1. Sigmoid is added in the Neural Network to obtain non-linearity but remember sigmoid is not the only one. You have other non-linear activations available.
2. A single neuron with Sigmoid function as the activation function is Logistic Regression.
3. The actual loss function for perceptron is the Hinge-loss just like in an SVM.
4. Why ReLu is better than sigmoid?

Ans: The main aim of differentiating loss function is convergence.If we use sigmoid action unit the minimum differentiation it can reach is 0.25 and they will vanish(useless) in higher number of layers but whereas in RELU the convergence is quite quick compared to sigmoid AU. So ReLU is most preferred activation unit.

5) why we are differentiating the output instead of the function ?

Ans: Say y= f(x), then, dy/dx is same as df/dx. We are simply using the outputs O21 instead of the functions in this explanation to make it easier to follow as in many NNs, the activation function is the same for all neurons. This is a just convention we used to make it simpler to understand and follow.

6) As we have learnt, training a NN means computing the weights. Why a mathematical concept like derivative is preferred or chosen to compute these weights ? Why not some other mathematical concept ?

Ans: we are trying to find a minimum of a function so based on derivative we can get to know in which direction we have to go so that we will reach a minimum.

7) The word memoization comes from (memo) i.e. keeping note of the important items for reference so that it can be checked out and used when necessary.

Ans: Yes. Here in this technique, instead of computing the derivatives every time, it computes them only once and stores the values and when these derivatives are needed for further computations, the values will be retrieved easily.

8) In mini-batch SGD we randomly take 100 points out of 10000 points in first epoch than in the second epoch will there be a possibility of the some of same 100 points used in previous epoch to appear again or the points in the second epoch will completely be new?

Ans: Points in each iteration are completely new. For 10000 points and batch of size 100 each, we will iterate 100 times for each epoch and each iteration has different data points.

9) does mini batch affect the accuracy?

Ans: Algorithmically speaking, using larger mini-batches allows you to reduce the variance of your stochastic gradient updates, and this in turn allows you to take bigger step-sizes, which means the optimization algorithm will make progress faster. However, the amount of work done (in terms of number of gradient computations) to reach a certain accuracy in the objective will be the same. with a mini-batch size of n, the variance of the update direction will be reduced by a factor n, so the theory allows you to take step-sizes that are n times larger, so that a single step will take you roughly to the same accuracy as n steps of SGD with a mini-batch size of 1.

10) If we use Gradient Descent then the weights will be updated after every data point, right.  
But if we use mini-batch SGD with batch size say 100 points then, first all of the 100 points of the batch will be passed, the loss is accumulated then after that weights will be updated. My main confusion is that, in case of SGD we will not update the weights after each data points, we will update the weights after accumulating the loss of batch of points, right? Means weight will be updated only after when we pass all of the points in the batch and not after each data point in a batch.

Ans:Correct.

11) If we are performing batch wise training(batch stochastic gradient descent) we’d accumulate the loss over the given batch and then we’d perform backpropagation.

12) The purpose of the activation function is to introduce non-linearity into the network. Another way to think of it: without a non-linear activation function in the network, a NN, no matter how many layers it had, would behave just like a single-layer perceptron, (because linear functions added together just give you a linear function).

13) A reason for sigmoid function popularity in neural networks is because the sigmoid function satisfies a property between the derivative and itself such that it is  
computationally easy to perform. Derivatives of the sigmoid function are usually employed in learning algorithms.  
Derivative(sigma(z)) = sigma(z)(1- sigma(z))

14) In an artificial neural network algorithm, what happens if my learning rate is wrong (too high or too low)?

Ans: In addition to the difference in convergence time (small learning rate = slow convergence and vice versa), you can reduce the risk of getting stuck in a local minima by adopting a larger learning rate in the beginning and shrinking it over time. This is essentially the principle behind algorithms such as[Simulated Annealing](http://minds.jacobs-university.de/sites/default/files/uploads/teaching/share/KirkpatrickSimulatedAnnealing.pdf).

15) bias term is Neural network is different than bias with respect to error.  
bias term in NN – The bias node in a neural network is a node that is always ‘on’. That is, its value is set to 1 without regard for the data in a given pattern. It is analogous to the intercept in a regression model and serves the same function. it is used to shift the input function / be more flexible about the learned function.  
bias for a model – The bias is an error from erroneous assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting). we can analyze this effect for any model.

16) There are two notions of speed:  
1. Computational speed  
2. Speed of convergence of an algorithm  
Computational speed is simply the speed of performing numerical calculations and it is usually higher with a larger batch size. When batch size is too big , it’s can easy to overfitting your model. [Graph](https://qph.fs.quoracdn.net/main-qimg-85084b3b59ca170751dbb5c98ba49650.webp)shows the effect of the batch size on the validation accuracy of the model. One can easily see that the batch size, which contributes heavily in determining the learning parameters, will affect the prediction accuracy.

17) Why does big batch size lead to overfitting the model?

<https://stats.stackexchange.com/questions/164876/tradeoff-batch-size-vs-number-of-iterations-to-train-a-neural-network>

18) larger dropout => we are dropping out more edges/weights. The opposite of dropout-rate is called  
keep\_probability = 1 – dropout-rate. Most code-bases like Keras use keep\_probability instead of dropout rate as you will see in code examples later.

19) ReLU in itself cannot fully avoid exploding gradient problem. It is a combination of things that avoid exploding gradients: appropriate weight initialization(He initialization) + ReLU + gradient clipping + regularization (dropout/L1/L2/BatchNorm) + adaptive-learning-rates(ADAM like algos), we explain all in upcoming videos. ReLU just helps in the sense that it’s gradients are never greater than 1.

By the way, exploding gradients are fairly uncommon in MLPs. They do occur more often in RNNs that we discuss in a later chapter.

20) Firstly the main advantage of ReLU over sigmoid is that it solves the vanishing gradient problems and the brings sparsity in the weights.

21)

1. In case of ReLU, the gradient is 1 for values greater than 1 and 0 otherwise.  
2. In case of sigmoid the maximum value the gradient can take is 0.25.  
3. In case of tanh the maximum value a gradient can take is 1.0(but it is highly likely it can take other real values in between 1 and 0 as well).

Vanishing Gradient Problem, is an issue encountered during Training An Artificial Neural Network, Where training the parameters of early Layers of the Network using Back-Propagation become difficult(that means convergence takes a Long time) due to vanishing of Gradient. The convergence is delayed or does not happen at all. In simplest terms it is due to the repeated multiplication of gradient values in case of sigmoid or tanh which might lead to the learning becoming insignifiant. For example as you’ve stated that w’ = w – value dependent on z \* learning rate. At lower layers in general this “value dependent on z” is an extremely small number. Have a look at this to understand.

22) The purpose of the activation function is to introduce non linearity into the model which is possible by tanh,sigmoid as they are inherently non linear,but how does the ReLU function introduces non linearity as it is a linear function in positive value of z?

Ans: A function is said to be a linear function for every x and y if it follows the below property:  
f(x)+f(y)=f(x+y)  
1. ReLU function maybe piecewise linear in the regions [0,-inf] to [0, inf] but it definitely is not when you combine the 2 pieces as you can see below:  
f(-1) + f(1) != f(0) for ReLU  
2. Have a look at this example code where I’m trying to approximate a parabola (y = x^2) using ReLU. (It can be approximated to a good extent using ReLU(x)+ReLU(-x)+ReLU(2\*x-2)+ ReLU(-2\*x+2)+ReLU(3\*x-3)+ ReLU(-3\*x+3))  
code : <https://ideone.com/WbZfQ4>  
graphs : <https://imgur.com/VDTV3xF>  
Using this method you can approximate any complex non-linear function.

23) I understand that RELU is not linear when x is negative and positive.. but in positive plane the relu is always linear.. and in one of the link you provided in earlier videos it was mentioned that sigmoid is used because of its non linear properties and hence it can map any non linear dependency because of that property only.. then why is relu working better(from that perspective only)??

Ans: Two of the major benefits of using ReLUs are sparsity and a reduced likelihood of vanishing gradient.  
1. One major benefit is the reduced likelihood of the gradient to vanish. This arises when a>0. The gradient of ReLU has a constant value. In contrast, the gradient of sigmoids becomes increasingly small(the maximum being 0.25) as the absolute value of x increases. The constant gradient of ReLUs results in faster learning. For large values of x the gradient of sigmoid becomes significantly small as well.

2. The other benefit of ReLUs is sparsity. Sparsity arises when a≤0. The more such units that exist in a layer the more sparse the resulting representation. Sigmoids on the other hand are always likely to generate some non-zero value resulting in dense representations. Sparse representations seem to be more beneficial than dense representations.

24) For weights initialization why should we consider negative values too? They are going to lessen the z value(sometimes, the effective z value might become zero and if the algorithm is ReLU, it might be problematic). Instead why cant we take a normal distribution or uniform distribution of all positive values for weights?

Ans: Note that the input (x\_i’s) is standardized before being passed into a NN, which implies that some of the input-vectors values would -ve. If we now initialize the weights with only +ve weights, the chances of a dead ReLU would be more as we have some -ve negative values with all +ve weights. Hence, we initialize the weights to have both +ve and -ve weights randomly.

25) Weight initialization helps a lot in optimziation for deep-leanring. Without it, SGD and it’s variants would be much slower and tricky to converge to the optimal weights. While some theoritical arguments exist for it, there are no rigorous mathematical proofs for all. Some of thes initializations are inspired by empirical and experimental data.

26) He-initialization is often used for ReLU and Leaky ReLu units

27) Xavier/Glorot for tanh activation functions

28) When we are doing Batch Norm of inputs to layer 5(for.e.g.) do we use mean and variance parameter of only layer 4 outputs or across all the layers before?

Ans: The mean and variance of the layer 4 outputs which is the input to the layer 5.

29) how can we find that in front of which neuron we have to put a BN layer?

Ans: We typically place BN before later layers in a deep-NN as they have a higher chance of observing covariance shift. But, we got to manually experiment with various placements of BN layers and see if it helps the speed of convergence and overall performance. The big problem with designing the architecture of deep-NN’s is that we have only rules of thumb but we need lots of experimentation to figure out the best architecture. The architecture of a deep-NN is itself a very important hyper-parameter that needs lots of experimentation to figure out.

30) The original paper on BN suggests applying BN before an activation layer such as ReLU . But here is a excerpt from one of the discussions about having it after a non-linear layer:  
**“From a statistics point of view BN before activation does not make sense to me. BN is normalizing the distribution of features coming out of a convolution, some these features might be negative which will be truncated by a non-linearity like ReLU. If you normalize before activation you are including these negative values in the normalization immediately before culling them from the feature space. BN after activation will normalize the positive features without statistically biasing them with features that do not make it through to the next convolutional layer.”**

Some practitioners also suggest having it between a dropout layer and a non-linear layer . Regarding the number of BN layers – there is no thumb rule as such . Adding after every non-linear layer doesn’t hurt . In case of RNNs it is still an area of study . Try it between a non-linear and a drop out layer.

31) The input to each neuron should be standardized. Why do we have this requirement? Is it a hard and fast requirement?

Ans: By standardizing the inputs, we ensure that the distribution of inputs is kept the same over many epochs and iterations of the SGD. It is not a hard requirement, but this would ensure that we converge faster and a lack of standardization could hamper the speed and possibility of convergence especially when we have deep networks as explained in the above video.

Even in many experiments, BN has shown consistently to help us when we train deeper networks.

32) If we have a batch normalization layer before L5 then , it will be applied to the output of L4. You are getting confused between the batches and layers(we are not performing batch normalization on the input data , but difference in the distributions of batches slows down the training) . When we are sending datapoints in batches of 5 and let’s call them batch\_data1 and batch\_data2 . If the distributions of these batches are of different distributions then it causes internal covariate shift(slows down the convergence) . To increase the stability of a neural network, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation.

33) Batch norm is similar to dropout in the sense that it multiplies each hidden unit by a random value at each step of training. In this case, the random value is the standard deviation of all the hidden units in the minibatch. Because different examples are randomly chosen for inclusion in the minibatch at each step, the standard deviation randomly fluctuates.

Batch norm also subtracts a random value (the mean of the minibatch) from each hidden unit at each step.

Both of these sources of noise mean that every layer has to learn to be robust to a lot of variation in its input, just like with dropout

34) If you see carefully you’ll observe that in batch normalization you calculate the mean and the variance initially .  
1. xi’ = [xi – mean(x)]/var(x) -> Here you perform your standardization  
2. Then you do shifting and scaling on the standardized batch using Xnorm = xi’\*gamma + beta . where Xnorm is the normalized data point, gamma and beta are the learning rates.

35) When we have an internal covariance shift, an activation unit sees a very different set of inputs in each epoch. If inputs are significantly different from one epoch to another, the weights in that layer that are updated based on these inputs through SGD would not converge fast to optimal values. So, BN typically helps in faster convergence and hence faster training.

36)Converse function local minima = global minima

ex. Squared loss.

37)Saddle Point: A point of a function or surface which is a stationary point but not an extremum.

38)Inflection Point: An inflection point is a point on a curve at which the sign of the curvature (i.e., the concavity) changes.

39) Can you clarify what differentiates local minima from a saddle point? as both of them have tangent of slope=0 .

Is it like, after local minima point the value of loss function increases whereas after a saddle point again the loss function decreases (or) Is any other reasons to differentiate them?

Ans: Saddlepoint is minimax point. derivative at saddle point is zero but it is not a local extremum. you can check whether it is a local extremum or saddle point based on second order partial derivative test.  
let’s consider function x^2 – y^2. at (0,0) derivative is zero but when you just move in the x-direction around (0,0), the function looks like f(x, 0) = x^2 – 0^2 = x^2, f(x) = x^2 has local minima at x = 0.  
When you just move in the y-direction around (0,0), the function looks like f(0, y) = 0^2 – y^2 = -y^2, f(y) = -y^2 has local maxima at y = 0. the x and y-directions disagree over whether this input should be a maximum or a minimum point. These type of points called saddle points

40) Contours can represent either minima or maxima.Basically we get a number for each contour.If number increases from outer contour to inner contour we say it is local maxima and if number decreases from outer contour to inner contour we say it is local minima. contour plots helps to find local minima and maxima but from contour plots we can’t say which one is extremum points

41) Say you have a dataset of 10 examples. You have a batch size of 2, and you’ve specified you want the algorithm to run for 3 epochs.  
Therefore, in each epoch, you have 5 batches (10/2 = 5). Each batch gets passed through the algorithm, therefore you have 5 iterations per epoch. Since you’ve specified 3 epochs, you have a total of 15 iterations (5\*3 = 15) for training.

2. If batch size=n then iteration = epoch

42) Why SGD is not moving in the perfect direction in contours?

Ans: he zig zag nature we see in SGD or mini-batch SGD is because the derivative we get from SGD or mini-batch SGD is an approximation to what we would have got with GD

43) In a nutshell, SGD is stochastically approximating the gradient that would be generated using GD. Which is faster depends a lot on the size of data. As you rightly pointed out, using GD would require fewer iterations than using SGD. But each iteration in SGD is faster as we are only using a subset of points.

SGD takes more steps/iterations as it is not moving in the perfect direction every time like GD as SGD is appropriating the gradient while GD computes the exact gradient. Moving in the negativ direction of the exact gradient is the fastest way to converge.

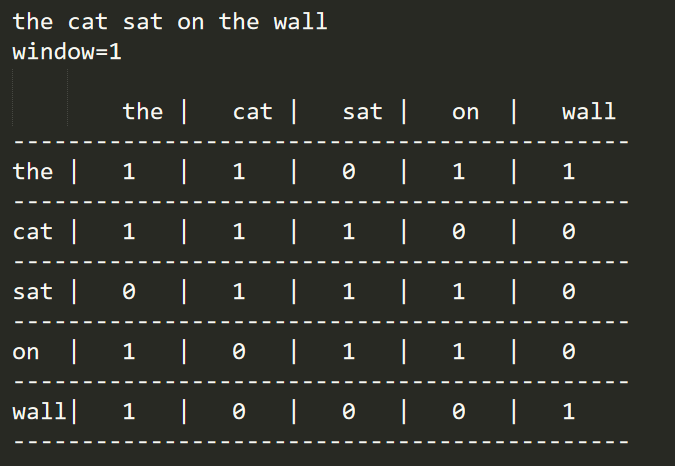
## 44)Purpose of Auto Encoders

An autoencoder learns to compress data from the input layer into a short code, and then uncompress that code into something that closely matches the original data. This forces the autoencoder to engage in dimensionality reduction, for example by learning how to ignore noise. Some architectures use stacked sparse autoencoder layers for image recognition. The first encoding layer might learn to encode easy features like corners, the second to analyze the first layer's output and then encode less local features like the tip of a nose, the third might encode a whole nose, etc., until the final encoding layer encodes the whole image into a code that matches (for example) the concept of "cat".[[5]](https://en.wikipedia.org/wiki/Autoencoder#cite_note-domingos-5) The decoding layers will learn to decode the learnt code representation back into its original form as close as possible. An alternative use is as a generative model: for example, if a system is manually fed the codes it has learned for "cat" and "flying", it may attempt to generate an image of a flying cat, even if it has never seen a flying cat before

45)Summary on CBOW:

1. Suppose we have n unique words in text corpus (example : Amazon Fine Food Review) then we will have n focus words.  
2. If we set context words to 5 then it will take 5 adjacent words to the focus words.  
3. Now each context word will be represented by v-dimension where v is the length of the set of all words therefore in our case v is n (number of unique words).  
4. Suppose we pass 5 Context words of n dimension to N dimensional Hidden Layer then output it will generate N X n dimensional representation matrix where N is the number of activation function in hidden layer and n is the number of unique words.  
5. when this N x n dimensional matrix is passed to Softmax Layer it will predict the focus word from n unique words.  
6. This means if we have n unique words in the dataset then we will have n class classification problem

46) In CBOW, suppose in the sentence ” the cat sat on the wall” i have to convert this context word “cat” into a binary vector , given vocabulary V=[“the”,”cat”,”sat”,”on”,”the”,”wall”]. so how will the vector form of context word “cat” will be?

Ans: 

the is a co-occurrence matrix, each cell(i,j) value represents the number of occurrences that word i and word j occurred within the window size i.e |index(i)-index(j)|<=window.

Considering above example, we have to give below inputs  
the (Context\_1)=> [1,1,0,1,1]  
cat (Context\_2)=> [1,1,1,0,0]  
on (Context\_3)=> [1,0,1,1,0]  
the (Context\_4)=> [1,1,0,1,1]  
wall (Context\_5)=> [1,0,0,0,1]

below output vector will act as focus  
sat (Out\_1)=> [0,1,1,1,0]

In the above pic in first row, ‘the cat’ occurs once so -🡪1 , ‘the sat’ never occurs combine 🡪0 , ‘the on’ once 🡪1,’the wall’ once 🡪1.

47) The physical meaning of convolution is nothing but “manipulation of pixels” (in a number of ways that you choose to). Essentially a kernel (i.e. a mask/window) is CONVOLVED with the image in order to accomplish tasks such as smoothing (blurring), sharpening(highlight sharp transitions), edge detection, etc. The pixel manipulation depends on the matrix(kernel matrix/convolution matrix) that you choose.

For example – if you use a unit matrix of size 3×3 [1 1 1; 1 1 1; 1 1 1] , known as box filter(equal coefficients), it blurs the image. Perform convolution and divide the result by the sum of coefficients of the box filter. You’ll observe that your original image’s pixel values are replaced with an average value which causes the blurring.

48) the direct matrix multiplication between kernel and image is called correlation

49)Convolution is something like dot product.

50) In CNN “valid” means “no padding”. “same” results in padding the input such that the output has the same length as the original input.

51) Some reasons why padding is important:

1. It’s easier to design networks if we preserve the height and width and don’t have to worry too much about tensor dimensions when going from one layer to another because dimensions will just “work”.  
2. It allows us to design deeper networks. Without padding, reduction in volume size would reduce too quickly. (with increasing the stride reduction in volume is much more apparent)  
3. Padding actually improves performance by keeping information at the borders.

52) Relate kernels to weights of MLP. And, It’s best possible values should find out on backpropagation like that we have done in the MLP.

53) Suppose for an RGB image of nxnx3 size, if we have 10 kernels, depending on padding and stride, we get output image of nxnx10 size. Now the number of channels for the output image is different than the input image. So I need to understand how are input and output images now related? It would be helpful if you provide some link where we can visualize such outputs and see what the final image represents and how is it different from the input image.

Ans:  Each kernel detects different types of features of our input image.

54) Why are we applying multiple level of convolution? At first convolution layer we are already using multiple (m) kernels. We can get all types of kernel in that layer only. Please elaborate?

Ans: Because of the way convolution works. As we have shown, convolution-operator can learn edges and simple-patterns very easily on raw-images. Hence, the first layer learns edges. The next layers build on top of the first layers slowly building more complicated features with each new layer. Convolutions filters are generally stacked with an increasing number of filters in each layer. Each successive layer can have two to four times the number of filters in the previous layer. This helps the network learn hierarchical features.

55) convolution-operator can learn edges and simple-patterns very easily on raw-images. Hence, the first layer learns edges. The next layers build on top of the first layers slowly building more complicated features with each new layer.

56) I am not be able to understand that how ReLU units are applied. I mean that after convolution, an image is generated, then how that image is given as an input to a ReLU unit/s?

Ans: In the convolution operation, we multiply cell-wise a part of the input image and the kernel/filter to generate one-pixel/cell value in the output image. We apply ReLu to the output of the convolution operator before we place it in a cell/pixel of the output image. So, we perform: ReLU( Convolution(input-image, kernel) )

57) The only difference between MLP and Convolution in terms of the optimization and weights part is that in MLP, we have vector multiplication as W.X after which we apply ReLu as ReLu(W.X). In convolution, we perform matrix-matrix convolution followed by ReLU as ReLU(Conv(Kernel, X)) and all the wights in the kernel are parameters which will be learned through optimization just like W is learned in MLP through optimization. Except that all of the math of back-prop is exactly the same.

58) Let say we have an input image of size 16\*16. Then we applied convolution over it with padding = same and Strides = 1 with kernel size 4\*4. Now this convolution operation will generate an output matrix of size 16\*16 where each cell in a matrix is a pixel which is nothing but a real number. Let’s call this output matrix as X. Now 16\*16=256, so it means there are 256 numbers in matrix X. Now as per your reply what I have understood is that each of these 256 numbers of matrix X will pass through a single ReLU unit one by one and finally generate an output image of same size which is 16\*16, right?  
In a nutshell, after every convolution operation a matrix is generated and each cell value of that matrix pass through a single ReLU unit one-by-one and finally generate an output image of same size, here it is 16\*16

59) each cell value of a kernel/filter matrix is initialized in the same way as weights are initialized in MLP from he-normal or Xavier/Glorot initialization. In CNN also we use he-normal or Xavier/Glorot initializations for initializing cell values of kernel/filter matrix. And we keep on updating these kernel cell values using back-propagation. In short we learn kernel cell values similarly like weights in MLP

60) just like in brain every neuron is detecting some specific angle edges, for example, neuron 1 is detecting horizontal edge, neuron 2 is detecting vertical edge, neuron 3 is detecting 45 degree angle edge etc. We are taking ‘m’ kernels here, so will each kernel also similarly detects specific angle edges for example, kernel 1 is detects horizontal edge, kernel 2 is detects vertical edge, kernel 3 is detects 45 degree angle edge and so on.

61) Padding is used to increase the size of output image, whereas stride is used to decrease the size of output image. Padding and Stride perform contradictory operations. So, it is necessary to use padding while we apply stride length?

Ans: Some reasons why padding is important:  
1. It’s easier to design networks if we preserve the height and width and don’t have to worry too much about tensor dimensions when going from one layer to another because dimensions will just “work”.  
2. It allows us to design deeper networks. Without padding, reduction in volume size would reduce too quickly. (with increasing the stride reduction in volume is much more apparent)  
3. Padding actually improves performance by keeping information at the borders.

So if you want to design a network with deeper layers it would be advisable not to use stride and use padding in subsequent convolution layers. I’d suggest you take a look at this: <https://stats.stackexchange.com/questions/246512/convolutional-layers-to-pad-or-not-to-pad>.

62) Does pooling retain the number of channels ? for e.g if the image dimensions are say 224\*224\*32 after convolution…would max pooling still keep the 32 channels?

Ans: it only reduces the spatial size, not channels.

63) The reason as to why you apply an activation function is to introduce non-linearity. They basically decide whether a neuron should be activated or not. Whether the information that the neuron is receiving is relevant for the given information or should it be ignored. So RelU helps achieve that and that’s the reason we use it.

<https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f>

64) [ReLU is not linear](https://www.quora.com/Why-is-ReLU-non-linear). The simple answer is that *ReLU* output is not a straight line, it bends at the x-axis. The more interesting point is what’s the consequence of this non-linearity. In simple terms, linear functions allow you to dissect the feature plane using a straight line. But with the non-linearity of *ReLU*s, you can build arbitrary shaped curves on the feature plane.

65) In mathematics a function is considered linear whenever a fucntion f:A→Bf:A→B if for every xx and yy in the domain AA has the following property: f(x)+f(y)=f(x+y)f(x)+f(y)=f(x+y). By definition the ReLU is max(0,x)max(0,x). Therefore, if we split the domain from (−∞,0](−∞,0] or [0,∞)[0,∞) then the function is linear. However, it's easy to see that f(−1)+f(1)≠f(0)f(−1)+f(1)≠f(0). Hence by definition ReLU is not linear.

66) Max-Pooling adds invariance to small translations and deformations of the object. To to this you’ll have to lose the precision on where the feature occurs . This makes the neural net robust to distortions in the object . Considering max of pixels in a region (max pooling) is one of the ways to check the presence of a feature in a region and not a precise point . Experimentations have yielded better results while using max pooling and hence it remains the popular pooling method. There seems to be new literature on substituting pooling with adjusting stride – <https://arxiv.org/abs/1412.6806> which seems to yield good results as well.

67) max pooling solves location invariant problem. What about scale invariant and rotation invariant problems?

Ans: The overall architecture of the CNN itself wherein we apply small-sized filters/kernels on the input image and gradually apply filters/kernels, again and again along with pooling, would take care of the scale-invariance as this is what most [image-pyramids](https://en.wikipedia.org/wiki/Pyramid_(image_processing)) do intuitively.

Additionally, note that we generate new data from original data through data augmentation [[video](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/data-augmentation/)] when we train most CNN models. Some of the most popular transformation in data augmentation is rotation, sheer etc. Data Augmentation takes care of most invariances as explained in this [video](https://www.appliedaicourse.com/course/applied-ai-course-online/lessons/data-augmentation/) in detail

68) we typically max-pool on the output of the convolution-layer.

69) difference between Convolution Layer and Fully Connected Layer?

Ans: 1. In a FC layer each neuron is connected to every neuron in the previous layer, and each connection has it’s own weight. This is a totally general purpose connection pattern and makes no assumptions about the features in the data.  
2. In contrast, in a convolutional layer, each neuron is only connected to a few nearby (i.e. local) neurons in the previous layer, and the same set of weights (and local connection layout) is used for every neuron. This connection pattern only makes sense for cases where the data can be interpreted as spatial with the features to be extracted being spatially local and equally likely to occur at any input position. The typical use case for convolutional layers is for image data where, as required, the features are local, and equally likely to occur anywhere.  
I’d suggest you to take a look at this: <https://stats.stackexchange.com/questions/312838/difference-between-conv-and-fc-layers>.

70) When augmentation is taking care of the rotation,shear,noise etc why are we again introducing Max Pooling  
Does pooling takes care of things which are missed by us.  
What is exact difference betwwen ppooling and Augmentation

Ans: Both of them try to achieve location, scale and rotation invariance by two different strategies. Max-pooling is achieving this using pooling operations and data-augmentation explicitly addss new data points to force the CNN model to learn these invariances. Very often, we use both of these strategies together to achieve better results than using just one.

This is just like dropouts and L2 regularization being two methods using which we can regularize the network which can also work together to achieve good results.

71) What is the purpose of dense layers in CNNs. Is it to reduce the dimensionality of inputs to final softmax layer eg. from 65K to 128 ? What if we just flatten the convolutional output and pass it to softmax ?

Ans: Multiple Dense layers enable non-linearity between the output of the convolution layers and the final soft-max classifier. If we input the output of the convolutional-layers directly into a soft-max, we get a simple linear model on the conv-layers-output. If we want to build a non-linear model on top of the conv-layers-output, it is best to use multiple dense layers. But, we also need to add some form of regularization like dropout here as it is very easy to overfit in the dense layers.

72)After convolution layer flatten we are having fully connected layers in CNN similar to MLP we are feeding all features into hidden layers.So,basically we CNN itself does feature engineering using kernels and passes that features into hidden layers(FC).

73) As you pointed out Fully Connected layers can be considered like MLP’s, so that means a fully connected layer here has 4096 neurons and each is connected to the input 7\*7\*512 which is produced that the last max pool operation.  
and this input is first flattened (7\*7\*512 = 25k) and then connected to these 4096 neurons ?

Ans: That is correct. So the number of weights are 25k \* 4096 essentially

74) What is the major difference between Alexnet and Vggnet (as that vgg is performing better than alex )?

Ans: 1. In AlexNet as we can notice the first layer has a filter of size 11×11 and the second layer has a 5×5 filter and there is max pooling after every convolutional layer. The size of filters is purely experimental.  
2. In VGG Net all the convolution kernels are of size 3×3. MaxPooling is done after 2 or 3 layers of convolutions.

75) If our dataset is significantly different from Imagenet and if we have tons of data, it is best to train the model from scratch.

But in many instances, the first few layers of a model trained on imagenet might still be useful as the first few layers are simply detecting basic edges and shapes which are common in most images we encounter.

76) I have one question in the fine tuning part. It is been said that we need to freeze the top 25 layers. I want to know how we got that number 25 because when i go through the diagram of VGG16 upto 5 convolutional layers we dont have 25 layers it is less than 25 layers. So on what basis we get to know to freeze 25 layers ? (refer code below)

# set the first 25 layers (up to the last conv block)  
# to non-trainable (weights will not be updated)  
for layer in model.layers[:25]:  
layer.trainable = False

# compile the model with a SGD/momentum optimizer  
# and a very slow learning rate.  
model.compile(loss=’binary\_crossentropy’,  
optimizer=optimizers.SGD(lr=1e-4, momentum=0.9),  
metrics=[‘accuracy’])

Ans. Actually, VGG-16 does NOT 25 layers as per the VGG research paper. But, in the Keras [implementation](https://gist.github.com/baraldilorenzo/07d7802847aaad0a35d3#file-vgg-16_keras-py), some of the components of a single layer in the research paper are broken into multiple logical layers like convolution and padding are taken two different layers. This is an implementation choice made by Keras.

The Keras code for VGG16 is as follows:  
def VGG\_16(weights\_path=None):model = Sequential()model.add(ZeroPadding2D((1,1),input\_shape=(3,224,224)))model.add(Convolution2D(64, 3, 3, activation=’relu’))model.add(ZeroPadding2D((1,1)))model.add(Convolution2D(64, 3, 3, activation=’relu’))model.add(MaxPooling2D((2,2), strides=(2,2)))

*model.add(ZeroPadding2D((1,1)))  
model.add(Convolution2D(128, 3, 3, activation=’relu’))  
model.add(ZeroPadding2D((1,1)))  
model.add(Convolution2D(128, 3, 3, activation=’relu’))  
model.add(MaxPooling2D((2,2), strides=(2,2)))*

*model.add(ZeroPadding2D((1,1)))  
model.add(Convolution2D(256, 3, 3, activation=’relu’))  
model.add(ZeroPadding2D((1,1)))  
model.add(Convolution2D(256, 3, 3, activation=’relu’))  
model.add(ZeroPadding2D((1,1)))  
model.add(Convolution2D(256, 3, 3, activation=’relu’))  
model.add(MaxPooling2D((2,2), strides=(2,2)))*

*model.add(ZeroPadding2D((1,1)))  
model.add(Convolution2D(512, 3, 3, activation=’relu’))  
model.add(ZeroPadding2D((1,1)))  
model.add(Convolution2D(512, 3, 3, activation=’relu’))  
model.add(ZeroPadding2D((1,1)))  
model.add(Convolution2D(512, 3, 3, activation=’relu’))  
model.add(MaxPooling2D((2,2), strides=(2,2)))*

*model.add(ZeroPadding2D((1,1)))  
model.add(Convolution2D(512, 3, 3, activation=’relu’))  
model.add(ZeroPadding2D((1,1)))  
model.add(Convolution2D(512, 3, 3, activation=’relu’))  
model.add(ZeroPadding2D((1,1)))  
model.add(Convolution2D(512, 3, 3, activation=’relu’))  
model.add(MaxPooling2D((2,2), strides=(2,2)))*

*model.add(Flatten())  
model.add(Dense(4096, activation=’relu’))  
model.add(Dropout(0.5))  
model.add(Dense(4096, activation=’relu’))  
model.add(Dropout(0.5))  
model.add(Dense(1000, activation=’softmax’))*

*if weights\_path:  
model.load\_weights(weights\_path)*

return model

Now in the above definition of VGG16 in Keras, top 25 layers imply all the layers till the last conv-block. Note that the code shown in the video uses this definition of VGG16 in Keras and hence you see the top-25 layers being frozen.

77) One of the best real-time object detection systems in a video is YOLO (you-only-look-once). There are multiple version of the YOLO algorithm which has become the standard algorithm for real-time object detection in the recent past.

If you have gone through our deep-learning chapters, you can easily understand the YOLO algorithm. Here are a few references and code-examples for YOLO:

1. <https://towardsdatascience.com/yolo-you-only-look-once-real-time-object-detection-explained-492dc9230006>  
2. Keras: <https://github.com/experiencor/keras-yolo2>  
3. TensofrFlow: <https://www.youtube.com/watch?v=4eIBisqx9_g>

Let us know if you had any issues understanding YOLO.

78)