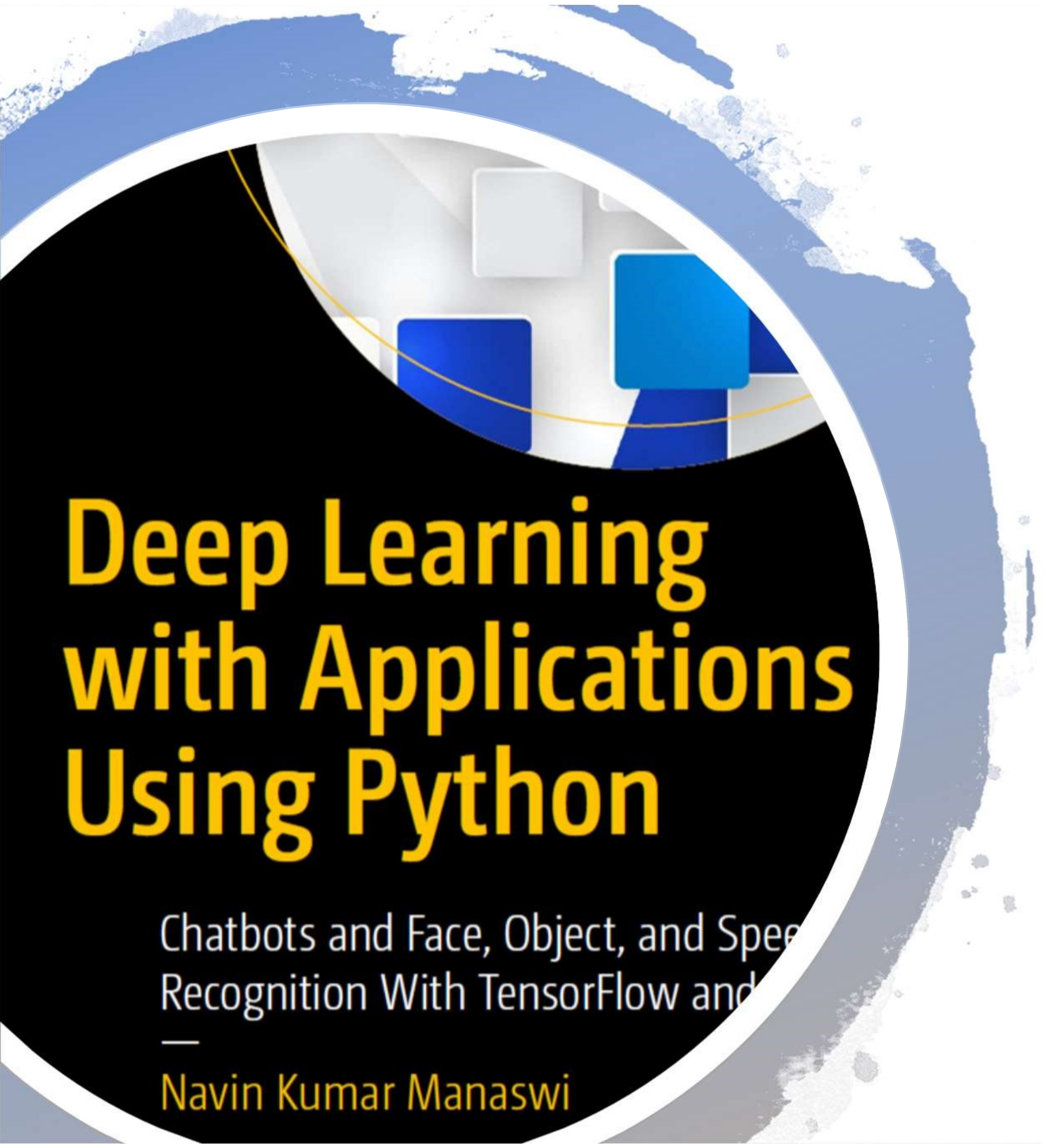

Deep Learning with Applications Using Python
Chatbots and Face, Object, and Spee
Recognition With TensorFlow and
Navin Kumar Manaswi

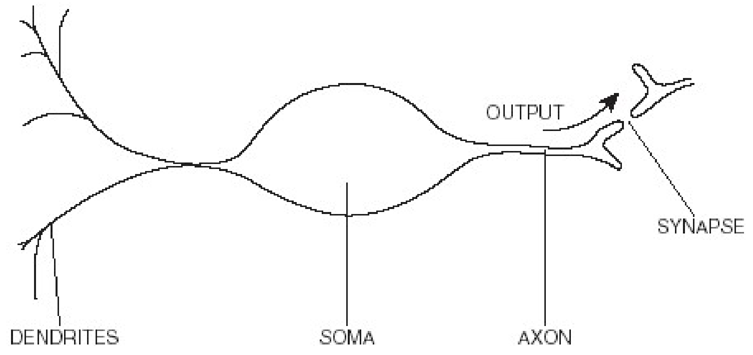
Deep Learning with Applications Using Python

https://github.com/Apress/Deep-Learning-Apps-Using-Python

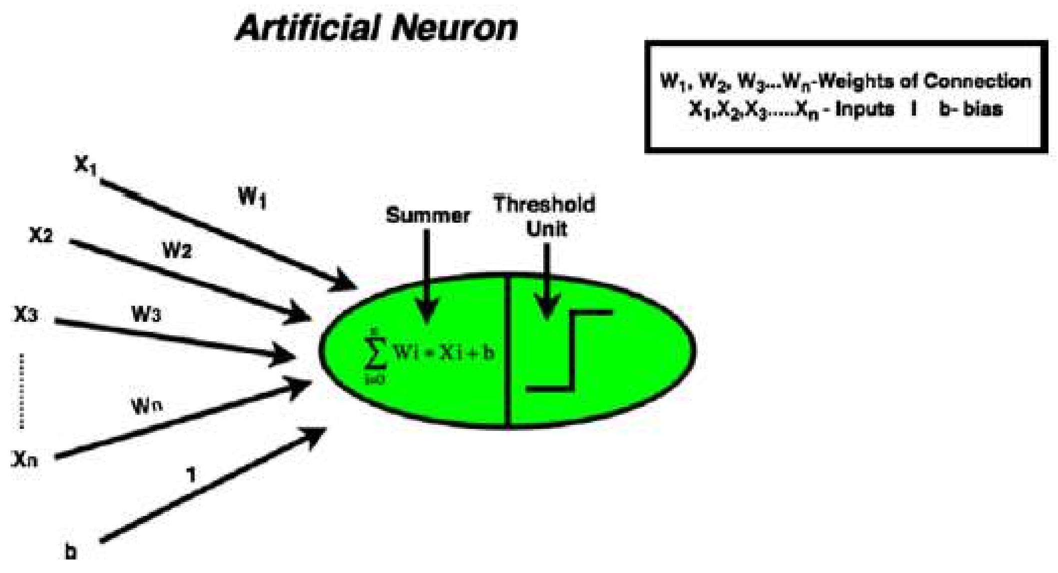
(Some of the slides taken from Prof. Dawn J. Lawrie's CS484 – AI class)

Biological Neurons

- The human brain is made up of billions of simple processing units – neurons.
- Inputs are received on dendrites, and if the input levels are over a threshold, the neuron fires, passing a signal through the axon to the synapse which then connects to another neuron.

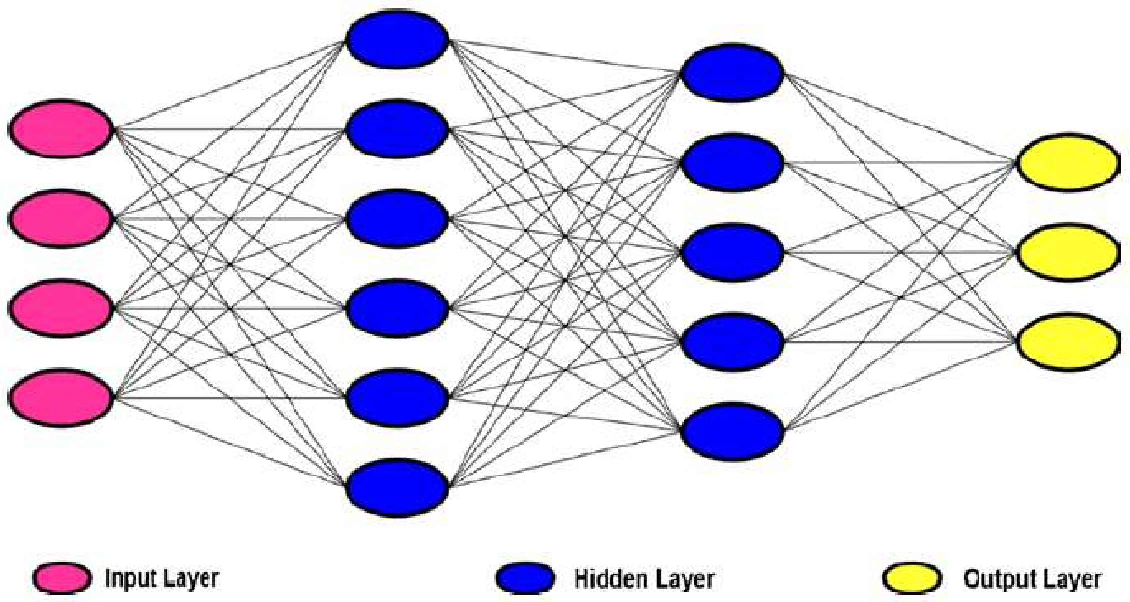


Artificial Neural Network (ANN)


Artificial Neuron
W1, W2, W3...Wn-Weights of Connection
X1,X2,X3.....Xn - Inputs | b- bias
X1
X2
X3
Xn
b
W1
W2
W3
Wn
1
Summer
Threshold Unit
$\sum_{i=0}^{n} Wi * Xi + b$

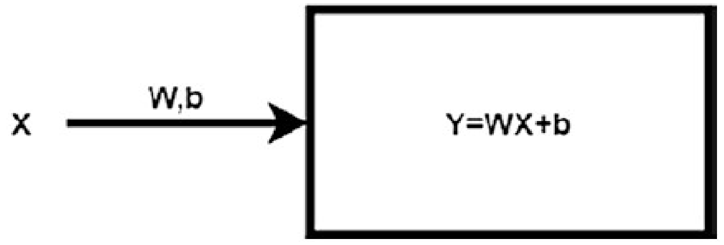
- ANN is a computational network – system of nodes and interconnection between nodes
- Inspired by biological neural networks which are complex networks of neurons in human brains
- Initially, the weights (representing the interconnection) and bias are not good enough to make the decision (classification, etc.)
- Similar to babies, the ANN goes through the process of learning
- The weights are tuned per iteration to create a good classifier
- The process of tuning the weights is called learning or training

Multi-layer Perceptron

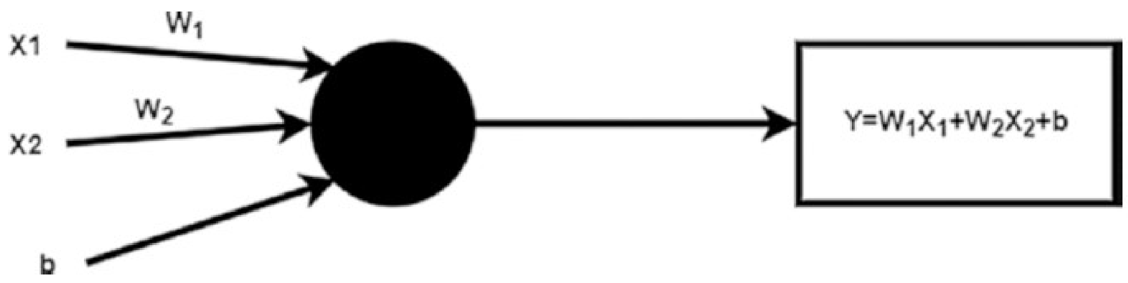


- ANN which has input layer, output layer, and two or more trainable weight layers (consisting of Perceptrons) is called multilayer perceptron or MLP.
- MLP utilizes a supervised learning technique called backpropagation for training.
- Its multiple layers and non-linear activation distinguish MLP from a linear perceptron.
- It can distinguish data that is not linearly separable.

Single-layer Perceptron



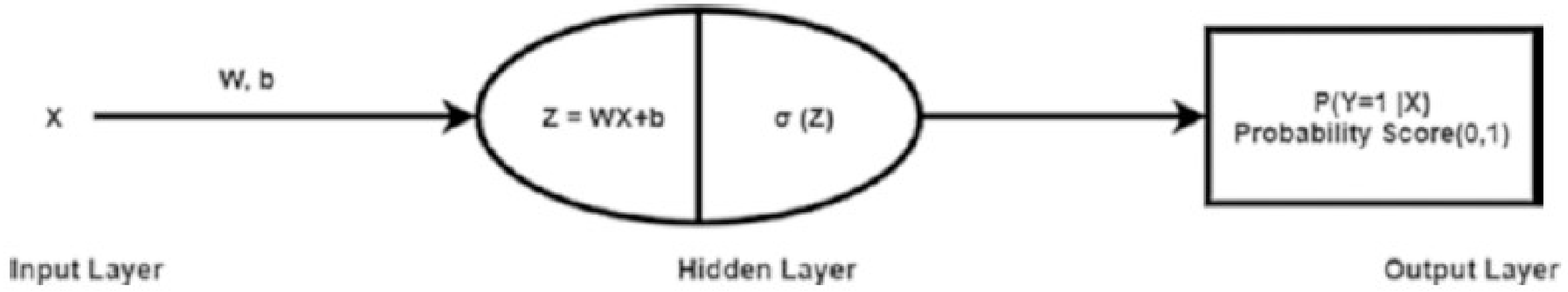
- A single-input model has vector X with weight W and bias b
- Output Y is WX + b which is a linear model



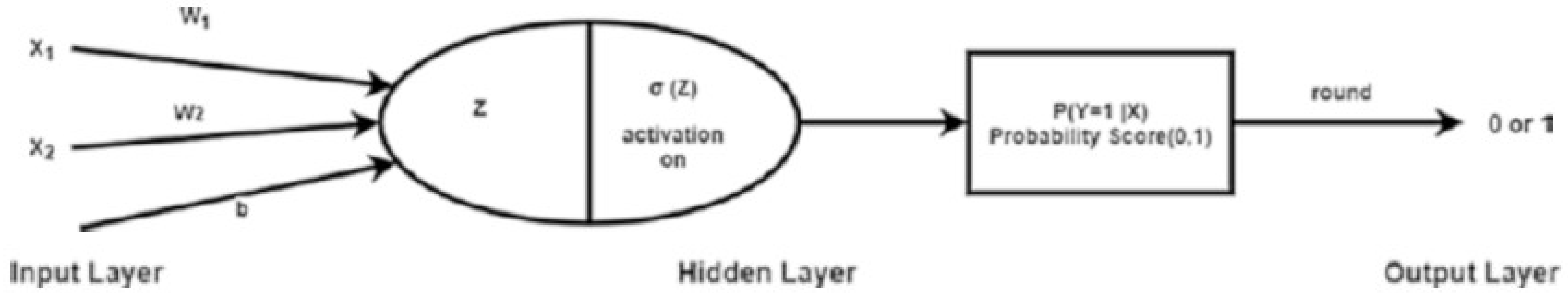
- A liner model has two input features X1 and X2 with corresponding weights and bias b
- Output Y is W1X1 + W2X2 + b

- A simple linear binary classifier
- Takes inputs and associated weights and combines them to produce output that is used for classification
- No hidden layers
- Logistic regression is the single layer perceptron

Perceptron Model for Classification



- Single layer neural network with one input (X) and one output (Y).
- The output Y is σ (Z) where Z is WX + b and σ is sigmoid activation function.



- Multiple inputs (X1 and X2) and one output (Y), also called perceptron.
- Perceptrons can only classify linearly separable functions.

Training Perceptrons

- Learning involves choosing values for the weights
- The perceptron is trained as follows:
  - First, inputs are given random weights (usually between –0.5 and 0.5).
  - An item of training data is presented. If the perceptron mis-classifies it, the weights are modified according to the following:

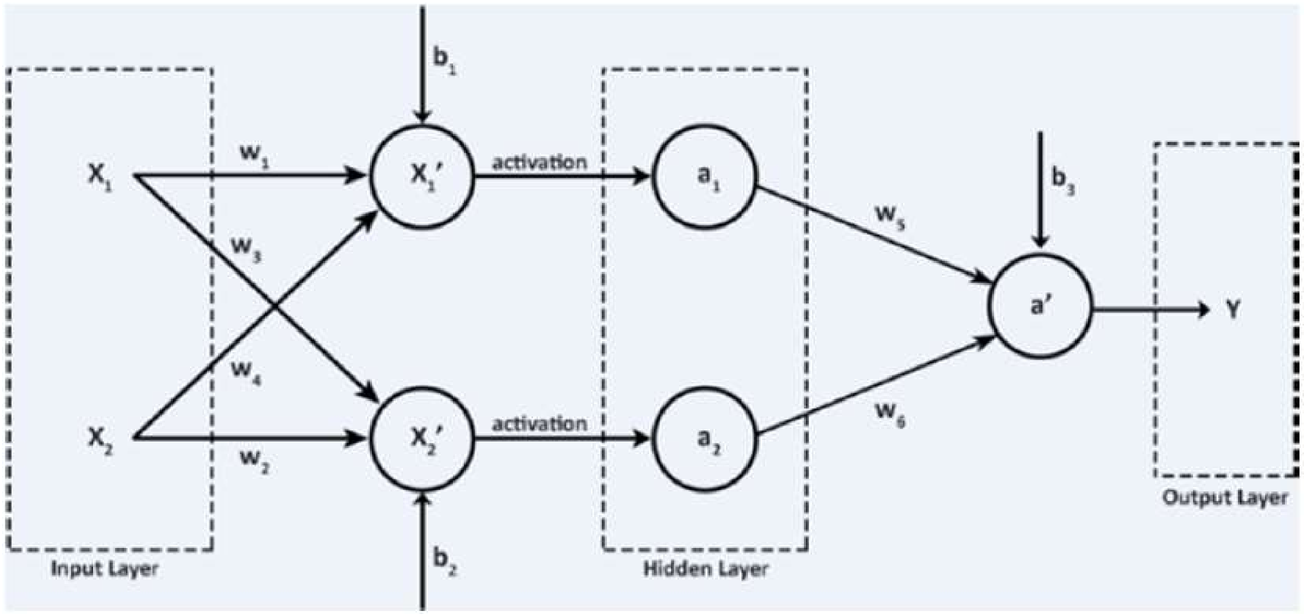
$$w_i \leftarrow w_i + \left(a \times x_i \times (t - o)\right)$$

  - where *t* is the target output for the training example, *o* is the output generated by the perceptron and *a* is the learning rate, between 0 and 1 (usually small such as 0.1)
- Cycle through training examples until successfully classify all examples
  - Each cycle known as an **epoch**

Convergence

- Perceptron training rule only converges when training examples are linearly separable and a has a small learning constant
- Another approach uses the *delta rule* and gradient descent
  - Same basic rule for finding update value
  - Changes
    - Do not incorporate the threshold in the output value (un-thresholded perceptron)
    - Wait to update weight until cycle is complete
  - Converges asymptotically toward the minimum error hypothesis, possibly requiring unbounded time, but converges regardless of whether the training data are linearly separable

Two-layer Neural Network

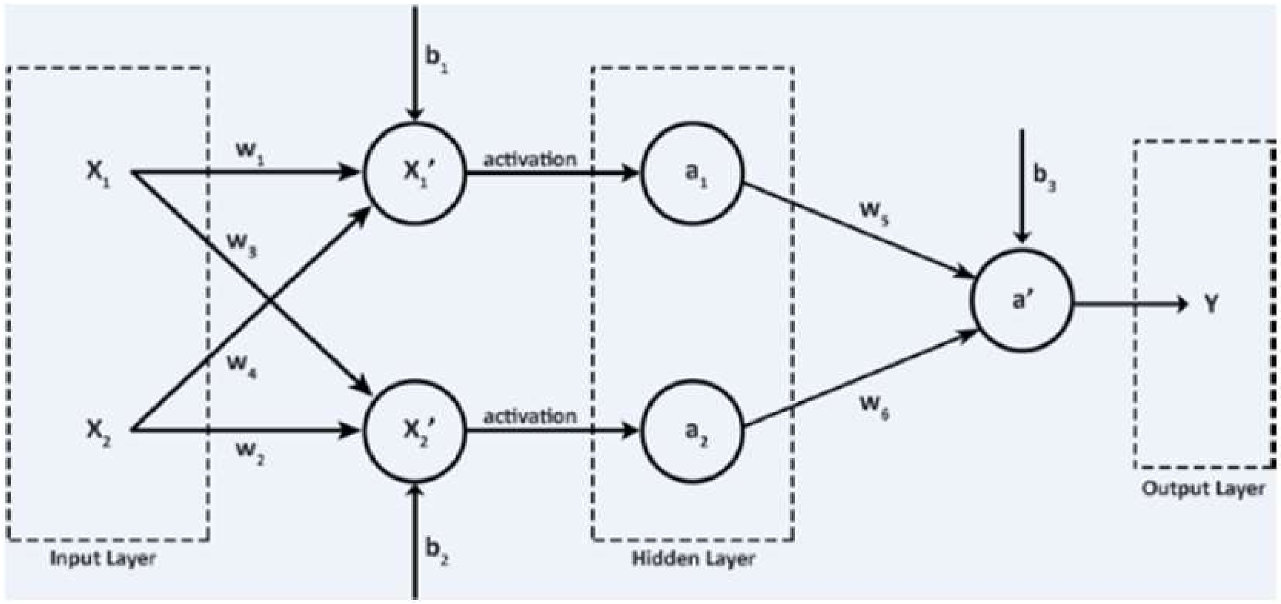


- Two-layer neural network with a hidden layer and an output layer
- Two input feature vectors X1 and X2 connecting to two neurons, X1' and X2'
- The parameters (weights) associated from the input layer to the hidden layer are w1, w2, w3, w4, b1, b2.
- (2×1)(2×2)(2×1)(2×1) is the dimension of the input and hidden layers.

X1' and X2' compute the linear combination

$$\begin{bmatrix} X1' \\ X2' \end{bmatrix} = \begin{bmatrix} w1 & w2 \\ w3 & w4 \end{bmatrix} \begin{bmatrix} X1 \\ X2 \end{bmatrix} + \begin{bmatrix} b1 \\ b2 \end{bmatrix}$$

Two-layer Neural Network



- The linear input X1’ and X2’ passes through the activation unit a1 and a2.
- a1 is σ (X1’) and a2 is σ(X2’)

$$\begin{bmatrix} a1 \\ a2 \end{bmatrix} = \sigma \begin{bmatrix} X1' \\ X2' \end{bmatrix}$$

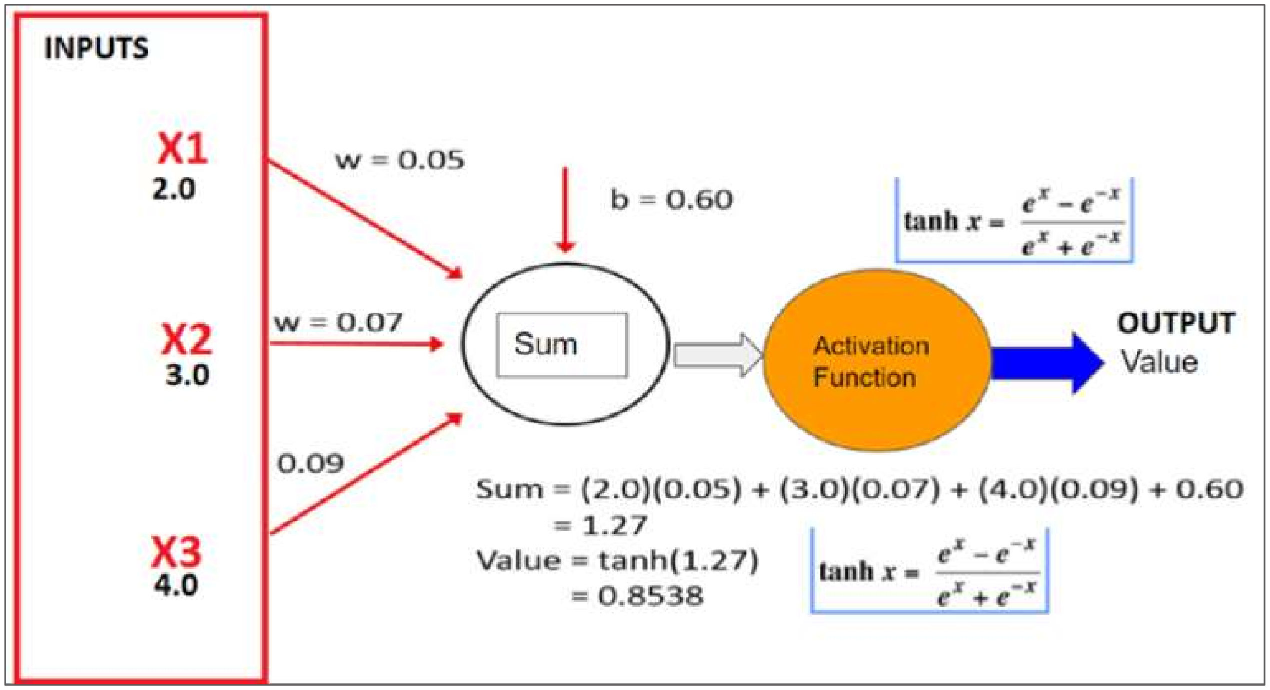
a’ is linear combination of (w5*a1 + w6*a2) + b3

$$\mathrm{a}' = \begin{bmatrix} w5 & w6 \end{bmatrix} \begin{bmatrix} a1 \\ a2 \end{bmatrix} + \begin{bmatrix} b3 \end{bmatrix}$$

a’ will passthrough nonlinear sigmoid function to the final output layer.

Y = σ(a’)

Activation Functions


INPUTS
X1
2.0
X2
3.0
X3
4.0
w = 0.05
w = 0.07
0.09
b = 0.60
Sum
Activation Function
OUTPUT
Value
$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Sum = (2.0)(0.05) + (3.0)(0.07) + (4.0)(0.09) + 0.60
= 1.27
Value = tanh(1.27)
= 0.8538
$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

- The neurons become active beyond a certain threshold, known as *activation potential*.
- In neural network, activation function attempts to put the output into a small range.
- Most popular output functions are sigmoid, hyperbolic tangent (tanh), ReLU and ELU

Backpropagation

- Multilayer neural networks learn in the same way as perceptrons.
- However, there are many more weights, and it is important to assign credit (or blame) correctly when changing weights.
- *E* sums the errors over all of the network output units

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

Backpropagation Algorithm

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers
- Until termination condition is met, Do
  - For each <*x*,*t*> in training examples, Do

    Propagate the input forward through the network:

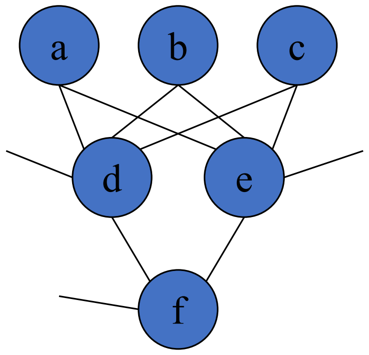
    1. Input the instance x to the network and compute the output o_u of every unit u in the network

    Propagate the errors backward through the network:

    2. For each network output unit k, calculate its error term δ_k $$\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$$
    3. For each hidden unit h, calculate its error term δ_h $$\delta_h \leftarrow o_h(1-o_h)\sum_{k\in outputs} w_{kh}\delta_k$$
    4. Update each network weight w_{ji} $$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

       where $$\Delta w_{ji} = \alpha \delta_j x_{ji}$$

Learning AND



Training Data:

AND(1,0,1) = 0

AND(1,1,1) = 1

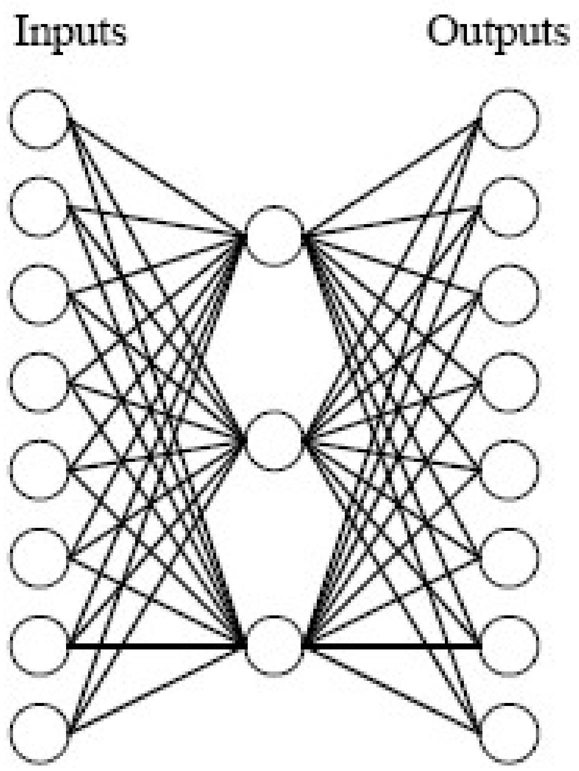
Alpha = 0.1

- Initial Weights:
- w_da = .2
- w_db = .1
- w_dc = -.1
- w_d0 = .1

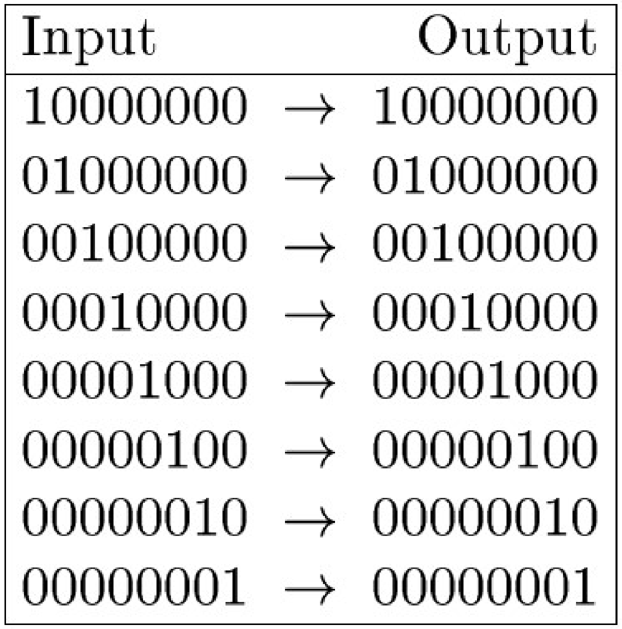
- w_ea = -.5
- w_eb = .3
- w_ec = -.2
- w_e0 = 0

- w_fd = .4
- w_fe = -.2
- w_f0 = -.1

Hidden Layer Representation



Target Function:



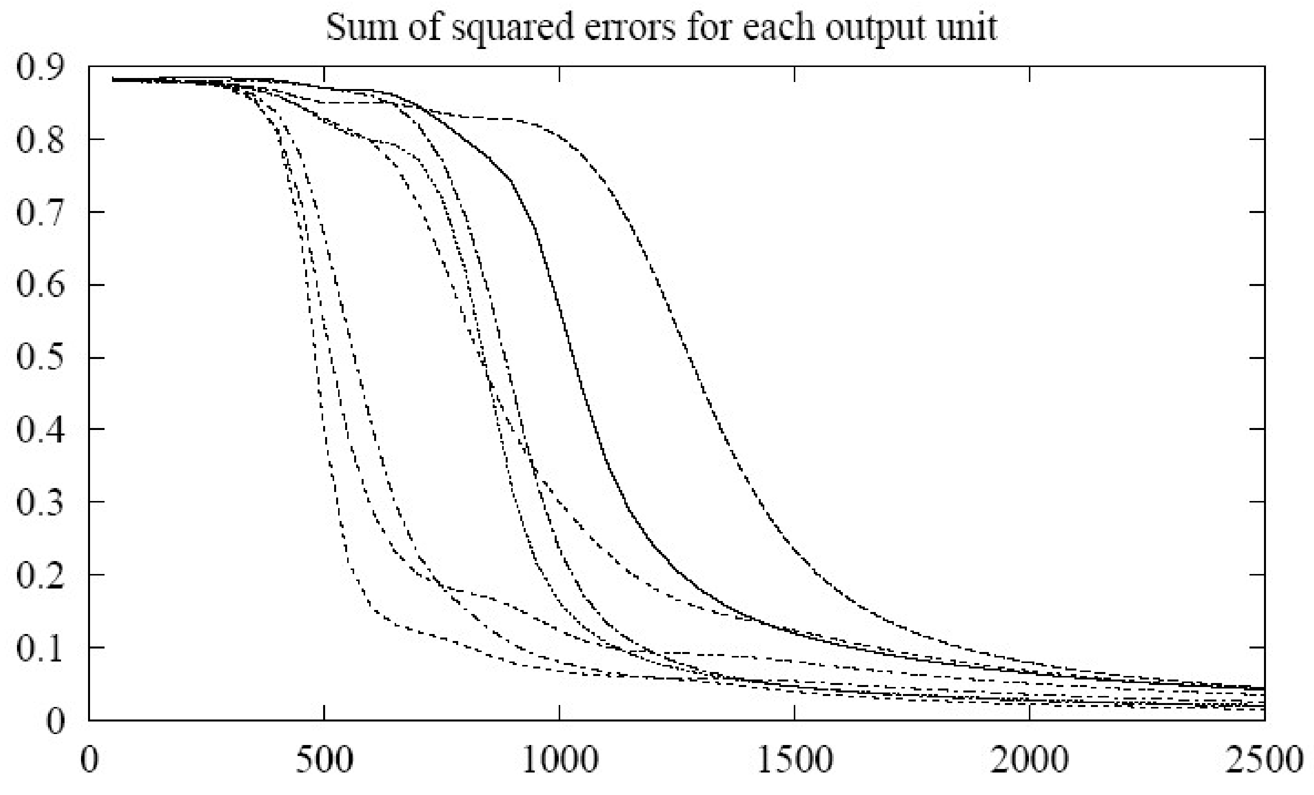
Input		Output
10000000	$\rightarrow$	10000000
01000000	$\rightarrow$	01000000
00100000	$\rightarrow$	00100000
00010000	$\rightarrow$	00010000
00001000	$\rightarrow$	00001000
00000100	$\rightarrow$	00000100
00000010	$\rightarrow$	00000010
00000001	$\rightarrow$	00000001

Can this be learned?

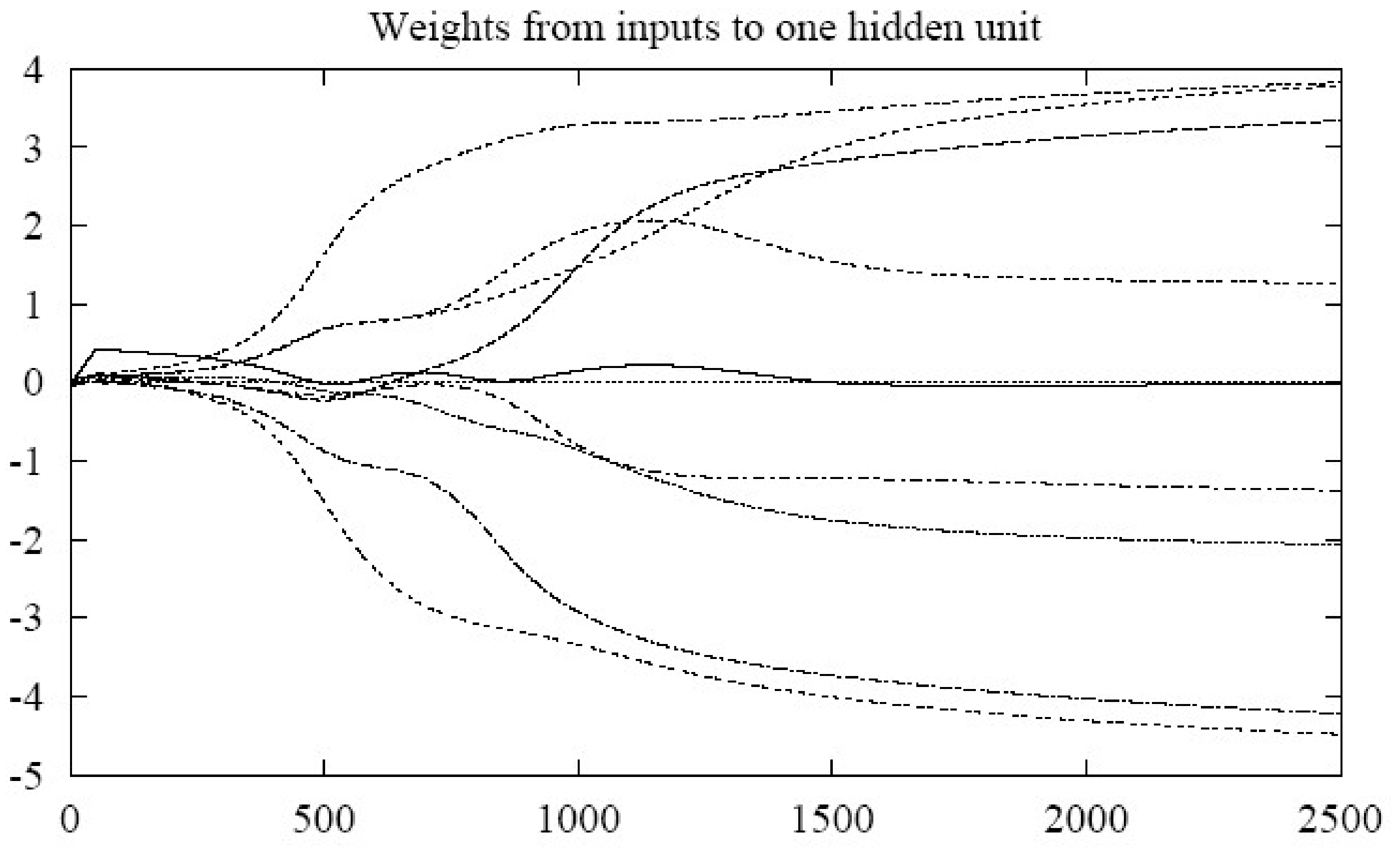
Yes!

Input		Hidden Values		Output
10000000	→	.89 .04 .08	→	10000000
01000000	→	.15 .99 .99	→	01000000
00100000	→	.01 .97 .27	→	00100000
00010000	→	.99 .97 .71	→	00010000
00001000	→	.03 .05 .02	→	00001000
00000100	→	.01 .11 .88	→	00000100
00000010	→	.80 .01 .98	→	00000010
00000001	→	.60 .94 .01	→	00000001

Plot of Squared Error



Evolving Weights



Momentum

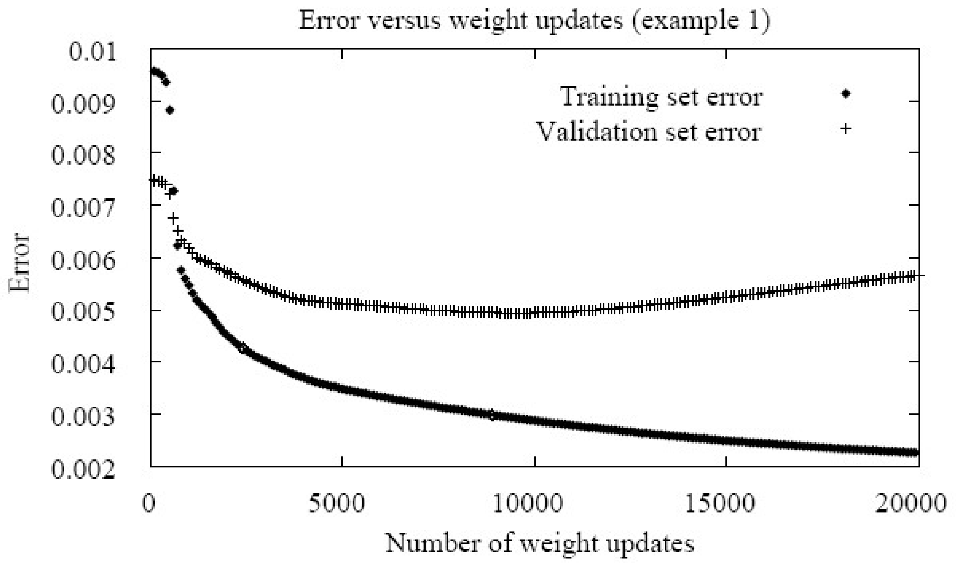
- One of many variations
- Modify the update rule by making the weight update on the *n*th iteration depend partially on the update that occurred in the (*n*-1)th iteration

$$\Delta w_{ji}(n) = \alpha \delta_j x_{ji} + \beta \Delta w_{ji}(n-1)$$

- Minimizes error over training examples
- Speeds up training since it can take 1000s of iterations

When to Stop Training?

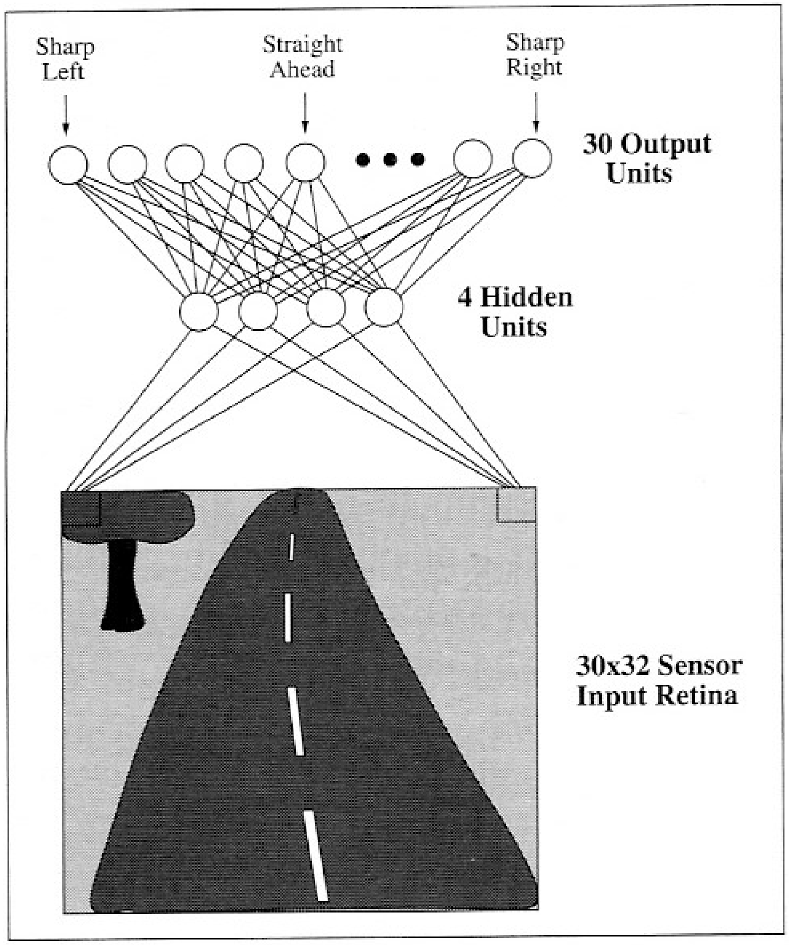
- Continue until error falls below some predefined threshold
  - Bad choice because Backpropagation is susceptible to overfitting
  - Won't be able to generalize as well over unseen data

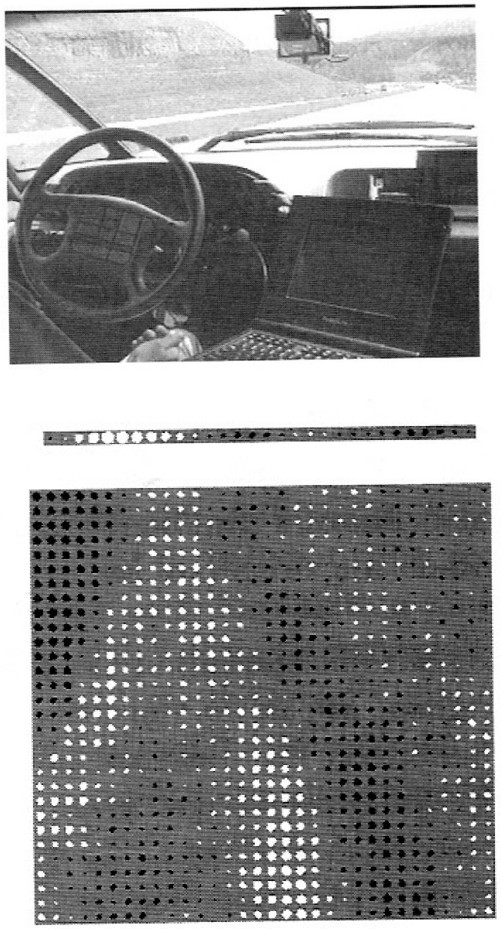


Example: ALVINN

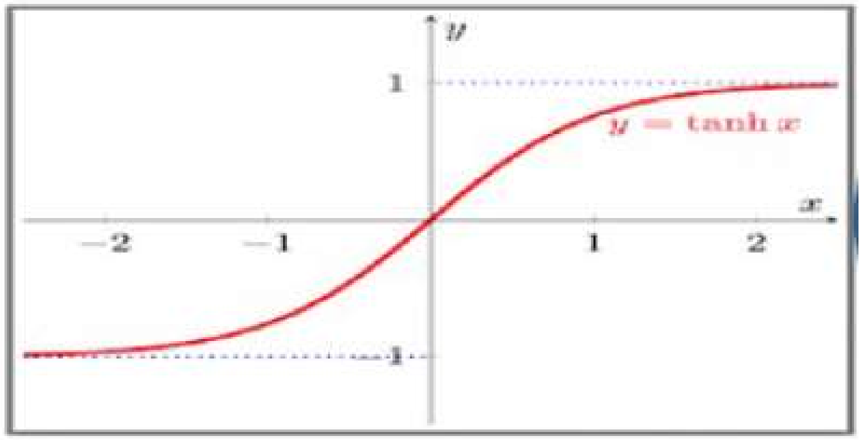
- ALVINN uses a learned ANN to steer an autonomous vehicle driving at normal speeds on public highways
  - Input to network: 30x32 grid of pixel intensities obtained from a forward-pointed camera mounted on the vehicle
  - Output: direction in which the vehicle is steered
  - Trained to mimic observed steering commands of a human driving the vehicle for approximately 5 minutes

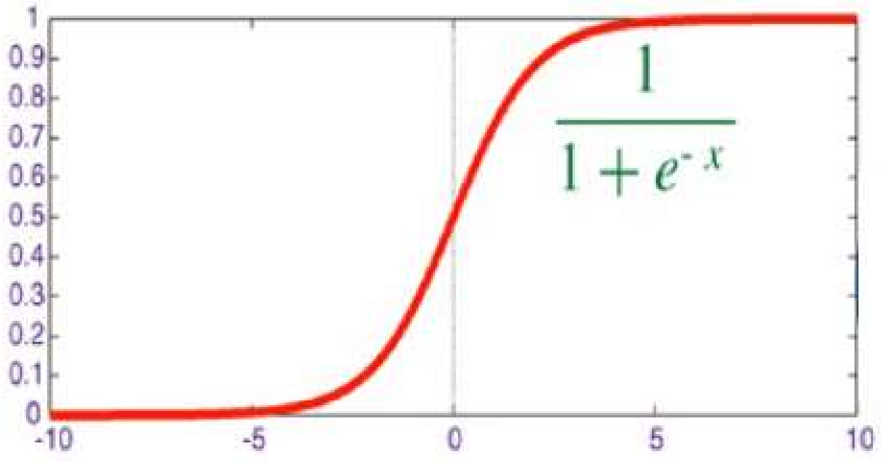
Example: ALVINN





Tanh and Sigmoid Activation Functions

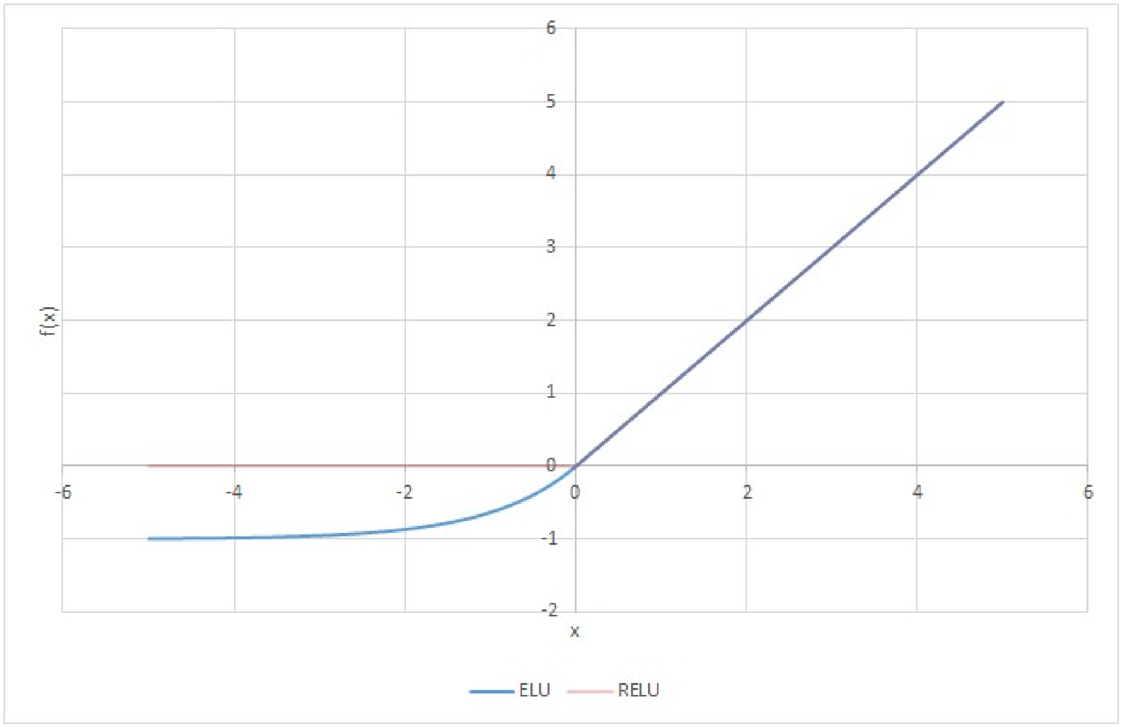




- Tangent Hyperbolic: Output always lies between -1 and 1 no matter what inputs are

- Sigmoid: Output always lies between 0 and 1 no matter what inputs are

ReLU and ELU Activation Functions

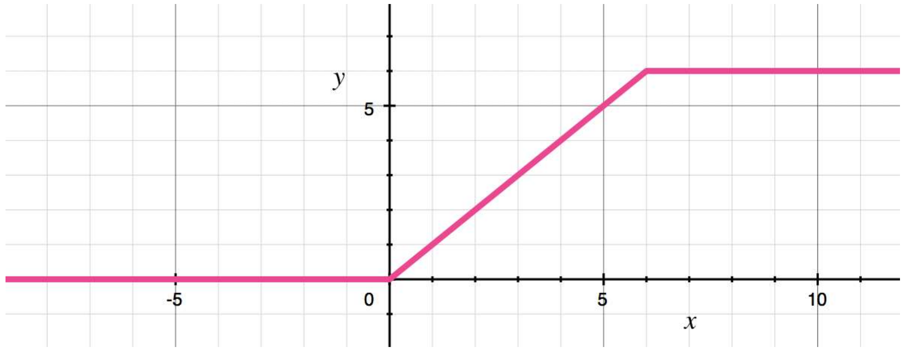


- Rectified Linear Unit (ReLU): If input <= 0, output is 0 else equal to input

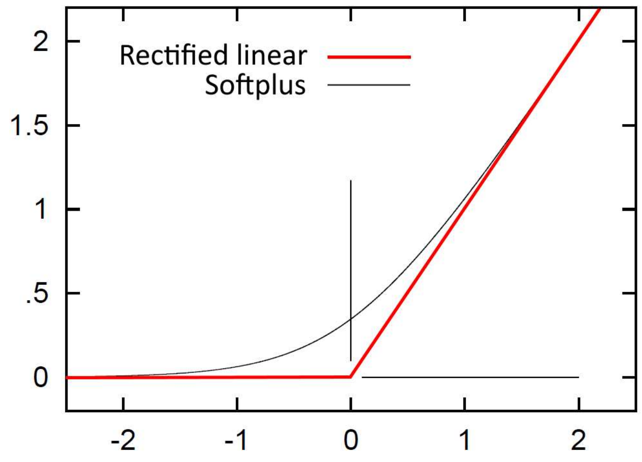
- Exponential Linear Unit (ELU): If input >0 same as ReLU, else slightly below 0

https://mlfromscratch.com/activation-functions-explained/

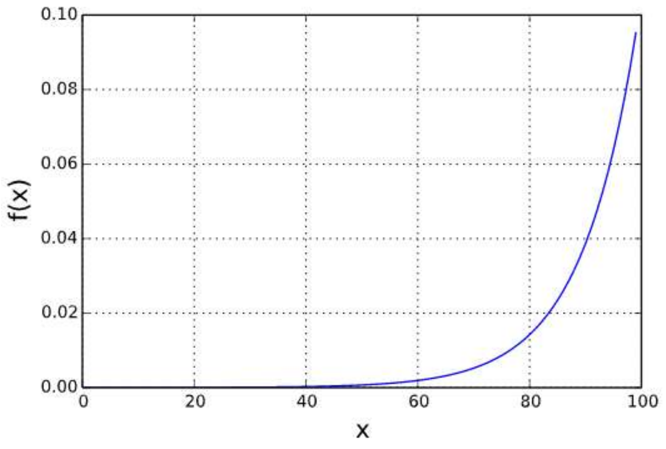
Relu6, Softplus & Softsign Activation Functions



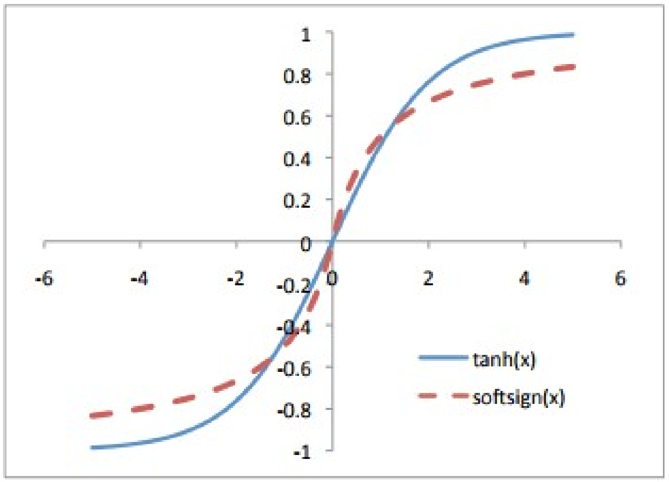
- ReLU6: If input <= 0, output is 0 else if input <= 6 equal to input else equal to 6



Softplus: $f(x) = \ln(1+e^x)$



Softmax: $f(x) = \ln(1+e^x)$



- Sofsign: $y = x / (1 + |x|)$

Loss Functions

- The loss function (aka cost function) is to be minimized to get the best values for each parameter (such as weight and bias) of the model.
- For evaluation of the model, cost function need to be defined.
- The minimization of cost function is the driving force for finding the optimum value of each parameter.
- Some cost functions are:
  - L1 or L2 for regression
  - Cross entropy for classification

Optimizer

- Optimizer helps to reach best values of the parameters.
- In each iteration, the value changes in the direction suggested by an optimizer.
- Given a set of 16 weight values (w1, w2, w3, …, w16) and 4 biases (b1, b2, b3, b4), the initial assignment is zero of one or any number.
- Optimizer suggests whether w1 and other params should increase or decrease in the next iteration of (learning algorithm backpropagation) while trying to minimize the loss.

Some Optimizers

- Adaptive techniques – adadelta, adagrad help converging faster for complex neural networks.
- Adam outperforms adaptive techniques, however, is computationally costly.
- Stochastic Gradient Descent
- Adaptive learning rates

Evaluation Metrics

- Metrics are used for evaluation of regressor or classifier.
- Some of the metrics are classification accuracy, logarithmic loss and area under ROC curve.
- Classification accuracy is the ratio of the number of correct predictions to the number of all predictions.

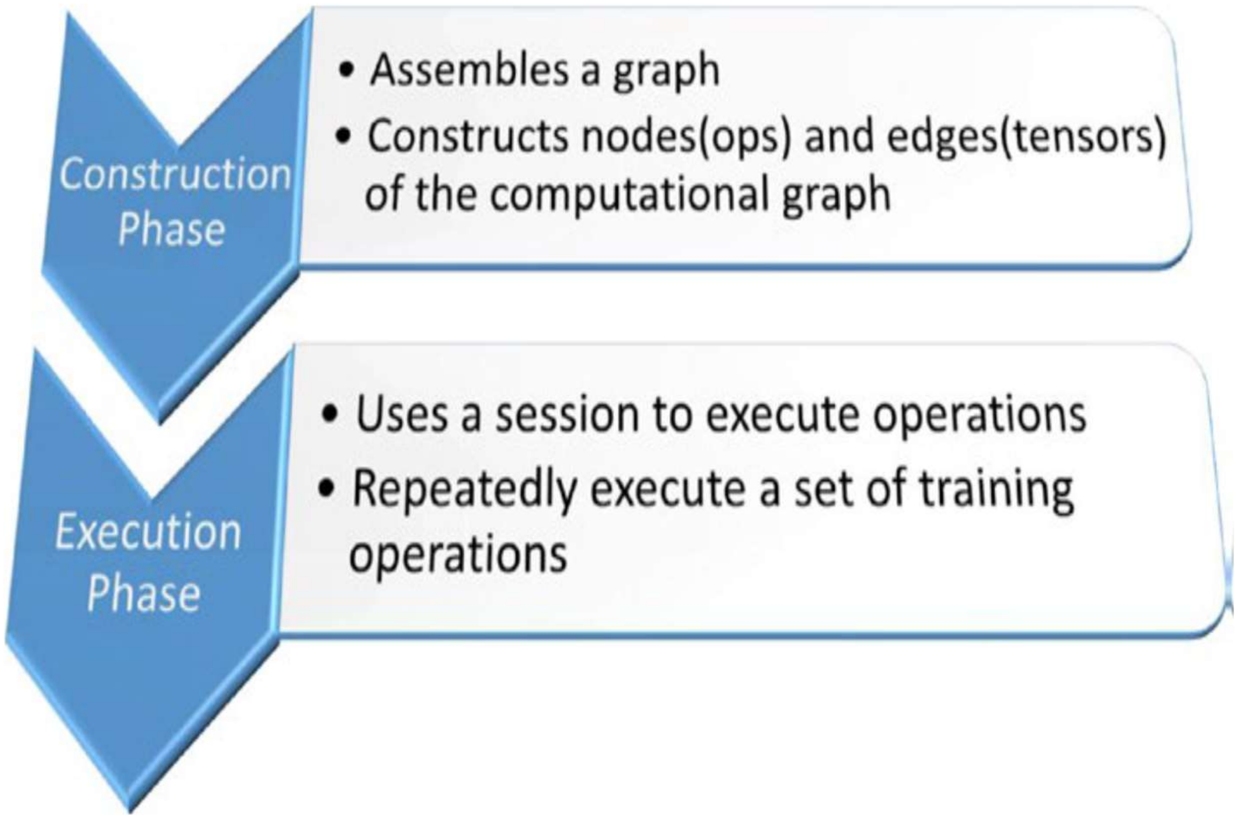
Basics of TensorFlow

- Deep learning framework released by Google in November 2015
- Deep learning does a wonderful job in pattern recognition, especially in the context of images, sound, speech, language, and time-series data.
- Installation: https://www.tensorflow.org/install/
- Jupyter Notebooks: https://github.com/Apress/Deep-Learning-Apps-Using-Python

What is a Tensor

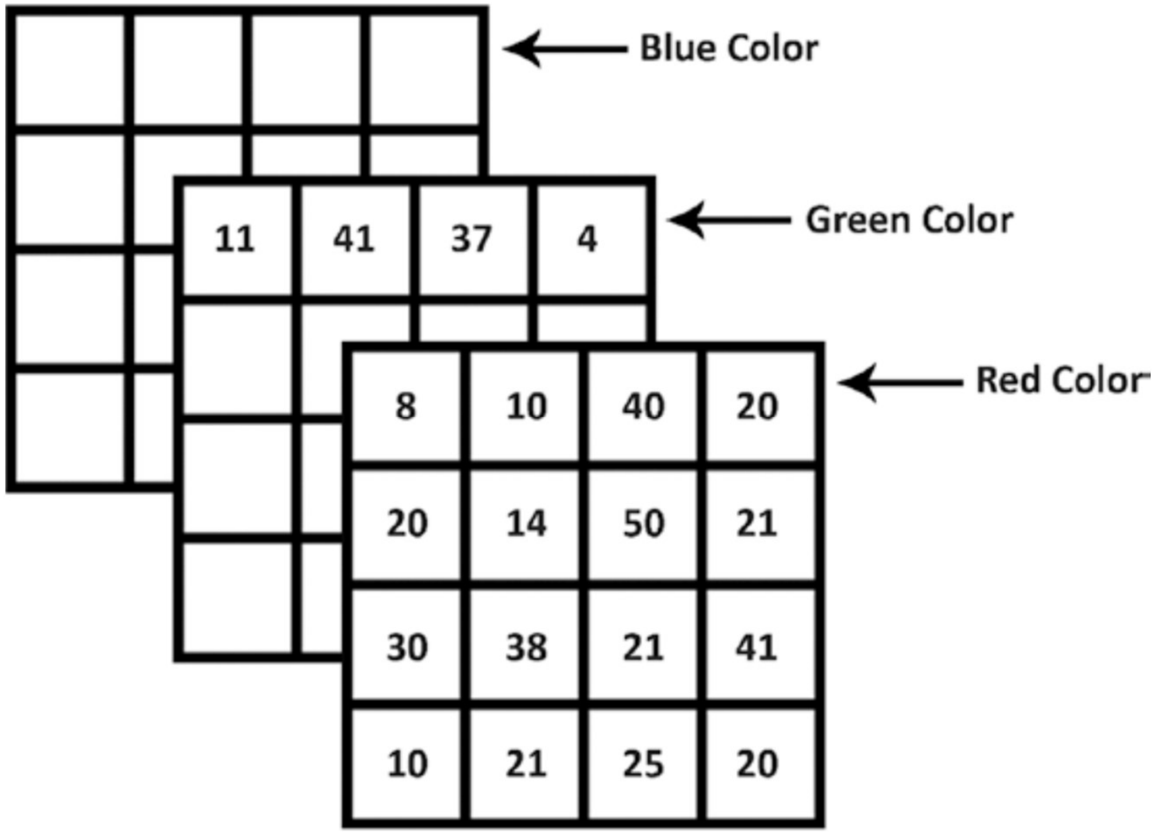
- A tensor is a mathematical object and a generalization of scalars, vectors and matrices.
- A tensor can be represented as a multidimensional array.
- A tensor with zero rank (order) is a scalar.
- A tensor with rank 1 is a vector/array.
- Matrix is a tensor of rank 2.
  - 5: This is a rank 0 tensor; this is a scalar with shape [ ].
  - [2.,5., 3.]: This is a rank 1 tensor; this is a vector with shape [3].
  - [[1., 2., 7.], [3., 5., 4.]]: This is a rank 2 tensor; it is a matrix with shape [2, 3].
  - [[[1., 2., 3.]], [[7., 8., 9.]]]: This is a rank 3 tensor with shape [2, 1, 3].

Structure of a TensorFlow program


Construction Phase
• Assembles a graph
• Constructs nodes(ops) and edges(tensors) of the computational graph
Execution Phase
• Uses a session to execute operations
• Repeatedly execute a set of training operations

- Build the computational graph in the construction phase.
- Run the computational graph in the execution phase.
- A *computational graph* is a series of TensorFlow operations arranged into a graph of nodes.
- The simplest operation is a constant that takes no inputs but passes outputs to other operations that do computation.
- An example of an operation is multiplication (or addition or subtraction that takes two matrices as input and passes a matrix as output).
- A computational graph needs to be ruin in a session which encapsulates control and state of TF runtime.

Convert Image to a Tensor



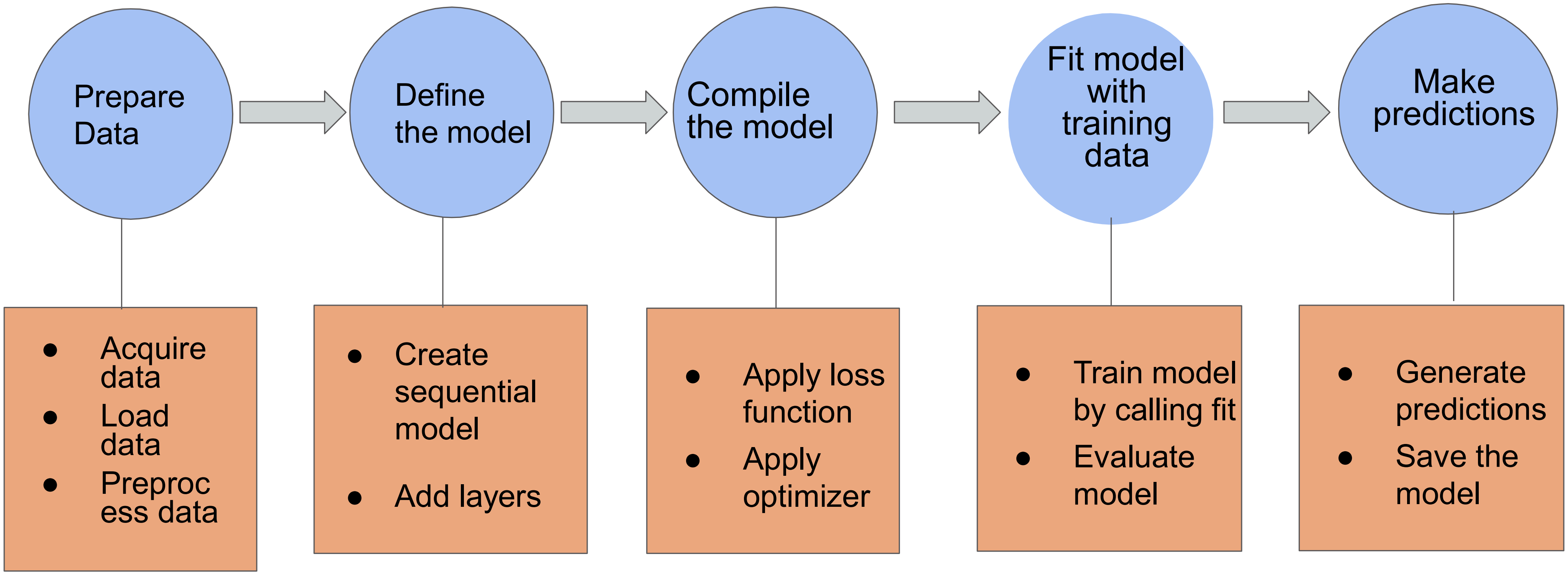
Some Common Tensorflow Loss Functions

- tf.contrib.losses.absolute_difference
- tf.contrib.losses.add_loss
- tf.contrib.losses.hinge_loss
- tf.contrib.losses.compute_weighted_loss
- tf.contrib.losses.cosine_distance
- tf.contrib.losses.get_losses
- tf.contrib.losses.get_regularization_losses
- tf.contrib.losses.get_total_loss
- tf.contrib.losses.log_loss
- tf.contrib.losses.mean_pairwise_squared_error
- tf.contrib.losses.mean_squared_error
- tf. contrib.losses.sigmoid_cross_entropy
- tf.contrib.losses.softmax_cross_entropy

Using TensorFlow

- TensorFlow basics
- MLP: Implementing a hidden layer - MLP

Deep Learning with Keras



Using Keras for MLP training

- MLP in Iris Dataset

Convolutional Neural Networks (CNN)

- A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers.
- *CNN* is a deep, feed-forward ANN in which the neural network preserves the hierarchical structure by learning internal feature representations and generalizing the features in the common image problems like object recognition and other computer vision problems.
- It is not restricted to images; it also achieves state-of-the-art results in natural language processing problems and speech recognition.
- A convolution is a linear operation that involves the multiplication of a set of weights with the input
- CNN takes two-dimensional input, the multiplication is performed between an array of input data and a two-dimensional array of weights, called a filter or a kernel.

CNN - Convolution

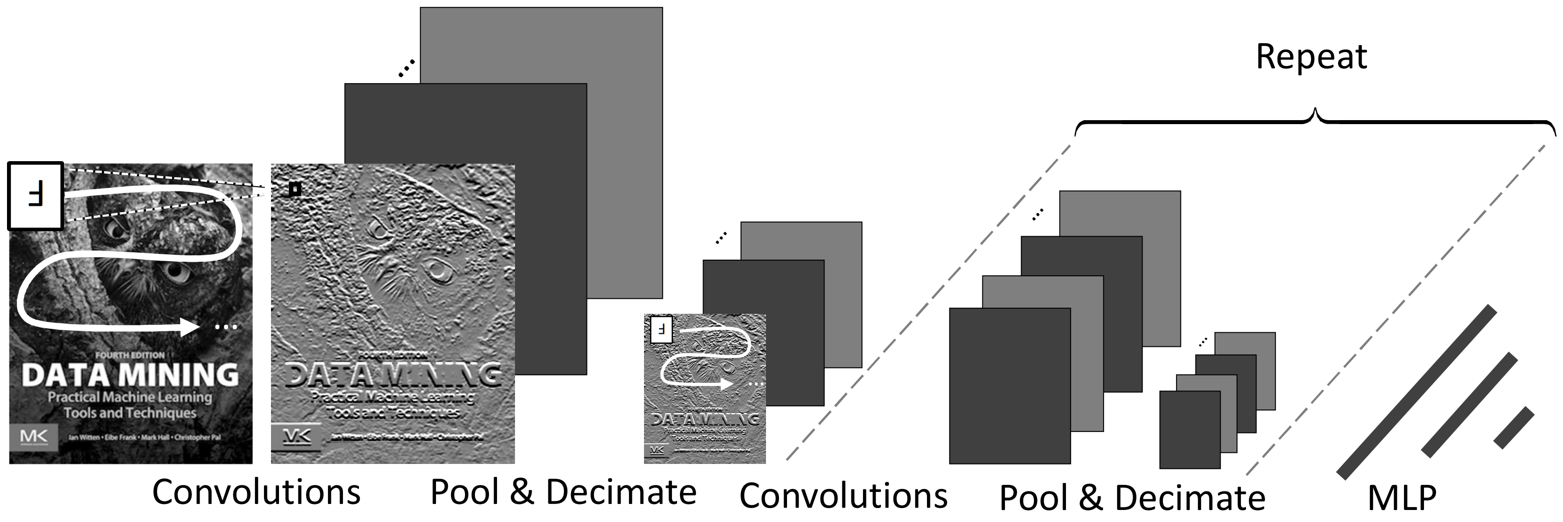
- Imagine filtering an image to detect edges, one could think of edges as a useful set of spatially organized ‘features’
- Imagine now if one could learn many such filters jointly along with other parameters of a neural network on top
- Each filter can be implemented by multiplying a relatively small spatial zone of the image by a set of weights and feeding the result to an activation function
- Because this filtering operation is simply repeated around the image using the same weights, it can be implemented using convolution operations
- The result is a CNN for which it is possible to learn both the filters and the classifier using SGD and the backpropagation algorithm

CNN – Pooling and Subsampling

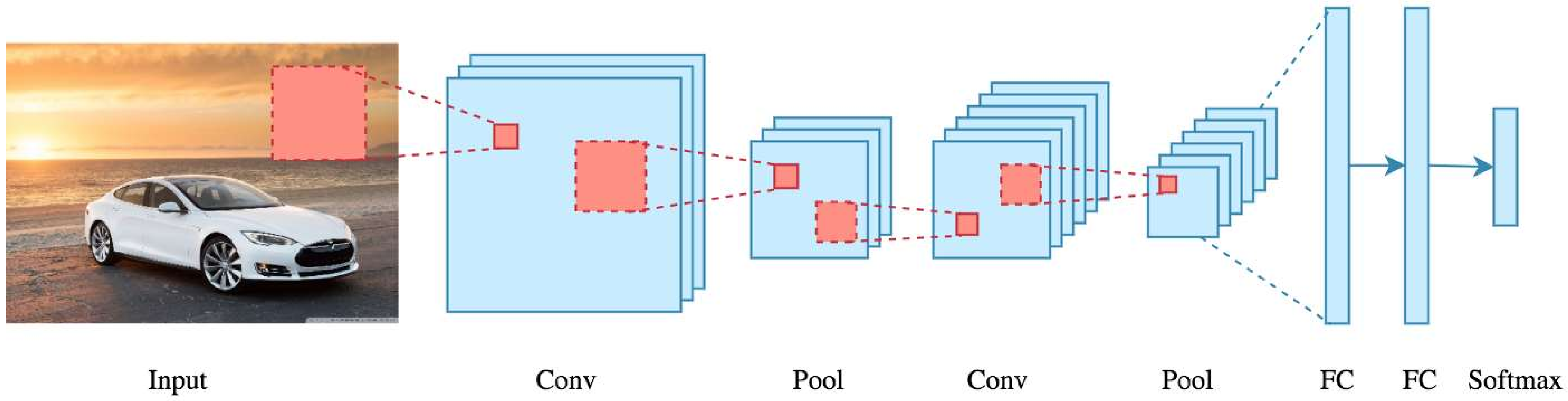
- In a convolutional neural network, once an image has been filtered by several learnable filters, each filter bank’s output is often aggregated across a small spatial region, using the average or maximum value.
- Aggregation can be performed within non-overlapping regions, or using subsampling, yielding a lower-resolution layer of spatially organized features
- This gives the model a degree of invariance to small differences as to exactly where a feature has been detected
- If aggregation uses the max operation, a feature is activated if it is detected anywhere in the pooling zone
- The result can be filtered and aggregated again

A typical CNN architecture

- Many feature maps are obtained from convolving learnable filters across an image
- Results are aggregated or pooled & decimated
- Process repeats until last set of feature maps are given to an MLP for final prediction



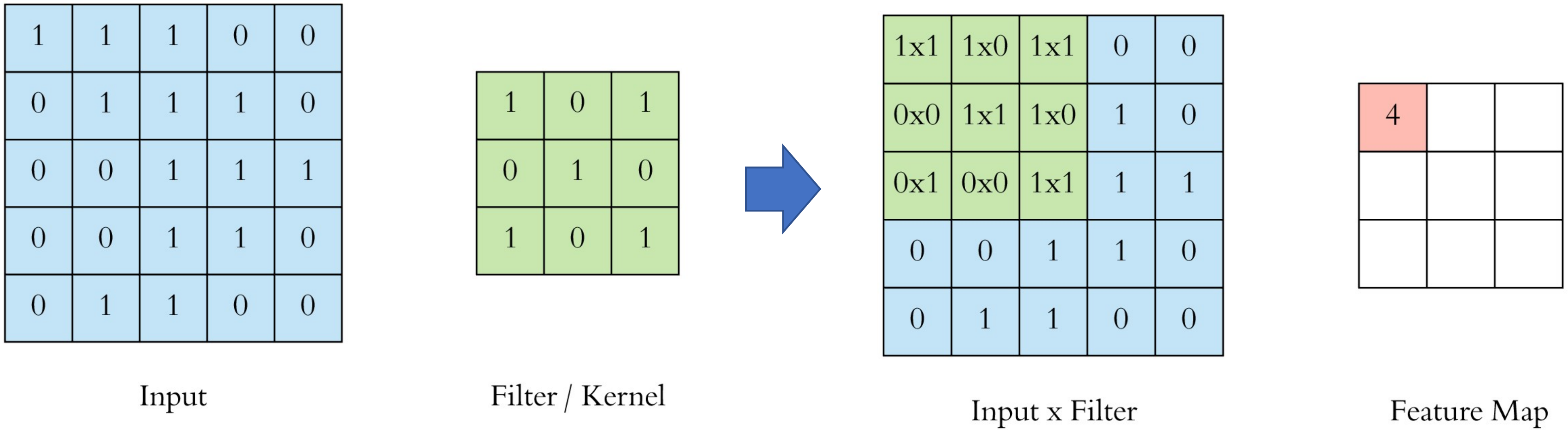
Convolutional Neural Networks (CNN)



- There is an input image that we're working with. We perform a series convolution + pooling operations, followed by a number of fully connected layers.
- If we are performing multiclass classification the output is softmax.

Convolution

- Convolution is a mathematical operation that merges tow sets of information.
- Convolution is applied on the input data using a convolution filter to produce a feature map.
- Convolution operation is performed by sliding filter over the input.



1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Input

1	0	1
0	1	0
1	0	1

Filter / Kernel

1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

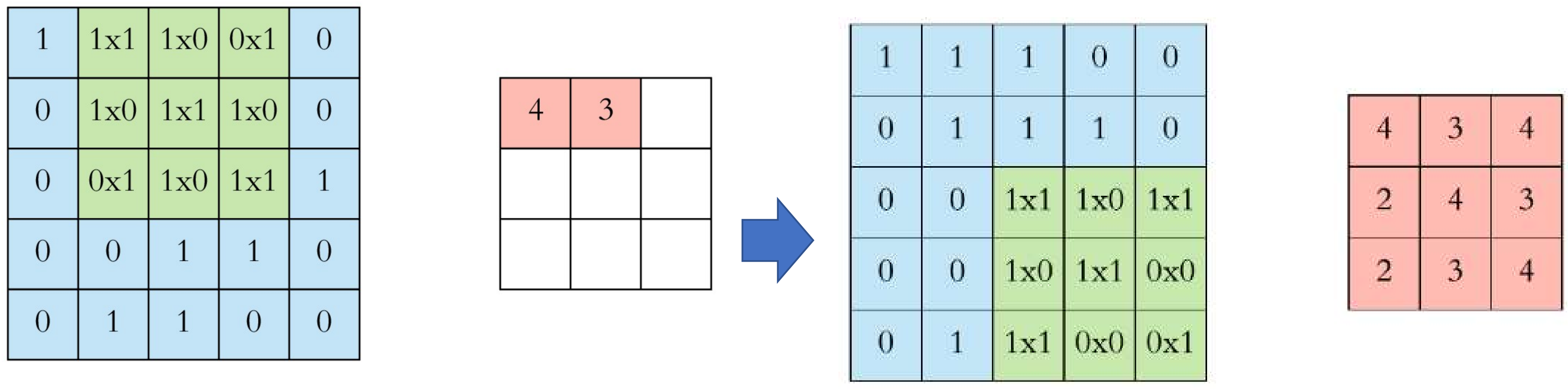
Input x Filter

4		

Feature Map

Convolution

- At every location, we do element-wise matrix multiplication & sum the result.
- Slide the filter to the right and perform the same operation.
- Continue same way and aggregate the convolution results in the feature map.

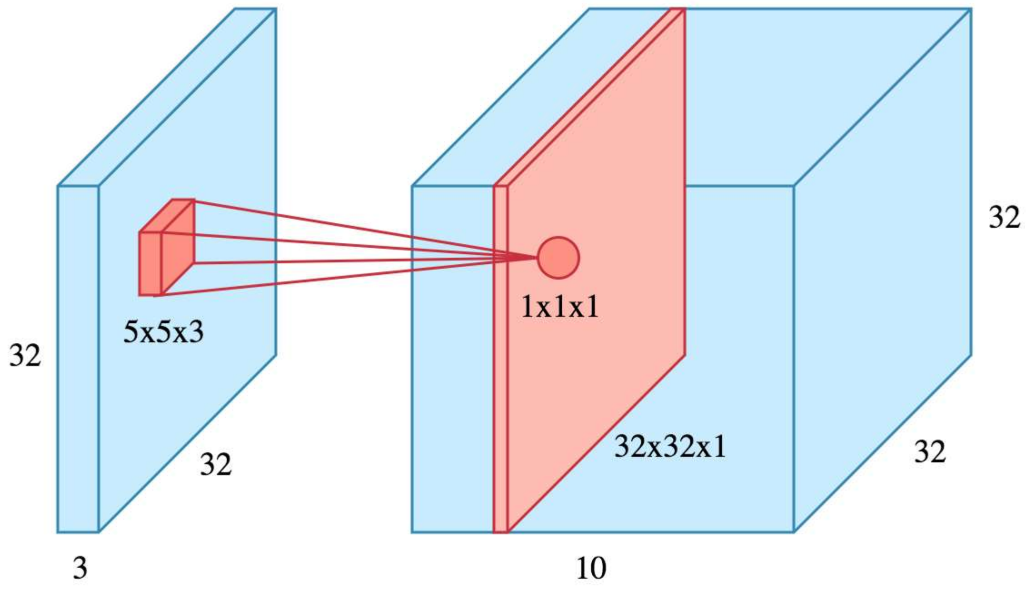


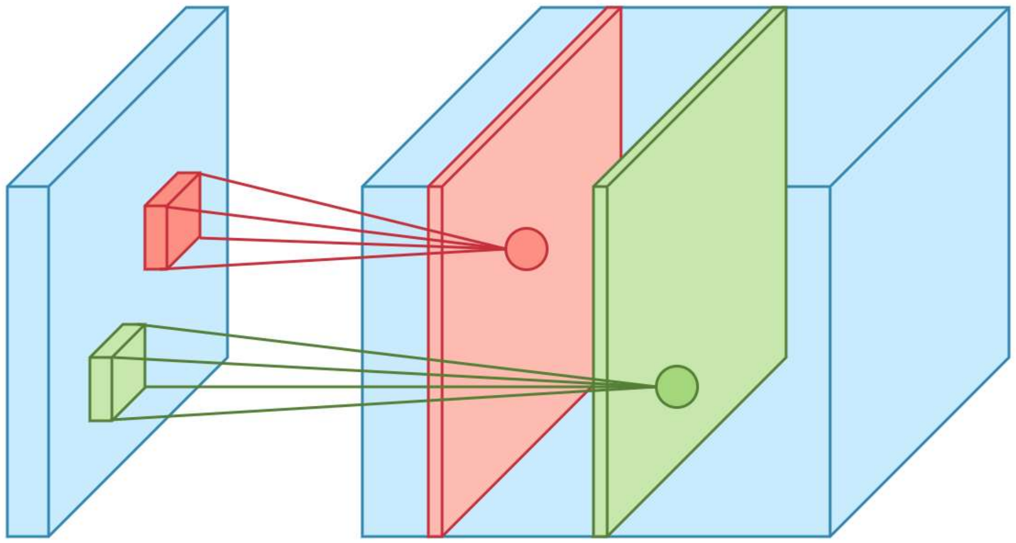
Input x Filter

Feature Map

3D Convolution

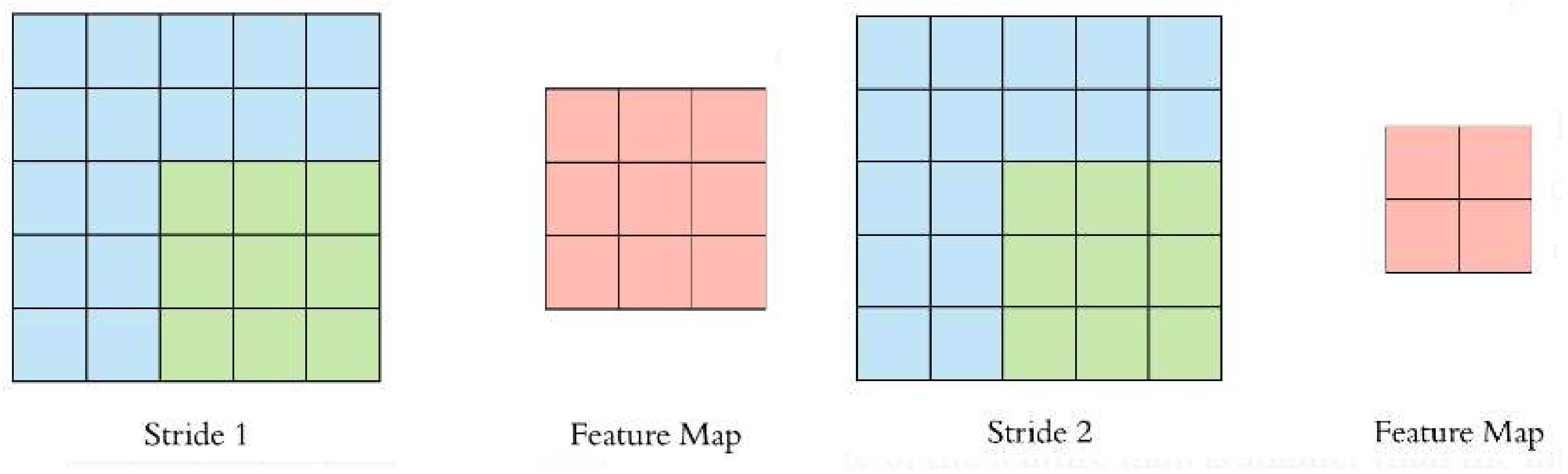
- Image is a 3D matrix with dimensions of height, width and depth (RGB).
- A convolution filter is 3D as well.
- Multiple convolutions are performed on an input, each using different filter and resulting in distinct feature map.
- All feature maps are stacked together as a final output of the convolution layer.





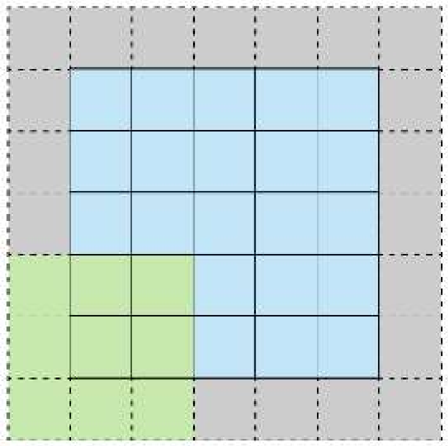
Stride

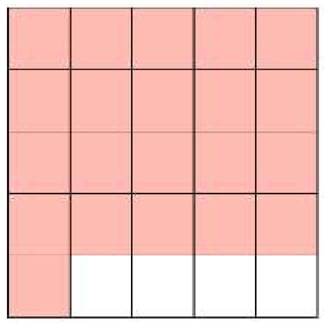
- Stride specifies how much we move the convolution filter at each step.
- Bigger stride are used for less overlap and resulting feature map is smaller.



Padding

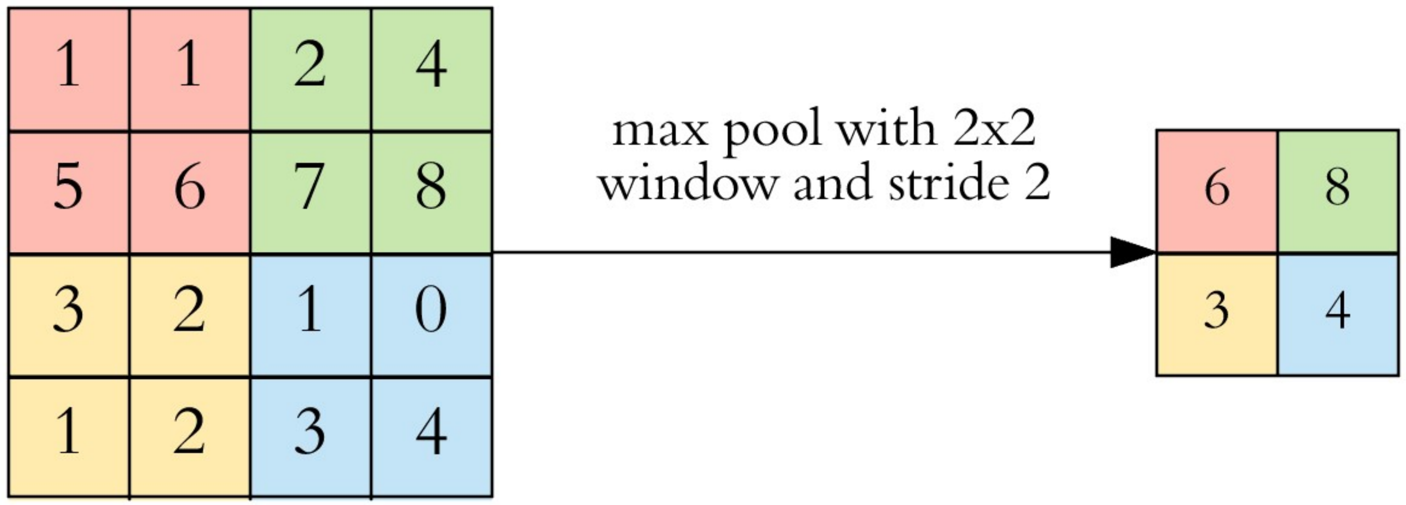
- To maintain the same dimensionality as input, use padding to surround the input with zeroes or values at the edge.
- Padding is commonly used to preserve the size of feature maps.


Stride 1 with Padding

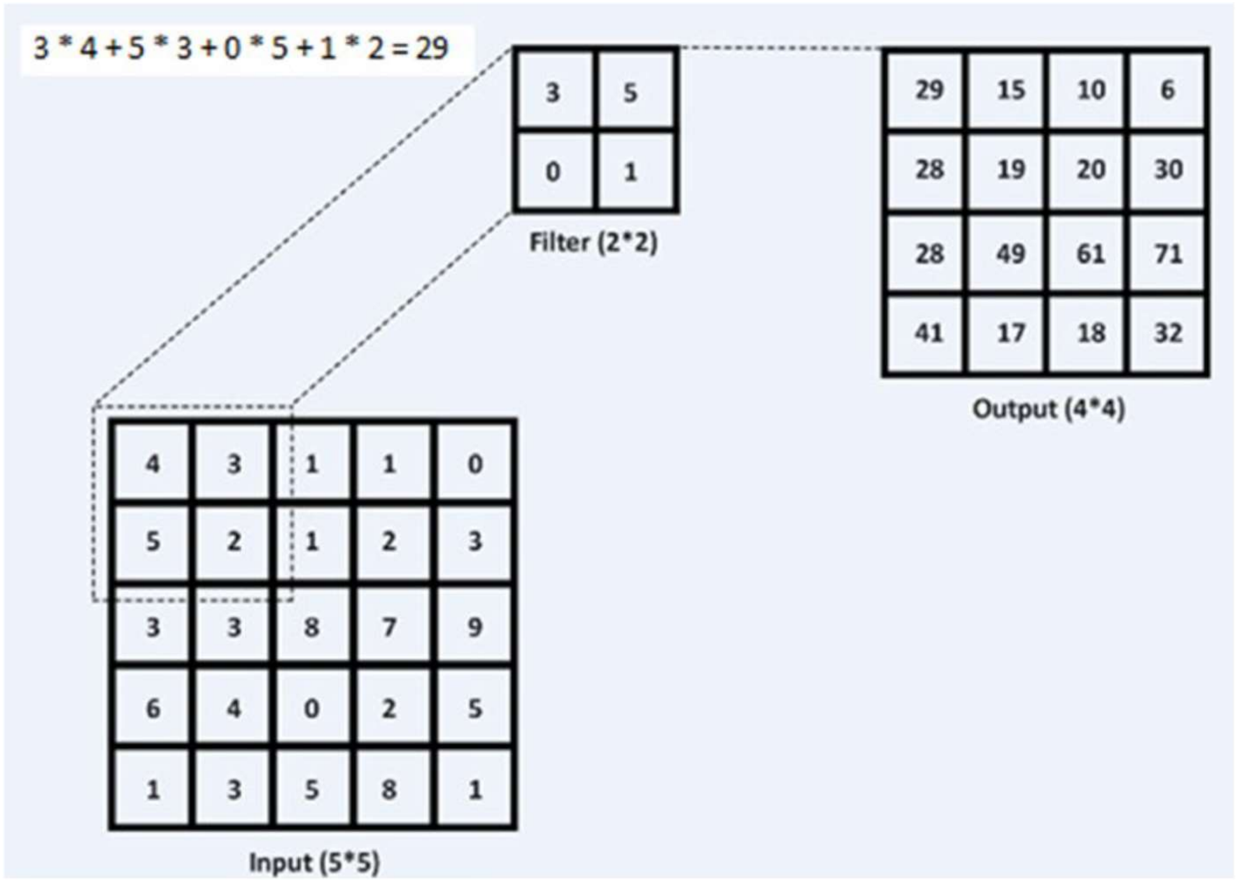

Feature Map

Pooling

- Pooling is used to reduce the dimensionality – thus reduce number of parameters, shorten training time etc.
- Pooling downsamples each feature map independently, reducing the height and width, keeping the depth intact.
- Most common type of pooling is max pooling.



Subsampling



- Input volume: 5x5x3 (width x height x number of channel)
- Six filters where each filter receives input from 2x2 pixels size of image
- Each filter will require 5 input weights (4 for pixels + 1 for bias)