Summer Training Report

On

Room Classifier

A Project Report submitted in partial fulfilment of the requirements for the award of

Bachelor in Engineering

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

Priyadarshini

(Roll no: LCO17373)

Under the supervision of

(Er. Sudhakar Kumar)

CHANDIGARH COLLEGE OF ENGINEERING AND TECHNOLOGY

(DEGREE WING)

Government Institute under Chandigarh (UT) Administration, Affiliated to Panjab University, Chandigarh

Sector-26, Chandigarh. PIN-160019

July, 2019

CHANDIGARH COLLEGE OF ENGINEERING AND TECHNOLOGY (DEGREE WING)
Government Institute under Chandigarh (UT) Administration | Affiliated to Panjab University , Chandigarh
Sector-26, Chandigarh. PIN-160019 | Tel. No. 0172-2750947, 2750943
Website: www.ccet.ac.in | Email: principal@ccet.ac.in | Fax. No.:0172-2750872

Department of Computer Sc. & Engineering

CANDIDATE'S DECLARATION

I hereby declare that the work presented in this report entitled **"Room Classifier"**, in fulfilment of the requirement for the award of the degree Bachelor of Engineering in Computer Science & Engineering, submitted in CSE Department, Chandigarh College of Engineering & Technology(Degree wing) affiliated to Panjab University, Chandigarh, is an authentic record of my own work carried out during my degree under the guidance of **Er. Sudhakar Kumar**. The work reported in this has not been submitted by me for award of any other degree or diploma.

Date: 31 July, 2020 Priyadarshini

Place: Shimla (Himachal Pradesh) LCO17373

CHANDIGARH COLLEGE OF ENGINEERING AND TECHNOLOGY (DEGREE WING)
Government Institute under Chandigarh (UT) Administration | Affiliated to Panjab University , Chandigarh
Sector-26, Chandigarh. PIN-160019 | Tel. No. 0172-2750947, 2750943
Website: www.ccet.ac.in | Email: principal@ccet.ac.in | Fax. No. :0172-2750872

Department of Computer Sc. & Engineering

CERTIFICATE

This is to certify that the Project work entitled **"Room Classifier"** submitted by **Priyadarshini, LC017373** in fulfilment for the requirements of the award of Bachelor of Engineering Degree in Computer Science & Engineering at Chandigarh College of Engineering and Technology (Degree Wing), Chandigarh is an authentic work carried out by him under my supervision and guidance.

To the best of my knowledge, the matter embodied in the project has not been submitted to any other University / Institute for the award of any Degree.

Date:

Place:

Er. Sudhakar Kumar
Dept. of CSE
CCET (Degree Wing)
Chandigarh

CHANDIGARH COLLEGE OF ENGINEERING AND TECHNOLOGY (DEGREE WING)
Government Institute under Chandigarh (UT) Administration | Affiliated to Panjab University , Chandigarh
Sector-26, Chandigarh. PIN-160019 | Tel. No. 0172-2750947, 2750943
Website: www.ccet.ac.in | Email: principal@ccet.ac.in | Fax. No. :0172-2750872

Department of Computer Sc. & Engineering

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my mentor Mr. Sudhakar and team for guiding and helping me throughout my trainee period from 30th June to 29th July. It was a good learning experience under him. I would like to thank them for sharing their valuable knowledge and time.

I am highly indebted to my respected teachers of Chandigarh College of Engineering and Technology, Sector-26, Chandigarh for their guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

I would like to express my gratitude towards my parents for their kind co-operation and encouragement which helped me in completion of this project.

My thanks and appreciation also go to my friends in developing the project.

CHANDIGARH COLLEGE OF ENGINEERING AND TECHNOLOGY (DEGREE WING)
Government Institute under Chandigarh (UT) Administration | Affiliated to Panjab University, Chandigarh
Sector-26, Chandigarh. PIN-160019 | Tel. No. 0172-2750947, 2750943
Website: www.ccet.ac.in | Email: principal@ccet.ac.in | Fax. No. :0172-2750872

Department of Computer Sc. & Engineering

ABSTRACT

In this work we use artificial intelligence (deep learning) to detect when rooms in the house are messy or clean (via cameras & TensorFlow). This proposes a room visual detection algorithm and a novel adaptive tiling-based selective dirt area coverage scheme for reconfigurable morphology robot. The visual dirt detection technique utilizes a three-layer filtering framework which includes a periodic pattern detection filter, edge detection, and noise filtering to effectively detect and segment out the dirt area from the complex floor backgrounds. Then adaptive tiling-based area coverage scheme has been employed to generate the *tetromino* morphology to cover the segmented dirt area. The proposed algorithms have been validated in Matlab environment with real captured dirt images and simulated *tetrominoes* tile set. Experimental results show that the proposed three-stage filtering significantly enhances the dirt detection ratio in the incoming images with complex floors with different backgrounds. Further, the selective dirt area coverage is performed by excluding the already cleaned area from the unclean area on the floor map by incorporating the tiling pattern generated by adaptive *tetromino* tiling algorithm.

CONTENTS

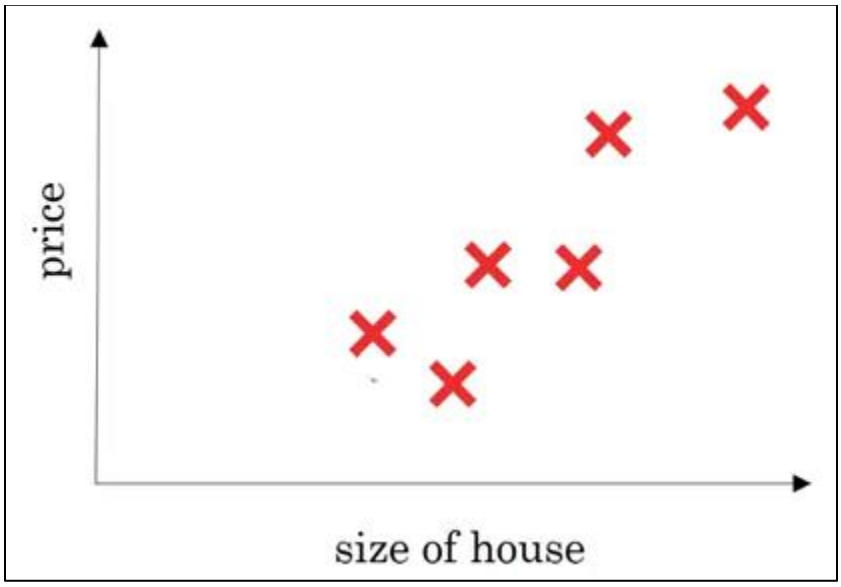
CHAPTER 1- INTRODUCTION

1.1 INTRODUCTION TO DEEP LEARNING:

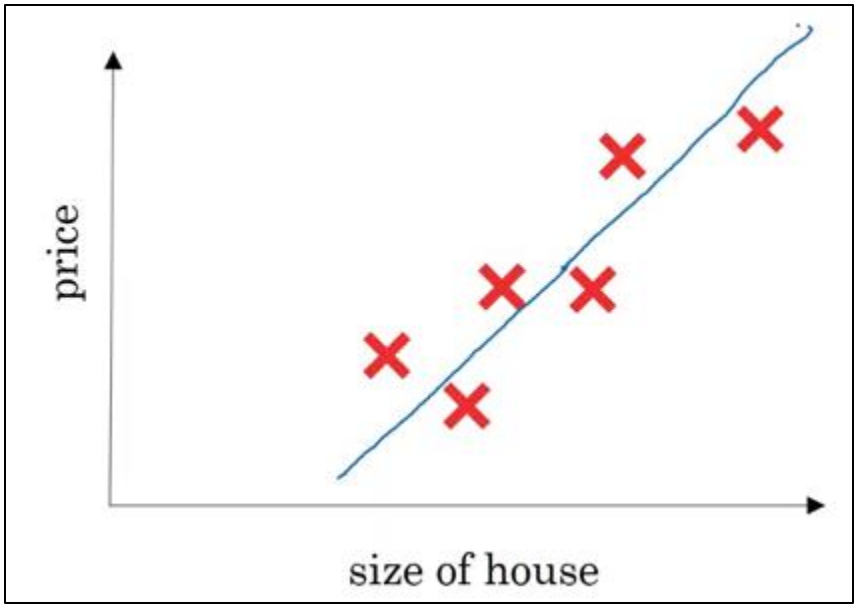
Neural Network-

Let's begin with the crux of the matter and a very critical question. What is a neural network?

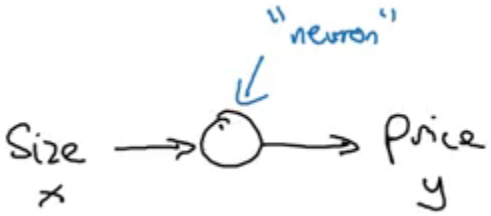
Consider an example where we have to predict the price of a house. The variables we are given are the size of the house in square feet (or square meters) and the price of the house. Now assume we have 6 houses. So first let's pull up a plot to visualize what we're looking at:



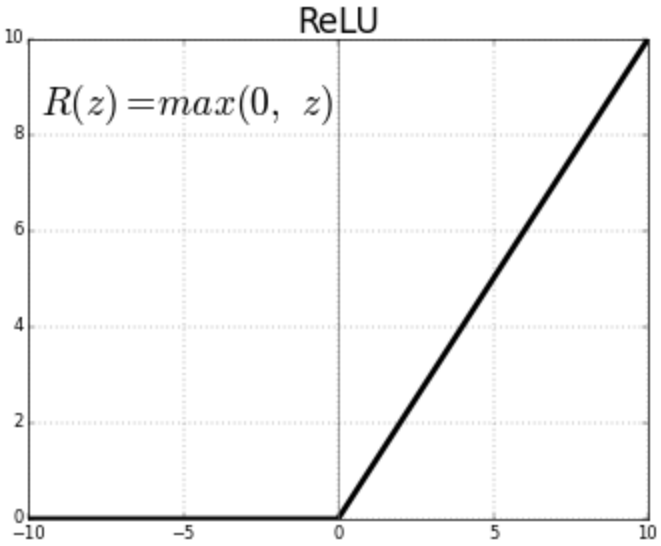
On the x-axis, we have the size of the house and on the y-axis we have it's corresponding price. A linear regression model will try to draw a straight line to fit the data:



So, the input(x) here is the size of the house and output(y) is the price. Now let's look at how we can solve this using a simple neural network:

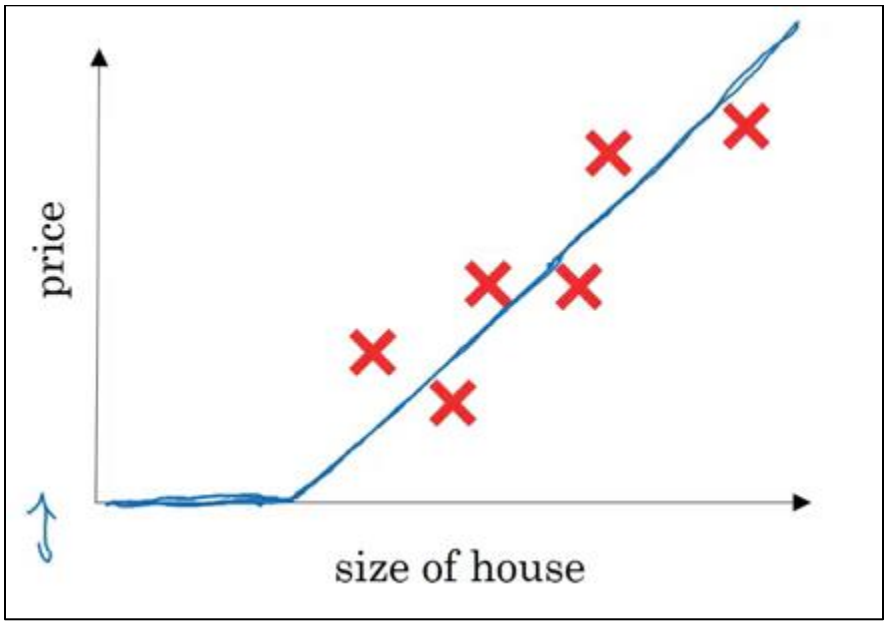


Here, a neuron will take an input, apply some activation function to it, and generate an output. One of the most commonly used activation function is ReLU (Rectified Linear Unit):



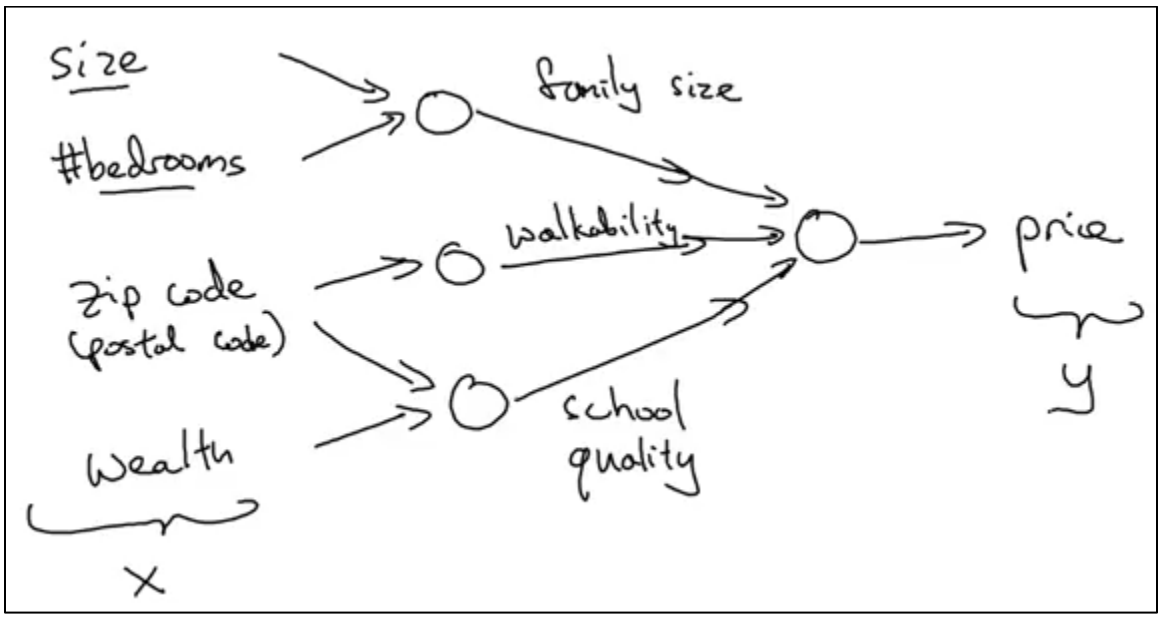
ReLU takes a real number as input and returns the maximum of 0 or that number. So, if we pass 10, the output will be 10, and if the input is -10, the output will be 0. We will discuss activation functions in detail later in this article.

For now let's stick to our example. If we use the ReLU activation function to predict the price of a house based on its size, this is how the predictions may look

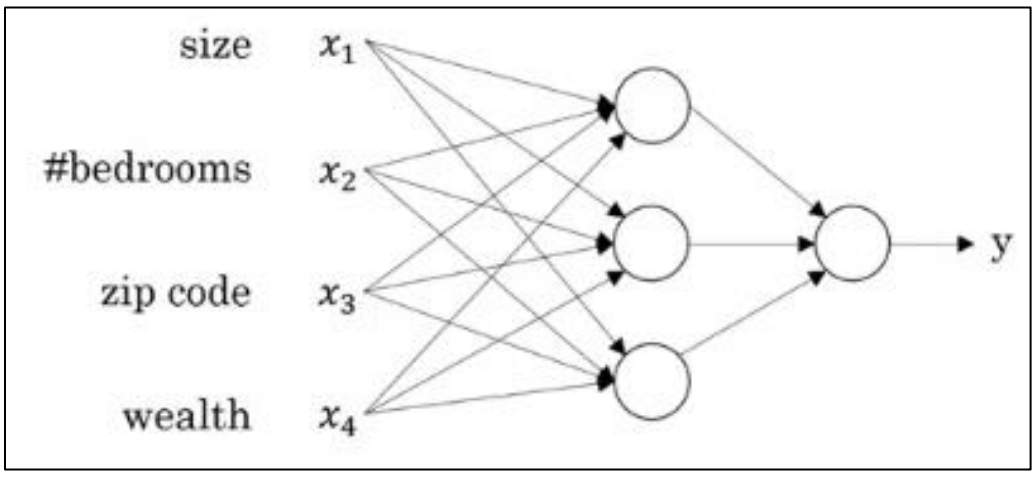


So far, we have seen a neural network with a single neuron, i.e., we only had one feature (size of the house) to predict the house price. But in reality, we'll have to consider multiple features

like number of bedrooms, postal code, etc.? House price can also depend on the family size, neighbourhood location or school quality. How can we define a neural network in such cases?



It gets a bit complicated here. Refer to the above image as you read – we pass 4 features as input to the neural network as x, it automatically identifies some hidden features from the input, and finally generates the output y. This is how a neural network with 4 inputs and an output with single hidden layer will look like:

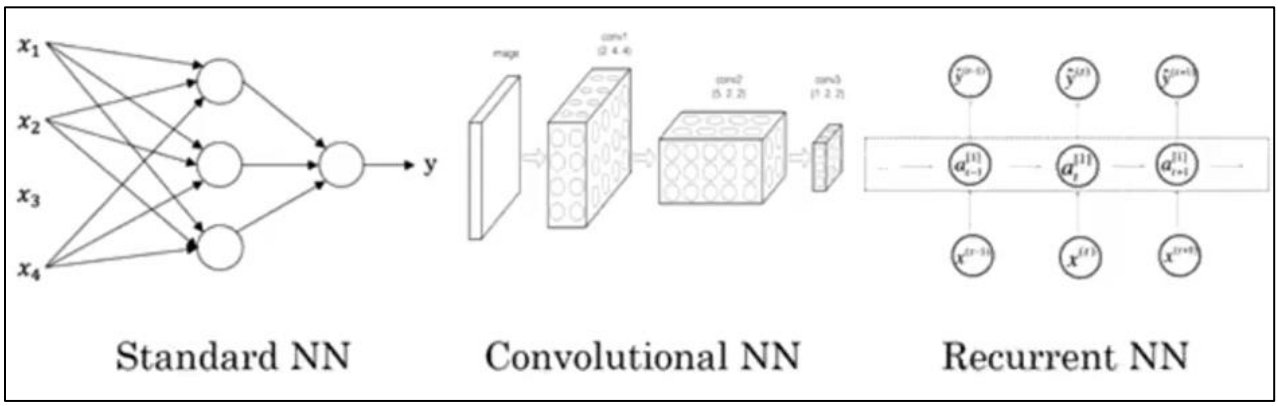


Now that we have an intuition of what neural networks are, let's see how we can use them for supervised learning problems.

Supervised Learning with Neural Networks

Supervised learning refers to a task where we need to find a function that can map input to corresponding outputs (given a set of input-output pairs). We have a defined output for each given input and we train the model on these examples. Below is a pretty handy table that looks at the different applications of supervised learning and the different types of neural networks that can be used to solve those problems:

Below is a visual representation of the most common Neural Network types:



In this article, we will be focusing on the standard neural networks. As you might be aware, supervised learning can be used on both structured and unstructured data.

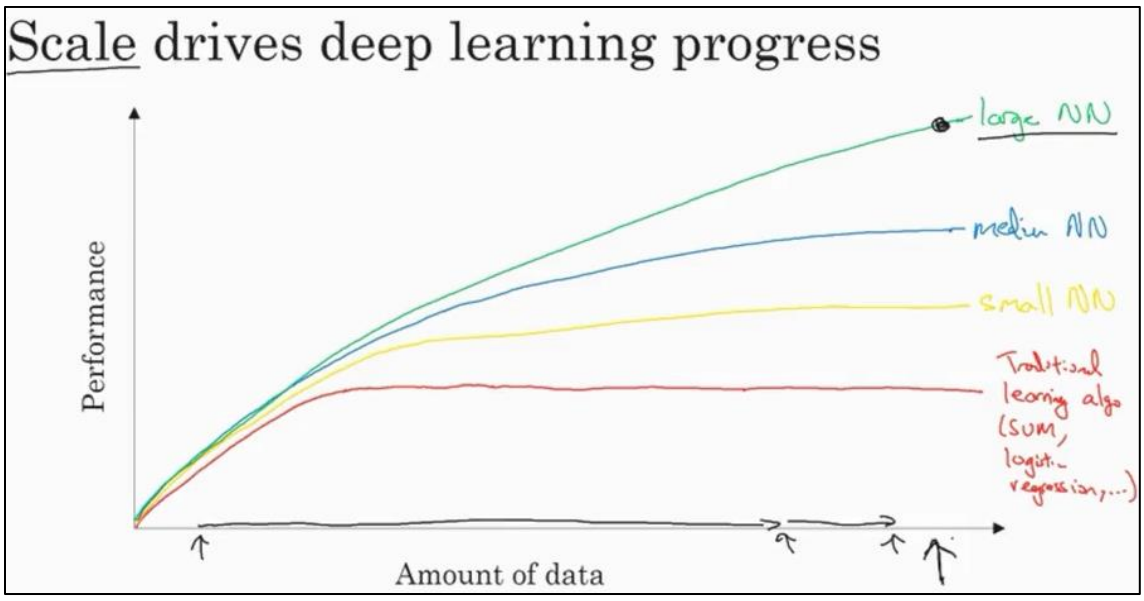
In our house price prediction example, the given data tells us the size and the number of bedrooms. This is structured data, meaning that each feature, such as the size of the house, the number of bedrooms, etc. has a very well defined meaning.

In contrast, unstructured data refers to things like audio, raw audio, or images where you might want to recognize what's in the image or text (like object detection). Here, the features might be the pixel values in an image, or the individual words in a piece of text. It's not really clear what each pixel of the image represents and therefore this fall under the unstructured data umbrella.

Simple machine learning algorithms work well with structured data. But when it comes to unstructured data, their performance tends to take quite a dip. This is where neural networks have proven to be so effective and useful. They perform exceptionally well on unstructured data. Most of the ground-breaking research these days has neural networks at it's core.

Why is Deep Learning Taking off?

To understand this, take a look at the below graph:

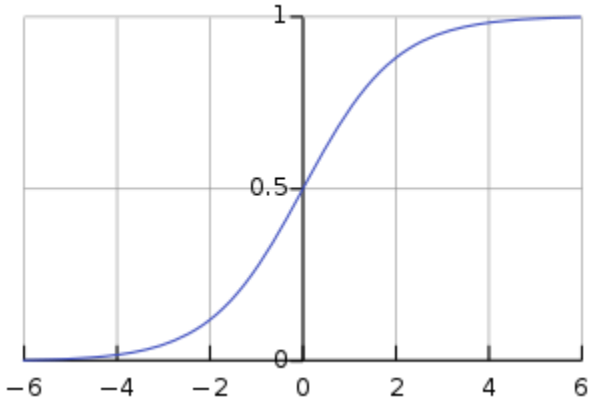


As the amount of data increases, the performance of traditional learning algorithms, like SVM and logistic regression, does not improve by a whole lot. In fact, it tends to plateau after a certain point. In the case of neural networks, the performance of the model increases with an increase in the data you feed to the model.

There are basically three scales that drive a typical deep learning process:

1.Data
2.Computation Time
3.Algorithms

To improve the computation time of the model, activation function plays an important role. If we use a sigmoid activation function, this is what we end up with:



The slope, or the gradient of this function, at the extreme ends is close to zero. Therefore, the parameters are updated very slowly, resulting in very slow learning. Hence, switching from a sigmoid activation function to ReLU (Rectified Linear Unit) is one of the biggest breakthroughs we have seen in neural networks. ReLU updates the parameters much faster as the slope is 1 when x>0. This is the primary reason for faster computation of the models.

2.2 NEURAL NETWORK BASICS:

This module is further divided into two parts:

Part I: Logistic Regression as a Neural Network:

Binary Classification

In a binary classification problem, we have an input x, say an image, and we have to classify it as having a cat or not. If it is a cat, we will assign it a 1, else 0. So here, we have only two outputs – either the image contains a cat or it does not. This is an example of a binary classification problem.

We can of course use the most popular classification technique, logistic regression, in this case.

Logistic Regression

We have an input X (image) and we want to know the probability that the image belongs to class 1 (i.e. a cat). For a given X vector, the output will be:

y = w(transpose)X + b

Here w and b are the parameters. Since our output y is probability, it should range between 0 and 1. But in the above equation, it can take any real value, which doesn't make sense for getting the probability. So logistic regression also uses a sigmoid function to output probabilities:

$$\hat{y} = \sigma(w^T x + b)$$

For any value as input, it will only return values in the 0 to 1 range. The formula for a sigmoid function is:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

So, if z is very large, exp(-z) will be close to 0, and therefore the output of the sigmoid will be 1. Similarly, if z is very small, exp(-z) will be infinity and hence the output of the sigmoid will be 0.

Logistic Regression Cost Function

To train the parameters w and b of logistic regression, we need a cost function. We want to find parameters w and b such that at least on the training set, the outputs you have (y-hat) are close to the actual values (y).

We can use a loss function defined below:

$$\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y}-y)^2$$

The problem with this function is that the optimization problem becomes non-convex, resulting in multiple local optima. Hence, gradient descent will not work well with this loss function. So, for logistic regression, we define a different loss function that plays a similar role as that of the above loss function and also solves the optimization problem by giving a convex function:

$$\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1-y) \log(1-\hat{y}))$$

Loss function is defined for a single training example which tells us how well we are doing on that particular example. On the other hand, a cost function is for the entire training set. Cost function for logistic regression is:

$$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

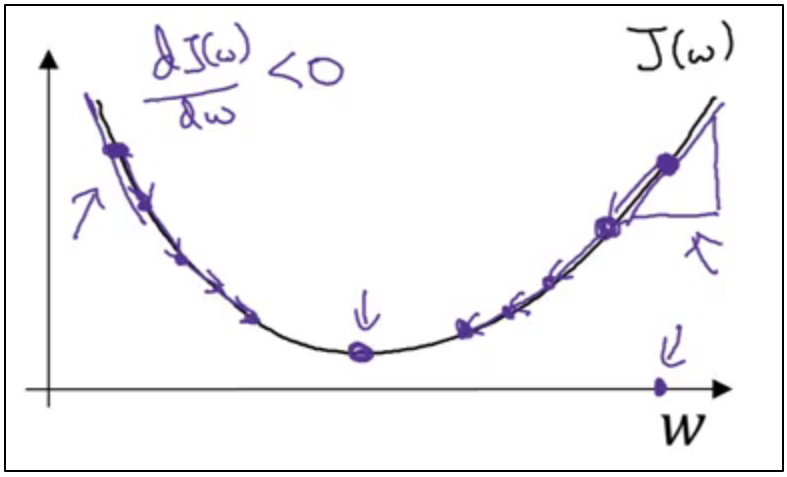
We want our cost function to be as small as possible. For that, we want our parameters w and b to be optimized.

Gradient Descent

This is a technique that helps to learn the parameters w and b in such a way that the cost function is minimized. The cost function for logistic regression is convex in nature (i.e. only one global minima) and that is the reason for choosing this function instead of the squared error (can have multiple local minima).

Let's look at the steps for gradient descent:

1.Initialize w and b (usually initialized to 0 for logistic regression)
2.Take a step in the steepest downhill direction
3.Repeat step 2 until global optimum is achieved



`

The updated equation for gradient descent becomes:

$$w := w - \alpha \frac{dJ(w)}{dw}$$

Here, α is the learning rate that controls how big a step we should take after each iteration. If we are on the right side of the graph shown above, the slope will be positive. Using the updated equation, we will move to the left (i.e. downward direction) until the global minima is reached. Whereas if we are on the left side, the slope will be negative and hence we will take a step towards the right (downward direction) until the global minima is reached. Pretty intuitive, right?

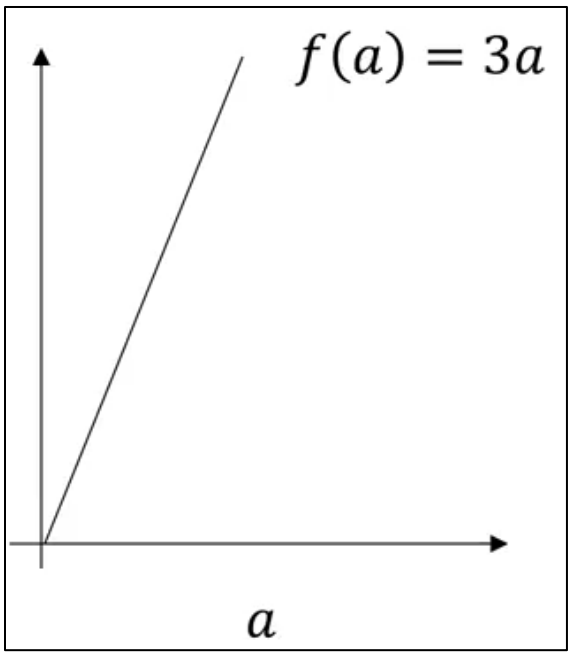
The updated equations for the parameters of logistic regression are

$$w := w - \alpha \frac{\partial J(w,b)}{\partial w}$$

$$b := b - \alpha \frac{\partial J(w,b)}{\partial b}$$

Derivatives

Consider a function, f(a) = 3a, as shown below:



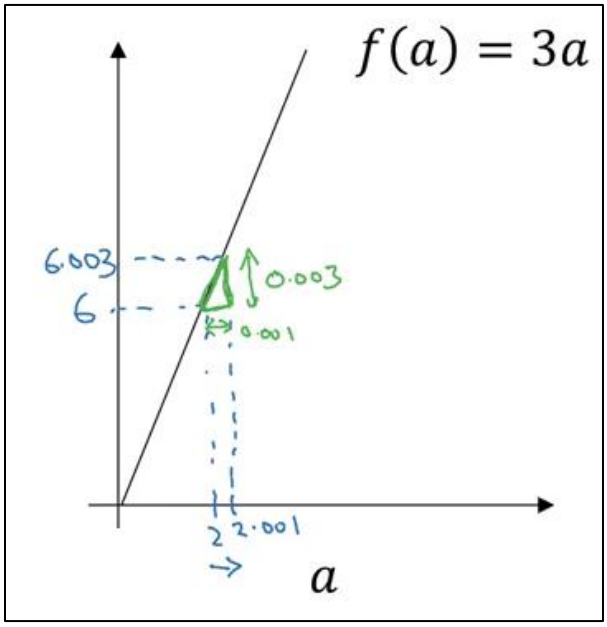
The derivative of this function at any point will give the slope at that point. So,

f(a=2) = 3*2 = 6

f(a=2.001) = 3*2.001 = 6.003

Slope/derivative of the function at a = 2 is:

Slope = height/width



Slope = 0.003 / 0.001 = 3

This is how we calculate the derivative/slope of a function. Let's look at a few more examples of derivatives.

Part II – Python and Vectorization:

Up to this point, we have seen how to use gradient descent for updating the parameters for logistic regression. In the above example, we saw that if we have 'm' training examples, we have to run the loop 'm' number of times to get the output, which makes the computation very slow.

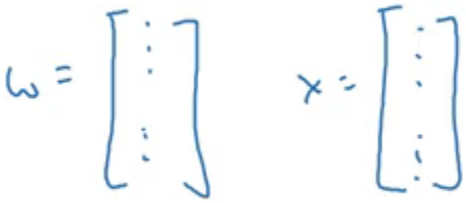
Instead of these for loops, we can use vectorization which is an effective and time efficient approach.

Vectorization-

Vectorization is basically a way of getting rid of for loops in our code. It performs all the operations together for 'm' training examples instead of computing them individually. Let's look at non-vectorized and vectorized representation of logistic regression:

Non-vectorized form:

```
z = 0
for i in range(nx):
    z += w[i] * x[i]
z +=b
```



Now, let's look at the vectorized form. We can represent the w and x in a vector form:

Now we can calculate Z for all the training examples using:

Z = np.dot(W,X)+b (numpy is imported as np)

The dot function of NumPy library uses vectorization by default. This is how we can vectorize the multiplications. Let's now see how we can vectorize an entire logistic regression algorithm.

Vectorizing Logistic Regression-

Keeping with the 'm' training examples, the first step will be to calculate Z for all of these examples:

```
Z = np.dot(W.T, X) + b
```

Here, X contains the features for all the training examples while W is the coefficient matrix for these examples. The next step is to calculate the output(A) which is the sigmoid of Z:

```
A = 1 / 1 + np.exp(-Z)
```

Now, calculate the loss and then use backpropagation to minimize the loss:

```
dz = A – Y
```

Finally, we will calculate the derivative of the parameters and update them:

```
dw = np.dot(X, dz.T) / m
db = dz.sum() / m
W = W – αdw
b = b – αdb
```

Broadcasting in Python-

Broadcasting makes certain parts of the code much more efficient. But don't just take my word for it! Let's look at some examples:

-obj.sum(axis = 0) sums the columns while obj.sum(axis = 1) sums the rows
-obj.reshape(1,4) changes the shape of the matrix by broadcasting the values

$$\begin{bmatrix}1\\2\\3\\4\end{bmatrix} + \begin{bmatrix}100\\100\\100\\100\end{bmatrix} 100 = \begin{bmatrix}101\\102\\103\\104\end{bmatrix}$$

If we add 100 to a (4×1) matrix, it will copy 100 to a (4×1) matrix. Similarly, in the example below, (1×3) matrix will be copied to form a (2×3) matrix:

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + \begin{bmatrix}100 & 200 & 300\\100 & 200 & 300\end{bmatrix} = \begin{bmatrix}101 & 202 & 303\\104 & 205 & 306\end{bmatrix}$$

(m, n) (2,3) (1, n) ⇝ (m, n) (2,3)

The general principle will be:

(m, n) matrix

$+$ $-$ $*$ $/$

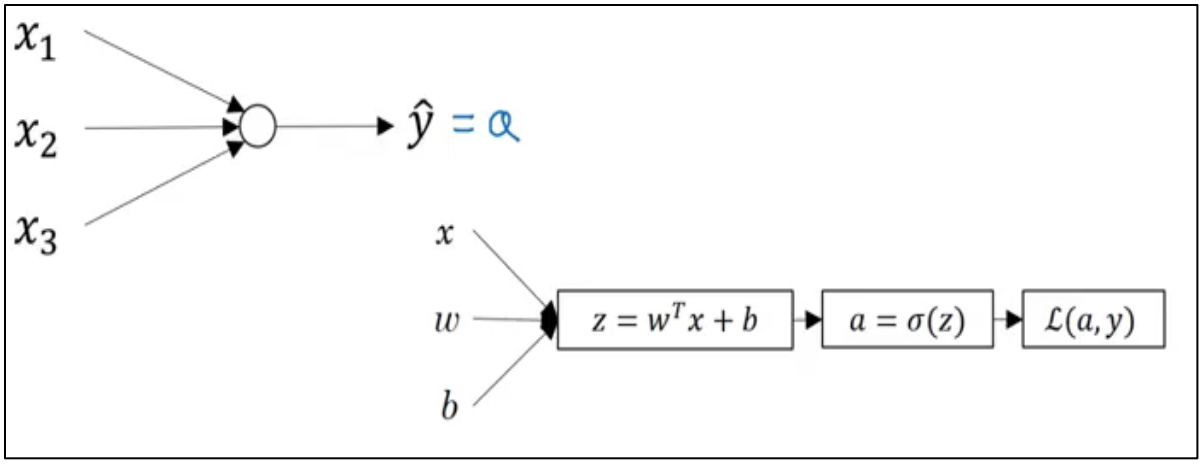
(1, n) ⇝ (m, n)

(m, 1) ⇝ (m, n)

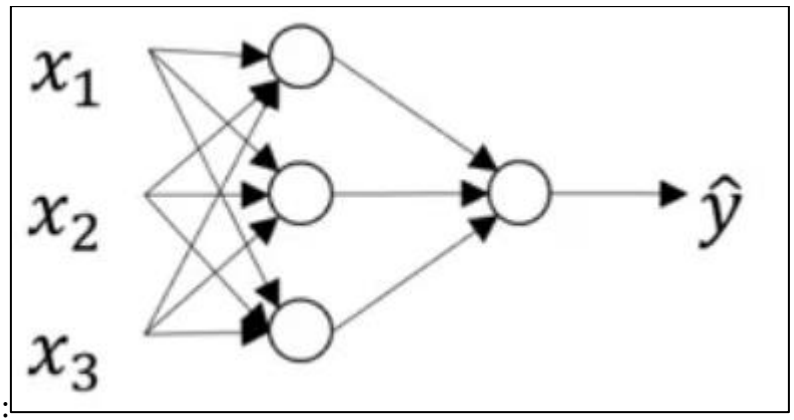
2.3 SHALLOW NEURAL NETWORKS:

Neural Networks Overview

In logistic regression, to calculate the output (y = a), we used the below computation graph:



In case of a neural network with a single hidden layer, the structure will look like



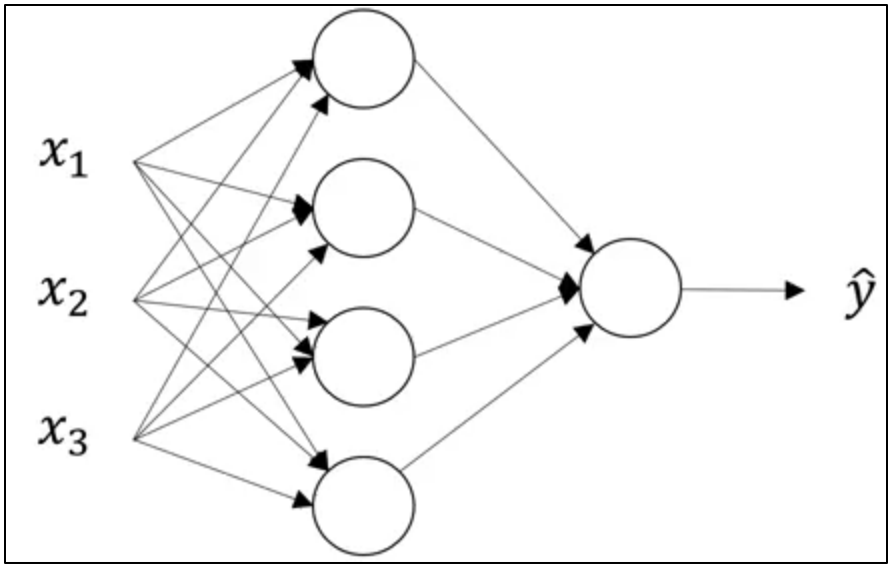
:

And the computation graph to calculate the output will be:

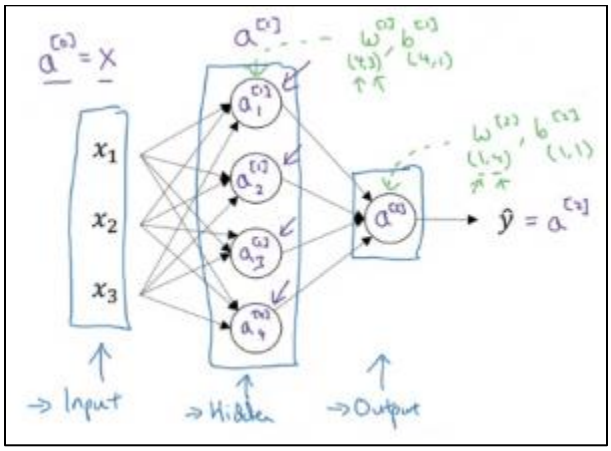
**X1 **
X2 => z1 = XW1 + B1 => a1 = Sigmoid(z1) =>
z2 = a1W2 + B2 => a2 = Sigmoid(z
=> l(a2,Y)
X3 /

Neural Network Representation

Consider the following representation of a neural network:



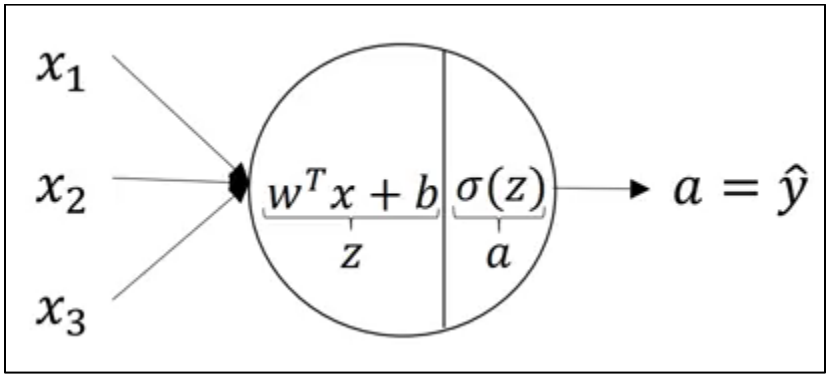
Can you identify the number of layers in the above neural network? Remember that while counting the number of layers in a NN, we do not count the input layer. So, there are 2 layers in the NN shown above, i.e., one hidden layer and one output layer. The first layer is referred as a[0], second layer as a[1], and the final layer as a[2]. Here 'a' stands for activations, which are the values that different layers of a neural network passes on to the next layer. The corresponding parameters are w[1], b[1] and w[1], b[2]:



This is how a neural network is represented. Next we will look at how to compute the output from a neural network.

Computing a Neural Network's Output

Let's look in detail at how each neuron of a neural network works. Each neuron takes an input, performs some operation on them (calculates z = w[T] + b), and then applies the sigmoid function:



This step is performed by each neuron. The equations for the first hidden layer with four neurons will be:

$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]},\ a_1^{[1]} = \sigma(z_1^{[1]})$$
$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]},\ a_2^{[1]} = \sigma(z_2^{[1]})$$
$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]},\ a_3^{[1]} = \sigma(z_3^{[1]})$$
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]},\ a_4^{[1]} = \sigma(z_4^{[1]})$$

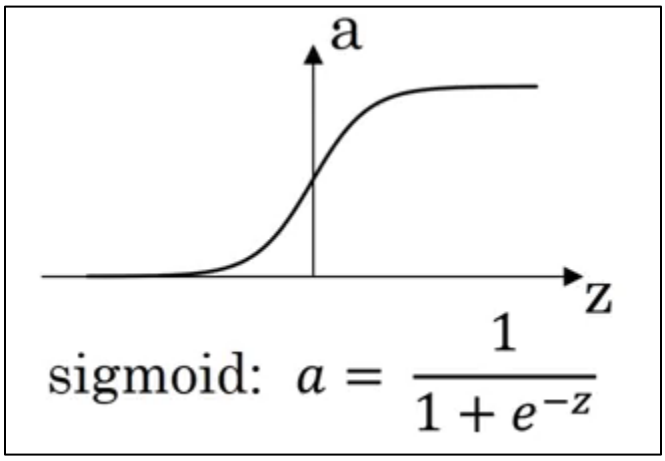
So, for given input X, the outputs for each neuron will be:

To compute these outputs, we need to run a for loop which will calculate these values individually for each neuron. But recall that using a for loop will make the computations very slow, and hence we should optimize the code to get rid of this for loop and run it faster.

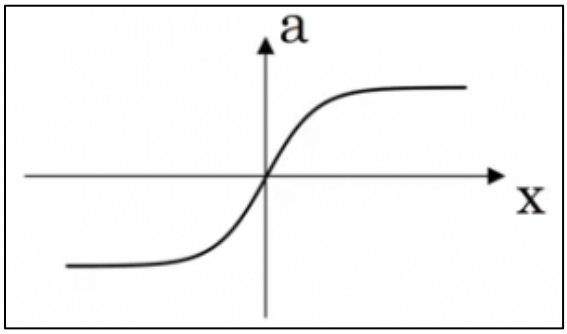
Activation Function

While calculating the output, an activation function is applied. The choice of an activation

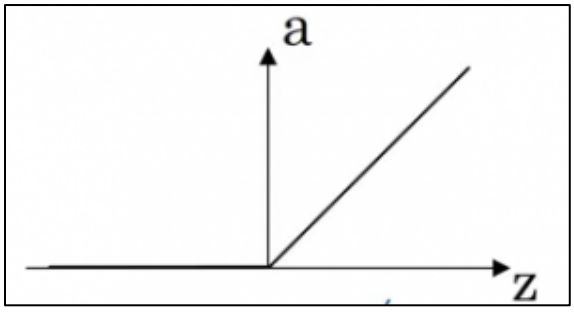
function highly affects the performance of the model. So far, we have used the sigmoid activation function:



However, this might not the best option in some cases. Why? Because at the extreme ends of the graph, the derivative will be close to zero and hence the gradient descent will update the parameters very slowly.
There are other functions which can replace this activation function:
tanh:



ReLU (already covered earlier):



Activation Function	Pros	Cons
Sigmoid	Used in the output layer for binary classification	Output ranges from 0 to 1
tanh	Better than sigmoid	Updates parameters slowly when points are at extreme end
sReLU	Updates parameters faster as slope is 1 when x>0	Zero slope when x<0

We can choose different activation functions depending on the prob

Why do we need non-linear activation functions?

If we use linear activation functions on the output of the layers, it will compute the output as a linear function of input features. We first calculate the Z value as:

$$Z = WX + b$$

In case of linear activation functions, the output will be equal to Z (instead of calculating any non-linear activation):

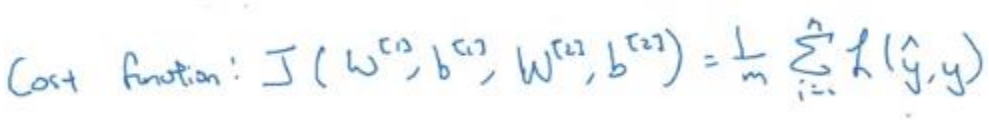
$$A = Z$$

Using linear activation is essentially pointless. The composition of two linear functions is itself a linear function, and unless we use some non-linear activations, we are not computing more interesting functions. That's why most experts stick to using non-linear activation functions. There is only one scenario where we tend to use a linear activation function. Suppose we want to predict the price of a house (which can be any positive real number). If we use a sigmoid or tanh function, the output will range from (0,1) and (-1,1) respectively. But the price will be more than 1 as well. In this case, we will use a linear activation function at the output layer.

Once we have the outputs, what's the next step? We want to perform backpropagation in order to update the parameters using gradient descent.

Gradient Descent for Neural Networks

The parameters which we have to update in a two-layer neural network are: w[1], b[1], w[2] and b[2], and the cost function which we will be minimizing is:



The gradient descent steps can be summarized as:

Repeat:

Compute predictions (y'(i), i = 1,...m)

Get derivatives: dW[1], db[1], dW[2], db[2]

Update: W[1] = W[1] - α * dW[1]

b[1] = b[1] - α * db[1]

W[2] = W[2] - α * dW[2]

b[2] = b[2] - α * db[2]

Let's quickly look at the forward and backpropagation steps for a two-layer neural networks.

Forward propagation:

```
Z[1] = W[1]*A[0] + b[1]    # A[0] is X
A[1] = g[1](Z[1])
Z[2] = W[2]*A[1] + b[2]
A[2] = g[2](Z[2])
```

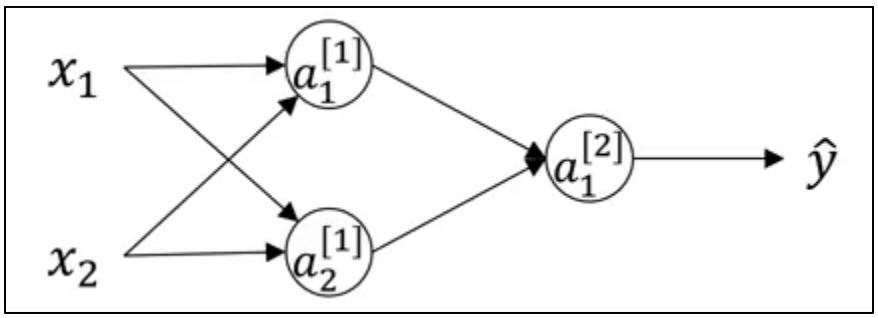
Backpropagation:

```
dZ[2] = A[2] - Y
dW[2] = (dZ[2] * A[1].T) / m
db[2] = Sum(dZ[2]) / m
dZ[1] = (W[2].T * dZ[2]) * g'[1](Z[1])  # element wise product (*)
dW[1] = (dZ[1] * A[0].T) / m   # A[0] = X
db[1] = Sum(dZ[1]) / m
```

These are the complete steps a neural network performs to generate outputs. Note that we have to initialize the weights (W) in the beginning which are then updated in the backpropagation step. So let's look at how these weights should be initialized.

Random Initialization

We have previously seen that the weights are initialized to 0 in case of a logistic regression algorithm. But should we initialize the weights of a neural network to 0? It's a pertinent question. Let's consider the example shown below:



If the weights are initialized to 0, the W matrix will be:

$$w^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Using these weights:

$$a_1^{[1]} = a_2^{[1]}$$

And finally at the backpropagation step:

$$dz_1^{[1]} = dz_2^{[1]}$$

No matter how many units we use in a layer, we are always getting the same output which is similar to that of using a single unit. So, instead of initializing the weights to 0, we randomly initialize them using the following code:

w[1] = np.random.randn((2,2)) * 0.01

b[1] = np.zero((2,1))

We multiply the weights with 0.01 to initialize small weights. If we initialize large weights, the activation will be large, resulting in zero slope (in case of sigmoid and tanh activation function). Hence, learning will be slow. So we generally initialize small weights randomly.

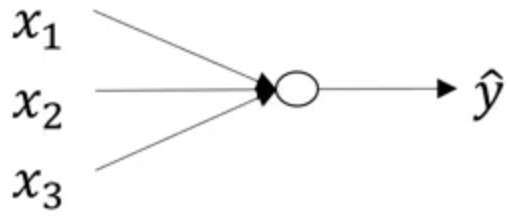
2.4 DEEP NEURAL NETWORKS:

It's finally time to learn about deep neural networks! These have become today's buzzword in the industry and the research field. No matter which research paper I pick up these days, there is inevitably a mention of how a deep neural network was used to power the thought process behind the study.

Deep L-Layer Neural Network

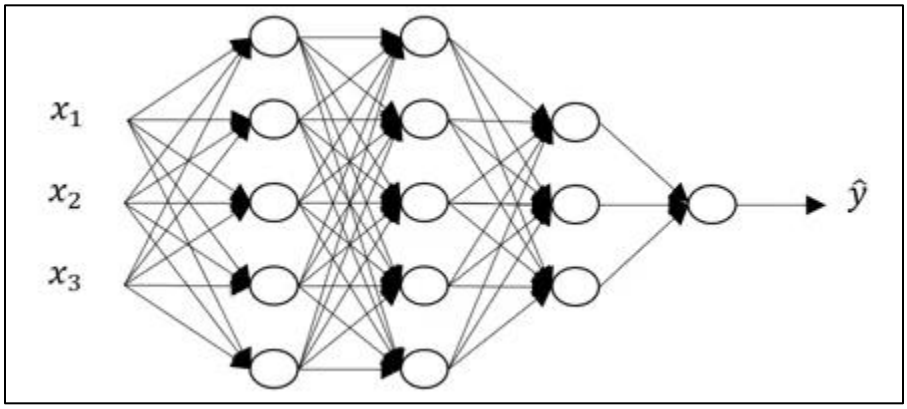
In this section, we will look at how the concepts of forward and backpropogation can be applied to deep neural networks. But you might be wondering at this point what in the world deep neural networks actually are?

Shallow vs depth is a matter of degree. A logistic regression is a very shallow model as it has only one layer (remember we don't count the input as a layer):



logistic regression

A deeper neural network has more number of hidden layers:



Let's look at some of the notations related to deep neural networks:

L is the number of layers in the neural network

n[l] is the number of units in layer l

a[l] is the activations in layer l

w[l] is the weights for z[l]

These are some of the notations which we will be using in the upcoming sections. Keep them in mind as we proceed, or just quickly hop back here in case you miss something.

Forward Propagation in a Deep Neural Network

For a single training example, the forward propagation steps can be written as:

z[l] = W[l]a[l-1] + b[l] a[l] = g[l](z[l])

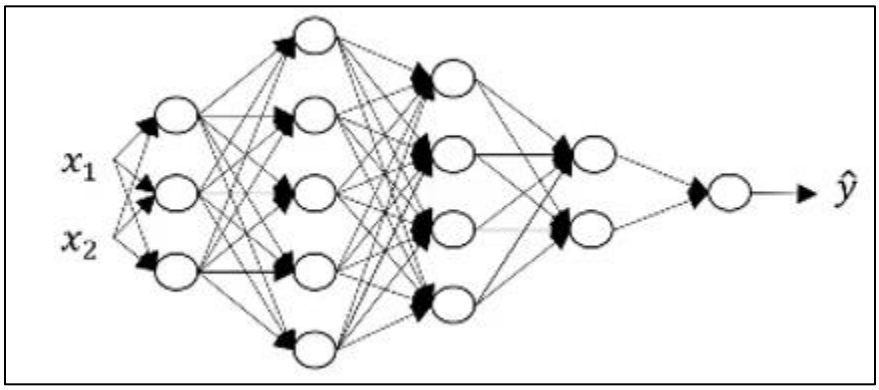
We can vectorize these steps for 'm' training examples as shown below:

Z[l] = W[l]A[l-1] + B[l] A[l] = g[l](Z[l])

These outputs from one layer act as an input for the next layer. We can't compute the forward propagation for all the layers of a neural network without a for loop, so its fine to have a for loop here. Before moving further, let's look at the dimensions of various matrices that will help us understand these steps in a better way.

Getting your matrix dimensions right

Analyzing the dimensions of a matrix is one of the best debugging tools to check how correct our code is. We will discuss what should be the correct dimension for each matrix in this section. Consider the following example:



Can you figure out the number of layers (L) in this neural network? You are correct if you guessed 5. There are 4 hidden layers and 1 output layer. The units in each layer are:

n[0] = 2, n[1] = 3, n[2] = 5, n[3] = 4, n[4] = 2, and n[5] = 1

The generalized form of dimensions of W, b and their derivatives is:

W[l] = (n[l], n[l-1])

b[l] = (n[l], 1)

dW[l] = (n[l], n[l-1])

db[l] = (n[l],1)

Dimension of Z[l], A[l], dZ[l], dA[l] = (n[l],m)

where 'm' is the number of training examples. These are some of the generalized matrix dimensions which will help you to run your code smoothly.

Forward and Backward Propagation

The input in a forward propagation step is a[l-1] and the outputs are a[l] and cache z[l], which is a function of w[l] and b[l]. So, the vectorized form to calculate Z[l] and A[l] is:

Z[l] = W[l] * A[l-1] + b[l]

A[l] = g[l](Z[l])

We will calculate Z and A for each layer of the network. After calculating the activations, the next step is backward propagation, where we update the weights using the derivatives. The input for backward propagation is da[l] and the outputs are da[l-1], dW[l] and db[l]. Let's look at the vectorized equations for backward propagation:

dZ[l] = dA[l] * g'[l](Z[l])
dW[l] = 1/m * (dZ[l] * A[l-1].T)
db[l] = 1/m * np.sum(dZ[l], axis = 1, keepdims = True)

dA[l-1] = w[l].T * dZ[l]

This is how we implement deep neural networks.

Deep Neural Networks perform surprisingly well (maybe not so surprising if you've used them before!). Running only a few lines of code gives us satisfactory results. This is because we are feeding a large amount of data to the network and it is learning from that data using the hidden layers.

Choosing the right hyperparameters helps us to make our model more efficient. We will cover the details of hyperparameter tuning in the next article of this series.

Parameters vs Hyperparameters

This is an oft-asked question by deep learning newcomers. The major difference between parameters and hyperparameters is that parameters are learned by the model during the training time, while hyperparameters can be changed before training the model.

Parameters of a deep neural network are W and b, which the model updates during the backpropagation step. On the other hand, there are a lot of hyperparameters for a deep NN, including:

Learning rate – α

- Number of iterations
- Number of hidden layers
- Units in each hidden layer
- Choice of activation function

CHAPTER 2. APPLICATIONS:

1. Self-driving cars

Companies building these types of driver-assistance services, as well as full-blown self-driving cars like Google's, need to teach a computer how to take over key parts (or all) of driving using digital sensor systems instead of a human's senses. To do that companies generally start out by training algorithms using a large amount of data.

You can think of it how a child learns through constant experiences and replication. These new services could provide unexpected business models for companies.

2. Deep Learning in Healthcare

Breast or Skin-Cancer diagnostics? Mobile and Monitoring Apps? or prediction and personalised medicine on the basis of Biobank-data? AI is completely reshaping life sciences, medicine, and healthcare as an industry. Innovations in AI are advancing the future of precision medicine and population health management in unbelievable ways. Computer-aided detection, quantitative imaging, decision support tools and computer-aided diagnosis will play a big role in years to come.

3. Voice Search & Voice-Activated Assistants

One of the most popular usage areas of deep learning is voice search & voice-activated intelligent assistants. With the big tech giants have already made significant investments in this area, voice-activated assistants can be found on nearly every smartphone. Apple's Siri is on the market since October 2011. Google Now, the voice-activated assistant for Android, was launched less than a year after Siri. The newest of the voice-activated intelligent assistants is Microsoft Cortana.

4. Automatically Adding Sounds to Silent Movies

In this task, the system must synthesize sounds to match a silent video. The system is trained using 1000 examples of video with sound of a drumstick striking different surfaces and creating different sounds. A deep learning model associates the video frames with a database of pre-

rerecorded sounds in order to select a sound to play that best matches what is happening in the scene.

The system was then evaluated using a Turing-test like a setup where humans had to determine which video had the real or the fake (synthesized) sounds.

This uses application of both convolutional neural networks and Long short-term memory **(**LSTM) recurrent neural networks (RNN).

5. Automatic Machine Translation

This is a task where given words, phrase or sentence in one language, automatically translate it into another language.

Automatic machine translation has been around for a long time, but deep learning is achieving top results in two specific areas:

- Automatic Translation of Text
- Automatic Translation of Images

Text translation can be performed without any pre-processing of the sequence, allowing the algorithm to learn the dependencies between words and their mapping to a new language.

6. Automatic Text Generation

This is an interesting task, where a corpus of text is learned and from this model new text is generated, word-by-word or character-by-character.

The model is capable of learning how to spell, punctuate, form sentences and even capture the style of the text in the corpus. Large recurrent neural networks are used to learn the relationship between items in the sequences of input strings and then generate text.

7. Automatic Handwriting Generation

This is a task where given a corpus of handwriting examples, generate new handwriting for a given word or phrase.

The handwriting is provided as a sequence of coordinates used by a pen when the handwriting samples were created. From this corpus, the relationship between the pen movement and the letters is learned and new examples can be generated ad hoc.

8. Image Recognition

Another popular area regarding deep learning is image recognition. It aims to recognize and identify people and objects in images as well as to understand the content and context. Image recognition is already being used in several sectors like gaming, social media, retail, tourism, etc.

This task requires the classification of objects within a photograph as one of a set of previously known objects. A more complex variation of this task called object detection involves specifically identifying one or more objects within the scene of the photograph and drawing a box around them.

9. Automatic Image Caption Generation

Automatic image captioning is the task where given an image the system must generate a caption that describes the contents of the image.

In 2014, there was an explosion of deep learning algorithms achieving very impressive results on this problem, leveraging the work from top models for object classification and object detection in photographs.

Once you can detect objects in photographs and generate labels for those objects, you can see that the next step is to turn those labels into a coherent sentence description.

Generally, the systems involve the use of very large convolutional neural networks for the object detection in the photographs and then a recurrent neural network (RNN) like an Long short-term memory **(**LSTM) to turn the labels into a coherent sentence.

10. Automatic Colorization

Image colorization is the problem of adding color to black and white photographs. Deep learning can be used to use the objects and their context within the photograph to color the image, much like a human operator might approach the problem. This capability leverage the high quality and very large convolutional neural networks trained for ImageNet and co-opted for the problem of image colorization. Generally, the approach involves the use of very large convolutional neural networks and supervised layers that recreate the image with the addition of color.

11. Advertising

Advertising is another key area that has been transformed by deep learning. It has been used by both publishers and advertisers to increase the relevancy of their ads and boost the return on investment of their advertising campaigns. For instance, deep learning makes it possible for ad networks and publishers to leverage their content in order to create **data-driven predictive advertising**, **real-time bidding (RTB) for their ads**, **precisely targeted display advertising** and more.

12. Predicting Earthquakes

Harvard scientists used Deep Learning to teach a computer to perform viscoelastic computations, these are the computations used in predictions of earthquakes. Until their paper, such computations were very computer intensive, but this application of Deep Learning improved calculation time by 50,000%. When it comes to earthquake calculation, timing is important and this improvement can be vital in saving a life.

13. Neural Networks for Brain Cancer Detection

A team of French researchers noted that spotting invasive brain cancer cells during surgery is difficult, in part because of the effects of lighting in operating rooms. They found that using neural networks in conjunction with Raman spectroscopy during operations allows them to detect the cancerous cells easier and reduce residual cancer post-operation. In fact, this piece is one of many over the last few weeks that match advanced image recognition and classification with various types of cancer and screening apparatus–more in the short list below.

14. Neural Networks in Finance

Futures markets have seen a phenomenal success since their inception both in developed and developing countries during the last four decades. This success is attributable to the tremendous leverage the futures provide to market participants. This study analyzes a trading strategy which benefits from this leverage by using the **Capital Asset Pricing Model (CAPM)** and cost-of-carry relationship. The team applies the technical trading rules developed from spot market prices, on futures market prices using a CAPM based hedge ratio. Historical daily prices of twenty stocks from each of the ten markets (five developed markets and five emerging markets) are used for the analysis.

15. Energy Market Price Forecasting

Researchers in Spain and Portugal have applied artificial neural networks to the energy grid in an effort to predict price and usage fluctuations. The daily and intraday markets for the region are organized in a daily session where next-day sale and electricity purchase transactions are carried out and in six intraday sessions that consider energy offer and demand, which may arise in the hours following the daily viability schedule fixed after the daily session. In short, being able to make adequate predictions based on the patterns of consumption and availability yields to far higher efficiency and cost savings.

CHAPTER 3 – Room Classifier

3.1 INTRODUCTION:

It is easy for the human brain to process images and analyse them. When the eye sees a certain image, the brain can easily segment it and recognize its different elements. The brain automatically goes through that process, which involves not only the analysis of this images, but also the comparison of their different characteristics with what it already knows in order to be able to recognize these elements. There is a field in computer science that tries to do the same thing for machines, which is Image Processing. Thing for machines, which is Image Processing. Image processing is the field that concerns analysing images so as to extract some useful information from them. This method takes images and converts them into a digital form readable by computers, it applies certain algorithms on them, and results in a better-quality image or with some of their characteristics that could be used in order to extract some important information from them. Image processing is applied in several areas, especially nowadays, and several software have been developed that use this concept. Now we have self-driven cars which can detect other cars and human beings to avoid accidents. Also, some social media applications, like Facebook, can do facial recognition thanks to this technique. Furthermore, some software uses it in order to recognise the characters in some images.

Similarly, the Room Classifier is the ability of computers to recognize clean and messy room. Scene recognition is one of the determining tasks of the computer vision. Our project focuses on this task. In this project our aim is to classify whether room is messy or clean.

Considering a hotel, there are many rooms and customers generally wants to find their rooms clean. This task involves finding messy rooms in the hotel. Our project, may help this issue and automatically find the rooms that needed to be cleaned.

Each room need different cleaning products. Another application of this project might be help advising cleaning products to customers with respect to which rooms are messy in their house. This might be very helpful within the context of smart homes. Also helpful for hostel rooms.

3.2 SYSTEM REQUIREMENTS

(i) HARDWARE REQUIREMENTS: -

The system must have the following hardware requirements:

1. Pentium IV Processors or high-speed processor

2. Internet connection.

3. RAM: 128 MB

(ii) SOFTWARE REQUIREMENTS: -

The system must have the following software requirements:

1. Anaconda (Jupyter Notebook)

2. Operating system: Windows XP, Windows 7, Windows 8, Windows 10.

3. Python 3 - Language

3.3 TECHNOLOGY AND CONCEPTS USED:

Machine Learning-

Learning algorithms are widely used in computer vision applications. Before considering image related tasks, we are going to have a brief look at basics of machine learning.

Machine learning has emerged as a useful tool for modelling problems that are otherwise di_cult to formulate exactly. Classical computer programs are explicitly programmed by hand to perform a task. With machine learning, some portion of the human contribution is replaced by a learning algorithm. As availability of computational capacity and data has increased, machine learning has become more and more practical over the years, to the point of being almost ubiquitous.

It can be used in two ways:

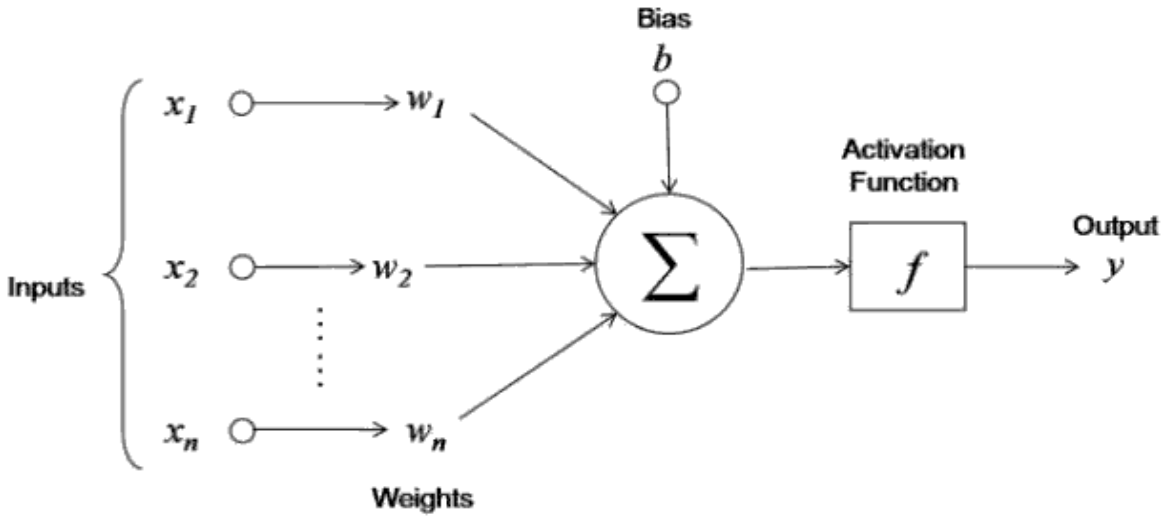
- Supervised Learning
- Unsupervised Learning

Neural networks-

Neural networks are a popular type of machine learning model. A special case of a neural network called the convolutional neural network (CNN) is the primary focus of this thesis.

Before discussing CNNs, we will discuss how regular neural networks work.

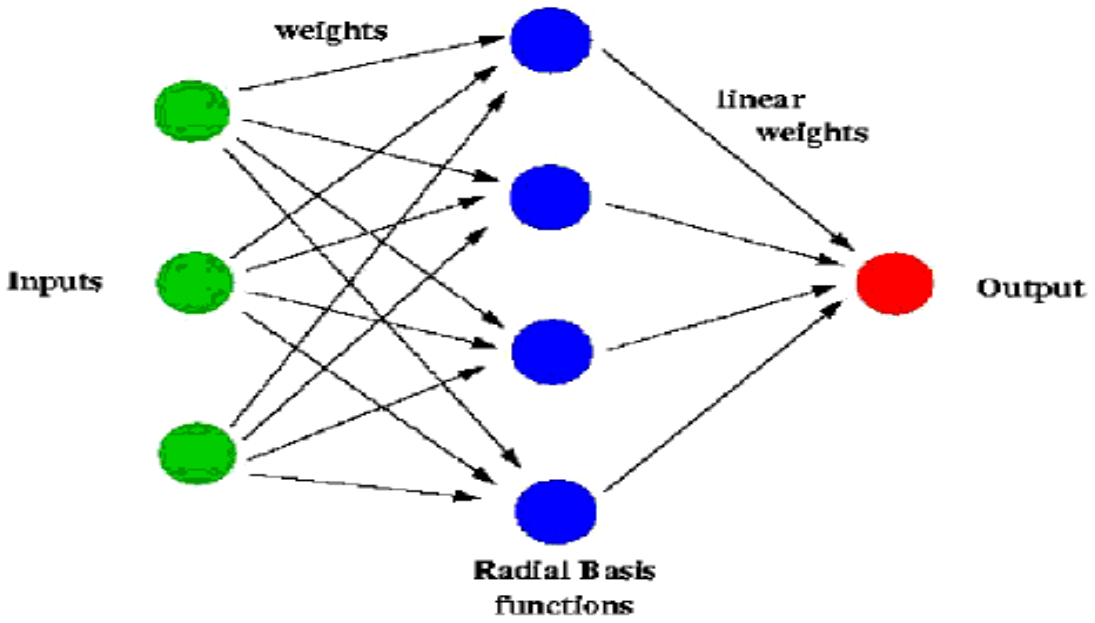
Neural networks were originally called arti_cial neural networks, because they were developed to mimic the neural function of the human brain.



The neuron is trained by carefully selecting the weights to produce a desired output for each input.

Multi-layer networks:

A neural network is a combination of arti_cial neurons. The neurons are typically grouped into layers.



A multi-layer network typically includes three types of layers: an input layer, one or more hidden layers and an output layer. The input layer usually merely passes data along without modifying it. Most of the computation happens in the hidden layers. The output layer converts the hidden layer activations to an output, such as a classification. A multilayer feed-forward network with at least one hidden layer can function as a universal approximator.

Import Libraries

The following libraries of machine learning were used in making of this project:

- **Numpy:**

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

i. A powerful N-dimensional array object.
ii. Sophisticated (broadcasting) functions.
iii. Tools for integrating C/C++ and Fortran code.
iv. Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multidimensional container of generic data.

Arbitrary data-types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

- **Pandas:**

Pandas is a high-level data manipulation tool developed by Wes McKinney. It is built on the Numpy package and its key data structure is called the DataFrame. DataFrames allow you to store and manipulate tabular data in rows of observations and columns of variables. There are several ways to create a DataFrame. In particular, it offers data structures and operations for manipulating numerical tables and time series. Pandas is mainly used for data analysis. Pandas allows importing data from various file formats such as comma-separated values, JSON, SQL, Microsoft Excel. Pandas allows various data manipulation operations such as merging, reshaping, selecting, as well as data cleaning, and data wrangling features.

- **Os:**

The OS module in python provides functions for interacting with the operating system. OS, comes under Python's standard utility modules. This module provides a portable way of using operating system dependent functionality. The *os* and *os.path* modules include many functions to interact with the file system.

- **matplotlib.pyplot as plt:**

matplotlib.pyplot is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.
In matplotlib.pyplot various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes (please note that "axes" here and in most places in the documentation refers to the axes part of a figure and not the strict mathematical term for more than one axis).

- **OpenCV:**

Open Source Computer Vision Library is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

3.4 Implementation of the Project:

Description of the dataset

I first built a small dataset consisting of ~200 images. The location is diverse, including bedrooms, living rooms, dining rooms, study rooms and kitchens, which might help to discourage the model from capturing unrelated features but instead focus more on the "messiness".

All the images are downloaded from Google and pre-processed. The dataset can be found on Kaggle. There are 192 images in the training set (96 per class); 20 images (10 per class) in the validation set; 10 images (5 per class) in the test set. The test set is left untouched during the training and tuning, and it is only used in the final step to assess the generalization of the model.

Design

In this project, I used the pretrained CNN model Xception as the body, and build a custom head to decode the feature in the end. Xception is an improvement from Inception-V3. It has 79% Top-1 and 94.5% Top-5 accuracy on ImageNet. The pretrained model takes 299x299x3 image input and outputs a 2048x1 feature vector after global average pooling. Then I added a fully connected layer of 10 units and an output layer with 1 unit for binary classification. L2 regularization is used in both layers to prevent overfitting. Retraining is done using Keras with Tensorflow as backend.

Pre-processing and augmentation

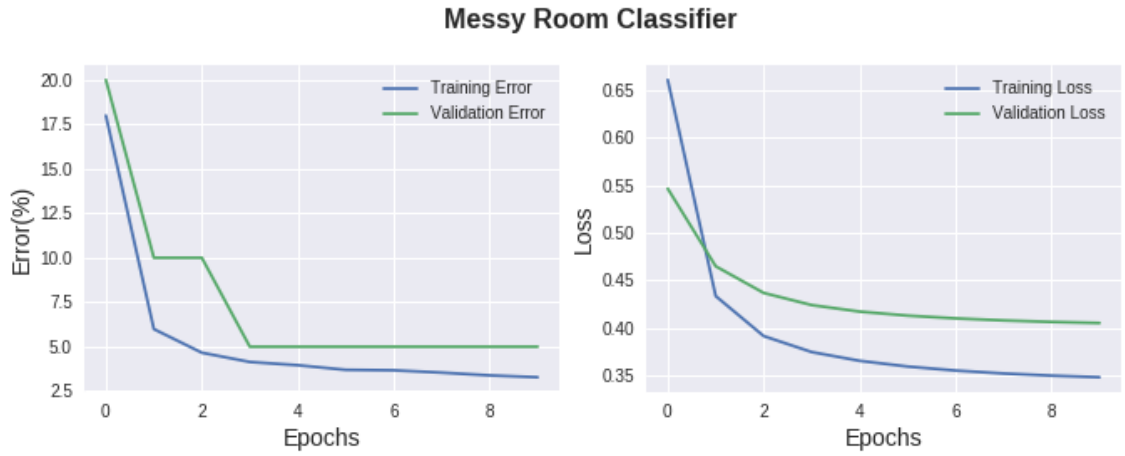
All the images are cropped to square and resized to 299x299 with OpenCV-Python as described in `preprocessing.py`. For convenience, the images are saved as numpy arrays in `room_dataset.py`. Images are then standardized to N(0, 1) on per channel basis. Since the dataset is very small, the training set is augmented, i.e. train images are randomly rotated, shifted, sheared, zoomed or flipped to some degree.
One trick I found particularly useful (for small dataset) is to do the augmentation beforehand and save the whole augmented data (e.g. for 20 epochs) with its corresponding labels. We can also use the pretrained model to extract feature vectors from these augmented data (`aug_and_feature_extract.py`).
In this way, we only need to compute for the two-layer head later on. In other words, we no longer need to do real time augmentation and calculate the whole model again and again (forward pass still time-consuming even though the Xception is non-trainable). This is useful when we are not sure about the hyper parameters and need to tune the model many times in the beginning. However, we might end up with limited variability because once we augment the data, we settle on it. For this specific project, it seems no issues with this approach so far.

Training and prediction

Following the above procedures, x_train becomes 3840x2048 array and y_train becomes 3840x1 vector after "pre-augmentation and feature extraction" (192 x 20 = 3840). That means we get 3840 observations to work with rather than just 192. However, every 192 rows in the array is actually just a slight variation from the original images. To avoid the possibility that two or more images with the same origin are selected in the same batch (e.g. batch size of 32), we do not shuffle the training set when fitting the model. That said, within each 192 samples, the order is randomized when we do the augmentation.

Adam is used for optimization with a default learning rate of 1e-3. Binary crossentropy is used as the loss function. I also used 0.01 as weight decay for L2 regularization. The 2-layer custom head is saved in the model directory.

The training result is shown below. The model fast converges and achieves 96.7%, 95% (19/20) accuracy on training and validation set respectively.



To evaluate the generalization capacity of the model, I used a test set that the model has never seen during the training and tuning. As shown below, the model rather accurately predicts the "probability" of being messy room.

Other approaches

As described in the Jupyter notebook, I also tried to use simple logistic regression or SVM to deal with this task (without data augmentation). Since 2048 dimension is way too many considering the size of the data, I used PCA to reduce the dimension to 18 which retains 95%

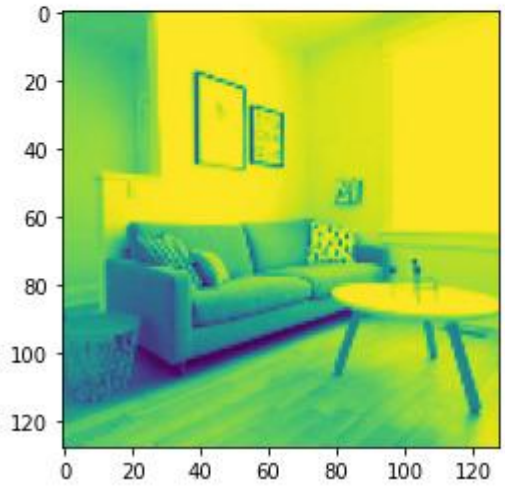
of variance. From that, I got 86% training accuracy and 80% validation accuracy by logistic regression. Linear SVM gives comparable results.

The PCA with top two principal components (retains 80% variance) shows that the data might not be linearly separable. Finally, SVM with polynomial or Gaussian kernel easily overfits the data.

3.5 Outcome of the Project

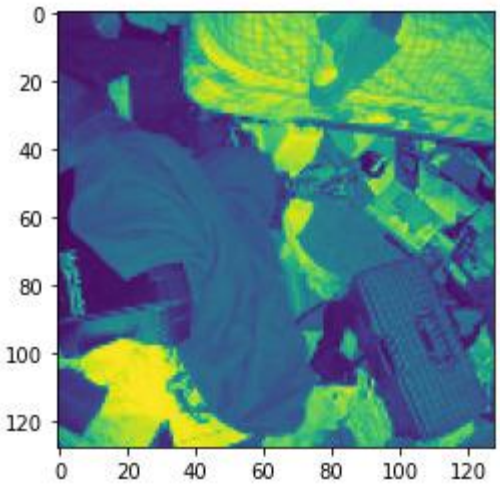
```
n = 4
plt.imshow(val_data[n])
if(val_result[0][n]==1):
    print("messy room")
else:
    print("clean room")
```

```
clean room
```



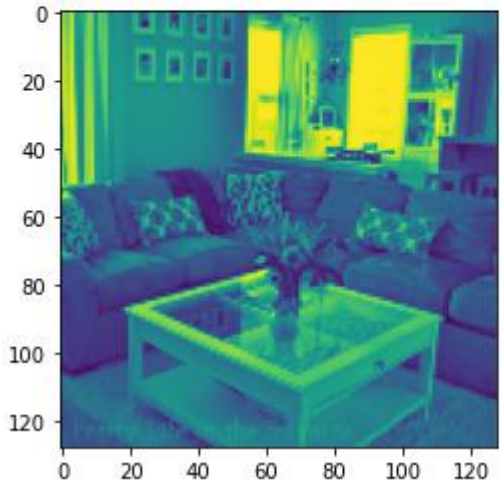
```
n = 3
plt.imshow(val_data[n])
if(val_result[0][n]==1):
    print("messy room")
else:
    print("clean room")
```

```
messy room
```




```
n = 0
plt.imshow(val_data[n])
if(val_result[0][n]==1):
    print("messy room")
else:
    print("clean room")
```

```
clean room
```



CHAPTER 4 – Conclusion

In this project, I followed the transfer learning methodology to apply the pretrained model (object classification) to solve a new problem (scene classification). I successfully trained a classifier that is able to distinguish messy rooms from clean rooms with reasonable accuracy. The idea might be useful for developing robot cleaner that can-do housework when it "sees" the room is messy. (at least in my dream)

One of the limitations is that the messy rooms in the dataset are rather extreme cases. One might never have such messiness ever. However, it is very hard to find realistic messy room images online, because people usually post something nice to look or something extremely awful that they want to complain about. Consequently, the model may perform poorly to distinguish slight messiness from slight cleanness, although the "probability" of messiness might provide some hints.

For similar reasons, some of the clean rooms in the dataset are also a bit unrealistic, although I tried to exclude those visually appealing rooms taken by photographers or designer because the light, colour, view point or other factors might mislead the model. However, due to the limitation of image sources, some clean rooms are indeed more spacious and pleasant than being clean per se.