Machine Learning Python Programming Training

Kevin Lee, Genpact

Agenda

- Introduction of Machine Learning
- Machine Learning current implementation and implementation
- Machine Learning Concepts – Hypothesis function, Cost function, Gradient Descent, Learning Rate
- Machine Learning Process / Workflow
- Machine Learning Types – Supervised and Unsupervised
- Machine Learning Algorithms: SVM, KNN, Decision Tree, Random Forest, Logistic Regression, Linear Regression, K-means Clustering
- Artificial Neural Network (ANN)
- Activation Functions
- Loss function
- Optimizer
- Deep Neural Network (DNN)
- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)
- Transfer Learning

Honey, do you know about Machine Learning?

Many peopole around us expects us to know about Machine Learning. Why?

- Programming
- Statistics / Modeling
- Working with data all the times

Big Picture of Data Science, AI, Machine Learrning, Deep Learning

```
## Left aligned image
<img src="attachment:image.png" width="400" align="left">
```

- Data Science : a field of pertaining to data systems and processes, aimed at maintaining data sets and deriving meaning out of data.
- AI (Artificial Intelligence) : a filed of enabling machines to execute reasoning by replicating human intelligence.
- Machine Learning : a subset of AI that Machine/system can automatically learn and improve from experience.
- Deep Learning : a subset of Machine Learning that system uses artifical neural network to learn.

Current AI Implementation

Voice Regnition

Recommendation - Amazon, Netflix, Spotify

Chatbot (Online Customer Service)

AI Assistant (AI System for accomplishing Real-World Tasks over the phone)

Amazon Go (Casher less Grocery Store)

AlphaGO – Beat "Go" World Champion

Why is AI(ML) so popular now?

- Innovative
  - Can solve many complex business problems that were not resolved before
- New business development
  - Cost effective
- Automate a lot of works
  - Can replace or enhance human labors
  - “Pretty much anything that a normal person can do in <1 sec, we can now automate with AI” Andrew Ng
- Accurate
  - Better than humans in some applications

AI Market Expected Growth

Machine Learning Introduction

An application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed.

What is it?

How does human learn?

- Through Experiences

How does Machine learn?

- Data
- Algorithm (Hypothesis Function)

Machine Learning Functions

Machine Learning Functions
Hypothesis function

- ML Algorithm/Model that we will train with data
- Linear Regression, Logistic Regression, Support Vector Machine, Decision Tree, Artificial Neural Network
- hθ(x) = θ0* x0 + θ1* x1 + θ2* x2 + θ3* x3 +,, + θn* xn (e.g., Y = 2x + 30)
- Parameters (ultimate goals of ML) : θ0, θ1, θ2 , θ3,,,θn

Cost function (loss function)

- To measure how well hypothesis function fits into data.
- Measured Difference between actual data point and hypothesized data point. (e.g., Y – hθ(x))
- J(θ0, θ1, θ2,,θn) = (1/2m)*sum[(Y - H)^2] for linear model

Gradient Descent (optimizer)

- Engine that minimizes cost function
- Repeat
  - θ0 := θ0 – alpha* d/dθ0 * J(θ0, θ1, θ2,,θn)
  - θ1 := θ1 – alpha* d/dθ1 * J(θ0, θ1, θ2,,θn)
- Learning rate (alpha) – size of learning step in gradient descent

How Machine Learning works

Hypothesis function vs cost function vs gradient descent

Teaching Young Childrens about Cat Pictures

1. Show first sets of pictures
2. Tell kids that some are “cat” pictures and some are NOT.
3. Explain why some are NOT “cat” pictures
4. Show next sets of pictures
5. Repeat 2-3
6. Keep showing CAT pictures using step 4 and 5 until kids understand “cat” pictures.

Machine Learning Process

1. Select Data and hypothesis function
2. Start with random parameters of hypothesis function
3. Process first sets of data in hypothesis function
4. Find predicted values from hypothesis function
5. Calculate cost function (actual – predicted)
6. Minimize cost function using Gradient Descent
7. Update parameters of hypothesis function based on cost function minimization
8. Repeat steps (3 and 7) with next sets of data until the cost function become 0 or at the lowest point.

1. Select Data and Hypothesis Function

Data :

- Y = Survival Days in months
- X = number of drug exposures

Hypothesis function:

- Linear Regression ($h\theta(x) = \theta x$)

2. Start with random parameters of hypothesis function

hθ(x) = 4x at θ = 4

3 & 4 Process first set of data in hypothesis function and predict values from hypothesis function

hθ(x) = 4x at θ = 4

x = [1,2,3]

- at x = 1, y = 4
- at x = 2, y = 8
- at x = 3, y = 12

5. Calculate cost function : difference between actual and predicted

Cost function : J(θ)=(1/2m)*sum[(Y - H)^2]

J(4) =(1/6)*sum((2-4)^2 + (4-8)^2 + (6-12)^2) = 56/6 = 9.333

Cost Function in different Parameter

- J(0) =56/6 = 9.333

- J(1) =14/6 = 2.333
- J(2) =0/6 = 0
- J(3) =14/6 = 2.333
- J(4) =56/6 = 9.333

Optimum θ is 2 : Best fitted model is H = 2X.

Gradient Descent

- Definition – Engine that minimize cost function.
- The process of getting to the lowest error value (optimum cost function)
- It is like walking down in the mountain to find the gold located in valley.

6. Minimize cost function using Gradient Descent

- Take the derivative of cost function since it will give the direction to move forward : d/dθ J(θ)
- Make steps down the cost function : θ := θ – d/dθ J(θ)
- The size of steps is determined by learning rate : θ := θ – alpha* d/dθ J(θ)

Learning Rate (alpha)

At θ:= θ – alpha* d/dθJ(θ) at θ =4 assuming d/dθJ(4) = 2.

If the learning rate (alpha) = 0.1,
θ:= 4 - 0.1* 2 = 3.8
If the learning rate (alpha) = 0.5
θ:= 4 – 0.5* 2 = 3

7. Update parameters of hypothesis function based on cost function

- Initial Hypothesis function - hθ(x) = 4x
- Cost function - J(θ) = (1/2m)*sum[(Y - h)^2] = 1/(*2*3)* sum(((2-4)^2 + (4-8)^2 + (6-12))^2)) = 56/6=9.333
- Gradient Descent – θ := θ – alpha* d/dθJ(4) = 4 – 0.1*2 = 3.8
- Modified Hypothesis function - hθ(x) = 3.8x

8.Repeat steps (3 and 7) with next sets of data

At second iteration,

- Initial Hypothesis function - hθ(x) = 3.8x
- Cost function - J(θ) = 1/(*2*3) sum(((2-3.8)^2 + (4-7.6)^2 + (6-11.4)^2)) = 45.36/6 = 7.56
- Gradient Descent – θ := θ – alpha* d/dθ J(3.8) = 3.8 – 0.1*2 = 3.6
- Final Hypothesis function - hθ(x) = 3.6x

And the iteration will go on until the cost function gest to 0 like J(θ)=0 or it no longer decreases or no more data.

The cost function decreases and slope (θ) decreases as well to the optimum point.

Machine Learning Types and Algorithms

Find the right job for right jobs/problems

Machine Learning Types

Supervised

- Input data labeled : has correct answers
- Specific purpose
- Types
  - Classification for distinct output values
  - Regression for continuous output values

Unsupervised

- Input data not-labeled : No correct answers
- Exploratory
- Types : Clustering (clusters)

Supervised Machine Learning

Classification

- Categorical target

- Binary or Multiple Classification
- Example : Yes/No, 0 to 9, mild/moderate/severe
- Algorithms: SVM, K-nearest Neighbor, Naïve Bayes, Decision Tree, Random Forests, XGBoost, Logistic Regression

Support Vector Machine (SVM)

- SVM is one of the most powerful classification model (both binary & multi), especially for complex, but small/mid-sized datasets.
- Best for binary classification
- Easy to train
- To find a line/ hyper-plane (in multidimensional space that separate outs classes) that maximizes margin between different classes.
  Pros:
- It works really well with clear margin of separation.
- It works well in high dimensional spaces.
- It works well where number of dimensions is greater than the number of samples.

Cons:

- It requires higher computation power, so it does not work well with larger data.
- It also doesn't perform very well when the data set has a lot of noises.

Why Python for Machine Learning

Python is most popular / effective programming language for Machine Learning

Popular ML packages

- Scikit-Learn
  - simple and efficient ML tool, open-source
  - Classification, Regression, SVM, Clustering, Dimensionality Reduction, Decision Trees, Random Forests
- TensorFlow
  - developed by Google, open source since Nov.'2015
  - Artificial Neural Network, Convolutional NN, Recurrent NN, Autoencoder, Reinforcement Learning
- Keras

    - The official high-level API of TensorFlow
    - Easier and more friendly than TensorFlow
    - Developed by Google and contributed by MS, AWS
- Pytorch
    - Dynamic ML model
    - Good with NLP

In [35]:
```
## import libraries
import numpy as np
import pandas as pd

## algorithm spliting train and test data
from sklearn.model_selection import train_test_split

## metric algorithm
from sklearn import metrics
```

In [1]:
```
## SVM Python Programming
### Import data
from sklearn import datasets
iris = datasets.load_iris()

X = iris["data"] # sepal length, sepal width, petal length, petal width
y = iris["target"] # 'setosa', 'versicolor', 'virginica'
```

In [27]:
```
## subsetting data 25 setosa, 25 versicolor and 25 virginica
X2 = np.concatenate((X[:25], X[50:75], X[100:125]), axis=0)
y2 = np.concatenate((y[:25], y[50:75], y[100:125]))
X_train, X_test, y_train, y_test = train_test_split(X2, y2, test_size=0.2) ## splitting
```

In [28]:
```
## import modules
from sklearn.svm import SVC

## Pick up SVM Classifier model
svm_clf = SVC(kernel="linear", C=float("inf"))

## Train the model
svm_clf.fit(X_train, y_train)
```

Out[28]:
```
SVC(C=inf, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

In [33]:
```
## Predict using the trained SVM
svm_pred = svm_clf.predict((X_test))

print("Prection : ", svm_pred)
print("Test label: ", y_test)
```

```
Prection :  [2 1 2 0 0 0 1 0 2 2 2 0 1 0 0]
Test label:  [2 1 2 0 0 0 1 0 2 2 2 0 1 0 0]
```

In [38]:
```
## Evaluation of trained SVM model
metrics.accuracy_score(svm_pred, y_test)
```

Out[38]:
```
1.0
```

KNN (K Nearest Neighbors)

- A simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems.
- Non-parametric & lazy algorithm
- KNN does not train KNN algorithm from training data, but memorize the location of training data itself.

How KNN works?

1. Calculate Distance between new data and training data
2. Find the class of closest neighbors (how many A and B in a given k)

3. Vote for labels which has most votes in a given k

Pros:

- The training phase is faster
- Simple and easy to implement
- Useful for a nonlinear data

Cons:

- Slower and costlier in terms of time and memory in prediction
- Large storage to store the entire training data

In [40]:
```
#import ML algorithm
from sklearn.neighbors import KNeighborsClassifier
```

In [42]:
```
#select and train model
#n_neighbors is the number of neighbor to use
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
```

Out[42]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

In [45]:
```
#predict output
knn_pred = knn.predict(X_test)

print("Prection from KNN : ", knn_pred)
print("Test label:         ", y_test)
```

```
Prection from KNN :  [2 1 2 0 0 0 1 0 2 2 2 0 1 0 0]
Test label:          [2 1 2 0 0 0 1 0 2 2 2 0 1 0 0]
```

In [46]:
```
## Evaluation of trained SVM model
metrics.accuracy_score(svm_pred, y_test)
```

Out[46]:
```
1.0
```

Decision Trees

The widely used machine learning algorithm that use tree to visually and explicitly represent decision and its conditions.

It is used for both classification (more popular) and regression.

In [48]:
```
#import ML algorithm
from sklearn.tree import DecisionTreeClassifier
```

In [50]:
```
#select and train model
d_tree = DecisionTreeClassifier(max_depth=4)
d_tree.fit(X_train, y_train)
```

Out[50]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=4,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

In [51]:
```
#predict output
dtree_pred = d_tree.predict(X_test)

print("Prediction from Decision Tree : ", dtree_pred)
print("Test label:                     ", y_test)
```

```
Prediction from Decision Tree :  [2 1 2 0 0 0 1 0 2 2 2 0 1 0 0]
Test label:                      [2 1 2 0 0 0 1 0 2 2 2 0 1 0 0]
```

In [52]:
```
## Evaluation of trained SVM model
metrics.accuracy_score(dtree_pred, y_test)
```

Out[52]: 1.0

Random Forests

The ensemble of decision trees that uses multiple decision trees.

Steps

1. Randomly choose bootstrap sample of size n (k samples from the training data).
2. Grow decision tree from bootstrap sample (sampling random sets of observations with replacement).
3. Repeat 1 and 2 on k times
4. Aggregate the prediction by each tree and choose the predictor with the majority vote. (e.g., Class A from tree1, Class B from tree 3 …)

Why Random Forest? Can build more robust model that has a better generalization performance and less subject to overfitting.

In [53]:
```
#import ML algorithm
from sklearn.ensemble import RandomForestClassifier
```

In [54]:
```
#select and train model (500 trees)
rdf = RandomForestClassifier(n_estimators=500, max_leaf_nodes=15, n_jobs=1)
rdf.fit(X_train, y_train)
```

Out[54]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=None, max_features='auto', max_leaf_nodes=15,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=500, n_jobs=1,
                       oob_score=False, random_state=None, verbose=0,
                       warm_start=False)
```

In [55]:
```
#predict output
rdf_pred = rdf.predict(X_test)

print("Prediction from Random  Forest : ", rdf_pred)
print("Test label:                      ", y_test)
```

```
Prediction from Random  Forest :  [2 1 2 0 0 0 1 0 2 2 2 0 1 0 0]
Test label:                       [2 1 2 0 0 0 1 0 2 2 2 0 1 0 0]
```

In [56]:
```
## Evaluation of trained SVM model
metrics.accuracy_score(rdf_pred, y_test)
```

Out[56]: 1.0

Logistic Regression

- Output: Binary target (Yes/No)
- Input: Continuous/categorical variables
- Example : predicting if the tumor is malignant based on tumor size

Logistic Regression Architecture

Hypothesis function : hθ(x) = 1 / (1 + e-z) where z= (θ0x0 + θ1x1 + θ2x2 + θ3x3 +,, + θnxn)

In [61]:
```
## Prepare binary data
X_binary = X[y !=2]
y_binary = y[y !=2]

## Splitt 80 by 20
X_train_binary, X_test_binary, y_train_binary, y_test_binary = train_test_split(X_binary, y_binary, test_size=0.2)
```

In [64]:
```
print(y_train_binary)
print(np.unique(y_train_binary, return_counts=True))
```

```
[1 0 0 0 1 0 0 0 1 0 0 1 1 1 1 1 0 1 0 0 0 1 0 0 1 0 0 1 1 0 0 1 1 0 1 0 0
 1 1 1 1 0 0 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0 1 1 1 1 0 0 0 0 0 1 0 0 1 1 1 0
 0 0 0 0 1 1]
(array([0, 1]), array([43, 37], dtype=int64))
```

In [65]:
```
#import ML algorithm
from sklearn.linear_model import LogisticRegression
```

In [66]:
```
#select and train model
lr = LogisticRegression()
lr.fit(X_train_binary, y_train_binary)
```

```
C:\Users\kevin\Anaconda3\envs\nlp\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default
solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

Out[66]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [68]:
```
#predict output
lr_pred = lr.predict(X_test_binary)

print("Prediction from Random  Forest : ", lr_pred)
print("Test label:                      ", y_test_binary)
```

```
Prediction from Random  Forest :  [1 1 1 0 0 1 1 1 1 0 1 1 1 0 1 0 1 1 0 0]
Test label:                       [1 1 1 0 0 1 1 1 1 0 1 1 1 0 1 0 1 1 0 0]
```

In [69]:
```
## Evaluation of trained SVM model
metrics.accuracy_score(rdf_pred, y_test)
```

Out[69]:
```
1.0
```

Regression

- To predict values of a desired target quantity when the target quantity is continuous.
- Output: Continuous variables
- Example : predicting house price per sqft
- Types: Linear Regression, Polynomial Regression, Decision Tree, Random Forest
- Hypothesis function:
  - hθ(x) = θ0x0 + θ1x1 + θ2x2 + θ3x3 +,, + θnxn
  - y = 10 + 2x
- Accuracy of Regression model
  - Square Error(SE): sum (y – pred)2
  - Mean Square Error (MSE) = (1/n)*SE
  - Root Mean Square Error (RMSE) = square root of MSE

In [74]:
```
## Preparing Lineal Regression data
X_linear = 2 * np.random.rand(100, 1)
y_linear = 4 + 3 * X_linear + np.random.randn(100, 1)
```

In [75]:
```
## Visualization of data
import matplotlib.pyplot as plt
%matplotlib inline

plt.scatter(X_linear, y_linear)
```

Out[75]:
```
<matplotlib.collections.PathCollection at 0x2550fbdc7b8>
```

In [76]:
```
## train and test data
X_train_linear, X_test_linear, y_train_linear, y_test_linear = train_test_split(X_linear, y_linear, test_size=0.2,
```

In [73]:
```
#import ML algorithm
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
```

In [77]:
```
#select and train model
linear = linear_model.LinearRegression()
linear.fit(X_train_linear, y_train_linear)
```

Out[77]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [80]:
```
#predict output
linear_pred = linear.predict(X_test_linear)

print("Prediction from Linear Regression : ", linear_pred.flatten())
print("Test label:                        ", y_test_linear.flatten())
```

```
Prediction from Linear Regression :  [ 6.60365759  5.35296545  6.99639136  7.3412966  10.17593911  9.14253255
  9.1602244   8.17130242  7.24875195  4.28438168  9.42021552 10.19721714
  6.26456404  6.57181408  6.49733174  9.26672637  8.32237061 10.07214698
  7.62016634  8.97343297]
Test label:                          [ 7.18711927  5.05061573  8.35940688  8.17784623  9.58126395  9.05657975
  8.50195982  8.80108391  7.44436507  5.3059028   9.80108251  9.12998923
  5.05971546  7.50788212  5.65235103  8.33722549  9.6126728  10.60188593
  7.90921293  8.14640463]
```

In [81]:
```
## Evaluation of trained SVM model
mean_squared_error(linear_pred, y_test_binary)
```

Out[81]: 55.123367533915754

Unsupervised Machine Learning

- Not-labeled input data : no correct answers
- Exploratory
- Clustering : the assignment of set of observations into subsets (clusters) so that data in the same clusters are more similar to each other than to those from different clusters.
- Algorithms : k-means
- Industry implementation - the grouping of documents, music, movies by different topics, finding customers that share similar - interests based on common purchase behaviors.
  - Customer segmentation
  - Document clustering
  - Image segmentation
  - Recommendation engines

k-means Clustering

- The simplest and popular unsupervised machine learning algorithm
- To group similar data points together and discover underlying patterns. To achieve this objective, K-means looks for a fixed number (k) of clusters in a dataset.
- the K-means algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible.

How it works

1. Randomly pick k centroids as initial cluster centers.
2. Assign each sample to the nearest centroid.
3. Move the centroids to the center of the samples that were assigned to it.
4. Repeat 2 and 3 until the cluster assignments do not change under a tolerance.

In [82]:
```
#import ML algorithm
from sklearn.cluster import KMeans
```

In [83]:
```
# build k means model with 2 clusters
kmeans = KMeans(n_clusters = 3, tol=0.0001, max_iter=100)
kmeans.fit(X)
```

Out[83]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=100,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

In [1]:
```
#print the results of the value of centroids
print(kmeans.cluster_centers_) # centroids

#print the results of the value of centroids
print("prediction from Kmeans: /n", kmeans.labels_)
print("labels:                 /n", y)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-146735bc6833> in <module>
      1 #print the results of the value of centroids
----> 2 print(kmeans.cluster_centers_) # centroids
      3
      4 #print the results of the value of centroids
      5 print("prediction from Kmeans: /n", kmeans.labels_)

NameError: name 'kmeans' is not defined
```

Artificial Neural Network

- Most powerful ML algorithm
- Game Changer
- Works very similar to human brain – Neural Network
- Types – DNN, CNN, RNN
- Architecture
  - Artificial Neurons
  - Input Layers
  - Hidden Layers
  - Output Layers

Human Neural Network vs Artificial Neural Network (ANN)

Human Neurons : Neural Network – 100 billions

Artificial Neurons : Artificial Neural Networks

Artificial Neuron

- Input (x1, x2, x3,,,,xn) – numeric values that goes into neuron model
- Weight (w1, w2, w3,,,,wn) – weight of each input values
- Bias (b1) – bias of (input * weight)
- Artificial neuron
  - Model (input values * weights + bias): z1 = x1* w1 + x2* w2 + x3* w3 +,,,, + xn* wn + b1

- Activation function : a1 = f(z1)

What does "Artificial Neuron" learn?

Machine Learning – To learn parameters

- Weights (w1, w2, w3, w4)
- Bias (b1)

How "Artificial Neuron" learns

Cost function of ANN

- actual output – predicted value
- J(w ,b) = 1/m(sum [ℓ(yi – ai)]) Gradient descent/optimization
- w := w – alpha*dJ/dw
- b := b – alpha*dJ/ db

Python Codes for Deep Neural Network

In [87]:
```
## Import keras and sklearn ML libraries
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
```

DNN architecture design

In [88]:
```
## DNN Architecture Design for above
model_dnn = Sequential()
model_dnn.add(Dense(4, input_dim=3, activation='relu'))
model_dnn.add(Dense(4, activation='relu'))
model_dnn.add(Dense(1, activation='sigmoid'))

model_dnn.summary()
```

```
WARNING:tensorflow:From C:\Users\kevin\Anaconda3\envs\nlp\lib\site-packages\tensorflow\python\framework\op_def_li
brary.py:263: colocate_with (from tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 4)                 16
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 20
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 5
=================================================================
Total params: 41
Trainable params: 41
Non-trainable params: 0
_________________________________________________________________
```

Python Codes for Deep Neural Network

In [86]:
```
## Import keras and sklearn ML libraries
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
```

```
Using TensorFlow backend.
```

In [ ]:
```

```

DNN architecture design

In [37]:
```
## DNN Architecture Design
model_dnn = Sequential()
model_dnn.add(Dense(4, input_dim=3, activation='relu'))
model_dnn.add(Dense(4, activation='relu'))
model_dnn.add(Dense(1, activation='sigmoid'))

model_dnn.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 4)                 16
_________________________________________________________________
dense_11 (Dense)             (None, 4)                 20
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 5
=================================================================
Total params: 41
Trainable params: 41
Non-trainable params: 0
_________________________________________________________________
```

Activation Functions

- Introduction – a node that is added to the output of NNs. Sometime, it could be added between two NNs.
- Purpose – to provide boundary of outputs.

Type

- Linear function
- Sigmoid or Logistic function
- Tanh or Hyperbolic Tangent function
- ReLU (Rectified Linear Unit) function
- Softmax function

Linear function

- f(x) = ax + b
- Ex. Regression

Sigmoid or Logistic function

- f(x) = 1 / (1 + e-x)
- Ex. Logistic Regression, DNN, CNN
- Purpose – to predict the probability as an output since it exists between 0 and 1

Tanh or Hyperbolic Tangent function

- f(x) = tahn(x) = 2 / (1 + e-2x) – 1
- Ex. RNN
- mainly used for classification between two classes
- Stronger gradients for Tanh for Sigmoid

ReLU (Rectified Linear Unit) function

- f(x) = 0 for x < 0 ; x for x>= 0
- Ex – CNN, DNN
- One of the most used activation function. Usually used between NNs
- Speeding up the computation

Softmax function

- f(xi) = e-xi / sum(e-xi)
- Activation function that will turn numbers to probabilities that sum to 1.
- Usage – multiple classification, NLP, RNN
- To provide the probability of a given output. The model will select the best probability.
- Text Sensitivity Analysis – read texts and predict the mood of text

Examples of Activation Function

- Hypothesis function : hθ(x) = 1 / (1 + e-z)
  - where z= (2x0 + x1 + 5x2 + 6x3 +0.5x4)
- Plug the parameters
  - Calculate hθ(x) using Activation Function if x = (0.1, 2, 0.2, 0.4) assuming x0 = 1
  - z = 2* 1 + 1* 0.1 + 5* 2 + 6* 0.2 + 0.5* 0.4 = 13.5
- Z = 13.5
  - Linear: hθ(x) = 13.5
  - Sigmoid: hθ(x) = 1/(1 + e-13.5) = 0.999998629
  - Tanh: hθ(x) = 2/(1 + e-13.5* 2) -1 =1
  - ReLU: hθ(x) = 13.5
- For Z = - 0.4
  - Linear: hθ(x) = - 0.4
  - Sigmoid: hθ(x) = 1/(1 + e0.4) = 0.401312
  - Tanh: hθ(x) = 2/(1 + e0.4* 2) -1 = - 0.37995
  - ReLU: hθ(x) = 0

```
Python codes for Activation Function

model = Sequential()
model.add(Dense(4, input_dim=3, activation='relu'))
model.add(Dense(2, activation='relu'))

model.add(Dense(2, activation='softmax'))
```

Loss Function in Machine Learning

- Loss function is a method that evaluate how well your algorithm models/fits your dataset.
  - If the prediction of the model is off a lot, the loss function performs a higher number.
  - If the prediction of the model is good, the loss function performs a lower number.
- Types:
  - Regressive loss function
  - Classification loss function

Regressive Loss Function

- For continuous target variables
- Usually activation function is regression.

Mean Square Error(MSE)

the average of the squared differences between the predicted and actual value

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

Mean Absolute Error(MAE)

the sum of absolute differences between our target and predicted variables

$$\text{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

Python code Example for regressive loss function :

model.compile(loss='mse', optimizer=Adam(), metrics=['accuracy'])

Classification Loss Function

For Classification target variables

Binary classification

- 1 or 0
- One node with sigmoid activation function
- Binary Cross Entropy loss function
- Example code:
  - model.compile(loss='binary_crossentropy', optimizer=Adam(), metrics=['accuracy'])

Multi classification

- Multiple targets : 1-9, mild/moderate/severe
- One hot encoding Implementation
- One node for each class resulting from softmax activation function
- Muliti Cross Entropy loss function
- Example code:
  - model.compile(loss='categorical_crossentropy', optimizer=Adam(), metrics=['accuracy'])

Optimizer in Machine Learning

- The goals
  - to minimize the cost/loss function.
  - to calculate and change the weights of model so that error from cost function becomes lower on each iteration.
- The Algorithm of walking down in the mountain to find the gold located in valley.
- Main parameter – learning rate (how much walking down for each step)

Optimizer types

Python codes examples for optimizer

- model.compile(loss='mse', optimizer=Adam(), metrics=['accuracy'])
- model.compile(loss='mse', optimizer=SGD(lr=0.0001), metrics=['accuracy'])
- model.compile(loss='mse', optimizer=RMSprop(lr=0.0001), metrics=['accuracy'])

Deep Neural Network (DNN) / Deep Learning

- DNN is the aggregates of individual artificial neurons.
- Basic architecture - Input layer, Hidden layer, Output layer.
- Every layer is made up of a set of neurons, which each layer is fully connected to all neurons in the layer before

Tensorflow DNN Playground

http://playground.tensorflow.org (http://playground.tensorflow.org)

In [95]:
```
###
print(X_train_binary.shape)
print(y_train_binary.shape)
print(y_train_binary)
```

```
(80, 4)
(80,)
[1 0 0 0 1 0 0 0 1 0 0 1 1 1 1 1 0 1 0 0 0 1 0 0 1 0 0 1 1 0 0 1 1 0 1 0 0
 1 1 1 1 0 0 1 1 0 0 0 1 0 1 1 0 0 1 0 1 0 1 1 1 1 0 0 0 0 0 1 0 0 1 1 1 0
 0 0 0 0 1 1]
```

In [105]:
```
## DNN Architecture Design
model_dnn2 = Sequential()
model_dnn2.add(Dense(4, input_dim=4, activation='relu'))
model_dnn2.add(Dense(2, activation='relu'))
model_dnn2.add(Dense(1, activation='sigmoid'))

model_dnn2.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_13 (Dense)             (None, 4)                 20
_________________________________________________________________
dense_14 (Dense)             (None, 2)                 10
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 3
=================================================================
Total params: 33
Trainable params: 33
Non-trainable params: 0
_________________________________________________________________
```

In [106]:
```
## Compile
model_dnn2.compile(loss='binary_crossentropy', optimizer=Adam(), metrics=['accuracy'])

## Train
#model_dnn2.fit(X_train_binary, y_train_binary, epochs=100, batch_size=40)
model_dnn2.fit(X_train_binary, y_train_binary, epochs=10)
```

```
Epoch 1/10
80/80 [==============================] - 1s 12ms/step - loss: 0.6639 - acc: 1.0000
Epoch 2/10
80/80 [==============================] - 0s 75us/step - loss: 0.6591 - acc: 1.0000
Epoch 3/10
80/80 [==============================] - 0s 150us/step - loss: 0.6544 - acc: 1.0000
Epoch 4/10
80/80 [==============================] - 0s 150us/step - loss: 0.6505 - acc: 1.0000
Epoch 5/10
80/80 [==============================] - 0s 175us/step - loss: 0.6468 - acc: 1.0000
Epoch 6/10
80/80 [==============================] - 0s 187us/step - loss: 0.6432 - acc: 1.0000
Epoch 7/10
80/80 [==============================] - 0s 202us/step - loss: 0.6397 - acc: 1.0000
Epoch 8/10
80/80 [==============================] - 0s 113us/step - loss: 0.6364 - acc: 1.0000
Epoch 9/10
80/80 [==============================] - 0s 113us/step - loss: 0.6331 - acc: 1.0000
Epoch 10/10
80/80 [==============================] - 0s 122us/step - loss: 0.6298 - acc: 1.0000
```

Out[106]:
```
<keras.callbacks.History at 0x25515756b38>
```

In [112]:
```
## Prediction
#dnn_pred = model_dnn2.predict(X_test_binary)

dnn_pred2 = model_dnn2.predict_classes(X_test_binary)

#print the results of the value of centroids
#print("prediction from Kmeans: ", dnn_pred.flatten())
print("prediction from Kmeans(classes): ", dnn_pred2.flatten())
print("labels:                          ", y_test_binary)
```

```
prediction from Kmeans(classes):  [1 1 1 0 0 1 1 1 1 0 1 1 1 0 1 0 1 1 0 0]
labels:                           [1 1 1 0 0 1 1 1 1 0 1 1 1 0 1 0 1 1 0 0]
```

In [113]:
```
## Evaluation - loss function and accuracy0
model_dnn2.evaluate(X_test_binary, y_test_binary)
```

```
20/20 [==============================] - 0s 9ms/step
```

Out[113]:
```
[0.6051263809204102, 1.0]
```

Convolutional Neural Network (CNN)

- Image-specific Artificial Neural Network – image search/recognition, face recognition, face verification, self-driving and more
- Its concept comes from brain visual cortex where many neurons have a small region of the visual field (in CNN, filter)
- The name of CNN comes from “convolution”, one of the most important operation in CNN.
- The first Convolutional Neural Network is LeNet-5, which can classify digits from hand-written number.
- Advantage compared to DNN
  - 2 or 3 dimensional read (image rather than vector)
  - Less parameters, much less computing power

Image Data

When the computer see the image, it sees pixel values.

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  64 191 118  24 118 138 191 118  17   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  17 193 253 252 252 252 252 253 252 227 120   5   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  93 252 253 252 233 141  69  79 227 252 252 137   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0 187 252 253 252  89   0   0   0  29 154 252 252  95   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0 207 252 253 157   6   0   0   0   0   7 158 252 220   0   0   0   0   0   0]
 [  0   0   0   0   0   0  70 253 253 231   0   0   0   0   0   0   0  43 241 255 154   0   0   0   0   0]
 [  0   0   0   0   0   0  70 252 252 230   0   0   0   0   0   0   0   0 135 253 206   0   0   0   0   0]
 [  0   0   0   0   0   0 101 252 252 167   0   0   0   0   0   0   0   0   0 222 240  50   0   0   0   0]
 [  0   0   0   0   0   0 184 252 252   0   0   0   0   0   0   0   0   0   0 138 252 111   0   0   0   0]
 [  0   0   0   0   0   0 184 252 147   0   0   0   0   0   0   0   0   0   0 138 252 183   0   0   0   0]
 [  0   0   0   0   0   0 185 253 253   0   0   0   0   0   0   0   0   0   0 139 253 184   0   0   0   0]
 [  0   0   0   0   0   0 184 252 200   0   0   0   0   0   0   0   0   0   0 138 252 183   0   0   0   0]
 [  0   0   0   0   0   0 101 252 252   0   0   0   0   0   0   0   0   0   0 138 252 183   0   0   0   0]
 [  0   0   0   0   0   0  44 236 252  95   0   0   0   0   0   0   0   0   0 233 252  89   0   0   0   0]
 [  0   0   0   0   0   0   0 207 252 220   0   0   0   0   0   0   0   0 158 253 210   6   0   0   0   0]
 [  0   0   0   0   0   0   0 145 253 244  65   7   0   0   0   0  62 243 252 135   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  34 217 253 252 154  30   0   0   9  78 236 252 157   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  84 222 252 252 227 184 185 197 252 252 221  32   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  36 177 252 252 252 253 252 252 176  35   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   5 107 179 252 190 137  54   4   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]]
```

CNN Architecture

1. Input layer (4 dimensional vector)
2. Convolution layer– Finding local pattern
3. Pooling layer
4. Flatten
5. Fully-connected layers – Finding global pattern
6. Output layer

1. Input layer (4 dimensional vector)

- Import image (.jpg) to numeric number - (28,28) for black/white or (28,28,3) for color
- If needed, convert image data into 3 dimensional vectors : (28,28) to (28,28,1)
- Convert to 4 dimension vector by adding number of images - 60,000 images to 60,000 numeric vector of (60000, 28,28,1)

2. Convolutions

Convolution – the mathematical combination of two functions to produce a third function.

- Filters (Kernel Matrix) are image feature identifiers (e.g., straight edges and curves)
- Input Image * Kernel
  - If high number, input data has filter(kernel) image.
  - If low number, input data does NOT have filter(kernel) image.

3. Pooling

Pooling with filters of 2 X 2

- Max Pooling – more popular than average pooling

- Average Pooling : (-5 -4 -10 -2) / 4 = -21/4= -5.25

Pooling progressively reduces the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.

4. Flatten Layer - Architecture

The flattening section of CNN architecture will flatten a pooled layer to one column so that the data will be passed through ANN to get processed further. The neuron of flatten layer detects a certain feature like nose, mouth,,,
In this CNN architecture, pooling layer of (4,4,20) will be flattened to flatten layer of (4x4x20=320), which means that it has 320 image features.

5. Fully-connected layer

The fully-connected layer section of CNN has three layers.

- Flatten layer
- Fully-connected layer
- Output layer

Fully-connected layers in CNN works as follows

- The neurons in fully-connected layer detect a certain feature of the image like nose, mouth or eyes

- It combines the features into a wider variety of attributes that makes CNN more capable for classifying the images

6. Output Layer

The output layer section outputs an N (e.g., 10) dimensional vector where N is the number of classes that the program has to choose from.

For example,

- The resulting vector for a digit classification program is [0,.1, .1, 0, 0, 0.75, 0, 0, 0, .05] for number 0 to 9
- It represents a 10% probability that the image is “1”, a 10% probability that the image is “2”, a 75% probability that the image is “5”, and a 5% probability that the image is “9”.
- Model will predict the image is “5”.

How Image features are progressed throughout CNN architecture

Famous CNN Architecture competition winner

- AlexNet (developed in 2012)
  - Famous winner of the 2012 ImageNet LSVRC-2012 competition by a large margin (15.3% VS 26.2% (second place) error rates)
- ZFNet (in 2013)
  - Top 5 error rate: 14.8%
- GoogLeNet(19) - 2014
  - Top 5 error rate: 6.67%
  - Number of Parameters : 4 million
- VGG Net(16) - 2014
  - Top 5 error rate: 6.67%
  - Number of Parameters : 4 million
- ResNet(152) - 2015
  - Top 5 error rate: 3.6%

Convolution Python Programm

In [114]:
```
## Keras for DNN
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
from keras.callbacks import EarlyStopping
from keras.optimizers import Adam
```

Below is CNN acchitecture that the data scientist builds for the workshop.

- Input layer : 28 by 28 pixel images
- First Convolution : 64 (3 by 3) filters
- First Pooling : (2 by 2) Maximum value pooling
- Second Convolution : 128 (3 by 3) filters
- Second Pooling : (2 by 2) Maximum value pooling
- Third Convolution : 256 (3 by 3) filters
- Third Pooling : (2 by 2) Maximum value pooling
- First Dense Layer : 128 neurons
- Output Layer : binary (0 or 1)

In [116]:
```
## Buiid CNN model

model_cnn = Sequential()

## First Convolution and Pooling
model_cnn.add(Conv2D(64, kernel_size=(3, 3), activation='relu', input_shape=(28,28,1)))
model_cnn.add(MaxPooling2D(pool_size=(2, 2)))

## Second Convolution and Pooling
model_cnn.add(Conv2D(128, kernel_size=(3, 3), activation='relu', padding='same'))
model_cnn.add(MaxPooling2D(pool_size=(2, 2)))

## Third Convolution and Pooling
model_cnn.add(Conv2D(256, kernel_size=(3, 3), activation='relu', padding='same'))
model_cnn.add(MaxPooling2D(pool_size=(2, 2)))

## Flatten Layer
model_cnn.add(Flatten())

## Fully-Connected Layer
model_cnn.add(Dense(128, activation='relu'))
model_cnn.add(Dropout(0.5))

## Output Layer
model_cnn.add(Dense(2, activation='softmax'))

## Compile CNN model
model_cnn.compile(loss=['categorical_crossentropy'], optimizer='adam', metrics=['accuracy'])

model_cnn.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 26, 26, 64)        640
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 13, 13, 128)       73856
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 6, 6, 256)         295168
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 3, 3, 256)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_18 (Dense)             (None, 128)               295040
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_19 (Dense)             (None, 2)                 258
=================================================================
Total params: 664,962
Trainable params: 664,962
Non-trainable params: 0
_________________________________________________________________
```

```
## Train the model
model_cnn.fit(X_tain, y_train, batch_size=128, epoch=10, validation_split=0.2)

## Predict with the model
cnn_preds = model_cnn.predict(X_test)

## Evaluate the trained model
model_cnn.evaluate(X_test, y_test)
```

CNN Implementation

- One of the most popular DNN
- Go-to-model for all the image-related problems
- Image Classification
- X-ray diagnosis

- Face Recognition
- Face Verification
- Object detection
- Autonomous vehicles

Recurrent Neural Network

Introduction – recurrent neural network model to use sequential information.

Why RNN?

- In traditional ANN, all inputs and outputs are independent of each other. But, in some case, they could be dependent.
- RNN is useful when inputs are dependent.
- Some problems such as text analysis and translation, we need to understand which words come before.
- RNN has a memory which captures previous information about what has been calculated so far.

Basic RNN Structure and Algorithms

$a^{<t>} = g(Waa*a^{<t-1>} + Wax*X^{<t>} + ba)$ when g is an activation function (e.g., tahn, ReLU)
$Y^{<t>} = g(Wya*a^{<t>} + by)$ when g is an activation function (e.g., sigmoid, softmax)

Note that $a^{<t>}$ includes the information from previous X.

RNN unit - LSTM (Long Short-Term Memory Unit)

- It is composed of 4 gates – input, forget, gate and output.
- LSTM remembers values over arbitrary time intervals and the 3 gates regulate the flow of information into and out of LSTM unit.
- LSTMs were developed to deal with the vanishing gradient problems.
- Relative insensitivity to gap length is an advantage of LSTM over RNNs.

Popular RNN Architecture

Text notation. # of inputs = # of outputs

Sentimental Analysis (PV signal) : x = text, y = 0/1 or 1 to 5

Music generation \ Picture Description: x= vector, y = text

Machine translation : x = text in English, y = text in French

Natural Language Processing (NLP) using RNN

Introduction of NLP – An area of artificial intelligence on an interaction between computer and human natural language. NLP can program computers to process and analyze natural language data.

- Input data – Language
- Output data - Language

Simple RNN architecture using NLP

- Input data – “I am smiling”, “I laugh now”, “I am crying”, “I feel good”, “I am not sure now”
- Embedding – to convert words to vector number
- LSTM – to learn language
- Softmax – to provide probability of output
- Output data - “very unhappy”, “unhappy”, “happy”, “very happy”

Natural Language Processing (NLP) procedures

1. Import data and preparation
2. Tokenizing – representing each word to numeric integer number : "the" to 50
3. Padding – fixing all the records to the same length
4. Embedding – representing word(numeric number) to vectors of numbers
   5o to [0.418, 0.24968, -0.41242, 0.1217, 0.34527, -0.044457, -0.49688, -0.17862, -0.00066023,,,,,]
5. Training with RNN models

1. Import Data and Preparation

- Import document to working area
- Convert document to sentences
- Split sentence to words
- Filter out punctuation
- Convert words to lower case

In [117]:
```
## Python Programming for importing word preparation
from keras.preprocessing.text import text_to_word_sequence

texts = ['I do not like her.', 'I love her']
line =[ ]
for text in texts:
    words = text_to_word_sequence(text)
    line.append(words)
print(line)
```

```
[['i', 'do', 'not', 'like', 'her'], ['i', 'love', 'her']]
```

2. Tokenization

- Tokens - words
- Tokenizing - Representing word(numeric number) to integer number
- Word "the" to 1
- Words are represented by dense vectors where a vector represents the projection of the word into a continuous vector space

In [118]:
```
## Python Programming for importing word preparation
from keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer()
tokenizer.fit_on_texts(line)
tokenized_words = tokenizer.texts_to_sequences(line)
print(tokenized_words)
```

```
[[1, 3, 4, 5, 2], [1, 6, 2]]
```

3. Padding

- Padding – Preparing tokenized input data into the same length so that input data can go into Embedding layer
- It will make all the records to fixed, consistent length.

In [119]:
```
from keras.preprocessing.sequence import pad_sequences
padded_lines = pad_sequences(tokenized_words , maxlen=5, padding='post')
print(padded_lines )
```

```
[[1 3 4 5 2]
 [1 6 2 0 0]]
```

4. Embedding

- Representing word(numeric number) to vectors of numbers
  Word “the” to [0.418, 0.24968, -0.41242, 0.1217, 0.34527, -0.044457, -0.49688, -0.17862, -0.00066023,,,,,]
- Words are represented by dense vectors where a vector represents the projection of the word into a continuous vector space
- Embedding layer has embedding matrix which will be trained as a part of model.
- The shape of embedding matrix is (vocabulary size, # of vector)
- Embedding layer types
- Keras Embedding layer – Random initiation on embedding matrix
- Pre-trained embedding methods
  - Word2Vec
  - GloVe

In [120]:
```
## import gensim model
from gensim.models import Word2Vec
```

In [121]:
```
## creat NLP model using gensim for embeding
model_NLP = Word2Vec(line, min_count=1)
print(model_NLP.wv.vocab)
print(line)
print(model_NLP['i'])
print(model_NLP['i'].shape)
```

```
{'i': <gensim.models.keyedvectors.Vocab object at 0x000002551B7784A8>, 'do': <gensim.models.keyedvectors.Vocab ob
ject at 0x000002551B7784E0>, 'not': <gensim.models.keyedvectors.Vocab object at 0x000002551B778518>, 'like': <gen
sim.models.keyedvectors.Vocab object at 0x000002551B778550>, 'her': <gensim.models.keyedvectors.Vocab object at 0
x000002551B778588>, 'love': <gensim.models.keyedvectors.Vocab object at 0x000002551B7785C0>}
[['i', 'do', 'not', 'like', 'her'], ['i', 'love', 'her']]
[ 0.00108667  0.00228122  0.00416964 -0.00212642  0.00176383  0.00457667
 -0.00246628  0.00039715  0.0017894  -0.00166932 -0.0025599  -0.00469731
 -0.00293878 -0.00332033 -0.00314927  0.00212122  0.0019047   0.00329956
  0.00315692  0.00171956 -0.00031396  0.00479963  0.00211227 -0.00199327
 -0.0019977  -0.00112897  0.00193015 -0.00276268 -0.0007386   0.00195724
 -0.00107421 -0.00486515 -0.00498707 -0.00340522 -0.00463967  0.00479073
  0.00374083 -0.00390519  0.00496752 -0.00179126 -0.00321203 -0.00014797
 -0.00176685 -0.00248731  0.00195765  0.00173438 -0.00478459 -0.00425575
 -0.00103728 -0.00322562 -0.00101246 -0.00169303 -0.00116279  0.00345117
 -0.00090775 -0.00186304 -0.00162541  0.00173169 -0.0047279  -0.00344206
 -0.00210958  0.00259895 -0.00470207 -0.00169391  0.00194483 -0.00433963
 -0.00087211  0.00396639 -0.00337992 -0.00483181  0.00366324  0.00410477
  0.00040189  0.00133792  0.00314862  0.00484033 -0.0038453  -0.00261557
  0.00301613 -0.00349523  0.00148091  0.00360574 -0.00317985  0.00087573
  0.00462336 -0.00011017  0.00107908 -0.00309016 -0.00309494 -0.0025392
 -0.00323007 -0.00425579  0.00101917 -0.00187253  0.0025162  -0.00326393
  0.00104248  0.00496165 -0.00281944  0.00491631]
(100,)
```

```
C:\Users\kevin\Anaconda3\envs\nlp\lib\site-packages\ipykernel_launcher.py:5: DeprecationWarning: Call to deprecat
ed `__getitem__` (Method will be removed in 4.0.0, use self.wv.__getitem__() instead).
  """
C:\Users\kevin\Anaconda3\envs\nlp\lib\site-packages\ipykernel_launcher.py:6: DeprecationWarning: Call to deprecat
ed `__getitem__` (Method will be removed in 4.0.0, use self.wv.__getitem__() instead).
```

5. Train with RNN model

In [124]:
```
## import RNN model
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from keras.layers import Dense, LSTM, Embedding
```

tokenizer = Tokenizer() tokenizer.fit_on_texts(X_train_rnn) X_input = tokenizer.texts_to_sequence(X_train_rnn)

In [128]:
```
model_rnn = Sequential()
model_rnn.add(Embedding(input_dim=5000, output_dim=100, input_length=50))
model_rnn.add(LSTM(200))
model_rnn.add(Dense(200, activation='relu'))
model_rnn.add(Dense(4, activation='softmax'))

model_rnn.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model_rnn.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 50, 100)           500000
_________________________________________________________________
lstm_4 (LSTM)                (None, 200)               240800
_________________________________________________________________
dense_22 (Dense)             (None, 200)               40200
_________________________________________________________________
dense_23 (Dense)             (None, 4)                 804
=================================================================
Total params: 781,804
Trainable params: 781,804
Non-trainable params: 0
_________________________________________________________________
```

```
## Train RNN model with
model_rnn.fit(X_train_rnn, y_train_rnn, epochs=50)

## Predict with the trained RNN model
rnn_preds = model_rnn.predict(X_test_rnn)

## Evaluate RNN model
model_rnn.evaluate(X_test_rnn, y_test_rnn)
```

RNN Implementation

- Speech Recognition
- Music Generation
- Sentiment Classification
- Machine Translation
- Video Activity Recognition
- Texts Classification and Extraction
- DNA Sequence Analysis
- Chatbot

Who will learn faster Python programming

- SAS Programmers
- Novice Programmers

Why experience programmers can learn other or new language faster?

Transfer Learning

Machine Learning method where a pre-trained model is reused as a starting point of the model development for another/similar task.

How model learns in Image Recognition

- Early layers – Edges Detection
- Middle layers – Shapes Detection
- Later layers – High Level Features Detection

Pre-trained Model : VGG

- Architecture – 16 layers or 19 layers
- CNN model
- 14,000,000 images with 1000 class objects
- 138,000,000 parameters
- With NVIDIA Titan Black GPU, training takes a week.
- Available in public

Transfer Learning Implementation

1. Select pre-trained model
2. Import pre-trained model
3. Determine which parameters/layers of the pre-trained model should be trained
4. Add more layers of the imported pre-trained model
5. Set up the output layers
6. Prepare data features (input data) for pre-trained model
7. Compile pre-trained and added layers
8. Train the new model

Why Transfer Learning?

- Training Faster
- A lack of data
- Cost-effective. It is expensive to train the complex CNN/RNN model. The most complex CNN/RNN model will take weeks/months to train using hundreds of GPU/TPU equipped computers.

In [129]:
```
## import pre-trained model
from keras.applications import vgg16
from keras.layers.core import Dense, Flatten
from keras.models import Model, Sequential
```

In [130]:
```
## import VGG16 model for transfer learning
input_shape= (224, 224, 3)
vgg_model = vgg16.VGG16(include_top=False, weights='imagenet', input_shape=input_shape)
```

VGG16 model in Keras

```
vgg16_model = Sequential()

vgg16_model.add(Conv2D(64, (3, 3), input_shape=input_shape, padding='same', activation='relu'))
vgg16_model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
vgg16_model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

vgg16_model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
vgg16_model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
vgg16_model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

vgg16_model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
vgg16_model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
vgg16_model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
vgg16_model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

vgg16_model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
vgg16_model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
vgg16_model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
vgg16_model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

vgg16_model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
vgg16_model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
vgg16_model.add(Conv2D(512, (3, 3), activation='relu', padding='same'))
vgg16_model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

vgg16_model.add(Flatten())
vgg16_model.add(Dense(4096, activation='relu'))
vgg16_model.add(Dense(4096, activation='relu'))
vgg16_model.add(Dense(1000, activation='softmax'))
```

In [131]:
```
## Summary of pre-trained VGG model
vgg_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
```

In [133]:
```
## add custom layers to pre-trained VGG model
X = vgg_model.output
X1 = Flatten()(X)
X2 = Dense(1024, activation='relu')(X1)
X3 = Dense(1024, activation='relu')(X2)
Target = Dense(15, activation='softmax')(X3)

model_vgg_tf = Model(input=vgg_model.input, output=Target)

model_vgg_tf.compile(loss="categorical_crossentropy", optimizer='adam', metrics=['accuracy'])
model_vgg_tf.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten_4 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_27 (Dense)             (None, 1024)              25691136
_________________________________________________________________
dense_28 (Dense)             (None, 1024)              1049600
_________________________________________________________________
dense_29 (Dense)             (None, 15)                15375
=================================================================
Total params: 41,470,799
Trainable params: 41,470,799
Non-trainable params: 0
_________________________________________________________________

C:\Users\kevin\Anaconda3\envs\nlp\lib\site-packages\ipykernel_launcher.py:8: UserWarning: Update your `Model` cal
l to the Keras 2 API: `Model(inputs=Tensor("in..., outputs=Tensor("de...)`
```

```
## Train the model
model_vgg_tf.fit(X_tain_tf, y_train_tf, batch_size=128, epoch=10, validation_split=0.2)

## Predict with the model
vgg_tf_preds = model_vgg_tf.predict(X_test_tf)

## Evaluate the trained model
```

```
model_vgg_tf.evaluate(X_test_tf, y_test_tf)
```

In [134]:
```
## add custom layers to pre-trained VGG model
## Build the same model with the different method
model_vgg_tf2 = Sequential()
model_vgg_tf2.add(vgg_model)
model_vgg_tf2.add(Flatten())
model_vgg_tf2.add(Dense(1024, activation='relu'))
model_vgg_tf2.add(Dense(1024, activation='relu'))
model_vgg_tf2.add(Dense(15, activation='softmax'))

model_vgg_tf2.compile(loss=['categorical_crossentropy'], optimizer='adam', metrics=['accuracy'])
model_vgg_tf2.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten_5 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_30 (Dense)             (None, 1024)              25691136
_________________________________________________________________
dense_31 (Dense)             (None, 1024)              1049600
_________________________________________________________________
dense_32 (Dense)             (None, 15)                15375
=================================================================
Total params: 41,470,799
Trainable params: 41,470,799
Non-trainable params: 0
_________________________________________________________________
```

CONCLUSION

- Machine Learning Introduction
- How Machine Learning works
- Machine Learning Types and algorithms
- Deep Learning Introduction - CNN, RNN and Transfer Learning

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please contact at:
Kevin Lee
AVP of Machine Learning and AI Consultant
Kevin.lee@genpact.com (mailto:Kevin.lee@genpact.com)
kevin.lee.ext@boehringer-ingelheim.com (mailto:kevin.lee.ext@boehringer-ingelheim.com)
Tweet: @HelloKevinLee
LinkedIn: www.linkedin.com/in/HelloKevinLee/ (http://www.linkedin.com/in/HelloKevinLee/)

Questions

In [ ]: