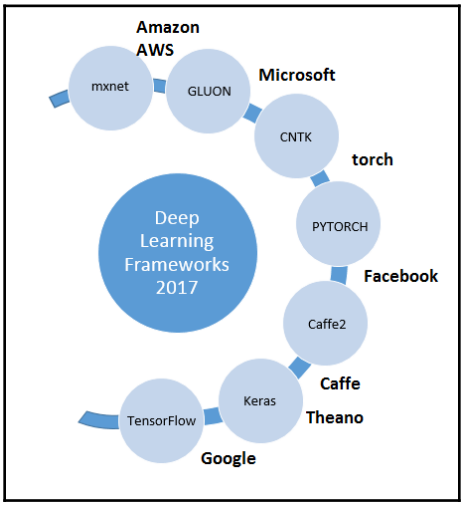
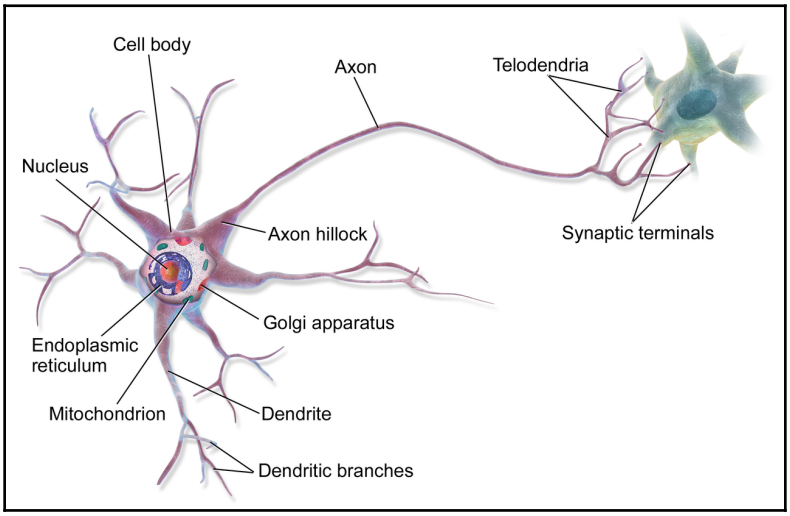
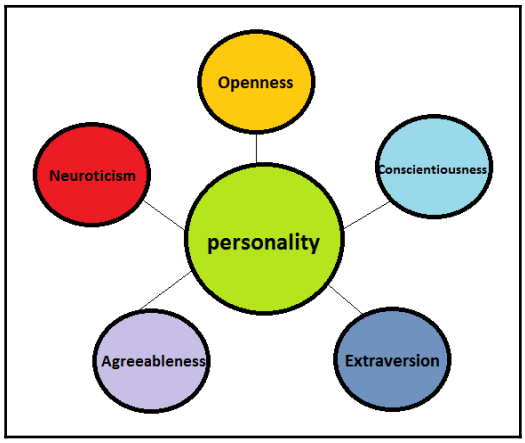
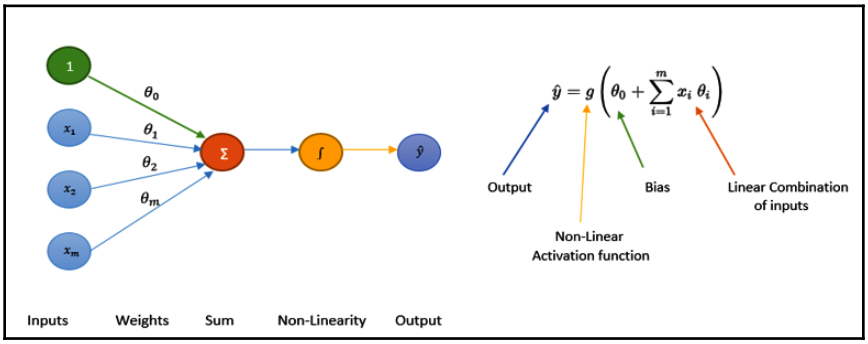
Chapter 01: Overview of Neural Networks



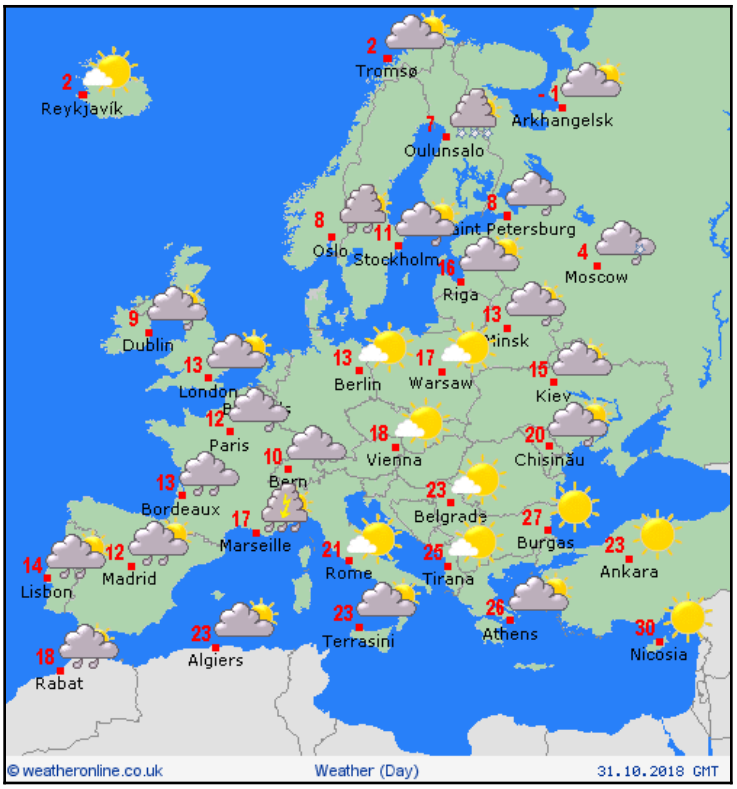


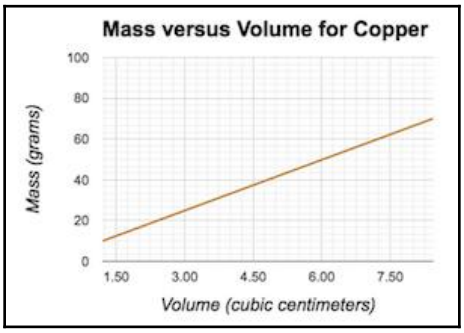

Openness
Neuroticism
Conscientiousness
personality
Agreeableness
Extraversion

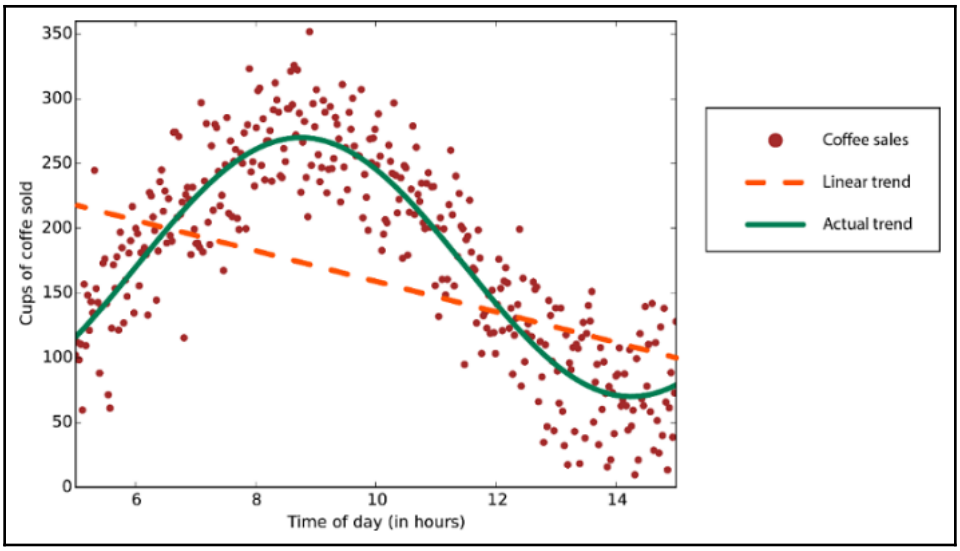
Chapter 02: A Deeper Dive into Neural Networks



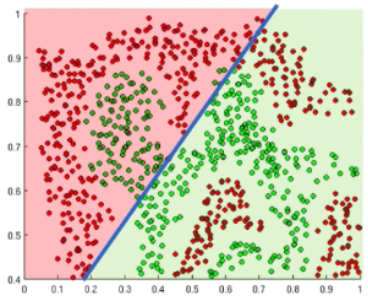
$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ . \\ . \\ . \\ x_n \end{bmatrix} \qquad x^T = [x_1\ x_2\ x_3 \ldots x_n]$$


2
Reykjavík
2
Tromsø
1
Arkhangelsk
7
Oulunsalo
8
Oslo
11
Stockholm
8
Saint Petersburg
4
Moscow
16
Riga
13
Minsk
9
Dublin
13
London
13
Berlin
17
Warsaw
15
Kiev
12
Paris
18
Vienna
20
Chisinau
10
Bern
13
Bordeaux
23
Belgrade
27
Burgas
17
Marseille
21
Rome
26
Tirana
23
Ankara
14
Lisbon
12
Madrid
23
Terrasini
26
Athens
30
Nicosia
23
Algiers
18
Rabat
© weatheronline.co.uk
Weather (Day)
31.10.2018 GMT

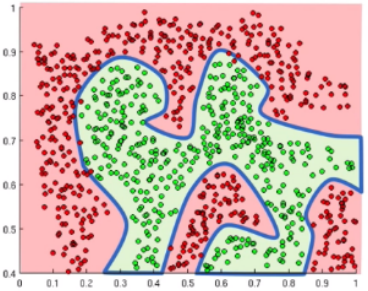

Mass versus Volume for Copper
Mass (grams)
100
80
60
40
20
0
1.50
3.00
4.50
6.00
7.50
Volume (cubic centimeters)



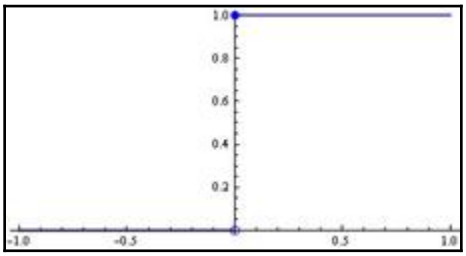
The purpose of activation functions is to ***introduce non-linearities*** *into the network*



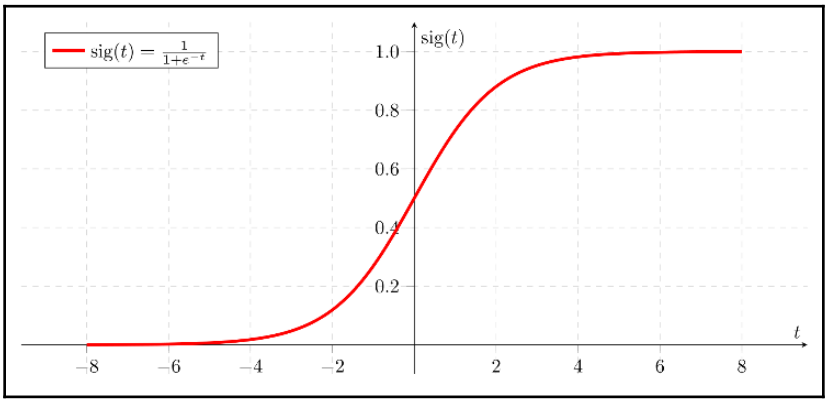
Linear Activation functions produce linear decisions no matter the network size

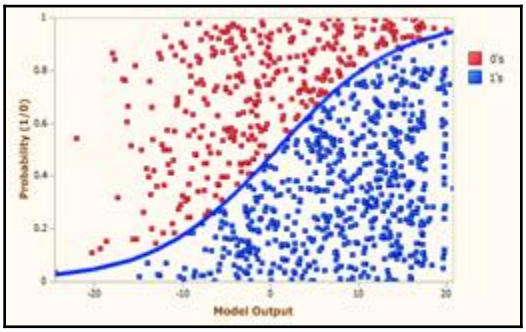


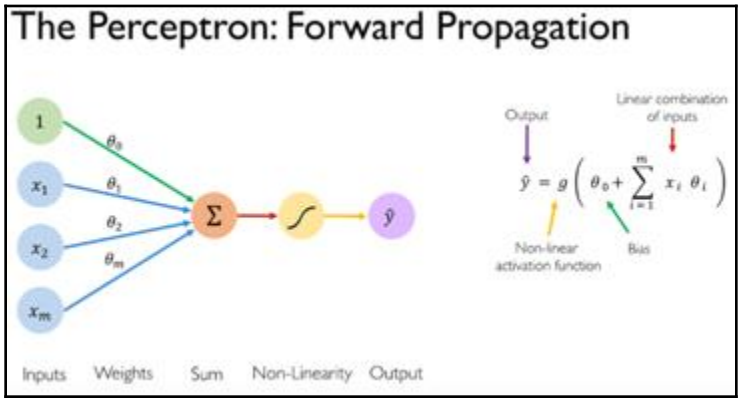
Non-linearities allow us to approximate arbitrarily complex functions

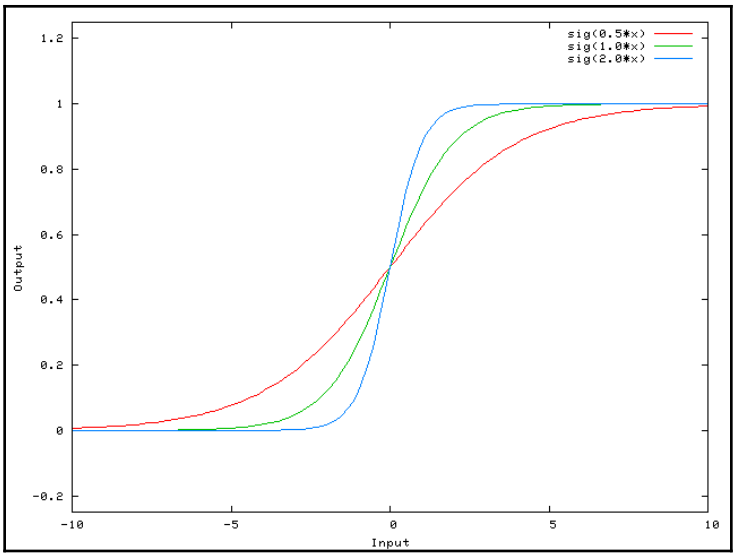


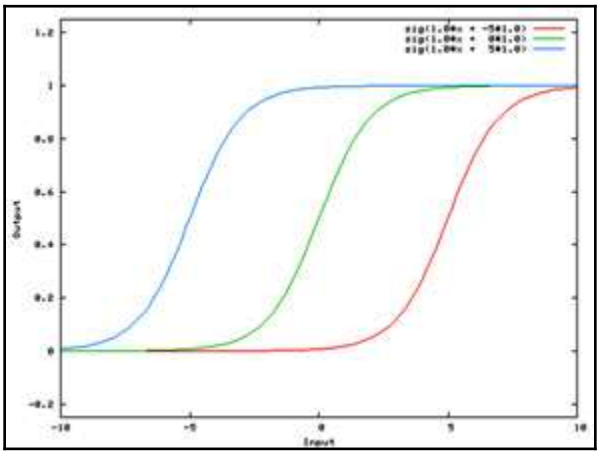
$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$







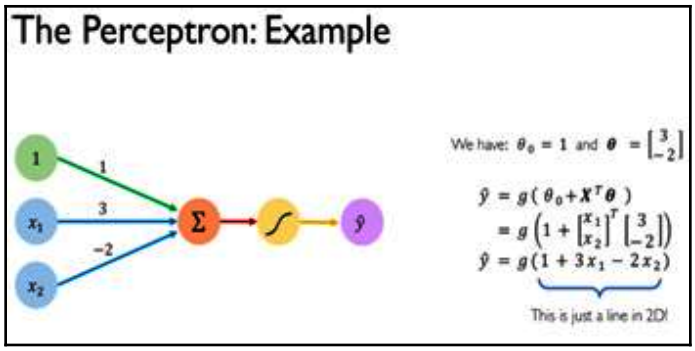


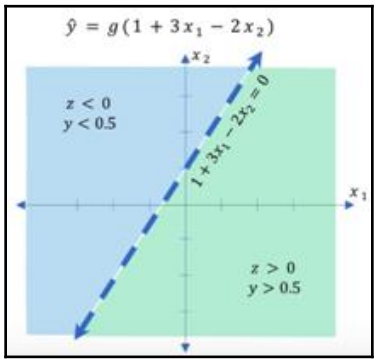


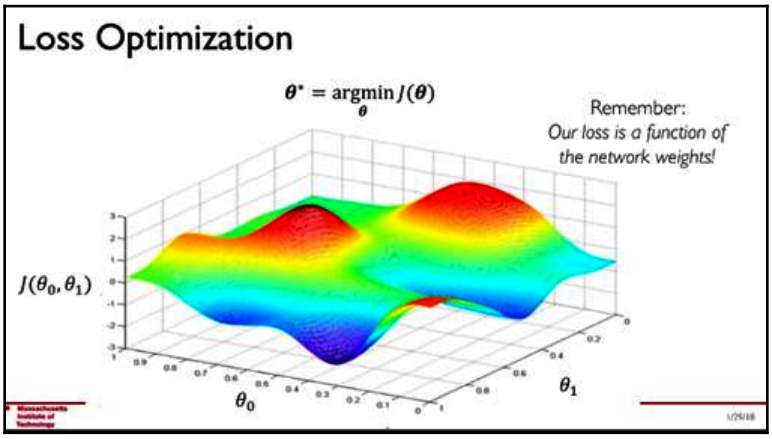
$$\hat{y} = g\left(\theta_0 + \sum_{i=1}^{m} x_i\,\theta_i\right)$$

$$\hat{y} = g\left(\theta_0 + \boldsymbol{X}^T\boldsymbol{\theta}\right)$$

where: $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}$

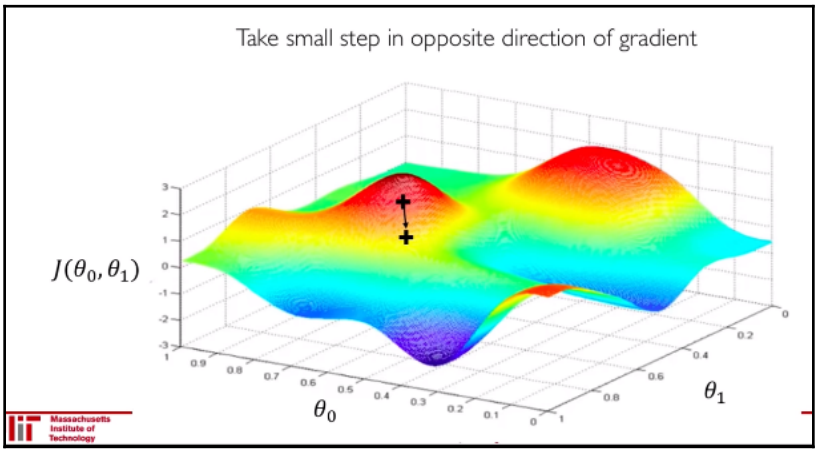

The Perceptron: Example
1
x_1
x_2
1
3
−2
Σ
$\hat{y}$
We have: $\theta_0 = 1$ and $\boldsymbol{\theta} = \begin{bmatrix} 3 \\ -2 \end{bmatrix}$
$\hat{y} = g(\theta_0 + \boldsymbol{X}^T\boldsymbol{\theta})$
$= g\left(1 + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T \begin{bmatrix} 3 \\ -2 \end{bmatrix}\right)$
$\hat{y} = g(1 + 3x_1 - 2x_2)$
This is just a line in 2D!

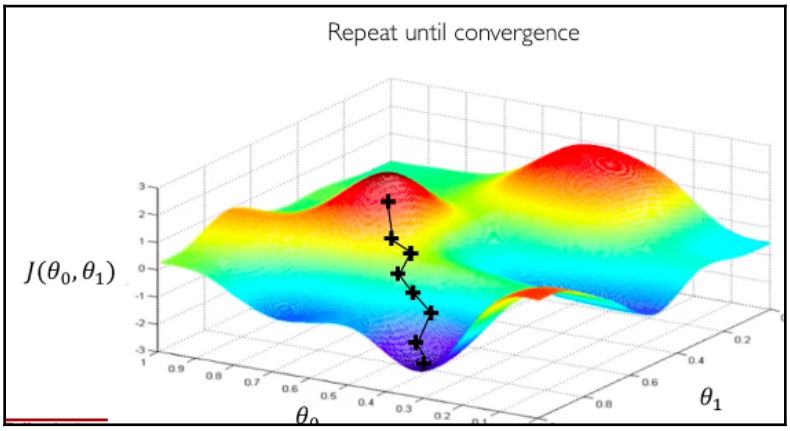

$\hat{y} = g(1 + 3x_1 - 2x_2)$
x_2
x_1
$z < 0$
$y < 0.5$
$z > 0$
$y > 0.5$
$1 + 3x_1 - 2x_2 = 0$

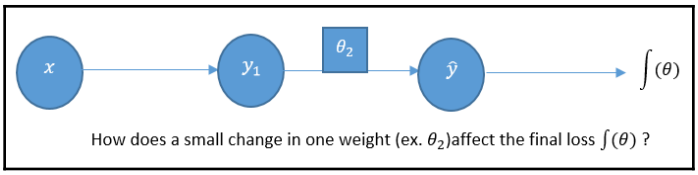

Loss Optimization
$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\text{argmin}} J(\boldsymbol{\theta})$
Remember:
Our loss is a function of the network weights!
$J(\theta_0, \theta_1)$
θ_0
θ_1

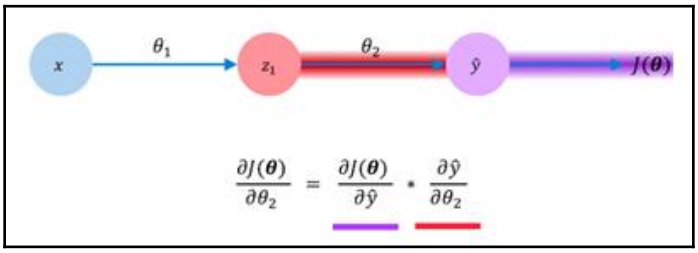
Algorithm

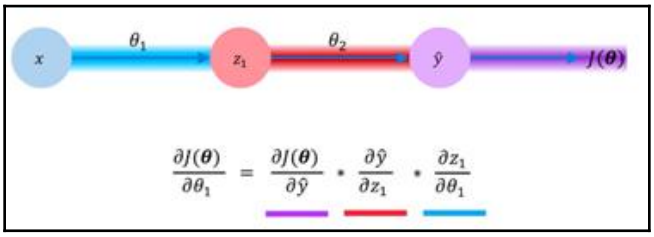
1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\theta)}{\partial \theta}$
4. Update weights, $\theta \leftarrow \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$
5. Return weights

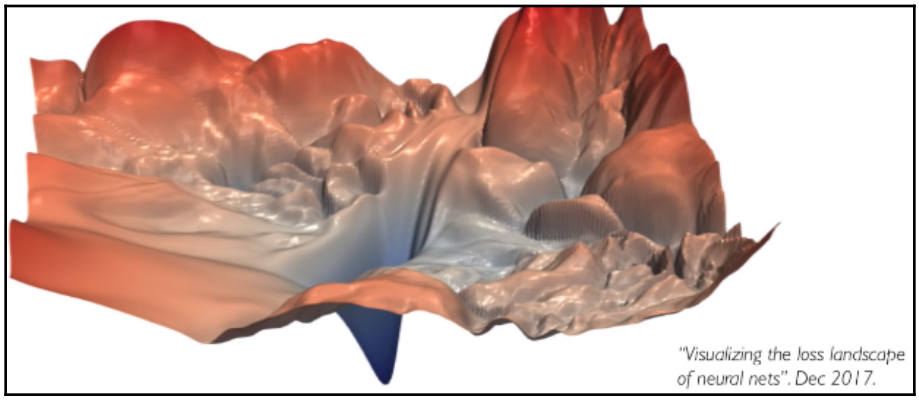






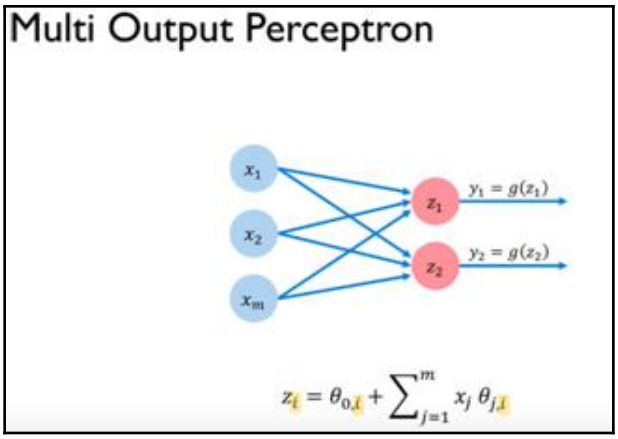


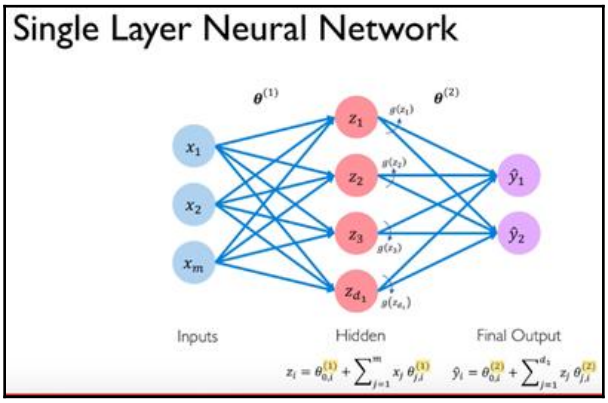


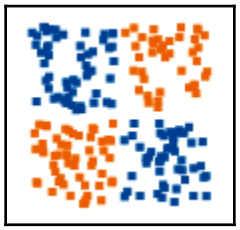


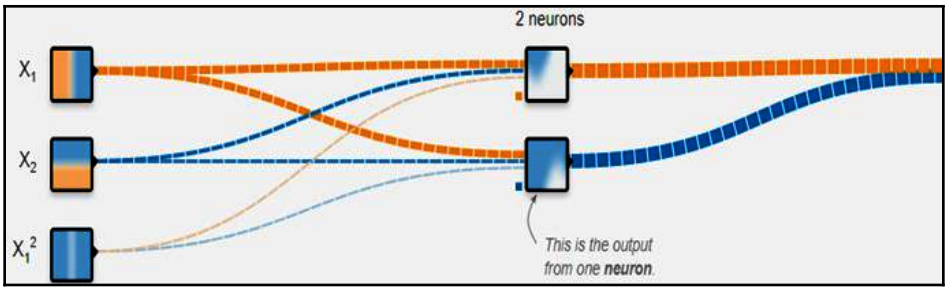
$$\theta \leftarrow \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$$

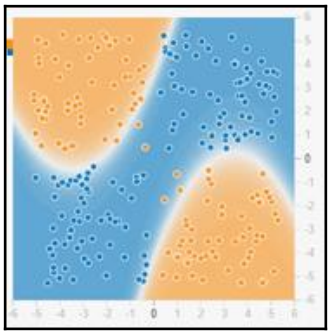
How can we set the learning rate?

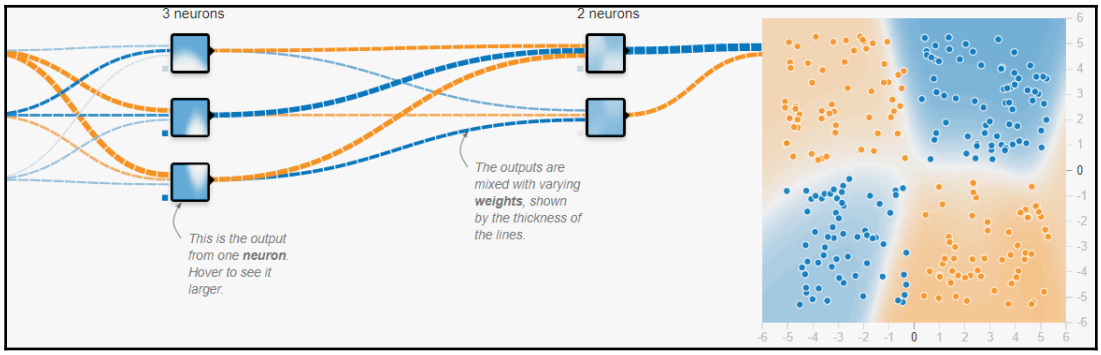

Multi Output Perceptron
x_1
x_2
x_m
z_1
z_2
$y_1 = g(z_1)$
$y_2 = g(z_2)$
$z_i = \theta_{0,i} + \sum_{j=1}^{m} x_j \theta_{j,i}$

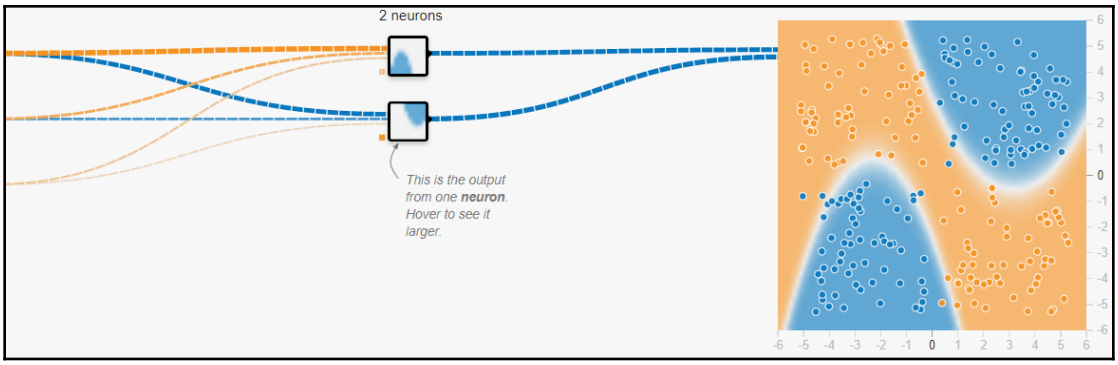

Single Layer Neural Network
$\theta^{(1)}$
$\theta^{(2)}$
x_1
x_2
x_m
z_1
z_2
z_3
z_{d_1}
$\hat{y}_1$
$\hat{y}_2$
Inputs
Hidden
Final Output
$z_i = \theta_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \theta_{j,i}^{(1)}$
$\hat{y}_i = \theta_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j \theta_{j,i}^{(2)}$

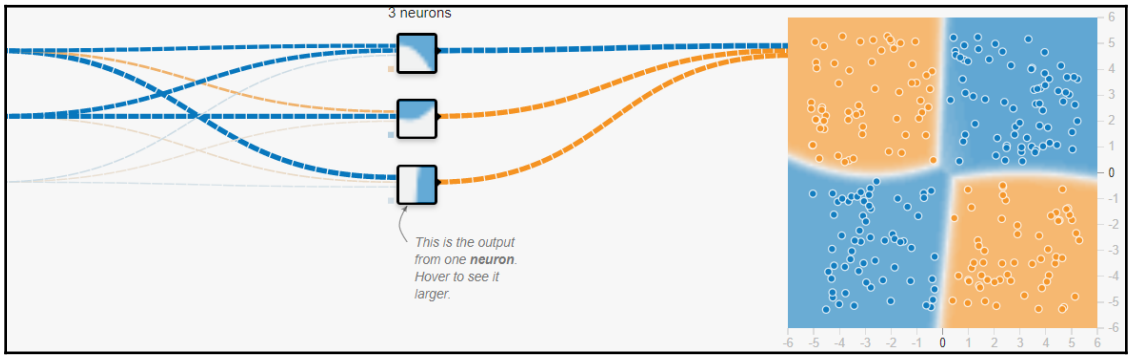


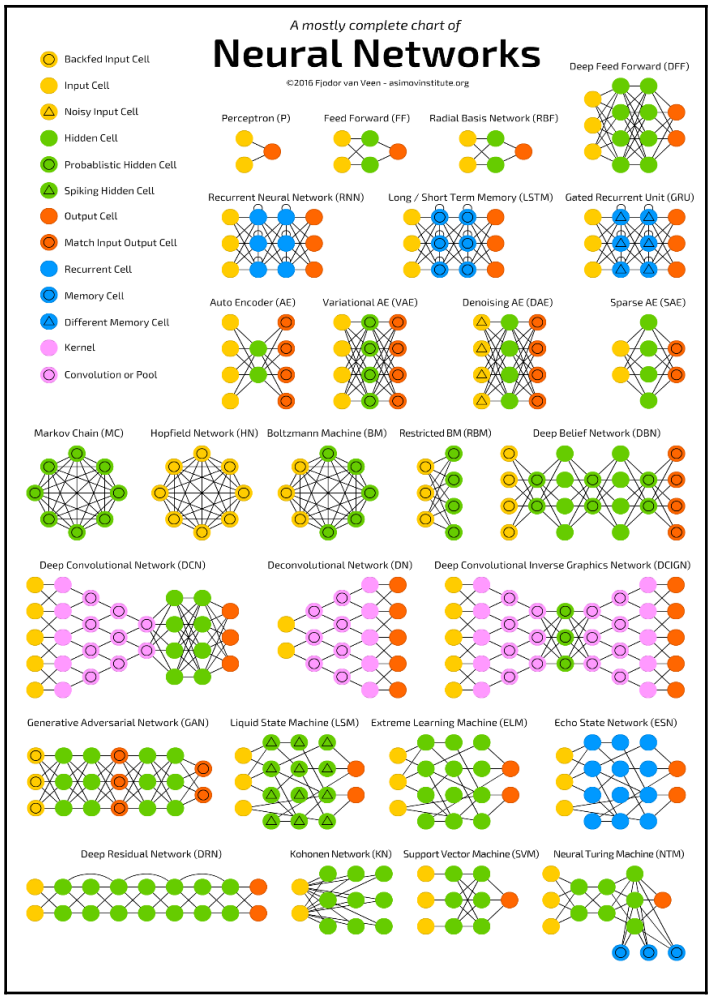

2 neurons
X_1
X_2
X_1^2
This is the output from one **neuron**.



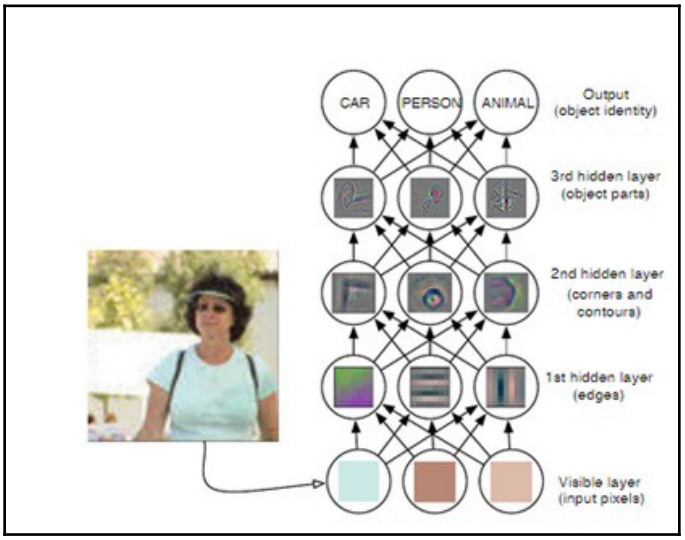

3 neurons
2 neurons
This is the output from one **neuron**. Hover to see it larger.
The outputs are mixed with varying **weights**, shown by the thickness of the lines.

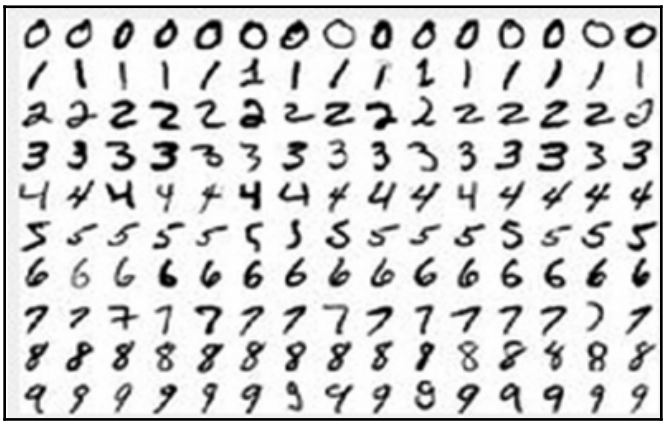

2 neurons
This is the output from one neuron. Hover to see it larger.
6
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6

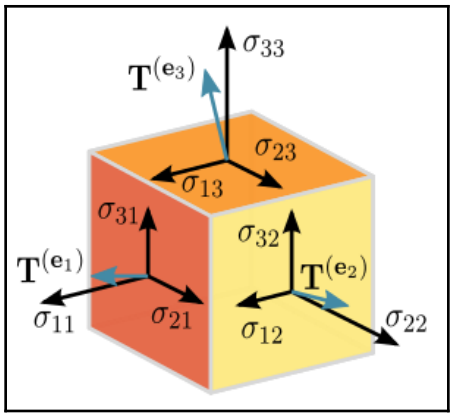

3 neurons
This is the output from one neuron. Hover to see it larger.
6
5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6

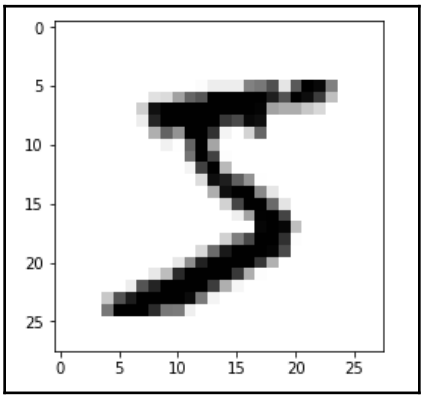

A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

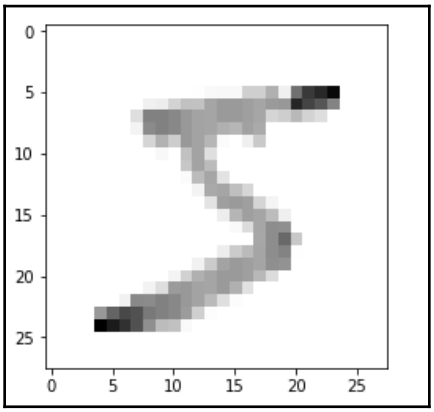
Chapter 03: Signal Processing - Data Analysis with Neural Networks

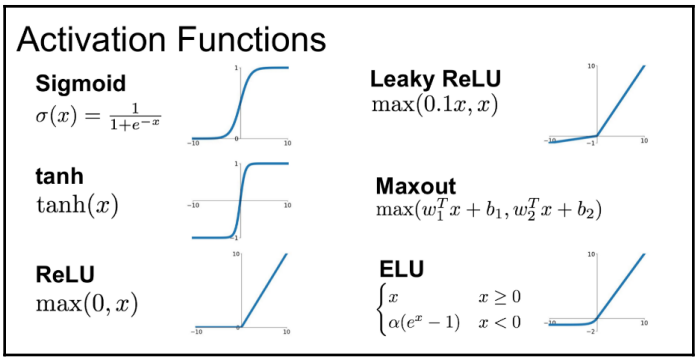


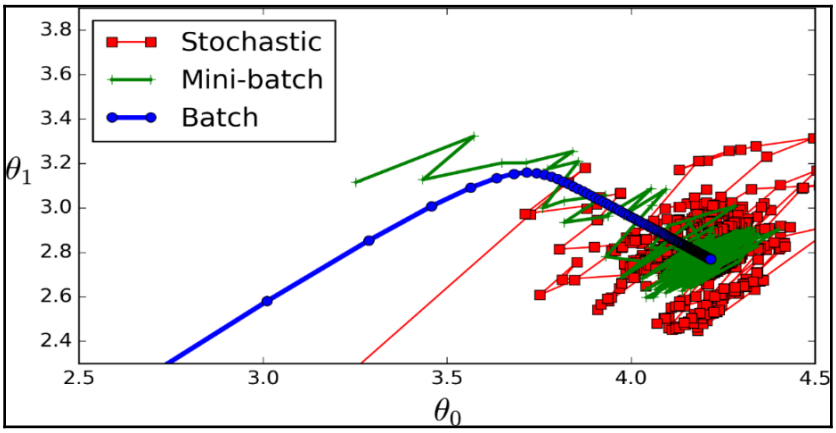


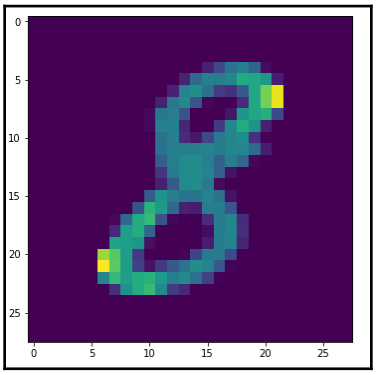

σ_{33}
$\mathbf{T}^{(\mathbf{e}_3)}$
σ_{23}
σ_{13}
σ_{31}
$\mathbf{T}^{(\mathbf{e}_1)}$
σ_{11}
σ_{21}
σ_{32}
$\mathbf{T}^{(\mathbf{e}_2)}$
σ_{12}
σ_{22}

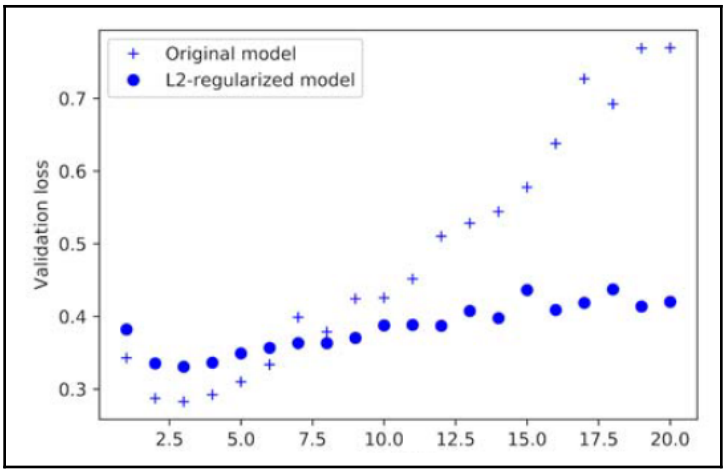

0
5
10
15
20
25
0
5
10
15
20
25

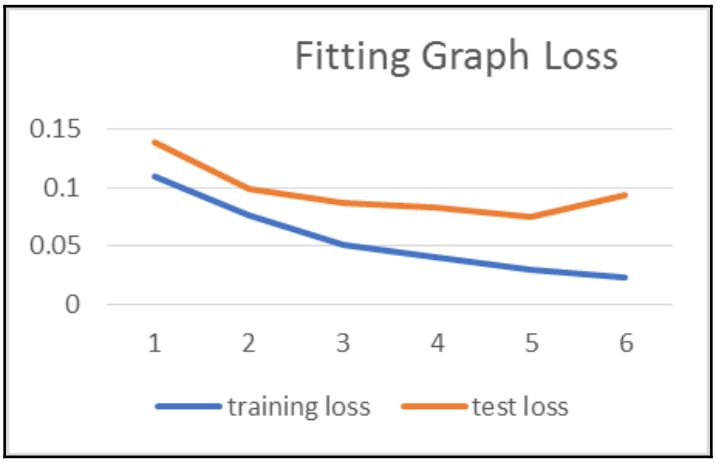

0
5
10
15
20
25
0
5
10
15
20
25

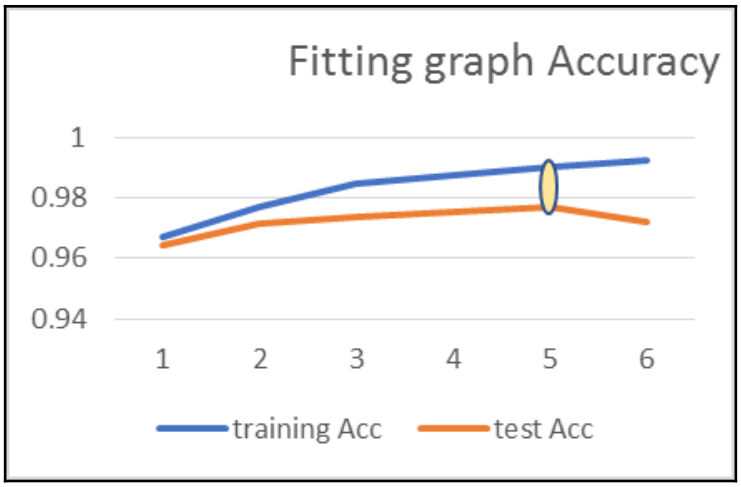

Activation Functions
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$
tanh
$\tanh(x)$
ReLU
$\max(0, x)$
Leaky ReLU
$\max(0.1x, x)$
Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$
ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$

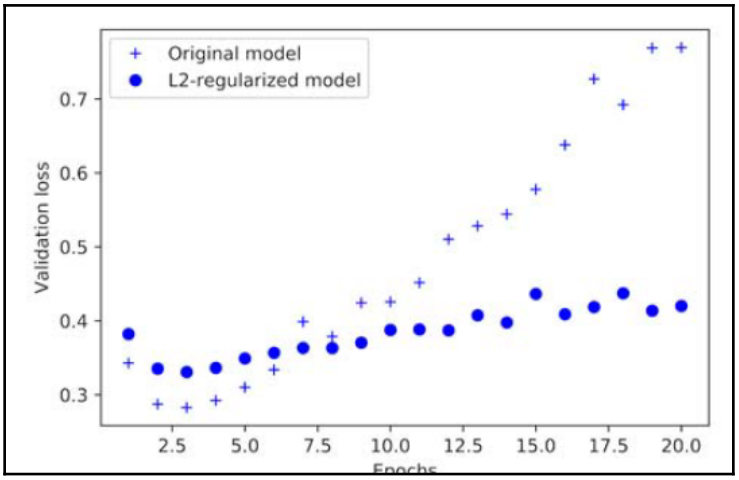

Stochastic
Mini-batch
Batch
θ_1
θ_0
3.8
3.6
3.4
3.2
3.0
2.8
2.6
2.4
2.5
3.0
3.5
4.0
4.5

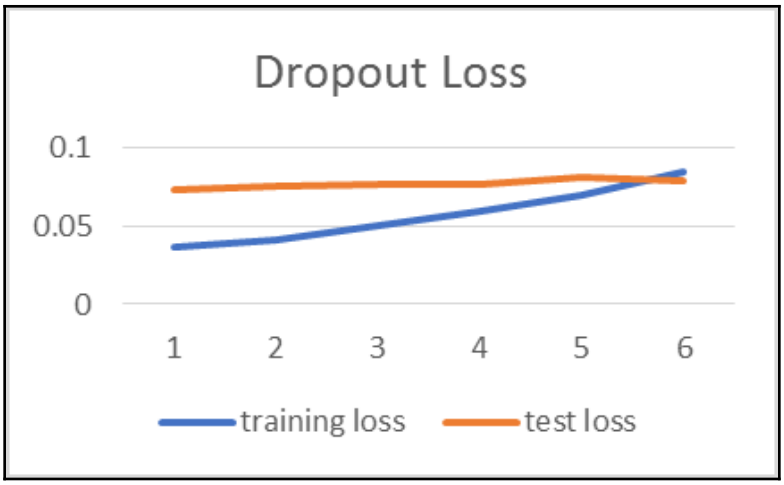

0
5
10
15
20
25

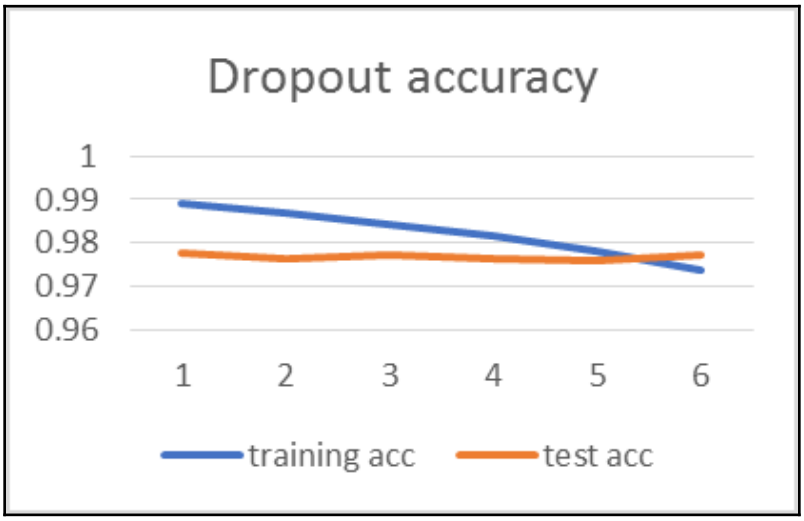

Original model
L2-regularized model
Validation loss
0.7
0.6
0.5
0.4
0.3
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

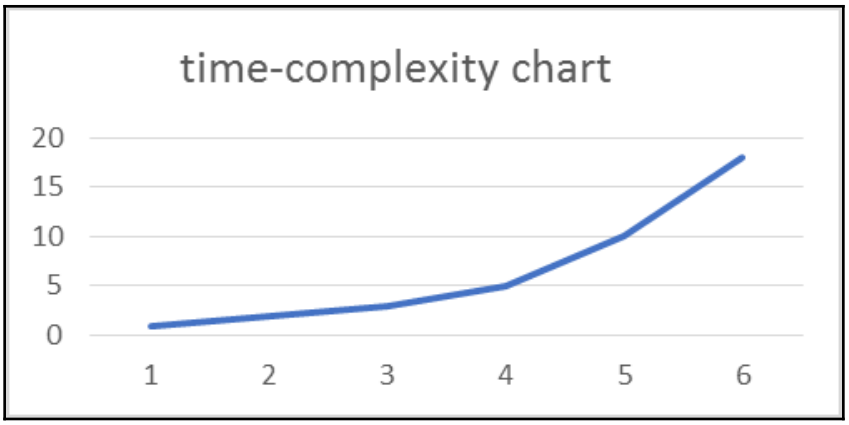

Fitting Graph Loss
0.15
0.1
0.05
0
1
2
3
4
5
6
training loss
test loss

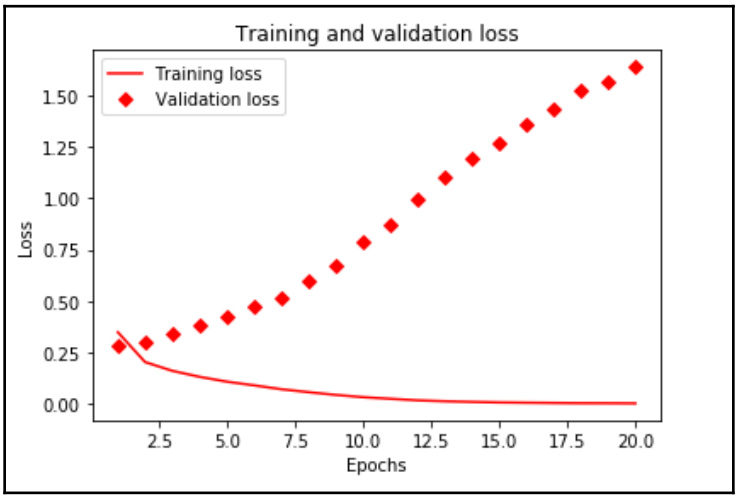

Fitting graph Accuracy
1
0.98
0.96
0.94
1
2
3
4
5
6
training Acc
test Acc

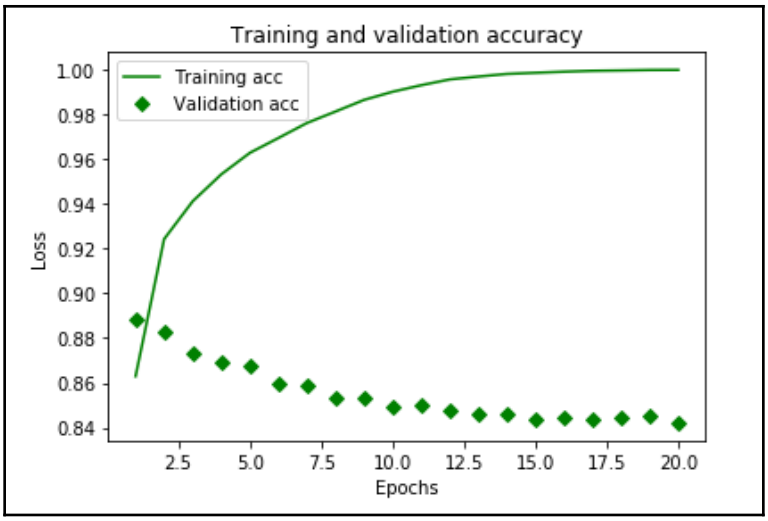

Original model
L2-regularized model
Validation loss
0.7
0.6
0.5
0.4
0.3
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Dropout Loss
0.1
0.05
0
1
2
3
4
5
6
training loss
test loss

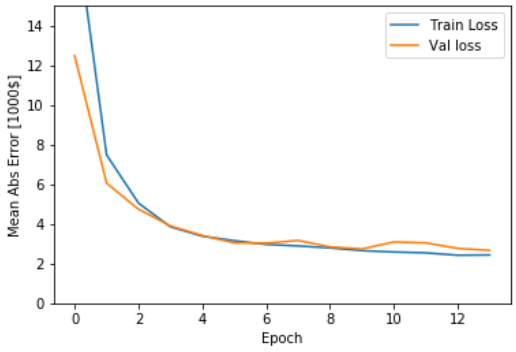

Dropout accuracy
1
0.99
0.98
0.97
0.96
1
2
3
4
5
6
training acc
test acc


time-complexity chart
20
15
10
5
0
1
2
3
4
5
6

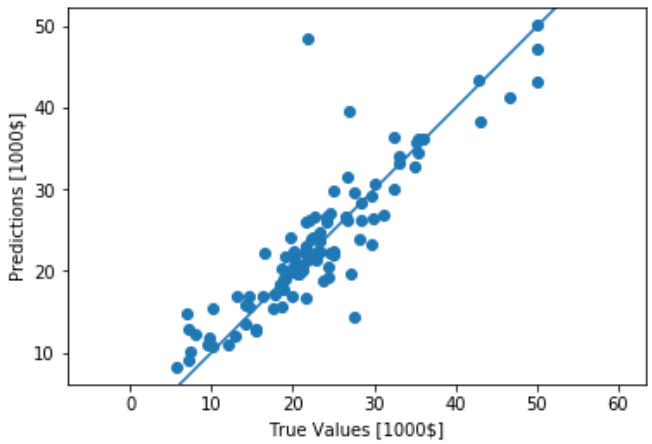

Training and validation loss
Training loss
Validation loss
Loss
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Epochs



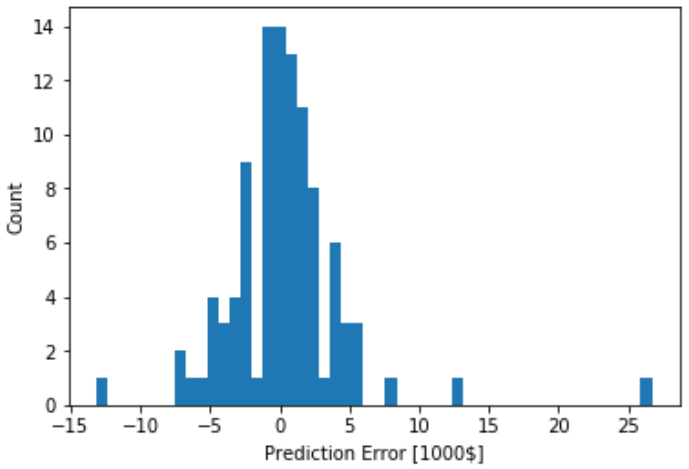

```python
import matplotlib.pyplot as plt

import numpy as np

def plot_history(history):
  plt.figure()
  plt.xlabel('Epoch')
  plt.ylabel('Mean Abs Error [1000$]')
  plt.plot(history.epoch, np.array(history.history['mean_absolute_error']),
           label='Train Loss')
  plt.plot(history.epoch, np.array(history.history['val_mean_absolute_error']),
           label = 'Val loss')
  plt.legend()
  plt.ylim([0, 15])

plot_history(network_metadata)
```

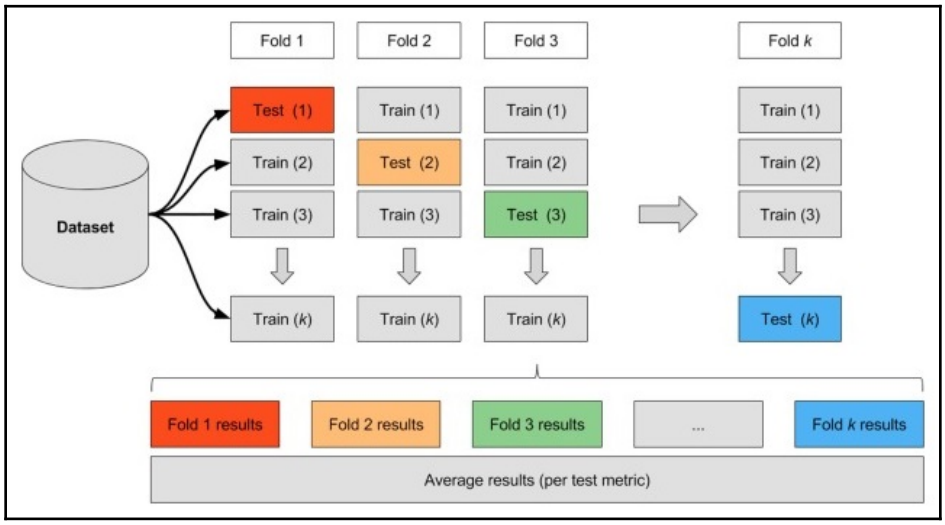


```
test_predictions = model.predict(x_test).flatten()

plt.scatter(y_test, test_predictions)
plt.xlabel('True Values [1000$]')
plt.ylabel('Predictions [1000$]')
plt.axis('equal')
plt.xlim(plt.xlim())
plt.ylim(plt.ylim())
_ = plt.plot([-100, 100], [-100, 100])
```

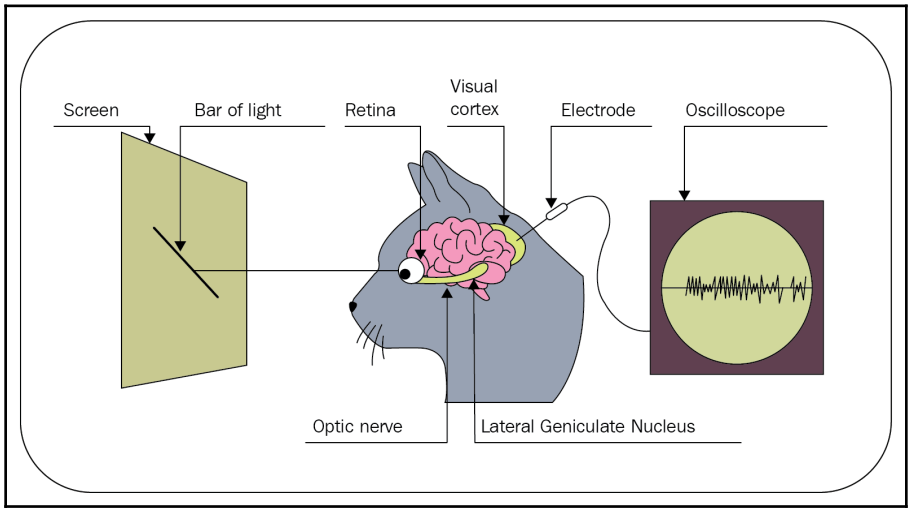


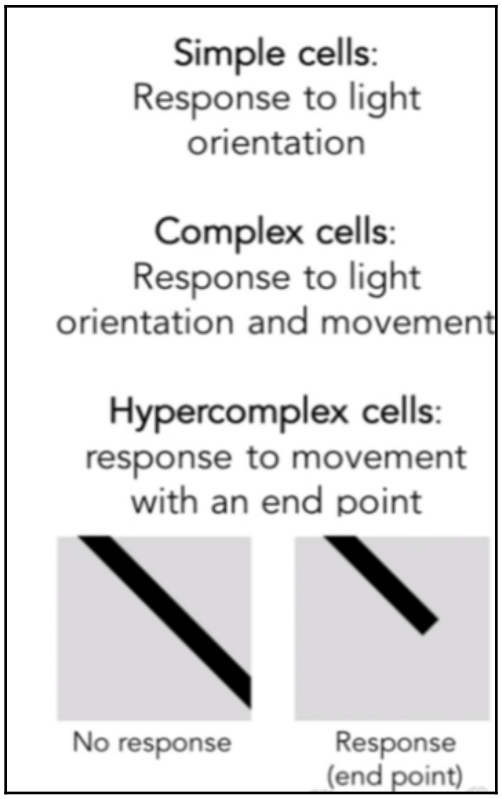
```
error = test_predictions - y_test
plt.hist(error, bins = 50)
plt.xlabel("Prediction Error [1000$]")
_ = plt.ylabel("Count")
```

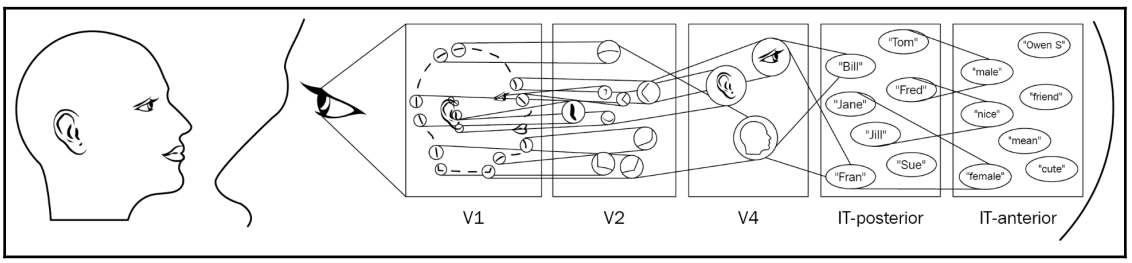


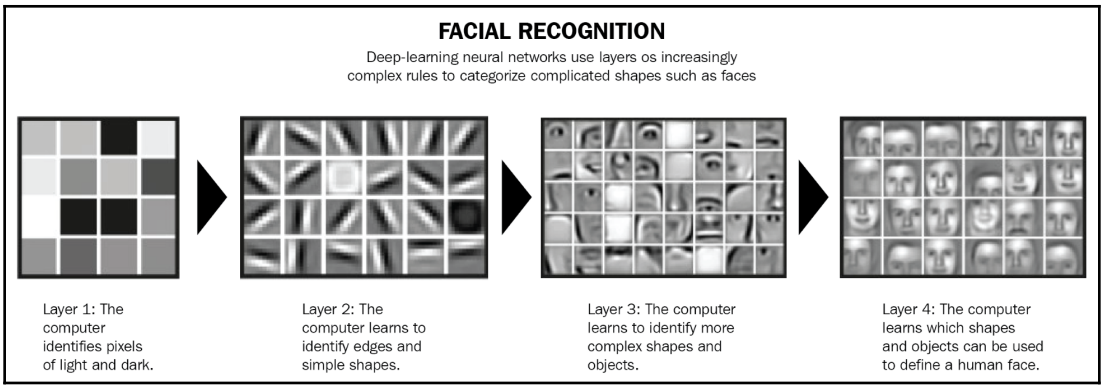


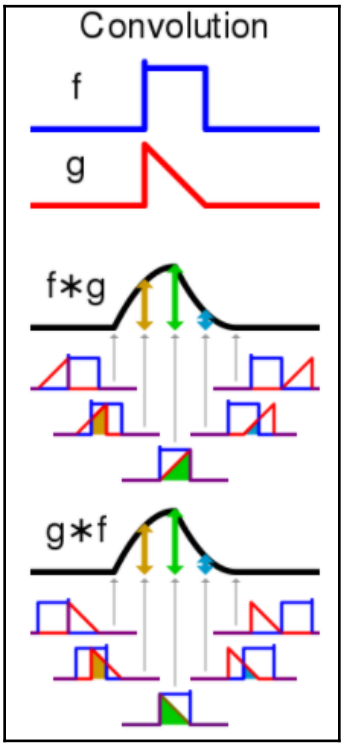
Chapter 04: Convolutional Neural Networks

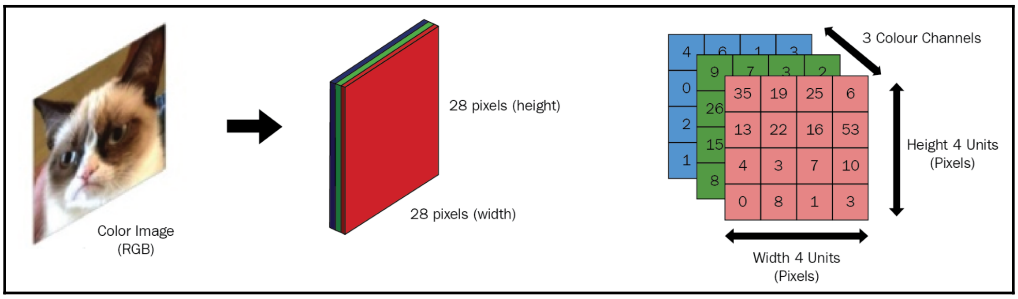


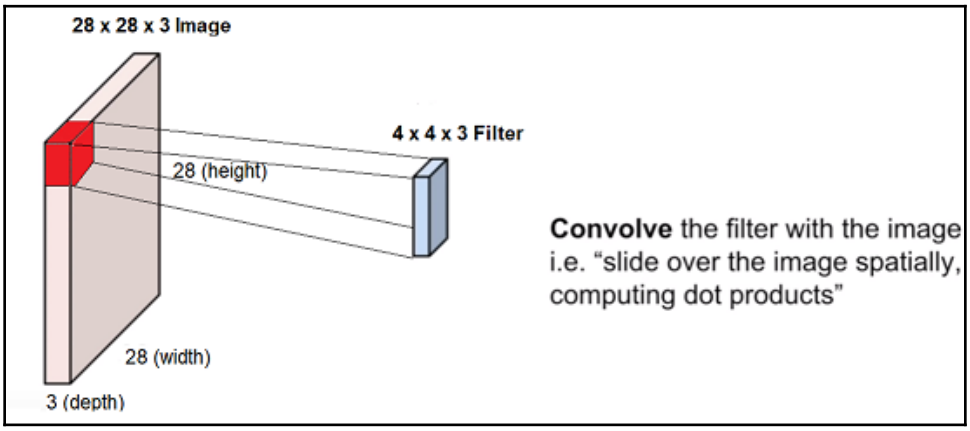

Simple cells:
Response to light orientation
Complex cells:
Response to light orientation and movement
Hypercomplex cells:
response to movement with an end point
No response
Response (end point)

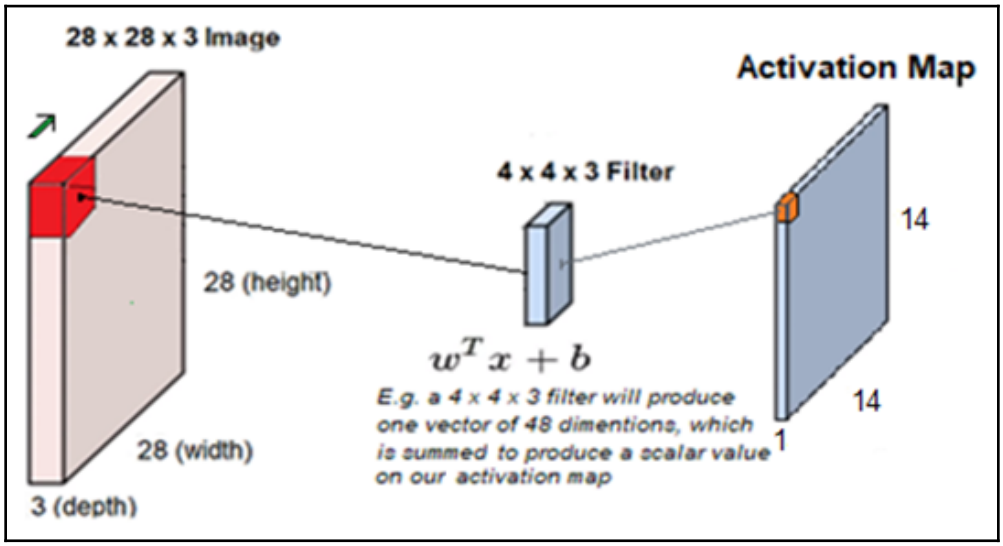

"Tom"
"Bill"
"Fred"
"Jane"
"Jill"
"Sue"
"Fran"
"Owen S"
"male"
"friend"
"nice"
"mean"
"cute"
"female"
V1
V2
V4
IT-posterior
IT-anterior

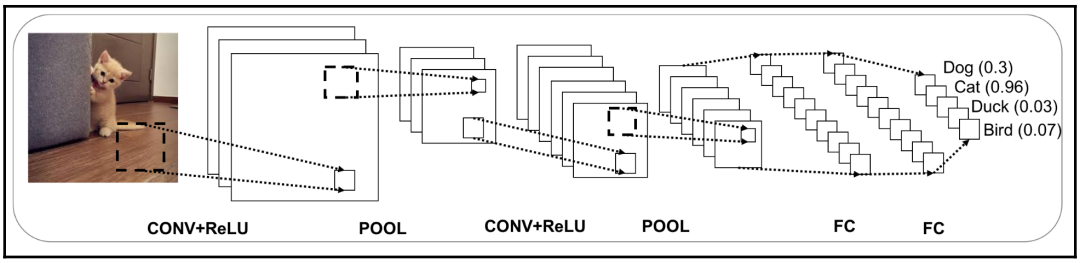

FACIAL RECOGNITION
Deep-learning neural networks use layers os increasingly complex rules to categorize complicated shapes such as faces
Layer 1: The computer identifies pixels of light and dark.
Layer 2: The computer learns to identify edges and simple shapes.
Layer 3: The computer learns to identify more complex shapes and objects.
Layer 4: The computer learns which shapes and objects can be used to define a human face.

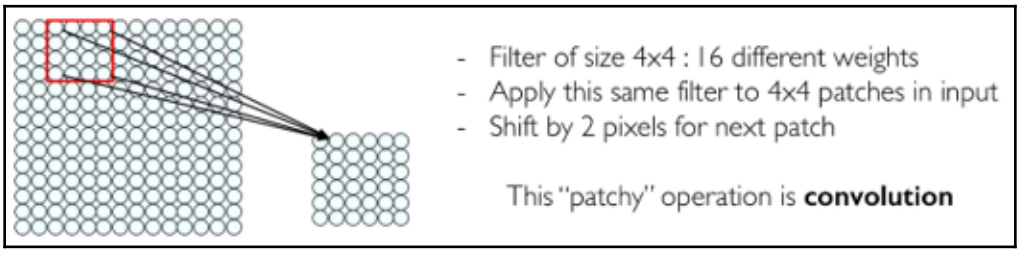

Convolution
f
g
f∗g
g∗f

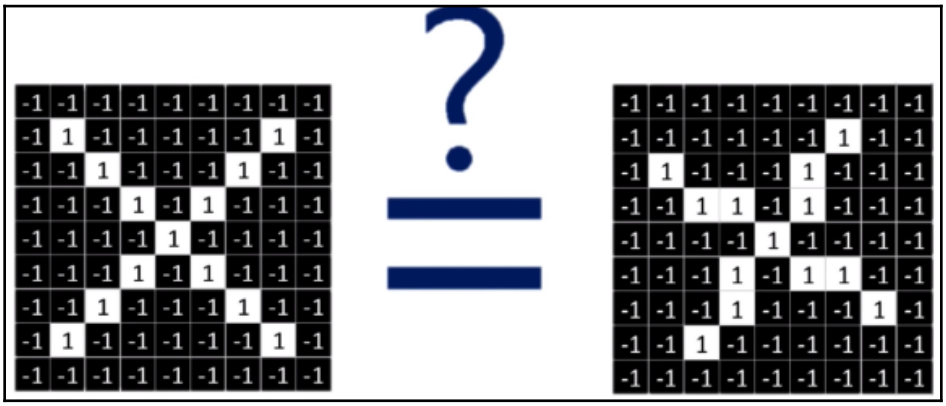

Color Image
(RGB)
28 pixels (height)
28 pixels (width)
3 Colour Channels
Height 4 Units
(Pixels)
Width 4 Units
(Pixels)

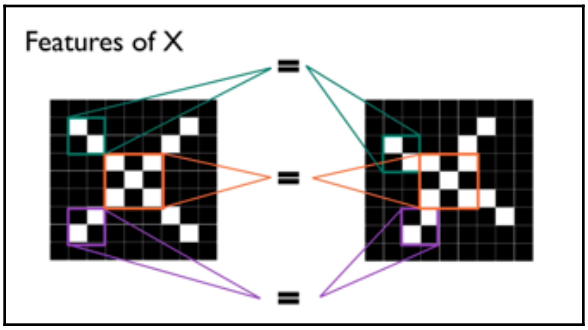

28 x 28 x 3 Image
4 x 4 x 3 Filter
28 (height)
28 (width)
3 (depth)
Convolve the filter with the image
i.e. "slide over the image spatially,
computing dot products"


28 x 28 x 3 Image
Activation Map
4 x 4 x 3 Filter
28 (height)
28 (width)
3 (depth)
$w^T x + b$
E.g. a 4 x 4 x 3 filter will produce one vector of 48 dimentions, which is summed to produce a scalar value on our activation map
14
14
1

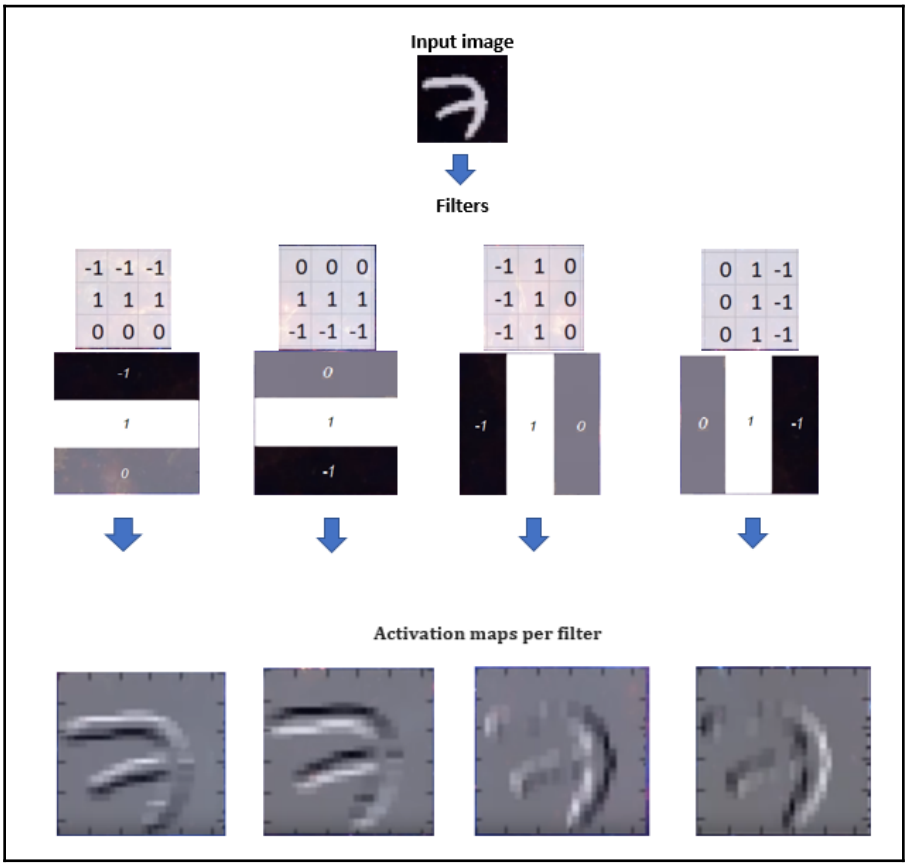

Dog (0.3)
Cat (0.96)
Duck (0.03)
Bird (0.07)
CONV+ReLU
POOL
CONV+ReLU
POOL
FC
FC

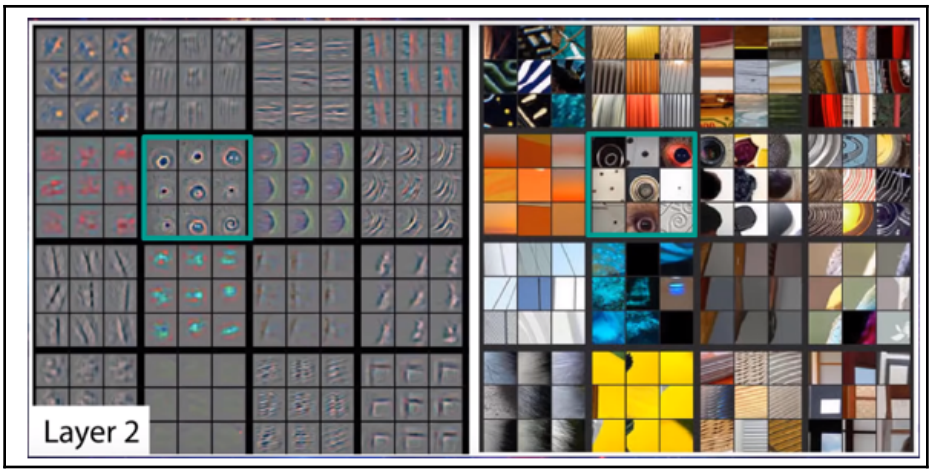

- Filter of size 4x4 : 16 different weights
- Apply this same filter to 4x4 patches in input
- Shift by 2 pixels for next patch
This "patchy" operation is **convolution**

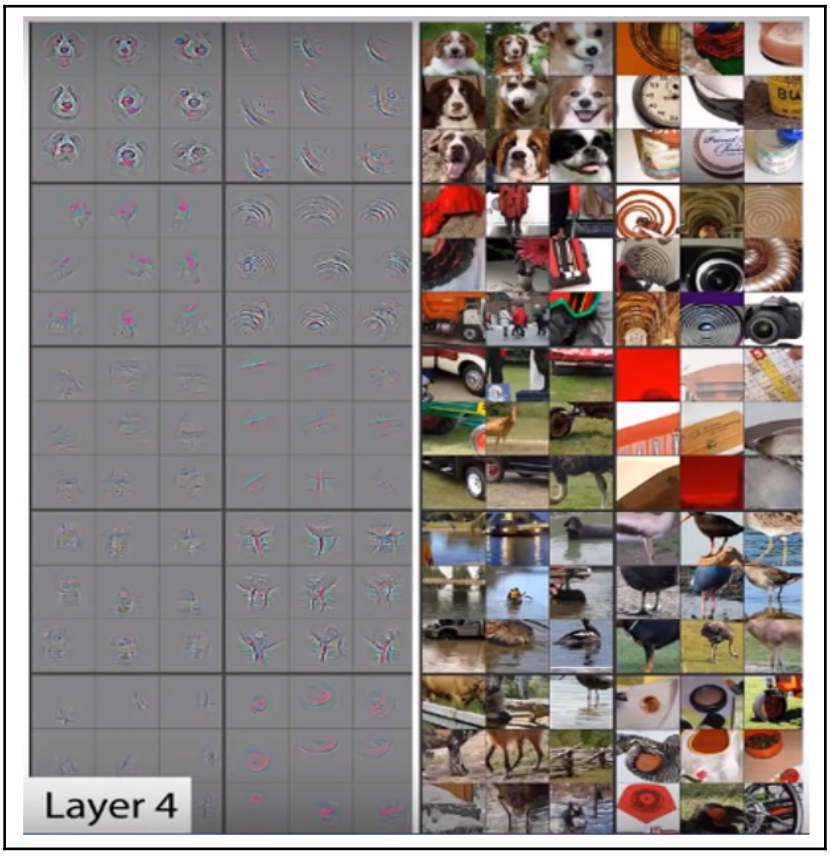

?

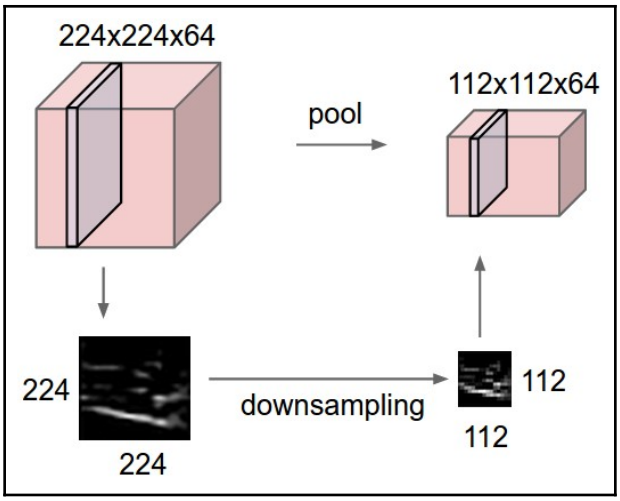

Features of X

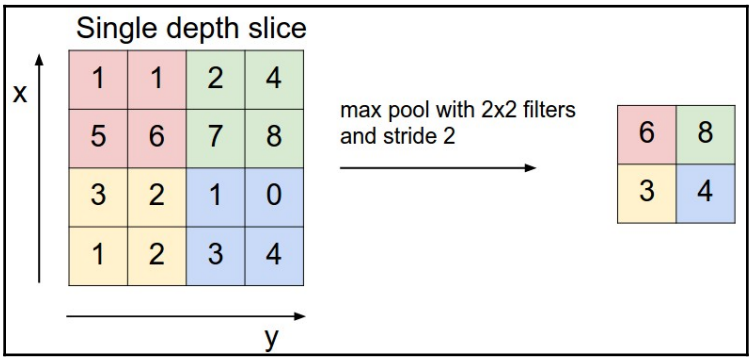
-1 -1 -1	0 0 0	-1 1 0	0 1 -1
1 1 1	1 1 1	-1 1 0	0 1 -1
0 0 0	-1 -1 -1	-1 1 0	0 1 -1

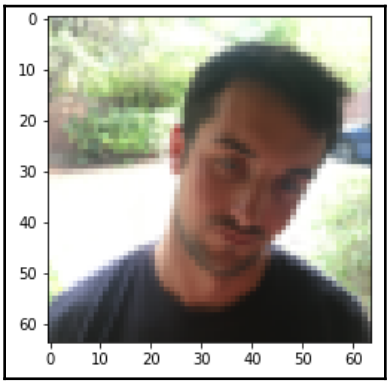


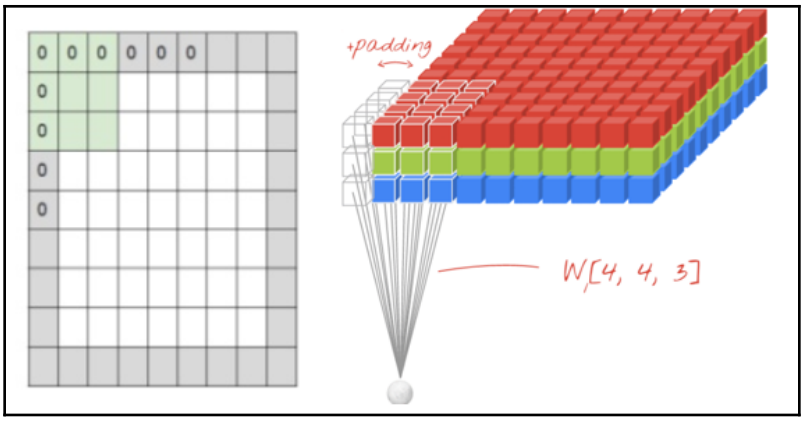

Layer 2


Layer 4


224x224x64
pool
112x112x64
224
224
downsampling
112
112


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4


0
10
20
30
40
50
60
0 10 20 30 40 50 60




```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_29 (Conv2D)           (None, 64, 64, 16)        1216
_________________________________________________________________
batch_normalization_29 (Batc (None, 64, 64, 16)        64
_________________________________________________________________
max_pooling2d_29 (MaxPooling (None, 32, 32, 16)        0
_________________________________________________________________
dropout_29 (Dropout)         (None, 32, 32, 16)        0
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 32, 32, 32)        12832
_________________________________________________________________
batch_normalization_30 (Batc (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d_30 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
dropout_30 (Dropout)         (None, 16, 16, 32)        0
_________________________________________________________________
flatten_15 (Flatten)         (None, 8192)              0
_________________________________________________________________
dense_29 (Dense)             (None, 128)               1048704
_________________________________________________________________
dense_30 (Dense)             (None, 1)                 129
=================================================================
Total params: 1,063,073
Trainable params: 1,062,977
Non-trainable params: 96
_________________________________________________________________
```

```
model.compile(optimizer = 'adam',
              loss = 'binary_crossentropy',
              metrics = ['accuracy'])
```

```
from keras.callbacks import EarlyStopping

early_stopping = keras.callbacks.EarlyStopping(monitor='val_loss')
```

```
model.fit(X_train, Y_train, validation_data=(X_test, Y_test),
                            epochs=20,
                            batch_size=50,
                            callbacks=[early_stopping])
```

```
Train on 600 samples, validate on 150 samples
Epoch 1/20
600/600 [==============================] - 24s 40ms/step - loss: 1.1793 - acc: 0.5583 - val_loss: 0.5761 - val_acc: 0.6267
Epoch 2/20
600/600 [==============================] - 21s 35ms/step - loss: 0.5251 - acc: 0.7850 - val_loss: 0.4851 - val_acc: 0.7600
Epoch 3/20
600/600 [==============================] - 20s 34ms/step - loss: 0.3586 - acc: 0.8400 - val_loss: 0.3339 - val_acc: 0.8667
Epoch 4/20
600/600 [==============================] - 20s 34ms/step - loss: 0.2379 - acc: 0.9117 - val_loss: 0.2766 - val_acc: 0.8800
Epoch 5/20
600/600 [==============================] - 20s 34ms/step - loss: 0.1905 - acc: 0.9267 - val_loss: 0.1928 - val_acc: 0.9400
Epoch 6/20
600/600 [==============================] - 20s 34ms/step - loss: 0.1688 - acc: 0.9250 - val_loss: 0.1650 - val_acc: 0.9600
Epoch 7/20
600/600 [==============================] - 20s 34ms/step - loss: 0.1287 - acc: 0.9533 - val_loss: 0.1554 - val_acc: 0.9400
Epoch 8/20
600/600 [==============================] - 21s 35ms/step - loss: 0.0968 - acc: 0.9600 - val_loss: 0.2802 - val_acc: 0.8800

<keras.callbacks.History at 0x2baa51ba7b8>
```

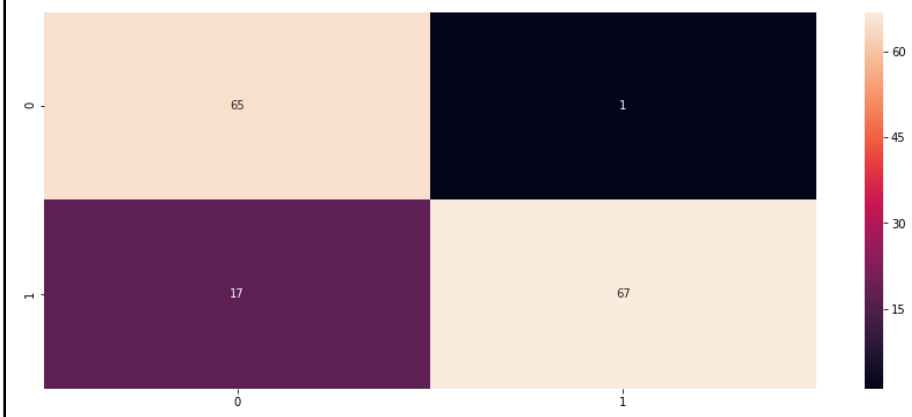
```
# Predict the test set results
Y_pred = model.predict_classes(X_test)
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, recall_score, precision_score, f1_score


print ("test accuracy: %s" %accuracy_score(Y_test, Y_pred))
print ("precision: %s"  %precision_score(Y_test, Y_pred))
print ("recall: %s" %recall_score(Y_test, Y_pred))
print ("f1 score: %s"  %f1_score(Y_test, Y_pred))


```

```
test accuracy: 0.88
precision: 0.9852941176470589
recall: 0.7976190476190477
f1 score: 0.8815789473684211
```

```
import seaborn as sns

cm = confusion_matrix(Y_test,Y_pred)
sns.heatmap(cm,annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2bb0ed3a1d0>
```




```
model.save('C:/Users/npurk/Desktop/Chapter_3_CNN/smile_detector.h5py')
model = keras.models.load_model('C:/Users/npurk/Desktop/Chapter_3_CNN/smile_detector.h5py')
```

```
input_image = X_test[8]
input_image = np.expand_dims(input_image, axis=0)
print(input_image.shape)
```

```
(1, 64, 64, 3)
```

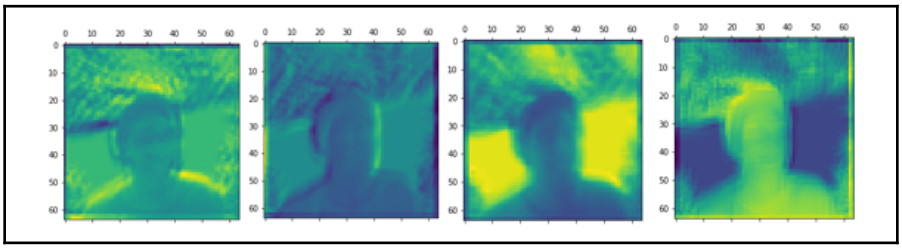
```
#retrive a blank multi-output model from the functional API
from keras.models import Model

#retrieve layer outputs for layers in previously trained sequential model
layer_outputs_smile_detector = [layer.output for layer in model.layers[:]]

#define multi-output model that takes input image tensors and outputs intermediate layer activations
multioutput_model = Model(inputs=model.input, outputs=layer_outputs)

#Generate activation tensors of intermediate layers
activations = multioutput_model.predict(input_image)
```

```
print('number of layers:', len(activations))
print('Datatype:', type(activations[0]))
print('1st layer output shape:', activations[0].shape)
print('4th layer output shape:', activations[3].shape)
print('8th layer output shape:', activations[7].shape)
```

```
number of layers: 8
Datatype: <class 'numpy.ndarray'>
1st layer output shape: (1, 64, 64, 16)
4th layer output shape: (1, 32, 32, 16)
8th layer output shape: (1, 16, 16, 32)
```

```
1 model.summary()
```

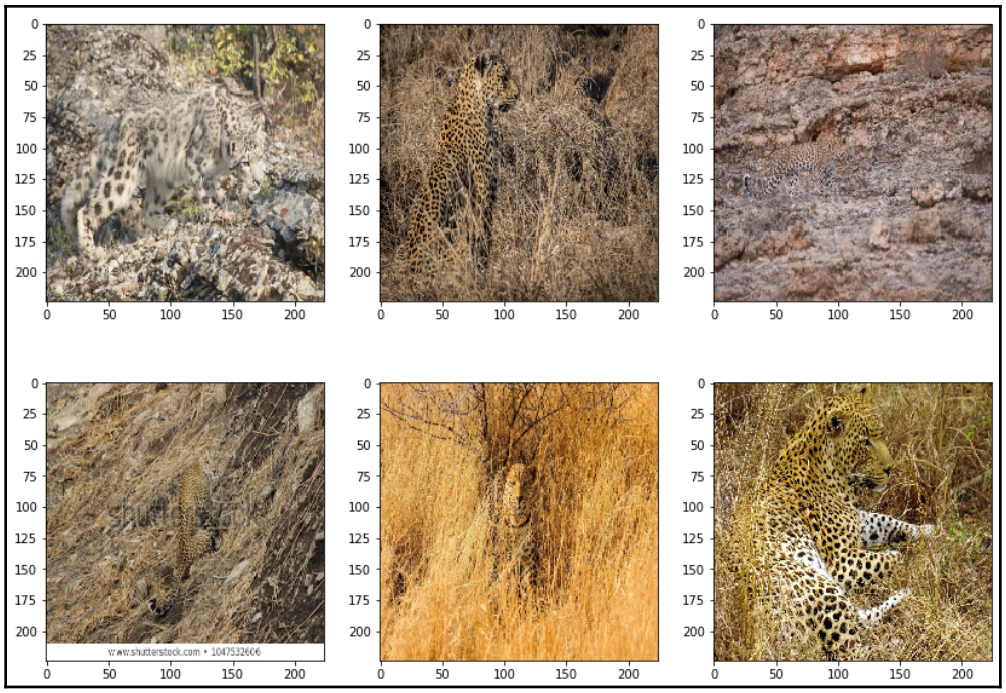
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_29 (Conv2D)           (None, 64, 64, 16)        1216
_________________________________________________________________
batch_normalization_29 (Batc (None, 64, 64, 16)        64
_________________________________________________________________
max_pooling2d_29 (MaxPooling (None, 32, 32, 16)        0
_________________________________________________________________
dropout_29 (Dropout)         (None, 32, 32, 16)        0
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 32, 32, 32)        12832
_________________________________________________________________
batch_normalization_30 (Batc (None, 32, 32, 32)        128
_________________________________________________________________
max_pooling2d_30 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
dropout_30 (Dropout)         (None, 16, 16, 32)        0
_________________________________________________________________
flatten_15 (Flatten)         (None, 8192)              0
_________________________________________________________________
dense_29 (Dense)             (None, 128)               1048704
_________________________________________________________________
dense_30 (Dense)             (None, 1)                 129
=================================================================
Total params: 1,063,073
Trainable params: 1,062,977
Non-trainable params: 96
_________________________________________________________________
```



$$\frac{\partial output}{\partial input}$$

```python
from keras.applications import ResNet50
from vis.utils import utils
from keras import activations

# Hide warnings on Jupyter Notebook
import warnings
warnings.filterwarnings('ignore')

# Build the ResNet50 network with ImageNet weights
model = ResNet50(weights='imagenet', include_top=True)

# Utility to search for layer index by name.
# Alternatively we can specify this as -1 since it corresponds to the last layer.
layer_idx = utils.find_layer_idx(model, 'fc1000')

# Swap softmax with linear
model.layers[layer_idx].activation = activations.linear
model = utils.apply_modifications(model)
```

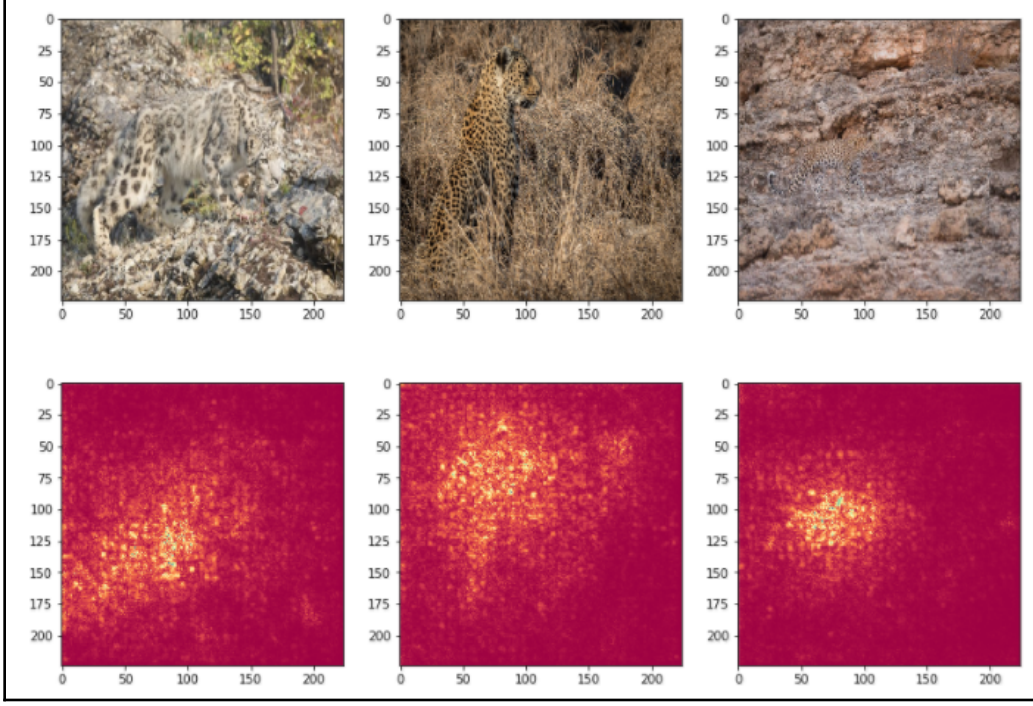
```python
from vis.utils import utils
from matplotlib import pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = (14, 10)

img1 = utils.load_img('C:/Users/npurk/Pictures/Saved Pictures/leo_1.jpg', target_size=(224, 224))
img2 = utils.load_img('C:/Users/npurk/Pictures/Saved Pictures/leo_2.jpg', target_size=(224, 224))
img3 = utils.load_img('C:/Users/npurk/Pictures/Saved Pictures/leo_3.jpg', target_size=(224, 224))
img4 = utils.load_img('C:/Users/npurk/Pictures/Saved Pictures/leo_4.jpg', target_size=(224, 224))
img5 = utils.load_img('C:/Users/npurk/Pictures/Saved Pictures/leo_5.jpg', target_size=(224, 224))
img6 = utils.load_img('C:/Users/npurk/Pictures/Saved Pictures/leo_6.jpg', target_size=(224, 224))

f, ax = plt.subplots(nrows=2, ncols=3)
ax[0, 0].imshow(img1)
ax[0, 1].imshow(img2)
ax[0, 2].imshow(img3)
ax[1, 0].imshow(img4)
ax[1, 1].imshow(img5)
ax[1, 2].imshow(img6)
```




```python
from vis.visualization import visualize_saliency
from vis.utils import utils


leopards = [img1,img2,img3,img4,img5,img6]
# Utility to search for layer index by name.
# Alternatively we can specify this as -1 since it corresponds to the last layer.
layer_idx = utils.find_layer_idx(model, 'fc1000')

gradients=[]
for i, img in enumerate(leopards):
    # 288 is the imagenet index corresponding to `leopard, panthera pardus`
    grads = visualize_saliency(model, layer_idx, filter_indices=288, seed_input=cougars[i])
    gradients.append(grads)
```

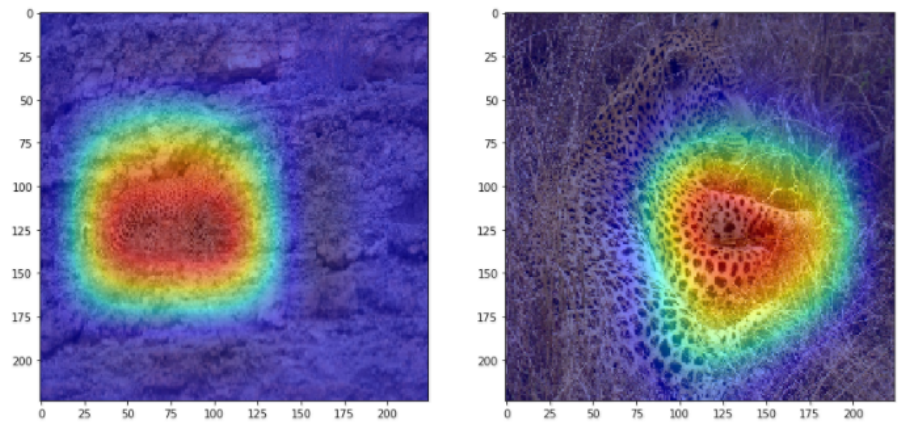
```
f, ax = plt.subplots(2, 3)
ax[0,0].imshow(leopards[0])
ax[0,1].imshow(leopards[1])
ax[0,2].imshow(leopards[2])
ax[1,0].imshow(gradients[0], cmap='Spectral')
ax[1,1].imshow(gradients[1], cmap='Spectral')
ax[1,2].imshow(gradients[2], cmap='Spectral')
```

```
<matplotlib.image.AxesImage at 0x1df4ba81208>
```



```python
import matplotlib.cm as cm
from vis.visualization import visualize_cam, overlay
from keras import activations

# Find the fully connected output layer
layer_idx = utils.find_layer_idx(model, 'fc1000')
#Find the penultimate convolutional layer
final_conv_layer = utils.find_layer_idx(model, 'res5c_branch2c')


plt.figure()
f, ax = plt.subplots(1, 2)

for i, img in enumerate([img3, img6]):

    grads = visualize_cam(model,                                    #ResNet50 model on ImageNet weights
                          seed_input=img,                           #Image with leopard
                          filter_indices=288,                       #Filter Index for leopard in imagenet dataset
                          layer_idx=layer_idx,                      #Last fully connected layer
                          penultimate_layer_idx=final_conv_layer) #Penultimate convolutional layer

    # overlay the heatmap on top of the original image.
    jet_heatmap = np.uint8(cm.jet(grads)[..., :3] * 255)
    ax[i].imshow(overlay(jet_heatmap, img))
```

```
<Figure size 1008x720 with 0 Axes>
```



```
from keras.preprocessing.image import img_to_array
# convert the image pixels to a 4-D numpy array
img_predict = img_to_array(img1)
img_predict = np.expand_dims(img_predict, axis=0)
img_predict.shape
```

```
(1, 224, 224, 3)
```

```
yhat = model.predict(img_predict)

from keras.applications.resnet50 import decode_predictions
# convert the probabilities to class labels
predictions = decode_predictions(yhat, top=1000)
# retrieve the most likely result, e.g. highest probability
label = predictions[0][0]
labels = predictions[0][:]
# print the classification
print('%s (%.2f%%)' % (label[1], label[2]*100))
```

```
snow_leopard (1080.04%)
```

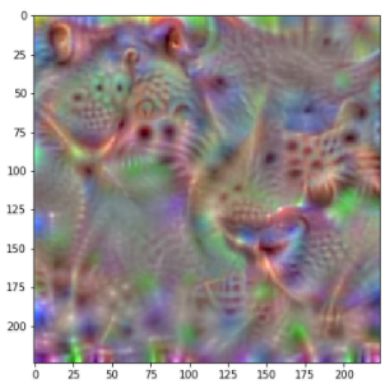
```
labels[:5]
```

```
[('n02128757', 'snow_leopard', 10.800396),
 ('n02130308', 'cheetah', 9.888457),
 ('n02100735', 'English_setter', 8.046139),
 ('n02128385', 'leopard', 7.3484097),
 ('n02110341', 'dalmatian', 6.5762362)]
```

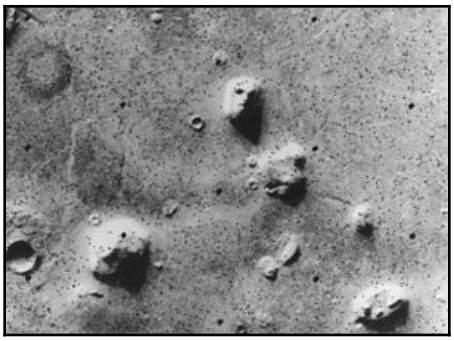
```
from keras.applications import VGG16
from vis.utils import utils
from keras import activations

# Build the VGG16 network with ImageNet weights
model = VGG16(weights='imagenet', include_top=True)

# Utility to search for layer index by name.
# Alternatively we can specify this as -1 since it corresponds to the last layer.
layer_idx = utils.find_layer_idx(model, 'predictions')

# Swap softmax with linear
model.layers[layer_idx].activation = activations.linear
model = utils.apply_modifications(model)
```

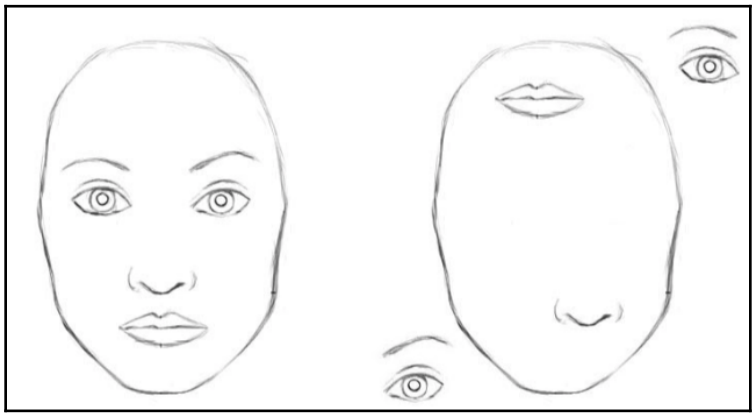
```
from vis.visualization import visualize_activation

from matplotlib import pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = (18, 6)

# 20 is the imagenet category for 'ouzel'
img = visualize_activation(model, layer_idx, filter_indices=[288])
plt.imshow(img)
```

```
<matplotlib.image.AxesImage at 0x1c381974f60>
```




```
img = visualize_activation(model, layer_idx, filter_indices=288, max_iter=500, verbose=True)
```





Chapter 05: Recurrent Neural Networks

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram

$$\begin{array}{ccc} \mathcal{S} & \longrightarrow & \\ \downarrow & & \\ \xi & \longrightarrow & \mathcal{O}_{X'} \\ & & \uparrow \\ \text{gor}_s & & \end{array} \qquad \begin{array}{ccc} = \alpha' & \longrightarrow & \\ \updownarrow & & \\ = \alpha' & \longrightarrow & \alpha \end{array} \qquad \begin{array}{c} X \\ \downarrow \\ \text{d}(\mathcal{O}_{\mathcal{X}_{X/k}}, \mathcal{G}) \end{array}$$

$$\text{Spec}(K_\psi) \qquad \text{Mor}_{Sets}$$

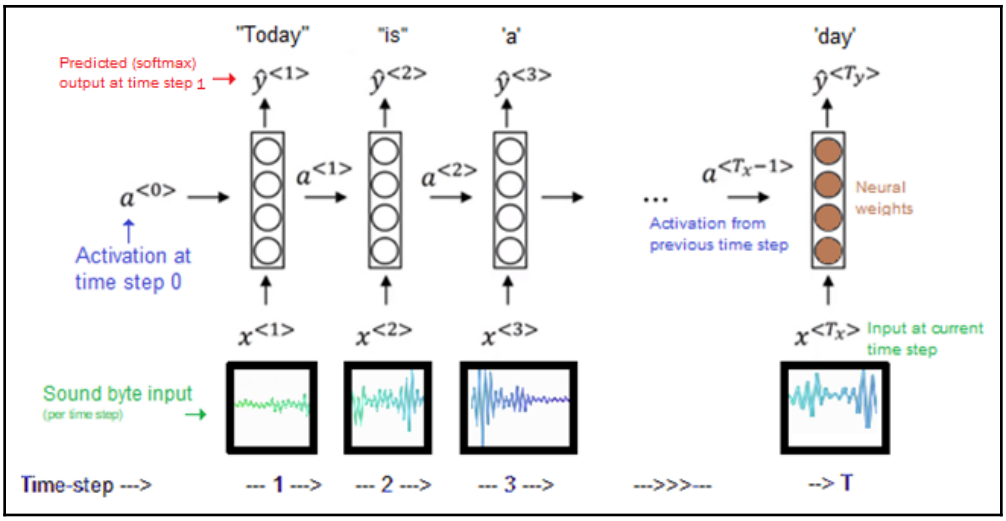
is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type $f_\star$. This is of finite type diagrams, and

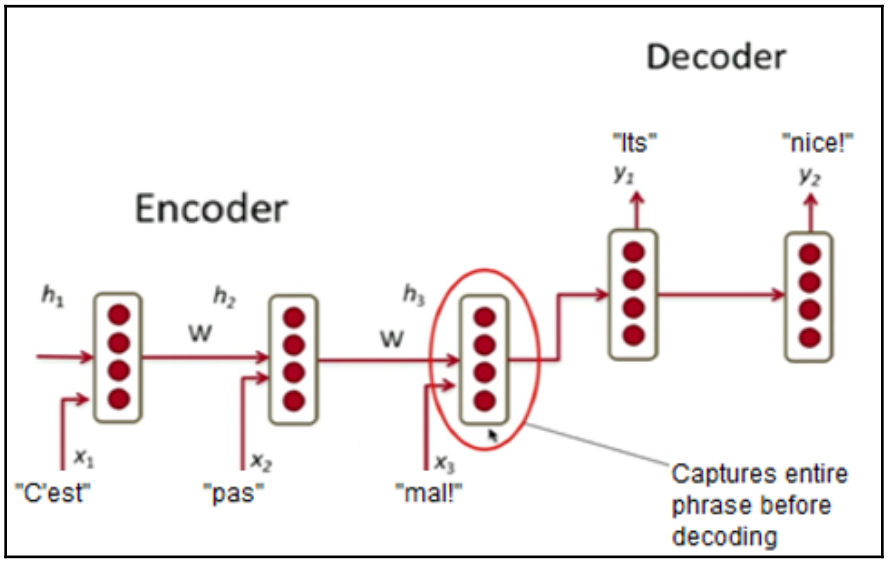
- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

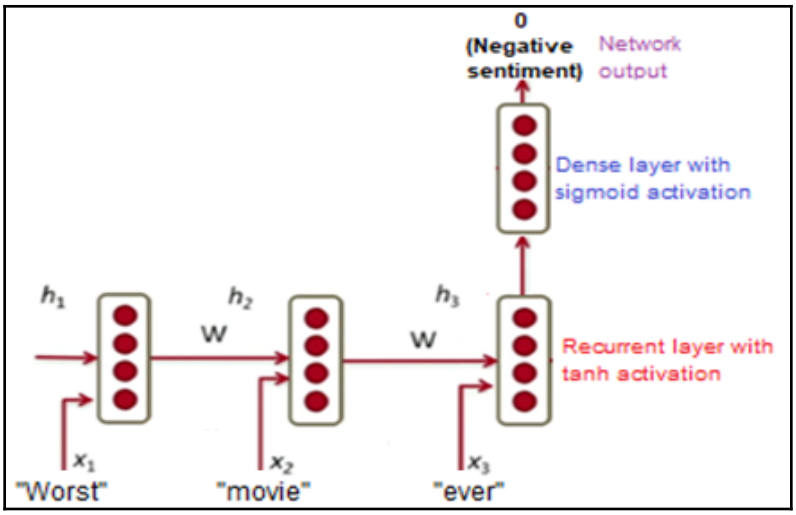
```
1 verify_predictions(22)
```

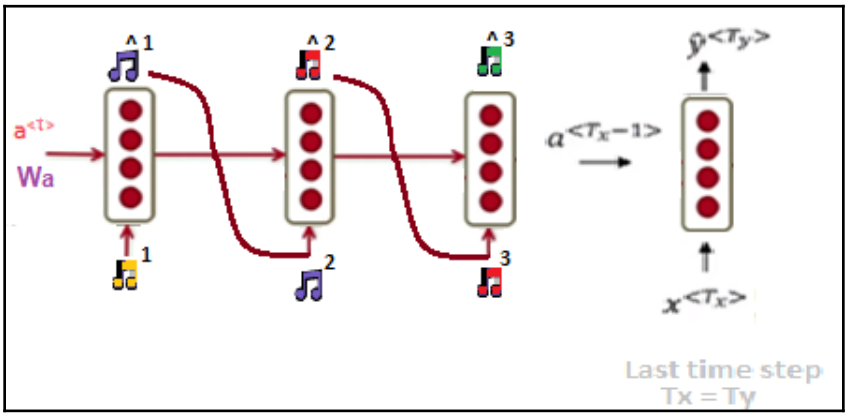
```
Network falsely predicts that review 22 is negative

(None,
 array([0.16930683], dtype=float32),
 "? how managed to avoid attention remains a mystery a potent mix of comedy and crime this one takes chances where tarantino pl
ays it safe with the hollywood formula the risks don't always pay off one character in one sequence comes off ? silly and falls
flat in the lead role thomas jane gives a wonderful and complex performance and two brief appearances by mickey rourke hint at
the high potential of this much under and mis used actor here's a director one should keep one's eye on")
```

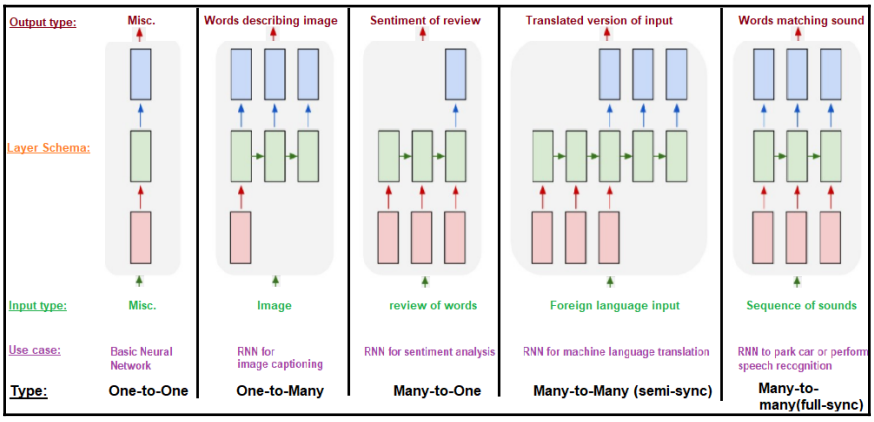
Advantages	Drawbacks
• Possibility of processing input of any length • Model size not increasing with size of input • Computation takes into account historical information • Weights are shared across time	• Computation being slow • Difficulty of accessing information from a long time ago • Cannot consider any future input for the current state

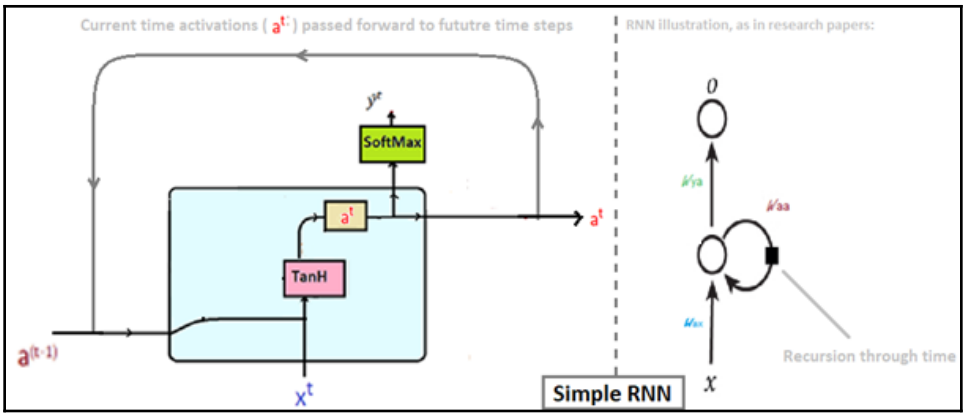

"Today"
"is"
'a'
'day'
Predicted (softmax) output at time step 1
$\hat{y}^{<1>}$
$\hat{y}^{<2>}$
$\hat{y}^{<3>}$
$\hat{y}^{<T_y>}$
$a^{<0>}$
$a^{<1>}$
$a^{<2>}$
$a^{<T_x-1>}$
Neural weights
Activation at time step 0
Activation from previous time step
$x^{<1>}$
$x^{<2>}$
$x^{<3>}$
$x^{<T_x>}$
Input at current time step
Sound byte input
(per time step)
Time-step --->
--- 1 --->
--- 2 --->
--- 3 --->
--->>>---
--> T

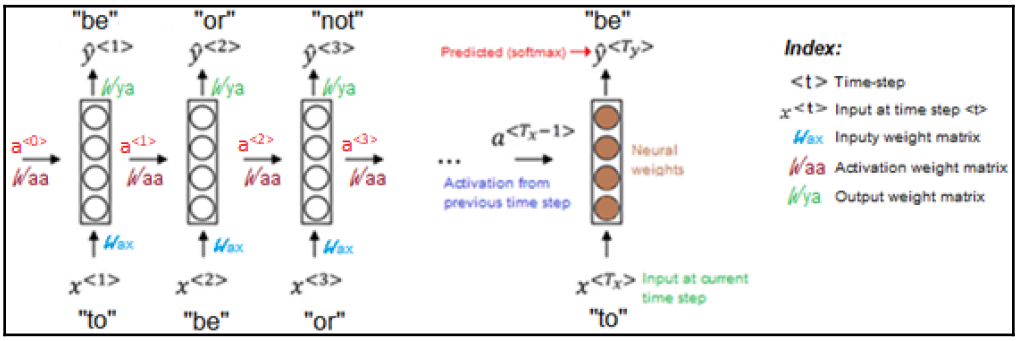

Decoder
Encoder
"Its"
y_1
"nice!"
y_2
h_1
h_2
h_3
W
W
x_1
x_2
x_3
"C'est"
"pas"
"mal!"
Captures entire phrase before decoding

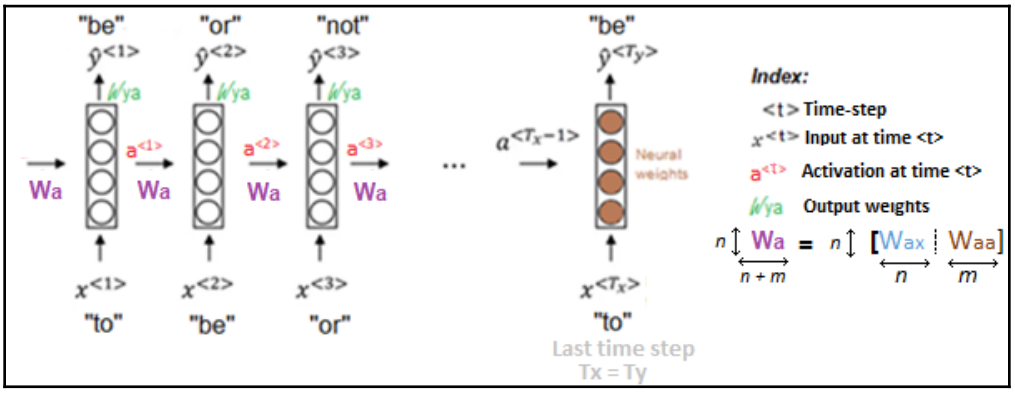

0
(Negative sentiment)
Network output
Dense layer with sigmoid activation
h_1
h_2
h_3
W
W
Recurrent layer with tanh activation
x_1
x_2
x_3
"Worst"
"movie"
"ever"

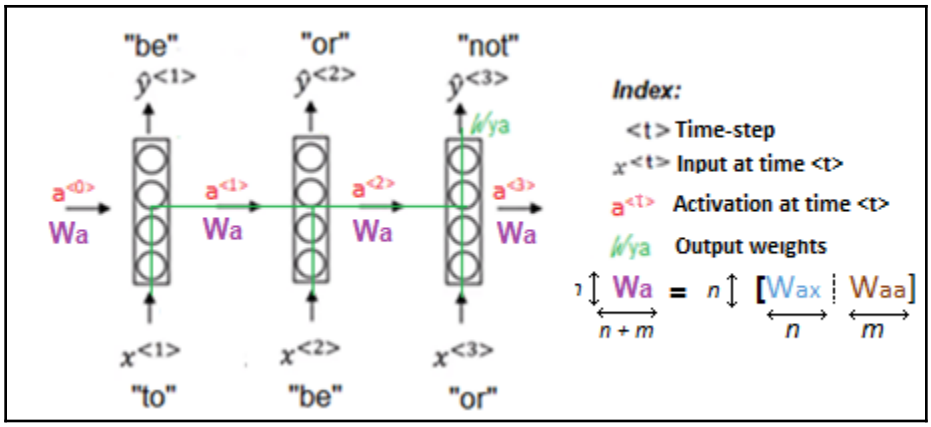

^1
^2
^3
$a^{<1>}$
Wa
1
2
3
$a^{<T_x-1>}$
$\hat{y}^{<T_y>}$
$x^{<T_x>}$
Last time step
Tx = Ty

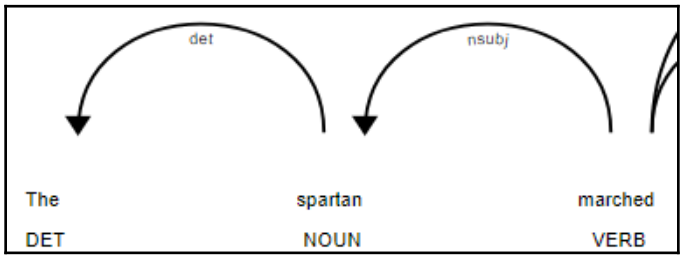

Output type:
Misc.
Words describing image
Sentiment of review
Translated version of input
Words matching sound
Layer Schema:
Input type:
Misc.
Image
review of words
Foreign language input
Sequence of sounds
Use case:
Basic Neural Network
RNN for image captioning
RNN for sentiment analysis
RNN for machine language translation
RNN to park car or perform speech recognition
Type:
One-to-One
One-to-Many
Many-to-One
Many-to-Many (semi-sync)
Many-to-many(full-sync)

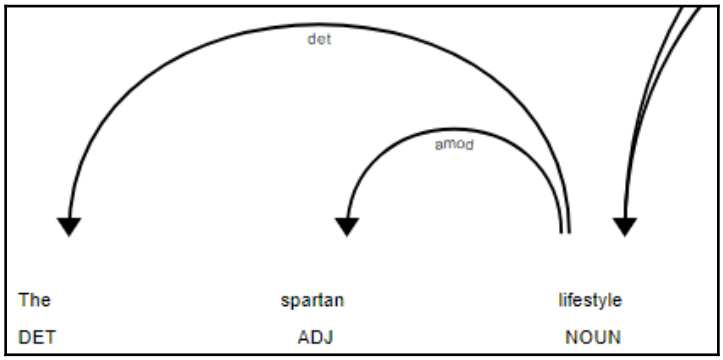

Current time activations (a^t) passed forward to fututre time steps
RNN illustration, as in research papers:
SoftMax
TanH
Simple RNN
Recursion through time

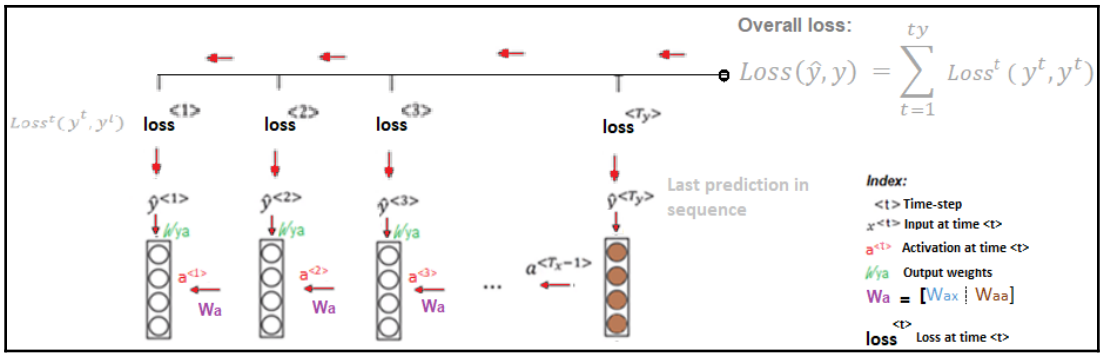

"be"
"or"
"not"
"be"
"to"
"be"
"or"
"to"
Predicted (softmax)
Activation from previous time step
Neural weights
Input at current time step
Index:
<t> Time-step
Input at time step <t>
Inputy weight matrix
Activation weight matrix
Output weight matrix

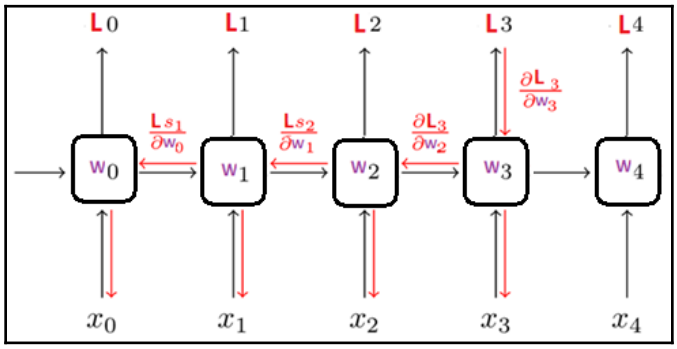

"be"
"or"
"not"
"be"
"to"
"be"
"or"
"to"
Wa
Neural weights
Last time step
Tx = Ty
Index:
<t> Time-step
Input at time <t>
Activation at time <t>
Output weights


"be"
"or"
"not"
"to"
"be"
"or"
Wa
Wya
Index:
<t> Time-step
Input at time <t>
Activation at time <t>
Output weights
Wa = [Wax | Waa]
n + m
n
m


det
nsubj
The
spartan
marched
DET
NOUN
VERB


det
amod
The
spartan
lifestyle
DET
ADJ
NOUN





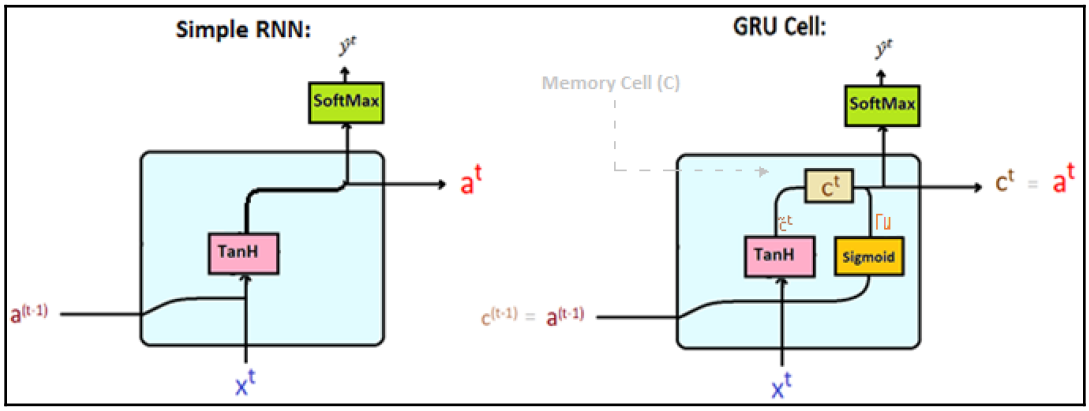
$$\frac{\partial L_3}{\partial W} = \frac{\partial L_3}{\partial y_3} \cdot \frac{\partial y_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial W}$$

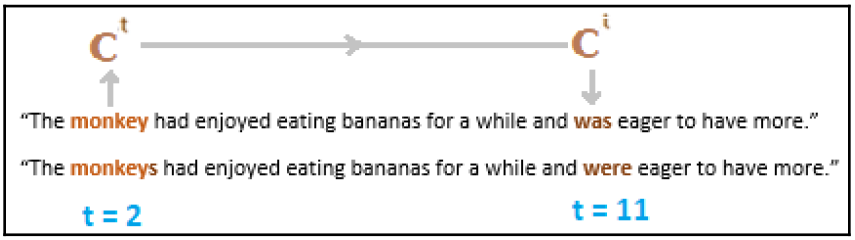
$$= \sum_{t=0}^{2} \frac{\partial L_3}{\partial y_3} \cdot \frac{\partial y_3}{\partial h_2} \cdot \frac{\partial h_2}{\partial h_t} \cdot \frac{\partial h_t}{\partial W}$$

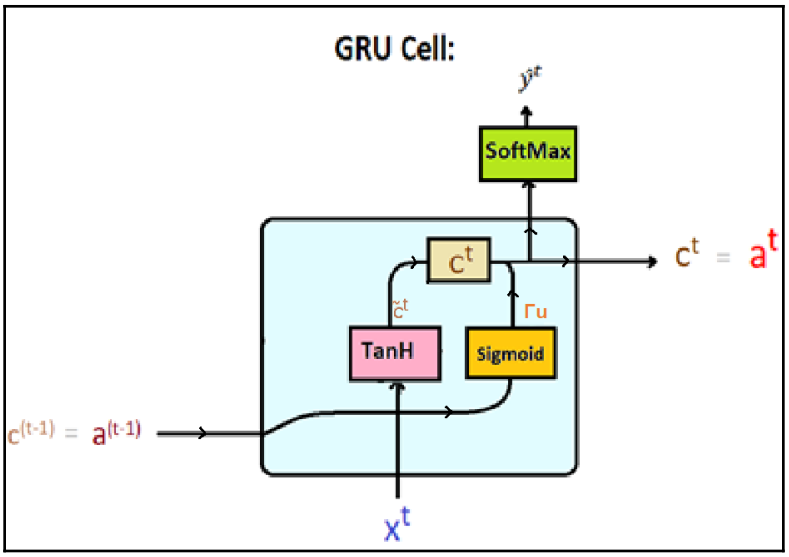
$$= \sum_{t=0}^{2} \frac{\partial L_3}{\partial y_3} \cdot \frac{\partial y_3}{\partial h_2} \cdot \left(\prod_{j=t+1}^{2} \frac{\partial h_j}{\partial h_{j-1}} \right) \cdot \frac{\partial h_t}{\partial W}$$

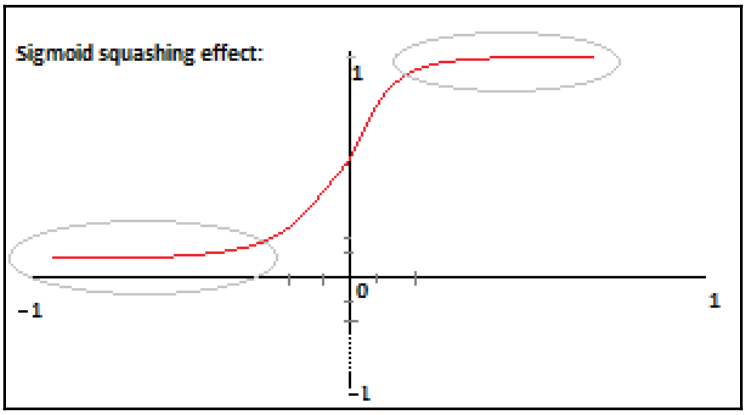
```python
from keras import optimizers

# All parameter gradients will be clipped to
# a maximum norm of 1.
sgd = optimizers.SGD(lr=0.01, clipnorm=1.)
```

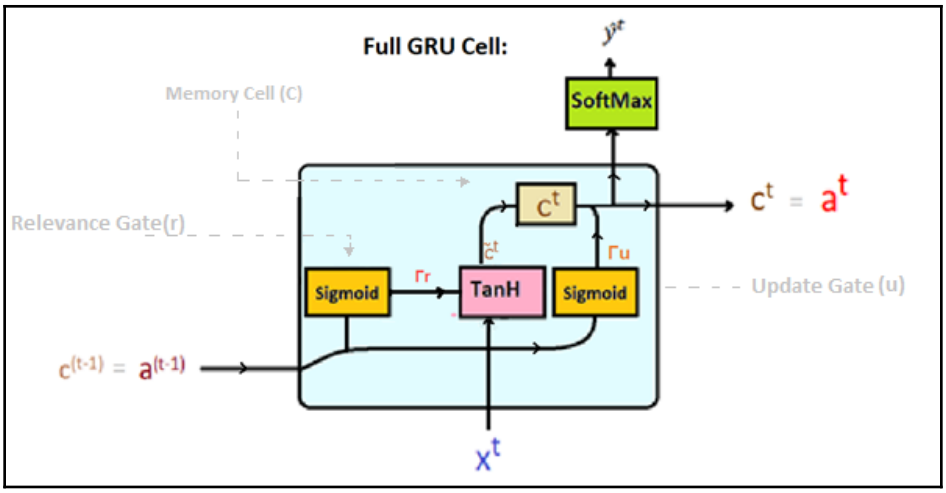
```python
from keras import optimizers

# All parameter gradients will be clipped to
# a maximum value of 0.5 and
# a minimum value of -0.5.
sgd = optimizers.SGD(lr=0.01, clipvalue=0.5)
```

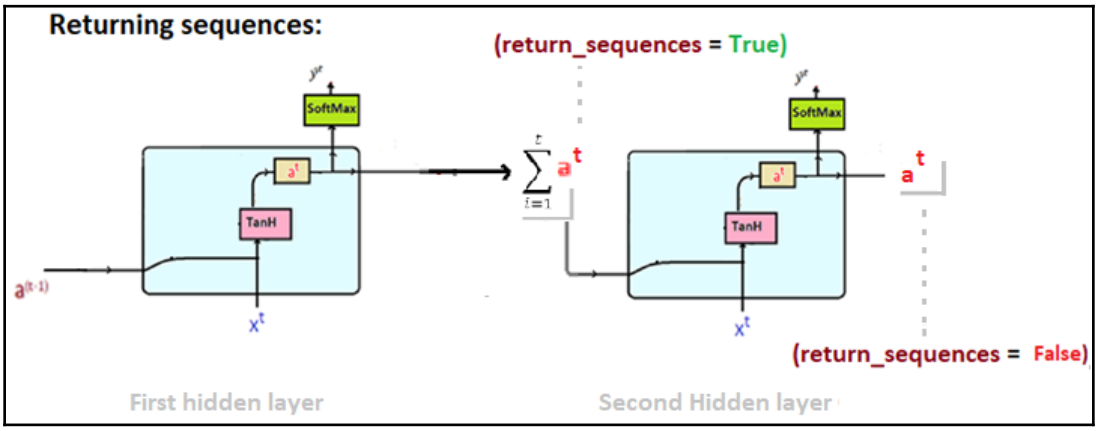


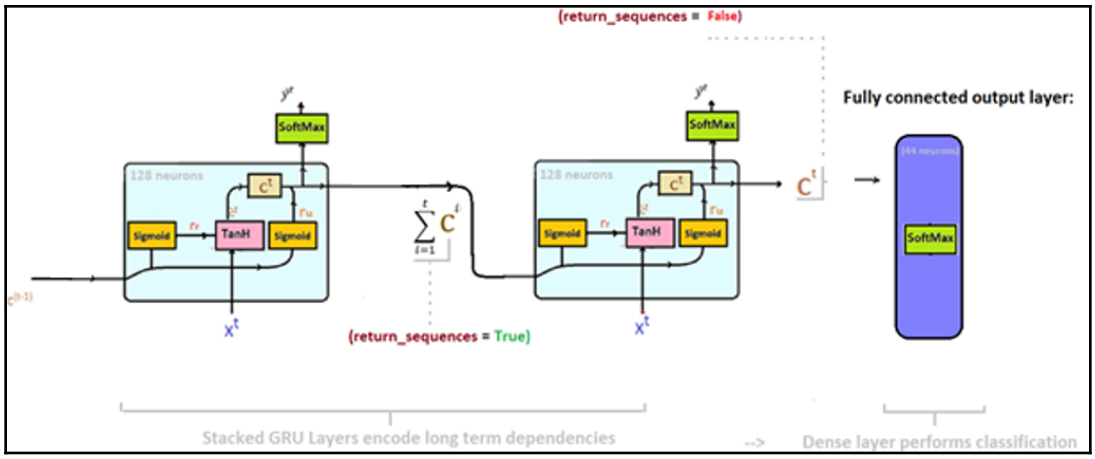



GRU Cell:
$y^{\langle t \rangle}$
SoftMax
c^t
$c^t = a^t$
$\tilde{c}^t$
Γ_u
TanH
Sigmoid
$c^{(t-1)} = a^{(t-1)}$
x^t


Sigmoid squashing effect:
1
0
−1
1
−1


Full GRU Cell:
Memory Cell (C)
SoftMax
Relevance Gate(r)
Sigmoid
TanH
Sigmoid
Update Gate (u)


Returning sequences:
(return_sequences = True)
SoftMax
TanH
SoftMax
TanH
(return_sequences = False)
First hidden layer
Second Hidden layer

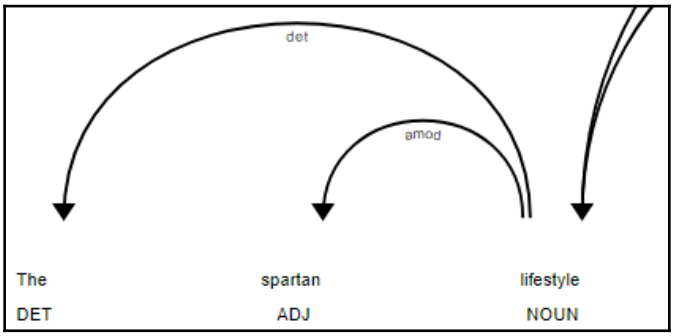


Normal order: →

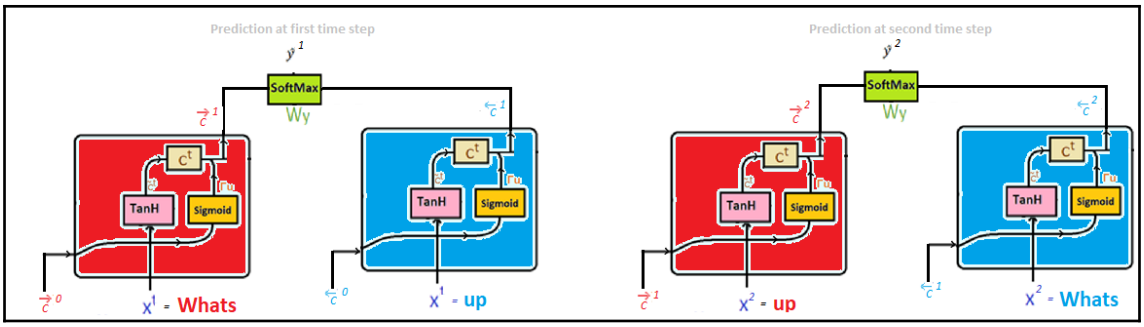
The spartan marched forward despite the obstacles thrown at him

Reverse Order: ←

him at thrown obstacles the despite forward marched spartan The



Time-step->	1	2
Ordered sequence->	Whats	up
Reverse sequence->	up	Whats

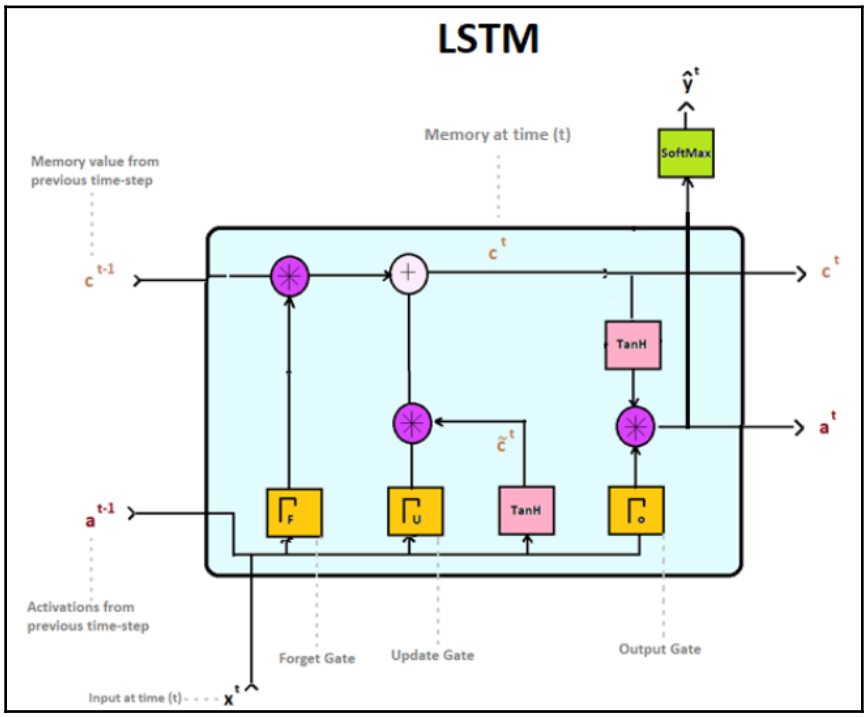

Prediction at first time step
Prediction at second time step
SoftMax
Wy
TanH
Sigmoid
x¹ = Whats
x¹ = up
x² = up
x² = Whats

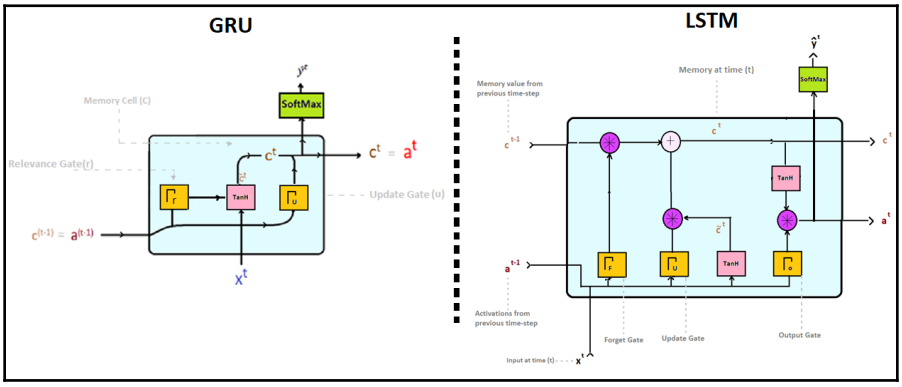
```
1 test_models(all_models, epochs=1)
```

```
Initiating compilation...
Compiled: SimpleRNN_model
Epoch 1/1
166730/166730 [==============================] - 33s 199us/step - loss: 2.2307

----- Generating text after Epoch: 0

----- *Sampling Threshold* : 0.3

----- *Input sequence to generate from* : "nd mine more of the same beauty tha"
nd mine more of the same beauty that in the ther and he wert will , and , th me the me the of in the fere , the port ou hame th
e pance , and he fore if in the wing the late , and whe ham . ham . and with me the heme , and , and lith , and me the wort and
my the for . in the peate , and ntte fore the fere , and list , and lord ham . in the ponge , the my lord , and lore , and with
ma that har . of the fere , and , and and , and will

----- *Sampling Threshold* : 0.5

----- *Input sequence to generate from* : "nd mine more of the same beauty tha"
nd mine more of the same beauty thas and in the himen . wor . in were more , in the lith , gham . not the sering of of her . in
g the galle , and here pall the peaue , ard hat . lowe , and ham . in the senge henat , and hame the harresen . and mente ing t
he wenthin the foll the thend in this ruen the helleere , for . me that ding ous andue the inge in in whe ham . gore the , and
you houd and and , the perte in s and lyour will the o

----- *Sampling Threshold* : 0.7

----- *Input sequence to generate from* : "nd mine more of the same beauty tha"
nd mine more of the same beauty that in the lour inghe ding mist on wire ke by . whe is aun that ine s of hiant hat sheraund qu
einel of s abe shower , thouthes ind . than ' nne so deuenger for mose so fer ancede the mande not mower ham . or saplee tha ki
ng the e tham . io h mest is phis ne came to the becheno , thit with meme hepere foll the forishis nor wing in shin with that .
ineu . not me hor , and , in thrre tone ming , thet i

----- *Sampling Threshold* : 1.0

----- *Input sequence to generate from* : "nd mine more of the same beauty tha"
nd mine more of the same beauty tham and audy myring , fouge ' ses hi kere thit rnse andes muthieers ched erit my yous afco tor
n tis ine myourille , py lowd stbenen apowee prayet , all , tf feryeakestring ot hey for allwingor this at gamse . (et is myof
ltaeda dod . god iphon so h ctay fither ffothis lenre , nsowhetr shafe it ser the weeles dane , orabughis szall , kndsge llot ?
hea , so fother hamll : th then ther phat , and 'it b

----- *Sampling Threshold* : 1.2

----- *Input sequence to generate from* : "nd mine more of the same beauty tha"
nd mine more of the same beauty thangrdi on welt thees cnoune hel m5awit) hom serusl te hent ay h mereriolr thes ghe : eresdoin
kfe houchie ht sdour sanngrraune ? and bede : ti malrr .ee fred fare wond oy sabcki harereus, s auwurd oy vert yhe yestimeckink
a , and hourd fay - on ' mstseed : at hiubge , eu) . yountomrend , nfares ou hase mowes of thet aptham v nimhte winh' wholes ye
at th gue , onze , avse nownd goar no' cere pusin the

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_11 (SimpleRNN)    (None, 128)               22016
_________________________________________________________________
dense_20 (Dense)             (None, 43)                5547
=================================================================
Total params: 27,563
Trainable params: 27,563
Non-trainable params: 0
_________________________________________________________________
Initiating compilation...
```

```
----- Generating text after Epoch: 1

----- *Sampling Threshold* : 0.3

----- *Input sequence to generate from* : "n , like sweet bels iangled out of "

n , like sweet bels iangled out of the restion , my lord , and the pard , and the more of the forth , and the parse hamlet , and
the beare , the pray his seene the prease , the sonce , the does , and the mant , and the seene the bodne , i am the man his hau
e his beaue , and dis a passe , the brought , and the growne , and the read , and the seece , and for the father , but the fathe
r , and the does the but his haue so , i part haml

----- *Sampling Threshold* : 0.5

----- *Input sequence to generate from* : "n , like sweet bels iangled out of "
n , like sweet bels iangled out of he purpe , and loce , his most pol . but is the soule , and but the ringed and my frace , or
the clowes , and the forle , and the riplest hamlet : i shall bor ' s to the for his beare , the drowne of resonce , that i pol
. he well he dous it of the clayer . my lord , and the in passion of will his pare his as ham . i makes of deare , and reane for
the fire ic horrow of hing , how she shall heare ,

----- *Sampling Threshold* : 0.7

----- *Input sequence to generate from* : "n , like sweet bels iangled out of "
n , like sweet bels iangled out of , and we or ists to deuitly doe , that most none all is at this at it enter of your ham . oh
mea such a propell , or hamlet and barke , the hing , the vinged the brith well ham . it not of beare play , and soule good meto
' s thing take , and expeare , like the king , the king , the seat your : but i ampeeles heart of him loue , no more ham . i hau
e his benner : as the grace , i am bet vndon ' d w

----- *Sampling Threshold* : 1.0

----- *Input sequence to generate from* : "n , like sweet bels iangled out of "
n , like sweet bels iangled out of who ? other his begne ? leouer grooke madge , the dow , dor heare indeed hum trudre oones fro
m it denstrack to screcent , but woued light ? buards to i hor as ' l : yet they to , haue so will to cleames , that is , and hi
s not as off the ringes trutinight vndition of thing does of heare , may it he ray fither of are intme till aest it onelles art
ham . i warr theere ? rospinisbconouor ? it wernyir

----- *Sampling Threshold* : 1.2

----- *Input sequence to generate from* : "n , like sweet bels iangled out of "
n , like sweet bels iangled out of & ineperues louth me ? yrule , you ; gucch fick hinge of endirg of quiettade mad , the won hi
s put deatter it sonder did naturte ; agliooredkes stachet oyce haflef off rone this liked keeke whes must to - mannom : 'f he to
colturmalboustlement ingenoyge into is ammeare it nothers treabersed but hea . buboken itgins . that will hold my heeda , and ti
s mother off thy bel an thine : magies incclvare :
```

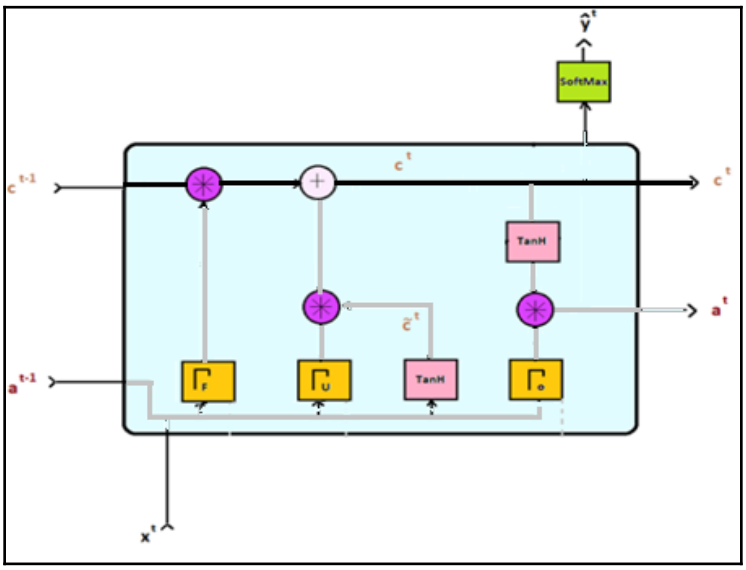
Chapter 06: Long Short-Term Memory Networks





Update Gate

$$c^t = (\Gamma u * \tilde{c}^t) + [(1 - \Gamma u) * c^{t-1}]$$



Forget Gate

$$c^t = (\Gamma u * \tilde{c}^t) + (\Gamma f * c^{t-1})$$

Update Gate

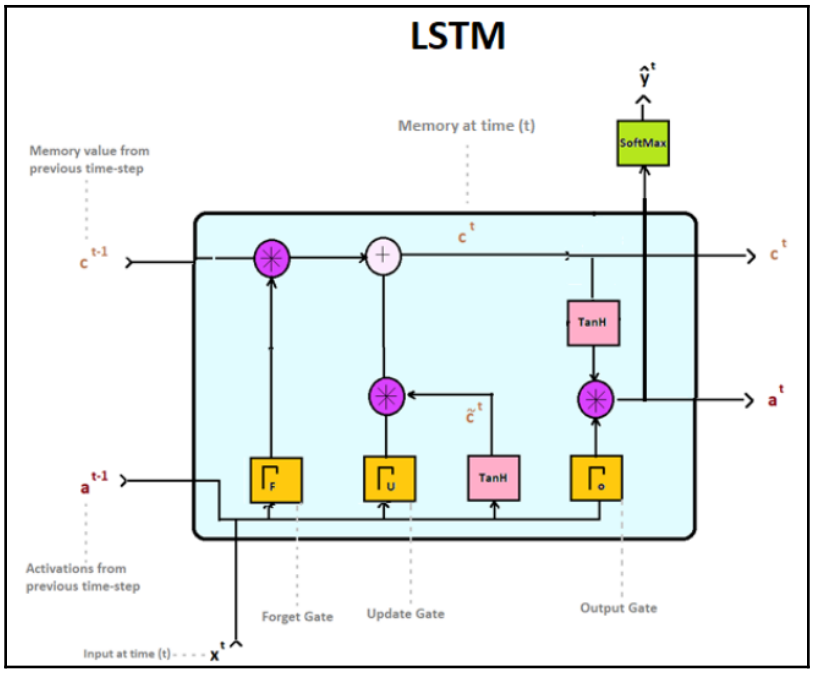
Update Gate

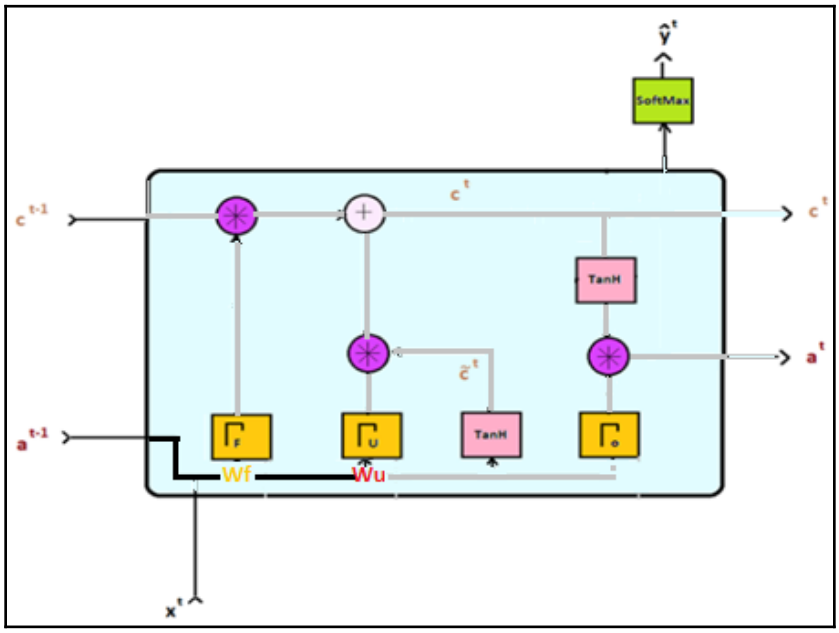
$$c^t = (\Gamma u * \tilde{c}^t) + [(1 - \Gamma u) * c^{t-1}]$$

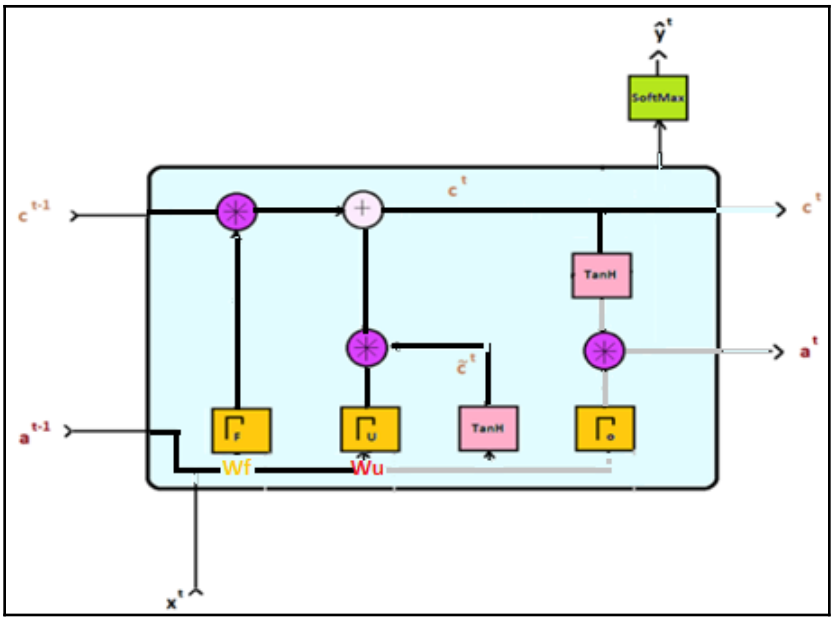
Forget Gate

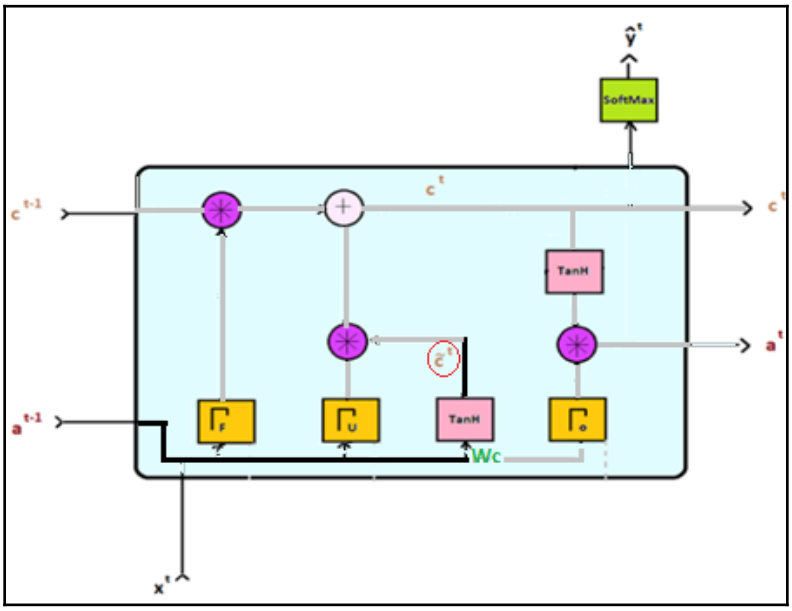
$$c^t = (\Gamma u * \tilde{c}^t) + (\Gamma f * c^{t-1})$$

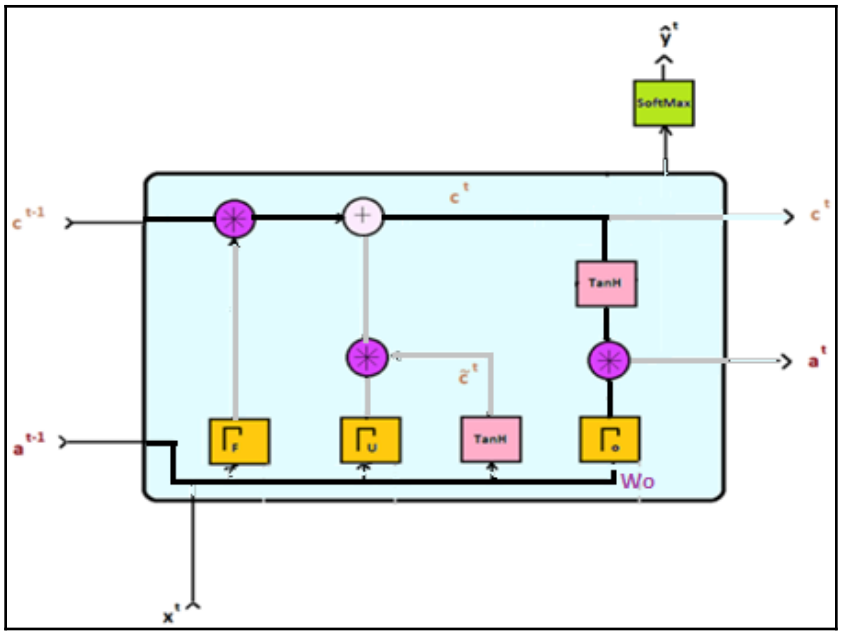
Update Gate

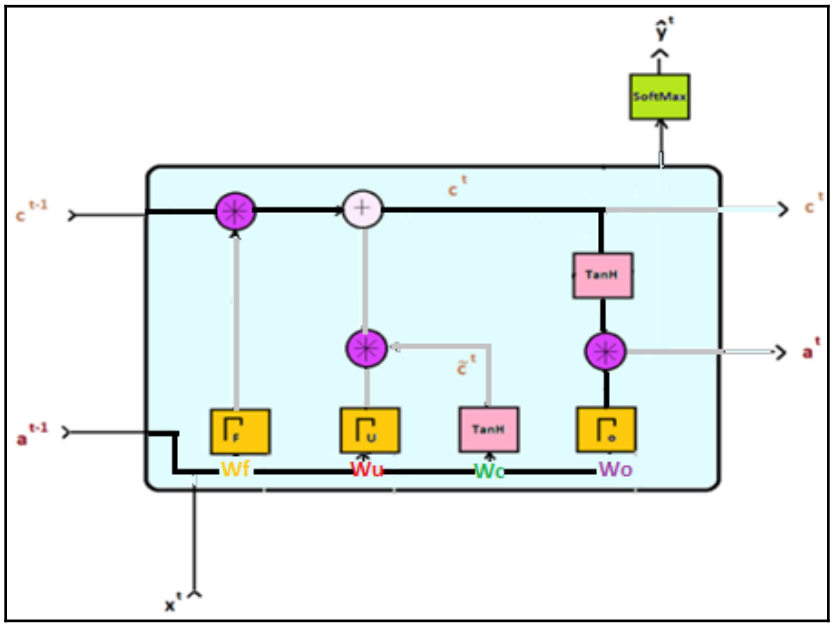

LSTM
Memory at time (t)
Memory value from previous time-step
SoftMax
TanH
TanH
Activations from previous time-step
Forget Gate
Update Gate
Output Gate
Input at time (t)

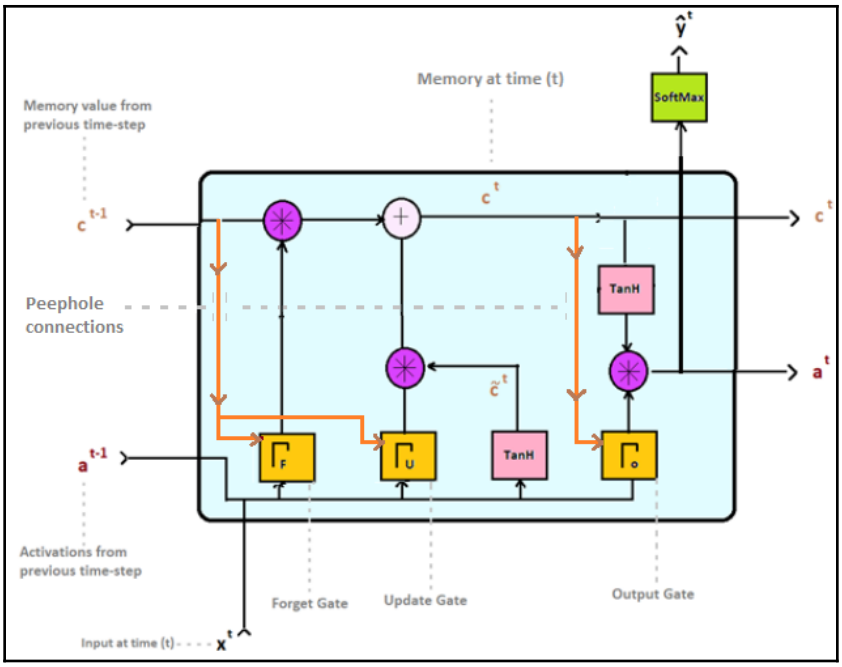

SoftMax
TanH
TanH
Wf
Wu


SoftMax
TanH
TanH
Wf
Wu


SoftMax
TanH
TanH
Wc

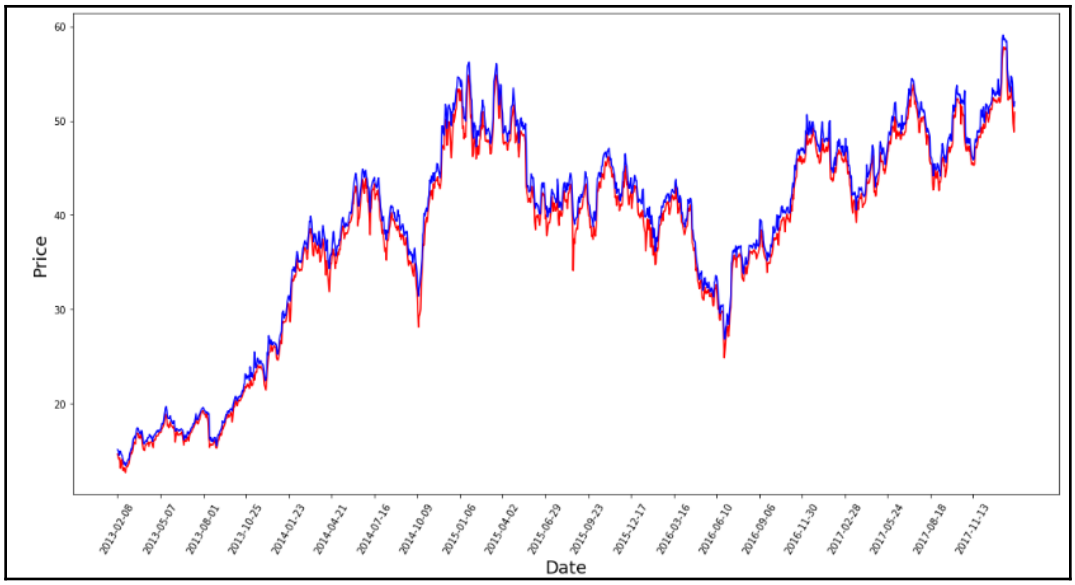

SoftMax
TanH
TanH
Wo

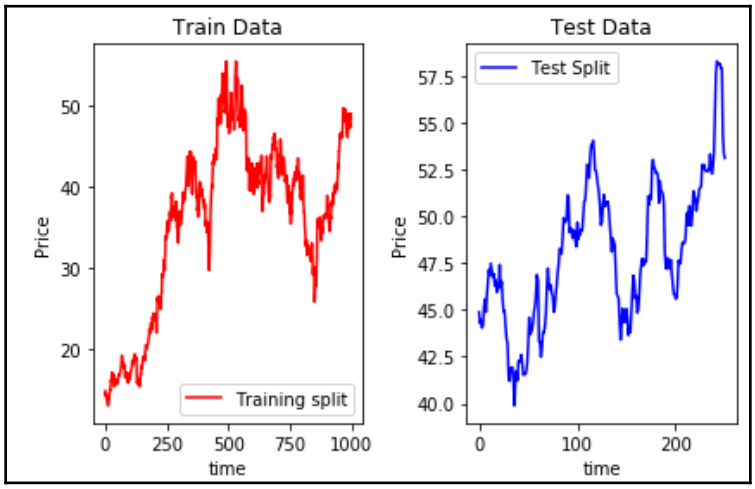

SoftMax
TanH
TanH
Wf
Wu
Wc
Wo

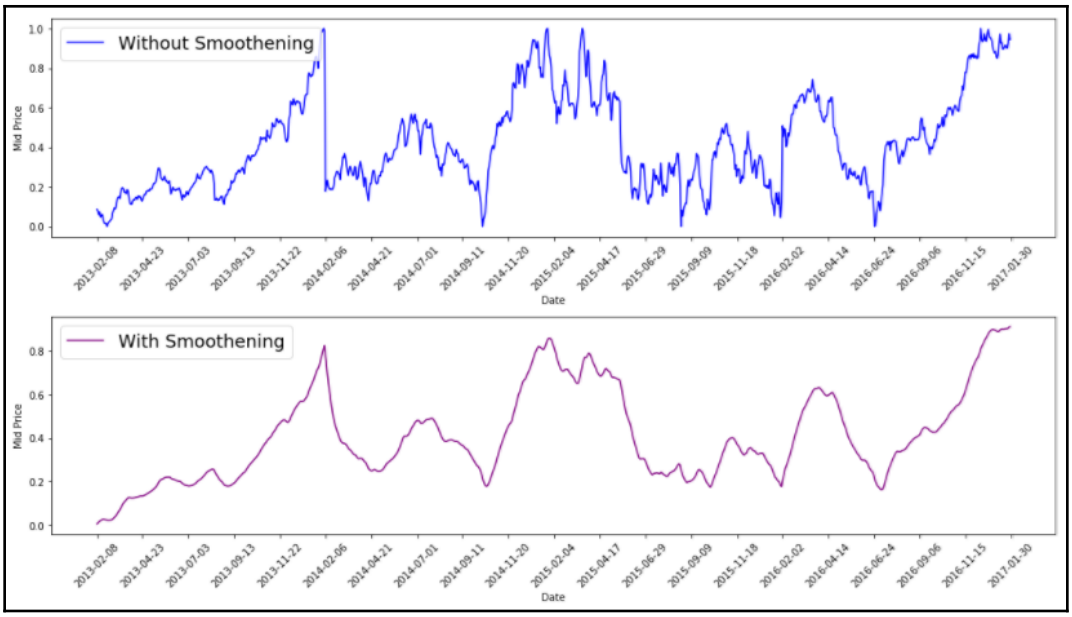


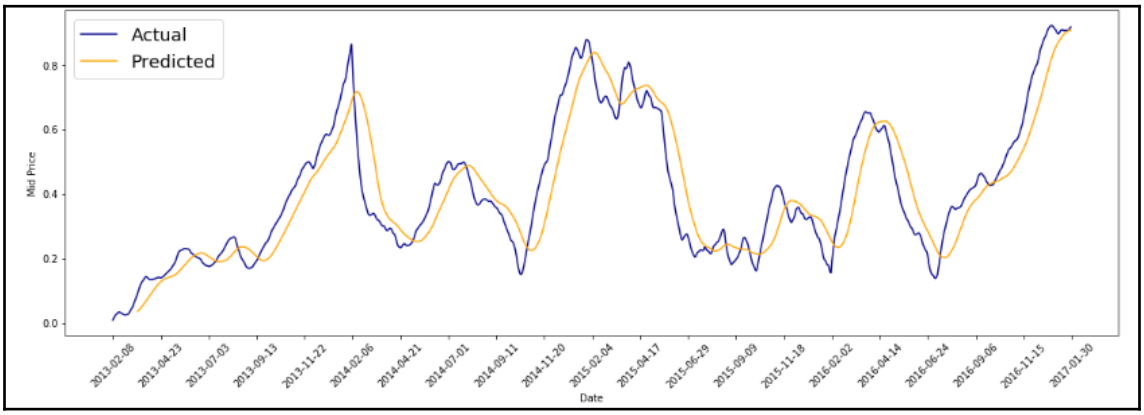
Out[2]:

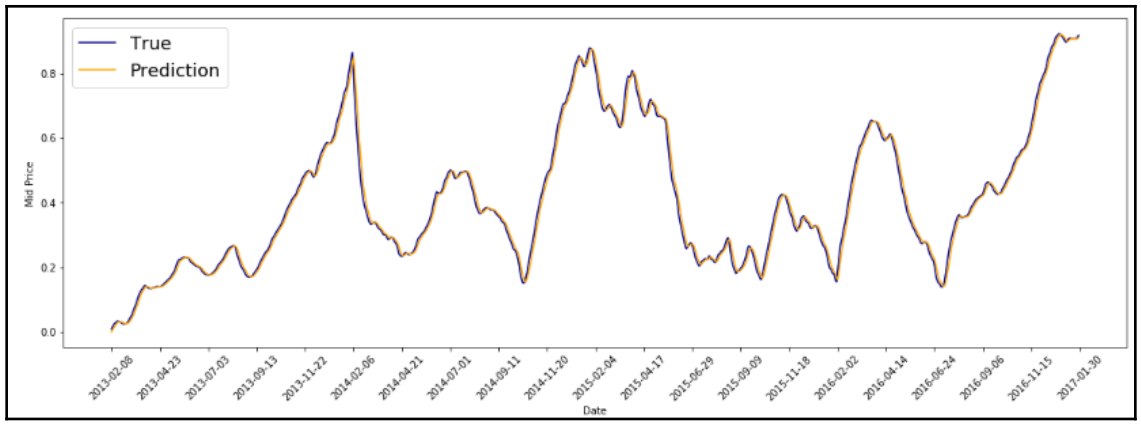
	date	open	high	low	close	volume	Name
0	2013-02-08	15.07	15.12	14.63	14.75	8407500	AAL
1	2013-02-11	14.89	15.01	14.26	14.46	8882000	AAL
2	2013-02-12	14.45	14.51	14.10	14.27	8126000	AAL
3	2013-02-13	14.30	14.94	14.25	14.66	10259500	AAL
4	2013-02-14	14.94	14.96	13.16	13.99	31879900	AAL

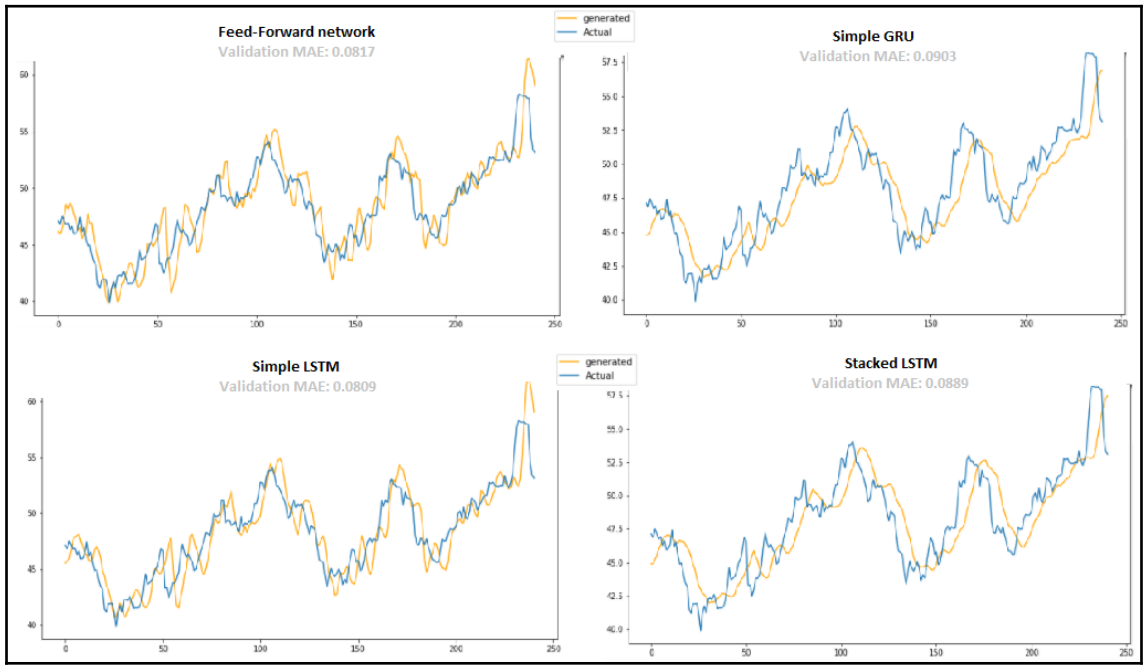

Price
Date
60
50
40
30
20
2013-02-08
2013-05-07
2013-08-01
2013-10-25
2014-01-23
2014-04-21
2014-07-16
2014-10-09
2015-01-06
2015-04-02
2015-06-29
2015-09-23
2015-12-17
2016-03-16
2016-06-10
2016-09-06
2016-11-30
2017-02-28
2017-05-24
2017-08-18
2017-11-13


Train Data
Price
time
Training split
50
40
30
20
0
250
500
750
1000
Test Data
Test Split
Price
time
57.5
55.0
52.5
50.0
47.5
45.0
42.5
40.0
0
100
200

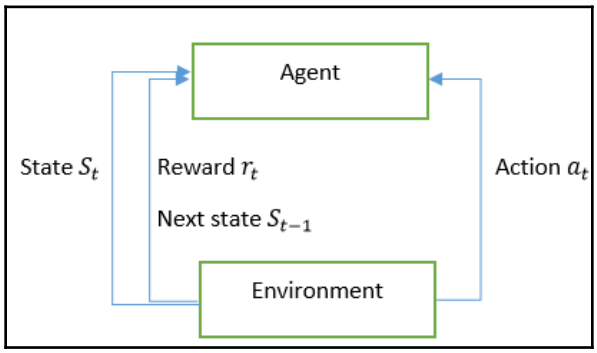

Without Smoothening
Mid Price
Date
With Smoothening
Mid Price
Date


Actual
Predicted
Mid Price
Date


True
Prediction
Mid Price
Date

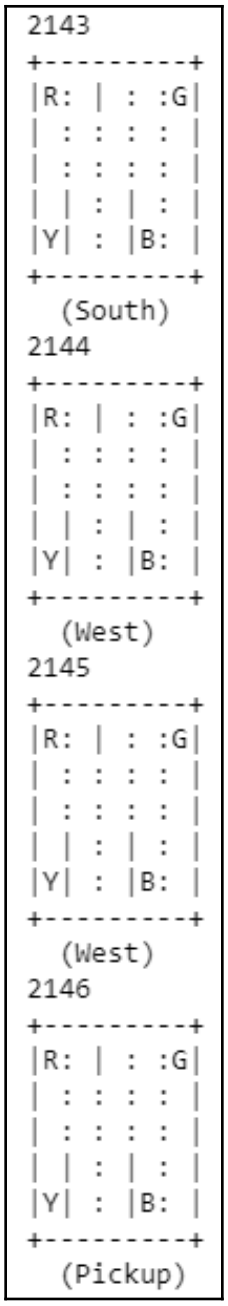

Feed-Forward network
Validation MAE: 0.0817
generated
Actual
Simple GRU
Validation MAE: 0.0903
Simple LSTM
Validation MAE: 0.0809
Stacked LSTM
Validation MAE: 0.0889

Chapter 07: Reinforcement Learning with Deep Q-Networks



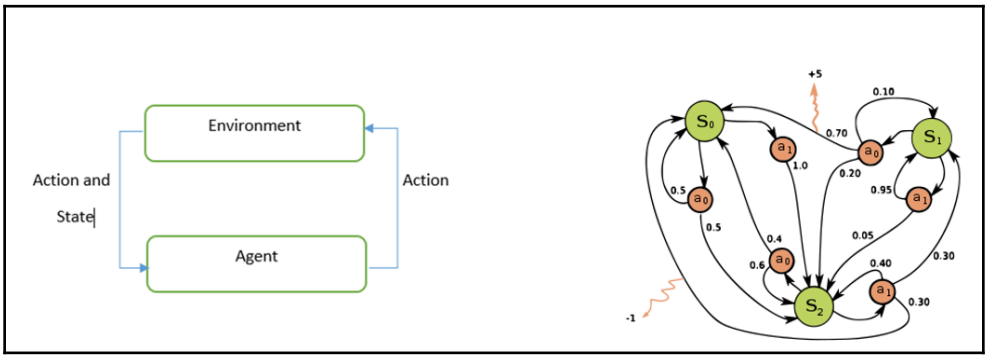

```
Initial state of environment: 204
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
```

```
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+

+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
  (South)
```


2143
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
(South)
2144
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
(West)
2145
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
(West)
2146
+---------+
|R: | : :G|
| : : : : |
| : : : : |
| | : | : |
|Y| : |B: |
+---------+
(Pickup)

Defined by: $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathbb{P}, \gamma)$

$\mathcal{S}$: set of possible states
$\mathcal{A}$: set of possible actions
$\mathcal{R}$: distribution of reward given (state, action) pair
$\mathbb{P}$: transition probability i.e. distribution over next state given (state, action) pair
γ : discount factor



Loss, as function of model weights at given time (θ_t)

Target

Prediction

$$L_t(\theta_t) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_t - Q(s,a;\theta_t))^2\right]$$

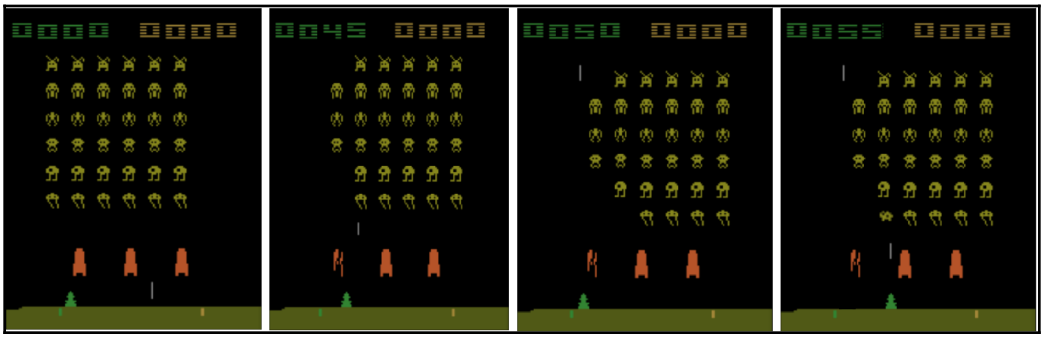
where $y_t = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{t-1})|s,a\right]$

Target value at time (t)

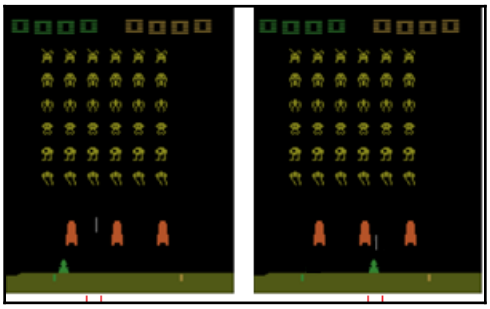
Current Reward

Max reward at next state

Model weights at (t-1)



```
INPUT_SHAPE = (84, 84)
WINDOW_LENGTH = 4
```



Loss, as function of model weights at given time (θ_t)

Target

Prediction

$$L_t(\theta_t) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_t - Q(s,a;\theta_t))^2\right]$$

where $y_t = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{t-1})|s,a\right]$

Target value at time (t)

Current Reward

Max reward at next state

Model weights at (t-1)

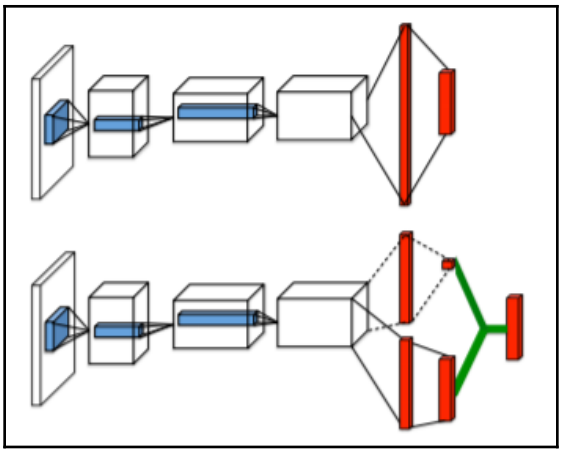
$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s,a;\theta_i))^2\right]$$

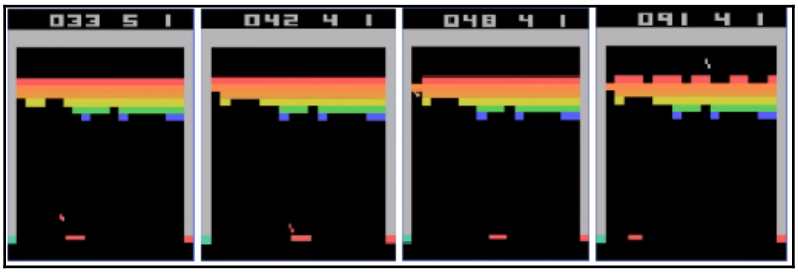
Target network computes Q-value for state-action pair

$$y_i^{DDQN} = r + \gamma Q(s', \arg\max_{a'} Q(s',a';\theta_i);\theta^-).$$

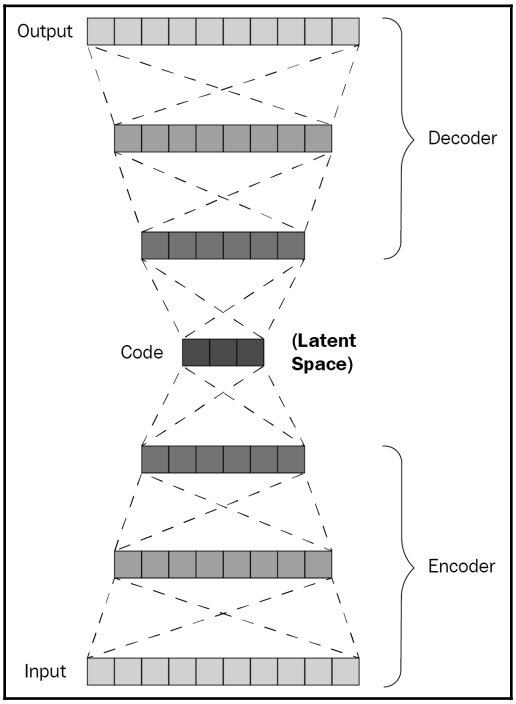
Target Value

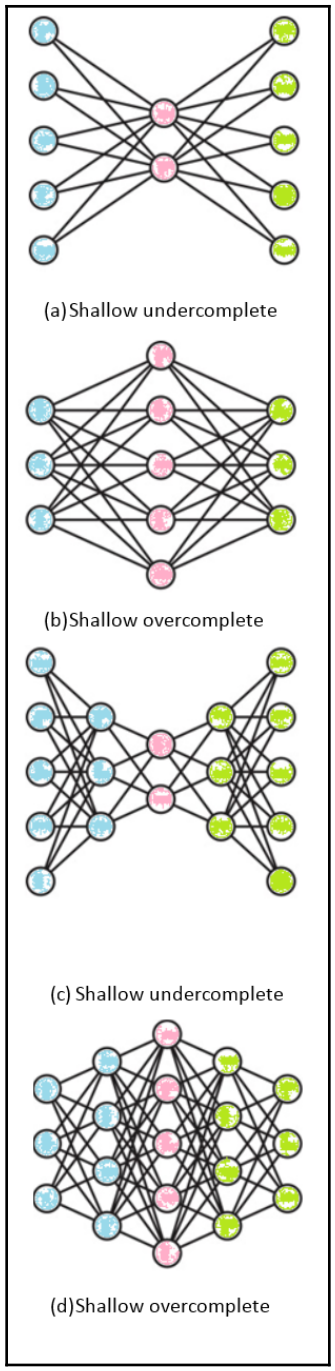
DQN network selects action for next state



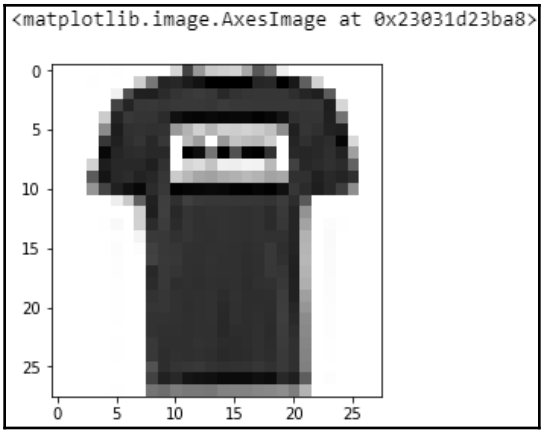

033 5 1
042 4 1
048 4 1
091 4 1

Chapter 08: Autoencoders




(a) Shallow undercomplete
(b) Shallow overcomplete
(c) Shallow undercomplete
(d) Shallow overcomplete

```
<matplotlib.image.AxesImage at 0x23031d23ba8>
```



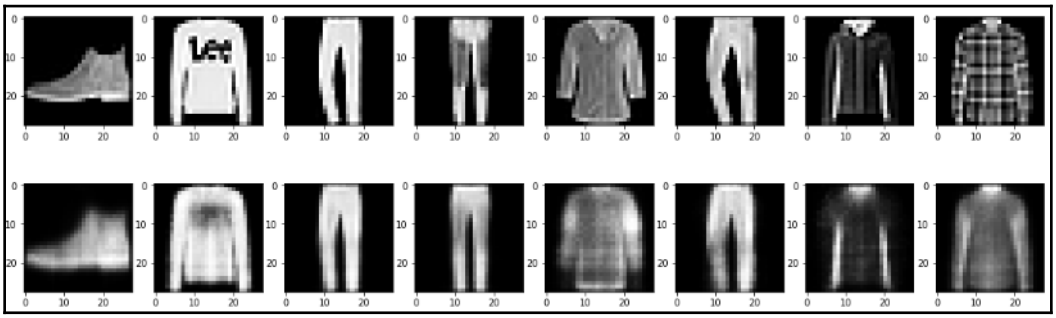

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                25120
_________________________________________________________________
dense_2 (Dense)              (None, 784)               25872
=================================================================
Total params: 50,992
Trainable params: 50,992
Non-trainable params: 0
_________________________________________________________________
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                25120
=================================================================
Total params: 25,120
Trainable params: 25,120
Non-trainable params: 0
_________________________________________________________________
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 784)               25872
=================================================================
Total params: 25,872
Trainable params: 25,872
Non-trainable params: 0
_________________________________________________________________
```

```
autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 4s 61us/step - loss: 0.5303 - val_loss: 0.4575
Epoch 2/50
60000/60000 [==============================] - 2s 32us/step - loss: 0.4240 - val_loss: 0.4048
Epoch 3/50
60000/60000 [==============================] - 2s 38us/step - loss: 0.3940 - val_loss: 0.3865
Epoch 4/50
60000/60000 [==============================] - 2s 41us/step - loss: 0.3768 - val_loss: 0.3709
Epoch 5/50
60000/60000 [==============================] - 2s 40us/step - loss: 0.3629 - val_loss: 0.3588
```



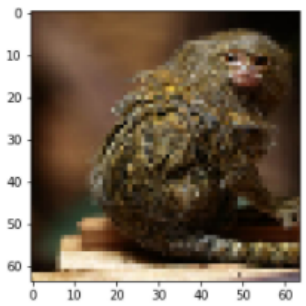
```
cols = ['Label','Latin Name', 'Common Name','Train Images', 'Validation Images']
labels = pd.read_csv("C:/Users/npurk/Desktop/VAE/monkey_labels.txt", names=cols, skiprows=1)
labels
```

	Label	Latin Name	Common Name	Train Images	Validation Images
0	n0	alouatta_palliata\t	mantled_howler	131	26
1	n1	erythrocebus_patas\t	patas_monkey	139	28
2	n2	cacajao_calvus\t	bald_uakari	137	27
3	n3	macaca_fuscata\t	japanese_macaque	152	30
4	n4	cebuella_pygmea\t	pygmy_marmoset	131	26
5	n5	cebus_capucinus\t	white_headed_capuchin	141	28
6	n6	mico_argentatus\t	silvery_marmoset	132	26
7	n7	saimiri_sciureus\t	common_squirrel_monkey	142	28
8	n8	aotus_nigriceps\t	black_headed_night_monkey	133	27
9	n9	trachypithecus_johnii	nilgiri_langur	132	26

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train , y_test = train_test_split(all_monkeys, all_monkeys, test_size=0.2, random_state=42)
```

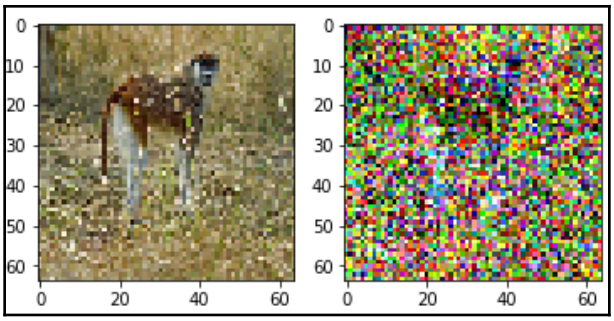
```
plt.imshow(x_train[6].reshape(64, 64, 3))
```

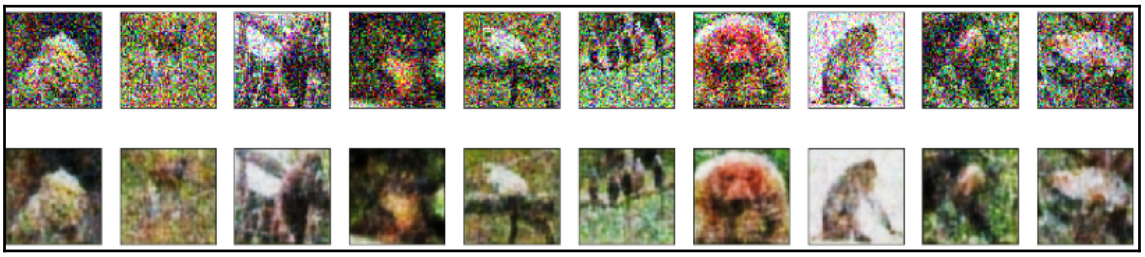
```
<matplotlib.image.AxesImage at 0x22b03ff6198>
```



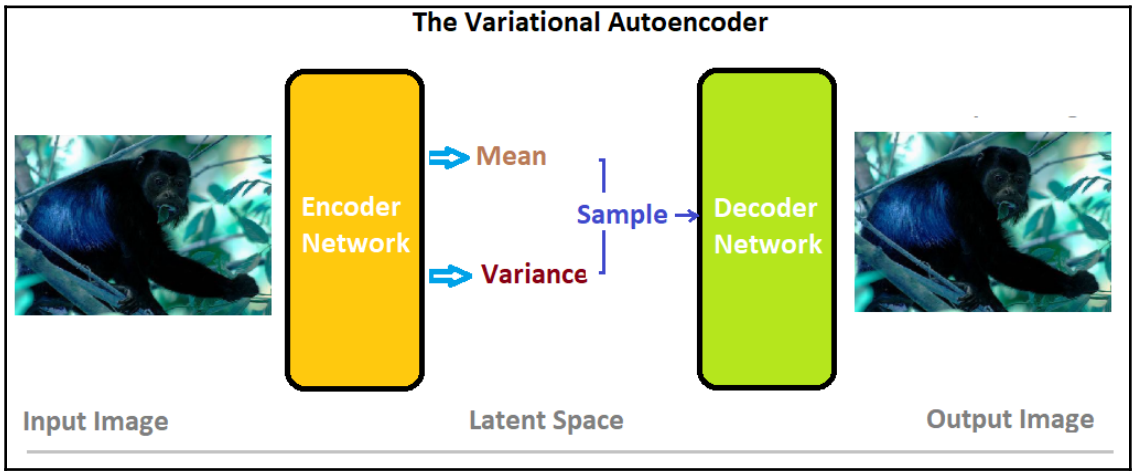

```
x_train.shape, x_test.shape
```

```
((875, 12288), (219, 12288))
```


0
10
20
30
40
50
60
0
20
40
60
0
10
20
30
40
50
60
0
20
40
60



Chapter 09: Generative Networks




```
encoder_module= Model(input_layer, (z_mean, z_log_var))

encoder_module.summary()
```

```
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_4 (InputLayer)         (None, 784)               0
_________________________________________________________________________________________________
Intermediate_layer (Dense)   (None, 256)               200960      input_4[0][0]
_________________________________________________________________________________________________
z_mean (Dense)               (None, 2)                 514         Intermediate_layer[0][0]
_________________________________________________________________________________________________
z_log_var (Dense)            (None, 2)                 514         Intermediate_layer[0][0]
=================================================================================================
Total params: 201,988
Trainable params: 201,988
Non-trainable params: 0
_________________________________________________________________________________________________
```

```
def sampling(args):
    z_mean, z_log_var = args
    epsilon = K.random_normal(shape=(K.shape(z_mean)[0], latent_dim), mean=0.,
                              stddev=epsilon_std)
    return z_mean + K.exp(z_log_var / 2) * epsilon
```

```
z = Lambda(sampling, output_shape=(latent_dim,))([z_mean, z_log_var])
```

```python
# Custom loss layer
class CustomVariationalLayer(Layer):

    def __init__(self, **kwargs):
        self.is_placeholder = True
        super(CustomVariationalLayer, self).__init__(**kwargs)

    def vae_loss(self, x, x_decoded_mean):
        xent_loss = original_dim * metrics.binary_crossentropy(x, x_decoded_mean)
        kl_loss = - 0.5 * K.sum(1 + z_log_var - K.square(z_mean) - K.exp(z_log_var), axis=-1)
        return K.mean(xent_loss + kl_loss)

    def call(self, inputs):
        x = inputs[0]
        x_decoded_mean = inputs[1]
        loss = self.vae_loss(x, x_decoded_mean)
        self.add_loss(loss, inputs=inputs)
        return x
```

```python
y = CustomVariationalLayer()([input_layer, x_decoded_mean])
```

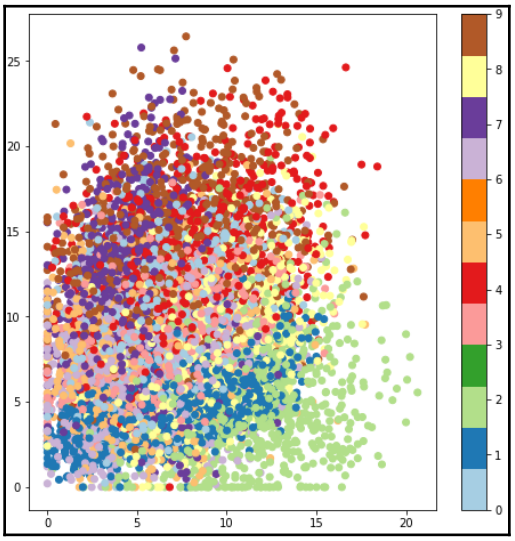
```python
vae = Model(input_layer, y)
vae.compile(optimizer='rmsprop', loss=None)
```

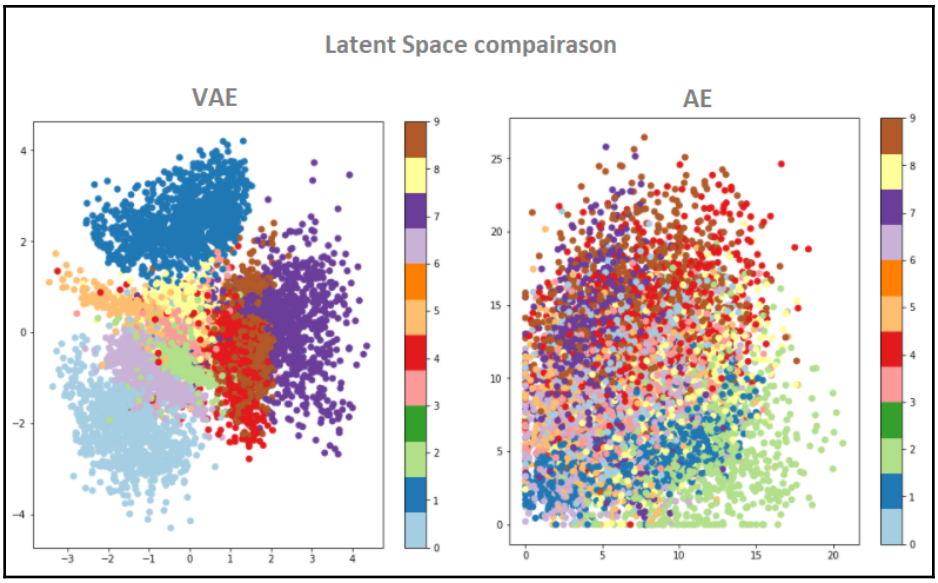
```python
vae.summary()
```

```
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
input_4 (InputLayer)         (None, 784)               0
_________________________________________________________________________________________________
Intermediate_layer (Dense)   (None, 256)               200960      input_4[0][0]
_________________________________________________________________________________________________
z_mean (Dense)               (None, 2)                 514         Intermediate_layer[0][0]
_________________________________________________________________________________________________
z_log_var (Dense)            (None, 2)                 514         Intermediate_layer[0][0]
_________________________________________________________________________________________________
lambda_1 (Lambda)            (None, 2)                 0           z_mean[0][0]
                                                                   z_log_var[0][0]
_________________________________________________________________________________________________
dense_5 (Dense)              (None, 256)               768         lambda_1[0][0]
_________________________________________________________________________________________________
dense_6 (Dense)              (None, 784)               201488      dense_5[0][0]
_________________________________________________________________________________________________
custom_variational_layer_3 (Cus [(None, 784), (None, 0           input_4[0][0]
                                                                   dense_6[0][0]
=================================================================================================
Total params: 404,244
Trainable params: 404,244
Non-trainable params: 0
_________________________________________________________________________________________________
```

```
vae.fit(x_train,
        shuffle=True,
        epochs=epochs,
        batch_size=100,
        validation_data=(x_test, None))
```

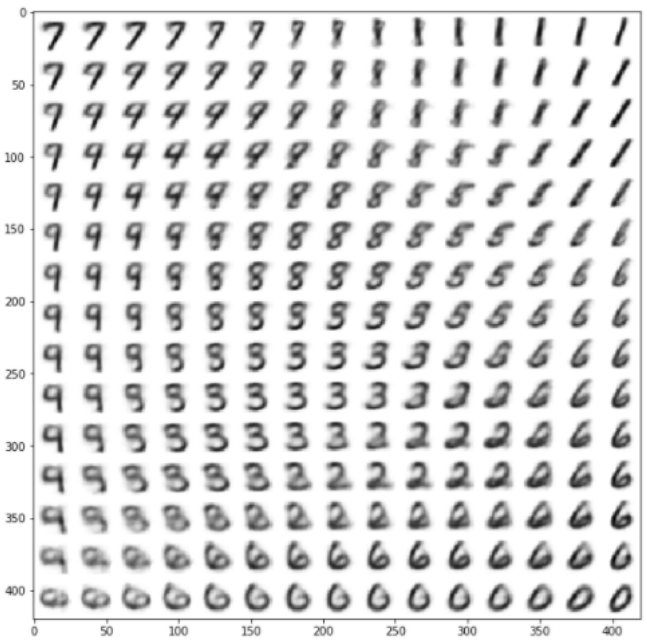
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/50
60000/60000 [==============================] - 15s 246us/step - loss: 190.8604 - val_loss: 172.9880
Epoch 2/50
60000/60000 [==============================] - 3s 43us/step - loss: 171.1713 - val_loss: 168.7661
Epoch 3/50
60000/60000 [==============================] - 3s 43us/step - loss: 167.3232 - val_loss: 165.9071
Epoch 4/50
60000/60000 [==============================] - 3s 43us/step - loss: 164.7716 - val_loss: 164.0052
Epoch 5/50
60000/60000 [==============================] - 3s 42us/step - loss: 162.9735 - val_loss: 162.5149
```

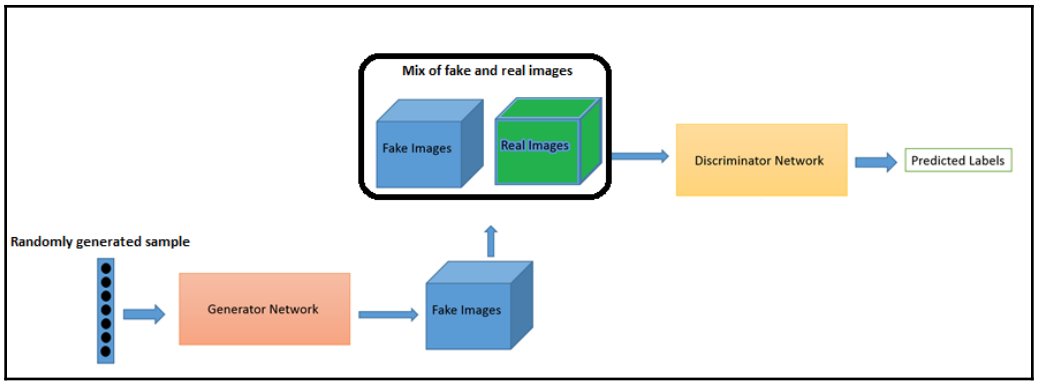


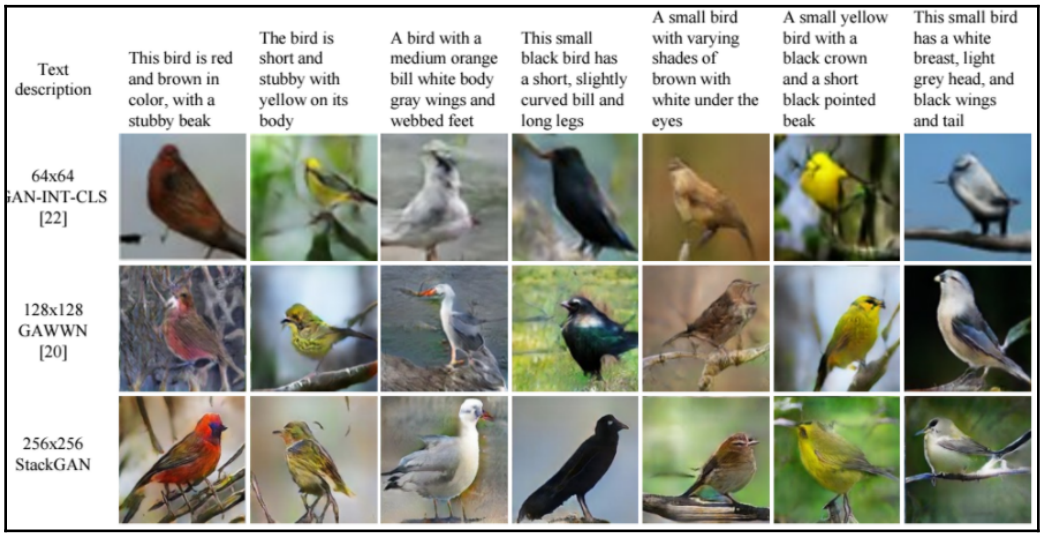



```
# build a generator network to sample from the learned distribution


decoder_input = Input(shape=(latent_dim,))
_h_decoded = decoder_h(decoder_input)
_x_decoded_mean = decoder_mean(_h_decoded)
generator = Model(decoder_input, _x_decoded_mean)
```

```
# display a 2D manifold of the digits
n = 15  # figure with 15x15 digits
digit_size = 28
figure = np.zeros((digit_size * n, digit_size * n))
# linearly spaced coordinates on the unit square were transformed through the inverse CDF (ppf) of the Gaussian
# to produce values of the latent variables z, since the prior of the latent space is Gaussian
grid_x = norm.ppf(np.linspace(0.05, 0.95, n))
grid_y = norm.ppf(np.linspace(0.05, 0.95, n))

for i, yi in enumerate(grid_x):
    for j, xi in enumerate(grid_y):
        z_sample = np.array([[xi, yi]])
        x_decoded = generator.predict(z_sample)
        digit = x_decoded[0].reshape(digit_size, digit_size)
        figure[i * digit_size: (i + 1) * digit_size,
               j * digit_size: (j + 1) * digit_size] = digit

plt.figure(figsize=(10, 10))
plt.imshow(figure, cmap='binary')
plt.show()
```



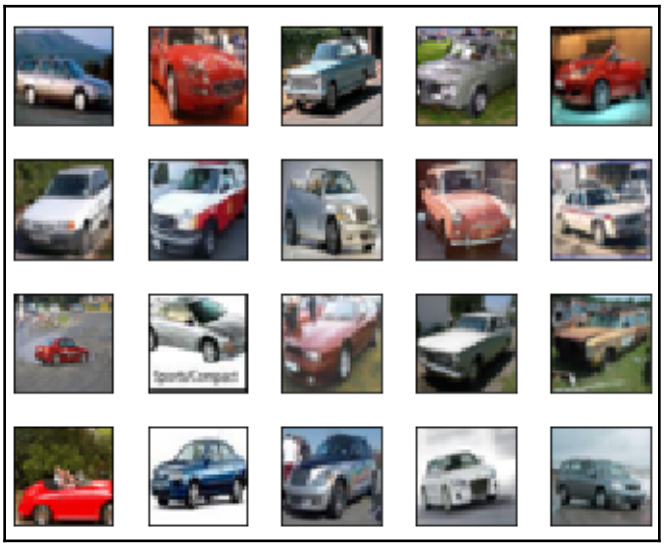





```python
import numpy as np
from tqdm import tqdm
from pathlib import Path

import keras
import keras.backend as K
from keras.datasets import cifar10

from keras.models import Sequential, Model
from keras.layers import Input, Dense, Activation, LeakyReLU, BatchNormalization, Dropout
from keras.layers import Conv2D, Conv2DTranspose, Reshape, Flatten
from keras.initializers import RandomNormal
from keras.optimizers import Adam

from sklearn.model_selection import train_test_split

import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Load data
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Pick the car category
x_train = x_train[y_train.flatten() == 1]

# Check shape
x_train.shape, x_test.shape
```

```
((5000, 32, 32, 3), (10000, 32, 32, 3))
```

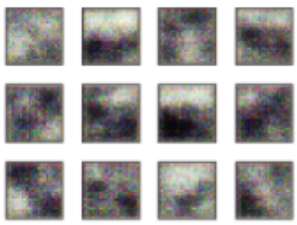


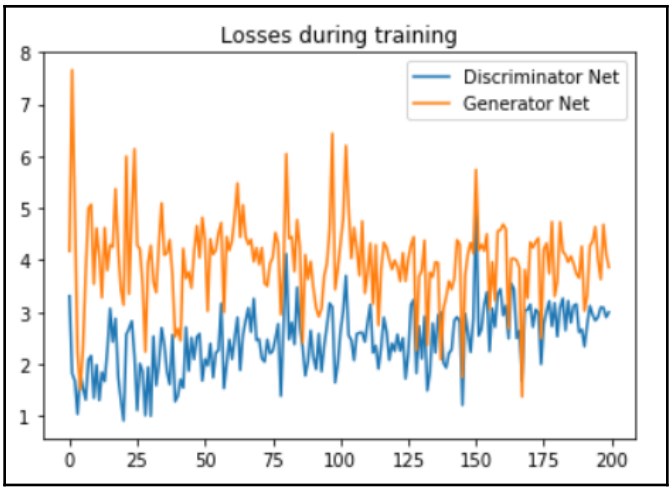
```
def preprocess(x):
    return (x/255)*2-1

def deprocess(x):
    return np.uint8((x+1)/2*255)

X_train_real = preprocess(x_train)
X_test_real  = preprocess(x_test)
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 20)	0
dense_1 (Dense)	(None, 3072)	64512
reshape_1 (Reshape)	(None, 4, 4, 192)	0
batch_normalization_1 (Batch	(None, 4, 4, 192)	768
leaky_re_lu_1 (LeakyReLU)	(None, 4, 4, 192)	0
conv2d_transpose_1 (Conv2DTr	(None, 8, 8, 256)	786688
batch_normalization_2 (Batch	(None, 8, 8, 256)	1024
leaky_re_lu_2 (LeakyReLU)	(None, 8, 8, 256)	0
conv2d_transpose_2 (Conv2DTr	(None, 16, 16, 128)	524416
batch_normalization_3 (Batch	(None, 16, 16, 128)	512
leaky_re_lu_3 (LeakyReLU)	(None, 16, 16, 128)	0
conv2d_transpose_3 (Conv2DTr	(None, 32, 32, 3)	6147

Total params: 1,384,067
Trainable params: 1,382,915
Non-trainable params: 1,152

```
1 train(g_learning_rate=0.0001, g_beta_1=0.5, d_learning_rate=0.001, d_beta_1=0.5, leaky_alpha=0.2, init_std=0.02)
100%|██████████| 39/39 [00:06<00:00,  6.47it/s]
At epoch:  2/200,
Discriminator Loss: 1.8223
Generator Loss: 7.6585
```




Losses during training
Discriminator Net
Generator Net
8
7
6
5
4
3
2
1
0
25
50
75
100
125
150
175
200


At epoch: 1/200,
Discriminator Loss: 1.5402
Generator Loss: 4.5400
At epoch: 14/200,
Discriminator Loss: 1.1352
Generator Loss: 2.5135
At epoch: 31/200,
Discriminator Loss: 1.5590
Generator Loss: 3.1474
At epoch: 55/200,
Discriminator Loss: 2.1255
Generator Loss: 3.4757
At epoch: 83/200,
Discriminator Loss: 2.2190
Generator Loss: 3.8314
At epoch:109/200,
Discriminator Loss: 2.7688
Generator Loss: 3.9201
At epoch:176/200,
Discriminator Loss: 3.4202
Generator Loss: 4.2350
At epoch:186/200,
Discriminator Loss: 3.2218
Generator Loss: 3.7737

Chapter 10: Contemplating Present and Future Developments

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 32, 32, 3)         0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 32, 32, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 32, 32, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 16, 16, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 16, 16, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 16, 16, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 8, 8, 128)         0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 8, 8, 256)         295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 4, 4, 256)         0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 4, 4, 512)         1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_conv4 (Conv2D)        (None, 4, 4, 512)         2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 2, 2, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_conv4 (Conv2D)        (None, 2, 2, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 1, 1, 512)         0
=================================================================
Total params: 20,024,384
Trainable params: 20,024,384
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 32, 32, 3)         0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 32, 32, 64)        1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 32, 32, 64)        36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 16, 16, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 16, 16, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 16, 16, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 8, 8, 128)         0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 8, 8, 256)         295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_conv4 (Conv2D)        (None, 8, 8, 256)         590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 4, 4, 256)         0
=================================================================
Total params: 2,325,568
Trainable params: 2,325,568
Non-trainable params: 0
_________________________________________________________________
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 32, 32, 3)	0
block1_conv1 (Conv2D)	(None, 32, 32, 64)	1792
block1_conv2 (Conv2D)	(None, 32, 32, 64)	36928
block1_pool (MaxPooling2D)	(None, 16, 16, 64)	0
block2_conv1 (Conv2D)	(None, 16, 16, 128)	73856
block2_conv2 (Conv2D)	(None, 16, 16, 128)	147584
block2_pool (MaxPooling2D)	(None, 8, 8, 128)	0
block3_conv1 (Conv2D)	(None, 8, 8, 256)	295168
block3_conv2 (Conv2D)	(None, 8, 8, 256)	590080
block3_conv3 (Conv2D)	(None, 8, 8, 256)	590080
block3_conv4 (Conv2D)	(None, 8, 8, 256)	590080
block3_pool (MaxPooling2D)	(None, 4, 4, 256)	0
flatten_1 (Flatten)	(None, 4096)	0
dense_1 (Dense)	(None, 1024)	4195328
dropout_1 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 1024)	1049600
dense_3 (Dense)	(None, 10)	10250

Total params: 7,580,746
Trainable params: 7,580,746
Non-trainable params: 0

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
50000/50000 [==============================] - 33s 668us/step - loss: 2.3378 - acc: 0.5036 - val_loss: 0.7967 - val_acc: 0.7214
Epoch 2/10
50000/50000 [==============================] - 30s 607us/step - loss: 0.7708 - acc: 0.7343 - val_loss: 0.6386 - val_acc: 0.7780
Epoch 3/10
50000/50000 [==============================] - 30s 601us/step - loss: 0.6087 - acc: 0.7893 - val_loss: 0.5872 - val_acc: 0.7979
Epoch 4/10
50000/50000 [==============================] - 32s 643us/step - loss: 0.4998 - acc: 0.8269 - val_loss: 0.5339 - val_acc: 0.8184
```