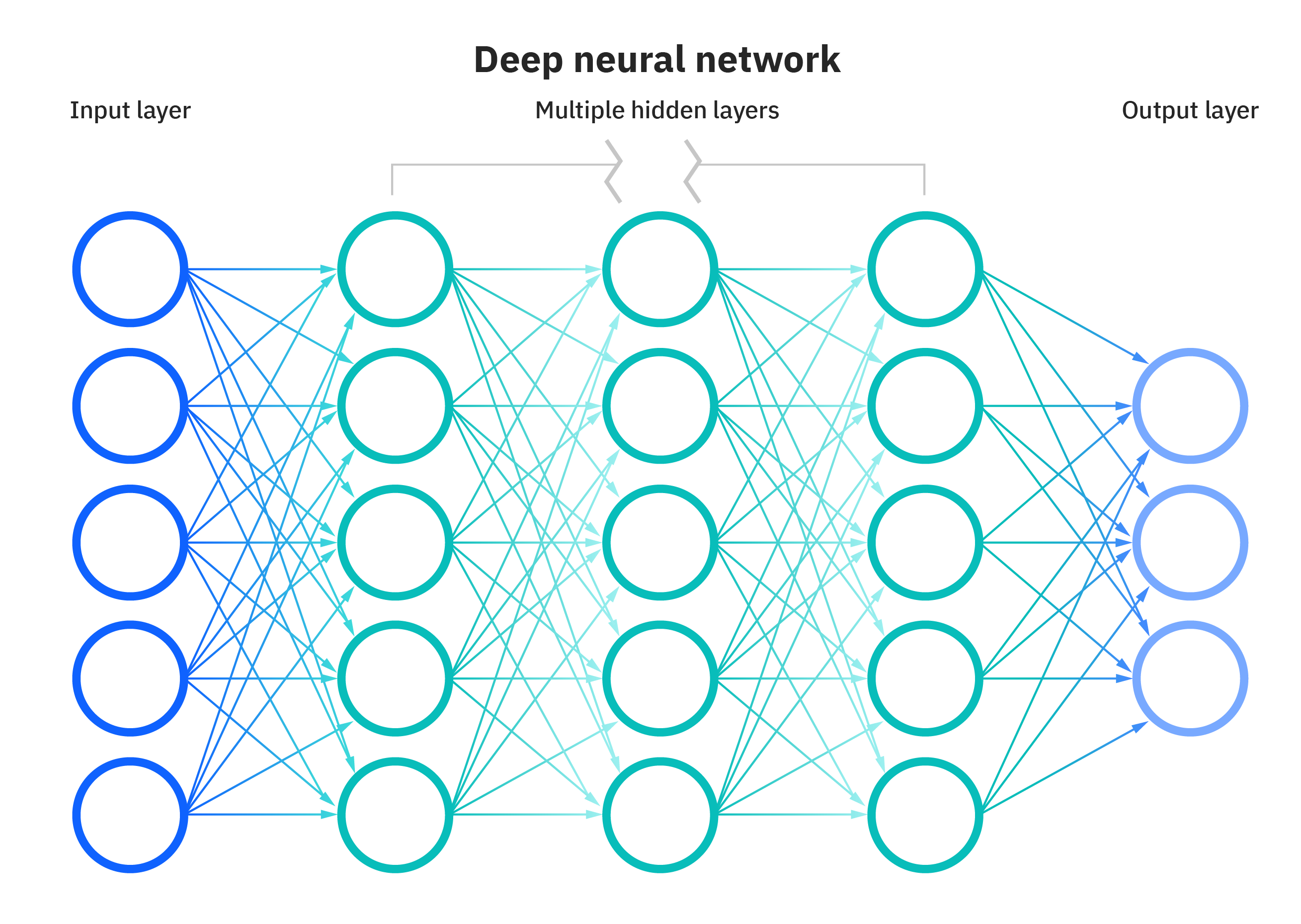
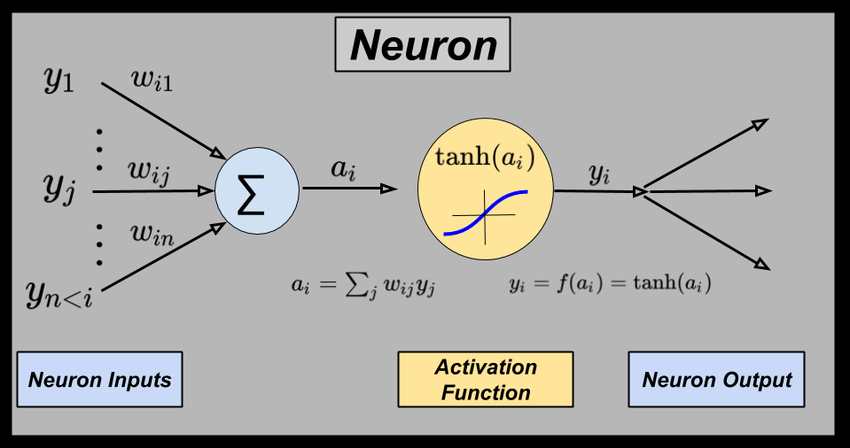
Neural Networks

Deep learning is a supervised ML algorithm that uses deep neural networks [nn with more than two layers] as its model. Artificial neural networks (ANNs) are comprised of neuron layers, containing an input layer, one or more hidden layers, and an output layer.



Each neuron takes in the sum of weighted output from the previous layer. This is the linear input from the previous layer. This is passed through an activation function (like ReLu or tanh) and is passed out as the output.



#### Important Terminologies

* Neuron, Layer-input,output,hidden, Weights, Biases,
* Feedforward NN: A NN where the data traverses in only one direction during inference
* Fully Connected Layer: A layer where each neuron is connected with every other neuron in the previous layer.
* Activation Function
* Forward and Backward propogation

#### Activation function

The **activation function** is a mathematical operation that is applied to the output of a neuron before it is passed on to the next neuron. The activation function determines how the neuron will respond to its inputs. It activates a neuron, based on a threshold. Some activation functions are:

1. Sigmoid function: **f(z) = 1/(1+e-z)**
2. Linear function : **f(z) = z.** (or ‘no function’).
3. ReLu (Rectified linear activation function): **f(z) = max(0,z).** It is preferred in hidden layers.

# Forward Propogation

As the input is passed from one layer to another, it is multiplied by the weight and added to bias. Then the whole thing is passed to an activation function before passed on to the next layer. The activation function determines if the neuron is activated for the next layer. Generally ReLu is used for hidden layer and sigmoid(or softmax) for the output layer (for classification).

For each layer L ,if g is the activation function,

fl(**X**) = g( **Wl.X** + bl)

So, the whole NN would look like

fnn(X) = f1(f2(f3(X))) # a NN is a nested function where each layer acts as a function

This whole process where the input propogates through the different layers to the output layer to perform inference is called **forward propogation**.

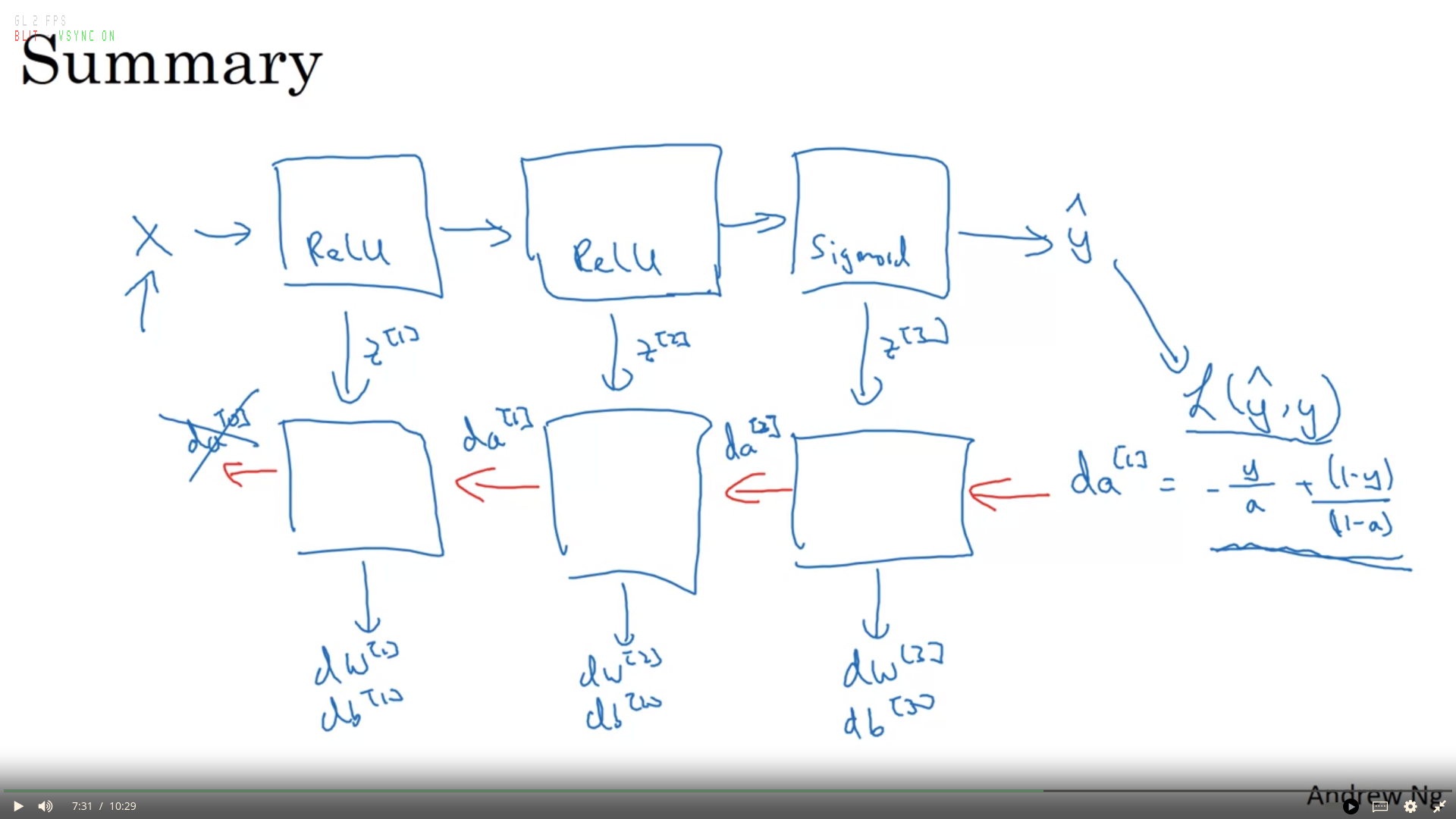
If we want to solve a regression or a classification problem discussed in previous chapters, the last (the rightmost) layer of a neural network usually contains only one unit. If the activation function of the last unit is linear, then the neural network is a regression model. If it is a logistic function, the neural network is a binary classification model.

# Backpropogation

Training is the process by which a neural network eats the data, and updates its parameters such that the cost function is reduced. **Backpropagation is the algorithm used to compute the gradients of the loss function with respect to the model's parameters**, and these gradients are subsequently used by gradient descent for parameter updates.

In neural networks, it is essential to initialize the weights and biases as random numbers, as vice versa will lead to equal gradients for all features. This lead to the no convergence, regardless of the number of epochs.

The backpropagation algorithm works as follows:

1. **Forward pass:** The network propagates the input data forward through the network, calculating the output of neural network. The error between the network's prediction and the target is used to form a loss function, and eventually a **cost function.   
   During actual implementation, we cache w[l], b[l] and z[l] for every layer, so that they are readily available for back propogation.( a[l] = g(z[l])**
2. **Backward pass:** This partial derivative of the cost function is calculated with respect to each parameter. The negative gradient of the cost function points in the direction of steepest descent, which helps to minimize the cost function.  
   During actual implementation,   
    At each layer **l**,  
    **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]**
3. **Optimisation:** The network's parameters are updated using the corresponding gradients. The learning rate, a hyperparameter, is multiplied by the gradient to determine the step size. This step size controls the rate at which the parameters are adjusted. One iteration of backpropagation involves updating the parameters based on the calculated gradients for the given batch of data. An epoch consists of multiple iterations, where each iteration updates the parameters using a different batch of data.

# Gradient Checking

It is a method for checking if gradient calculation is accurate or not. It can be used to verify if backpropogation implementation is correct or not.

# Exploding and Vanishing gradients

Exploding and vanishing gradients are issues that can occur during the training of neural networks, particularly those with many layers. These problems are related to the way gradients are propagated backward through the network during the backpropagation process. It is vastly prevalent in RNNs.

**Exploding Gradients:** When gradients grow exponentially as they are propagated backward through the layers of a deep neural network, it's referred to as the "exploding gradients" problem. As the gradients become extremely large, parameter updates can also become very large, causing the network's parameters to change significantly in each iteration. This can lead to instability in training, making it difficult for the network to converge to a good solution.

**Vanishing Gradients:** This often happens when the network has many layers, particularly in networks with activation functions that squash their inputs. As gradients become smaller and smaller, the updates to the parameters become insignificant, and the network learns at an extremely slow pace. This can lead to early layers of the network not learning effectively, resulting in poor overall performance.

One way to reduce this problem is to initialise the weights with a varience of 1/n (or 2/n).

# Types of Neural Networks

#### 

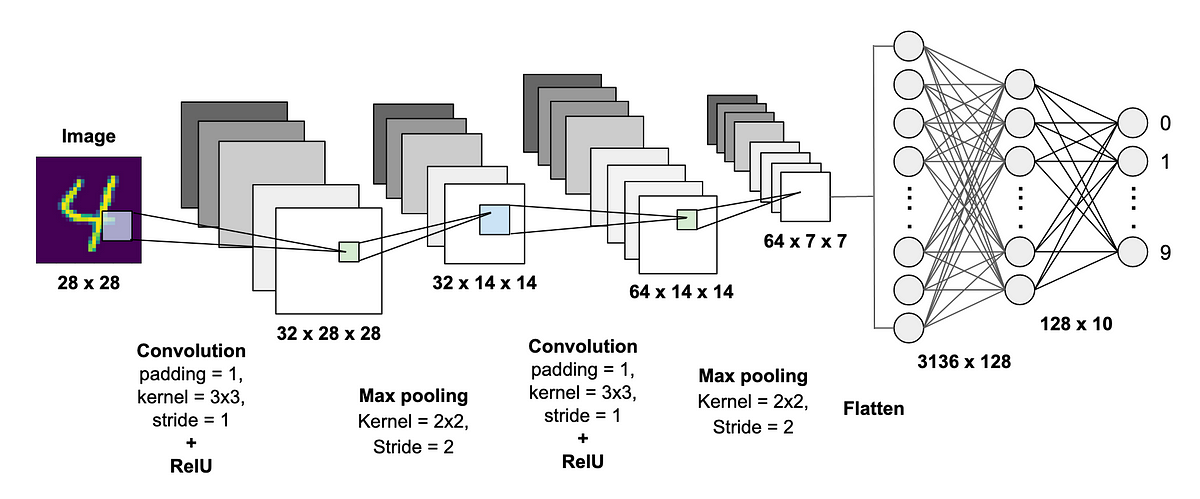
#### 1. Multilayer Perceptrons

They are nothing but normal **feedforward fully connected artificial neural networks**.So every neuron in one layer will be connected to every other neuron in the adjacent layer.

#### 2. Convolutional Neural Network

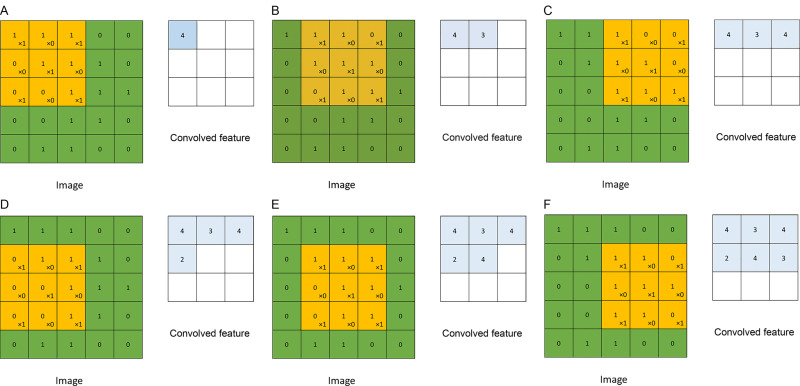
A convolutional neural network (CNN) is a special kind of feed forward NN that significantly reduces the number of parameters in a deep neural network with many units without losing too much in the quality of the model. It generally takes input in matrix form and was designed to work on image processing, as an image consists of block of information as a chunk.

A CNN typically has three layers: **convolutional layer**, **pooling layer**, and **fully connected layer**. These layers work together to learn patterns and edges in input data(images).



**Convolution Layer**

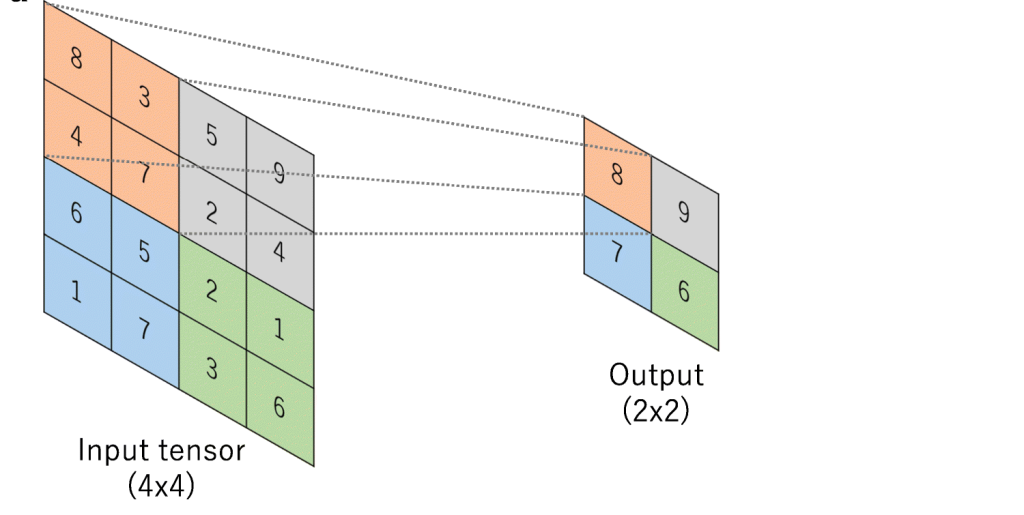
The convolution layer is the core building block of the CNN. It carries the main portion of the network’s computational load. This layer performs a dot product between two matrices, where one matrix is the set of learnable parameters otherwise known as a **kernel,** and the other matrix is the restricted portion of the input from the previous layer. The kernel is spatially smaller than an image but is more in-depth (to handle color channels).



During the forward pass, the kernel slides across the height and width of the image-producing the image representation of that receptive region. This produces a two-dimensional representation of the image known as an activation map that gives the response of the kernel at each spatial position of the image. The sliding size of the kernel is called a **stride**.

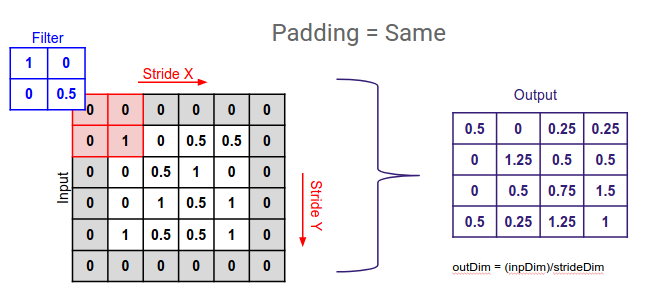
**The process of matrix multiplying the bigger matrix with the filter is called convolution.**

**Pooling Layer**

The pooling layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. **This helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights.** This vastly increases performance.The most popular pooling process is max pooling, which reports the maximum output from the neighborhood. Pooling layers reduce the spatial dimensions of the feature maps, helping to **decrease computational complexity** and control overfitting.

**Fully Connected Layer**

Neurons in this layer have full connectivity with all neurons in the preceding and succeeding layer as seen in regular FCNN. This is why it can be computed as usual by a matrix multiplication followed by a bias effect. The FC layer helps to map the representation between the input and the output.

****

**Padding**

Padding allows getting a larger output matrix,after covolution is performed. The cells added by padding usually contain zeroes.

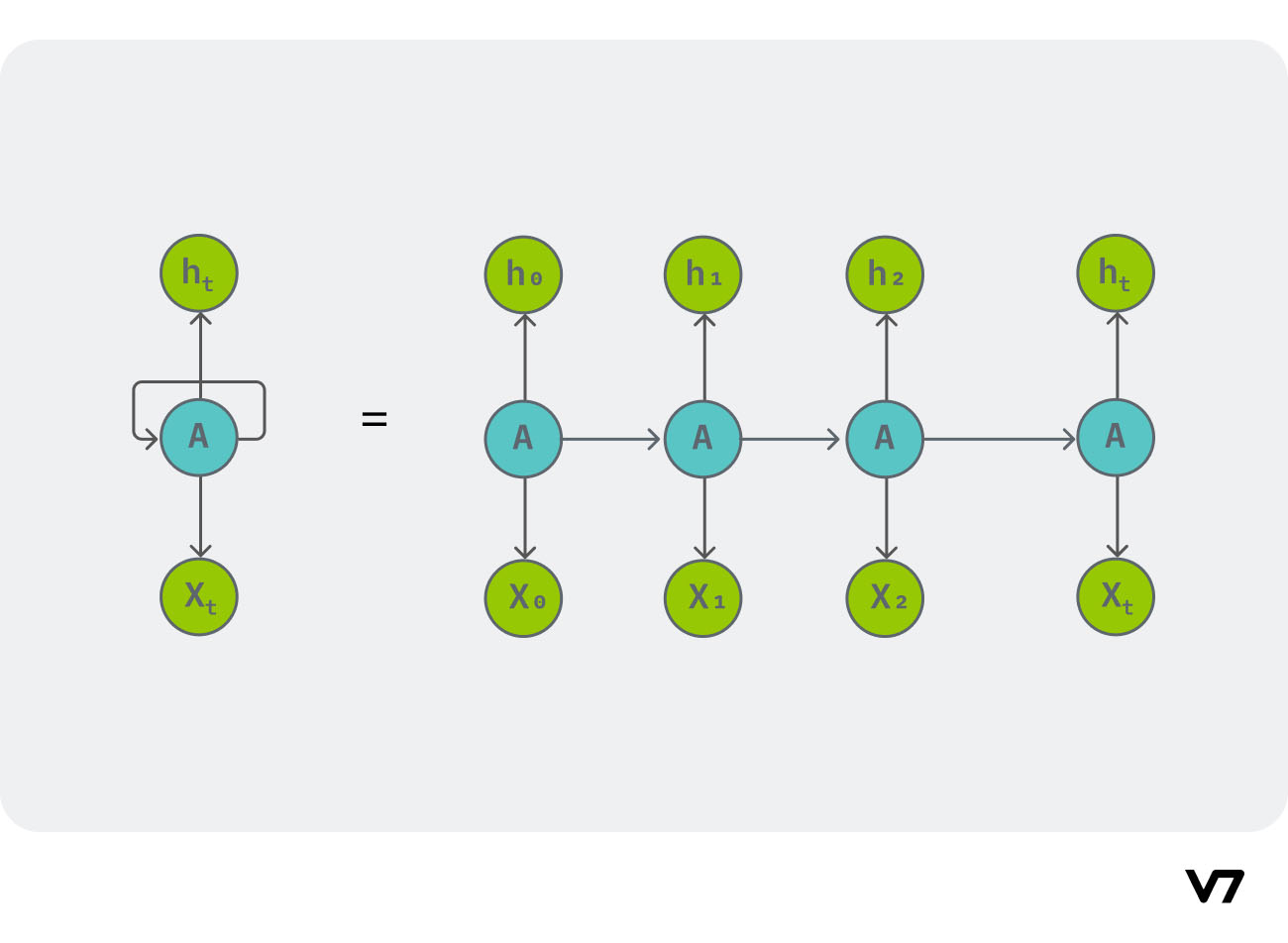
**Working of CNN on image data**

1. The input is consumed by the model as matrix. This matrix is composed of pixel values that correspond to the intensity of light at different positions in the image. This is passed into the CNN.
2. The CNN will have convolutional layer, which will perform convolution of the input with filter. This can be passed into pooling and fully connected layers. A CNN can have multiple of each layers. Usually, a pooling layer follows a convolution layer, and it gets the output of convolution as input.
3. The very last FC layer is often followed by a classification layer, which is typically a softmax layer for classification tasks. This layer computes the probability distribution over different classes based on the learned features from the preceding layers.

#### 3. Recurrent Neural Networks

RNNs are used to classify or generate **sequencial data** – data where the order of the data have importance. Also, RNNs are **able to handle (sequencial) input of varying length**. ie, it will perform even if there are 1000 data points or 20 data points for inference. Because of the sequencial nature, RNNs are used in text processing.

RNNs are able to handle sequencial data because they take in input as a sequential format, and not ‘all in one go’. RNNs are able to keep track of the sequence because of the **feedback loop** present in it - thus RNNs are not feed forward NNs. These feedback loops allow them to maintain a hidden state representing information from previous time steps in the sequence. Thus RNNs have memory, and can use that to process a sequence.



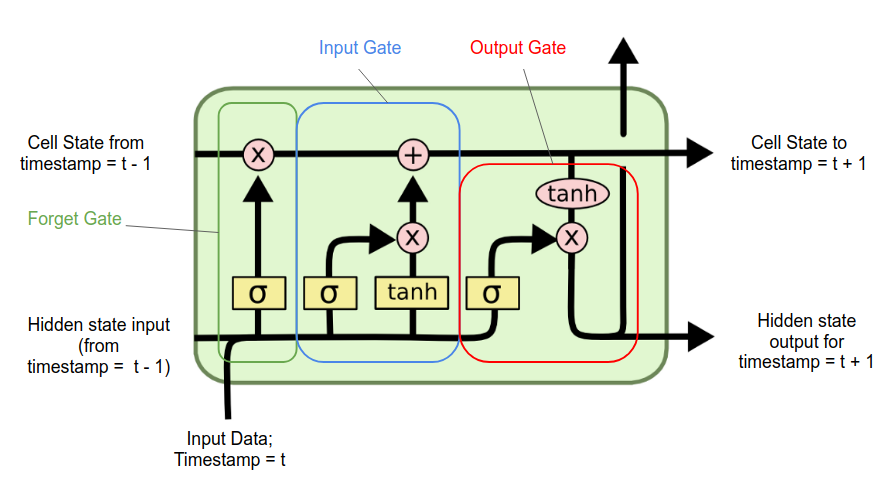
Each recurrent layer in an RNN takes two inputs – one is the input coming into the from the previous layer, and the other is the output of the same layer from the previous data. RNNs use a variation of backpropagation called **Backpropagation Through Time** (BPTT) to update the model's weights and biases. BPTT involves computing gradients through the entire sequence and updating the model parameters accordingly.

The major problems faced by RNNs are:

1. Exploding and vanishing gradients
2. As the length of the input sequence grows, the feature vectors from the beginning of the sequence tend to be “**forgotten**,” because the state of each unit, which serves as network’s memory, becomes significantly affected by the feature vectors read more recently.The short-term memory limitation of traditional recurrent neural networks is primarily due to the vanishing gradient problem.

**LSTM (Long Short-Term Memory) networks** were specifically designed to address both the exploding and vanishing gradient problems as well as the long-term dependencies issue, making them a significant improvement over vanilla RNNs in these aspects.

LSTM networks have two kinds of memory – **short and long term memory**. Each LSTM cell has three gates – the **forget gate**, the **input gate** and the **output gate**.



Long term memory

Short term memory

In each gate, the sigmoid activation function determines how much of the data is to be remembered.

**Forget gate**

This gate decides how much of the data is to be remembered in the long term memory. The forget gate takes in the weighted sum of the input vector from tth and (t-1)th state, and pass them through a sigmoid function. This gives an output between 0 and 1, which will be multiplied with the long term memory.

**Input gate**

It inserts new memory to the long term memory. It takes in the weighted sum of the input vector from tth and (t-1)th state, and pass them through a sigmoid function and a tanh function. Then those two values are multiplied and added to the long term memory.

**Output gate**

It gives out the output, derived from the input, long and short term memory which is to be passed to the next cell (or looped back in). It takes in the weighted sum of the input and short term memory and multiplies it with the tanh of the long term memory. This is the output, which is given out, as the short term memory of the next cell(or looped back in) and the y output from the cell.

**Read later:** http://colah.github.io/posts/2015-08-Understanding-LSTMs/

#### 4. Autoencoders

#### 5. Generative Adversarial Networks (GANs)

#### 6. Self-Attention Models