Basic Deep Learning in Computer Vision

Day 2

Morning: Introduction to Artificial Neural Network

Afternoon: Introduction to Convolution Neural Network

Programme

	Morning	Afternoon

Day 1	• Computer Vision
	• Image Libraries
	    • Activity 1: Getting Started with Libraries
		• Image Preprocessing
		• Image Augmentation
		    • Activity 2: Image preprocessing
		    • Activity 3: Image Augmentation

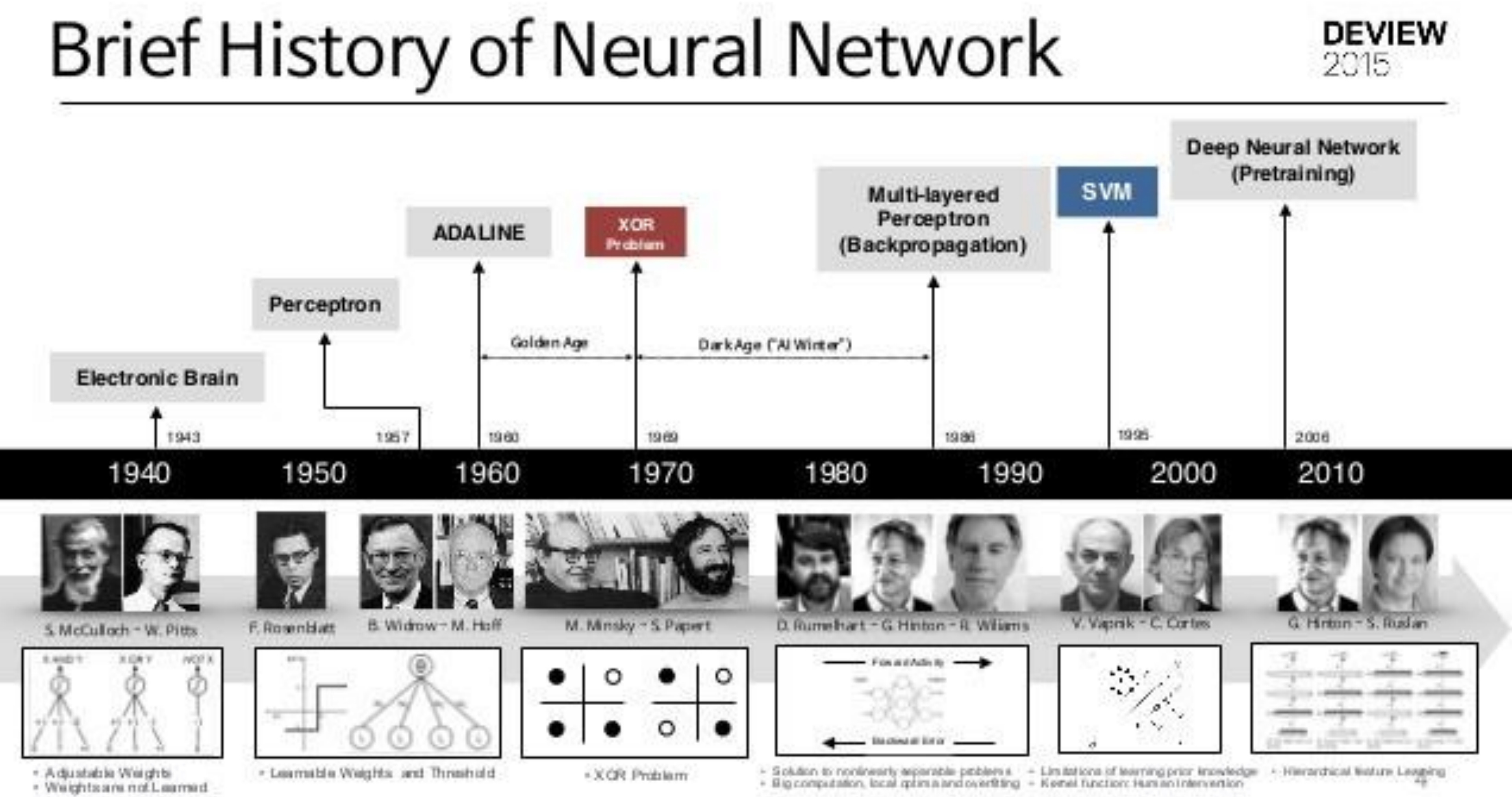
Day 2	• Basic of Neural Network
	    • Activity 4: Building NN with Python
	• Introduction to Keras
	    • Activity 5: Building NN with Keras
		• Image Convolution
		• Convolution Neural Network (CNN)
		    • Activity 6: Create and use CNN
		• Quiz

Artificial Neural Network (ANN)

Definition of ANN

- An ANN is based on ***a collection of connected units or nodes called artificial neurons***, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can ***transmit a signal to other neurons***. An artificial neuron that ***receives a signal then processes it and can signal neurons connected to it***. The "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs.
- The connections are called edges. Neurons and edges typically have a ***weight that adjusts as learning proceeds***. The weight increases or decreases the strength of the signal at a connection. ***Neurons may have a threshold*** such that a signal is sent only if the aggregate signal crosses that threshold.
- Typically, neurons are aggregated into ***layers***. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the ***input layer***), to the last layer (the ***output layer***), possibly after traversing the layers multiple times.
- Ref: https://en.wikipedia.org/wiki/Artificial_neural_network

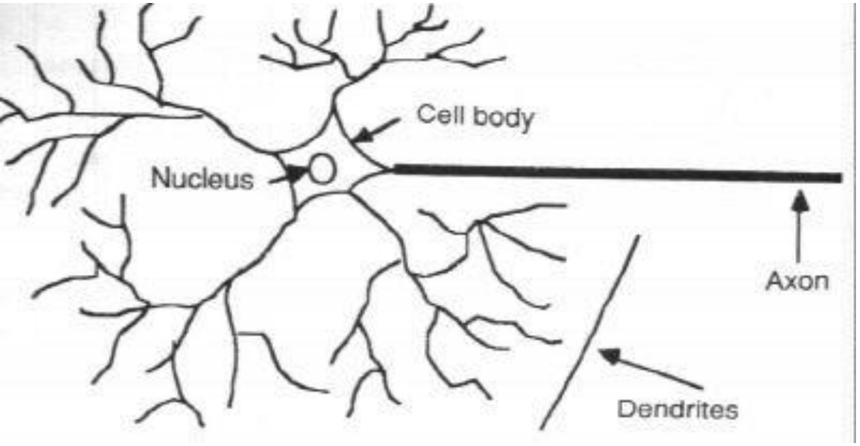
Brief History of ANN

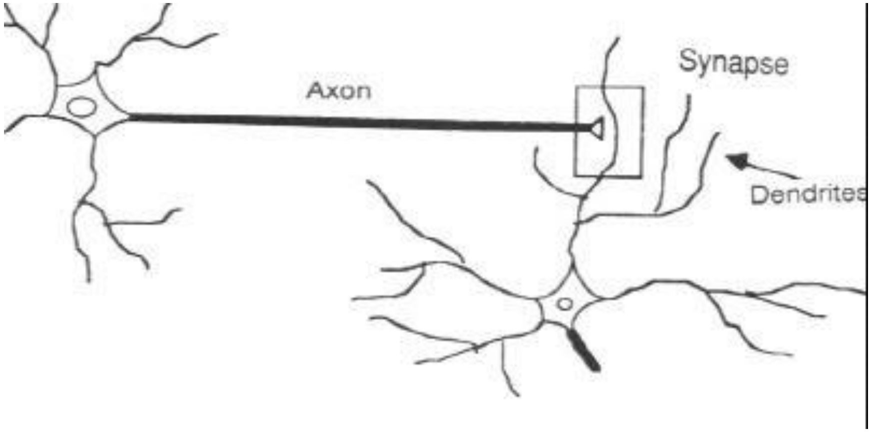


Ref: https://medium.com/coinmonks/neural-networks-bb11fb9a8266

Inspiration from neural science

- A **neuron** takes in several signals through **dendrites**, processes the signals and sends out a signal (spike of electrical activity) through the **axon**.
- The axon is split into several branches and connects to other dendrites through **synapse**, forming the biological neural network.

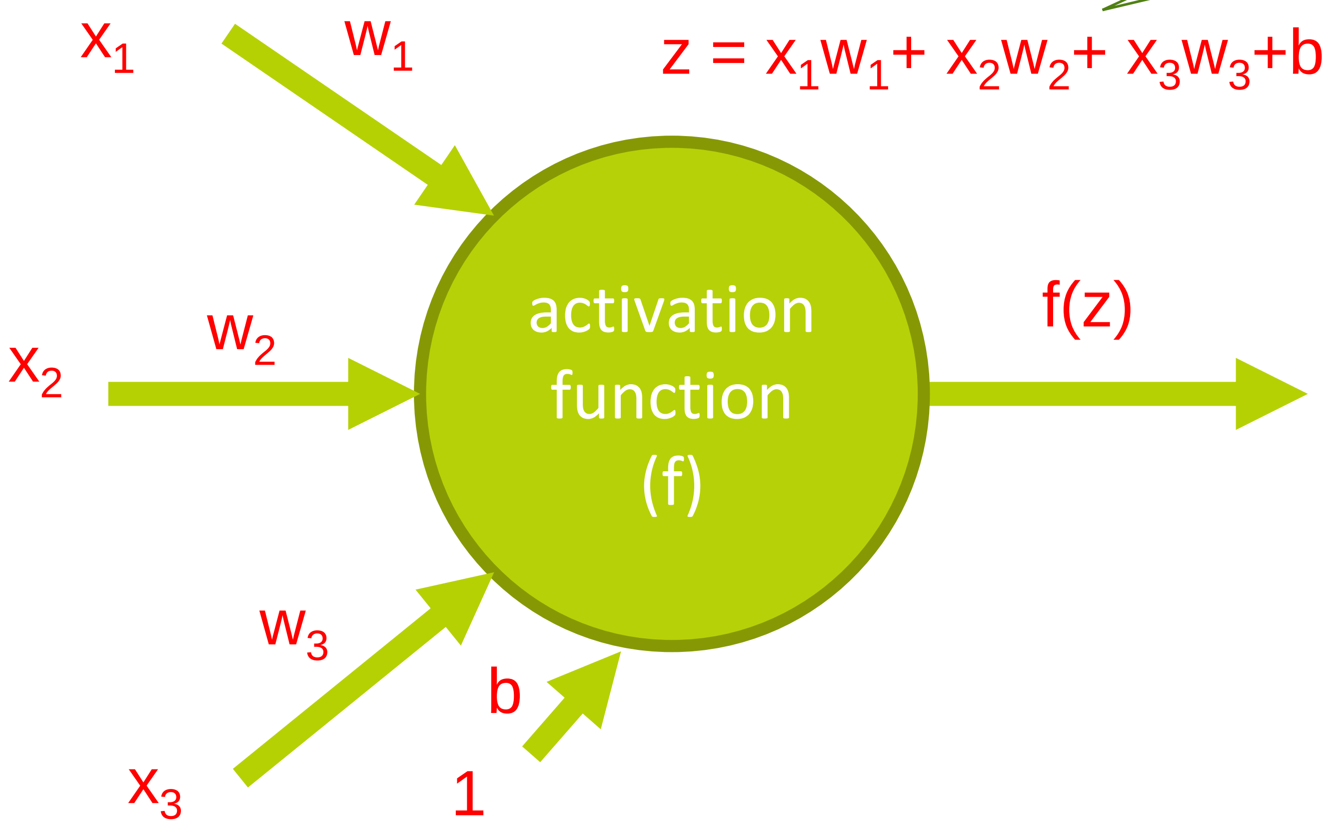




A Neural Node

This is similar to the formula of Linear Regression

$z = x_1w_1 + x_2w_2 + x_3w_3 + b$



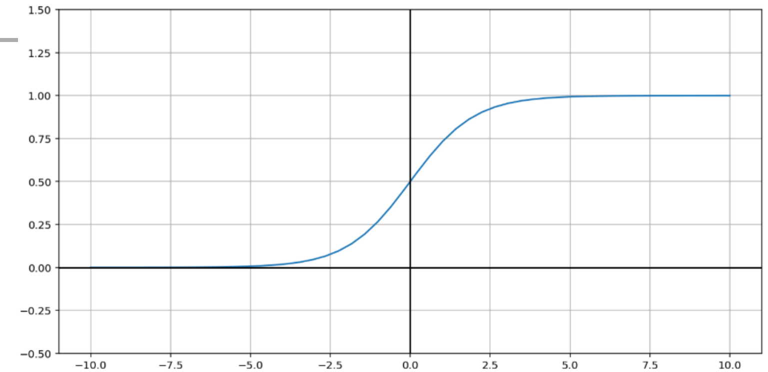
Activation function

- Taking inspiration from the biological neuron, there is a need to '***threshold***' the output signal. Such function is known as ***Activation Function***.
- Most commonly used function includes **Sigmoid**, **Tanh** and **Relu**

Activation function

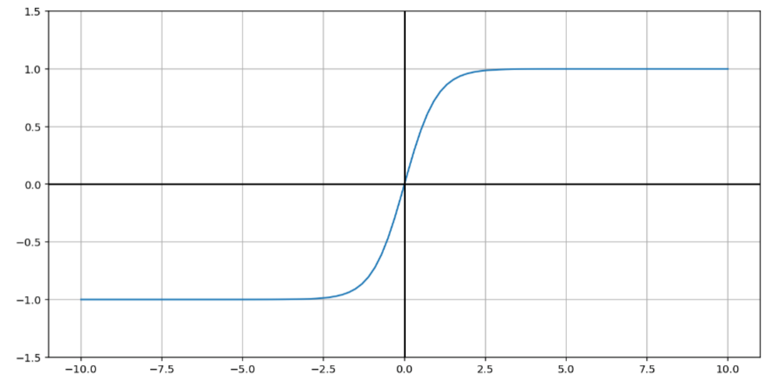
- **Sigmoid (σ)**
  - Limit to 0 to 1

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



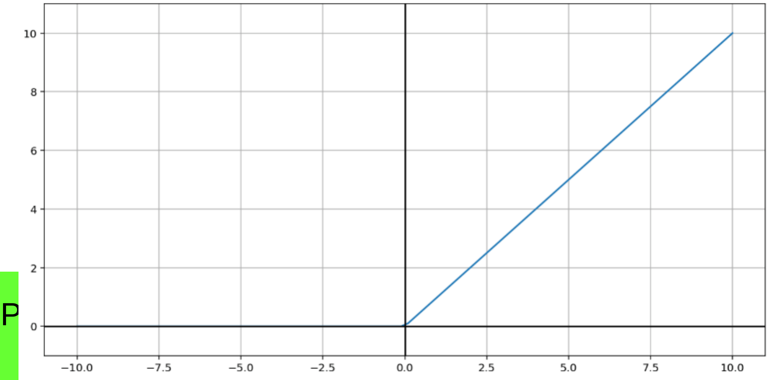
- **Tanh – Hyperbolic tangent**
  - Limit to -1 and 1

$$tanh(z) = \frac{\sinh(z)}{\cosh(z)} = \frac{e^{2x} - 1}{e^{2x} + 1}$$



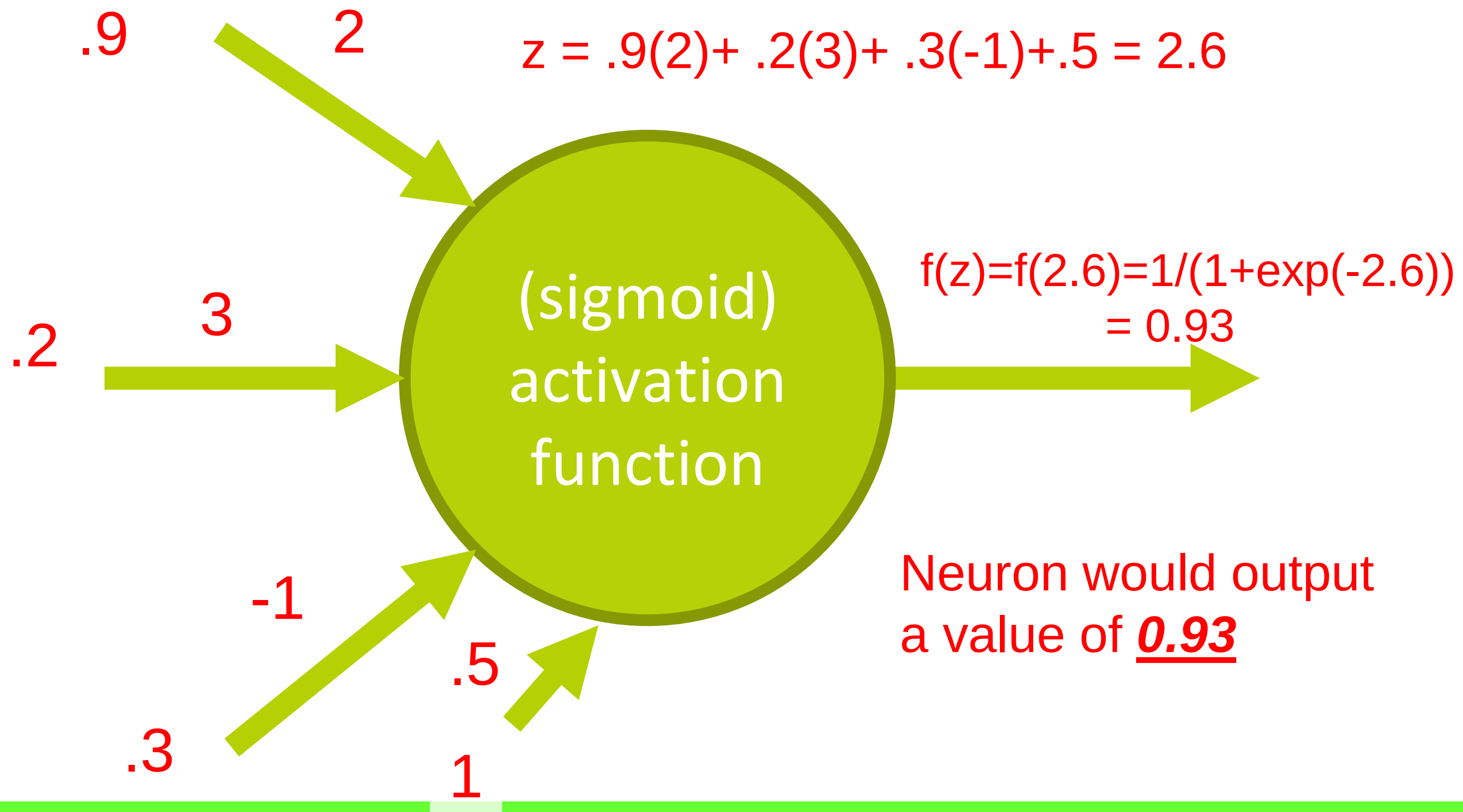
- **Relu – Rectified Linear Unit**
  - linear positive values
  - Ignore negative value

$$ReLU(z) = \begin{cases} 0, & z < 0 \\ z, & z \geq 0 \end{cases}$$



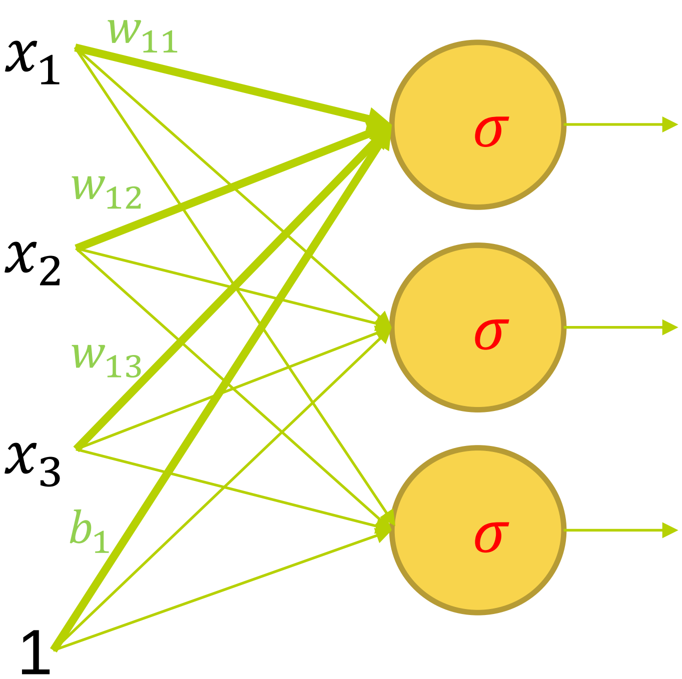
An example of a node

- When Sigmoid is used, the output equation of a neuron is similar to the *Logistic Regression.* Hence, a neuron **node** is also interpreted as a **classifier**.



Forming a hidden layer

- **Placing several nodes side by side, a layer is formed**



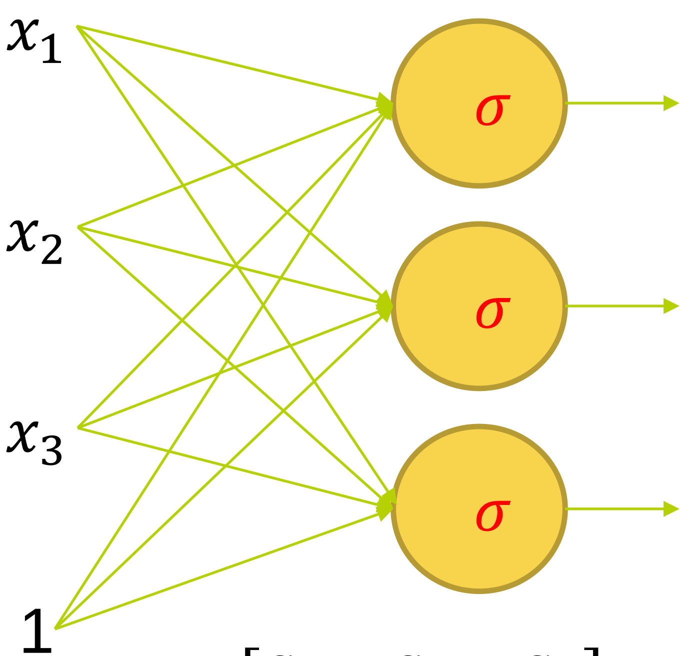
$$a_1 = \sigma(x_1 w_{11} + x_2 w_{12} + x_3 w_{13} + b_1)$$

$$a_2 = \sigma(x_1 w_{21} + x_2 w_{22} + x_3 w_{23} + b_2)$$

$$a_3 = \sigma(x_1 w_{31} + x_2 w_{32} + x_3 w_{33} + b_3)$$

Forming a hidden layer

- **Putting the equations together as a matrix**



$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} \sigma(x_1 w_{11} + x_2\, w_{12} + x_3 w_{13} + b_1) \\ \sigma(x_1 w_{21} + x_2\, w_{22} + x_3 w_{23} + b_2) \\ \sigma(x_1 w_{31} + x_2\, w_{32} + x_3 w_{33} + b_3) \end{bmatrix}$$

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \sigma \begin{bmatrix} w_{11} & w_{12} & w_{13} & b_1 \\ w_{21} & w_{22} & w_{23} & b_2 \\ w_{31} & w_{32} & w_{33} & b_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} a_1 & a_3 & a_3 \end{bmatrix} = \sigma \begin{bmatrix} x_1 & x_2 & x_3 & 1 \end{bmatrix} \begin{bmatrix} w_{11} & w_{12} & w_{13} & b_1 \\ w_{21} & w_{22} & w_{23} & b_2 \\ w_{31} & w_{32} & w_{33} & b_3 \end{bmatrix}^T$$

Activation function may change

$$a^{(i)} = \sigma\left(z^{(i)}\right) \; where \; z^{(i)} = x^{(i)} W^{(i)}$$

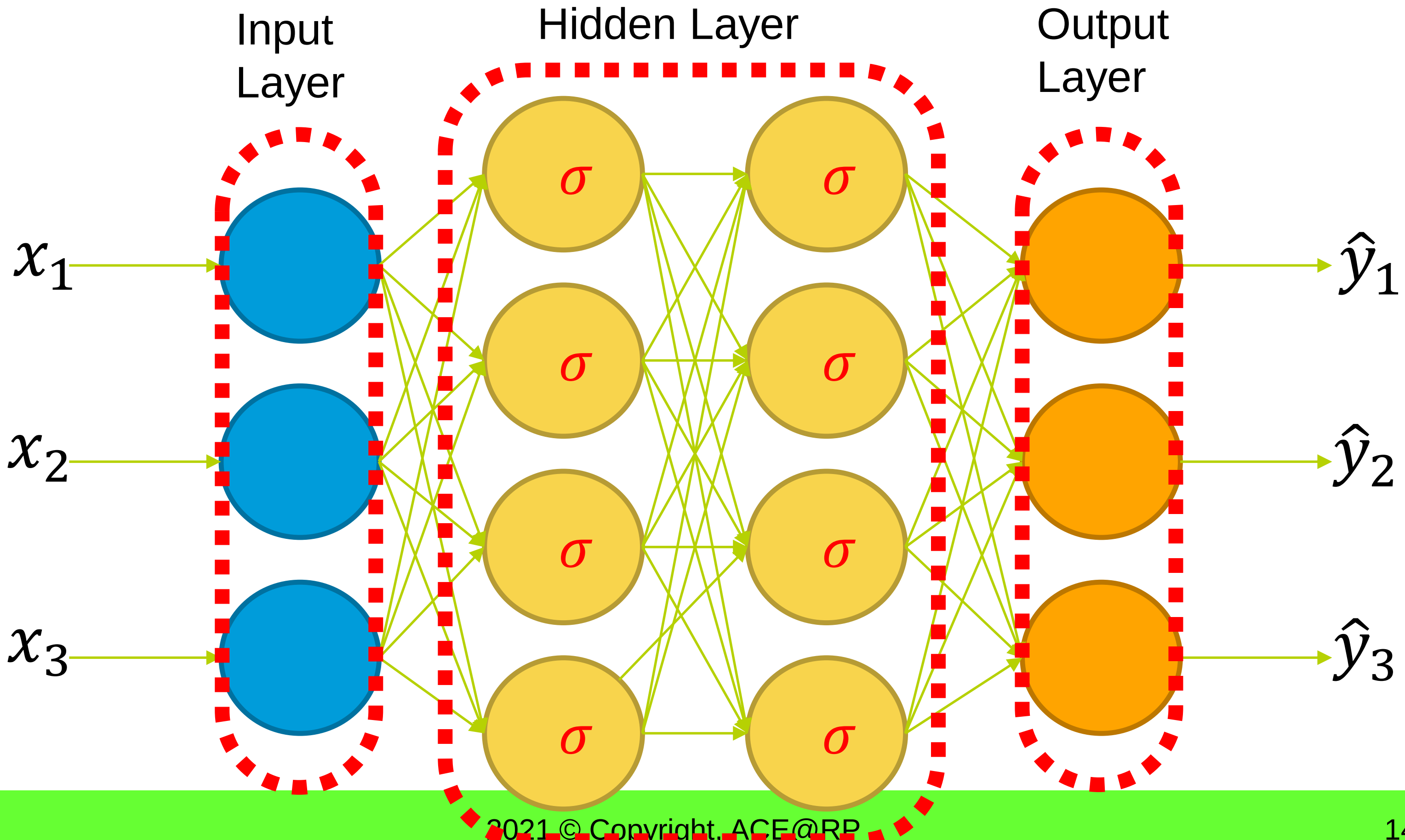
Input and Output Layer

- Input layer allows the features (or data) to enters the neural network.
  - The number of input nodes must be equal to the number of features
  - The output of the layer is equal to the feature values

- Output layer produces the predicted labels.
  - The number of output nodes must be equal to the number of labels
  - The output of output layer is calculated similar to the hidden layer
  - It has an additional activation function called **Softmax.**

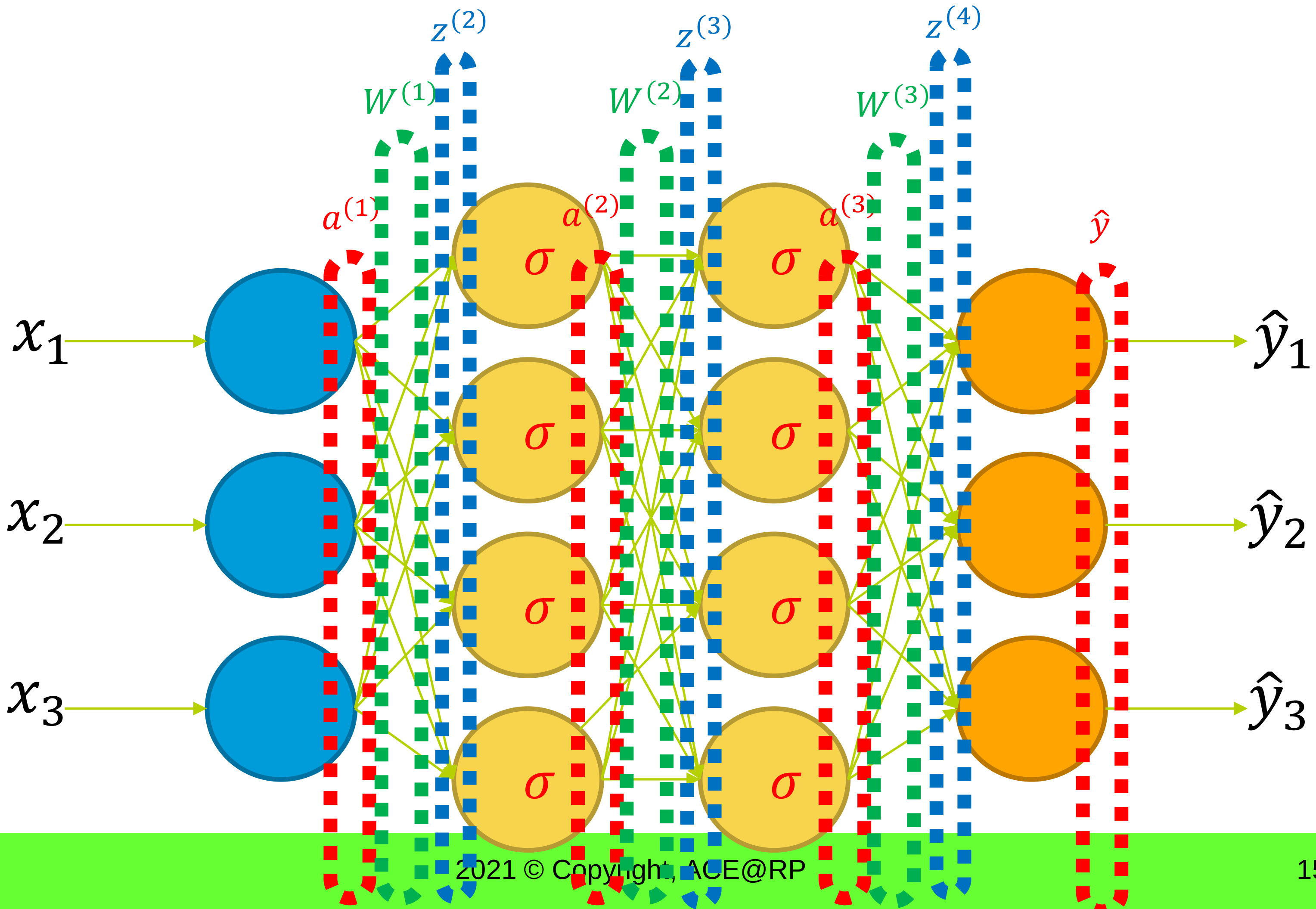
Forming a network

- One input layer → zero to many hidden layer(s) → one output layer
- The bias is normally not drawn or shown in a network diagram



Feedforward

- **Defining the values**



Feedforward

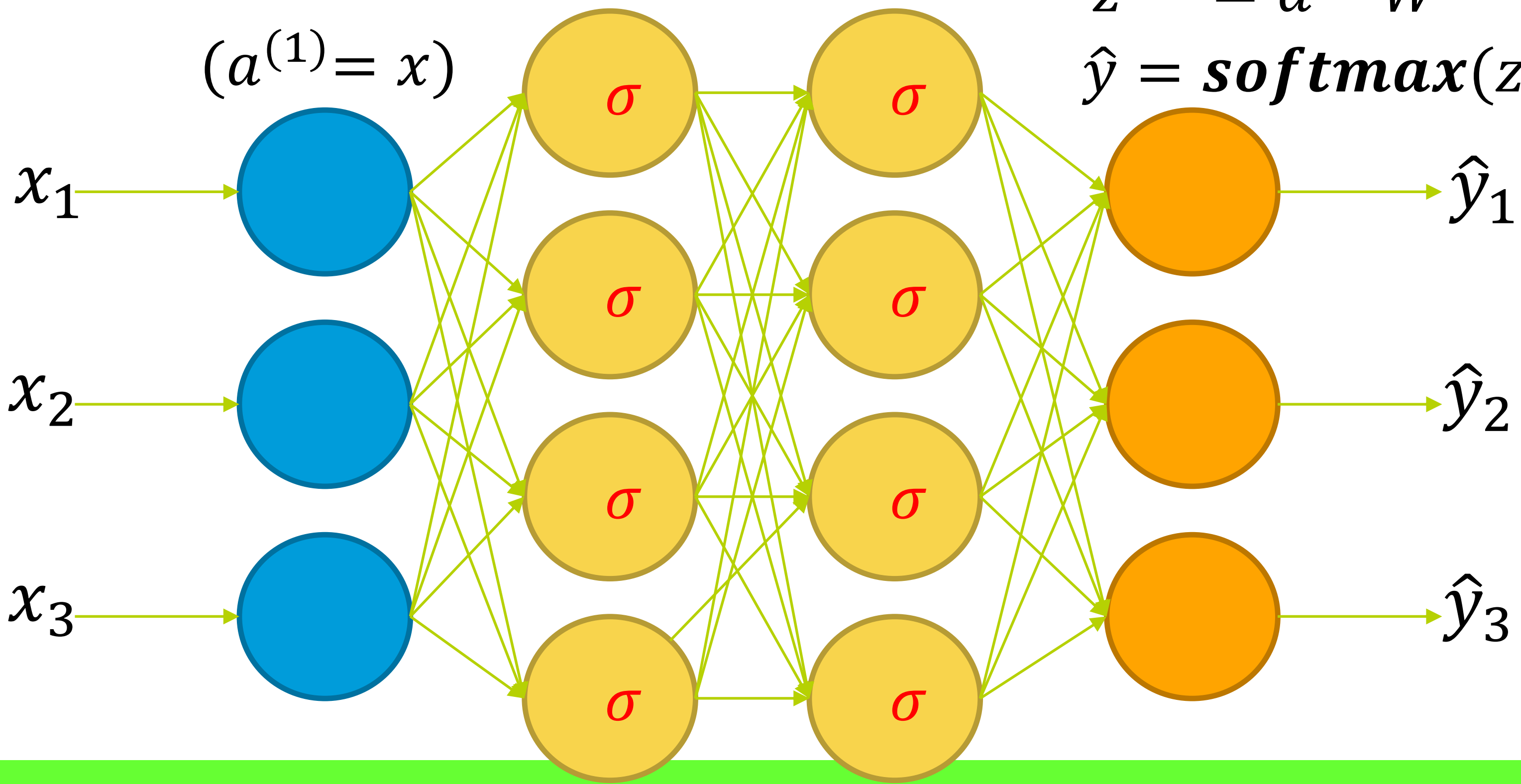
- **Calculate the output**

$$z^{(2)} = a^{(1)}W^{(1)} \qquad z^{(3)} = a^{(2)}W^{(2)}$$

$$a^{(2)} = \sigma(z^{(2)}) \qquad a^{(3)} = \sigma(z^{(3)})$$

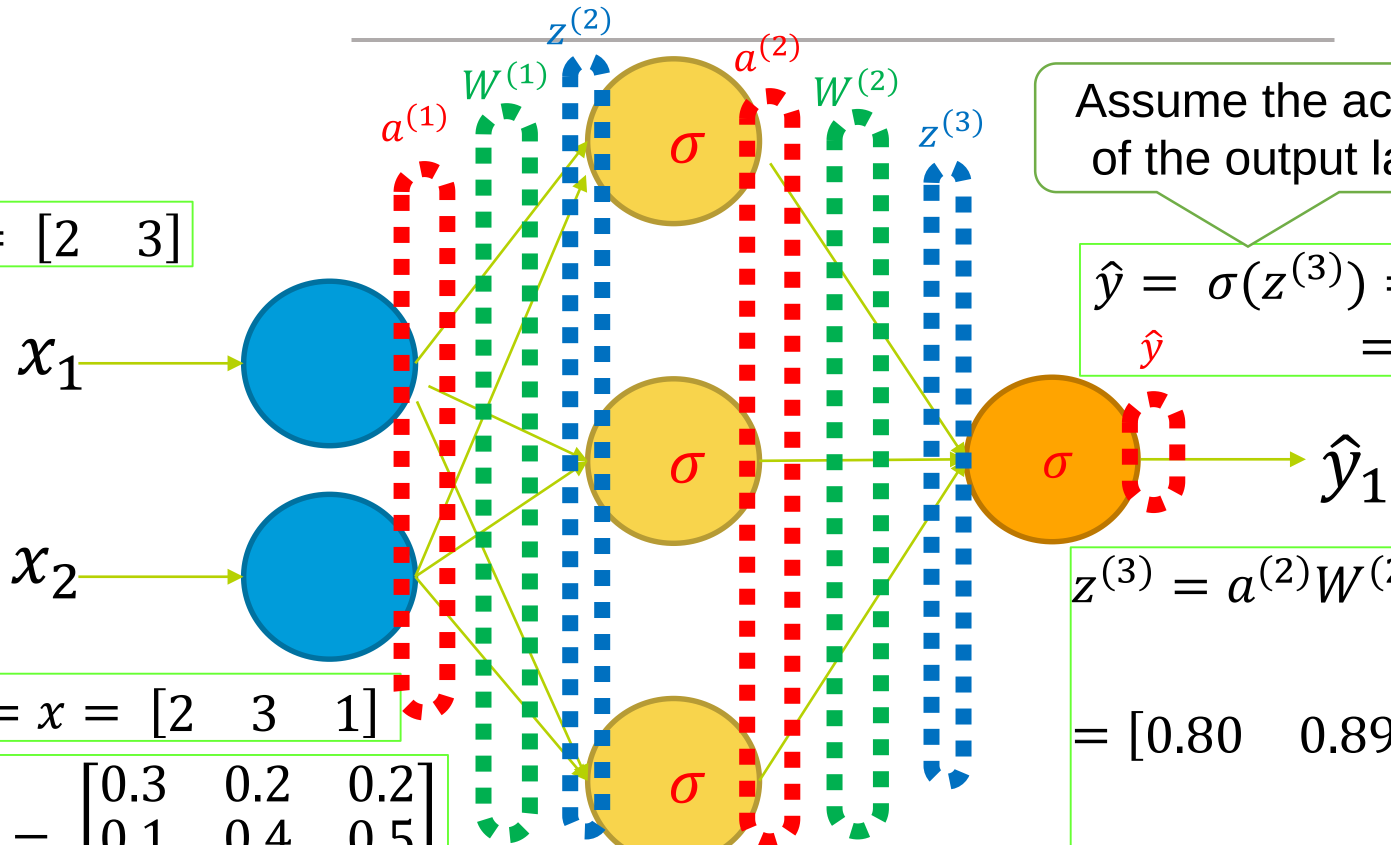
$$z^{(4)} = a^{(3)}W^{(3)}$$

$$\hat{y} = \boldsymbol{softmax}(z^{(4)})$$



Feedforward example

$x = [2 \quad 3]$



Assume the activation function of the output layer is Sigmoid

$$\hat{y} = \sigma(z^{(3)}) = \sigma[1.022] = [0.735]$$

$$a^{(1)} = x = [2 \quad 3 \quad 1]$$

$$W^{(1)} = \begin{bmatrix} 0.3 & 0.2 & 0.2 \\ 0.1 & 0.4 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix}$$

$$z^{(2)} = a^{(1)} W^{(1)} = [2 \quad 3 \quad 1] \begin{bmatrix} 0.3 & 0.2 & 0.2 \\ 0.1 & 0.4 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix} = [1.4 \quad 2.1 \quad 2.4]$$

$$a^{(2)} = \sigma\left(z^{(2)}\right) = \sigma[1.4 \quad 2.1 \quad 2.4] = [0.80 \quad 0.89 \quad 0.92]$$

$$W^{(2)} = \begin{bmatrix} 0.2 \\ 0.2 \\ 0.2 \\ 0.5 \end{bmatrix}$$

$$z^{(3)} = a^{(2)} W^{(2)} = [0.80 \quad 0.89 \quad 0.92 \quad 1] \begin{bmatrix} 0.2 \\ 0.2 \\ 0.2 \\ 0.5 \end{bmatrix} = [1.022]$$

Feedforward example

- **Multiple inputs can be calculated in just *one* feedforward pass. The number of inputs is known as Batch Size.**

$$x = \begin{bmatrix} 2 & 3 \\ 2 & 4 \\ \vdots & \vdots \\ 1 & 4 \\ 9 & 4 \end{bmatrix} \quad \cdots$$

$$z^{(2)} = a^{(1)} W^{(1)}$$

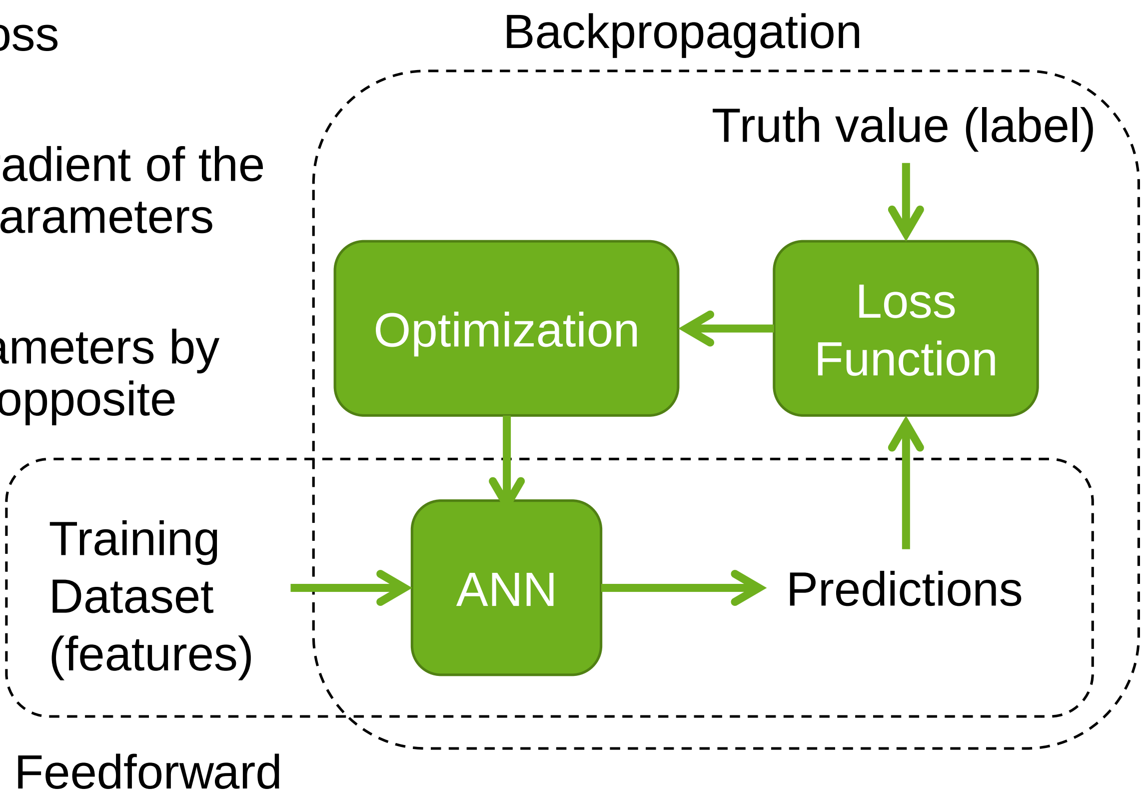
$$= \begin{bmatrix} 2 & 3 & 1 \\ 2 & 4 & 1 \\ \vdots & \vdots & \vdots \\ 1 & 4 & 1 \\ 9 & 4 & 1 \end{bmatrix} \begin{bmatrix} 0.3 & 0.2 & 0.2 \\ 0.1 & 0.4 & 0.5 \\ 0.5 & 0.5 & 0.5 \end{bmatrix}$$

$$= \begin{bmatrix} 1.4 & 2.1 & 2.4 \\ 1.5 & 2.5 & 2.9 \\ \vdots & \vdots & \vdots \\ 1.2 & 2.3 & 2.7 \\ 3.6 & 3.9 & 4.3 \end{bmatrix}$$

$$\cdots \quad \hat{y} = \begin{bmatrix} 0.735 \\ 0.738 \\ \vdots \\ 0.736 \\ 0.748 \end{bmatrix}$$

Training of weights

- **Work flow**
  - Step 1: Make a prediction (feedforward)
  - Step 2: Calculate Loss
  - Step 3: Calculate gradient of the loss function w.r.t. parameters
  - Step 4: Update parameters by taking a step in the opposite direction
  - Step 5: Iterate



Training of weights

- **Loss Function, $J(y, \hat{y})$,** calculate how **poorly** our model is performing by comparing what the **model is predicting** with the **actual value** it is supposed to output.
- Regression Loss – the model is predicting a continuous value
  - Mean Square Error
  - Mean Absolute Error

$$Loss = \frac{1}{n} * \sum_{i=0}^{n} (Y_i - Y_{pred_i})^2$$

- Classification Loss – the model is predicting a discrete value or specific class.
  - Binary Cross Entropy
  - Categorial Cross Entropy

$$Loss = (Y)(-log(Y_{pred})) + (1 - Y)(-log(1 - Y_{pred}))$$

Ref: https://deeplearningdemystified.com/article/fdl-3

Training of weights

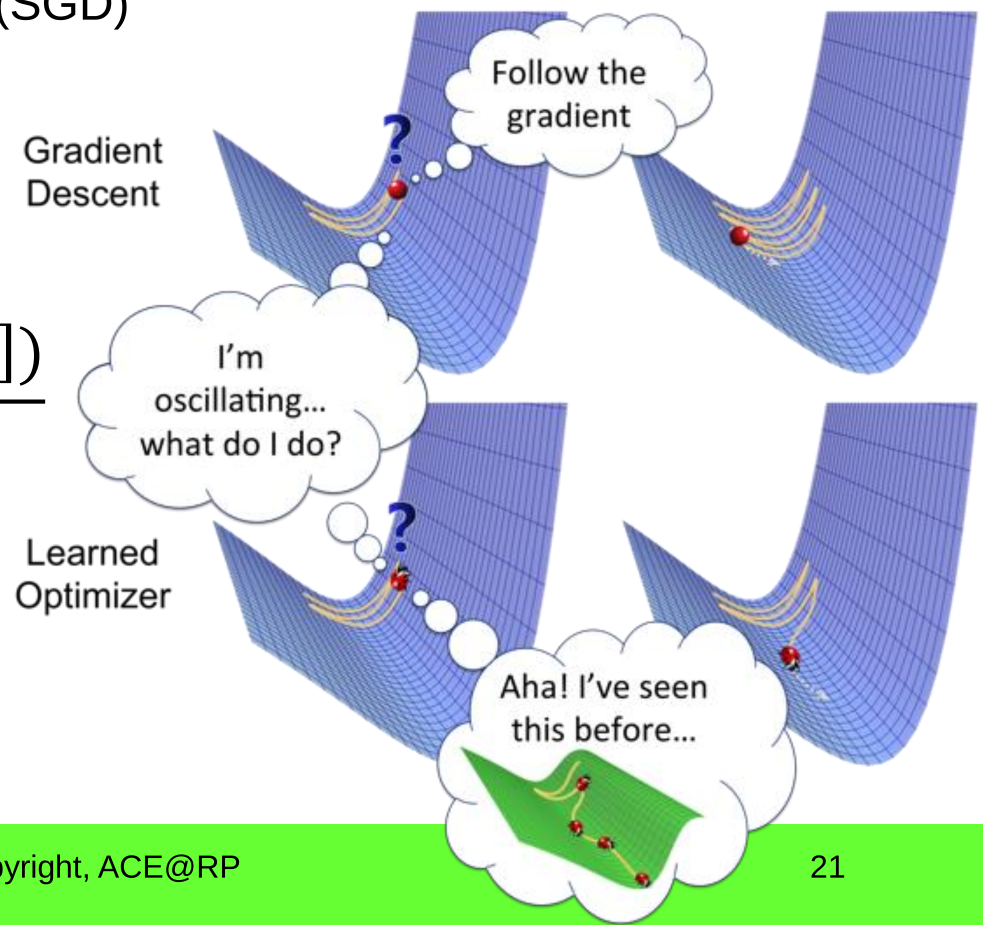
- **Optimization -** Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rate in order to reduce the losses.
  - Stochastic Gradient Descent (SGD)

$$w = w - \alpha \frac{\partial J(y, \hat{y})}{\partial w}$$

  - Mini-Batch Gradient Descent

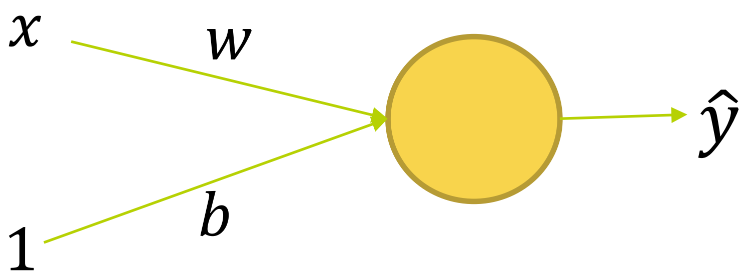
$$w = w - \alpha \frac{\partial J([y], [\hat{y}])}{\partial w}$$

  - Momentum
  - Adam
  - RMSProp



Ref: https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6

Training of weights (An Example)



Current Values:
$x = 2, y = 4$
$w = 0.2, b = 0.5$

- **Assume:**
  - Activation function = Linear
  - Loss function = MSE
  - Optimization = SGD (one sample used)
  - Learning rate = 0.1

- **Step 1: Make a prediction**

$$\hat{y} = xw + b$$
$$= 2 * 0.2 + 0.5$$
$$= 0.9$$

Training of weights (An Example)

- **Step 2: Calculate the Loss**

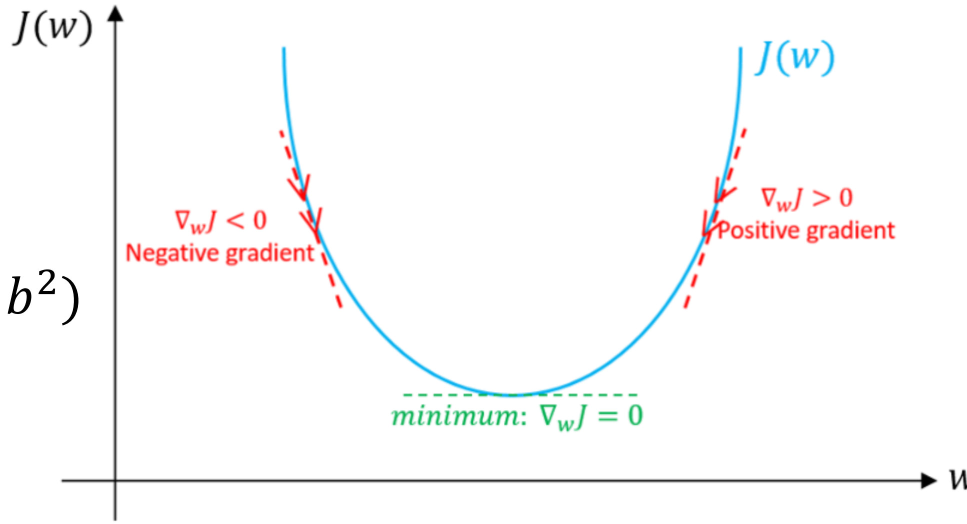
$$J(y,\hat{y}) = (y-\hat{y})^2$$
$$= (4 - 0.9)^2$$
$$= 9.61$$

$$J(y,\hat{y}) = (y-\hat{y})^2$$
$$= y^2 - 2y\hat{y} + \hat{y}^2$$
$$= y^2 - 2y(xw-b) + (xw-b)^2$$
$$= y^2 - 2xyw + 2by + x^2w^2 - 2bxw + b^2$$
$$= x^2w^2 - (2bx + 2xy)w + (y^2 + 2by + b^2)$$
$$or$$
$$= b^2 + (2y - 2xw)b + (y^2 - 2xyw + x^2w^2)$$

Training of weights (An Example)

- **Step 3: Calculate the gradient**

$$J = c_1 w^2 + c_2 w + c_3$$



$$J(y, \hat{y}) = (y - \hat{y})^2$$
$$= y^2 - 2y\hat{y} + \hat{y}^2$$
$$= y^2 - 2y(xw - b) + (xw - b)^2$$
$$= x^2 w^2 - (2bx + 2xy)w + (y^2 + 2by + b^2)$$

$$\frac{\partial J}{\partial w} = 2x^2 w - 2x(b + y)$$
$$= (2)(2 * 2)(0.2) - (2)(2)(0.5 + 4)$$
$$= 1.6 - 18$$
$$= -16.4$$

Training of weights (An Example)

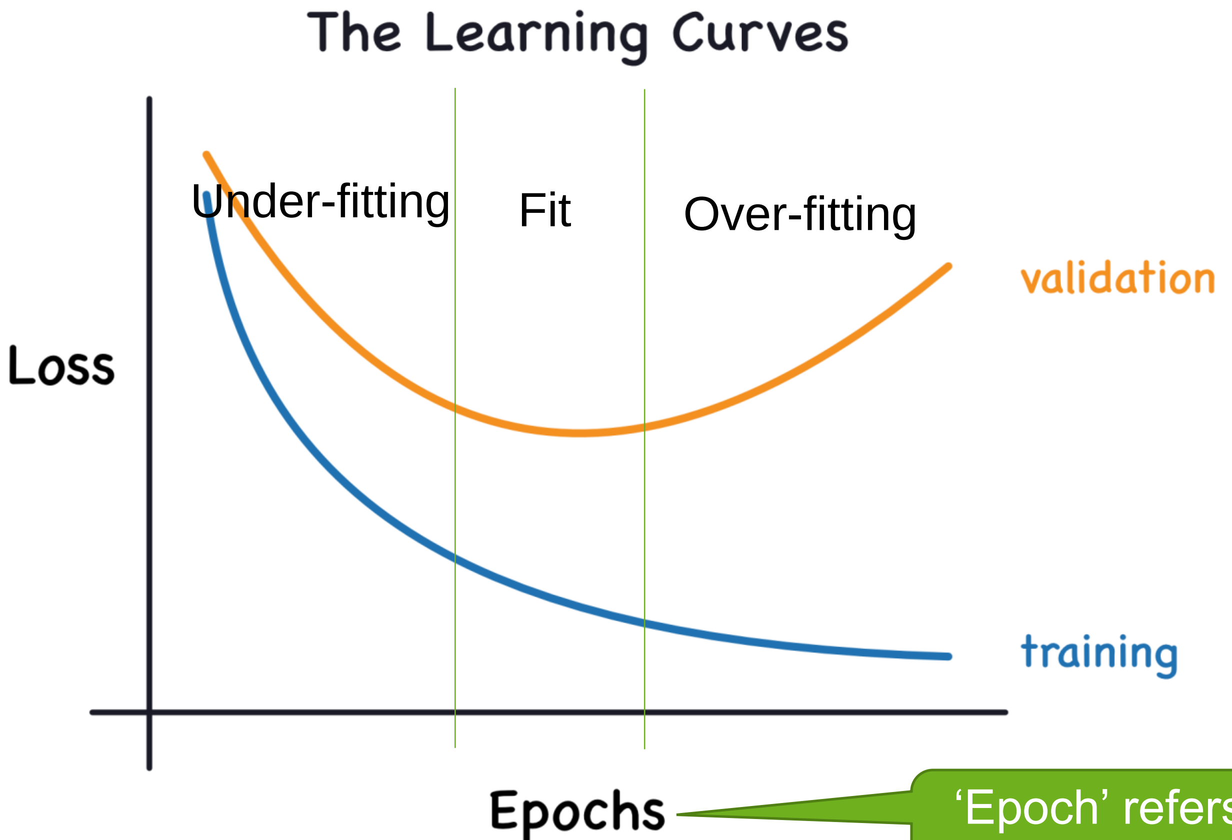
- **Step 4: Update the weights**

$$w = w - \alpha \frac{\partial J(y, \hat{y})}{\partial w}$$
$$= 0.2 - (0.1)(-16.4)$$
$$= 1.84$$

Try out the same approach to update the bias.

Training of weights (An Example)

- **Step 5: Iterate**
  - A learning curve will be produced



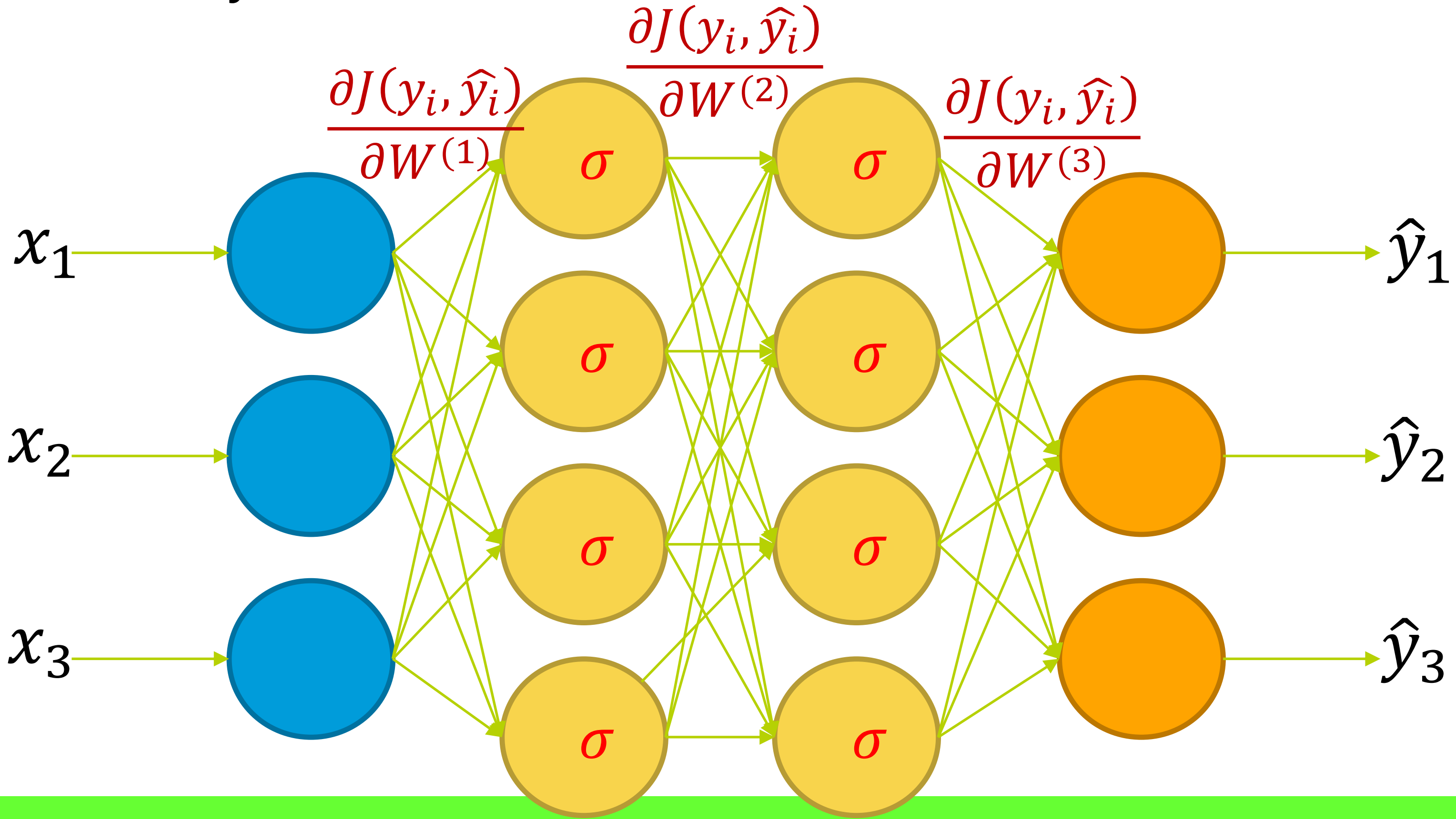
'Epoch' refers to **a single pass** of whole dataset.

Ref: https://www.kaggle.com/ryanholbrook/overfitting-and-underfitting

Backpropagation

- To train the weights of a neural network, we need to find the gradient of each weights.
- This is very difficult.

Want: $\frac{\partial J(y_i, \hat{y}_i)}{\partial W_k}$



Backpropagation

- Using **Chain Rule** and **Calculus**
- The calculated values can be “propagate back” from the higher layers to the lower layers.
- Although the formula is long, but it can be quickly calculated by the computer.

$$\frac{\partial J(y,\hat{y})}{\partial W^{(3)}} = \underbrace{(\hat{y}-y)} \cdot a^{(3)}$$

$$\frac{\partial J(y,\hat{y})}{\partial W^{(2)}} = \underbrace{\boxed{(\hat{y}-y)} \cdot W^{(3)} \cdot \sigma'\left(z^{(3)}\right)} \cdot a^{(2)}$$

$$\frac{\partial J(y,\hat{y})}{\partial W^{(1)}} = \underbrace{\boxed{(\hat{y}-y) \cdot W^{(3)} \cdot \sigma'\left(z^{(3)}\right)} \cdot W^{(2)} \cdot \sigma'\left(z^{(2)}\right)} \cdot X$$

Where: $\sigma'(z) = \sigma(z)(1-\sigma(z))$

Backpropagation (optional)

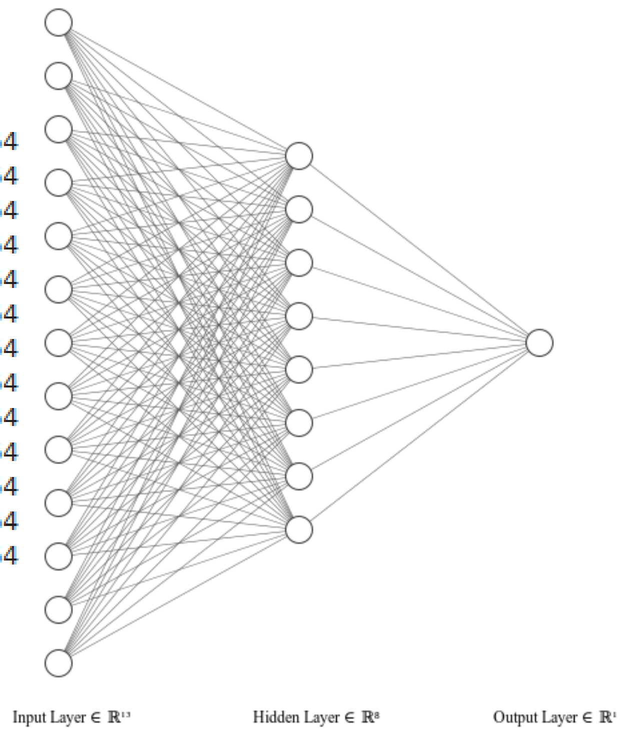
- **Explain by YouTube channel - 3Blue1Brown**
  - Check out the series on Neural Network.
  - https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

https://www.youtube.com/watch?v=tIeHLnjs5U8

15 Mins Break

Activity 4 – Building NN with Python

```
age                        float64
sex                        float64
chest_pain                 float64
resting_blood_pressure     float64
serum_cholestoral          float64
fasting_blood_sugar        float64
resting_ecg_results        float64
max_heart_rate_achieved    float64
exercise_induced_angina    float64
oldpeak                    float64
slope of the peak          float64
num_of_major_vessels       float64
thal                       float64
```




```
heart_disease              int64
```

Exercises:
- Which is better, increase the number of nodes or increase the number of iterations?

Step 1:
Watch and listen to the instructor's demonstration

15 mins

Step 2:
Work through the activities

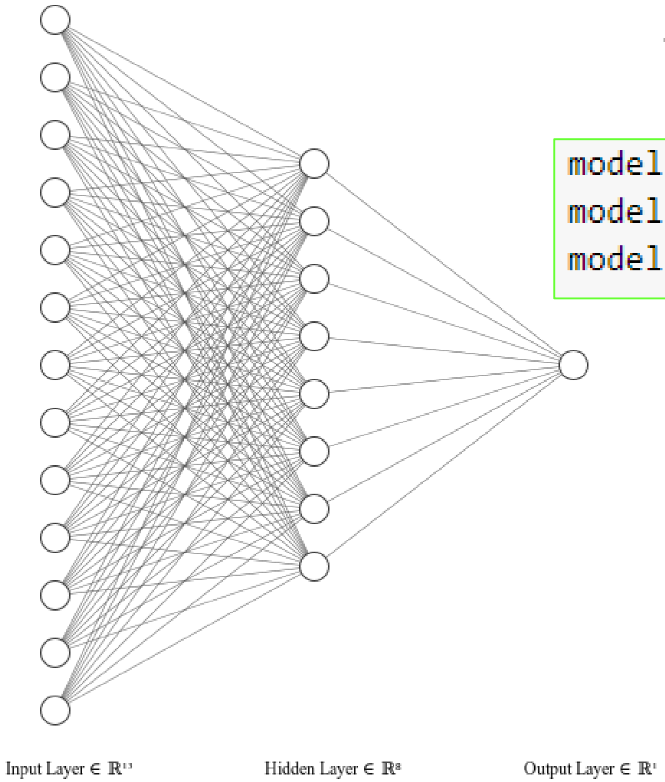
Individual Activity

20 mins

Keras

- **Keras is one of the 5 most popular deep learning framework**
- **Built on Tensorflow 2.0 which allows Keras to run on GPUs as well as TPUs.**
- **It is very easy to use and create deep neural networks**
- **Homepage: https://keras.io/**
- **API reference: https://keras.io/api/**

Activity 5 – Building NN with Keras




```
model = Sequential()
model.add(Dense(8, input_shape=(13,), activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

Exercises:
- Add a hidden layer with 10 nodes
- Change the optimizer

Step 1:
Watch and listen to the instructor's demonstration

10 mins

Step 2:
Work through the activities

Individual Activity

15 mins

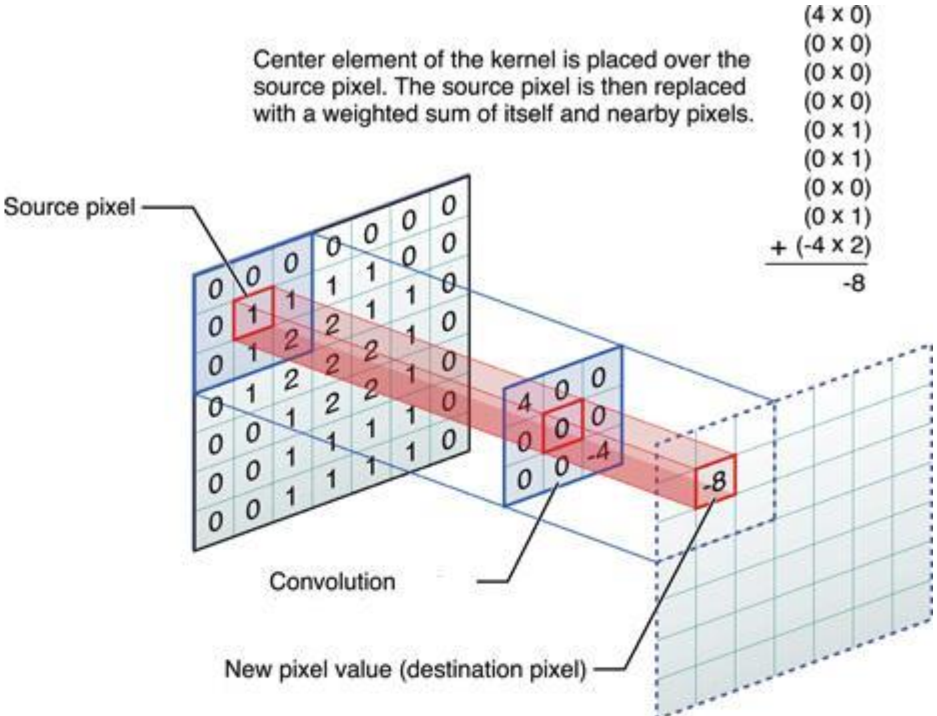
60 mins Lunch Break

Lunch break xx:xx – yy:yy

Image Convolution

Image Convolution

- In image processing, convolution is the process of transforming an image by applying a kernel over each pixel and its local neighbors across the entire image. The kernel is a matrix of values whose size and values determine the transformation effect of the convolution process.



https://medium.com/@bdhuma/6-basic-things-to-know-about-convolution-daef5e1bc411

Types of Kernel / Filters

- **Basic image pre-processing or enhancement**
  - Sharpening, Brightness, Blurring, etc
  - https://medium.com/@bdhuma/6-basic-things-to-know-about-convolution-daef5e1bc411
- **Feature extraction**
  - Line detector
  - edge detector

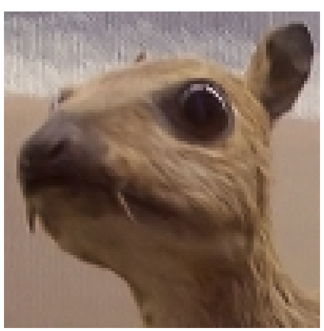
Vertical Line Detector

-1	2	-1
-1	2	-1
-1	2	-1

Horizontal Line Detector

-1	-1	-1
2	2	2
-1	-1	-1

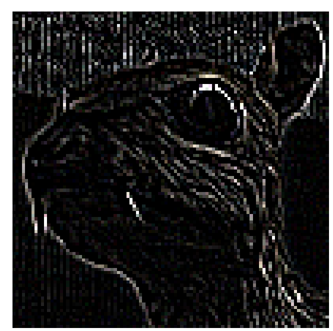
Input image



Convolution Kernel

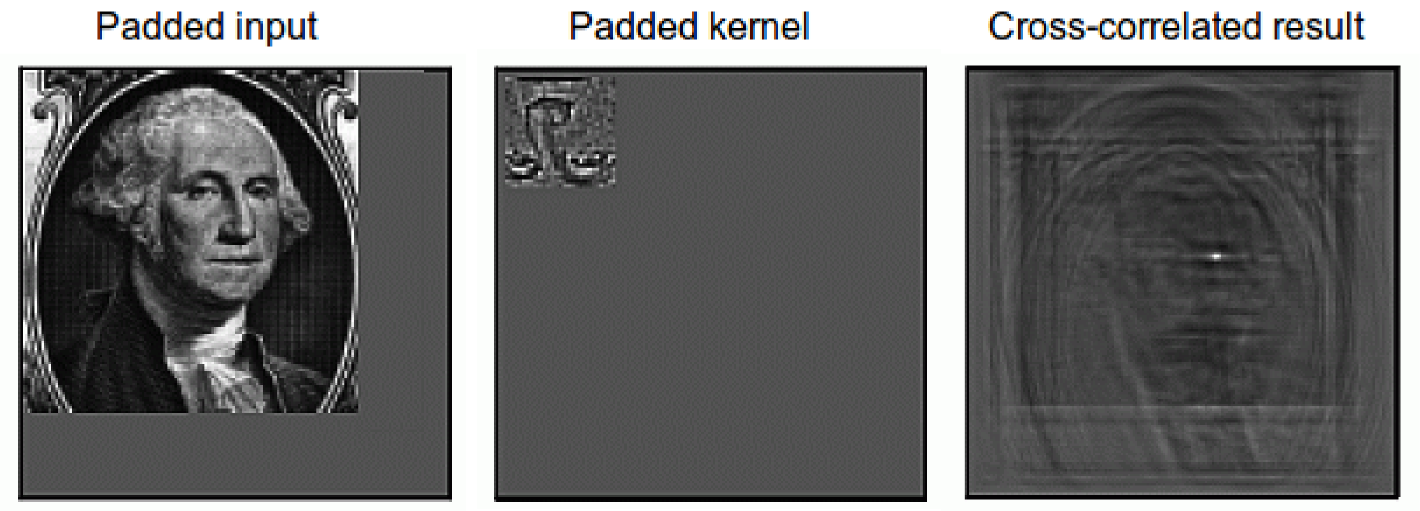
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Feature map



Types of Kernel / Filters

- **Pattern extraction**
  - Patterns, grids



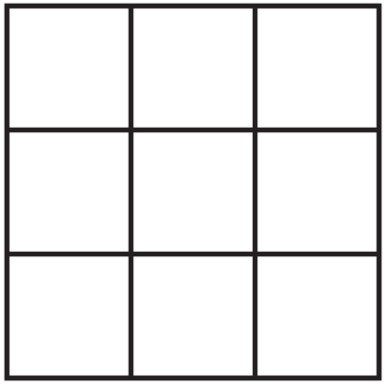
https://developer.nvidia.com/discover/convolution

- **Non-standard Kernels**
  - Transpose, Dilated, Deformable
  - https://towardsdatascience.com/types-of-convolution-kernels-simplified-f040cb307c37

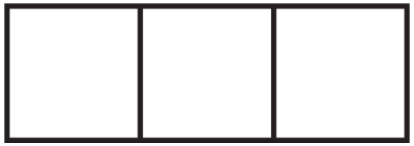
Grid Size

- **The number of pixels a kernel "sees" at once**
- **Typically use odd numbers so that there is a "center" pixel**
- **Kernel does not need to be square**

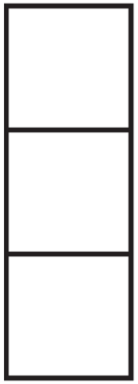
Height: 3, Width: 3



Height: 1, Width: 3



Height: 3, Width: 1



Padding

- **Using Kernels directly, there will be an "edge effect"**
- **Pixels near the edge will not be used as "center pixels" since there are not enough surrounding pixels**
- **Padding adds extra pixels around the frame**
- **So every pixel of the original image will be a center pixel as the kernel moves across the image**

0	0	0	0	0	0	0	0
0	3	3	4	4	7	0	0
0	9	7	6	5	8	2	0
0	6	5	5	6	9	2	0
0	7	1	3	2	7	8	0
0	0	3	7	1	8	3	0
0	4	0	4	3	2	2	0
0	0	0	0	0	0	0	0

$6 \times 6 \rightarrow 8 \times 8$

*

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

3×3

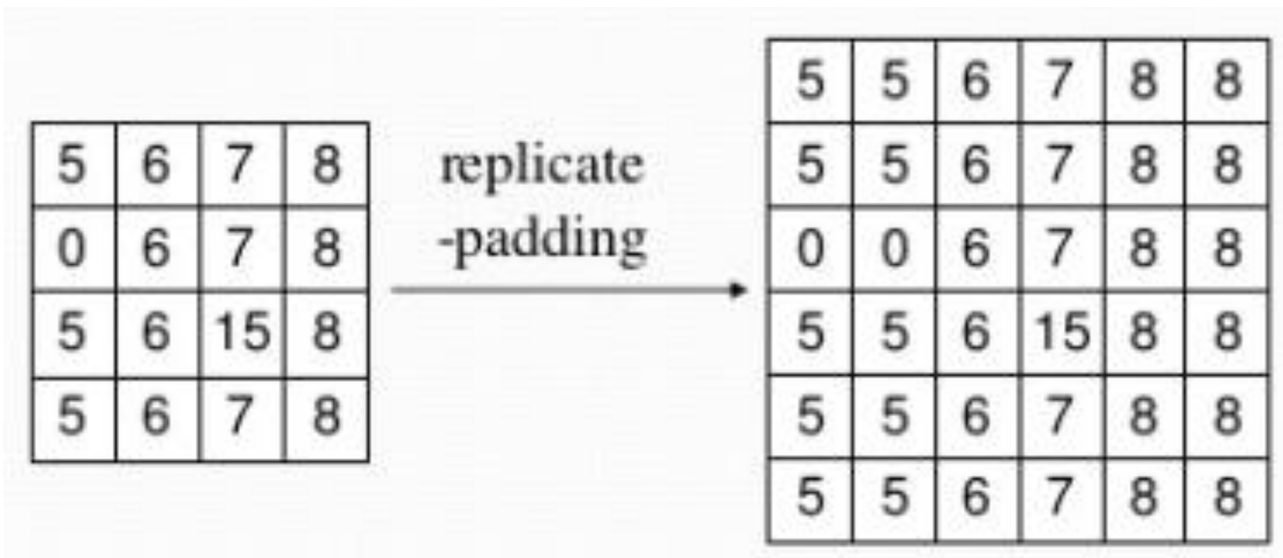
=

-10	-13	1			
-9	3	0			

6×6

Padding

- **Three form of padding**
  - Zero padding – Added pixels with a value of 0
  - One padding – Added pixels with a value of 1
  - Replicate padding – Added pixels with the edge values of the image

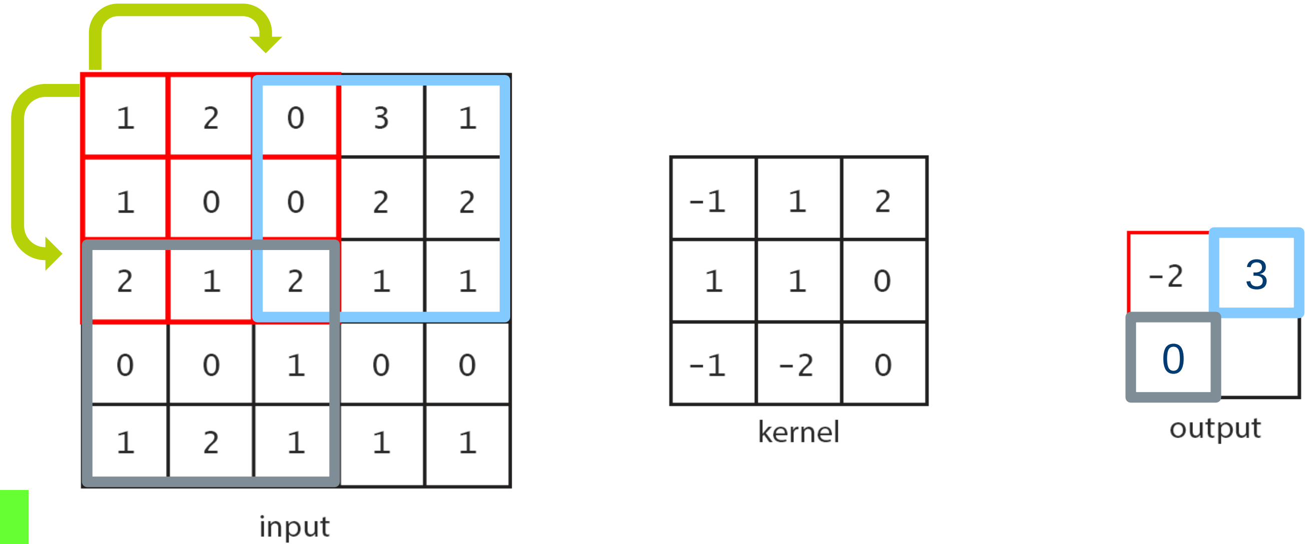


https://pt.slideshare.net/zertux/nonlinear-filtering

Stride

- **The "step size" as the kernel moves across the image**
- **Can be different for vertical and horizontal steps (but usually is the same value)**
- **When stride is greater than 1, it scales down the output dimension**

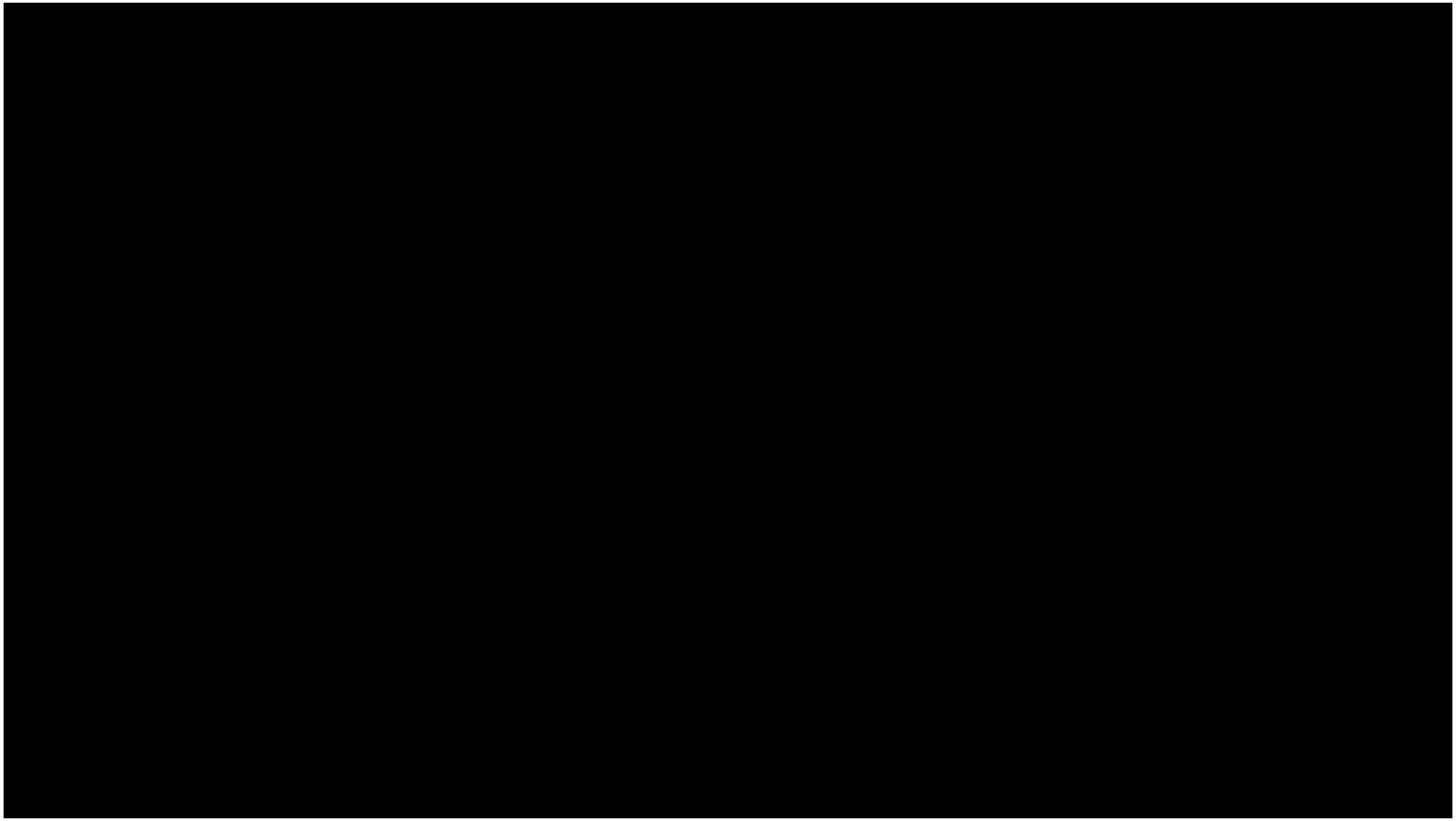
Stride 2 Example – No Padding



Depth

- **In images, each pixel may be represented by multiple values. This is also known as image mode.**
- **The number of values is referred to as “channels”**
  - RGB image – 3 channels
  - CMYK – 4 channels
- **Kernel of the same “depth” in order to convolute the input channels.**
- **The output of the convolution will have the same depth or channel.**

More on Image Convolution (optional)



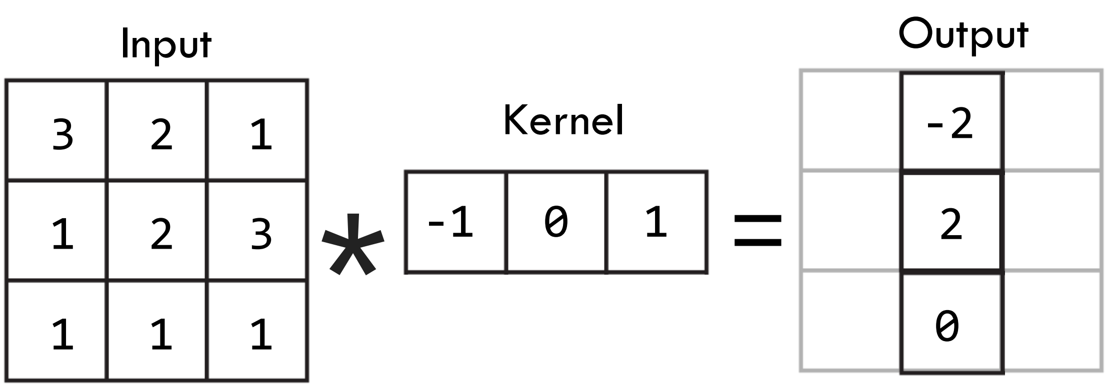
https://www.youtube.com/watch?v=8rrHTtUzyZA

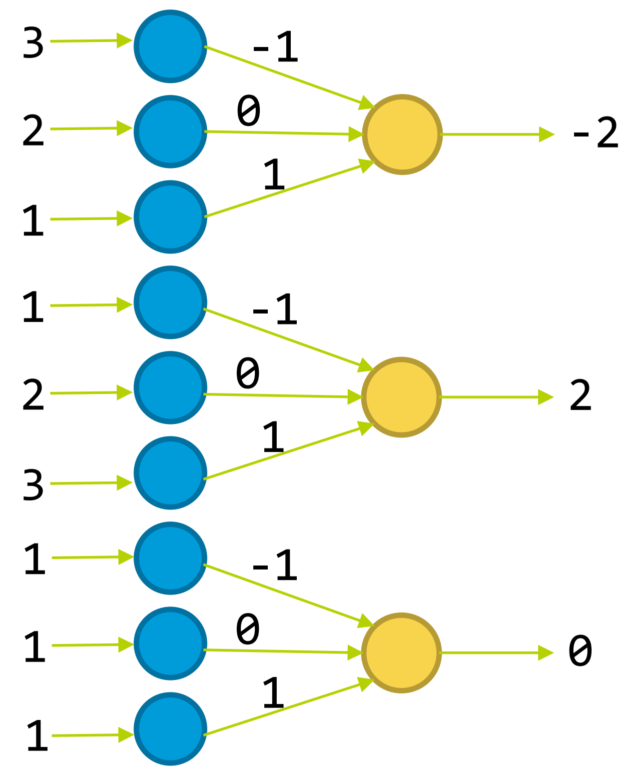
Convolution Neural Network

Basic Idea

- The image convolution formula is very similar to the output formula of neural network.

Image Convolution	Maps to	Neural Network
Input	➔	Input nodes
Kernel	➔	Weights
Output	➔	Output nodes



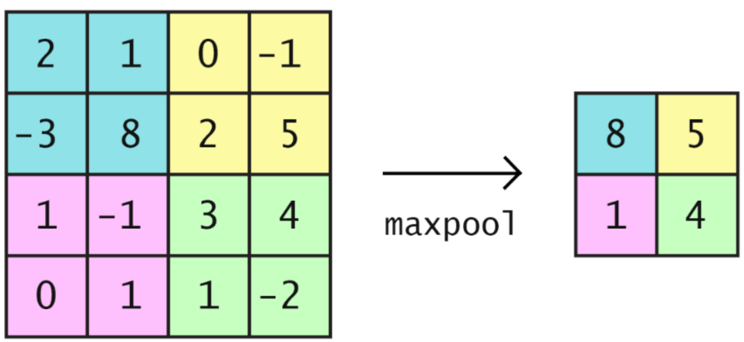


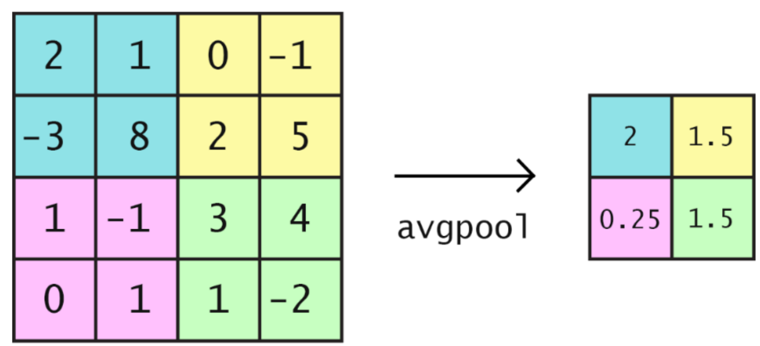
Basic Idea

- The network is not fully connected. Each output node (pixel) is only connected to its convolve input nodes (pixels)
- Same set of weights (kernels) across the entire image
- Training of the weights can be applied
  - The trained weights form the kernel that is best to analyse the images
- A ***convolution layer*** consists of n-number of kernels. Each kernel will convolve with the input image(s) to produce n-number of output images.
- The activation function of a convolution layer is Relu because (i) there is no requirement of 'thresholding' the convoluted output image, and (ii) there is no negative values in an image.

Other types of layers

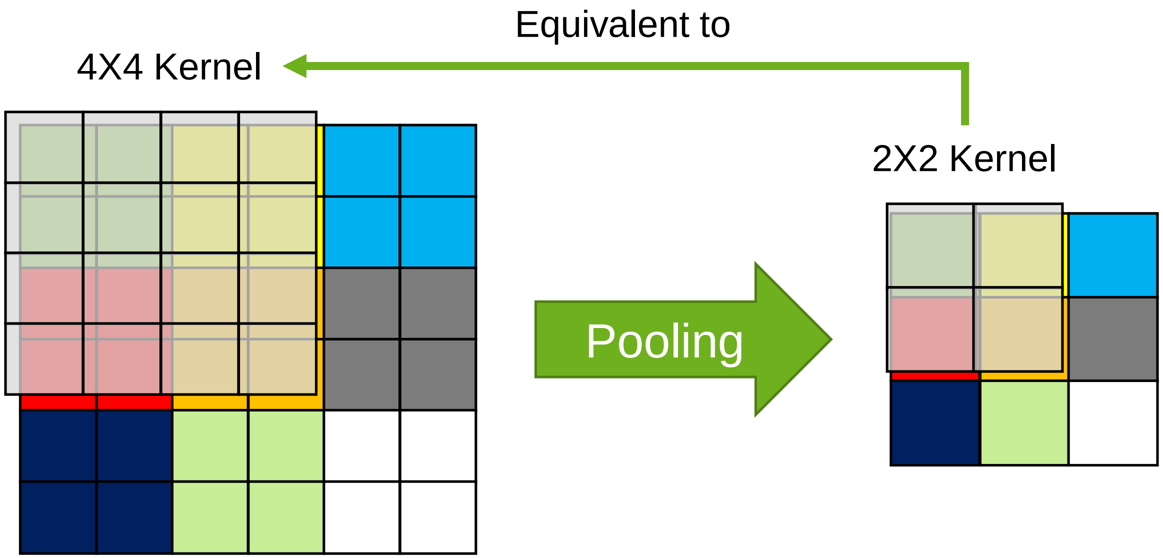
- **Pooling layer**
  - Shrinks the dimensions of an image (or reduce its size) by mapping a patch of pixels to one value.
    - Max-pooling
    - Average-pooling





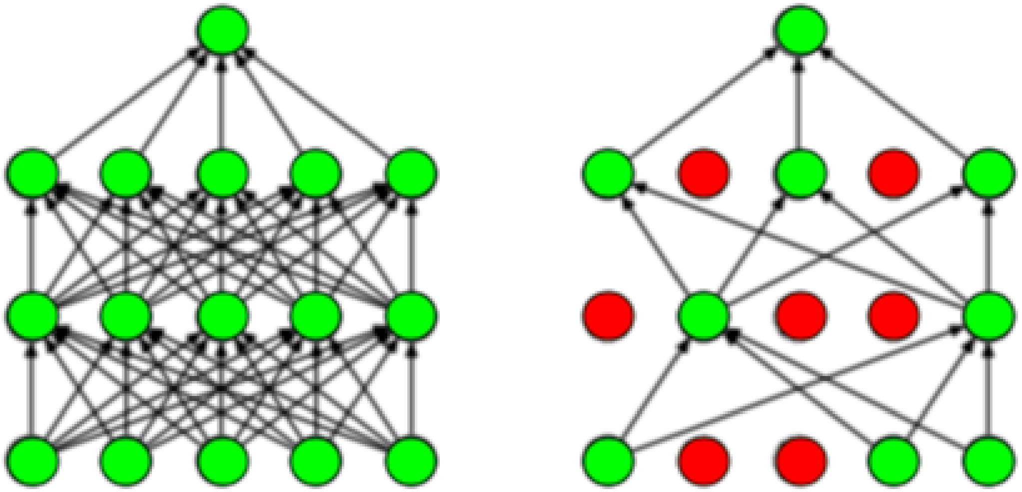
Other types of layers

- **Pooling layer**
  - To enables scale-invariant and shift-invariant
  - Simulate convolution of bigger dimension



Other types of layers

- **Dropout layer**
  - Randomly removes one or more nodes from a network
  - To prevent overfitting of the model
  - Dropout is ONLY performed during training

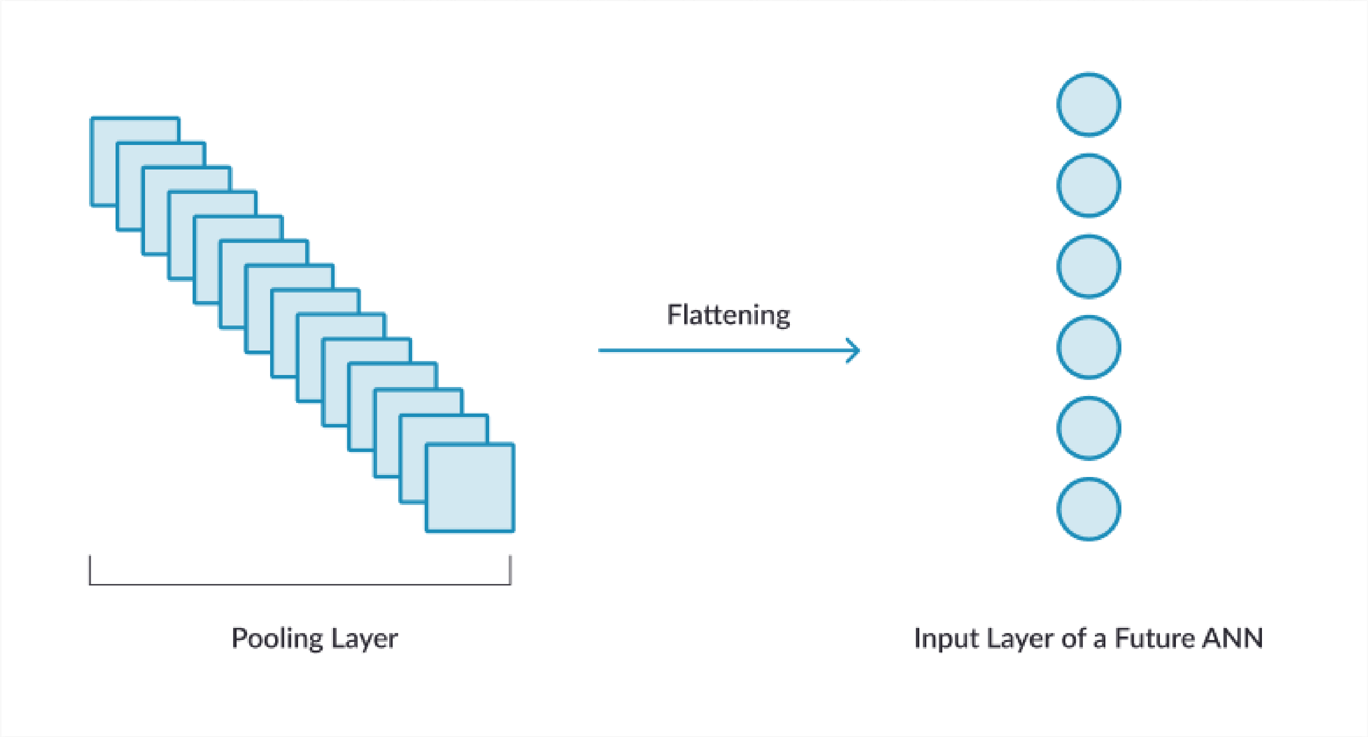


(a) Standard Neural Net (b) After applying dropout.

Ref: https://medium.com/analytics-vidhya/a-simple-introduction-to-dropout-regularization-with-code-5279489dda1e

Other types of layers

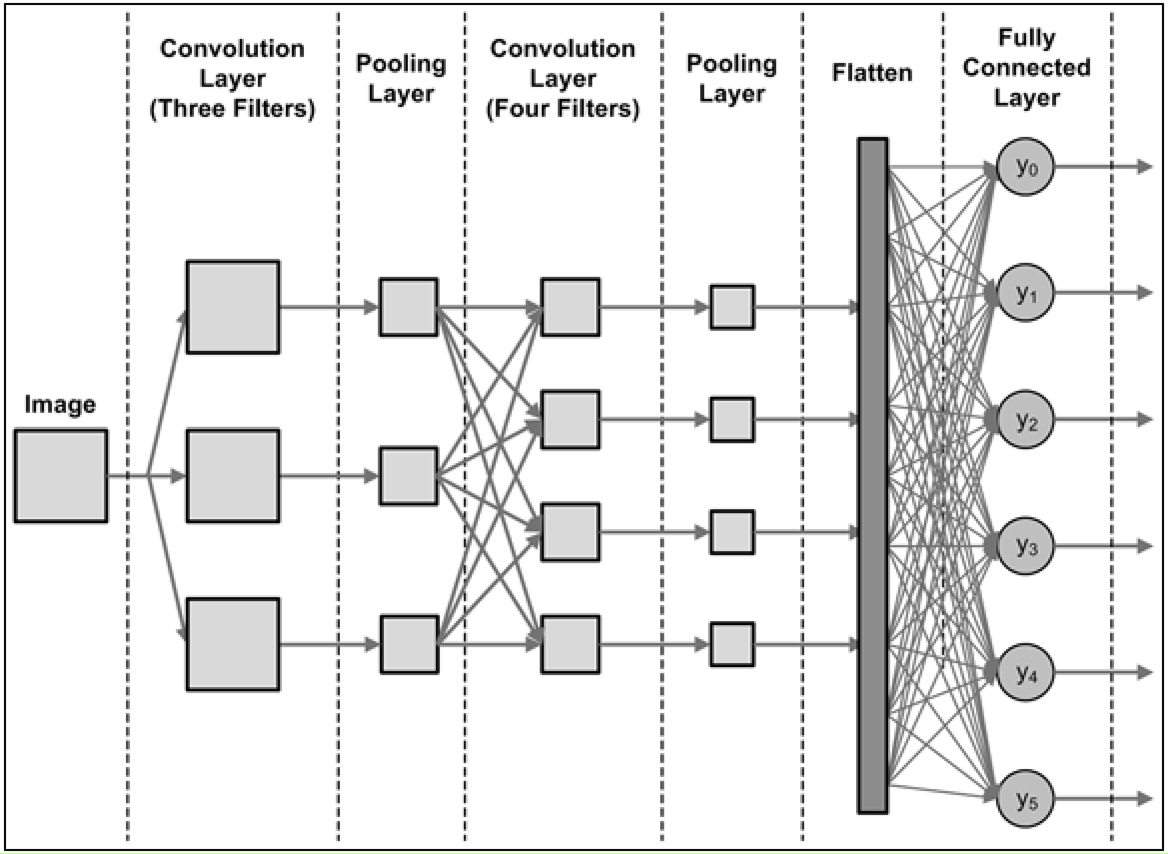
- **Flatten layer**
  - transforms a two-dimensional matrix of features into a vector that can be fed into a fully connected neural network classifier



https://missinglink.ai/guides/keras/using-keras-flatten-operation-cnn-models-code-examples/

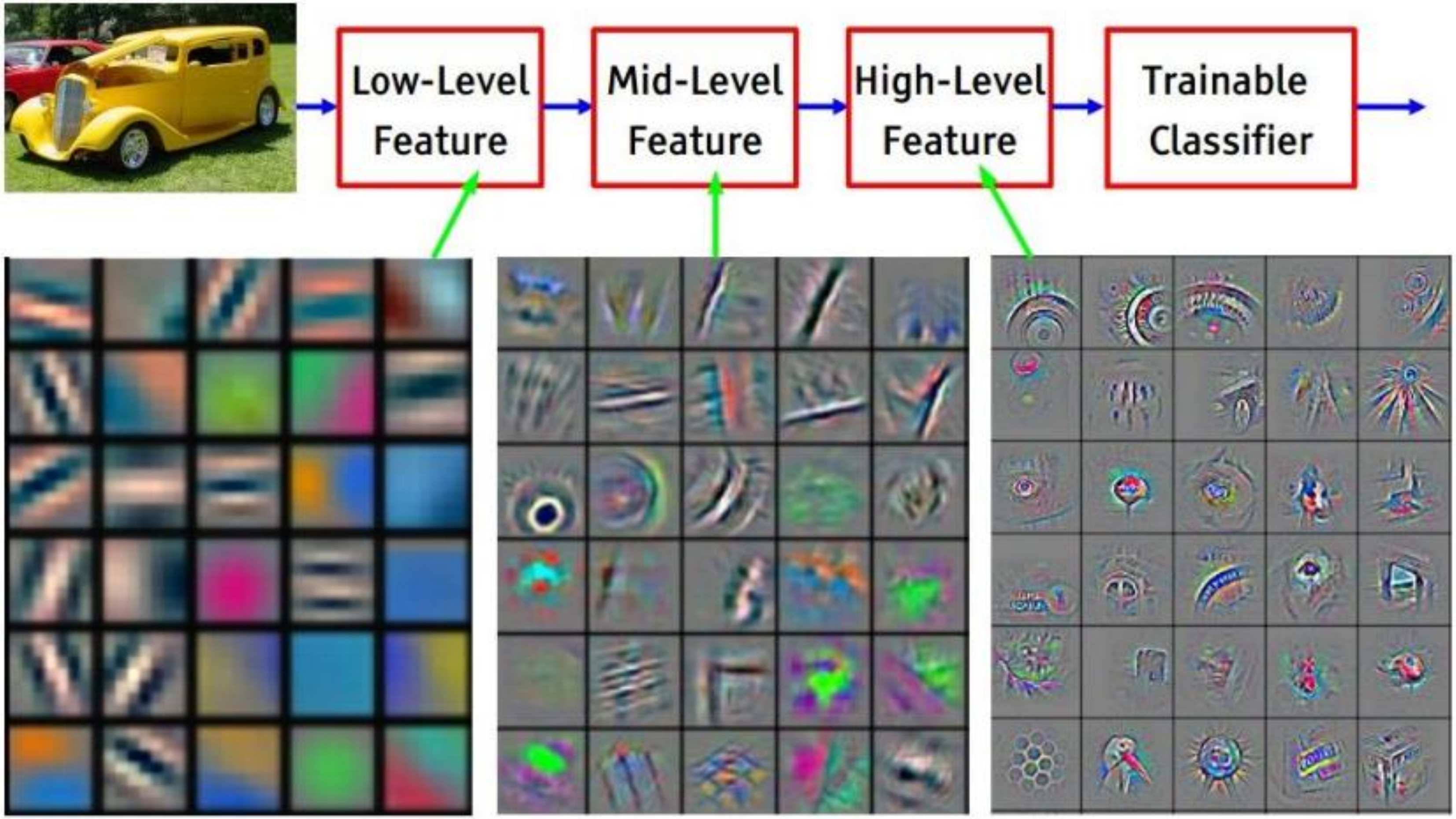
Forming the CNN

- **By combining the convolution, pooling, flatten and dense (fully connected) layers, they formed a Convolution Neural Network.**



How CNN works

- **Each layer extracts the features with increasing complexity**



How CNN works

- **The dense layers (classifier) puts these features together to determine what is the image.**

Ref: https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac

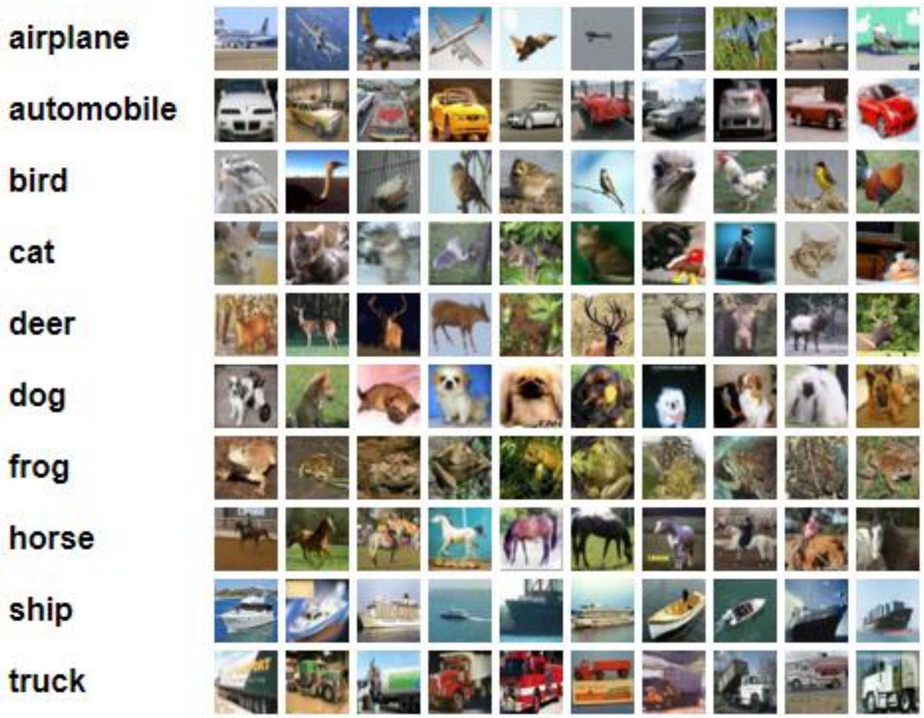
How CNN works (Optional)

- **Explain by YourTube channel – deeplizard**
  - Check out the series on CNN
  - https://www.youtube.com/playlist?list=PLZbbT5o_s2xq7LwI2y8_QtvuXZedL6tQU

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

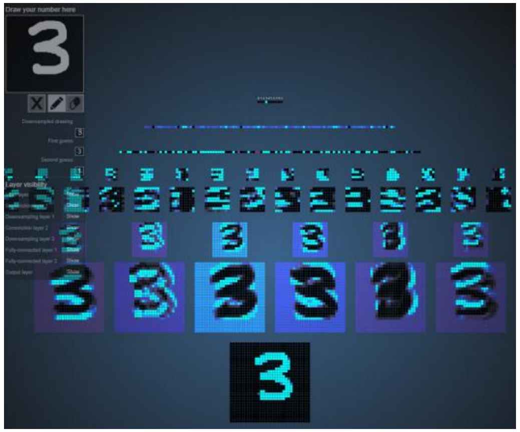
15 Mins Break

Activity 6 – CNN on CIFAR-10



Exercises:

- Visit https://www.cs.ryerson.ca/~aharley/vis/conv/flat.html for an interactive learning of the CNN.



Step 1:
Watch and listen to the instructor's demonstration

15 mins

Step 2:
Work through the activities

Individual Activity

20 mins

Survey

- **Scan this QR-Code for the course feedback survey**

Quiz

- **Go to this link for the quiz:**