Course : Machine Learning
Lecture On : ANN
Instructor : Shambhu Kumar

KEY TOPICS

- ❖ Perceptron
- ❖ Introduction to ANN
- ❖ Feedforward
- ❖ Back-Propagation
- ❖ CNN
- ❖ Open Q&A

Perceptrons

upGrad

- One of the earliest proposed (1960’s) simple device towards building a ANN. It uses Step function as Activation function.
- The perceptron takes a **weighted sum** of multiple inputs (along with a bias) as the cumulative input and applies a **step function** on the cumulative input, i.e. it returns 1 if the input is positive, else -1.

$$\text{Cumulative Input} = w^T.x + b = w_1x_1 + w_2x_2 + \ldots + w_kx_k + b$$

- Due to Step functions used, It works in binary mode – either fires or doesn’t fire (based on hard-cut threshold)
- Originally, A single Perceptron were designed to be a classifier.

If y(wT.x+b) > 0 : Class I else Class II

- However, a network of perceptron can be used to achieve much more complex task (universal function estimator) e.g. multi-class classification. The stacked perceptron architecture is also known as MLP.

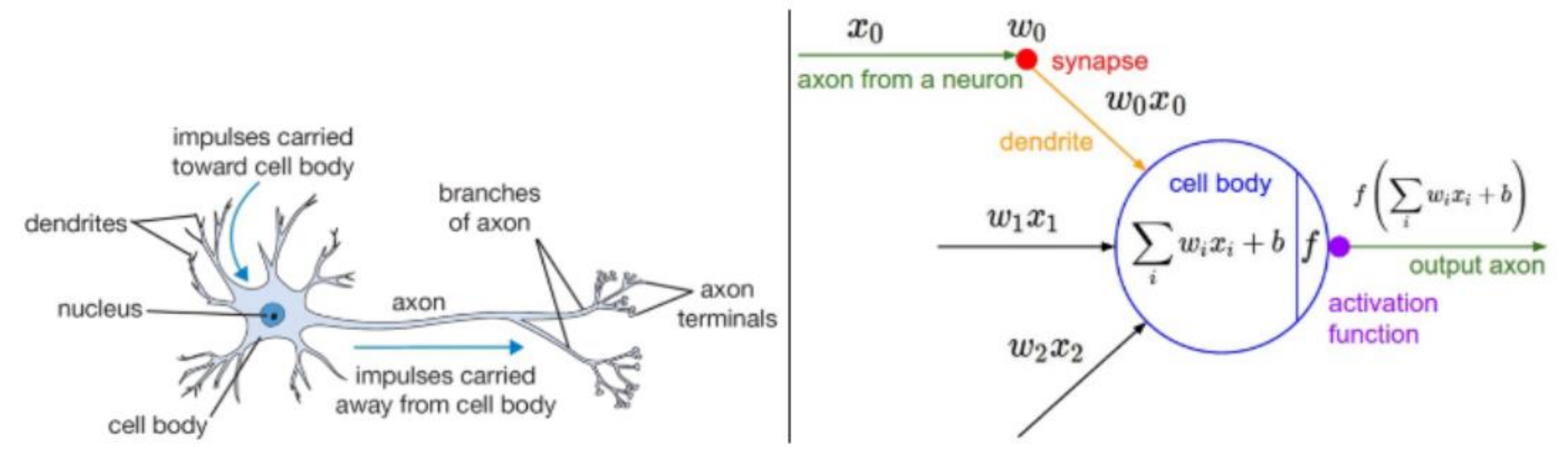
ANN

upGrad

- An ANN is a collection of many simple deices called Neurons. An artificial neuron is very similar to a perceptron, except that the activation function is not a step function.

- An ANN (or NN) simply try to mimic the working or functioning of human brains.

- The data in the network flows through each neuron by a connection. Every connection has a specific weight by which the flow of data is regulated.



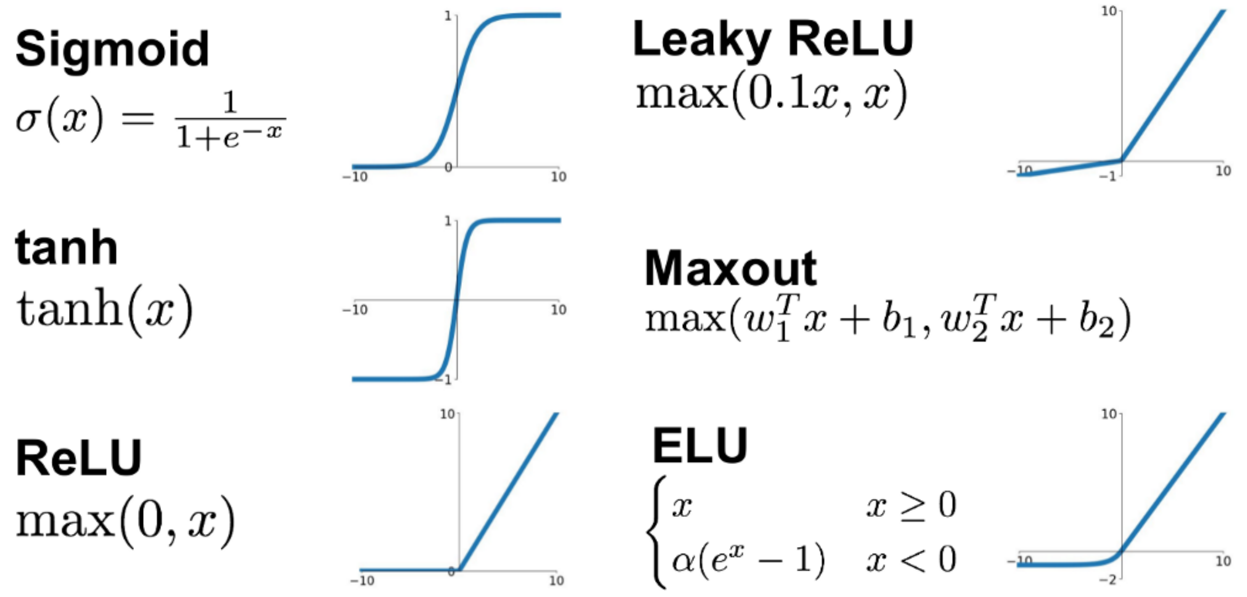
A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Why we use Activation functions

Our brain functions non-linearly for complex calculation, high parallelism, and for ability to handle imprecise and fuzzy information.

Activation *functions* are really important for a Artificial Neural Network to learn and make sense Non-linear complex functional mappings between the inputs and response variable.

AF *introduce non-linear properties to our Network. Their **main purpose is to convert a input signal of a node in a A-NN to an output signal.*** That output signal now is used as a input in the next layer in the stack.



https://ai.stackexchange.com/questions/5493/what-is-the-purpose-of-an-activation-function-in-neural-networks

Architecture/Assumption in ANN

upGrad

1. Neurons are arranged in layers and the layers are arranged sequentially.

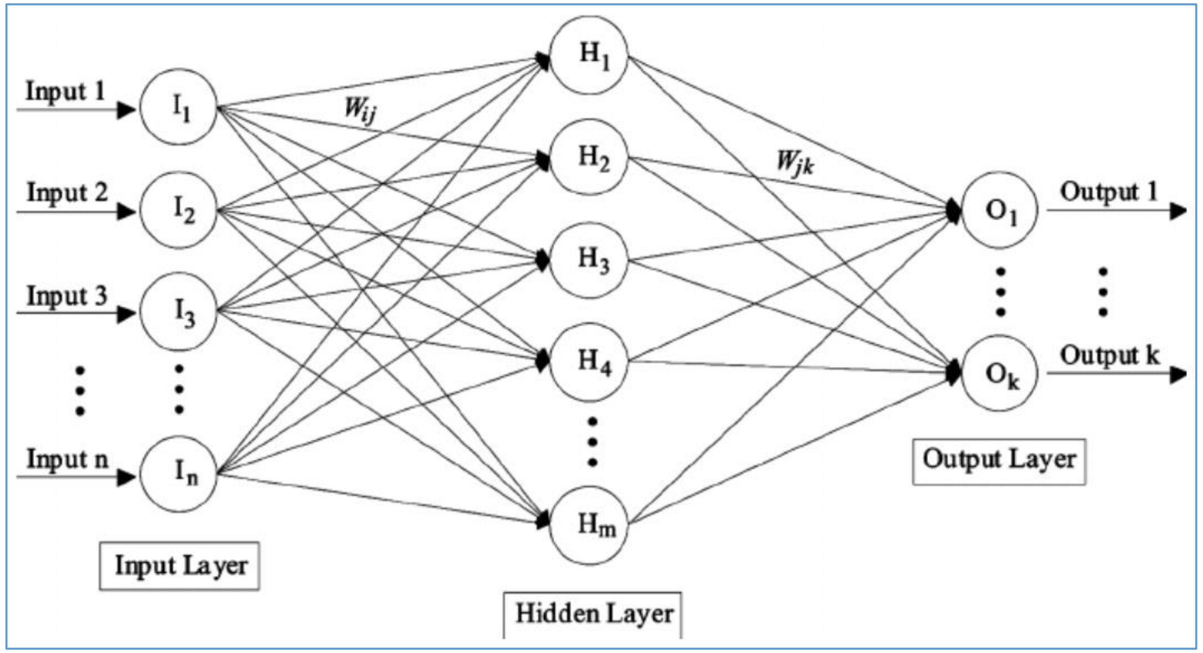
2. Neurons within the same layer do not interact with each other.

3. All the inputs enter the network through the input layer and all the outputs go out of the network through the output layer.

4. All neurons in layer L are connected to all neurons in layer L+1.

5. Every interconnection in the neural network has a weight and a bias associated with it.

6. All neurons in a particular layer use the same activation function.



Notes:

i. ANNs only take **numerical inputs.**
ii. ANNs – are trained on **weights and biases**
iii. Hyperparameters : No. of Layers, No. of neurons in Input/Hidden/Output layers, activation function

Parameter/Hyperparameters

upGrad

Q1. Assume a simple MLP model with 3 neurons and inputs= 1,2,3. The weights to the input neurons are 4,5 and 6 respectively. Assume the activation function is a linear constant value of 3. What will be the output ?

A: 96 (*3(1*4+2*5+6*3)*)

Q3. If you increase the number of hidden layers in a Multi Layer Perceptron, the classification error of test data always decreases. True or False?

A. True

B. False

B: False (due to Overfitting)

Q2. For which values of w1, w2 and b does a simple perceptron (step function) implement an AND function?

A. Bias = -1.5, w1 = 1, w2 = 1

B. Bias = 1.5, w1 = 2, w2 = 2

C. Bias = 1, w1 = 1.5, w2 = 1.5

D. None of these

Option A:

f(-1.5*1 + 1*0 + 1*0) = f(-1.5) = 0

f(-1.5*1 + 1*0 + 1*1) = f(-0.5) = 0

f(-1.5*1 + 1*1 + 1*0) = f(-0.5) = 0

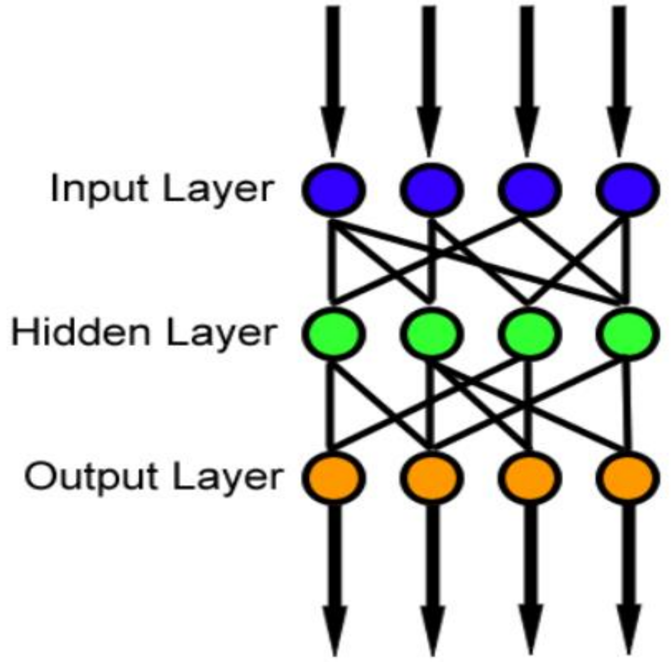
f(-1.5*1 + 1*1+ 1*1) = f(0.5) = 1

Feedforward in ANN

upGrad

- In a feedforward network, information always moves one direction; it never goes backwards or make a loop.
- In this case, we achieved output what we expected from the input without turning back or fine-tuning.
- FFN Algorithm:

```
1. $h^0 = x_i$
2. for $l$ in [1,2,.......,L]:
    1. $h^l = \sigma(W^l . h^{l-1} + b^l)$
3. $p_i = e^{W^o . h^L}$
4. $p_i$ = normalize($p_i$)
```

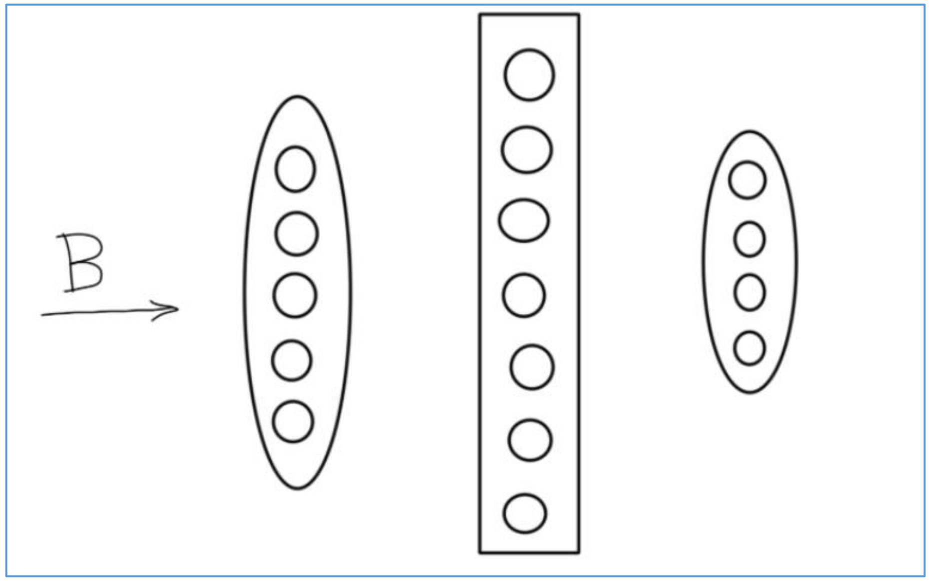


Batch Processing in Feedforward in ANN

upGrad

- Now, the entire calculation should happen for all the data points. This is almost impossible and computationally very expensive. Hence, in practice, the training happens in batches.

- A batch of size B is made of B data points and which is stacked side by side in a matrix. Thus, if the data points has 'd' features, then size of batch B is (d,B). This matrix in fed into ANN as input, also known as **Vectorized implementation.**



If **B = 32**, then

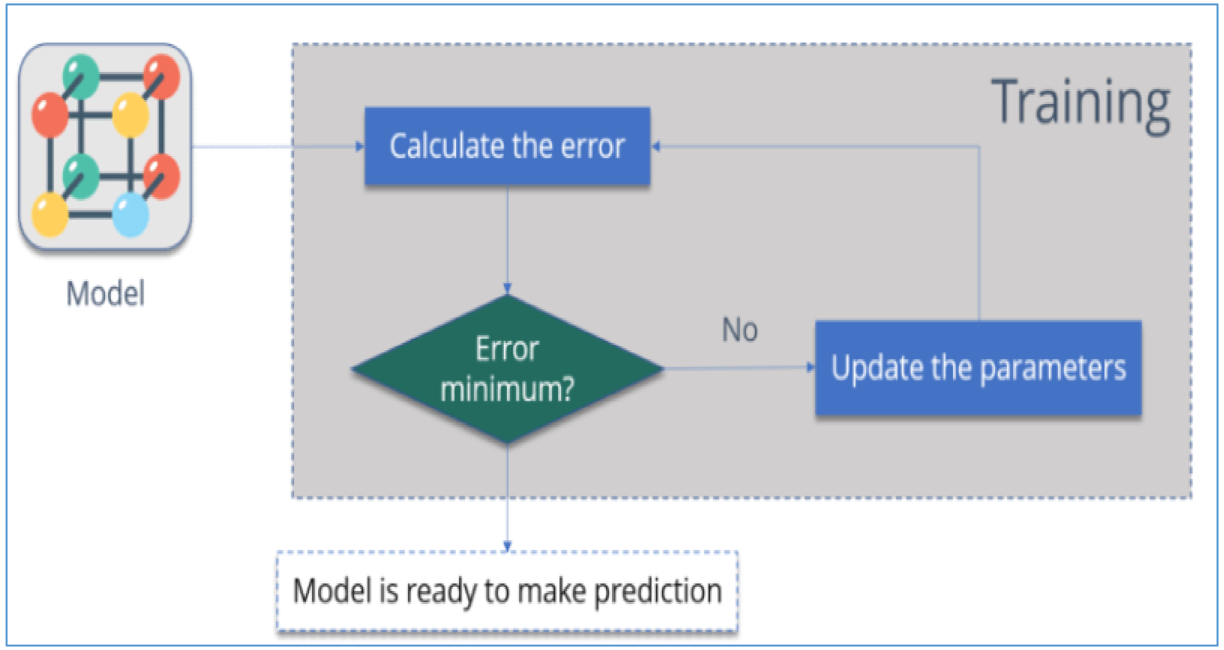
1. Size of B is:	1. 5x 32
2. Dimension of H1 is:	2. 7x32
3. Dimension of P is:	3. 4x 32

Backward Propagation

upGrad

- Our first choice Weights and Bias might not be a good choice, so in order to minimize the average loss, we need to fine tune the W and B matrix.

- The objective behind training a NN is to minimize the overall cost - The desired output (output from the last layer) minus the actual output is the cost (or the loss), and we need to tune the parameters **W and B** such that the total cost is minimized.

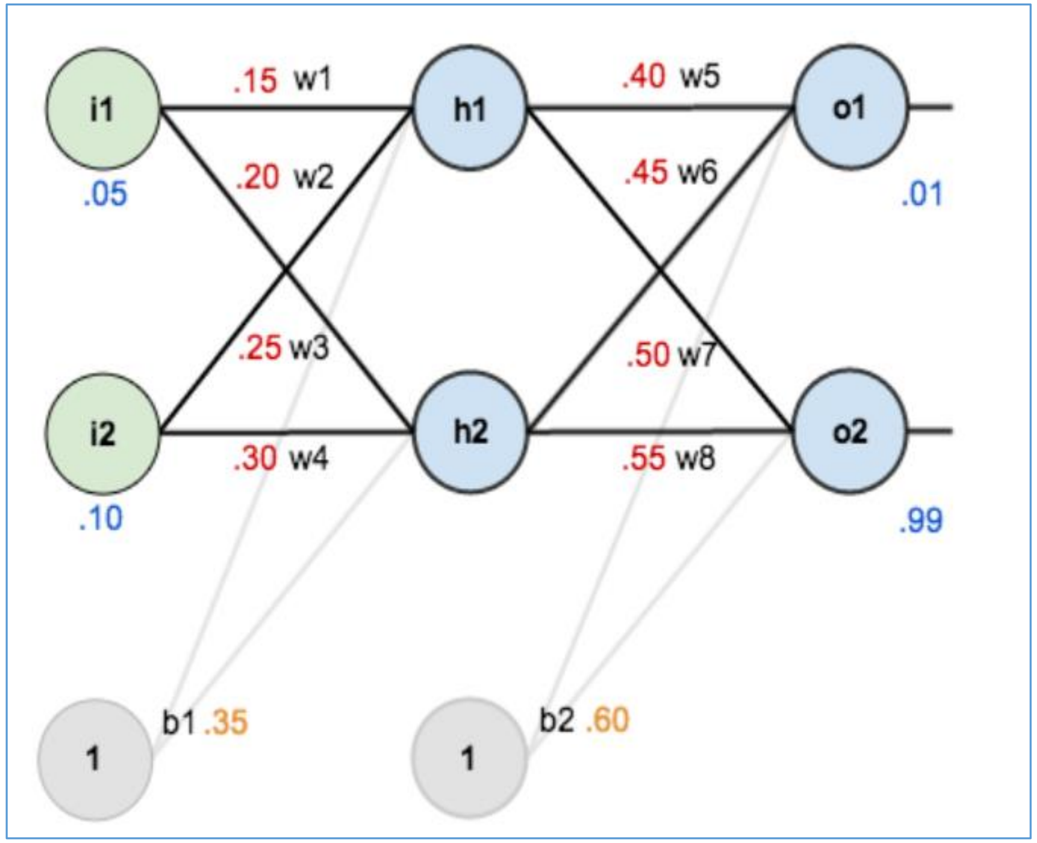


- we can't directly calculate the derivative of the loss function with respect to the weights and biases because the equation of the loss function does not contain the weights and biases. Therefore, we need the **chain rule** to help us calculate it.

$$W_{new} = W_{old} - \alpha . \frac{\partial L}{\partial W} .$$

Backward Propagation – Example

upGrad



Step – 1: Forward Propagation:

Net Input For h1:

net h1 = w1*i1 + w2*i2 + b1*1 → net h1 = 0.15*0.05 + 0.2*0.1 + 0.35*1 = 0.3775

Output Of h1:

out h1 = $1/1+e^{-\text{net h1}}$ → $1/1+e^{.3775}$ = 0.593269992

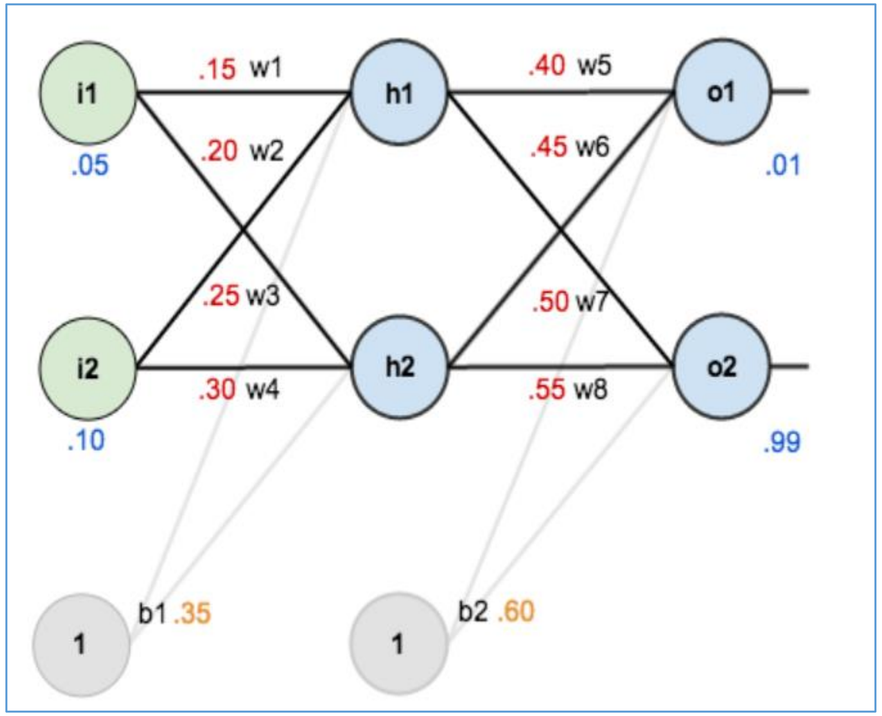
Output Of h2:

out h2 = 0.596884378

- ✓ 2 input data point, 1 hidden layer with 2 neuron, and 2 neuron in the output layer
- ✓ There are total 8 weights in the N/W
- ✓ The bias b1 = 0.35, b2 = 0.60
- ✓ Input 1 = 0.05, Input 2 = 0.10
- ✓ Actual Output (Ground Truth) = **0.01 and 0.99**

Backward Propagation – Example Cont…

upGrad



Step – 1: Forward Propagation:

We will repeat this process for the output layer neurons, using the output from the hidden layer neurons as inputs.

We have already got:

Out h1 = 0.593269992
Out h2 = 0.596884378

Output For o1:

net o1 = w5*out h1 + w6*out h2 + b2*1 → 0.4*0.593269992 + 0.45*0.596884378 + 0.6*1 = 1.105905967

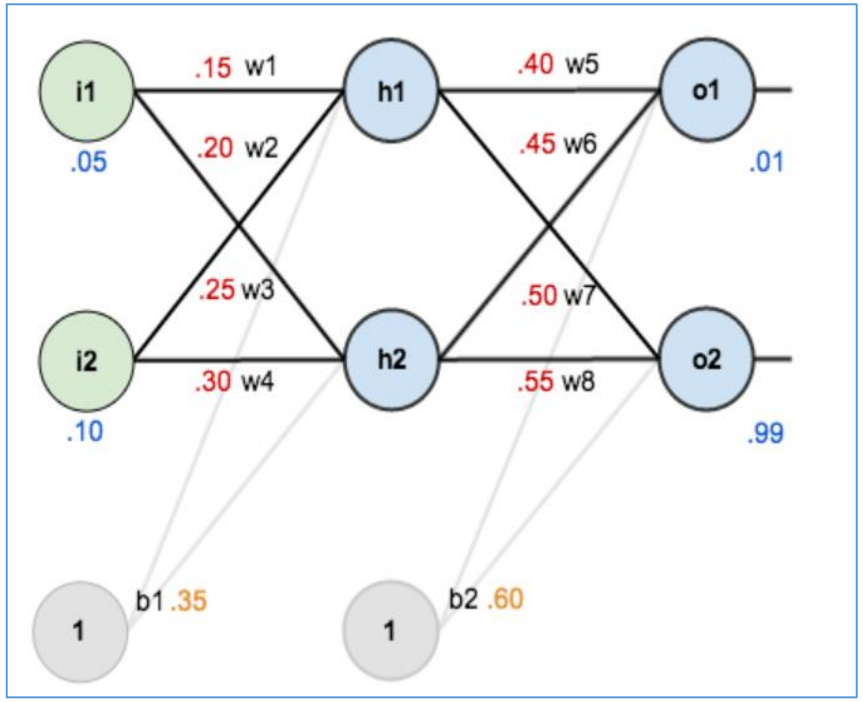
Out o1 = $1/1+e^{-\text{net o1}}$ → $1/1+e^{-1.105905967}$ = 0.75136507

Output For o2:

Out o2 = 0.772928465

Backward Propagation – Example Cont…

upGrad



Step –2: Calculating the Total Error

As we have calculated the final output (i.e. the predicted value), we can now calculate the total error. Let’s say we use MSE as our loss function.

Error For o1:

$E\ o1 = \Sigma 1/2(target - output)^2$ → $\frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$

Error For o2:

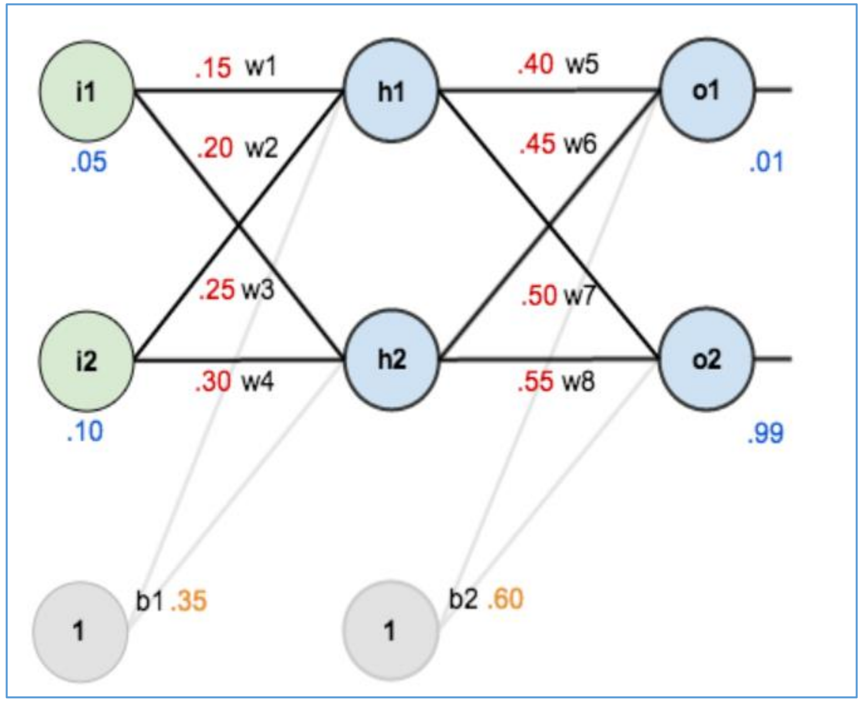
$E\ o2 = 0.023560026$

Total Error:

$E_{total} = E\ o1 + E\ o2$ → $0.274811083 + 0.023560026 = 0.298371109$

Backward Propagation – Example Cont…

upGrad

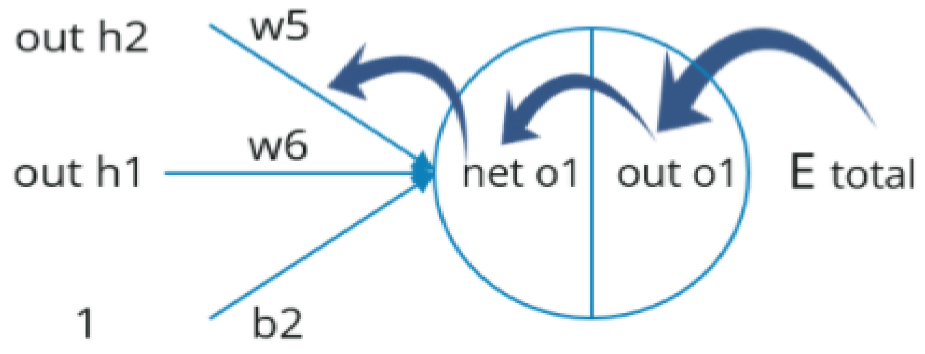


Step – 3: Backward Propagation

As we have calculated the total error now, we need to update each of the weights in the network so that they cause the actual output to be closer the target output, thereby minimizing the error for each output neuron and the network as a whole.

To start with, we can take **W5 as the starting point** and see how to calculate the updated W5. Similarly, we need to calculate W6, W7, W8 and then W1…W4. Now we need to find the contribution of W5 in the total error i.e. we want to know $\frac{\delta Etotal}{\delta w5}$

$$\frac{\delta Etotal}{\delta w5} = \frac{\delta Etotal}{\delta out\ o1} * \frac{\delta out\ o1}{\delta net\ o1} * \frac{\delta net\ o1}{\delta w5}$$

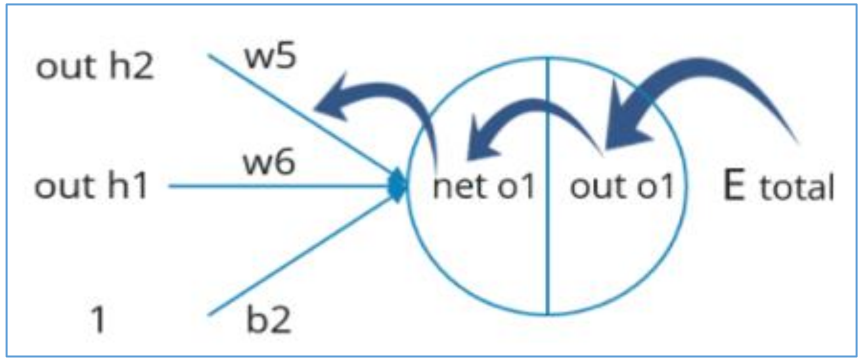


Backward Propagation – Example Cont…

upGrad

Step – 3: Backward Propagation

So, we need to calculate 3 different derivatives to know the impact of W5 on Total error:



$$E_{total} = 1/2(\text{target o1} - \text{out o1})^2 + 1/2(\text{target o2} - \text{out o2})^2$$

$$\frac{\delta Etotal}{\delta out\ o1} = -(\text{target o1} - \text{out o1}) = -(0.01 - 0.75136507) = 0.74136507$$

$$\text{out o1} = 1/1+e^{-neto1}$$

$$\frac{\delta out\ o1}{\delta net\ o1} = \text{out o1 } (1 - \text{out o1}) = 0.75136507\ (1 - 0.75136507) = 0.186815602$$

$$\text{net o1} = w5 * \text{out h1} + w6 * \text{out h2} + b2 * 1$$

$$\frac{\delta net\ o1}{\delta w5} = 1 * \text{out h1 } w5^{(1-1)} + 0 + 0 = 0.593269992$$

Backward Propagation – Example Cont…

Step – 3: Backward Propagation

Now we have calculated all the derivatives, we can now calculate the rate of change of error w.r.t change in weight W5.

$$\boxed{\frac{\delta Etotal}{\delta w5} = \frac{\delta Etotal}{\delta out\ o1} * \frac{\delta out\ o1}{\delta net\ o1} * \frac{\delta net\ o1}{\delta w5}} \longrightarrow \boxed{0.082167041}$$

And then updated weight of W5 can be calculated as :

$$\boxed{w5^{+} = w5 - n\frac{\delta Etotal}{\delta w5}} \longrightarrow \boxed{w5^{+} = 0.4 - 0.5 * 0.082167041}$$

$$\boxed{\text{Updated w5}} \longrightarrow \boxed{0.35891648}$$

Backward Propagation – Example Cont…

upGrad

Step – 3: Backward Propagation

Similarly, we can calculate W6*, W7*, and W8*.

$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$

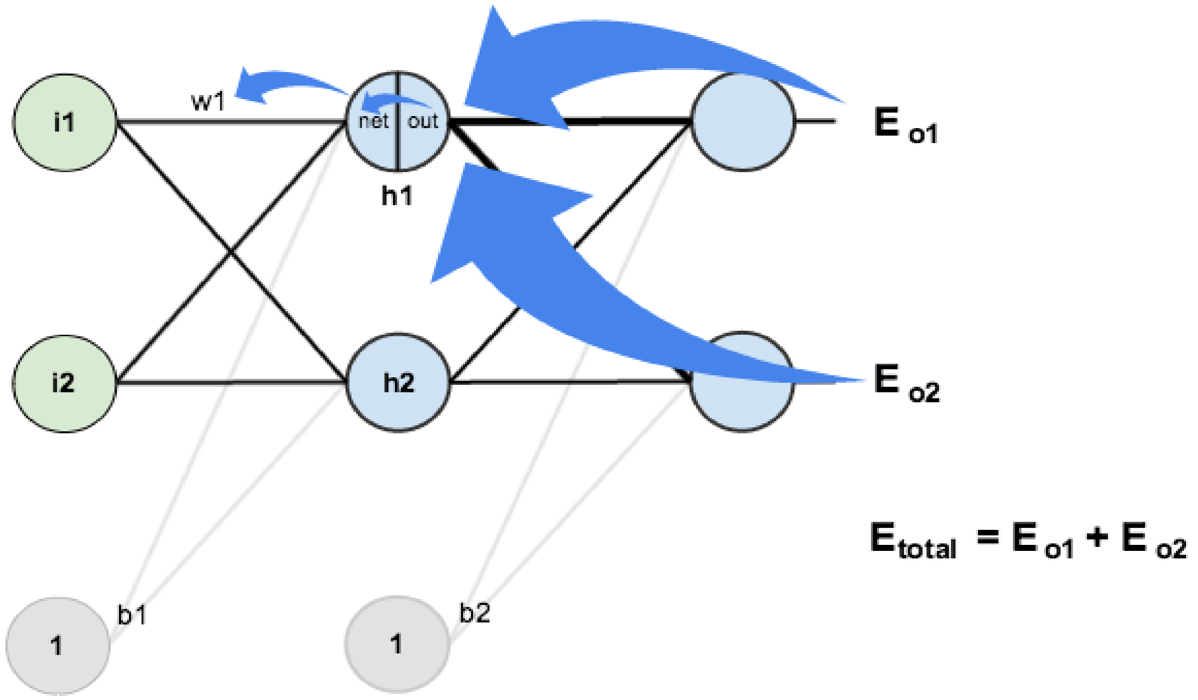
Step – 3: Backward Propagation to hidden Layer

Now, lets see how we can calculate the updated weight W1.

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\downarrow$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



Backward Propagation – Example Cont…

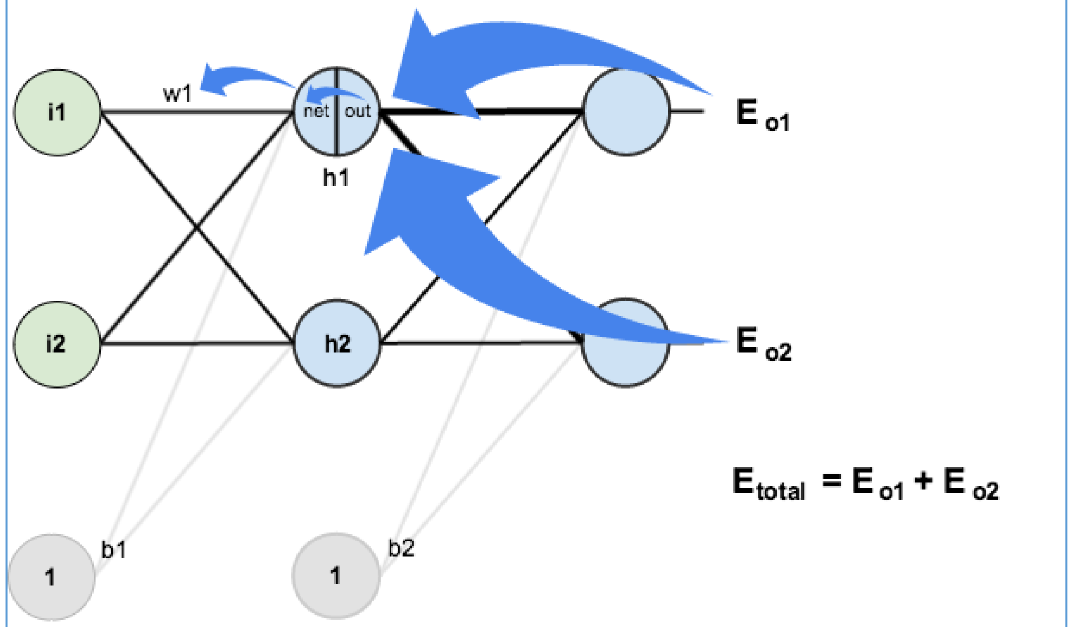
upGrad

Step – 3: Backward Propagation to one hidden Layer

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\downarrow$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



Calculation of $\frac{\partial E_{o1}}{\partial out_{h1}}$

1. $\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$

2. $\frac{\partial E_{o1}}{\partial net_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = 0.74136507 * 0.186815602 = 0.138498562$

3. $net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$

   $\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$

4. $\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}} = 0.138498562 * 0.40 = 0.055399425$

5. $\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = 0.055399425 + -0.019049119 = 0.036350306$

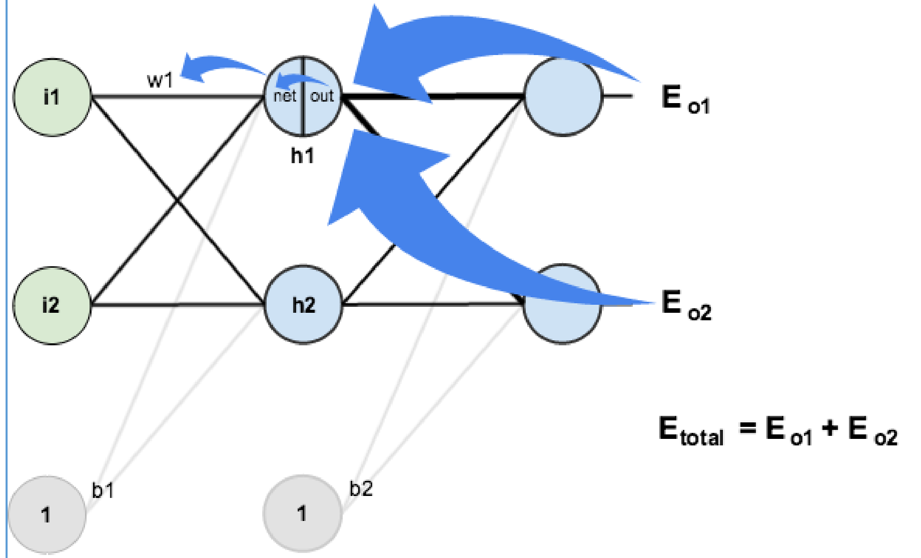
Backward Propagation – Example Cont…

upGrad

Step – 3: Backward Propagation to one hidden Layer

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



Calculation of $\frac{\partial out_{h1}}{\partial net_{h1}}$

1. $$out_{h1} = \frac{1}{1+e^{-net_{h1}}}$$

$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$$

Calculation of $\frac{\partial net_{h1}}{\partial w_1}$

$$net_{h1} = w_1 * i_1 + w_3 * i_2 + b_1 * 1$$

$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$$

Backward Propagation – Example Cont…

upGrad

Step – 3: Backward Propagation to one hidden Layer

Putting it all together:

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

We can now update w_1:

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

Repeating this for w_2, w_3, and w_4

$$w_2^+ = 0.19956143$$

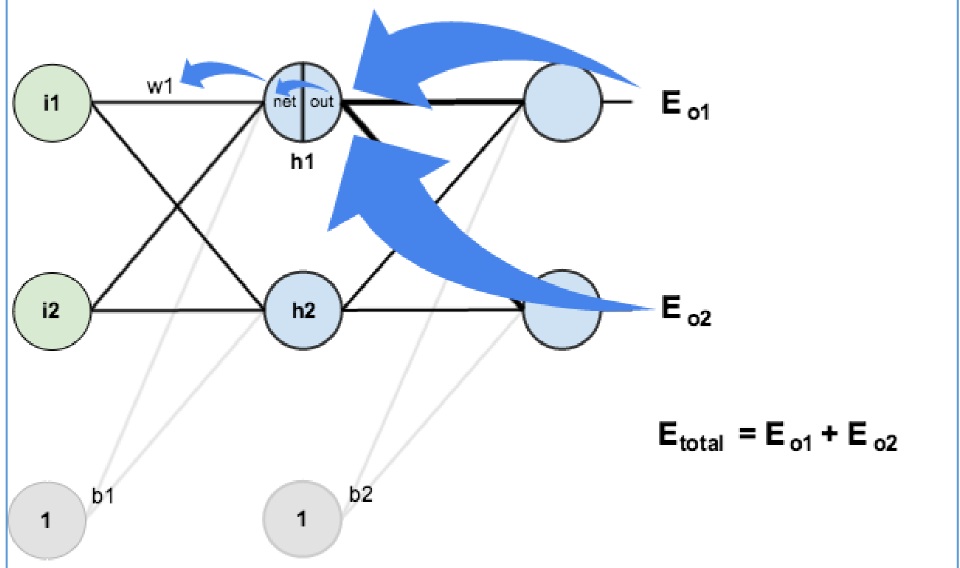
$$w_3^+ = 0.24975114$$

$$w_4^+ = 0.29950229$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\downarrow$$

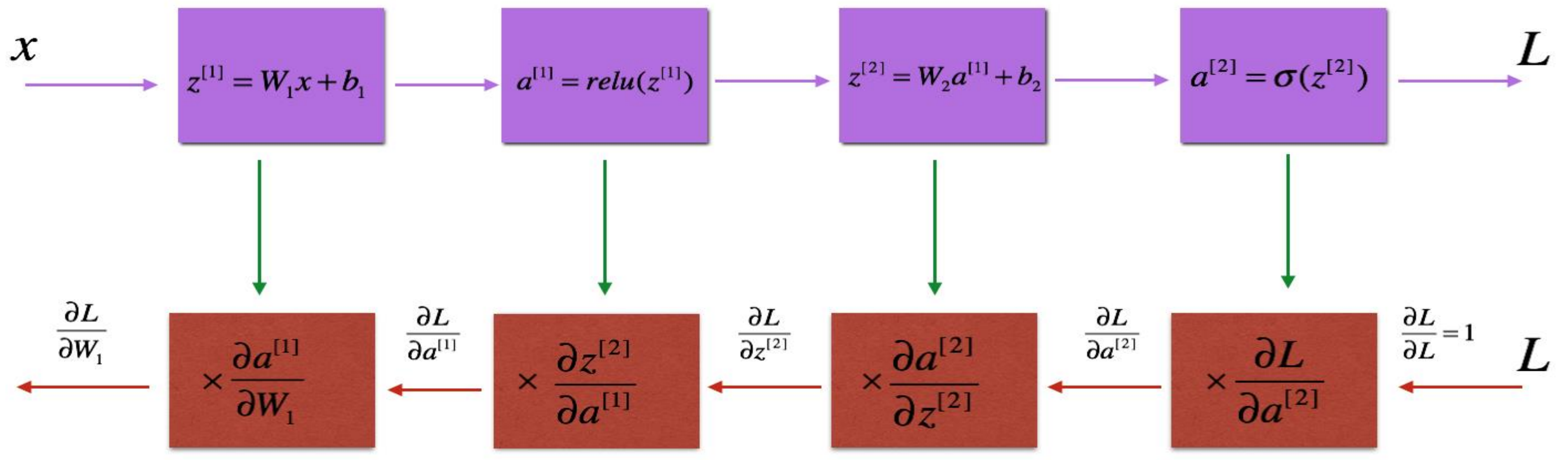
$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



After this first round of backpropagation, the total error is now down to 0.291027924 (from 0.298371109). But after repeating this process 10,000 times, the error comes down to 0.0000351085 with the two outputs neurons generate 0.015912196 (vs 0.01 target) and 0.984065734 (vs 0.99 target).

Backward Propagation

upGrad



x
$z^{[1]} = W_1 x + b_1$
$a^{[1]} = relu(z^{[1]})$
$z^{[2]} = W_2 a^{[1]} + b_2$
$a^{[2]} = \sigma(z^{[2]})$
L
$\frac{\partial L}{\partial W_1}$
$\times \frac{\partial a^{[1]}}{\partial W_1}$
$\frac{\partial L}{\partial a^{[1]}}$
$\times \frac{\partial z^{[2]}}{\partial a^{[1]}}$
$\frac{\partial L}{\partial z^{[2]}}$
$\times \frac{\partial a^{[2]}}{\partial z^{[2]}}$
$\frac{\partial L}{\partial a^{[2]}}$
$\times \frac{\partial L}{\partial a^{[2]}}$
$\frac{\partial L}{\partial L} = 1$
L

$$\boxed{\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial a^{[1]}} \frac{\partial a^{[1]}}{\partial W_1}}$$

We know that propagation is used to calculate the gradient of the loss function with respect to the parameters.

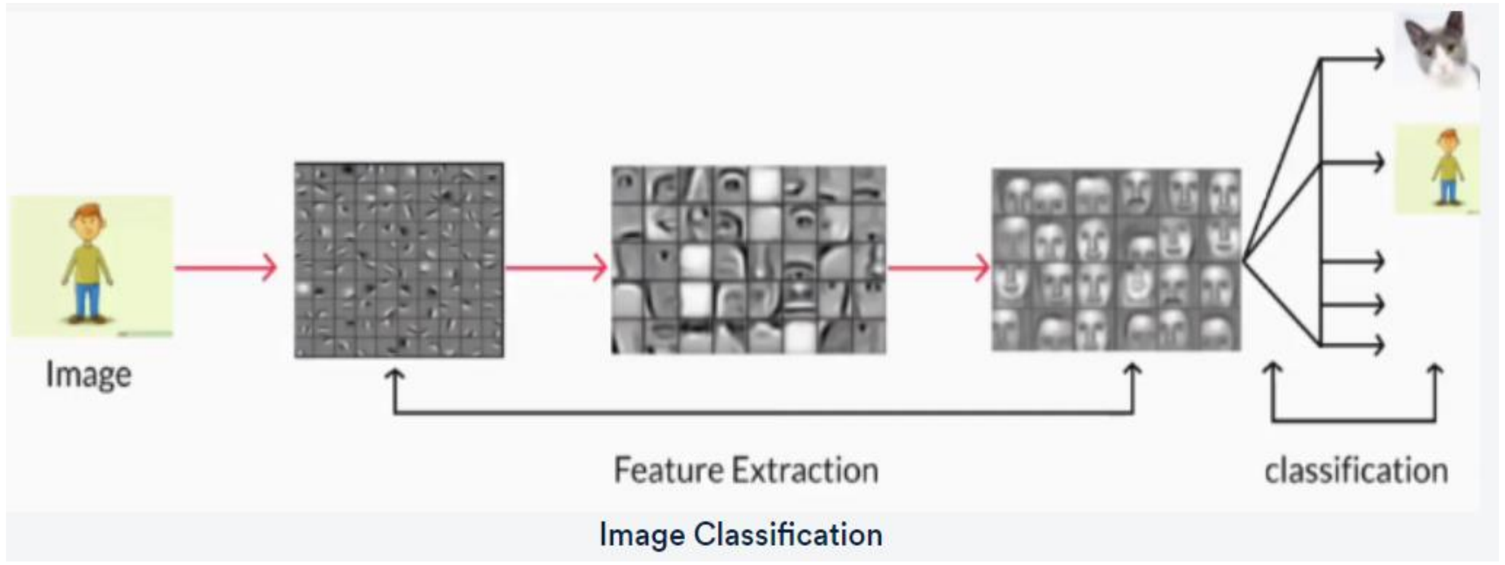
https://www.youtube.com/watch?v=QJoa0JYaX1I

https://www.edureka.co/blog/backpropagation/

https://www.analyticsvidhya.com/blog/2020/07/neural-networks-from-scratch-in-python-and-r/

CNN

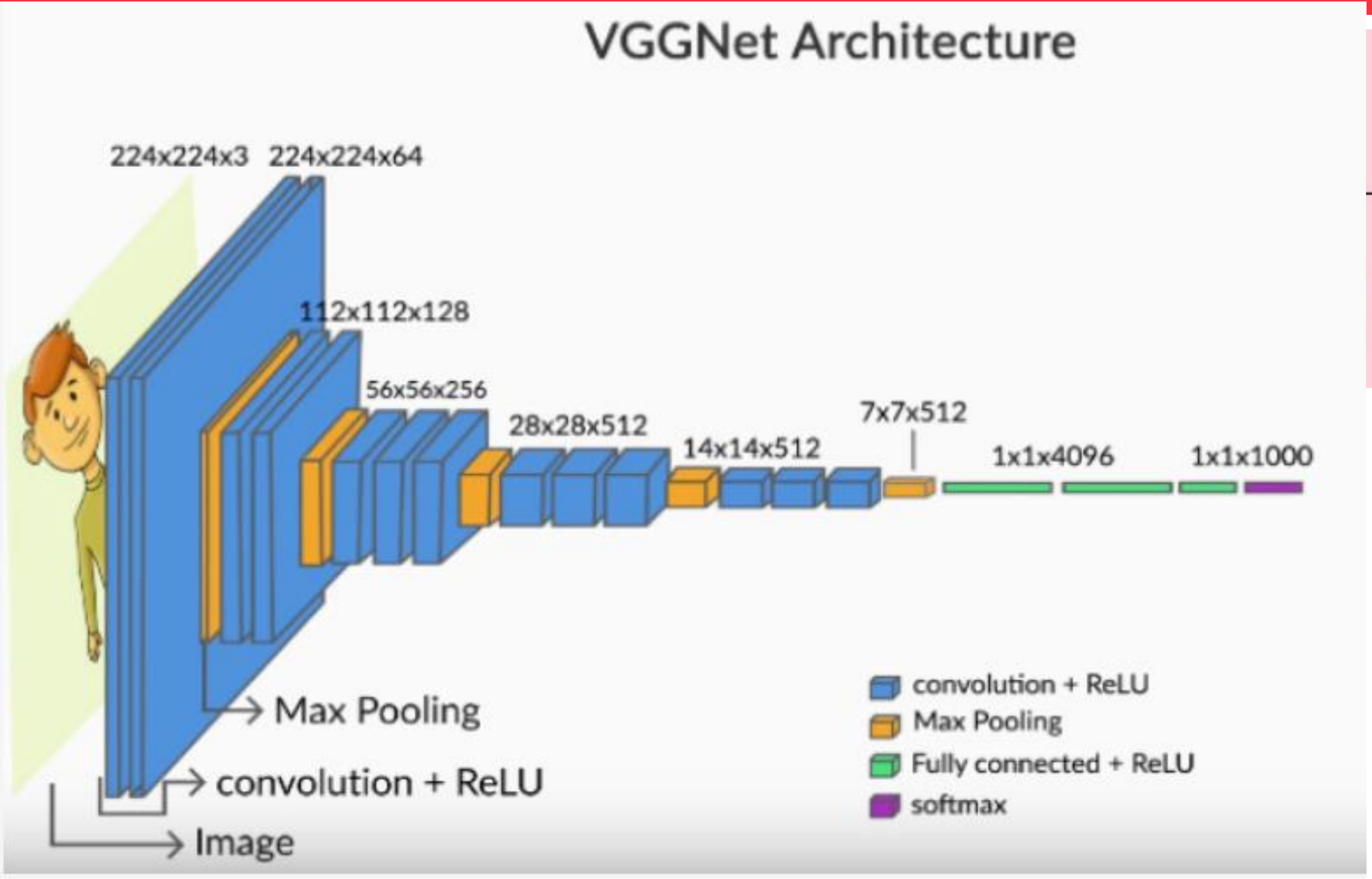
- ✓ Convolutional Neural Networks a special type of neural network that roughly imitates human vision.
- ✓ The architecture of CNNs uses many of the working principles of the animal visual system and thus they can overcome some of the challenges related to Image processing (Scale, Viewpoint, varying Illumination, different background etc..)
- ✓ **CNN acts as a feature extractor** for images, and thus, can be used in a variety of ways to process images/videos.

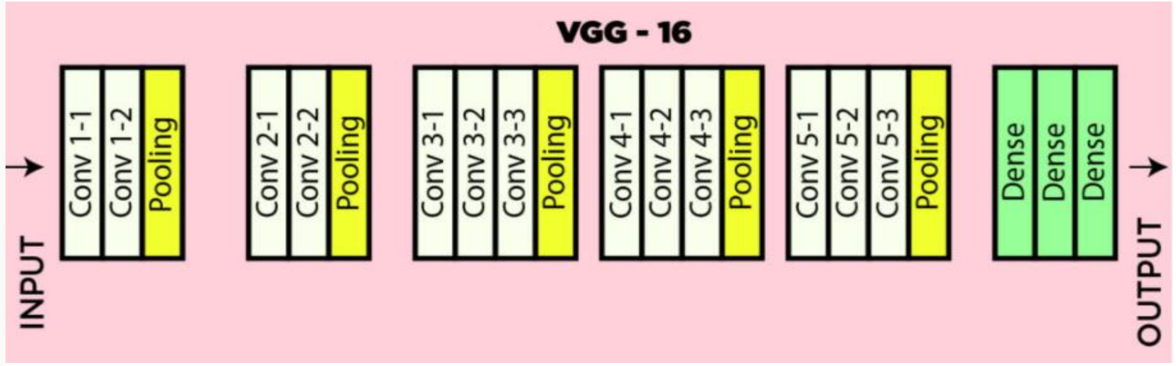


- ✓ **Optical character recognition (OCR)**
- ✓ Machine inspection
- ✓ Retail (e.g. automated checkouts)
- ✓ 3D model building (photogrammetry)
- ✓ **Medical imaging**
- ✓ Automotive safety
- ✓ Match move (e.g. merging CGI with live actors in movies)
- ✓ **Motion capture (mocap)**
- ✓ Surveillance
- ✓ **Fingerprint recognition and biometrics**

VGGNET (Visual Geometry Graph NET) Architecture

upGrad





- ✓ The final layer is a feature map of 4096 neuron
- ✓ **4096-dimensional feature vector** is fed into softmax layer to classify 1000 different type of images
- ✓ It has more than 130M parameter

The main Concept in CNN is:

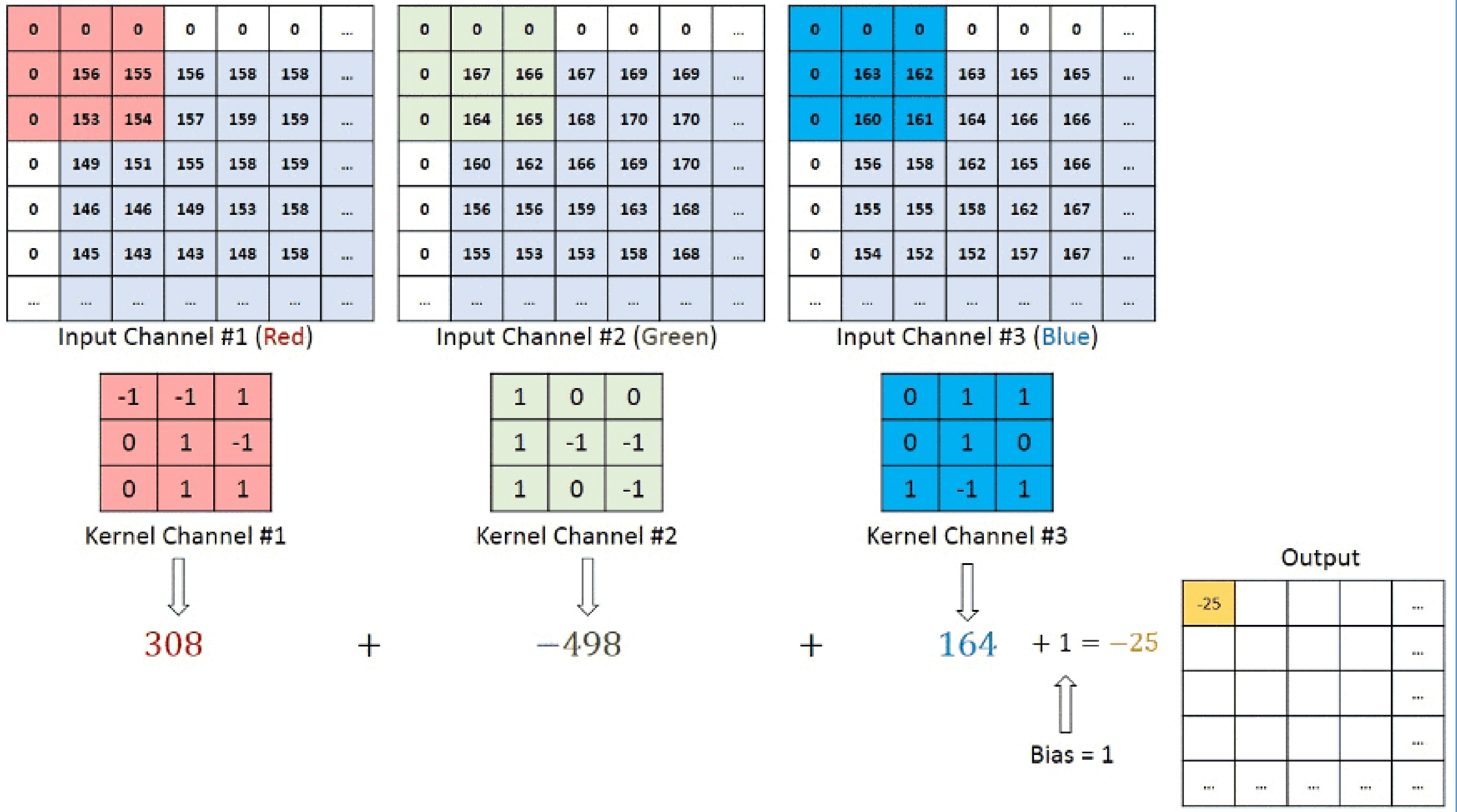
- ➢ **Convolutions**
- ➢ **Pooling**
- ➢ **Feature Maps**

https://www.kaggle.com/blurredmachine/vggnet-16-architecture-a-complete-guide/data

https://towardsdatascience.com/architecture-comparison-of-alexnet-vggnet-resnet-inception-densenet-beb8b116866d

Convolution

Convolution is a mathematical operation on two functions (*f* and *g*) that produces a third function (f*g) that expresses how the shape of one is modified by the other.



0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

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

Kernel Channel #1

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

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

$$308 + -498 + 164 + 1 = -25$$

Bias = 1

Output

-25				...
				...
				...
				...
...	...	...	...	...

- ✓ In convolution, the number of filters should match the depth of the image
- ✓ Red – (255,0,0)
  Blue – (0,255,0)
  White – (255,255,255)
- ✓ Black & White Image – only one channel
  Black – 0
  White - 255

Convolution… Stride and Padding

upGrad

✓ **Stride**: Stride is the number of pixels we move the filter (both horizontally and vertically) to perform convolution operation. A stride of length 1 helps to find more fine-grained features.

Stride results in reducing the dimension of the image. Another problem is you cannot convolve a (4, 4) image with a (3, 3) filter using a stride of 2. However this can be solved using Padding.

✓ **Padding**: Padding is the number of pixels we add all around the image. As you can in Figure 10 that the padding of 1 is used.

Given the following size :

- Image - n x n
- Filter - k x k
- Padding - P
- Stride - S

$$\text{Size of convolved image} = \left(\frac{n+2P-k}{S} + 1\right), \left(\frac{n+2P-k}{S} + 1\right)$$

After padding, we get an image of size (n + 2P) x (n+2P). After we convolve this padded image with the filter, we get:

Convolution… Stride and Padding

upGrad

Q1. Given an image of size 5x5, filter size 3x3, stride 2, what is the padding required (on either side) to make the output size same as input size?

→ 3 pixel on both side

Q2. The input image has been converted into a matrix of size 28 X 28 and a kernel/filter of size 7 X 7 with a stride of 1. What will be the size of the convoluted matrix?

→ 22X22

Q3. Given an input image of size 224x224, a filter of size 5x5 and stride of 2, what are the possible values of padding?

→ Not possible for any padding. (n+2P-k) should be divisible by stride 's'. So, (224 + 2xPadding - 5) should be divisible by 2. This is not possible for any value of padding.

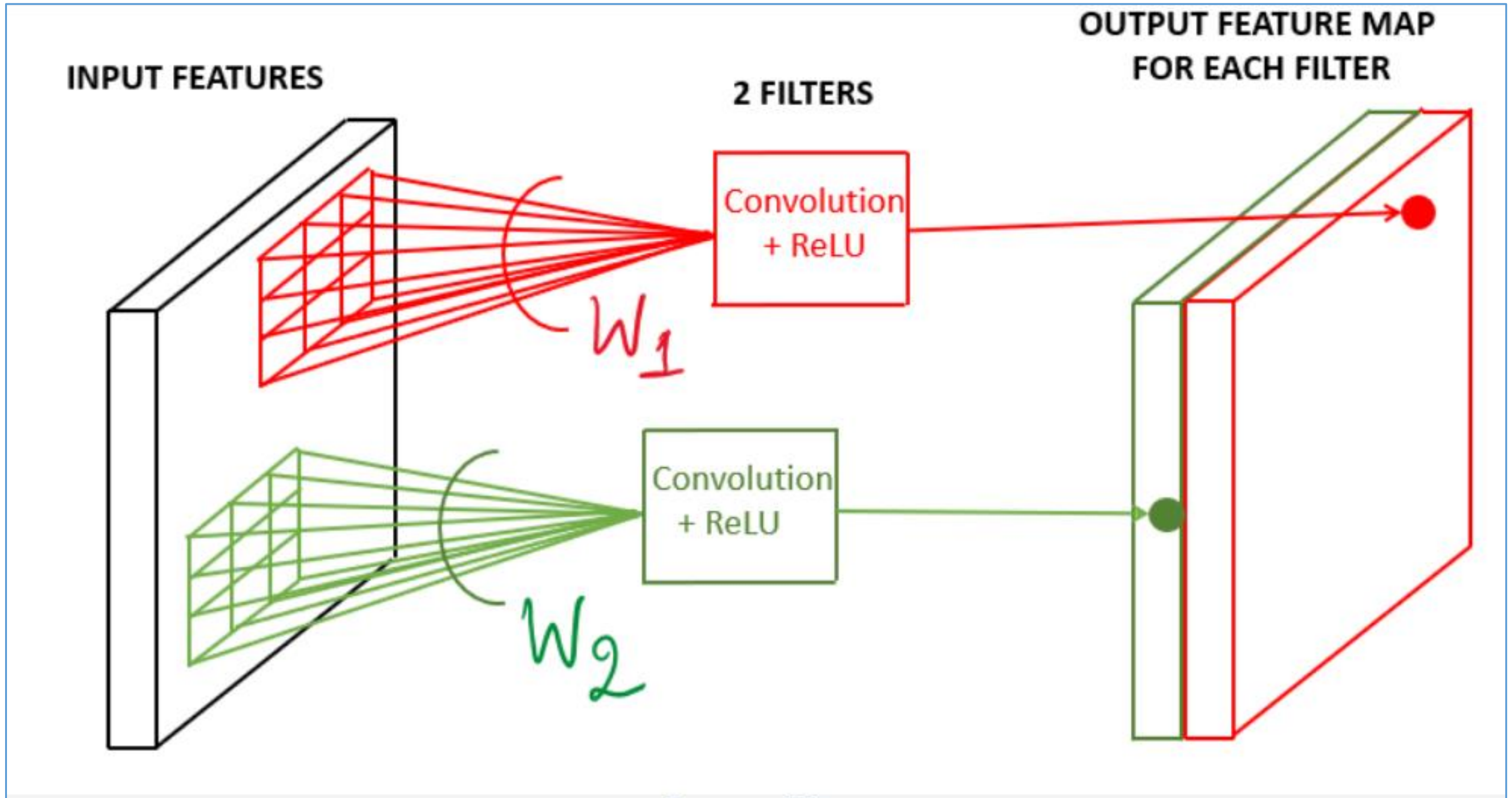
Q4. **True or False**: Doing a normal convolution without padding with a k x k filter, k>1, always reduces the size of the images.

→ True

Feature Maps

upGrad

A **neuron** is basically the dot product of the Image input and filter whose weights are learnt during training. For example, a (3, 3, 3) filter (or neuron) has 27 weights. No matter how big the image is, the parameters only depend on the filter size.



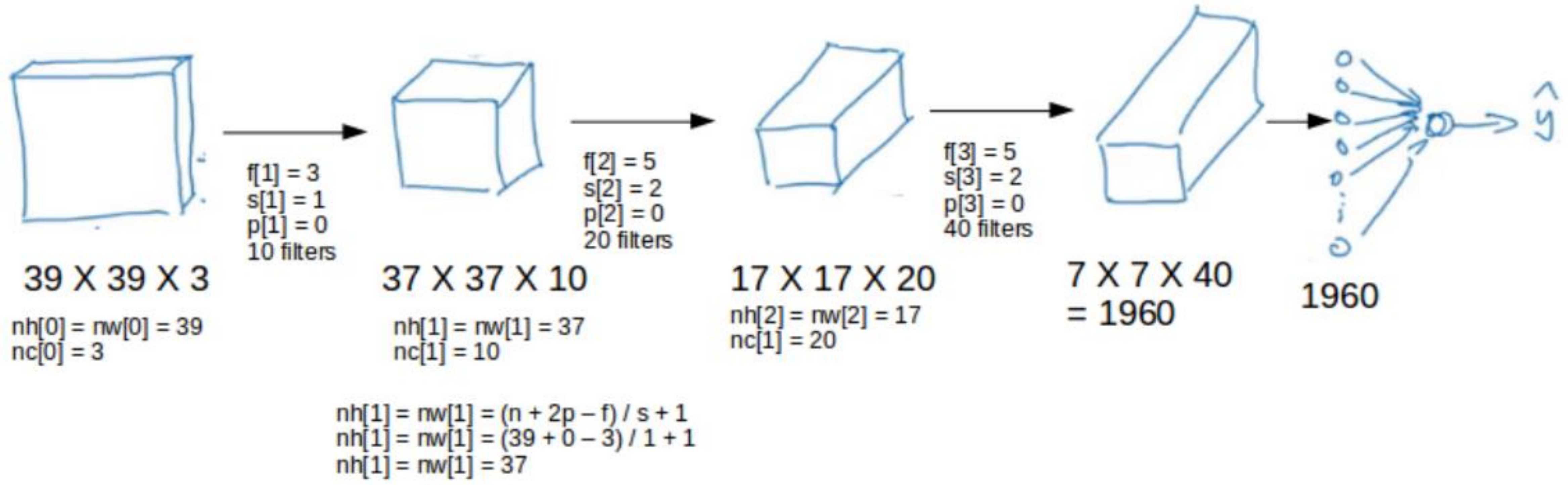
Feature Map

The term '**feature map**' refers to the (non-linear) output of the activation function, not what goes into the activation function (i.e. the output of the convolution).

Here, two different neurons W1 and W2 produce two feature maps.

Feature Maps

upGrad



We take an input image (size = 39 X 39 X 3 in our case), convolve it with 10 filters of size 3 X 3, and take the stride as 1 and no padding. This will give us an output of 37 X 37 X 10. We convolve this output further and get an output of 7 X 7 X 40 as shown above. Finally, we take all these numbers (7 X 7 X 40 = 1960), unroll them into a large vector, and pass them to a classifier that will make predictions.

Feature Maps

upGrad

Q1. If we use 32 filters (or kernels) of size 3x3x3 to convolve an image of size 128x128x3, how many feature maps will be created after the convolution?

→ **32**

Q2. Given an image of size 128x128x3, a stride of 1, padding of 1, what will be the size of the output if we use 32 kernels of size 3x3x3?

→ **128X128X32**

Q3. Calculate the number of trainable parameters if the input shape is (128x128x10)x3 and it is convolved with 32 3D filters of size (4x4x2) ?

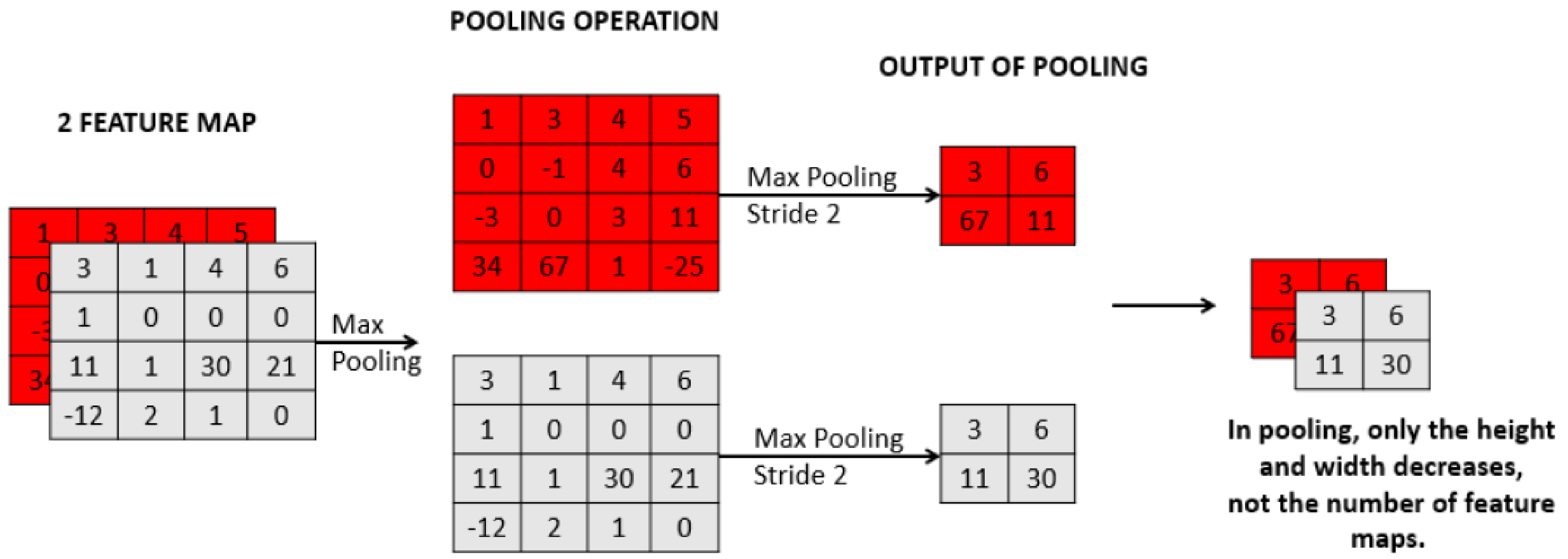
→ **32 x ((4x4x2) x 3 +1) = 3104 ;** **1 is added for bias.**

Pooling

upGrad

Pooling layer usually aggregates the feature maps from the previous layers. Pooling layer is used to compact the representation of the feature map. Thus the no. of feature maps remains same, and only height and width decreases.

- Max Pooling
- Average Pooling



There is no weight associated here, as we are just doing a mathematical operations.

Transfer Learning

upGrad

- **CNNs** are basically features extractors i.e. it learns the representation of an image.
- These learnings are not confirmed to a given dataset, but can be applied to any dataset.
- In practical, we use the pre-trained model and apply the transfer learning for our own task.

  ✓ Freeze the initials layers, remove the last few layers, add layers according to need and **train only these newly added layers.** (when dataset is less)

  ✓ **Fine tune the model –** retrain all or few layers with a small learning rate (when enough dataset)

```
import resnet
model = resnet.ResnetBuilder.build_resnet_18((img_channels, img_rows, img_cols), nb_classes)
```

https://github.com/Calysto/conx-notebooks/blob/60106453bdb66a83da7c2741d7644b7f8ee94517/MNIST.ipynb

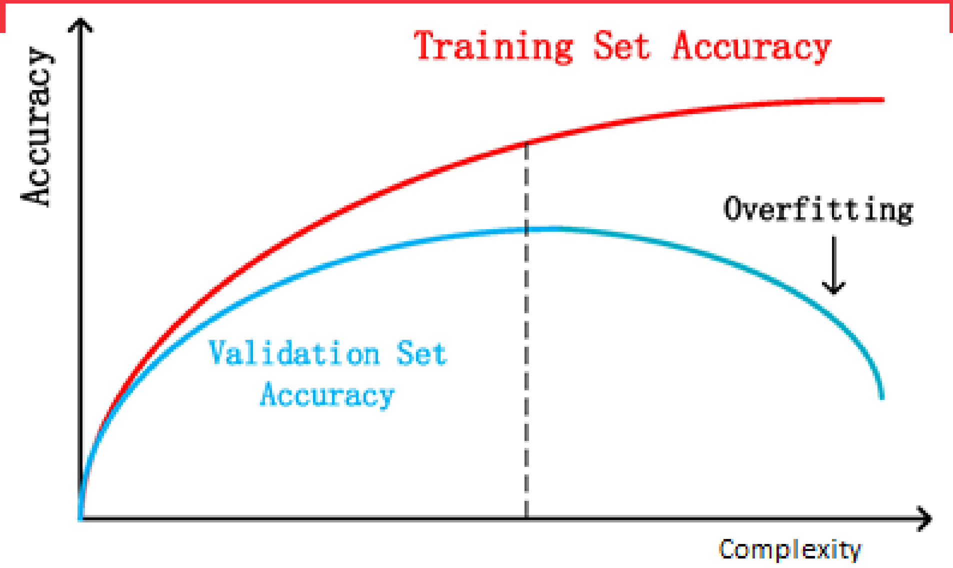
Overfitting

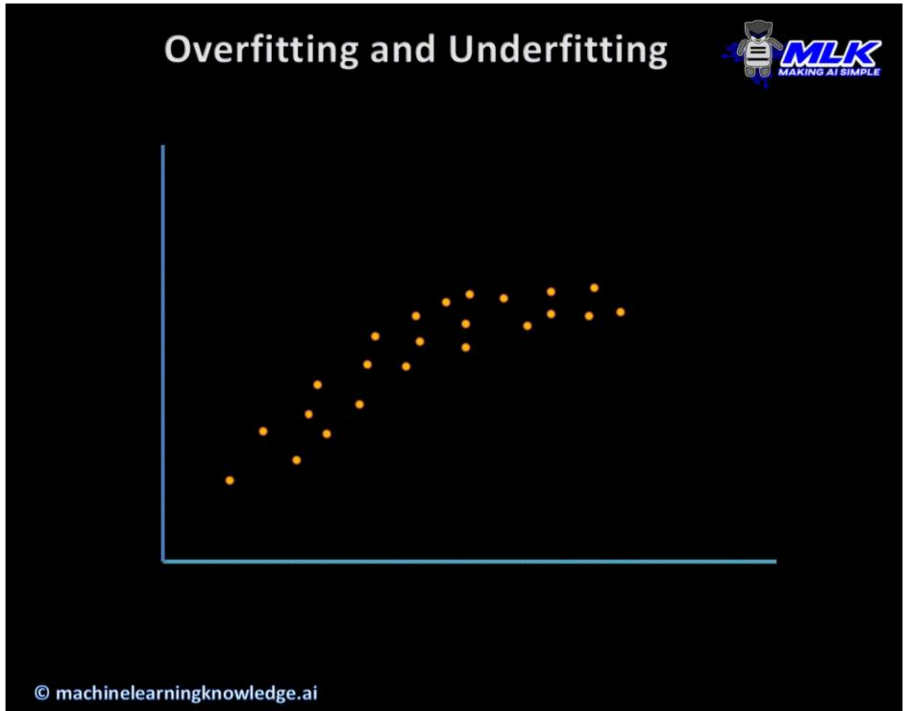
upGrad

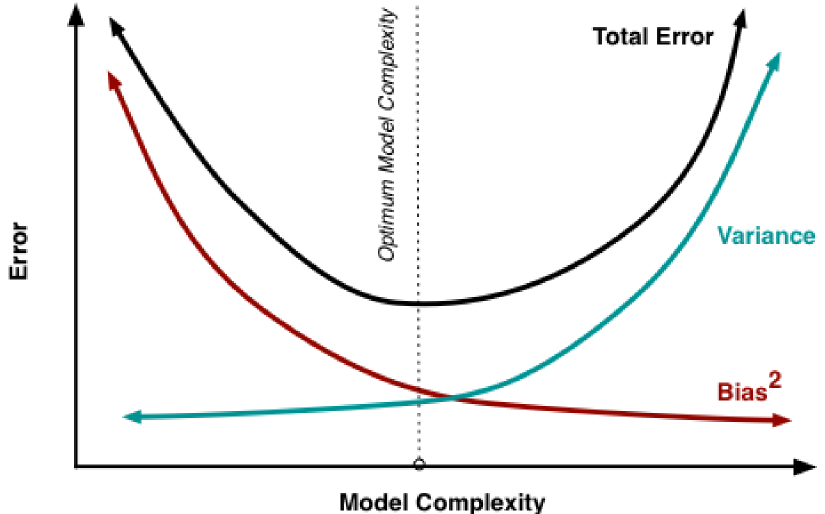
- ANN/CNN models are heavily over-parameterized and can often get to perfect results on training data. However, they do not produce the similar result on unseen data, suggesting the model is overfitted.

- There are multiple ways to avoid overfitting:

  - Data Augmentation
  - L1/L2 Regularization
  - Dropout
  - Early Stopping





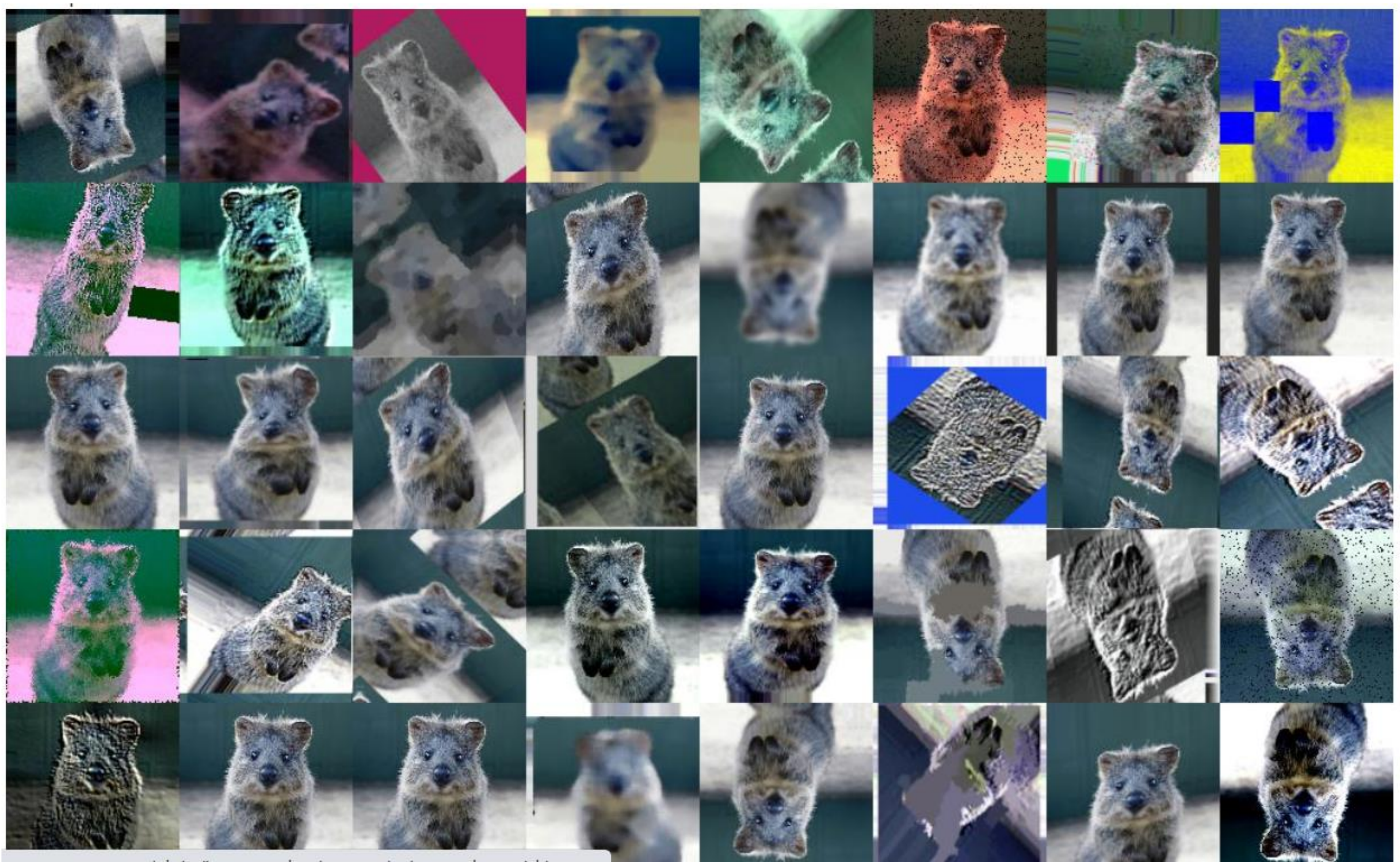


1. Data Augmentation

upGrad

- **Data Augmentation** simply means increasing size of the data that is increasing the number of images present in the dataset.

- Some of the popular image augmentation techniques are **flipping, translation, rotation, scaling, changing brightness**



- Can be implemented in many ways:

  - skimage
  - imgaug
  - keras

https://github.com/AgaMiko/data-augmentation-review

2. L1/L2 Regularization

upGrad

- **L1 or Lasso regularization:**

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}|\beta_j| = \mathrm{RSS} + \lambda\sum_{j=1}^{p}|\beta_j|.$$

- **L2 or Ridge regularization:**

$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2 = \mathrm{RSS} + \lambda\sum_{j=1}^{p}\beta_j^2$$

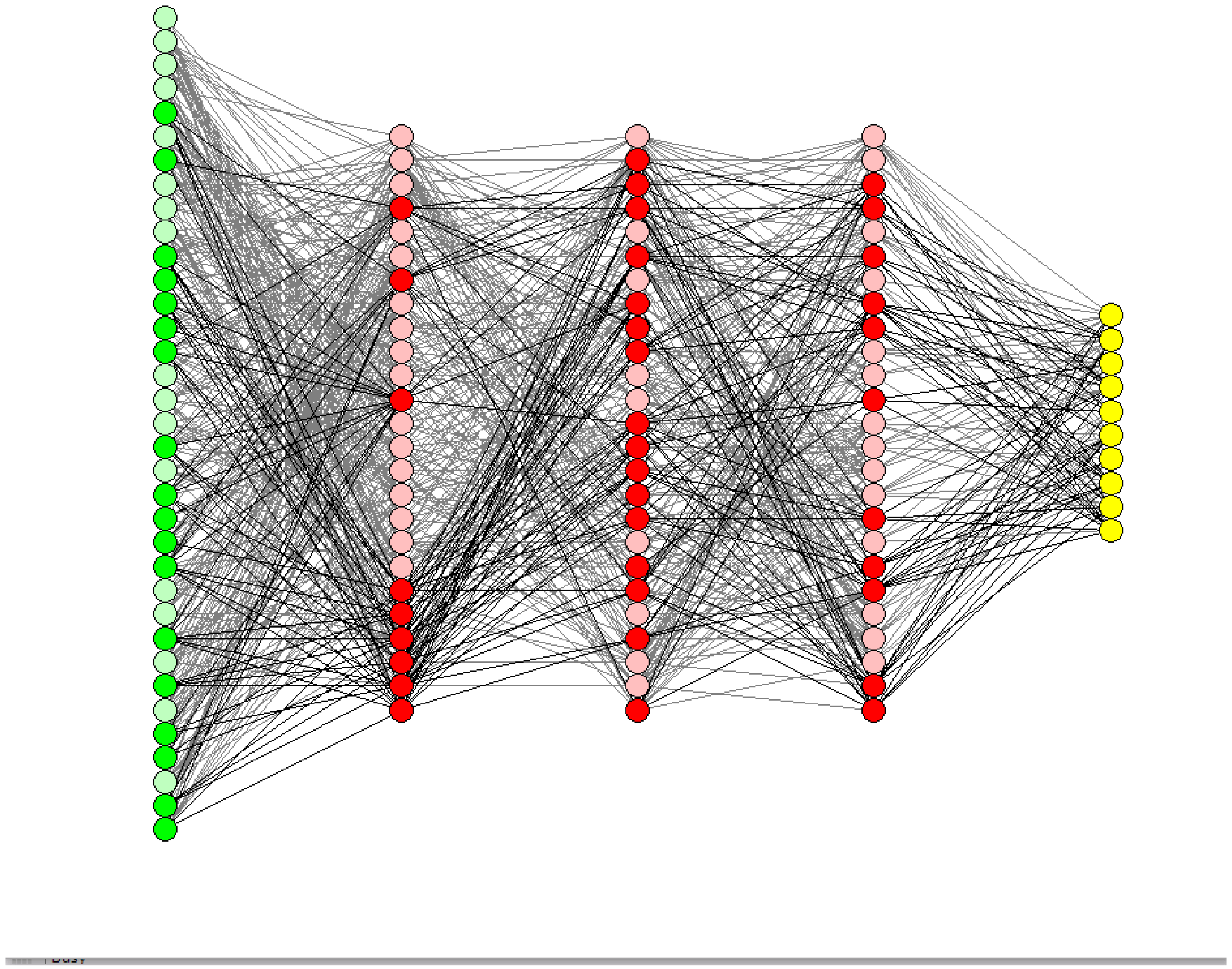
```
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', kernel_regularizer=tensorflow.keras.regularizers.l1(0.01)))

model.add(Conv2D(64, kernel_size=(3, 3), activation='relu', kernel_regularizer=tensorflow.keras.regularizers.l2(0.01)))
```

https://www.tensorflow.org/api_docs/python/tf/keras/regularizers/L1
https://www.tensorflow.org/api_docs/python/tf/keras/regularizers/L2

3. Dropout

upGrad

- **Dropout** is a most frequently used technique to avoid complexity/overfitting where randomly selected neurons are ignored during training process.
- It can also be thought of as an ensemble technique in machine learning. So each iteration has a different set of nodes and this results in a different set of outputs.



```
from keras.layers import Dropout
model.add(Conv2D(64, (3, 3), padding=‘5'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
```

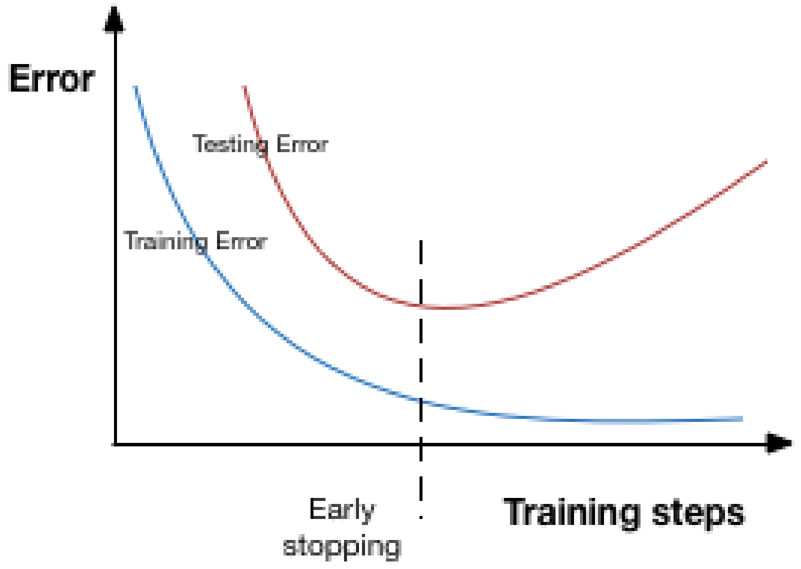
4. Early Stopping

- How to decide the no. of epoch for training?

- A good idea is to train on the training dataset but to stop training at the point when performance on a validation dataset starts to degrade

```
from keras.callbacks import EarlyStopping

EarlyStopping(monitor='val_loss', patience=5)
```

monitor: Quantity to be monitored.
patience: Number of epochs with no improvement after which training will be stopped.



upGrad
#LifeKoKaroLift

Q & A