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

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 [1]:
```
## Import keras and sklearn ML libraries
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
```

```
Using TensorFlow backend.
```

DNN architecture design

In [2]:
```
## DNN Architecture Design for above
## Input Layer   : 3 features
## Hiden Layer 1 : 4 Neurons
## Hiden Layer 2 : 4 Neurons
## Output Layer  : 0/1
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()
```

```
Model: "sequential_1"
_________________________________________________________________
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
_________________________________________________________________
```

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 [4]:
```
## import libraries
import numpy as np
import pandas as pd

## data visualization
import seaborn as sns
import matplotlib.pyplot as plt

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

## metric algorithm
from sklearn import metrics
```

In [5]:
```
## 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 [6]:
```
## 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 [7]:
```
###
print(X_train_binary.shape)
print(y_train_binary.shape)
print(y_train_binary)
```

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

In [8]:
```
## 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()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 4)                 20
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 10
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 3
=================================================================
Total params: 33
Trainable params: 33
Non-trainable params: 0
_________________________________________________________________
```

In [9]:
```
## 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 [==============================] - 0s 6ms/step - loss: 0.5769 - accuracy: 1.0000
Epoch 2/10
80/80 [==============================] - 0s 75us/step - loss: 0.5650 - accuracy: 1.0000
Epoch 3/10
80/80 [==============================] - 0s 87us/step - loss: 0.5536 - accuracy: 1.0000
Epoch 4/10
80/80 [==============================] - 0s 100us/step - loss: 0.5426 - accuracy: 1.0000
Epoch 5/10
80/80 [==============================] - 0s 100us/step - loss: 0.5317 - accuracy: 0.9750
Epoch 6/10
80/80 [==============================] - 0s 100us/step - loss: 0.5222 - accuracy: 0.9750
Epoch 7/10
80/80 [==============================] - 0s 87us/step - loss: 0.5127 - accuracy: 0.9750
Epoch 8/10
80/80 [==============================] - 0s 87us/step - loss: 0.5049 - accuracy: 0.9375
Epoch 9/10
80/80 [==============================] - 0s 100us/step - loss: 0.4979 - accuracy: 0.9125
Epoch 10/10
80/80 [==============================] - 0s 63us/step - loss: 0.4921 - accuracy: 0.8625
```

Out[9]:
```
<keras.callbacks.callbacks.History at 0x2c7b7455d68>
```

In [10]:
```
## 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 Deep Neural Network (classes): ", dnn_pred2.flatten())
print("labels:                                        ", y_test_binary)
```

```
prediction from Deep Neural Network (classes):  [0 1 1 0 0 1 0 1 0 1 0 1 1 1 1 1 1 0 1 1]
labels:                                         [0 1 0 0 0 1 0 1 0 1 0 0 1 1 0 1 1 0 1 1]
```

In [11]:
```
## Evaluation - loss function and accuracy0
model_dnn2.evaluate(X_test_binary, y_test_binary)
```

```
20/20 [==============================] - 0s 3ms/step
```

Out[11]:
```
[0.47636157274246216, 0.8500000238418579]
```

In [12]:
```
## Weight Parameters of DNN
weights = []

for layer in model_dnn2.layers:
    weight = layer.get_weights()
    weights.append(weight)

print("1st layer in model: ", len(weights[0]))
print("           weights: ", weights[0][0].shape)
print("           bias   : ", weights[0][1].shape)
print("2nd layer in model: ", len(weights[1]))
print("           weights: ", weights[1][0].shape)
print("           bias   : ", weights[1][1].shape)
print("3rd layer in model: ", len(weights[2]))
print("           weights: ", weights[2][0].shape)
print("           bias   : ", weights[2][1].shape)
```

```
1st layer in model:  2
           weights:  (4, 4)
           bias   :  (4,)
2nd layer in model:  2
           weights:  (4, 2)
           bias   :  (2,)
3rd layer in model:  2
           weights:  (2, 1)
           bias   :  (1,)
```

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   0  62 243 252 135   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 [13]:
```
## 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 achitecture 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 [14]:
```
## 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()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 64)        640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 13, 13, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 6, 6, 256)         295168
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 3, 3, 256)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               295040
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_8 (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 [15]:
```
## 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 [16]:
```
## 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 [17]:
```
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 [18]:
```
## import gensim model
from gensim.models import Word2Vec
```

In [19]:
```
## 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 0x000002C7AC4C5E10>, 'do': <gensim.models.keyedvectors.Vocab ob
ject at 0x000002C7AC4C5E80>, 'not': <gensim.models.keyedvectors.Vocab object at 0x000002C7BAC09390>, 'like': <gen
sim.models.keyedvectors.Vocab object at 0x000002C7BAC09518>, 'her': <gensim.models.keyedvectors.Vocab object at 0
x000002C7BAC09550>, 'love': <gensim.models.keyedvectors.Vocab object at 0x000002C7BAC09588>}
[['i', 'do', 'not', 'like', 'her'], ['i', 'love', 'her']]
[ 7.4280443e-04  3.2413856e-03  3.6032405e-03  2.8725807e-03
 -5.4042460e-04 -4.8689907e-03  2.1361690e-03  4.6974053e-03
 -1.1813007e-04  2.9026761e-03  3.0069775e-03  3.8300273e-03
  2.9024796e-03 -1.2849547e-03  1.1393602e-03  1.5652094e-03
  3.4335775e-03 -2.9674191e-03  3.6721402e-03 -2.6195198e-03
 -2.0329803e-03 -2.8221021e-04 -3.2852534e-03 -2.8181409e-03
  4.6398710e-03 -3.0831508e-03  6.5802422e-04  3.1203912e-03
  4.2622024e-03 -9.6891861e-04 -3.8999536e-03 -4.2834682e-03
 -1.2413448e-03  4.3786177e-03  3.0153214e-03  3.4713554e-03
  2.5882302e-03 -1.8858478e-03 -2.9310242e-03  1.8994365e-03
 -2.9839387e-03  4.1621947e-03  4.5129596e-03  2.5519568e-03
  1.3013338e-03 -7.6278846e-04  2.6538372e-03  1.9639027e-03
  3.7294475e-04  4.1285767e-03  8.6818135e-04 -2.8264641e-03
 -1.1623367e-03 -3.5358255e-03  4.4695581e-03  1.6692658e-04
  3.1678970e-03  1.1035359e-03  3.3168271e-03 -3.2156724e-03
 -9.4591574e-05 -1.5175407e-03 -2.4163916e-03 -4.4971043e-03
  4.8462814e-03 -3.7004440e-03  2.0799567e-03 -4.2970567e-03
 -4.0316768e-03 -4.8687300e-03 -3.5237920e-04  1.7451816e-03
 -2.3859185e-03  1.9417240e-03 -4.9138837e-03 -4.6502776e-03
  6.2879670e-04  9.9665631e-06 -2.0883530e-03 -3.9851042e-03
  2.4263684e-03 -1.7609072e-03 -1.8505133e-03  4.4076834e-03
 -3.3763803e-03  1.4864124e-03 -2.7493187e-03 -2.8639501e-03
  3.7011057e-03 -4.6742023e-03  1.4059769e-04 -6.9748744e-04
 -3.9695394e-03 -2.1735358e-03  1.7927841e-03 -4.2625694e-03
  3.6657567e-03 -1.7554016e-03  3.3009786e-03 -2.6657728e-03]
(100,)

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

5. Train with RNN model

In [20]:
```
## 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 [21]:
```
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()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 50, 100)           500000
_________________________________________________________________
lstm_1 (LSTM)                (None, 200)               240800
_________________________________________________________________
dense_9 (Dense)              (None, 200)               40200
_________________________________________________________________
dense_10 (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 [22]:
```
## import pre-trained model
from keras.applications import vgg16
from keras.layers.core import Dense, Flatten
from keras.models import Model, Sequential
```

In [23]:
```
## 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 [24]:
```
## Summary of pre-trained VGG model
vgg_model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
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 [25]:
```
## 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()
```

```
Model: "model_1"
_________________________________________________________________
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_2 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_11 (Dense)             (None, 1024)              25691136
_________________________________________________________________
dense_12 (Dense)             (None, 1024)              1049600
_________________________________________________________________
dense_13 (Dense)             (None, 15)                15375
=================================================================
Total params: 41,470,799
Trainable params: 41,470,799
Non-trainable params: 0
_________________________________________________________________

C:\Users\kevin\Anaconda3\lib\site-packages\ipykernel_launcher.py:8: UserWarning: Update your `Model` call 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 [26]:
```
## 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()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten_3 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_14 (Dense)             (None, 1024)              25691136
_________________________________________________________________
dense_15 (Dense)             (None, 1024)              1049600
_________________________________________________________________
dense_16 (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 [ ]: