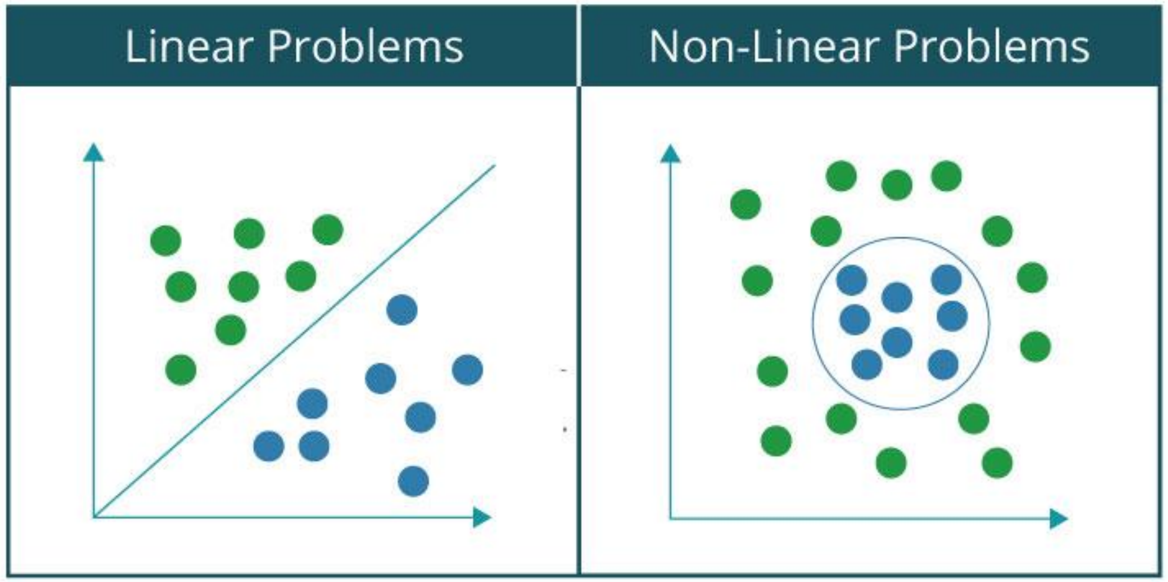






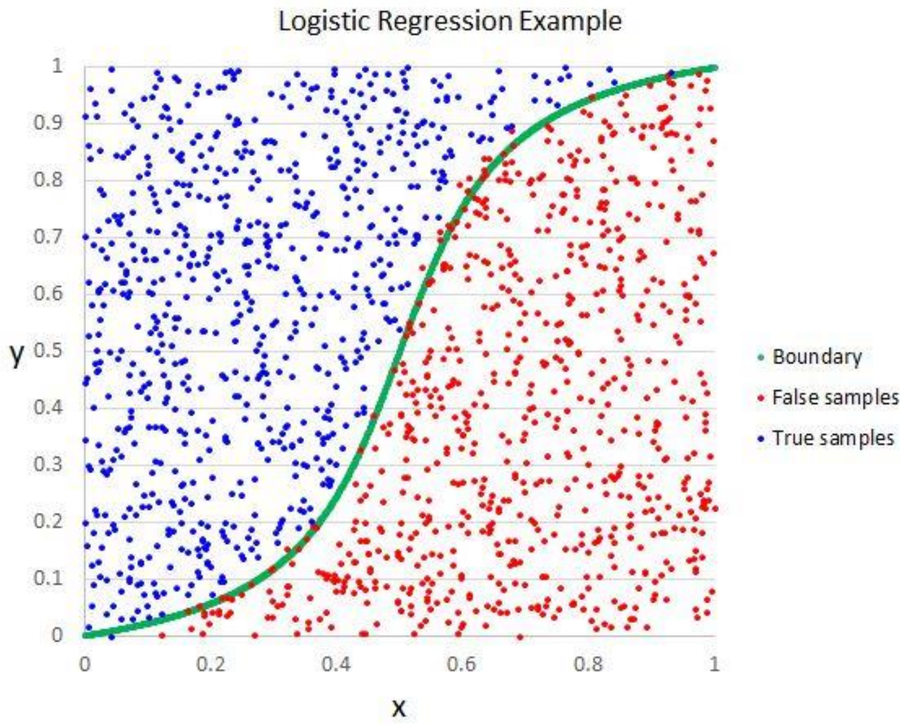






Relevant links:
https://www.edureka.co/blog/perceptron-learning-algorithm/

Logistic Regression



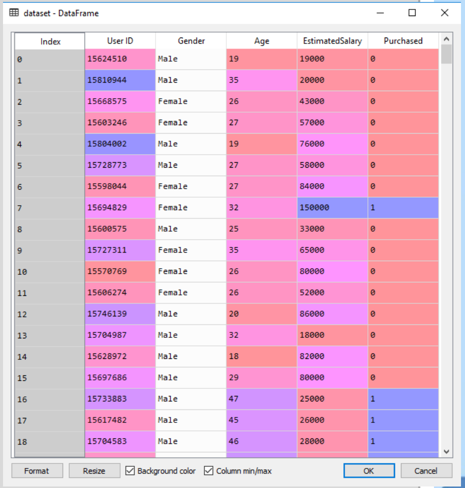
scikit-learn: Logistic Regression

scikit-learn - *Machine Learning in Python*

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

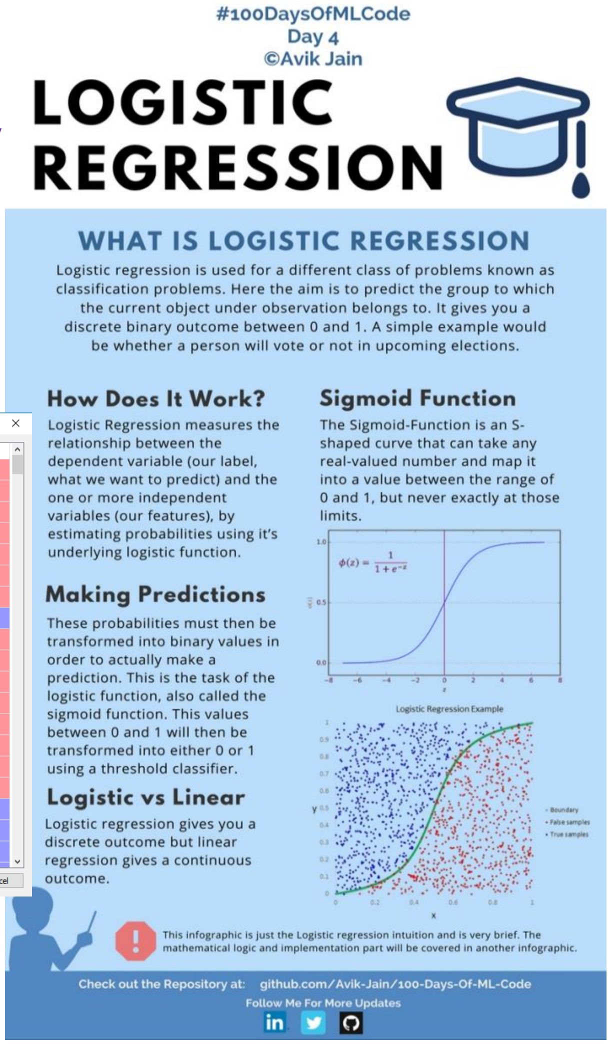
Dataset: https://github.com/Avik-Jain/100-Days-Of-ML-Code/blob/master/datasets/Social_Network_Ads.csv

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
# 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
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

# Data Preprocessing
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
Y = dataset.iloc[:, 4].values
X_train, X_test, Y_train, Y_test = train_test_split( X, Y,
                        test_size = 1/4, random_state = 0)
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fitting Logistic Regression Model to the training set
classifier = LogisticRegression()
classifier.fit(X_train, Y_train)
# Predecting the Result
Y_pred = classifier.predict(X_test)

#Evaluation
cm = confusion_matrix(Y_test, Y_pred)
```


dataset - DataFrame

Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0
15	15697686	Male	29	80000	0
16	15733883	Male	47	25000	1
17	15617482	Male	45	26000	1
18	15704583	Male	46	28000	1

Format
Resize
☑ Background color
☑ Column min/max
OK
Cancel


#100DaysOfMLCode
Day 4
©Avik Jain

LOGISTIC REGRESSION

WHAT IS LOGISTIC REGRESSION

Logistic regression is used for a different class of problems known as classification problems. Here the aim is to predict the group to which the current object under observation belongs to. It gives you a discrete binary outcome between 0 and 1. A simple example would be whether a person will vote or not in upcoming elections.

How Does It Work?

Logistic Regression measures the relationship between the dependent variable (our label, what we want to predict) and the one or more independent variables (our features), by estimating probabilities using it's underlying logistic function.

Making Predictions

These probabilities must then be transformed into binary values in order to actually make a prediction. This is the task of the logistic function, also called the sigmoid function. This values between 0 and 1 will then be transformed into either 0 or 1 using a threshold classifier.

Logistic vs Linear

Logistic regression gives you a discrete outcome but linear regression gives a continuous outcome.

Sigmoid Function

The Sigmoid-Function is an S-shaped curve that can take any real-valued number and map it into a value between the range of 0 and 1, but never exactly at those limits.

This infographic is just the Logistic regression intuition and is very brief. The mathematical logic and implementation part will be covered in another infographic.

Check out the Repository at: github.com/Avik-Jain/100-Days-Of-ML-Code

Follow Me For More Updates

https://github.com/Avik-Jain/100-Days-Of-ML-Code/blob/master/Code/Day%206%20Logistic%20Regression.md

https://www.geeksforgeeks.org/standardscaler-minmaxscaler-and-robustscaler-techniques-ml/

Linear Classifier

binary logistic regression

Perceptron as a Linear Classifier

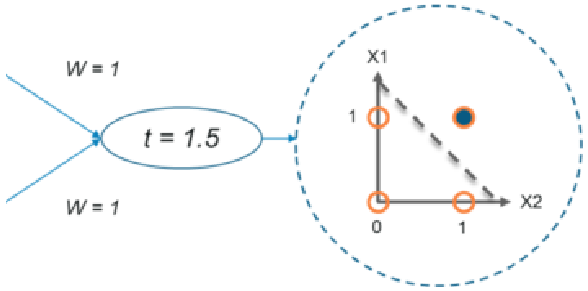
https://www.edureka.co/blog/perceptron-learning-algorithm/

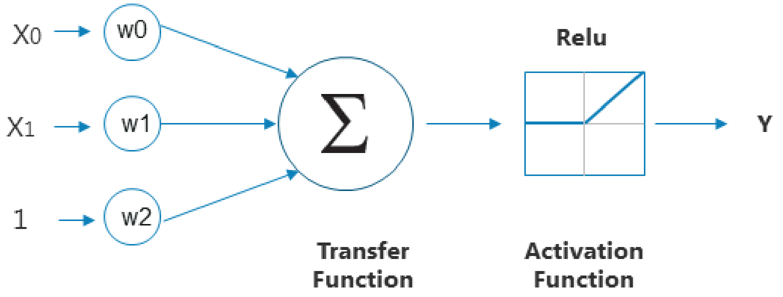
```
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()
#input1, input2 and bias
train_in = [[1., 1.,1], [1., 0,1], [0, 1.,1], [0, 0,1]]
#output
train_out = [[1.], [0], [0], [0]]
#weight variable initialized with random values using random_normal()
w = tf.Variable(tf.random_normal([3, 1], seed=12))
#Placeholder for input and output
x = tf.placeholder(tf.float32,[None,3])
y = tf.placeholder(tf.float32,[None,1])
#calculate output
output = tf.nn.relu(tf.matmul(x, w))
#Mean Squared Loss or Error
loss = tf.reduce_sum(tf.square(output - y))
#Minimize loss using GradientDescentOptimizer with a learning rate of 0.01
optimizer = tf.train.GradientDescentOptimizer(0.01)
train = optimizer.minimize(loss)
#Initialize all the global variables
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
print("w (initial): ", w.eval(session=sess))
#Compute output and cost w.r.t to input vector
for i in range(1000):
   sess.run(train, {x:train_in,y:train_out})
cost = sess.run(loss,feed_dict={x:train_in,y:train_out})
print('Epoch--',i,'--loss--',cost)
print("w (trained): ", w.eval(session=sess))
output_pred = tf.nn.relu(tf.matmul(train_in, w))
print("output_pred: ", output_pred.eval(session=sess))
```

AND Gate

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

$a*x1 + b*x2 = c$





```
tf.sigmoid(x, name=None)
tf.nn.relu(x, name=None)
tf.tanh(x, name=None)
```

Perceptron as a Linear Classifier

Linear Classifier

binary logistic regression

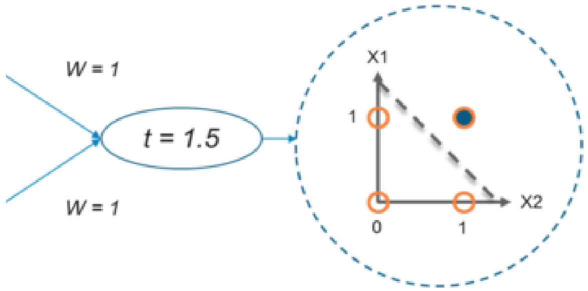
```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from datetime import datetime
%load_ext tensorboard

#input1, input2 and bias
X_train = [[1., 1.,1], [1., 0,1], [0, 1.,1], [0, 0,1]]
#output
y_train = [[1.], [0], [0], [0]]

logdir = "logs/scalars/" + datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=logdir)

# define model
model = Sequential()
model.add(Dense(1, activation='sigmoid'))

# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',
                                                    metrics=['accuracy'])
# fit the model
model.fit(X_train, y_train, epochs=3000, batch_size=4, verbose=1,
                                  callbacks=[tensorboard_callback])

# evaluate the model
loss, acc = model.evaluate(X_train, y_train, verbose=0)
print('Test Accuracy: %.3f' % acc)

# make a prediction
row = [0,1,1]
yhat = model.predict([row])
print('Predicted: %.3f' % yhat)
%tensorboard --logdir logs/scalars
```

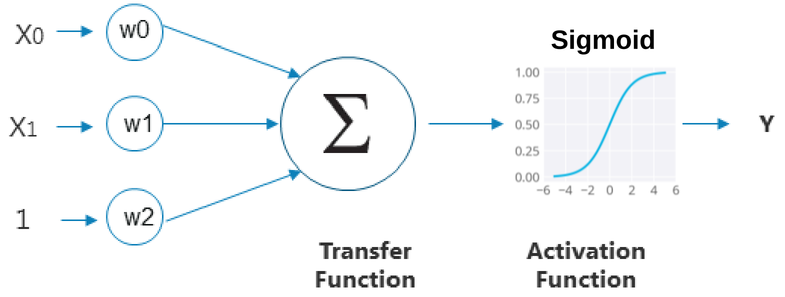
TensorFlow 2.0

AND Gate

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

*a*x1 + b*x2 = c*



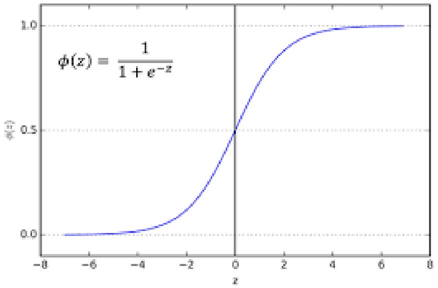


Classification on MNIST dataset

Each image is a 28x28 array, flattened out to be a 1-d tensor of size 784
MNIST.train: 55,000 examples
MNIST.validation: 5,000 examples
MNIST.test: 10,000 examples

Model: *take a **weighted sum** of the features and add a **bias** to get the **logit**. Convert the logit to a **probability** via the **logistic-sigmoid function**.*

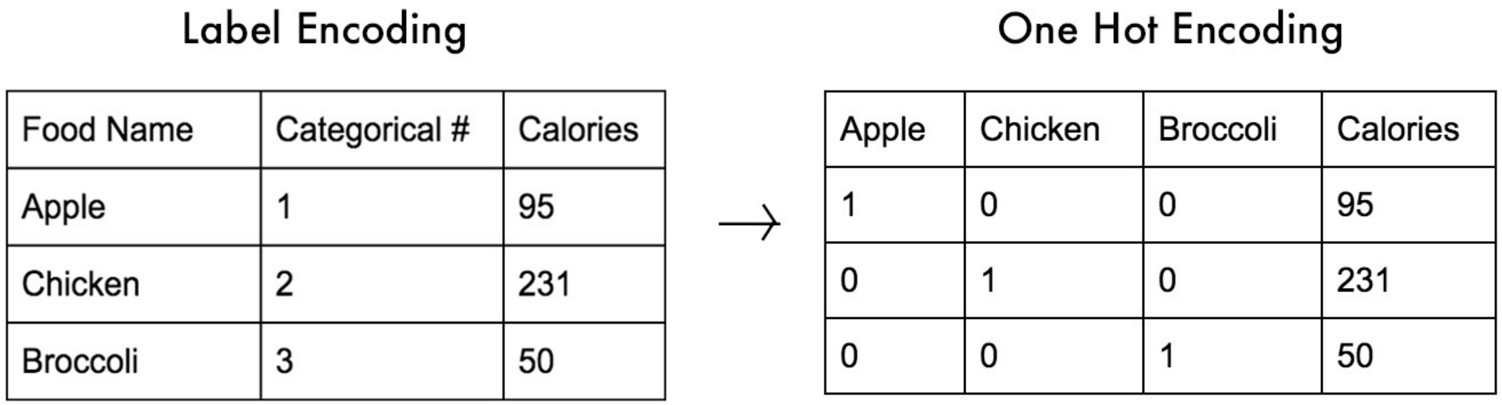
$$logits = X * W + b$$
$$Y_{predicted} = softmax(logits)$$
$$loss = cross\_entropy(Y, Y_{predicted})$$ *, where Y is a one-hot vector*



MNIST Dataset: https://en.wikipedia.org/wiki/MNIST_database; http://yann.lecun.com/exdb/mnist/
MNIST Dataset in csv format: https://pjreddie.com/projects/mnist-in-csv/

One-Hot representation

*One-Hot is a way to represent categorical data. It is a group of bits among which the legal combinations of values are only those with a single high (1) bit and all the others low (0). A similar implementation in which all bits are '1' except one '0' is sometimes called **one-cold**.*



Label Encoding

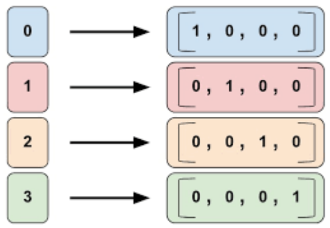
Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

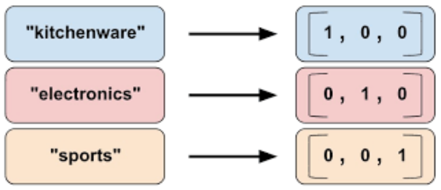
→

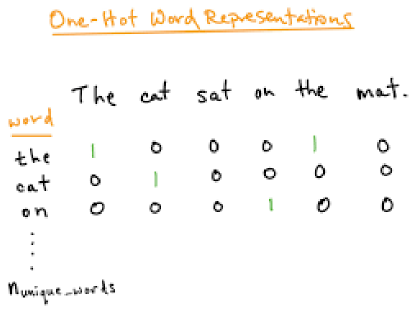
One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

Examples





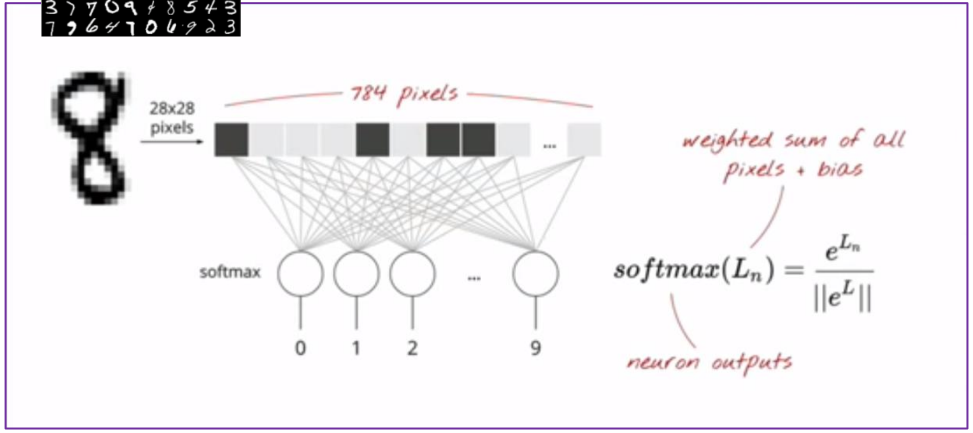


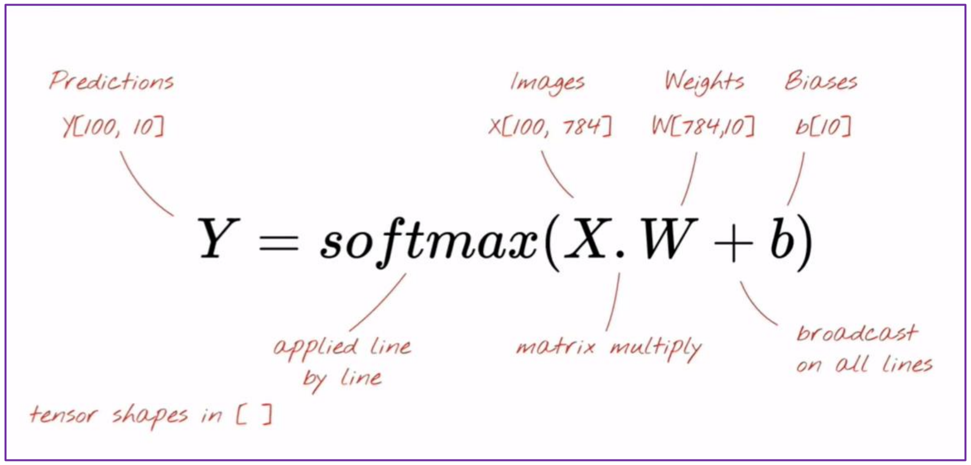
One-Hot Word Representations

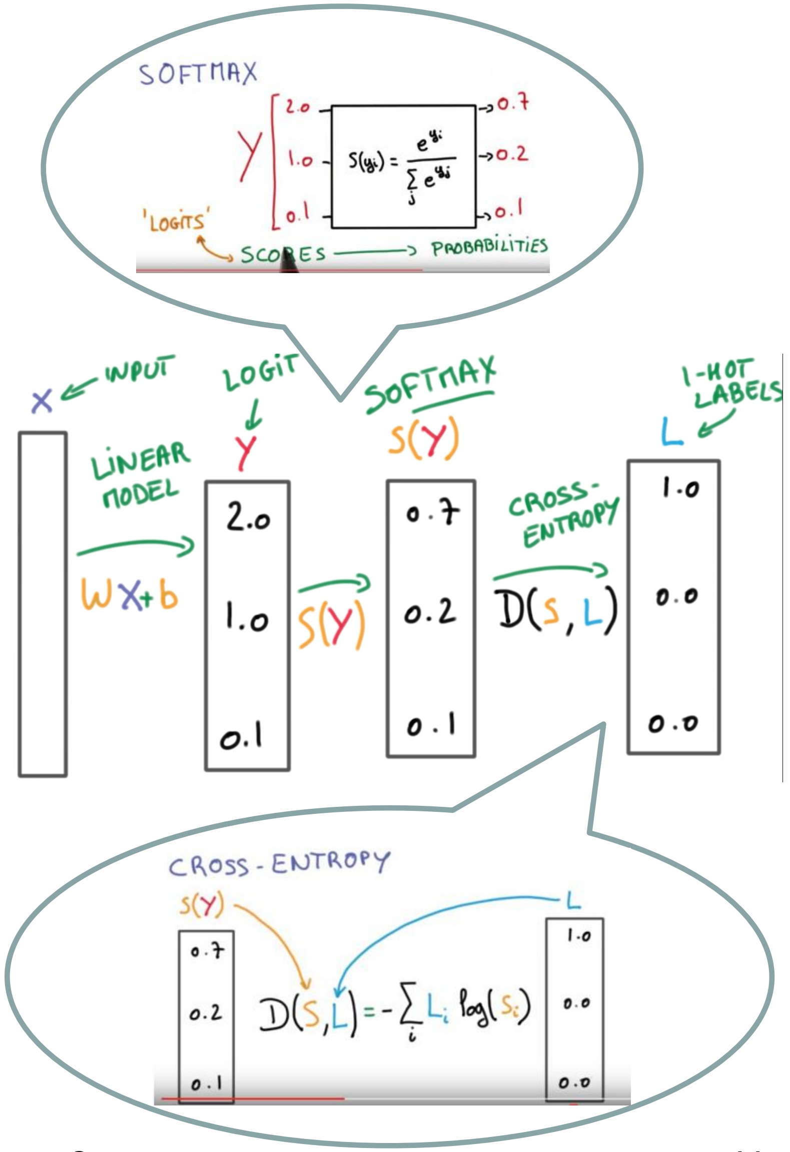
word	The	cat	sat	on	the	mat.
the	1	0	0	0	1	0
cat	0	1	0	0	0	0
on	0	0	0	1	0	0
⋮						
Nunique_words						

https://en.wikipedia.org/wiki/One-hot

Softmax and Cross Entropy


28x28
pixels
784 pixels
softmax
0
1
2
9
weighted sum of all pixels + bias
$softmax(L_n) = \frac{e^{L_n}}{||e^L||}$
neuron outputs



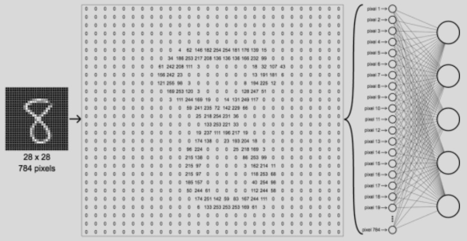


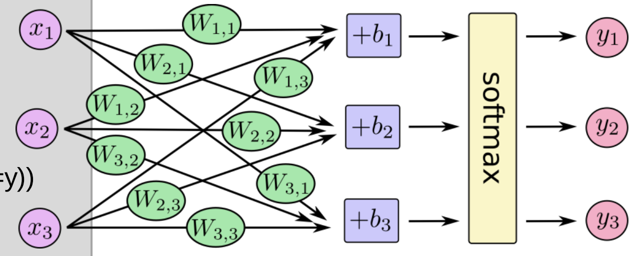
https://www.ritchieng.com/machine-learning/deep-learning/neural-nets/
https://www.youtube.com/watch?v=vq2nnJ4g6N0
https://www.youtube.com/watch?v=tRsSi_sqXjI

Linear Classifier

*A **single-layer neural network** based classification of handwritten digits form MNIST dataset*

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

sess = tf.InteractiveSession()

#Import data
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mnist.load_data()
train_images = tf.reshape(train_images,[60000,784])
test_images = tf.reshape(test_images,[10000,784])
train_images /= 255
test_images /= 255
unique_category_count = 10
y_train = tf.one_hot(train_labels, unique_category_count)
y_test = tf.one_hot(test_labels, unique_category_count)

# Create the model
x_ = tf.placeholder(tf.float32, [None, 784])
y_ = tf.placeholder(tf.float32, [None, unique_category_count])
W = tf.Variable(tf.zeros([784, unique_category_count]))
b = tf.Variable(tf.zeros([unique_category_count]))
y = tf.nn.softmax( tf.matmul(x, W) + b)

# Define loss and optimizer
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=y_, logits=y))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
tf.global_variables_initializer().run()
```



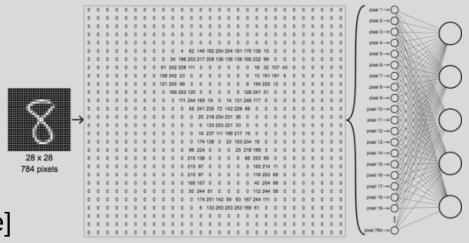


MNIST Dataset: https://en.wikipedia.org/wiki/MNIST_database; http://yann.lecun.com/exdb/mnist/
MNIST Dataset in csv format: https://pjreddie.com/projects/mnist-in-csv/

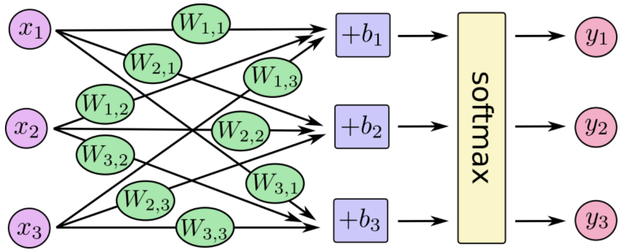
Linear Classifier

*A **single-layer neural network** based classification of handwritten digits form MNIST dataset*

```
# Train
batch_size = 100
epochs = 2
for e in range(epochs):
    print('Epoch: {}'.format(e+1))
    n_batches = train_images.shape[0]//batch_size
    for i in range(n_batches):
        print('Batch {} of {}'.format(i+1, n_batches))
        batch_xs = train_images[i * batch_size: (i+1) * batch_size]
        batch_ys = y_train[i * batch_size: (i+1) * batch_size]
        sess.run(train_step, feed_dict={x: sess.run(batch_xs), y_: sess.run(batch_ys)})

# Test trained model
correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy: ', sess.run(accuracy, feed_dict={x: sess.run(test_images), y_: sess.run(y_test)}))
```



```
Accuracy:  0.9073
```



Version 1

Linear Classifier

A single-layer neural network based classification of handwritten digits form MNIST dataset

```
import tensorflow as tf
from numpy import argmax
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

# Import data
(train_images, train_labels), (test_images, test_labels) =
                                    tf.keras.datasets.mnist.load_data()
# Prepare output
unique_category_count = 10
print("Shape before one-hot encoding: ", train_labels.shape)
y_train = tf.one_hot(train_labels, unique_category_count)
                                    # or tf.keras.utils.to_categorical()
y_test = tf.one_hot(test_labels, unique_category_count)
print("Shape after one-hot encoding: ", y_train.shape)
# Prepare input
train_images = train_images.astype('float32')
test_images = test_images.astype('float32')
print(train_images.shape)
train_images = tf.reshape(train_images,[60000,784])
                                    # or train_images.reshape(60000, 784)
test_images = tf.reshape(test_images,[10000,784])
print(train_images.shape)
# normalizing the data to help with the training
train_images /= 255
test_images /= 255

# define model
model = Sequential()
model.add(Dense(10, activation='softmax'))
```

```
# compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
# fit the model
model.fit(train_images, y_train, epochs=2, batch_size=100, verbose=1)
# evaluate the model
loss, acc = model.evaluate(test_images, y_test, verbose=0)
print('Test Accuracy: %.3f' % acc)
```

```
Test Accuracy:  0.914
```

```
# predict
k = 1
test_im = tf.reshape(test_images[k],[1,784])
print(test_labels[k])
print(y_test[k].numpy())
yhat = model.predict(test_im)
print('Predicted: %s (class=%d)' % (yhat, argmax(yhat)))
```

```
2
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
Predicted: [[2.4658944e-03 8.4701322e-05 9.6051437e-01 7.8007090e-03 1.5989290e-08
 1.1088169e-02 1.6074454e-02 2.4621694e-09 1.9715305e-03 2.2467559e-07]] (class=2)
```

Version 2

Linear Classifier

*A **single-layer neural network** based classification of handwritten digits form MNIST dataset*

```
import tensorflow as tf
from numpy import argmax
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten

# Import data
(train_images, train_labels), (test_images, test_labels) =
                                        tf.keras.datasets.mnist.load_data()
# Prepare input
train_images = train_images.astype('float32')
test_images = test_images.astype('float32')
print(train_images.shape)
# normalizing the data to help with the training
train_images /= 255
test_images /= 255

# define model
model = Sequential()
model.add(Flatten(input_shape=(28,28)))
model.add(Dense(10, activation='softmax'))
# compile the model
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# fit the model
model.fit(train_images, train_labels, epochs=20, batch_size=100,
verbose=1)
# evaluate the model
loss, acc = model.evaluate(test_images, test_labels, verbose=0)
print('Test Accuracy: %.3f' % acc)
```

```
Test Accuracy:  0.913 (with 2 epochs)
Test Accuracy:  0.924 (with 20 epochs)
```

```
# predict
k = 1
test_im = tf.reshape(test_images[k],[1,28,28])
print(test_labels[k])
yhat = model.predict(test_im)
print('Predicted: %s (class=%d)' % (yhat, argmax(yhat)))
```

```
2
Predicted: [[0.10497269 0.11022787 0.11731301 0.11628813 0.08157932 0.09450245
 0.10957752 0.08391336 0.09888434 0.08274125]] (class=2)
```

Since TensorFlow 2.x includes Keras, there is no need to install it separately. Simply access it as:

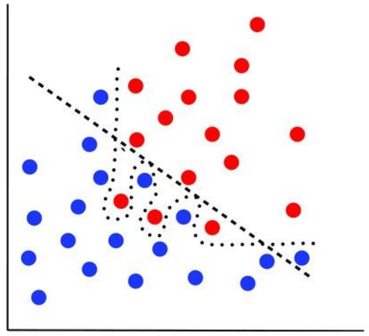
import tensorflow as tf
tf.keras. …

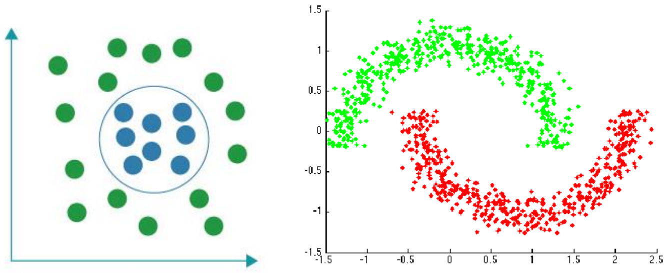
Difference between 'categorical_crossentropy‘ and 'sparse_categorical_crossentropy‘ losses:

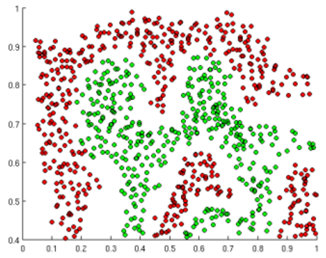
If use 'sparse_categorical_crossentropy‘, you do not need to transform the logist into categorical (via one hot representation).

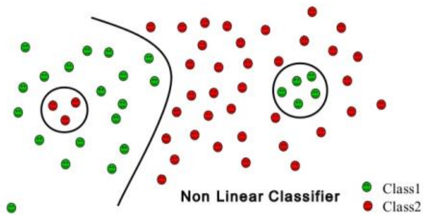
Classification

Linear model cannot serve non linear classification problems…



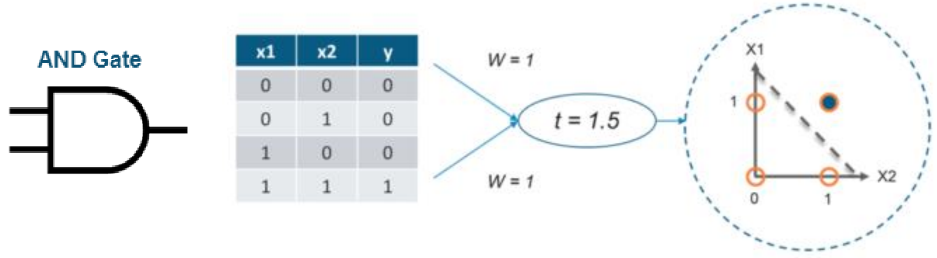




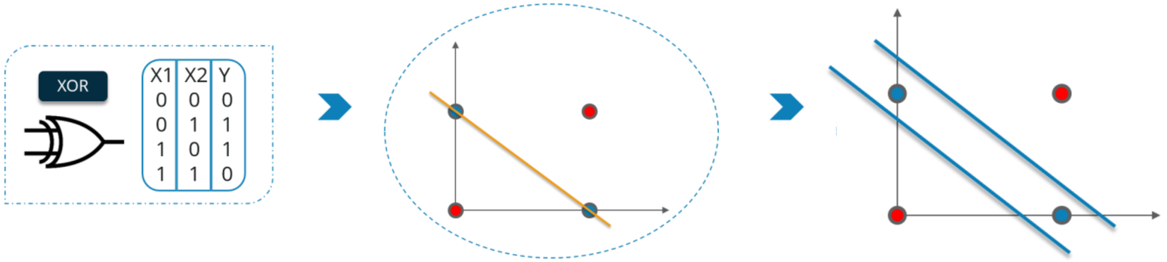


Let's go *Deeper!!!*

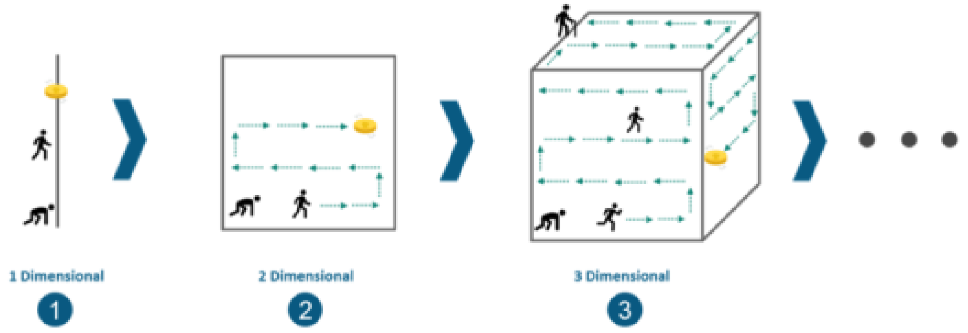
Neural Networks



High dimensionality is a one of the bottlenecks for Machine Learning. Handling and processing high dimensional data (where input & output is quite large) becomes very complex and resource exhaustive.



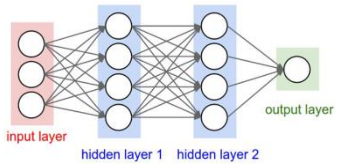
Deep learning is capable of handling the high dimensional data being efficient in focusing on the right features on its own (feature extraction process).

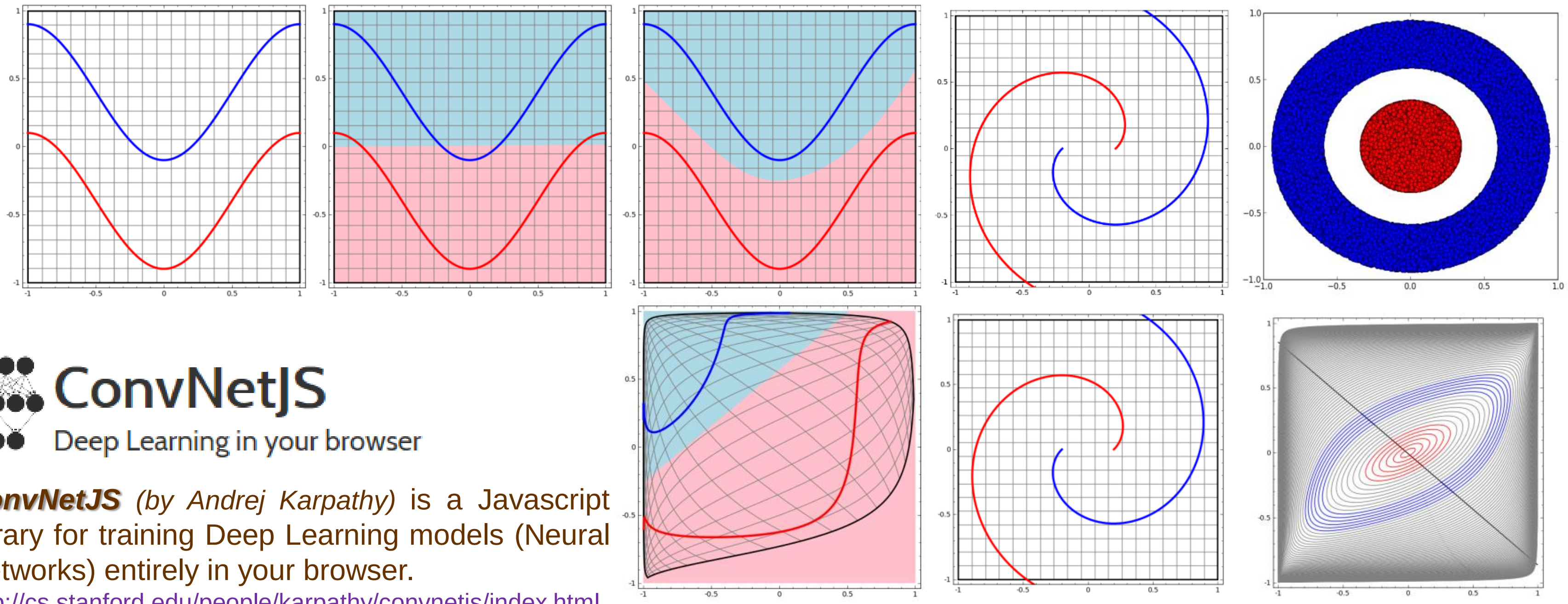


Relevant links:
https://www.edureka.co/blog/deep-learning-tutorial#deep-learning-use-case

Neural Networks

Neural Network change an original representation of data into the most appropriate representation to solve the problems …





ConvNetJS
Deep Learning in your browser

ConvNetJS *(by Andrej Karpathy)* is a Javascript library for training Deep Learning models (Neural Networks) entirely in your browser.
http://cs.stanford.edu/people/karpathy/convnetjs/index.html

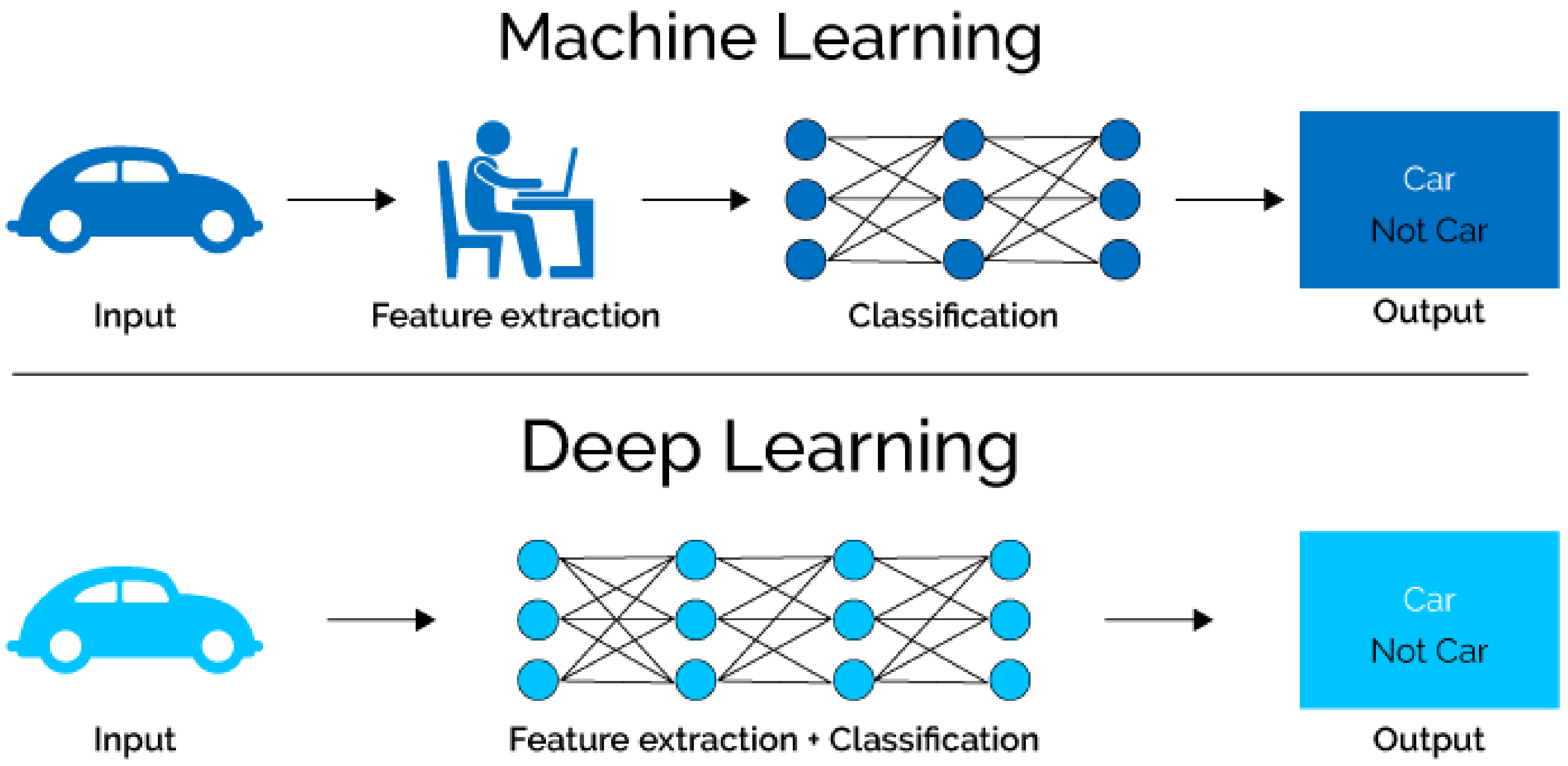
Play with ***ConvnetJS demo***: toy 2d classification with 2-layer neural network
http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

TensorFlow Playground - neural network right in your browser: http://playground.tensorflow.org/
https://cloud.google.com/blog/big-data/2016/07/understanding-neural-networks-with-tensorflow-playground

Relevant links:
http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/
https://www.youtube.com/watch?v=BR9h47Jtqyw

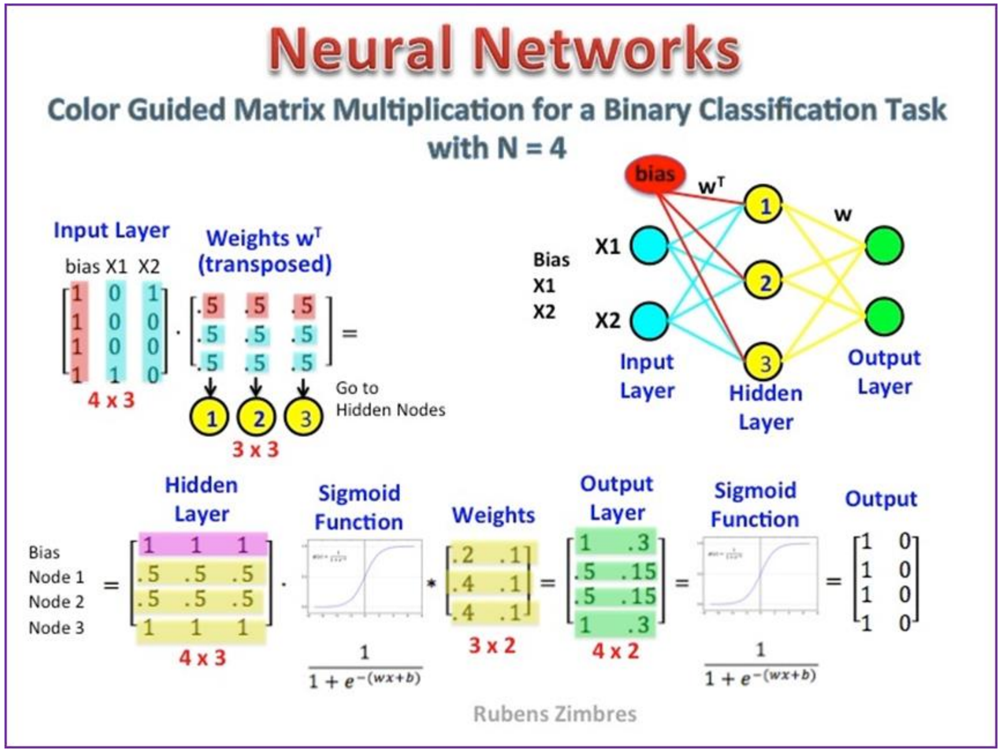
Neural Networks

One of the most exciting features of deep learning is its performance in ***feature learning***; the algorithms perform particularly well in being able to detect features from raw data.



Relevant links:
https://udarajay.com/applied-machine-learning-the-less-confusing-guide/
https://www.youtube.com/watch?v=aircAruvnKk

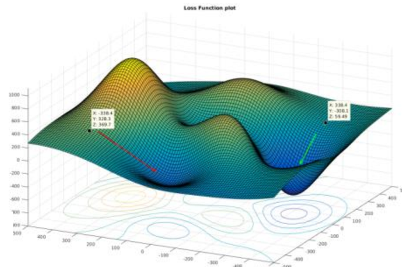
Neural Networks

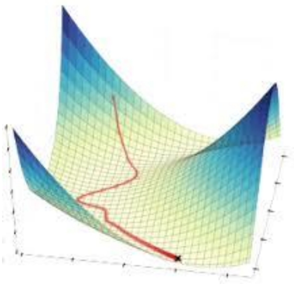


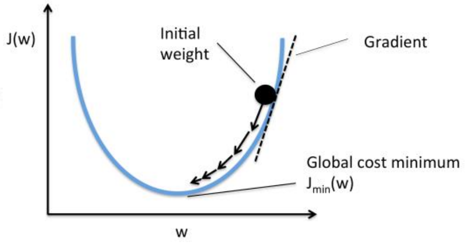
Forward propagation is a neural net's way of classifying a set of inputs (when input data goes through the nodes and their activation functions and finally form confidence levels for the nodes of output layer).

To train the net, the output from forward prop is compared to the output that is known to be correct, and the cost is the difference of the two. ***The point of training is to make that cost as small as possible***, across millions of training examples. To do this, the net tweaks the weights and biases step by step until the prediction closely matches the correct output.

One way to train the model is called as ***Backpropagation*** - algorithm that looks for the minimum value of the error function in weight space using a technique called the delta rule or ***gradient descent***. The weights that minimize the error function is then considered to be a solution to the learning problem.





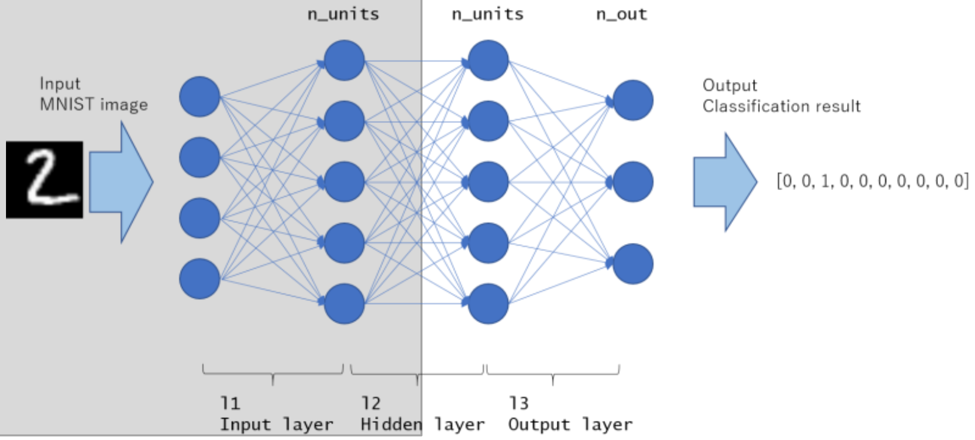


Relevant links:
https://www.edureka.co/blog/backpropagation/
https://en.wikipedia.org/wiki/Backpropagation
https://www.youtube.com/watch?v=IHZwWFHWa-w
https://www.youtube.com/watch?v=Ilg3gGewQ5U

Non Linear Classification with DNN

Multi Layer Perceptron (MLP) Classification of handwritten digits form MNIST dataset

```
import tensorflow.compat.v1 as tf
tf.compat.v1.disable_eager_execution()
import matplotlib.pyplot as plt
import numpy as np

# Import MNIST data
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mnist.load_data()
train_images = tf.reshape(train_images,[60000,784])
test_images = tf.reshape(test_images,[10000,784])
train_images /= 255
test_images /= 255
unique_category_count = 10
y_train = tf.one_hot(train_labels, unique_category_count)
y_test = tf.one_hot(test_labels, unique_category_count)

# Parameters
learning_rate = 0.001
training_epochs = 5
batch_size = 1000
display_step = 1

# Network Parameters
n_hidden_1 = 256 # 1st layer number of features
n_hidden_2 = 256 # 2nd layer number of features
n_input = 784      # MNIST data input (img shape: 28*28)
n_classes = 10      # MNIST total classes (0-9 digits)


x = tf.placeholder("float", [None, n_input])
y = tf.placeholder("float", [None, n_classes])
```



Non Linear Classification with DNN

TensorFlow

Multi Layer Perceptron (MLP) Classification of handwritten digits form MNIST dataset

```
# Create model
def multilayer_perceptron(x, weights, biases):
   # Use tf.matmul instead of "*" because tf.matmul can change it's dimensions on the fly (broadcast)
   print( 'x:', x.get_shape(), 'W1:', weights['h1'].get_shape(), 'b1:', biases['b1'].get_shape())
   # Hidden layer with RELU activation
   layer_1 = tf.add(tf.matmul(x, weights['h1']), biases['b1']) #(x*weights['h1']) + biases['b1']
   layer_1 = tf.nn.relu(layer_1)
   # Hidden layer with RELU activation
   print( 'layer_1:', layer_1.get_shape(), 'W2:', weights['h2'].get_shape(), 'b2:', biases['b2'].get_shape())
   layer_2 = tf.add(tf.matmul(layer_1, weights['h2']), biases['b2']) # (layer_1 * weights['h2']) + biases['b2']
   layer_2 = tf.nn.relu(layer_2)
   # Output layer with linear activation
   print( 'layer_2:', layer_2.get_shape(), 'W3:', weights['out'].get_shape(), 'b3:', biases['out'].get_shape())
   out_layer = tf.matmul(layer_2, weights['out']) + biases['out'] # (layer_2 * weights['out']) + biases['out']
   print('out_layer:',out_layer.get_shape())

   return out_layer

# Store layers weight & bias
weights = {
   'h1': tf.Variable(tf.random_normal([n_input, n_hidden_1])),       #784x256
   'h2': tf.Variable(tf.random_normal([n_hidden_1, n_hidden_2])), #256x256
   'out': tf.Variable(tf.random_normal([n_hidden_2, n_classes]))    #256x10
}
biases = {
   'b1': tf.Variable(tf.random_normal([n_hidden_1])),          #256x1
   'b2': tf.Variable(tf.random_normal([n_hidden_2])),          #256x1
   'out': tf.Variable(tf.random_normal([n_classes]))            #10x1
}
```

Non Linear Classification with DNN

Multi Layer Perceptron (MLP) Classification of handwritten digits form MNIST dataset

```
# Construct model
pred = multilayer_perceptron(x, weights, biases)
# Cross entropy loss function
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(labels=y, logits=pred))
# On this case we choose the AdamOptimizer
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
# Initializing the variables
init = tf.global_variables_initializer()
# Launch the graph
with tf.Session() as sess:
    sess.run(init)

    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(train_images.shape[0]/batch_size)
        # Loop over all batches
        for i in range(total_batch):
            batch_xs = train_images[i * batch_size: (i+1) * batch_size]
            batch_ys = y_train[i * batch_size: (i+1) * batch_size]
            # Run optimization op (backprop) and cost op (to get loss value)
            _, c = sess.run([optimizer, cost], feed_dict={x: sess.run(batch_xs), y: sess.run(batch_ys)})
            # Compute average loss
            avg_cost += c / total_batch
        # Display logs per epoch step
        if epoch % display_step == 0:
            print ("Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(avg_cost))
    print("Optimization Finished!")
```

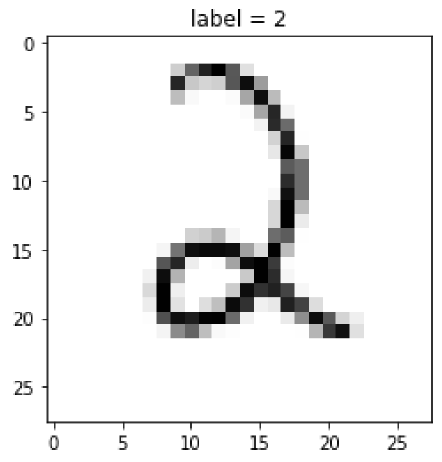
Non Linear Classification with DNN

Multi Layer Perceptron (MLP) Classification of handwritten digits form MNIST dataset

```
# Test model
correct_prediction = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
# To keep sizes compatible with model
print ("Accuracy:", accuracy.eval({x: sess.run(test_images), y: sess.run(y_test)}))

print("Test sample:")
#Get 28x28 image
sample_1 = tf.reshape(test_images[47], [28,28])
# Get corresponding integer label from one-hot encoded data
sample_label_1 = np.where(sess.run(y_test[47]) == 1)[0][0]
# Plot sample
plt.imshow(sess.run(sample_1), cmap='Greys')
plt.title('label = {}'.format(sample_label_1))
plt.show()
x_test = tf.reshape(test_images[47], [1,784])
predicted_value = tf.argmax(pred, 1)
print("is classified as: {}".format(predicted_value.eval(session=sess, feed_dict={x: sess.run(x_test)})))
```

```
Test Accuracy:  0.859 (with 5 epochs)
Test Accuracy:  0.919 (with 20 epochs)
Test Accuracy:  0.929 (with 30 epochs)
```

```
Test sample:
```



```
is classified as: [2]
```

Non Linear Classification with DNN

TensorFlow 2.0

Multi Layer Perceptron (MLP) Classification of handwritten digits form MNIST dataset

```
import tensorflow as tf
from numpy import argmax
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten

# Import data
(train_images, train_labels), (test_images, test_labels) =
                                        tf.keras.datasets.mnist.load_data()
# Prepare input
train_images = train_images.astype('float32')
test_images = test_images.astype('float32')
print(train_images.shape)
# normalizing the data to help with the training
train_images /= 255
test_images /= 255

# define model
model = Sequential()
model.add(Flatten(input_shape=(28,28)))
model.add(Dense(256, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(256, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(10, activation='softmax'))
# compile the model
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# fit the model
model.fit(train_images, train_labels, epochs=20, batch_size=100,
verbose=1)
```

```
# evaluate the model
loss, acc = model.evaluate(test_images, test_labels, verbose=0)
print('Test Accuracy: %.3f' % acc)
```

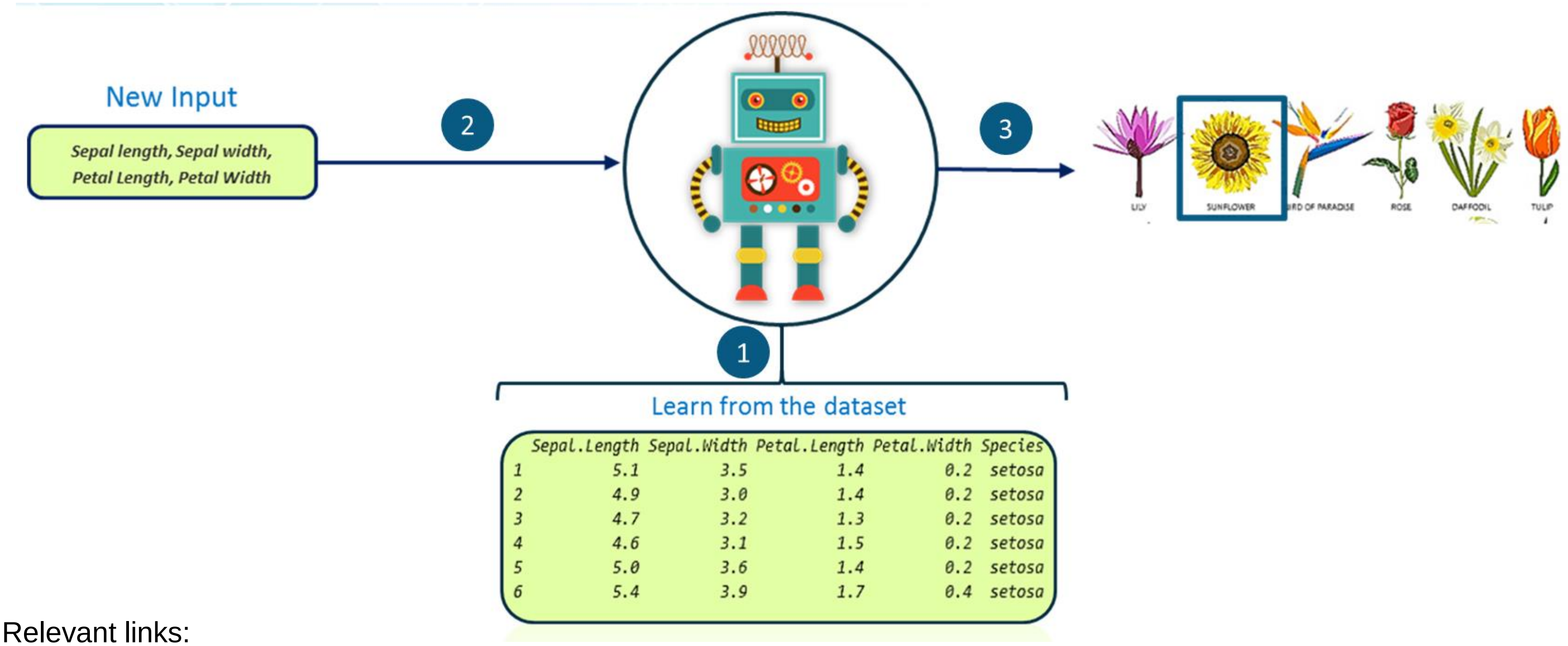
```
Test Accuracy:  0.980 (with 20 epochs)
Test Accuracy:  0.984 (with 40 epochs)
```

```
# predict
k = 1
test_im = tf.reshape(test_images[k],[1,28,28])
print(test_labels[k])
yhat = model.predict(test_im)
print('Predicted: %s (class=%d)' % (yhat, argmax(yhat)))
```

```
2
Predicted: [[1.2363079e-19 8.7495413e-25 1.0000000e+00 5.4430578e-20 9.0466738e-36
  7.3684840e-29 4.6826463e-26 9.1150330e-33 1.9784944e-20 1.6744042e-37]] (class=2)
```

Iris flower dataset

Iris flower dataset (Ronald Fisher's Iris data set): https://en.wikipedia.org/wiki/Iris_flower_data_set
*The dataset contains a set of 50 records under 5 attributes - **Petal Length** , **Petal Width** , **Sepal Length** , **Sepal Width** and **Class**. The data set consists of 50 samples from each of three species of Iris (**Iris setosa**, **Iris virginica** and **Iris versicolor**).*
https://gist.github.com/curran/a08a1080b88344b0c8a7
http://download.tensorflow.org/data/iris_training.csv
https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv



	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

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

Classification

```
import numpy as np
from numpy import argmax
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
# load the dataset
path =
'https://raw.githubusercontent.com/jbrownlee/Datasets/master/iris.csv'
df = read_csv(path, header=None)
# split into input and output columns
X, y = df.values[:, :-1], df.values[:, -1]
# ensure all data are floating point values and scale them
X = X.astype('float32')
scaler_X = MinMaxScaler()
scaler_X.fit(X)
X = scaler_X.transform(X)
# encode strings to integer
y = LabelEncoder().fit_transform(y)
# split into train and test datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
# determine the number of input features
n_features = X_train.shape[1]
# create dictionary of target classes
label_dict = {
   0: 'Setosa',
   1: 'Versicolor',
   2: 'Virginica'
}
```

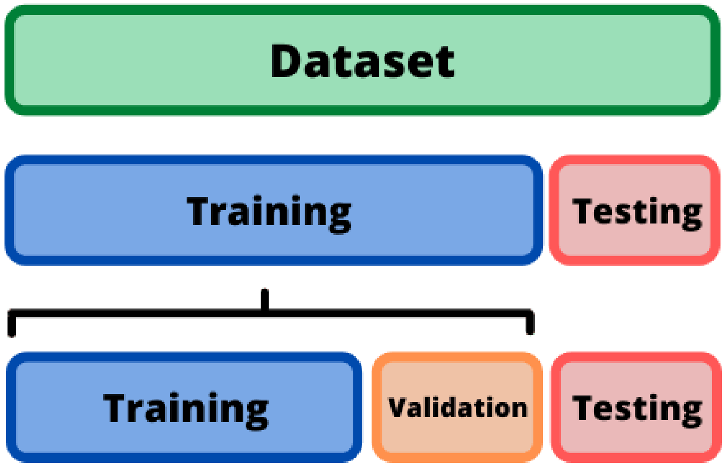
```
# define model
model = Sequential()
model.add(Dense(10, activation='relu', kernel_initializer='he_normal',
                                                  input_shape=(n_features,)))
model.add(Dense(8, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(3, activation='softmax'))
# compile the model
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
# fit the model
model.fit(X_train, y_train, epochs=500, batch_size=32, verbose=0)
# evaluate the model
loss, acc = model.evaluate(X_test, y_test, verbose=0)
print('Test Accuracy: %.3f' % acc)
```

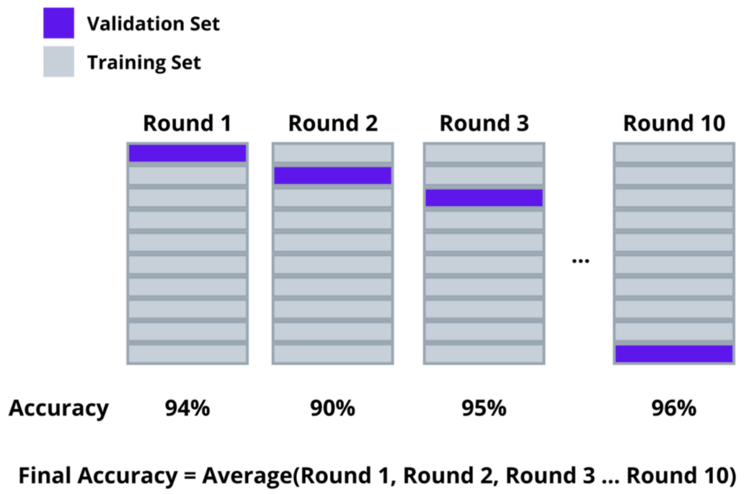
```
Test Accuracy:  0.980
```

```
# Classify new flower samples.
new_samples = np.array([[6.4, 3.2, 4.5, 1.5],
                  [6.4, 2.8, 5.6, 2.2],
                  [7.0, 3.2, 4.7, 1.4],
                  [5.1, 3.3, 1.7, 0.5],
                  [5.8, 3.1, 5.0, 1.7]], dtype=np.float32)
new_samples = scaler_X.transform(new_samples)
yhat = model.predict(new_samples)
for i in range(len(yhat)):
   print('Class prediction {} : {}'.format(i+1,
                                       label_dict[int(argmax(yhat[i]))]))
```

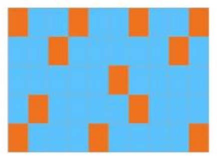
```
Class prediction 1 : Versicolor
Class prediction 2 : Virginica
Class prediction 3 : Versicolor
Class prediction 4 : Setosa
Class prediction 5 : Versicolor
```

Training and Evaluation





```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split (
                        X, y,
                        test_size = 0.2,
                        random_state = 1)
```



*… the error rate measured by **k-fold cross validation** might underestimate the true error rate once the model is applied to an unseen dataset…*

*The **validation set** is used to optimize the model parameters while the **test set** is used to provide an unbiased estimate of the final model…*

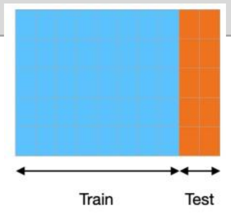
Relevant links:
https://sdsclub.com/how-to-train-and-test-data-like-a-pro/
https://towardsdatascience.com/addressing-the-difference-between-keras-validation-split-and-sklearn-s-train-test-split-a3fb803b733
https://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/
https://www.tensorflow.org/guide/keras/train_and_evaluate

Training and Evaluation

Fit the model…

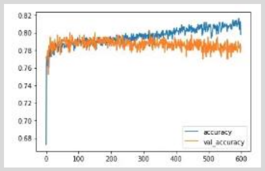
```
# fit model on training data and  pass some validation for monitoring validation loss and metrics
# at the end of each epoch
history = model.fit(x_train, y_train, epochs=2, validation_data=(x_val, y_val))
```

```
history = model.fit(   … , validation_split=0.2)
```

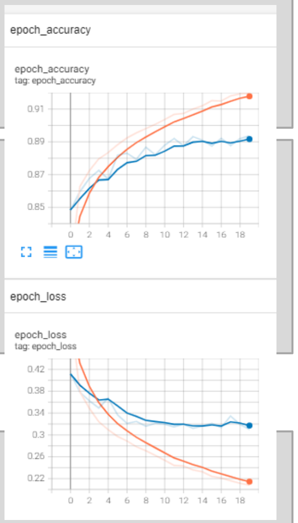


Visualize the training process…

```
import matplotlib.pyplot as plt
# plot the model accuracy and validation accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['accuracy','validation accuracy'])
```



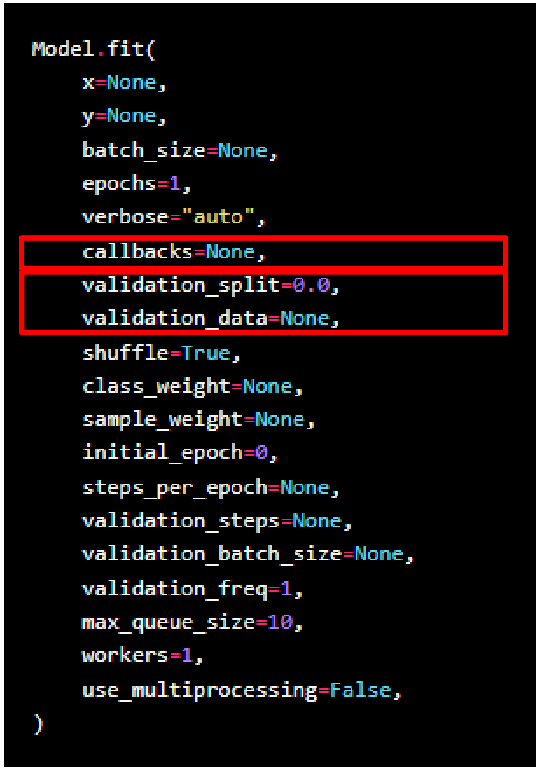
```
%load_ext tensorboard
…
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir="./logs")
model.fit( x_train, y_train, epochs=2, callbacks=[tensorboard_callback])
# run the tensorboard command to view the visualizations.
%tensorboard --logdir './logs'
```



Evaluate and predict…

```
# Evaluate the model on the test data using `evaluate`
results = model.evaluate(x_test, y_test)
# Generate predictions
predictions = model.predict(x_test)
```

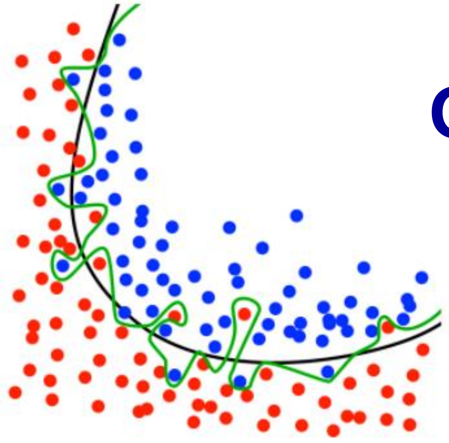
`fit` method



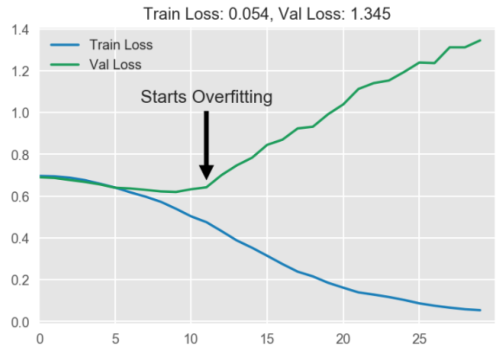
```
Model.fit(
    x=None,
    y=None,
    batch_size=None,
    epochs=1,
    verbose="auto",
    callbacks=None,
    validation_split=0.0,
    validation_data=None,
    shuffle=True,
    class_weight=None,
    sample_weight=None,
    initial_epoch=0,
    steps_per_epoch=None,
    validation_steps=None,
    validation_batch_size=None,
    validation_freq=1,
    max_queue_size=10,
    workers=1,
    use_multiprocessing=False,
)
```

Overfitting

Overfitting *is "the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably"*

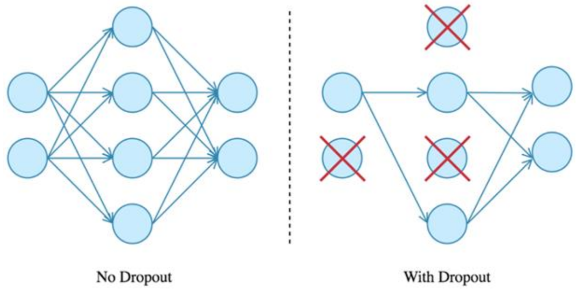

Classification with overfitting (green) vs. better classification boundary (black)

We may face a case, when training loss keeps going down, but the validation loss starts increasing after certain epoch. It is a sign of ***overfitting***. It tells us that our model is memorizing the training data, but it's failing to generalize to new instances.



The most popular regularization technique for deep neural networks is **Dropout**. It is used to prevent overfitting via temporarily "dropping"/disabling a neuron with probability ***p*** (a hyperparameter called the ***dropout-rate*** that is typically a number around 0.5) at each iteration during the training phase.

- The dropped-out neurons are resampled with probability *p* at every training step (a dropped out neuron at one step can be active at the next one). (Dropout is not applied during test time after the network is trained).
- Dropout can be applied to input or hidden layer nodes but not the output nodes.
- Applying dropout regularization, it is recommended to increase a number of training rounds.

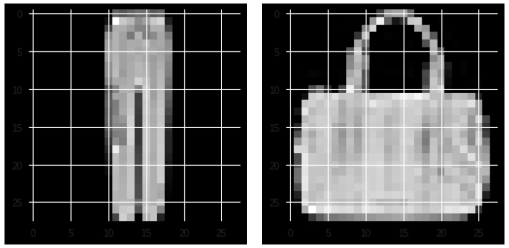


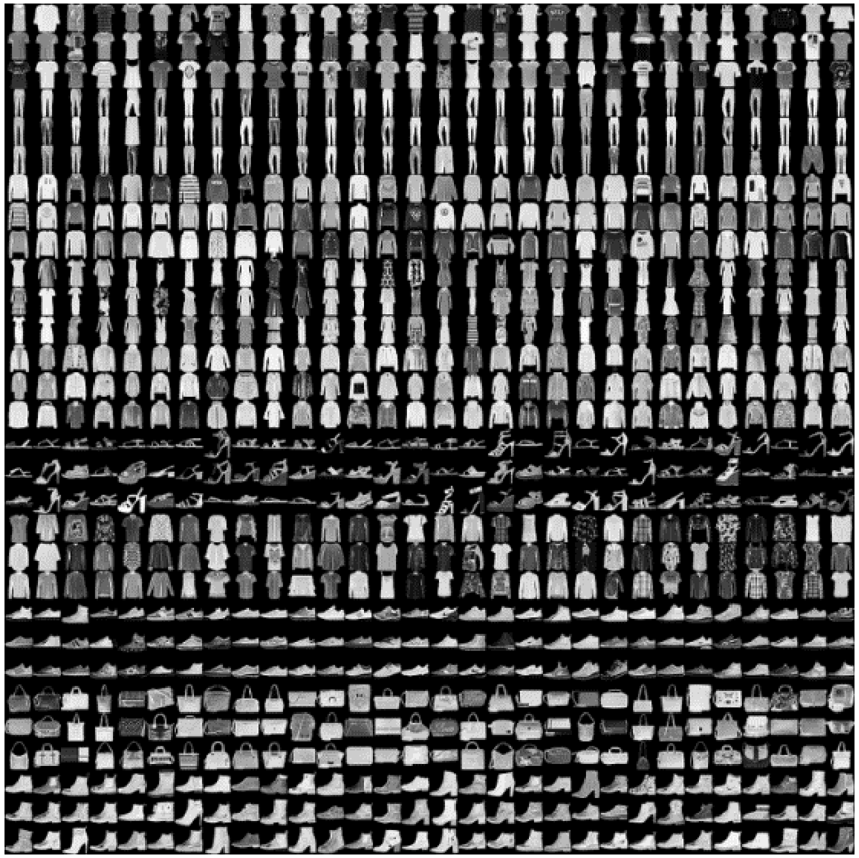
Reason. Dropout forces every neuron to be able to operate independently and prevents the network to be too dependent on a small number of neurons.

Relevant links:
https://en.wikipedia.org/wiki/Overfitting

Classification

Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. Han Xiao, Kashif Rasul, Roland Vollgraf
https://arxiv.org/abs/1708.07747
https://github.com/zalandoresearch/fashion-mnist
https://medium.com/tensorist/classifying-fashion-articles-using-tensorflow-fashion-mnist-f22e8a04728a





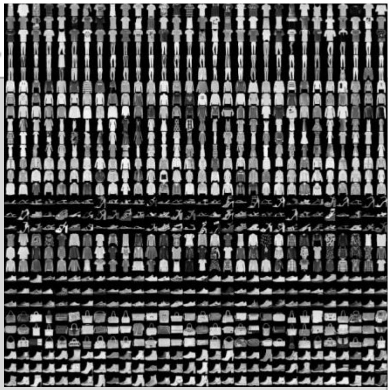
Label Description

- *0 T-shirt*
- *1 Trouser*
- *2 Pullover*
- *3 Dress*
- *4 Coat*
- *5 Sandal*
- *6 Shirt*
- *7 Sneaker*
- *8 Bag*
- *9 Ankle boot*

TensorFlow 2.0

Classification

Different model performance comparison with Fashion-MNIST dataset…

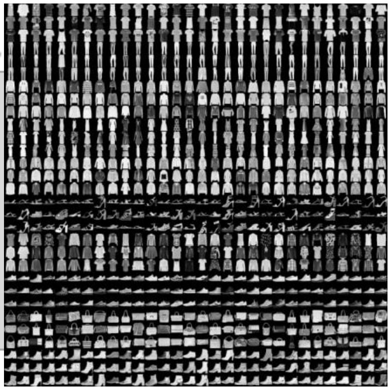


```
import tensorflow as tf
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Sequential, , load_model
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.optimizers import SGD
Import os
from numpy import argmax
import matplotlib.pyplot as plt
%matplotlib inline

batch_size = 128
num_classes = 10
epochs = 20
# the data, shuffled and split between train and test sets
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
# check some data
print (y_train[1000])
fig= plt.figure()
plt.imshow(x_train[1000], cmap='gray')
fig= plt.figure(figsize=(20,10))
for i in range (1,41):
    ax1 = fig.add_subplot(5,8,i)
    plt.xticks([], [])
    plt.yticks([], [])
    ax1.imshow(x_train[i], cmap='gray')
# normalizing the data to help with the training
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
```

```
# prepare output dictionary
di = {0:'T-shirt',
      1:'Trouser',
      2:'Pullover',
      3:'Dress',
      4:'Coat',
      5:'Sandal',
      6:'Shirt',
      7:'Sneaker',
      8:'Bag',
      9:'Ankle boot'}

def train_model(model):
    history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
                        verbose=1, validation_data=(x_test, y_test))
    score = model.evaluate(x_test, y_test, verbose=0)
    print('Test loss:', score[0])
    print('Test accuracy:', score[1])
    return model


# define and trian the model

…
```

Classification

Different model performance comparison with Fashion-MNIST dataset…



```
# Single layer with 1 unit
model0 = Sequential()
model0.add(Flatten(input_shape=(28,28)))
model0.add(Dense(1, activation='relu', kernel_initializer='he_normal'))
model0.add(Dense(num_classes, activation='softmax'))
model0.summary()
model0.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model_ = train_model(model0)
```

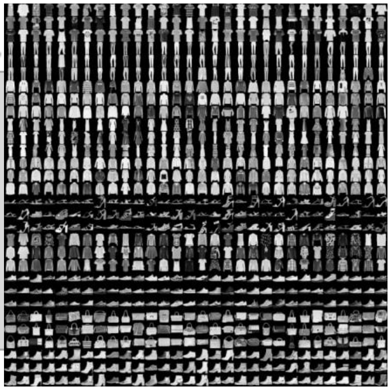
```
…
Epoch 20/20
469/469 [==============================] - 0s 719us/step - loss: 1.4434 - accuracy: 0.4016 - val_loss: 1.4556 - val_accuracy: 0.3984
Test loss: 1.4555656909942627
Test accuracy: 0.3984000086784363
```

```
# Single layer with 10 units
model1 = Sequential()
model1.add(Flatten(input_shape=(28,28)))
model1.add(Dense(10, activation='relu', kernel_initializer='he_normal'))
model1.add(Dense(num_classes, activation='softmax'))
model1.summary()
model1.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model_ = train_model(model1)
```

```
…
Epoch 20/20
469/469 [==============================] - 0s 860us/step - loss: 0.3749 - accuracy: 0.8705 - val_loss: 0.4374 - val_accuracy: 0.8457
Test loss: 0.437436580657959
Test accuracy: 0.8457000255584717
```

Classification

Different model performance comparison with Fashion-MNIST dataset…



```
# Single Layer with more units
model2 = Sequential()
model2.add(Flatten(input_shape=(28,28)))
model2.add(Dense(512, activation='relu', kernel_initializer='he_normal'))
model2.add(Dense(num_classes, activation='softmax'))
model2.summary()
model2.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model_ = train_model(model2)
```

```
…
Epoch 20/20
469/469 [==============================] - 1s 3ms/step - loss: 0.1522 - accuracy: 0.9437 - val_loss: 0.3311 - val_accuracy: 0.8922
Test loss: 0.3311215341091156
Test accuracy: 0.8921999931335449
```

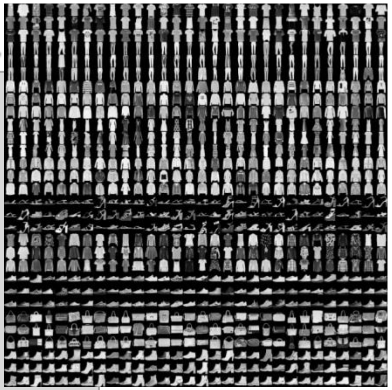
```
# Multiple Layers Without dropout
model3 = Sequential()
model3.add(Flatten(input_shape=(28,28)))
model3.add(Dense(512, activation='relu', kernel_initializer='he_normal'))
model3.add(Dense(512, activation='relu', kernel_initializer='he_normal'))
model3.add(Dense(num_classes, activation='softmax'))
model3.summary()
model3.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model_ = train_model(model3)
```

```
…
Epoch 20/20
469/469 [==============================] - 3s 7ms/step - loss: 0.1354 - accuracy: 0.9472 - val_loss: 0.3703 - val_accuracy: 0.8910
Test loss: 0.37025687098503113
Test accuracy: 0.890999972820282
```

TensorFlow 2.0

Classification

Different model performance comparison with Fashion-MNIST dataset…



```
# Multiple Layers With dropout
model4 = Sequential()
model4.add(Flatten(input_shape=(28,28)))
model4.add(Dense(512, activation='relu', kernel_initializer='he_normal'))
model4.add(Dropout(0.2))
model4.add(Dense(512, activation='relu', kernel_initializer='he_normal'))
model4.add(Dropout(0.2))
model4.add(Dense(num_classes, activation='softmax'))
model4.summary()
model4.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model_ = train_model(model4)
```

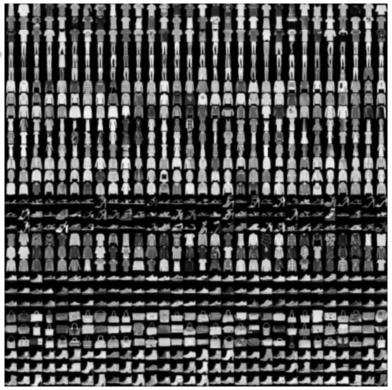
```
…
Epoch 20/20
469/469 [==============================] - 4s 8ms/step - loss: 0.2071 - accuracy: 0.9214 - val_loss: 0.3317 - val_accuracy: 0.8921
Test loss: 0.3317197263240814
Test accuracy: 0.8920999765396118
```

```
# Multiple Layers With dropout
model5 = Sequential()
model5.add(Flatten(input_shape=(28,28)))
model5.add(Dense(512, activation='relu', kernel_initializer='he_normal'))
model5.add(Dropout(0.2))
model5.add(Dense(512, activation='relu', kernel_initializer='he_normal'))
model5.add(Dropout(0.2))
model5.add(Dense(num_classes, activation='softmax'))
model5.summary()
model5.compile(loss='sparse_categorical_crossentropy', optimizer=RMSprop(), metrics=['accuracy'])
model_ = train_model(model5)
```

```
…
Epoch 20/20
469/469 [==============================] - 5s 11ms/step - loss: 0.2657 - accuracy: 0.9084 - val_loss: 0.3876 - val_accuracy: 0.8909
Test loss: 0.38759610056877136
Test accuracy: 0.8909000158309937
```

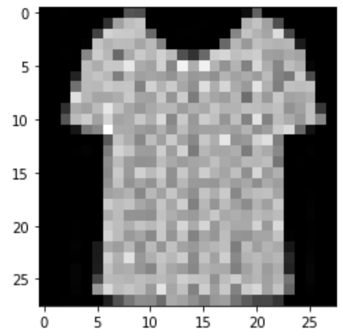
Classification

Different model performance comparison with Fashion-MNIST dataset…

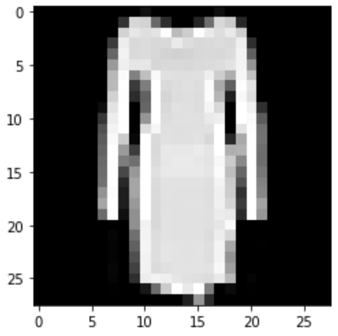


```
# saving the model
save_dir = "results/"
model_name = 'fashion_mnist.h5'
model_path = os.path.join(save_dir, model_name)
model_.save(model_path)
print('Saved trained model at %s ' % model_path)

#evaluation and prediction
model_ = load_model('results/fashion_mnist.h5')
loss_and_metrics = model_.evaluate(x_test, y_test, verbose=2)
print("Test Loss", loss_and_metrics[0])
print("Test Accuracy", loss_and_metrics[1])
```

```
is classified as: Shirt
```



```
is classified as: Dress
```



```
is classified as: Ankle boot
```