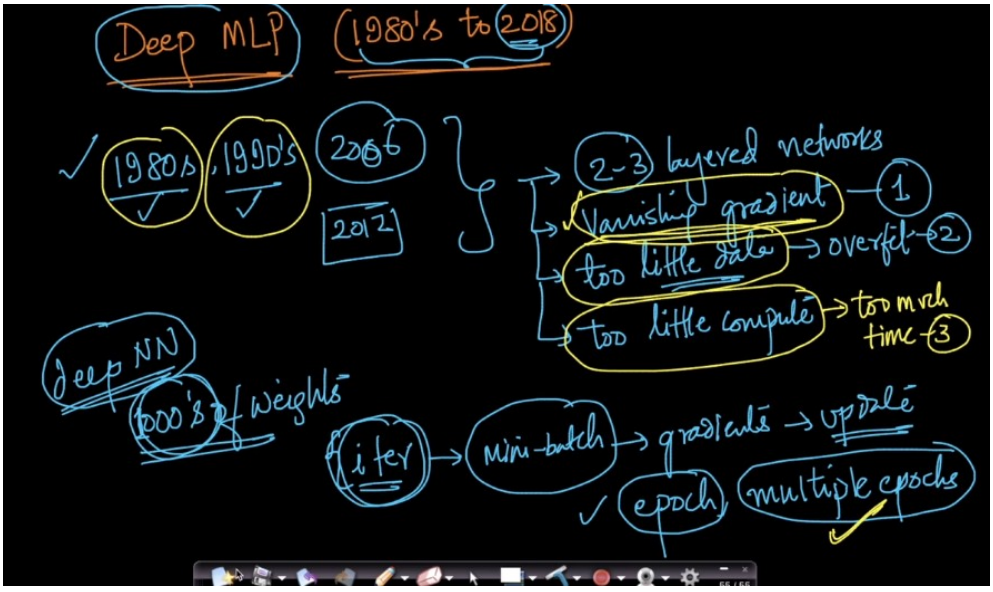
Deep Multi-layer perceptrons.

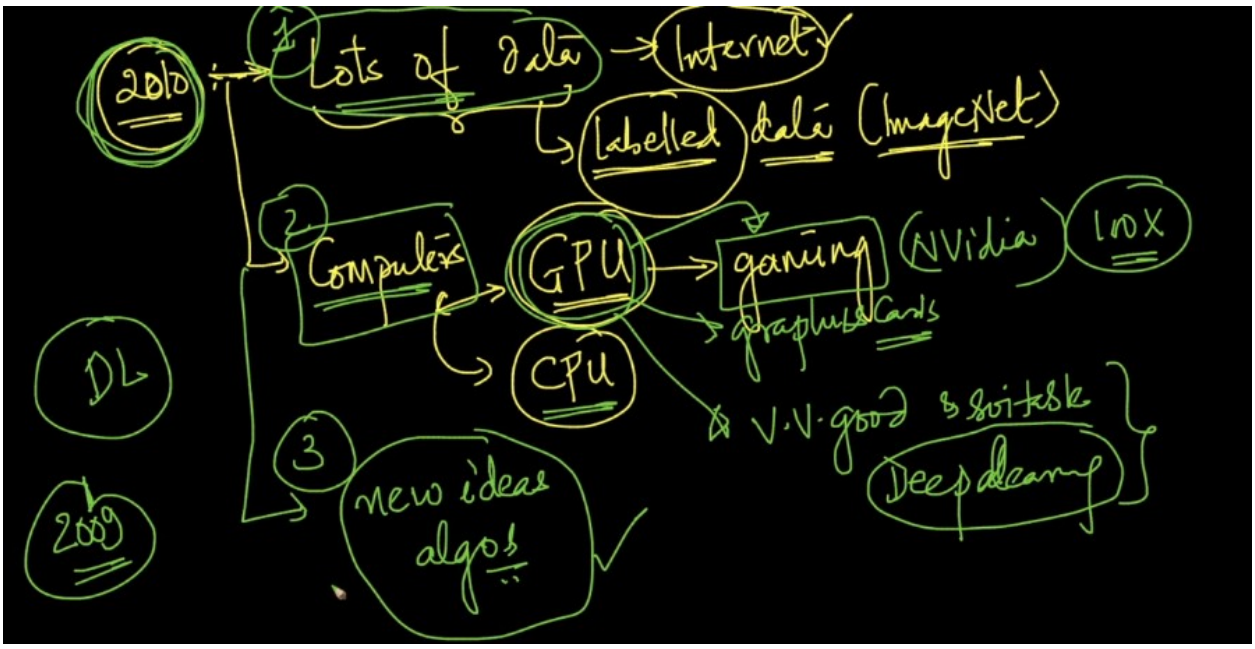
1. The biggest problem is the vanishing gradient problem to train deep networks.
2. We have little data to train the networks and It is easily can over fit the network.
3. We have too little compute power. We take lot of time to work.

In 1980:

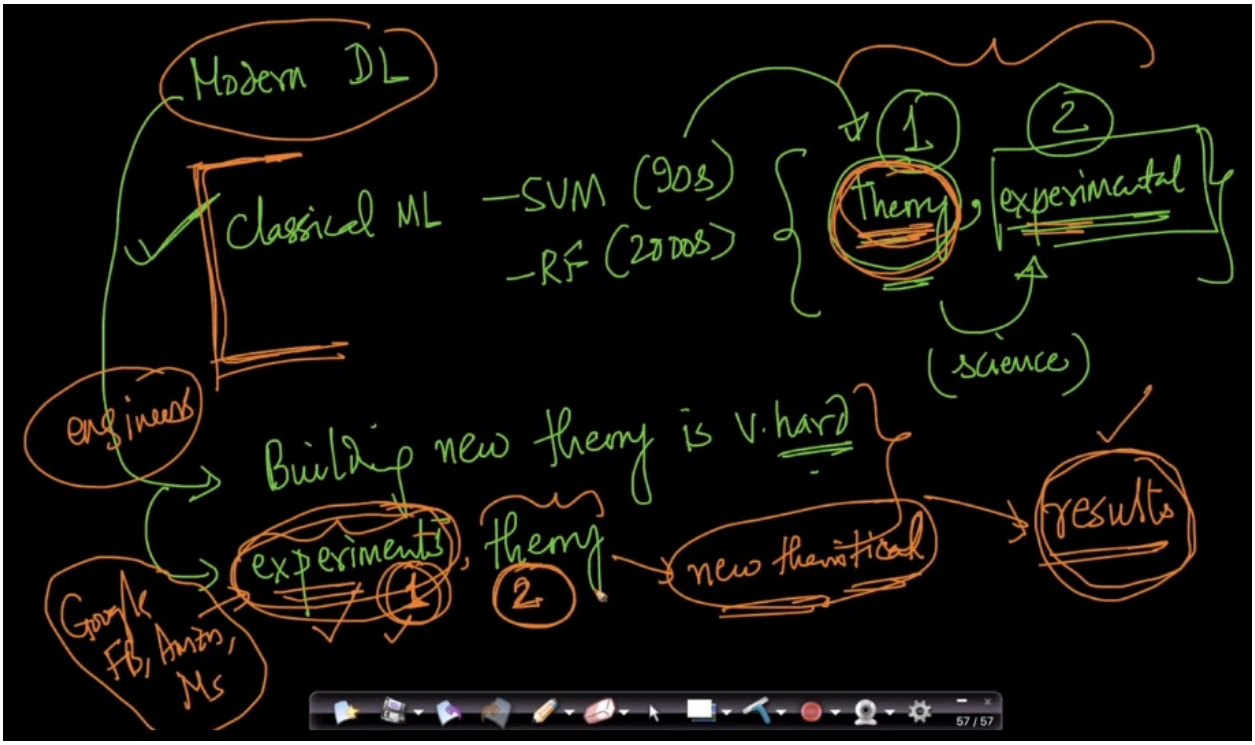


By the time we reach the 2010, lots of labelled data is generated by internet company.

In 2010:



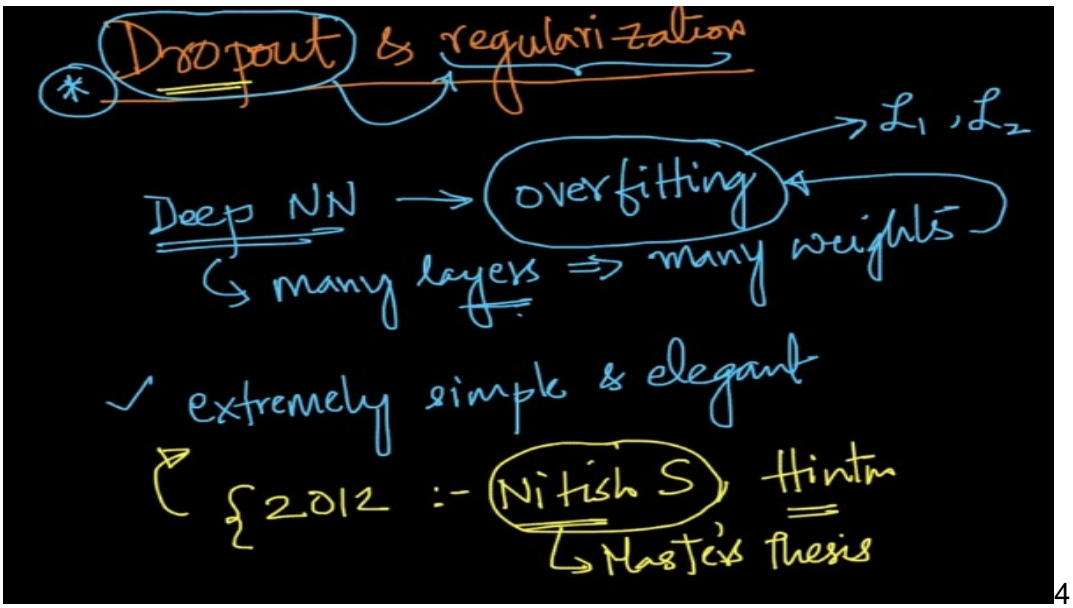
Modern DL:



s

Dropout layers and regularization:

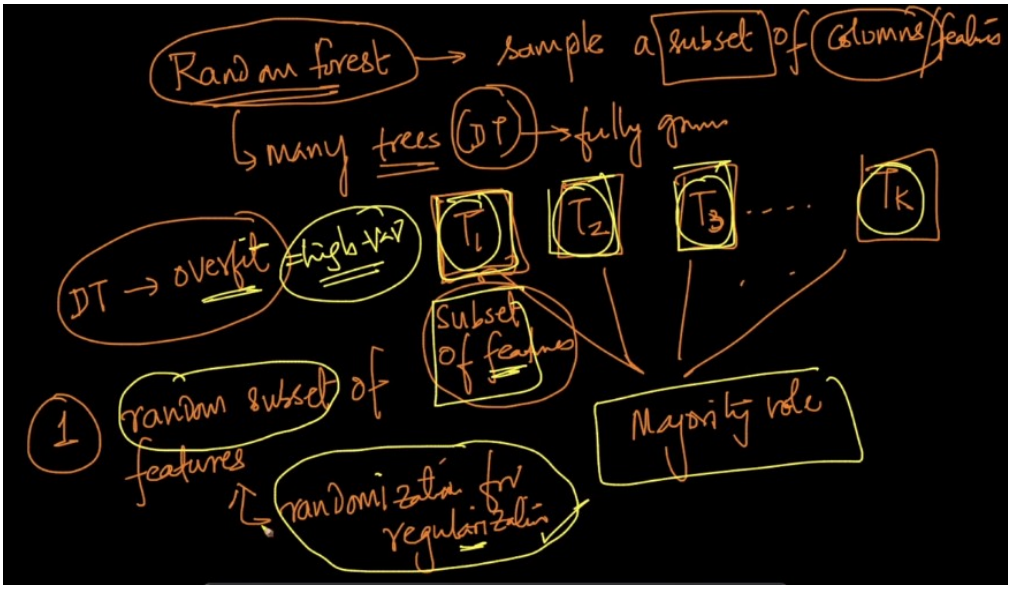
We have many weights so as overfitting, we use the dropout layers and regularization.



4

When we built random forest, we use the trees to look at the small parts of the data and fully grown and also overfit the models.

By using the randomization as the regularization. Using the random forest. This reduces the variance in the model.



The core idea is we are using the randomization of features to enable regularization.

Can we take the randomization for regularization for MLPs ?

Drop out layers:

Remove the neurons randomly between the Input and output layer. For only each iteration.

It is the probability rate lies between 0 and 1 these are called as the “p” value.

“P” value is the percentage the network will be removed. This is very similar to random subset of features in random forest.

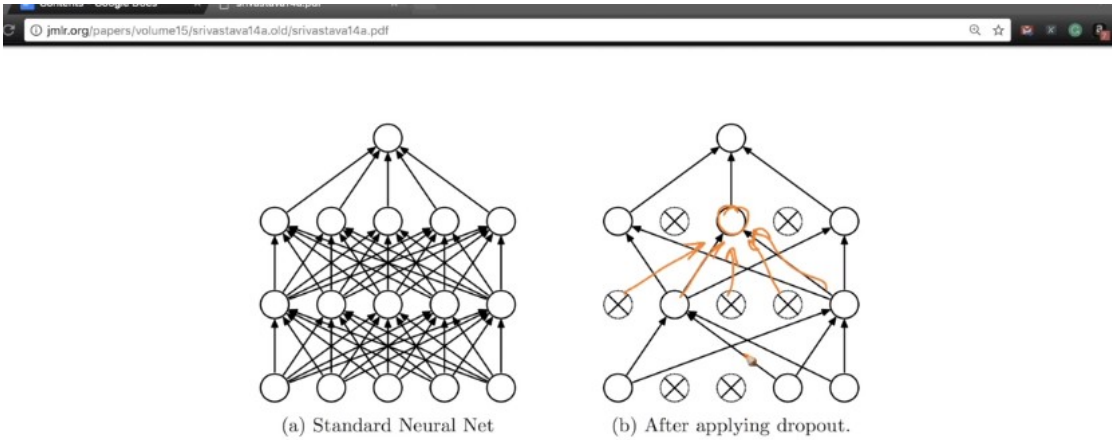
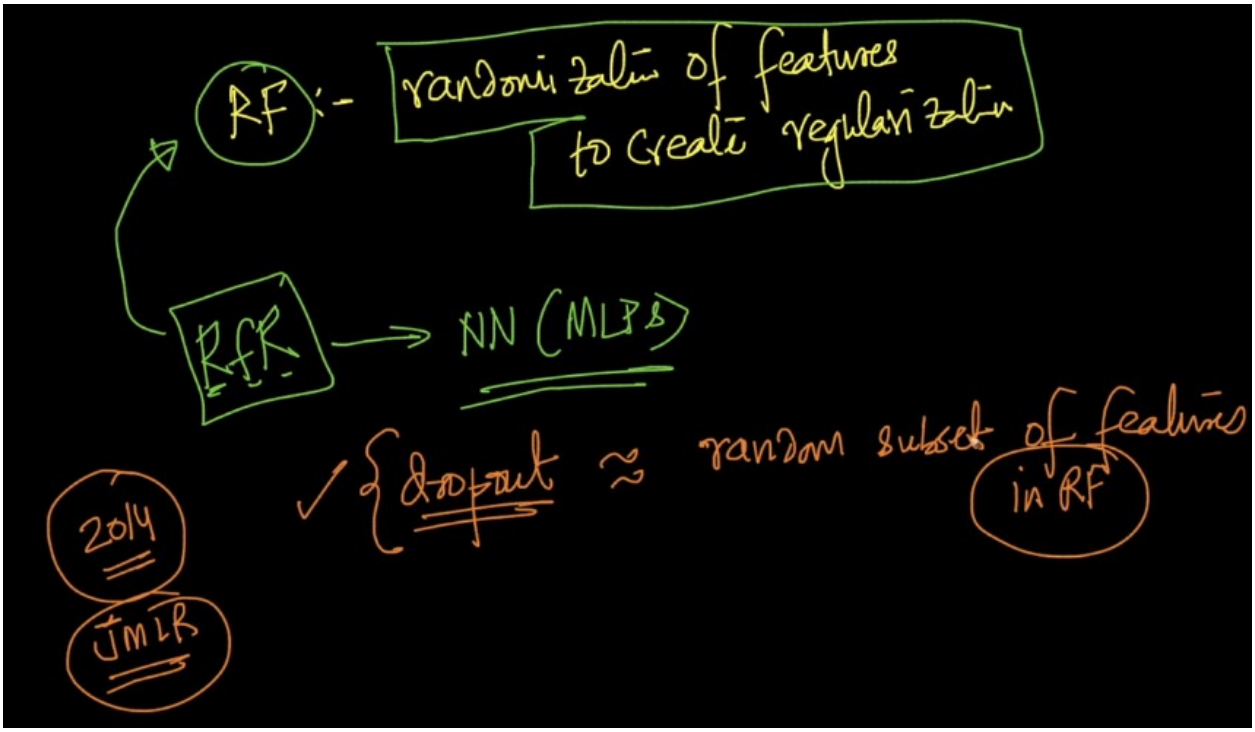
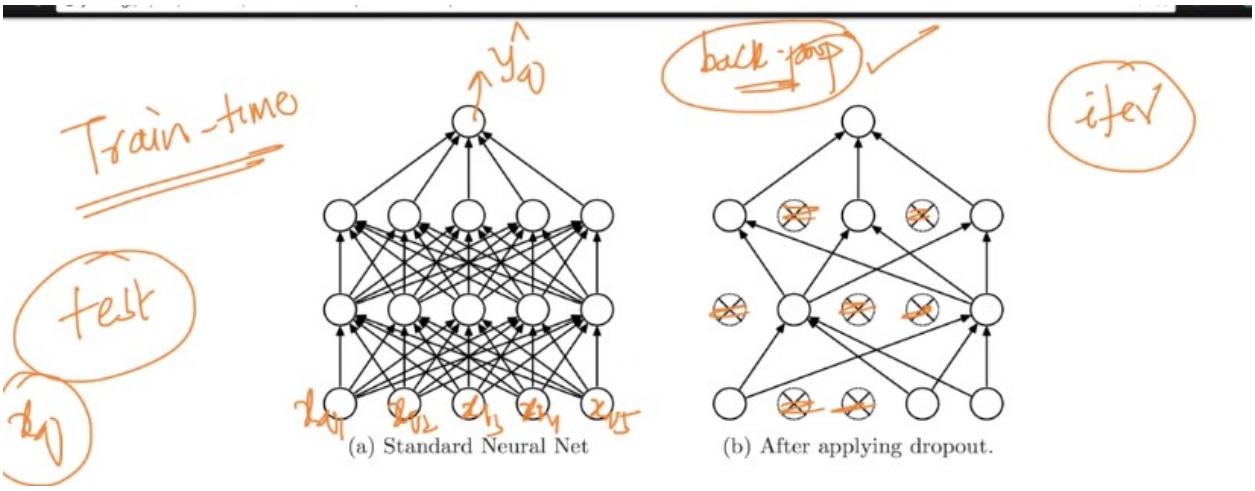


Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Drop out is very similar to random subset of features in random forest. As the inputs to the one neuron is varied at every iteration.



Drop out as having the value that makes the network deactivate. To create regularization.
At test time,



At test time, the network remains the same and the each weight of the network is multiplied with the “P” value.

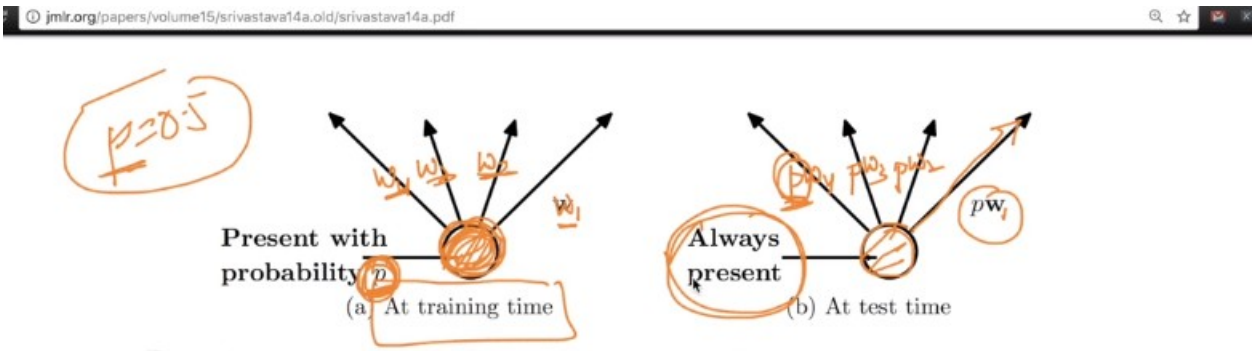
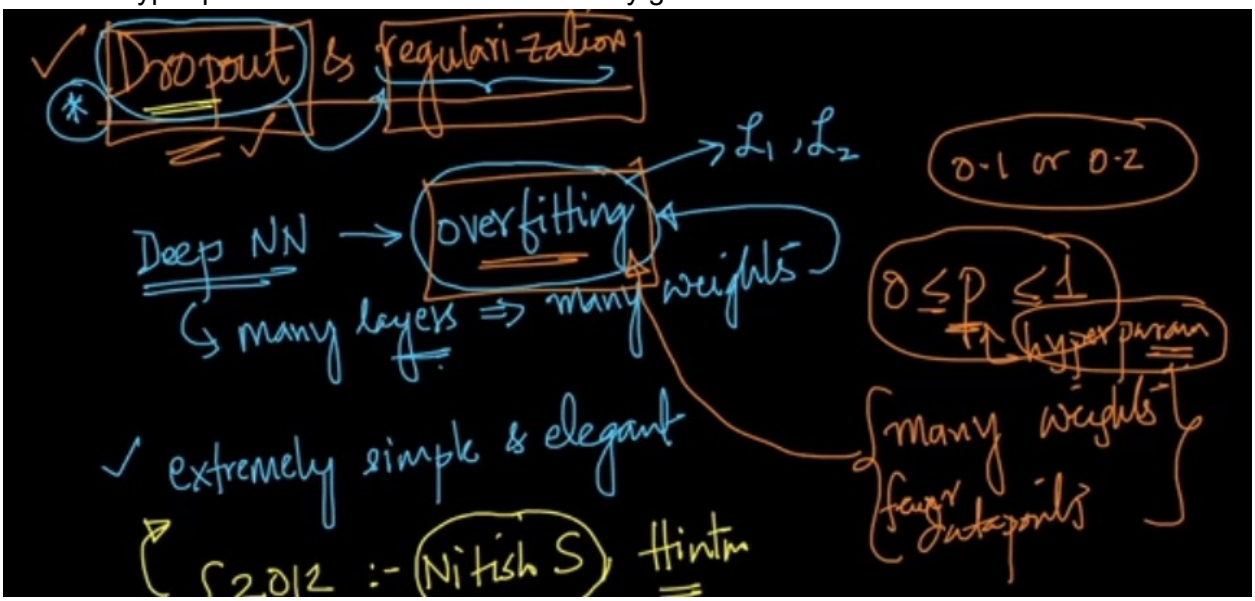


Figure 2: **Left**: A unit at training time that is present with probability p and is connected to units in the next layer with weights $\mathbf{w}$. **Right**: At test time, the unit is always present and the weights are multiplied by p. The output at test time is same as the expected output at training time.

When given a query point, if multiply with the weight value and “P” value.

If we have more weights than the number of data points then there are high chances for overfitting the **less is the value of P may be like 0.1 or 0.2.**

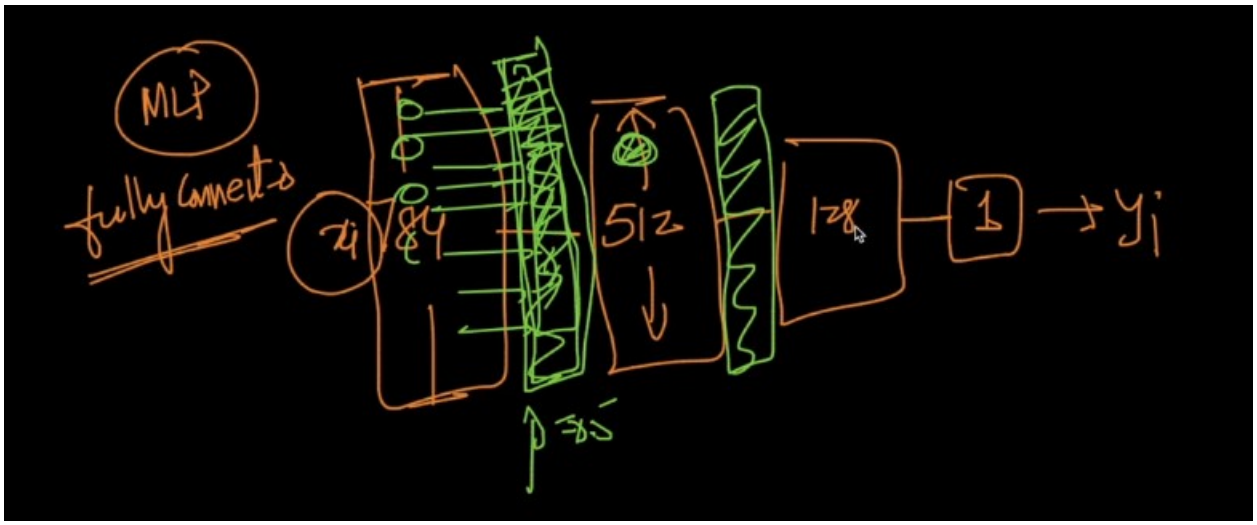
This is a hyper parameters can be determined by grid search and all other networks.



If there are many weights more than number of data points then we keep “P” value is smaller. Here “P” is the hyper parameter.

Fully connected Multilayer perceptron.

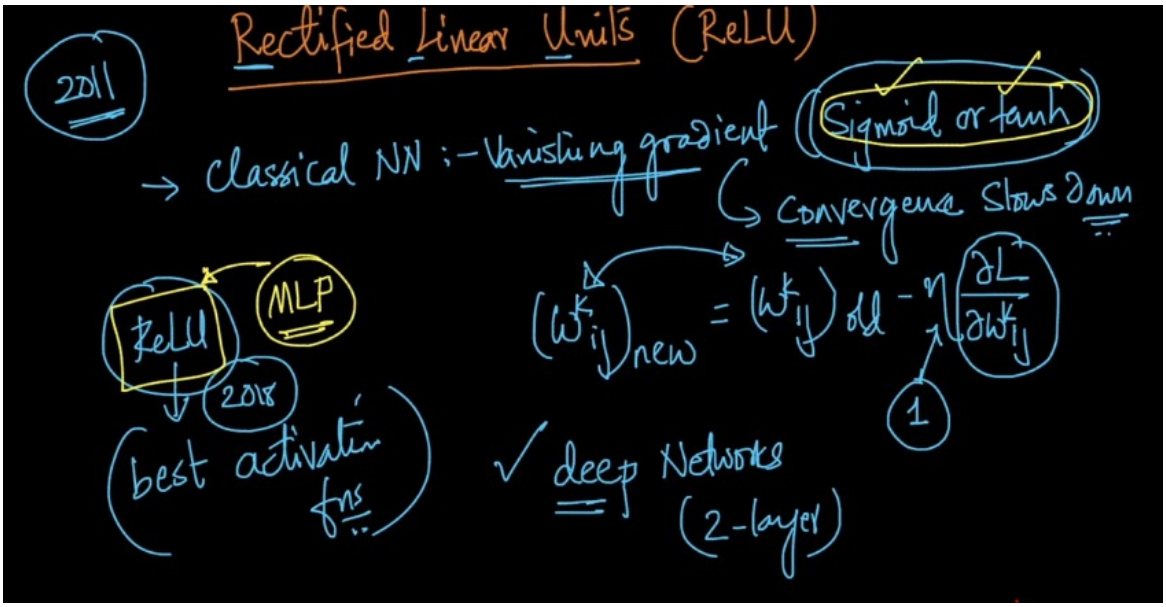
The dropout layer is applied after the layer in the neural network. The dropout layer randomly sends the data from the layer to the next layer.



People call this as the dropout network.

Rectified Linear Units(ReLU):
One of the problem is the vanishing gradient problem which is very often seen when we have the sigmoid and tanh activation and the convergence also slows down.

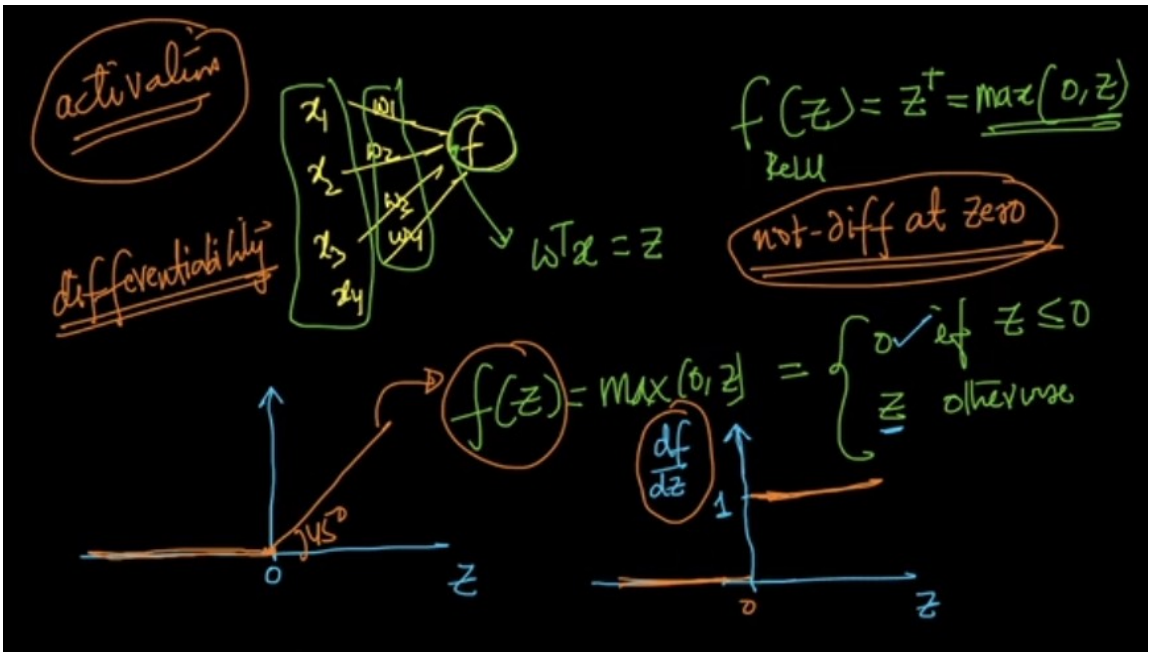
They are the default activation functions that are implemented in many networks.



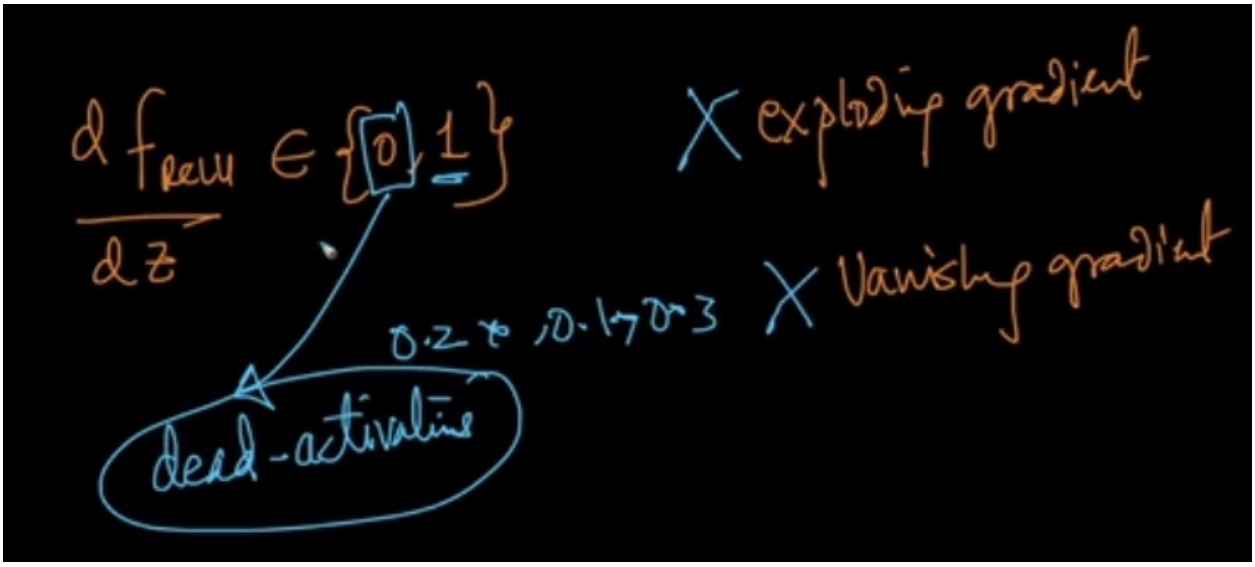
ReLU is the best activation function.

The slope of the activation function is always equal to 1 or 0. Just like the Hinge function in SVM.

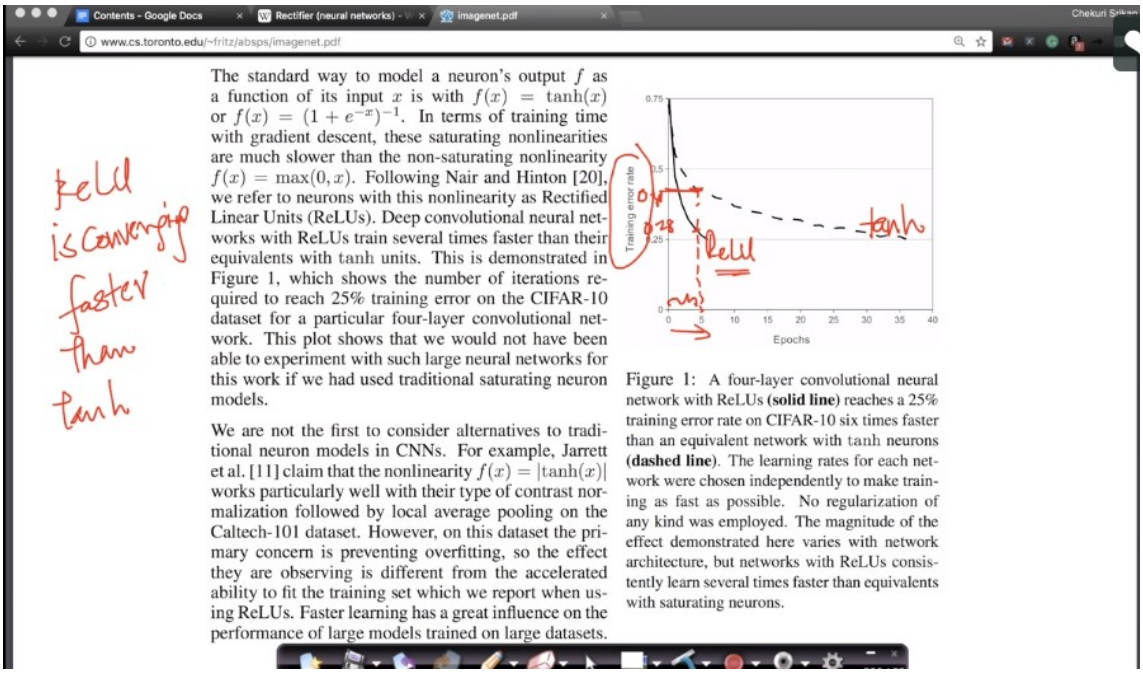
The derivative at ZERO is not defined.



We have ReLU to overcome the vanishing gradient problem.
Because of Zero we have the problem of dead activations.

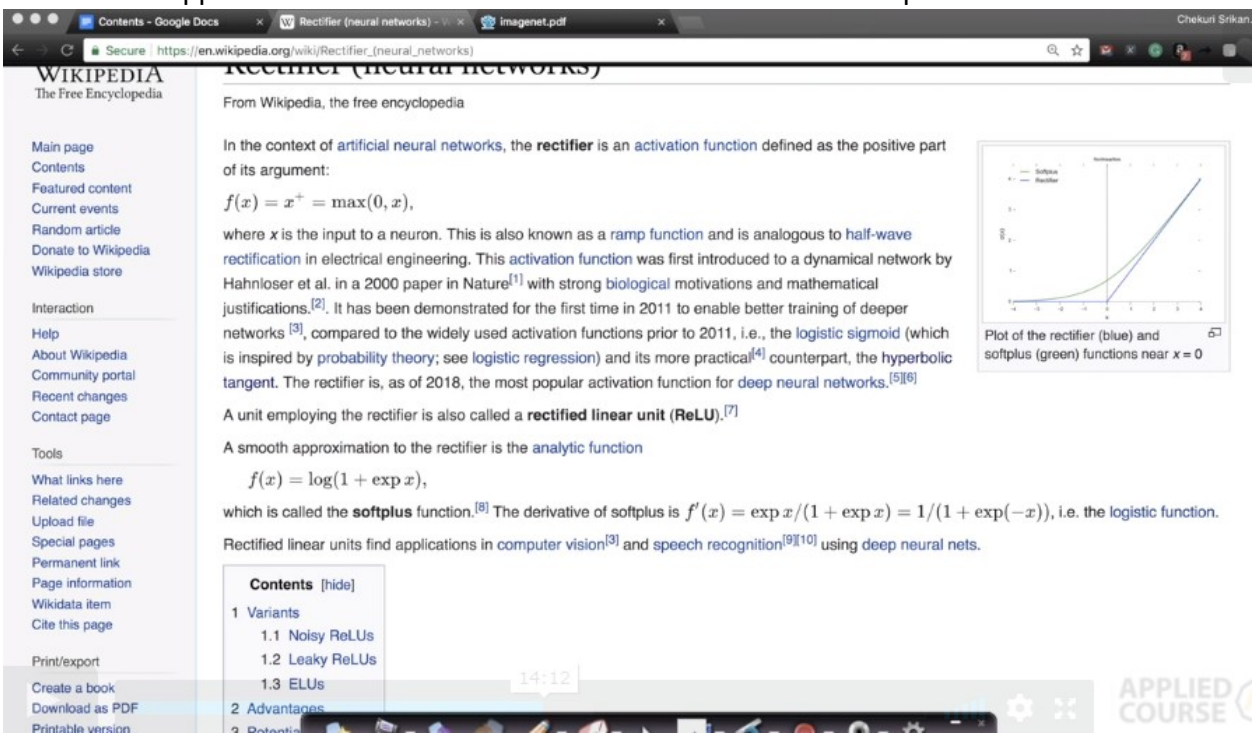


The solid line represents the model trained using ReLU. and dotted is the TanH.



Because we does not vanishing gradient or exploding gradient problem. So, the network converges faster than the TanH.

The smooth approximation to ReLU. Such a function is called the softplus function.



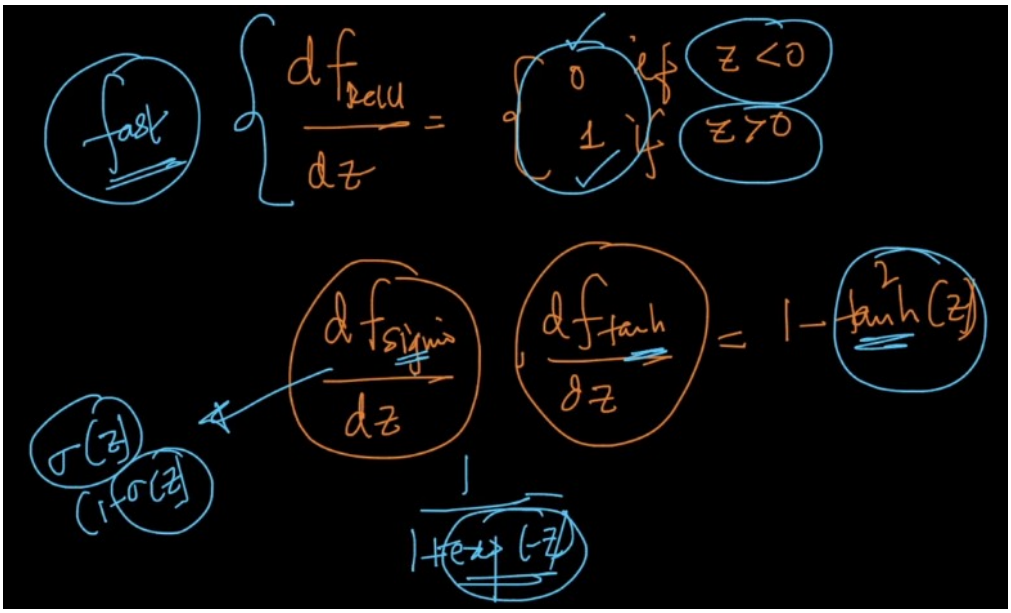
The derivative of the softplus function is the logistic function. It is not much widely used.

$f(x) = \log(1 + \exp x)$,

which is called the **softplus** function.[8] The derivative of softplus is $f'(x) = \exp x/(1 + \exp x) = 1/(1 + \exp(-x))$, i.e. the logistic function.

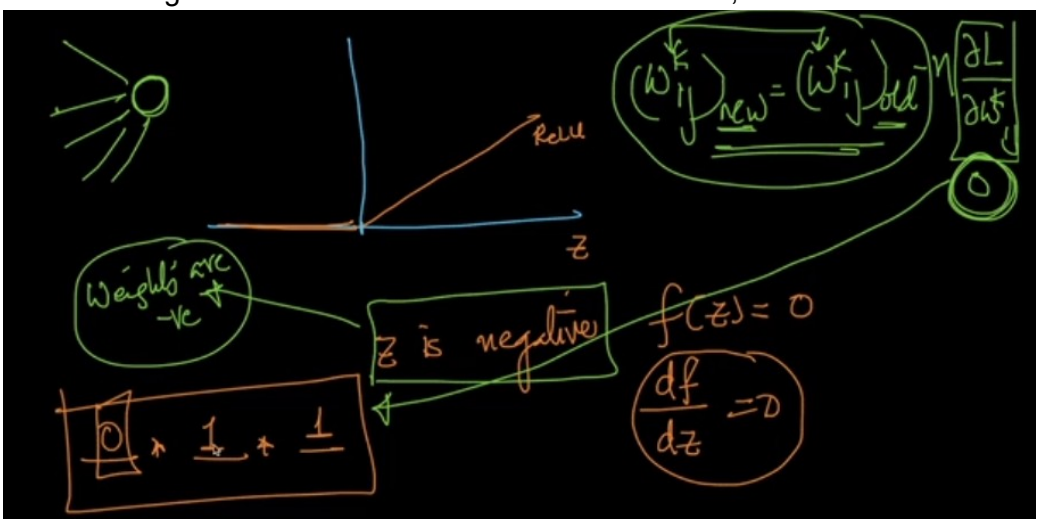
Rectified linear units find applications in computer vision[3] and speech recognition[9][10] using deep neural nets.

Computing the derivative is also much simpler.



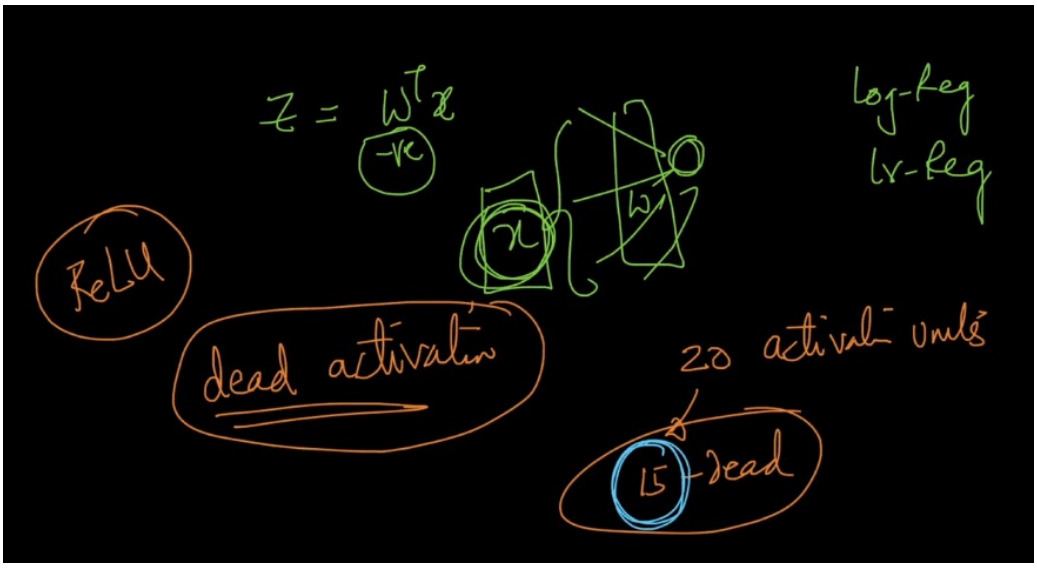
Noisy ReLU's - This is not the most used one in MLP's.
Leaky ReLU's:

If the Z is negative then the derivative also become zero, this makes the chain rule also zero.

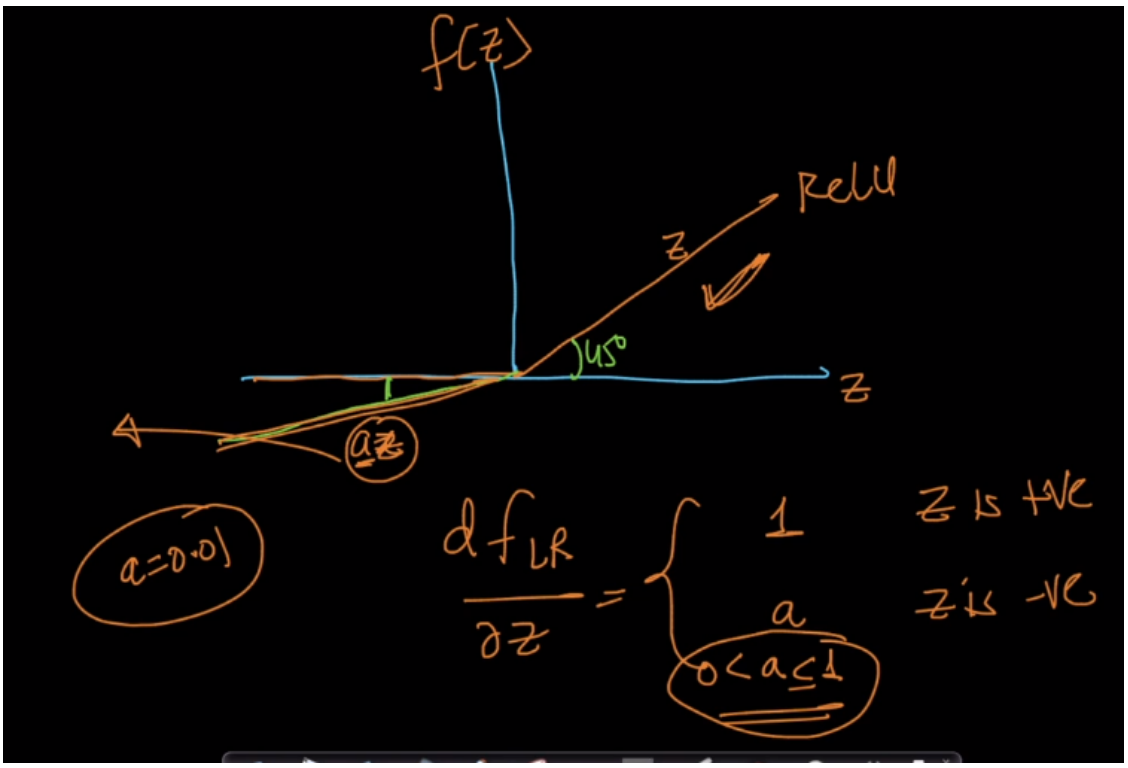


The weights are not changing anymore, when the weights are negative. Which we don't need. This is called the dead activation state.

The input to NN is always normalized.

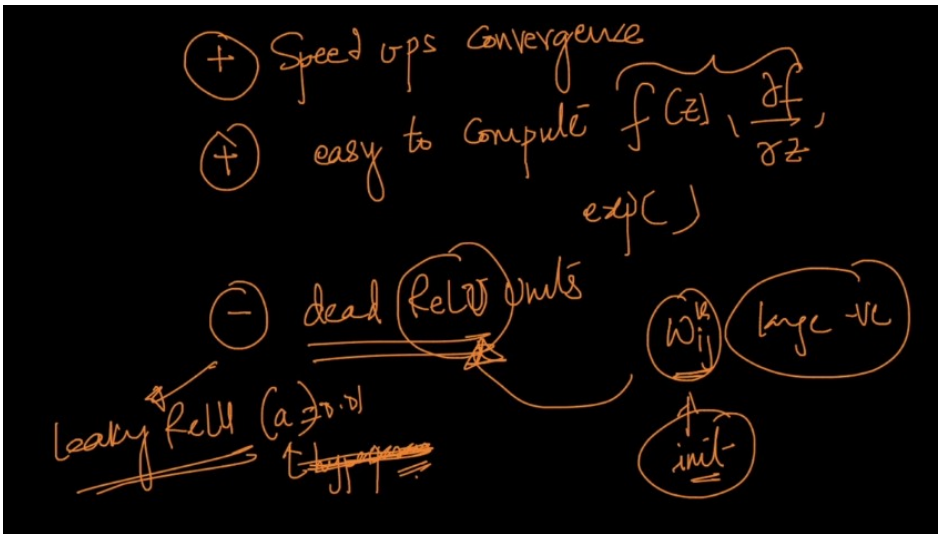


The fix for this problem is giving a small value to the negative input to the neurons in the NN.



Typically people use the ReLU, If we found the more dead neurons in the NN then we tend to use the Leaky ReLU.

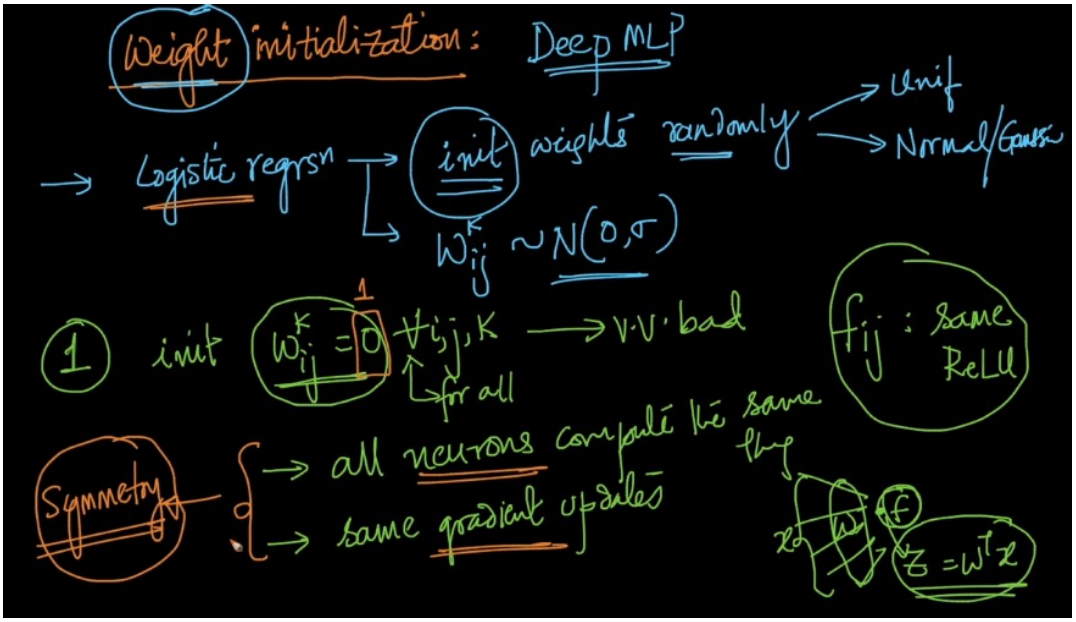
Advantages of ReLu:



Weight Initialization:
For logistic regression the weights in the SGD, we initialize the weights randomly. We will initialize the weights from the gaussian normal distribution.

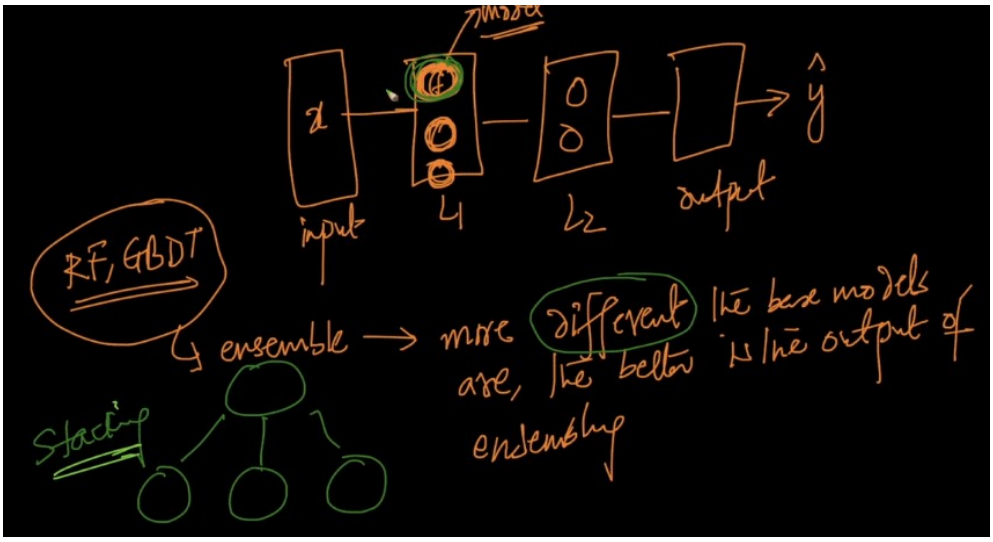
What happens ?

1.



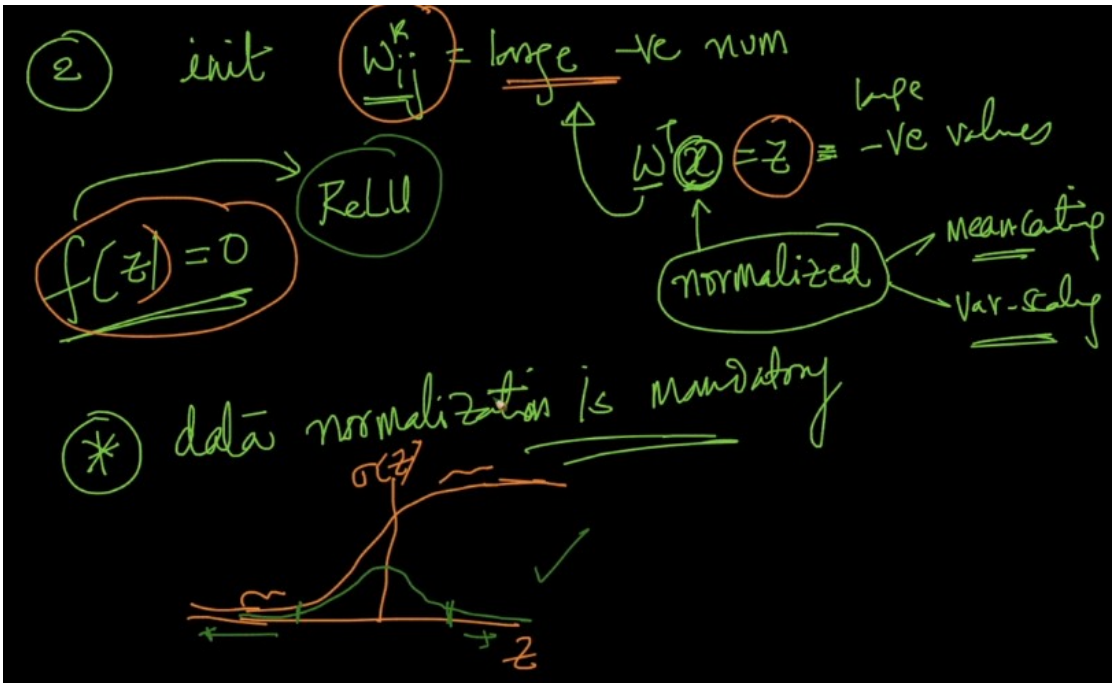
We want asymmetry in the NN. That makes the each layer in the NN to learn different things.

In ensembles the more different the base models are, the better is the output of the model.



If we have the same weights we learn the same. This we don't need.

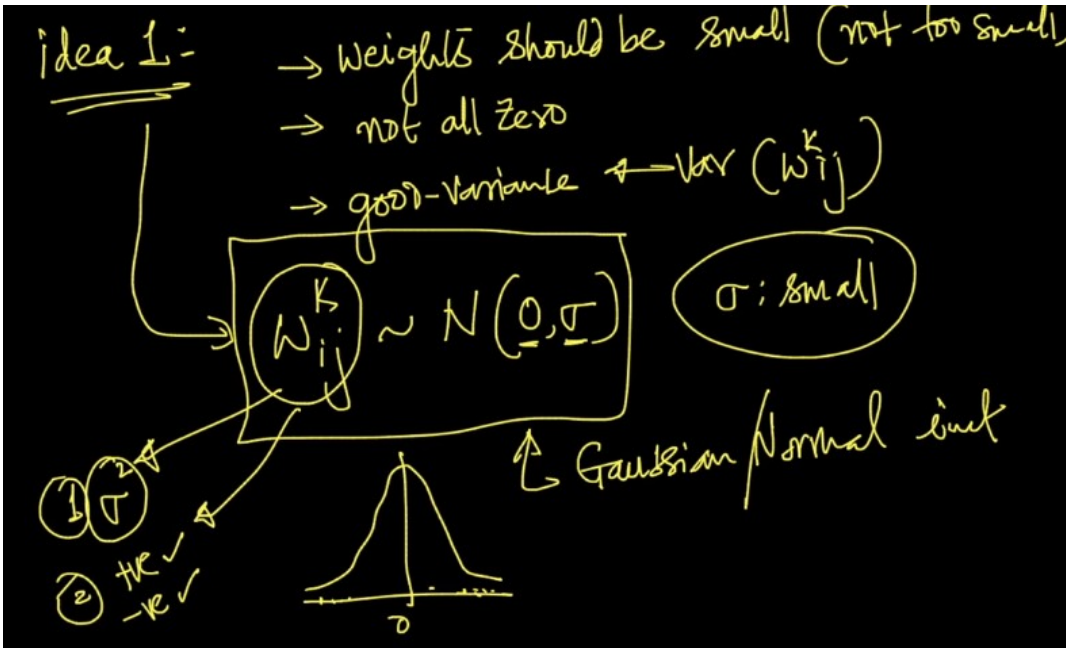
2. If we have negative values then there is the problem of dead neurons in Case of ReLU and other activation functions.



The data must be NORMALIZED for DNN.

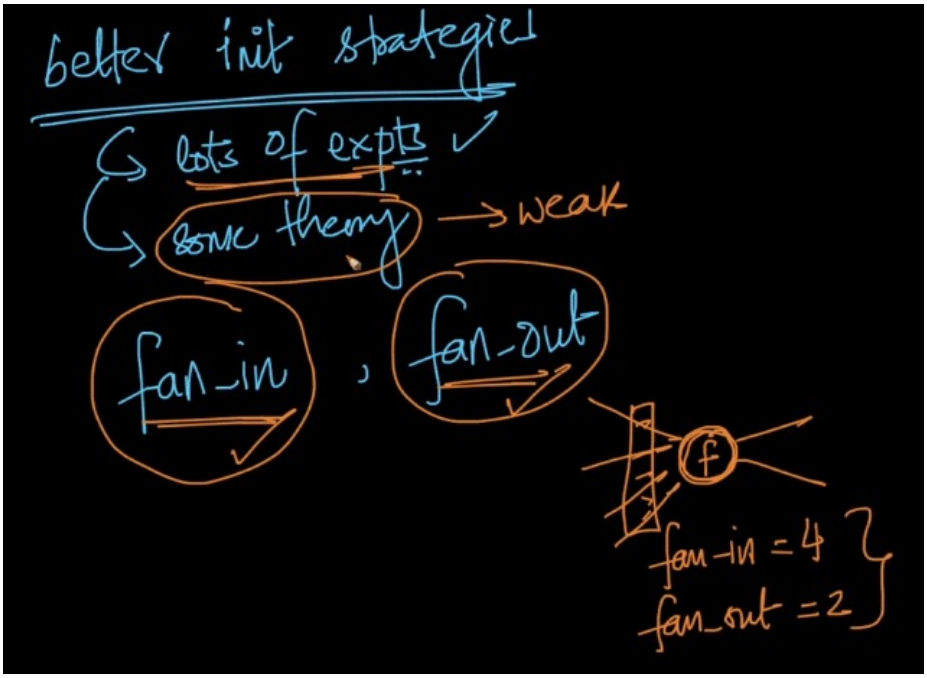
Solution for the above cases:

Idea 1: Weights from the normal distribution.



Can we come up with better Initialization strategies ?

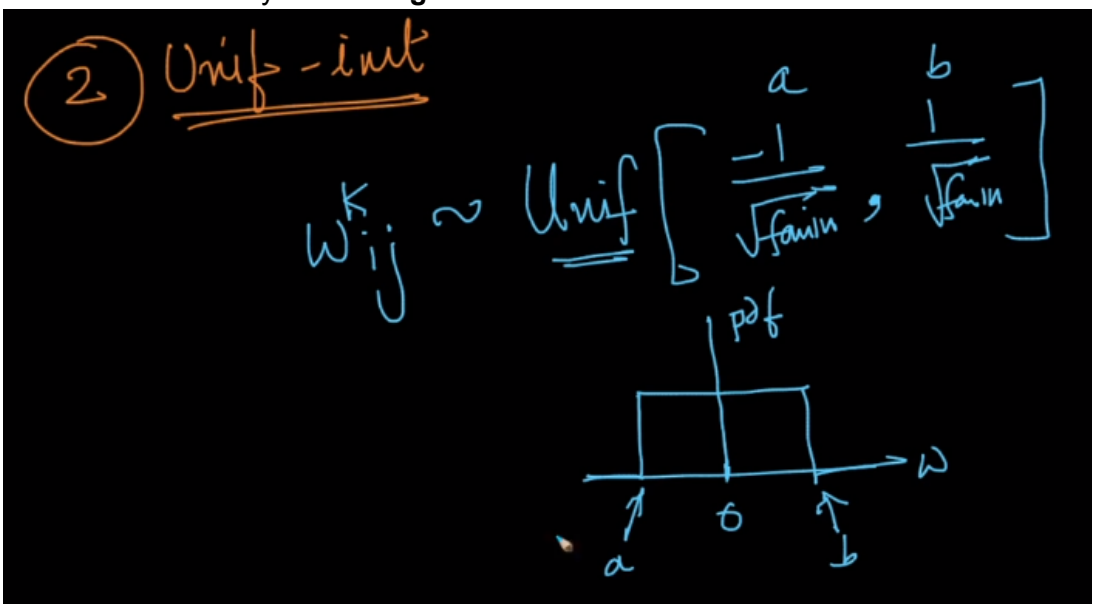
Fan IN: The number of inputs to the neuron.
Fan Out: The output from that neuron.



First technique:
Uniform initialization:
The weights from the uniform distribution with fanIN and fanOut as the function.

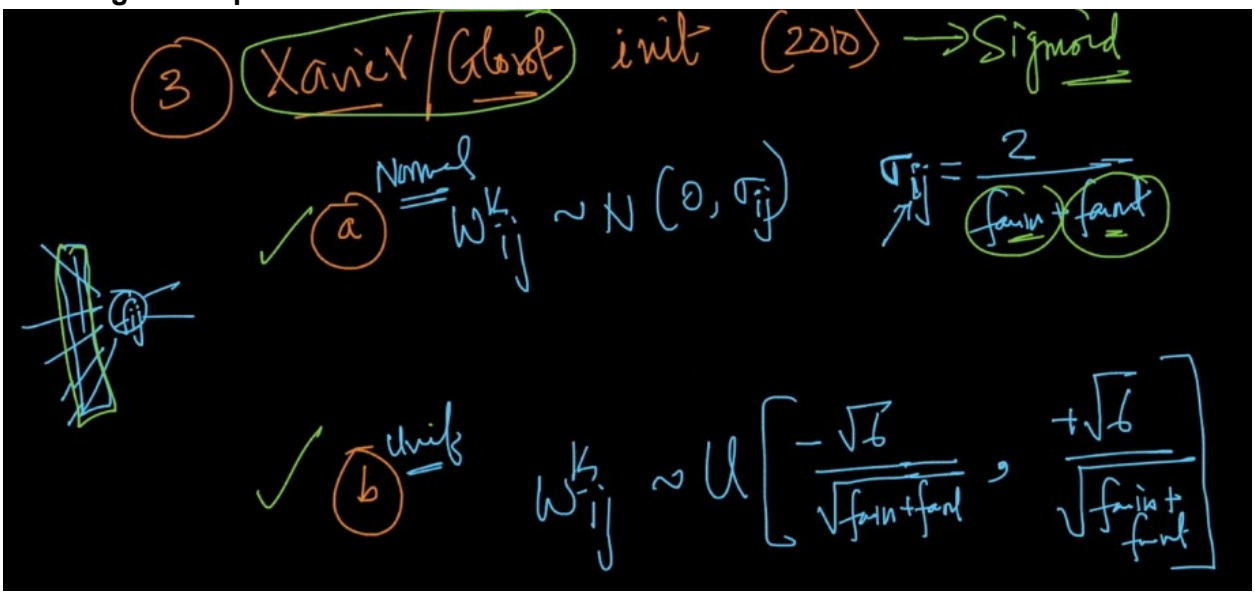
Idea - 2:

This also works fairly well for **sigmoid.**

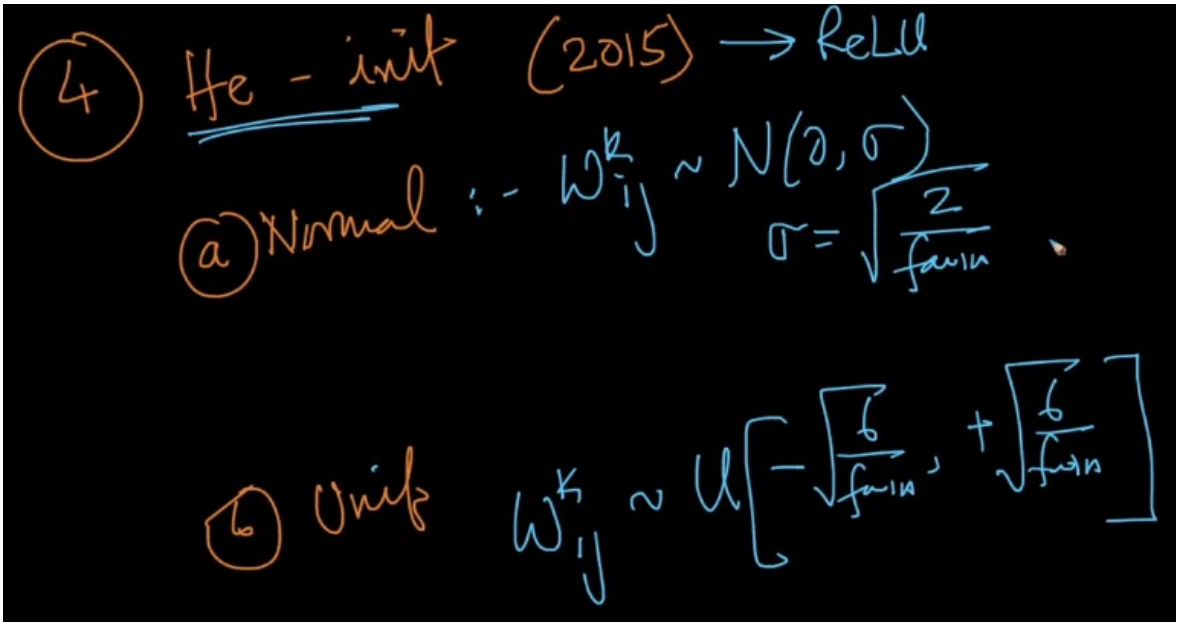


There is no concrete agreement among all the researches.

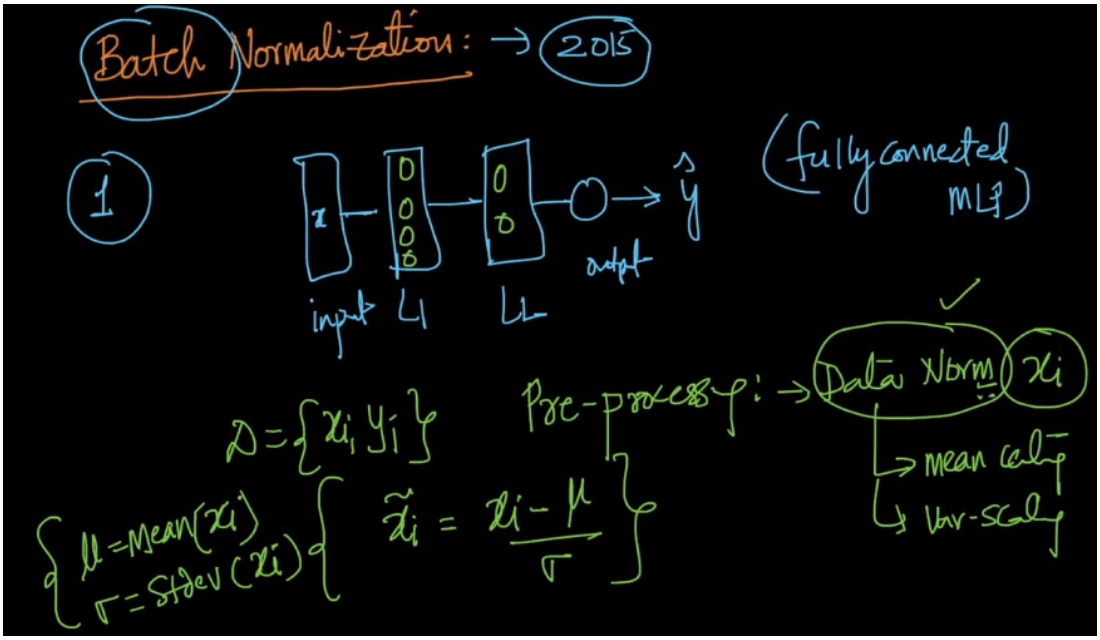
Idea - 3:
Xavier/Glorot initialization:
There are two variations in Xavier initialization.
The weights are picked from the normal distributions and normal distribution.



Idea - 4:
He - initialization:
This also have the normal and uniform distribution. It works with ReLU and Leaky ReLU.

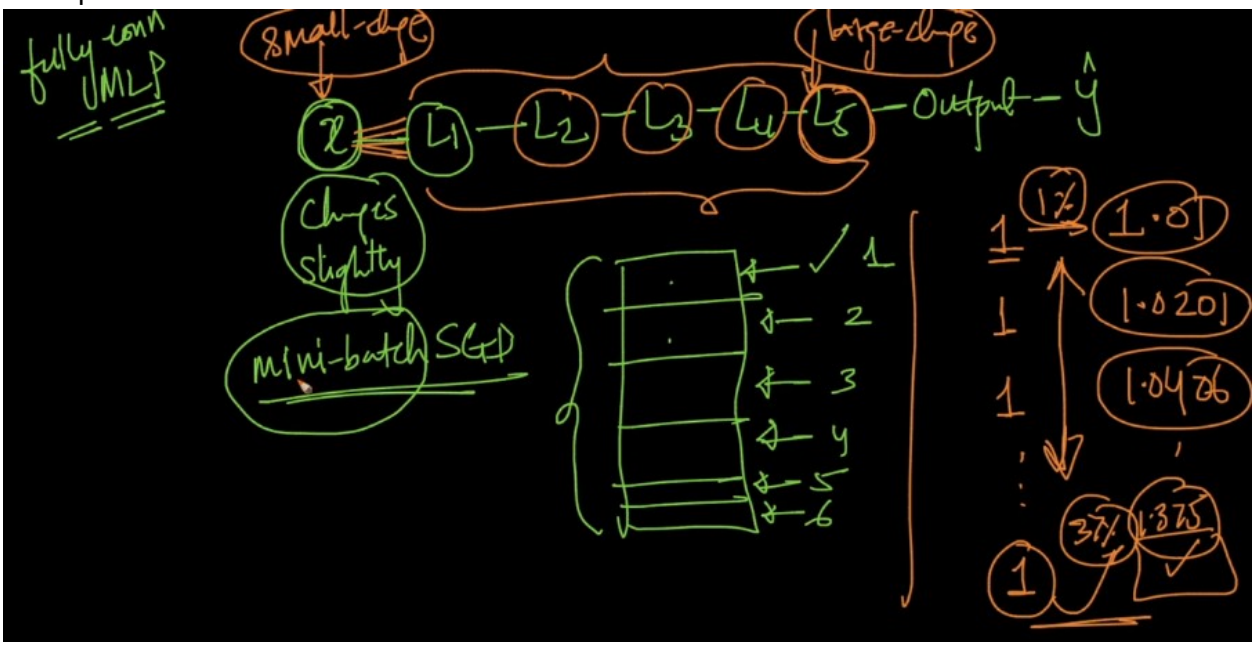


Batch Normalization:
The pre-processing steps in the NN is the data normalization.



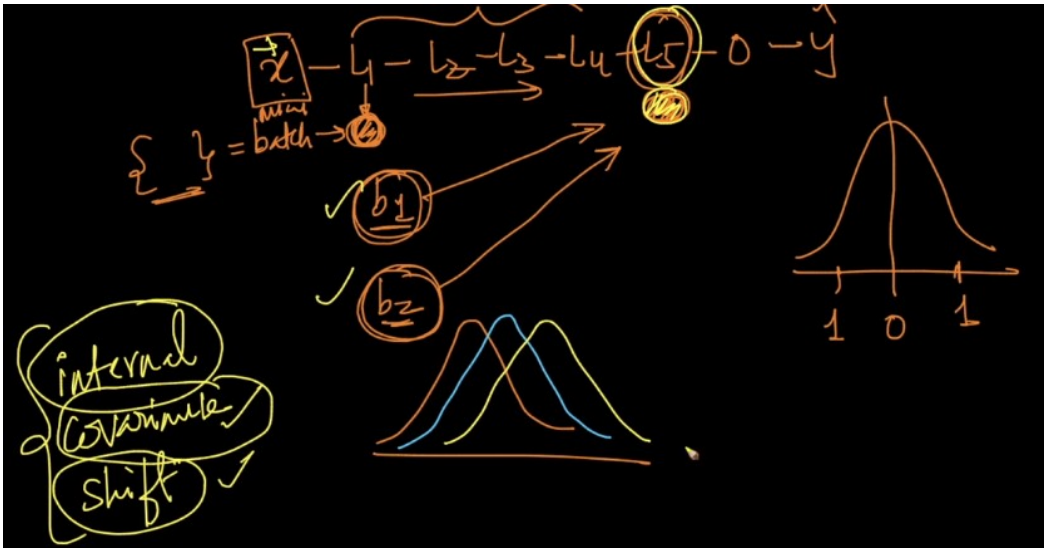
If the input changes slightly a small change in the input can give the large change.

Example:



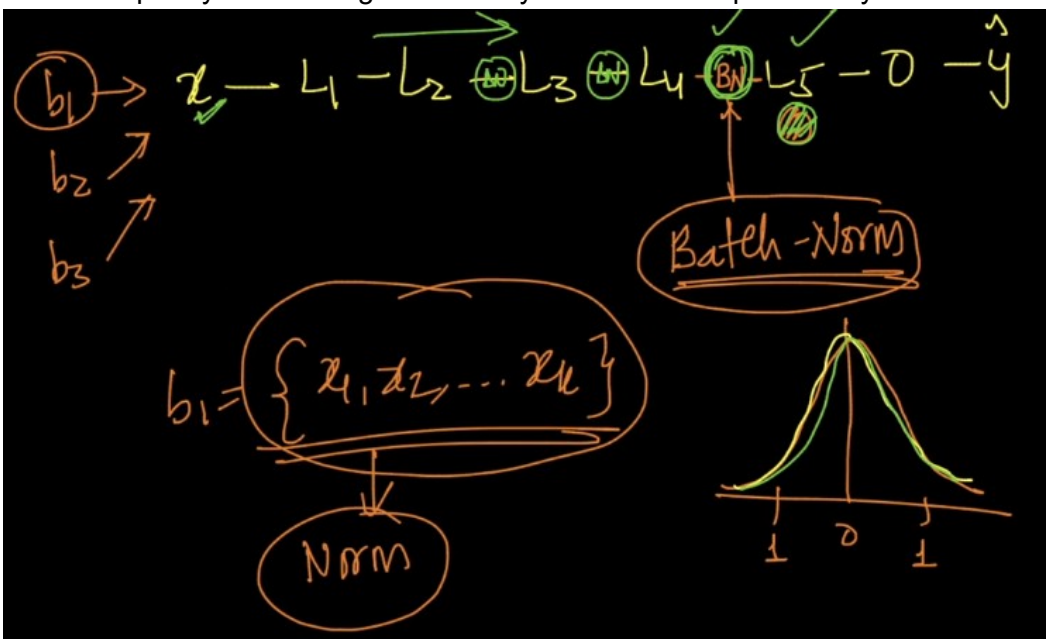
Between each batch there is less difference as the data is normalized the lower layers does not get affected so much,
But the layers deeper in the network will be affected more.

Ideally we want the data to be normalized for every layer, for each layer there will be different distribution.
The neurons at each layer can go Crazy. This problem is called the internal co-variance shift.



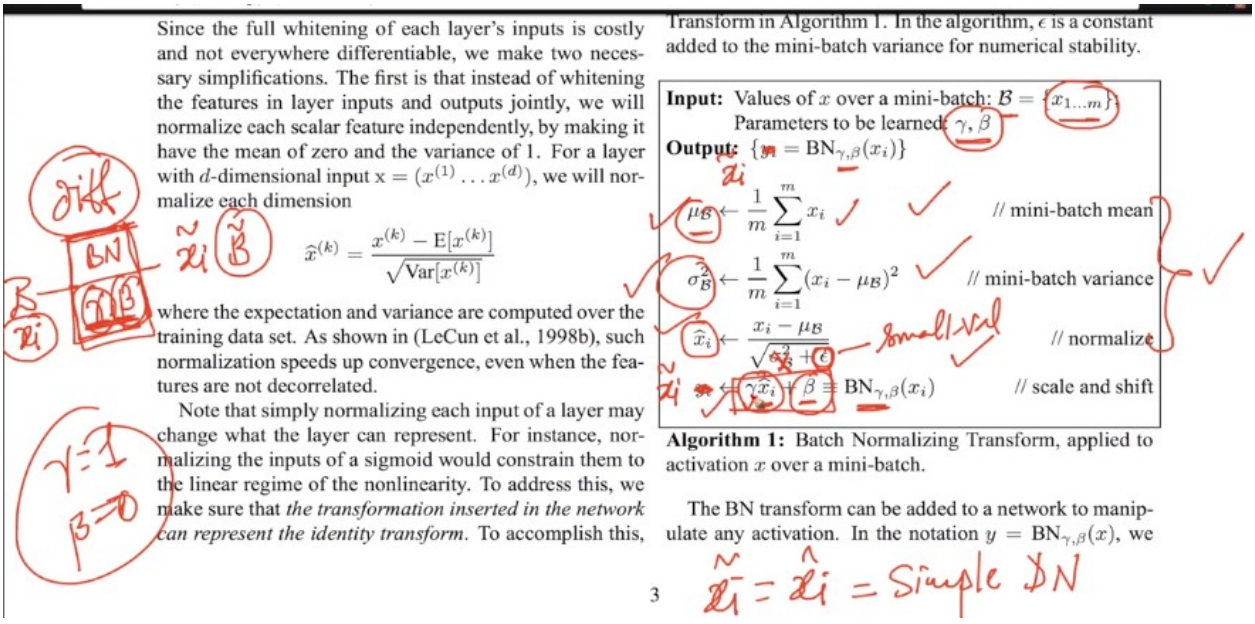
The solution for this is adding a new layer called Batch Normalization Layer, whenever we get a batch of inputs. We normalize only that batch.

We are explicitly normalizing for each layer. It works deep in the layers.



The batch norm is in between the two layers. It has two hyper parameters <gamma> and <delta>.

We will learn these parameters as the part of back propagation.



Since the full whitening of each layer's inputs is costly and not everywhere differentiable, we make two necessary simplifications. The first is that instead of whitening the features in layer inputs and outputs jointly, we will normalize each scalar feature independently, by making it have the mean of zero and the variance of 1. For a layer with d-dimensional input $\mathrm{x} = (x^{(1)} \ldots x^{(d)})$, we will normalize each dimension

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

where the expectation and variance are computed over the training data set. As shown in (LeCun et al., 1998b), such normalization speeds up convergence, even when the features are not decorrelated.

Note that simply normalizing each input of a layer may change what the layer can represent. For instance, normalizing the inputs of a sigmoid would constrain them to the linear regime of the nonlinearity. To address this, we make sure that *the transformation inserted in the network can represent the identity transform*. To accomplish this, Transform in Algorithm 1. In the algorithm, ϵ is a constant added to the mini-batch variance for numerical stability.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

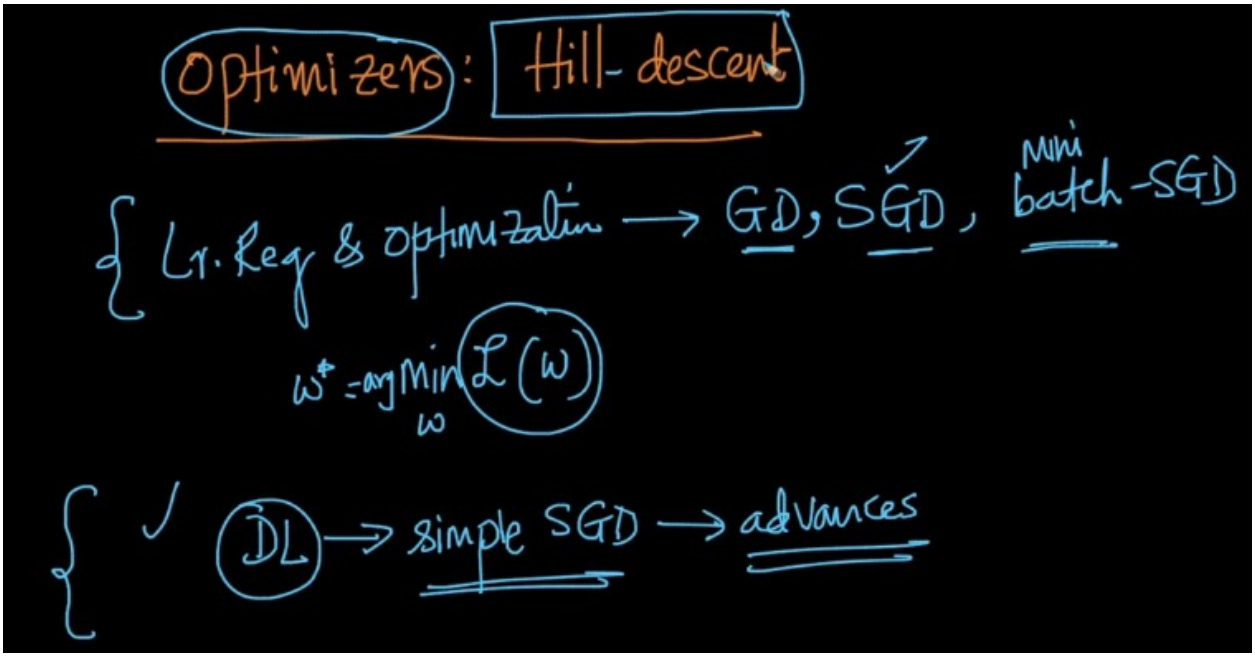
The BN transform can be added to a network to manipulate any activation. In the notation $y = \mathrm{BN}_{\gamma,\beta}(x)$, we

The major advantages are

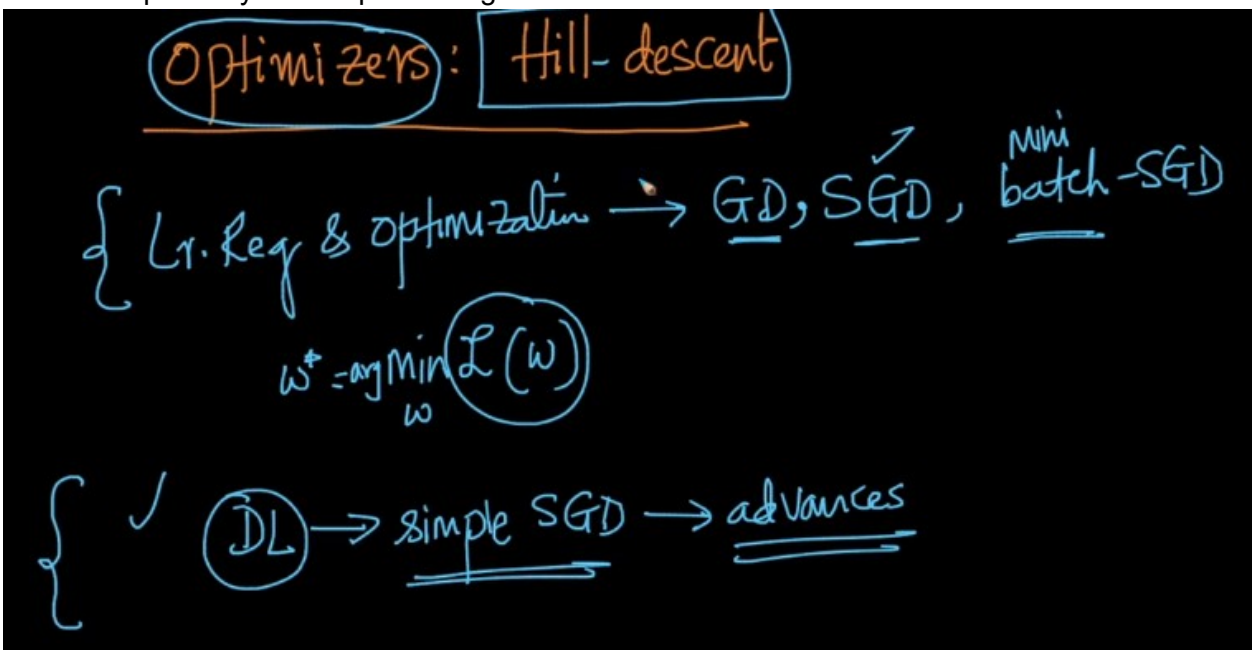
1. Makes faster convergence.

2. We can afford to have **larger learning rate** as the data is from the same distribution.

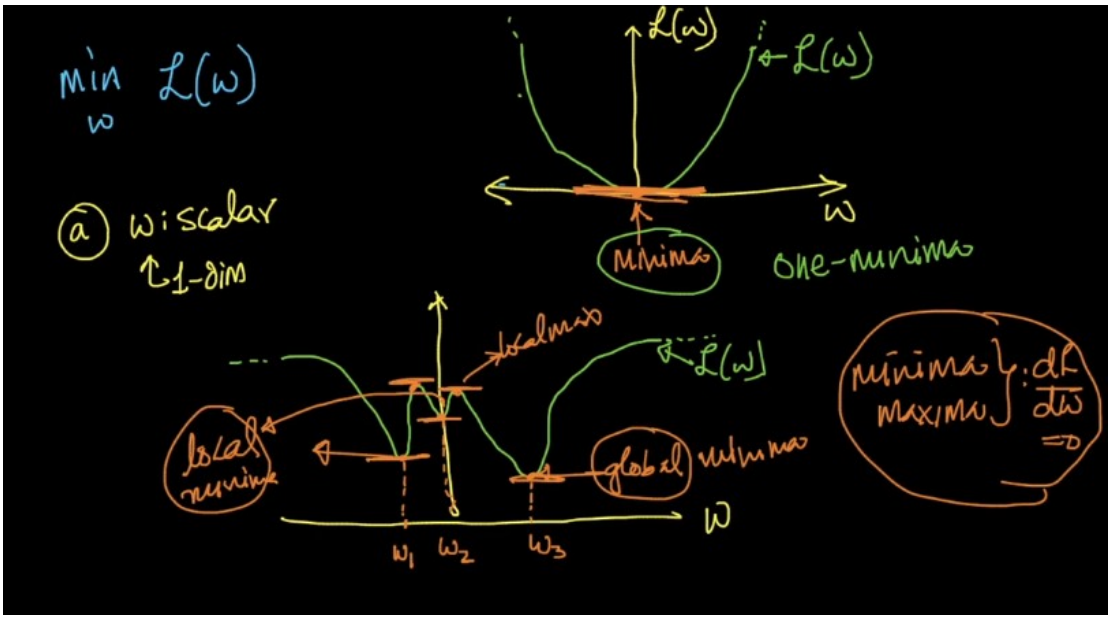
BN is also acts as the regularization. We can train deeper neural networks with the BN.

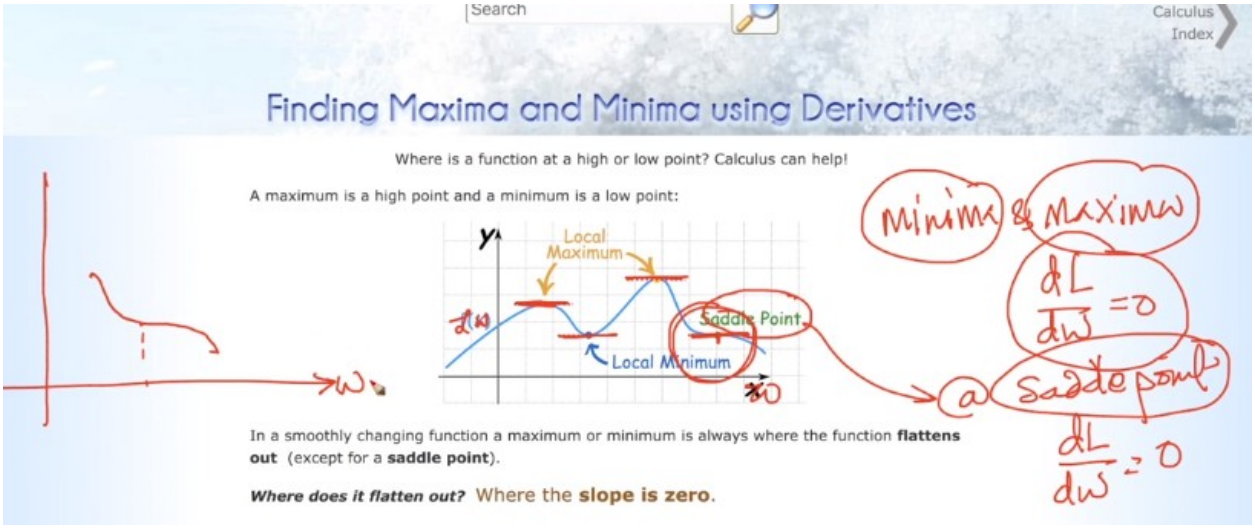


In the case of deep learning, the batch and mini batch SGD do not work very well.
These are primarily for Deep Learning.



Global minima, Local minima:

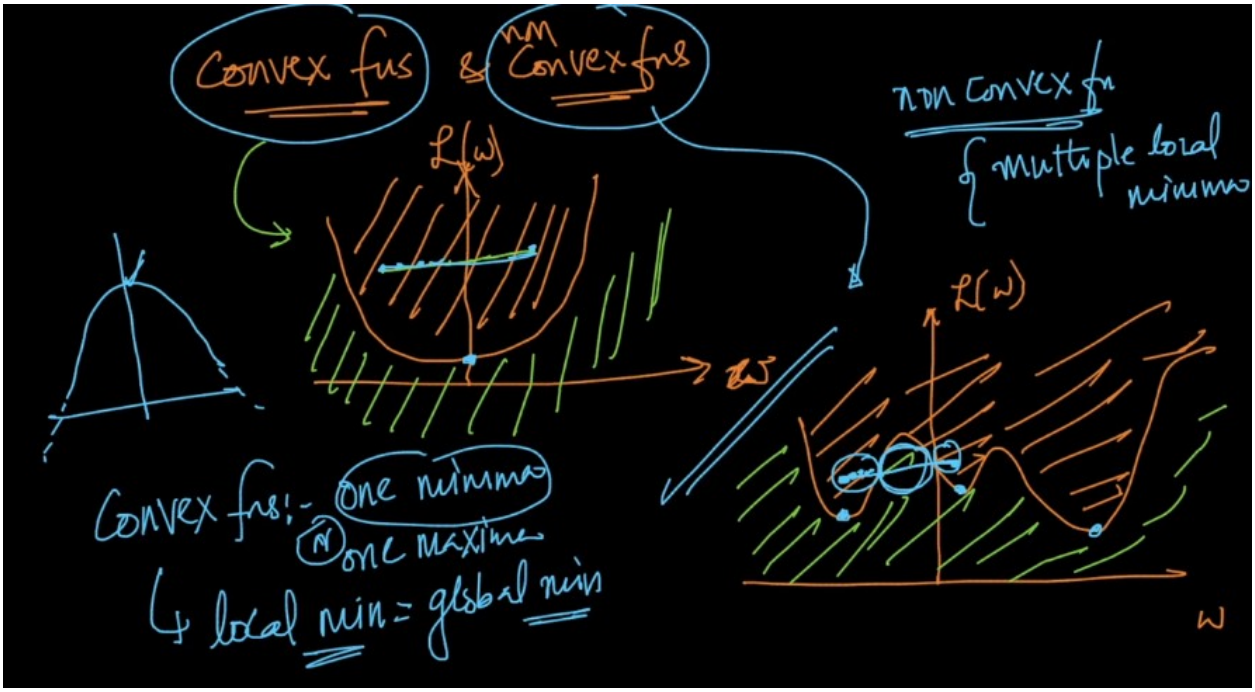




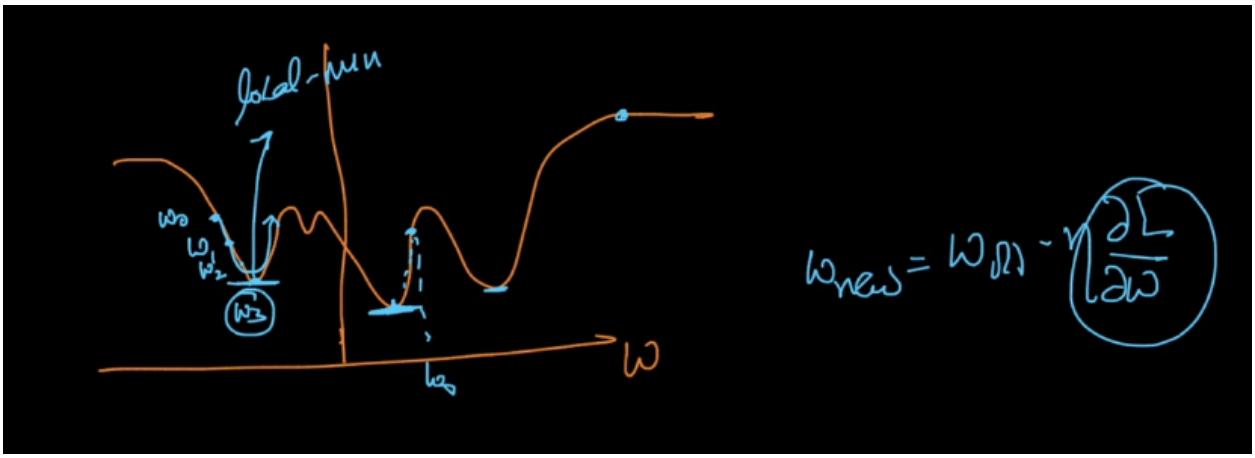
If we recall we keep updating the weight until the derivative becomes zero. This occurs at Minimum, Maximum and saddle point.

We will use the other optimizers that get rid of the zero derivative.

Convex functions and non convex functions:

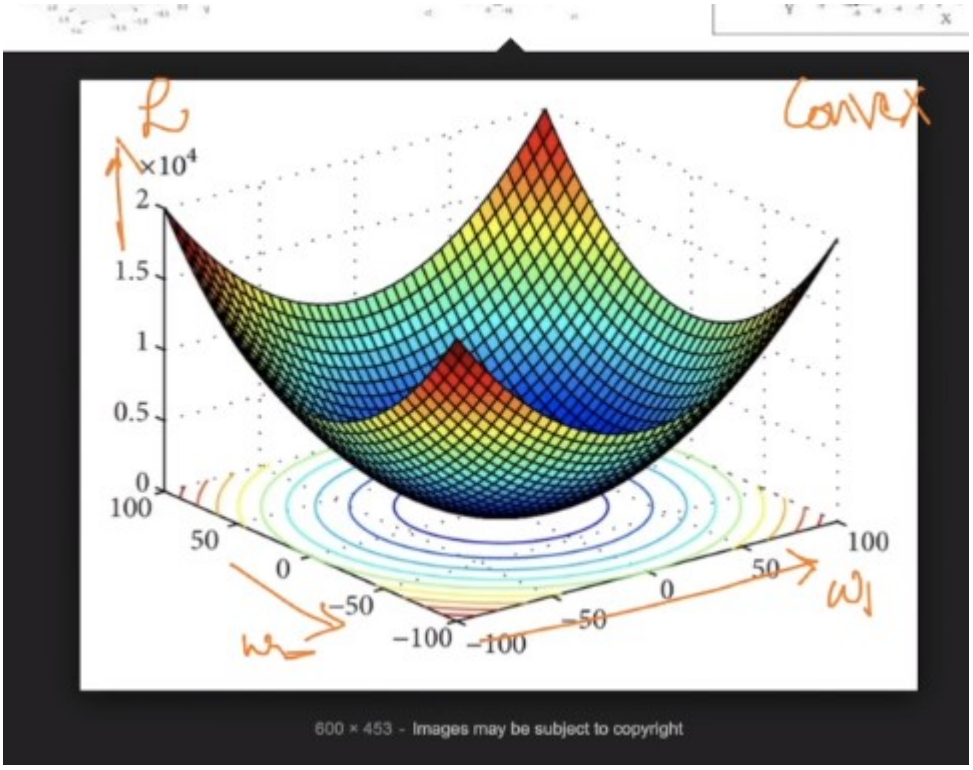


Logistic regression and Linear Regression all of them we have can be seen as he Convex function.
Hence for all of them the local min is the global minima. The loss function for MLP is Non-convex loss function. Which means the local minima and saddle points.

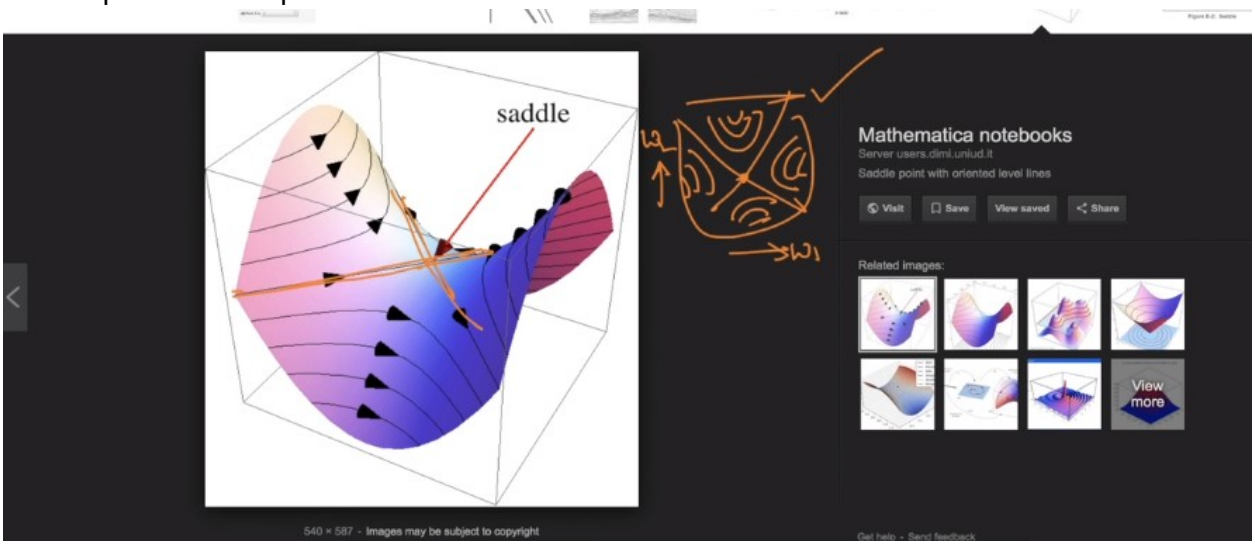


Because it is a non convex function, based on the initial weight we can land up a different minima.

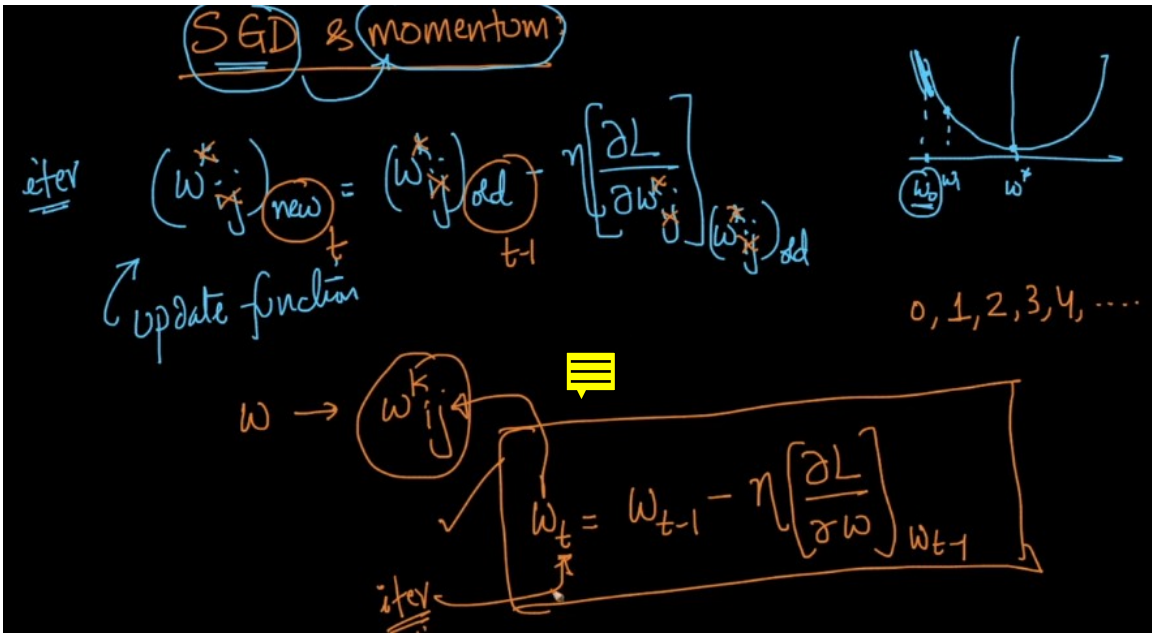
For 3D and contours:



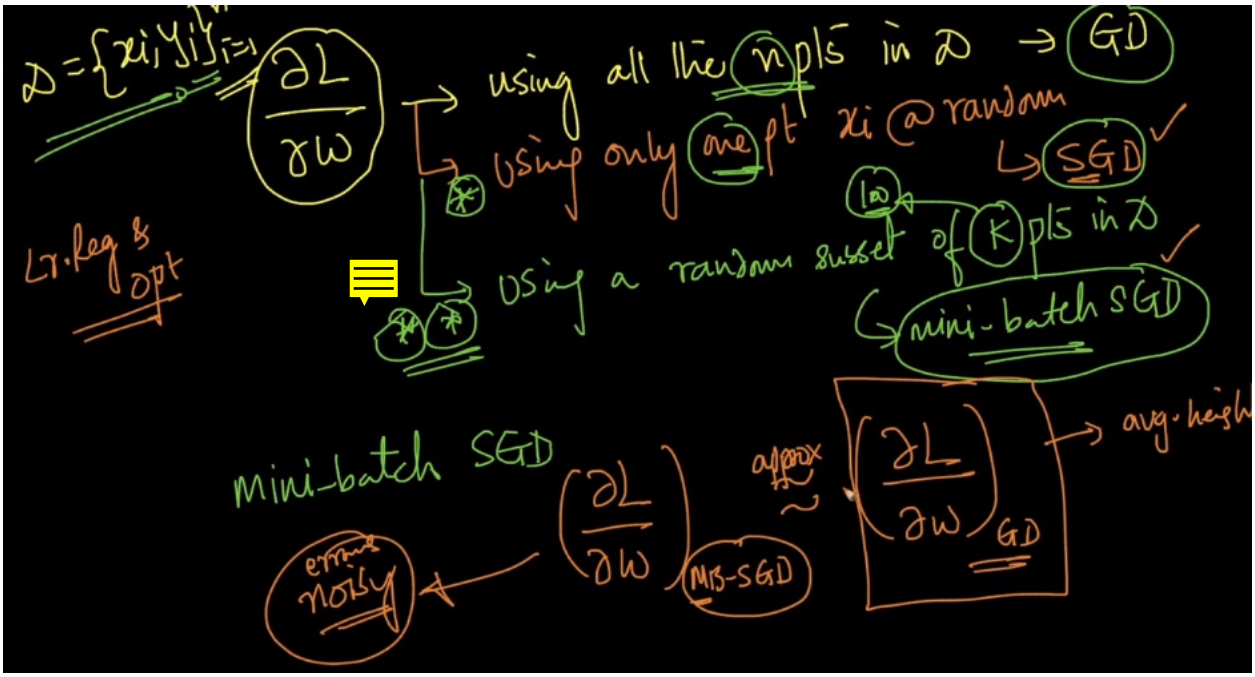
We take all the points at the same height are on the same line. These are called the contours points.
Saddle point contour plot:



Simple mini batch SGD gets stuck at the N-dimensional same.



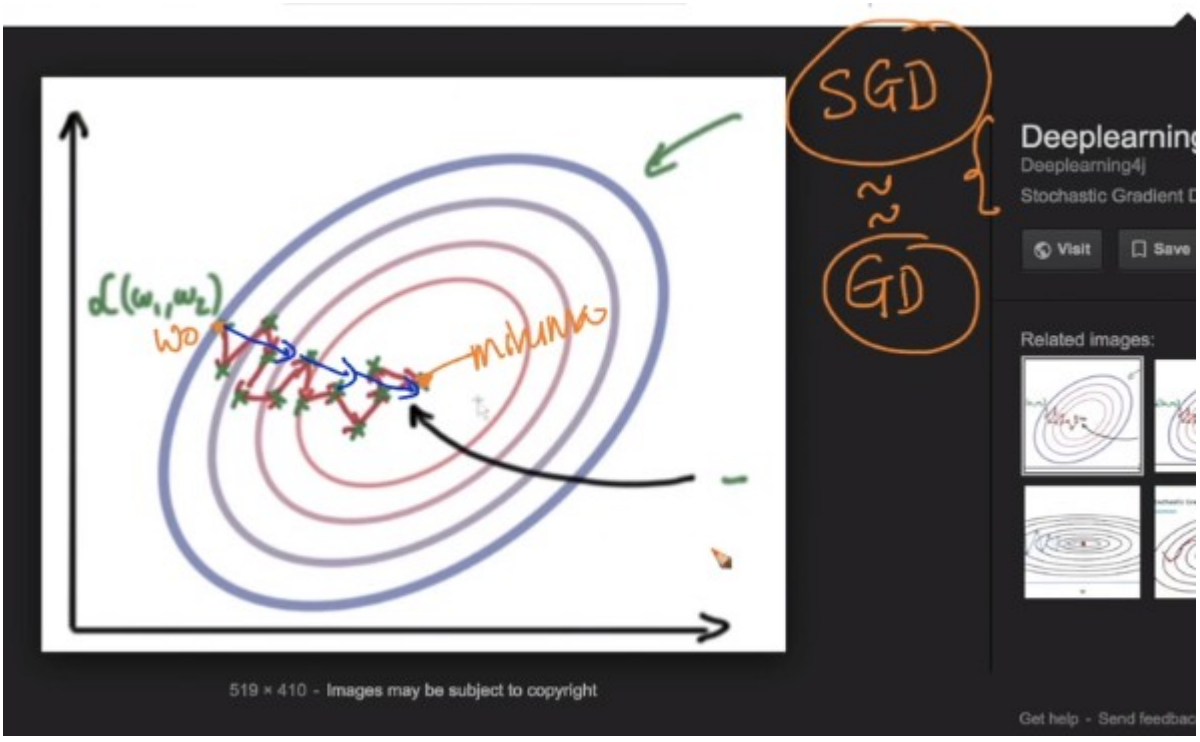
Given this update equation, the hardest part is computing the derivative.



THe SGD is the approx and erroneous.

Gradient descent always move towards minima using all the n points.

Using the SGD:



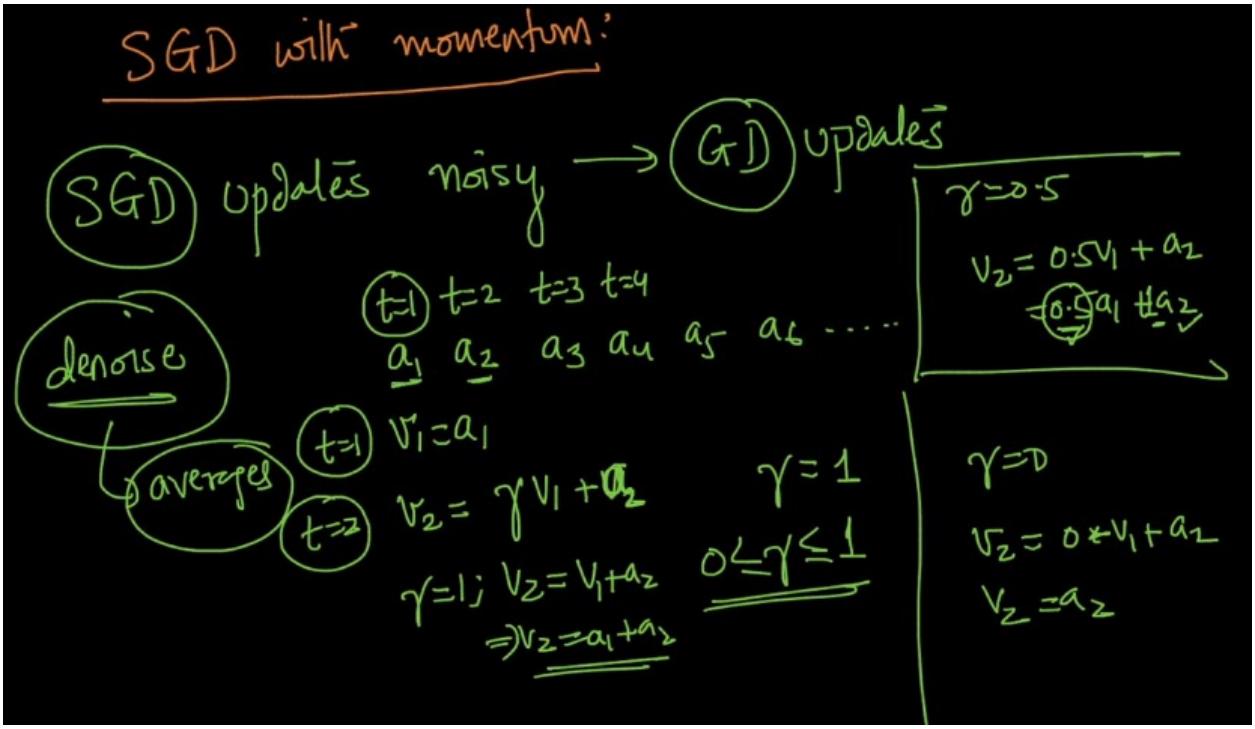
If we run the lots of iterations of SGD we can reach the MIN value as SGD.

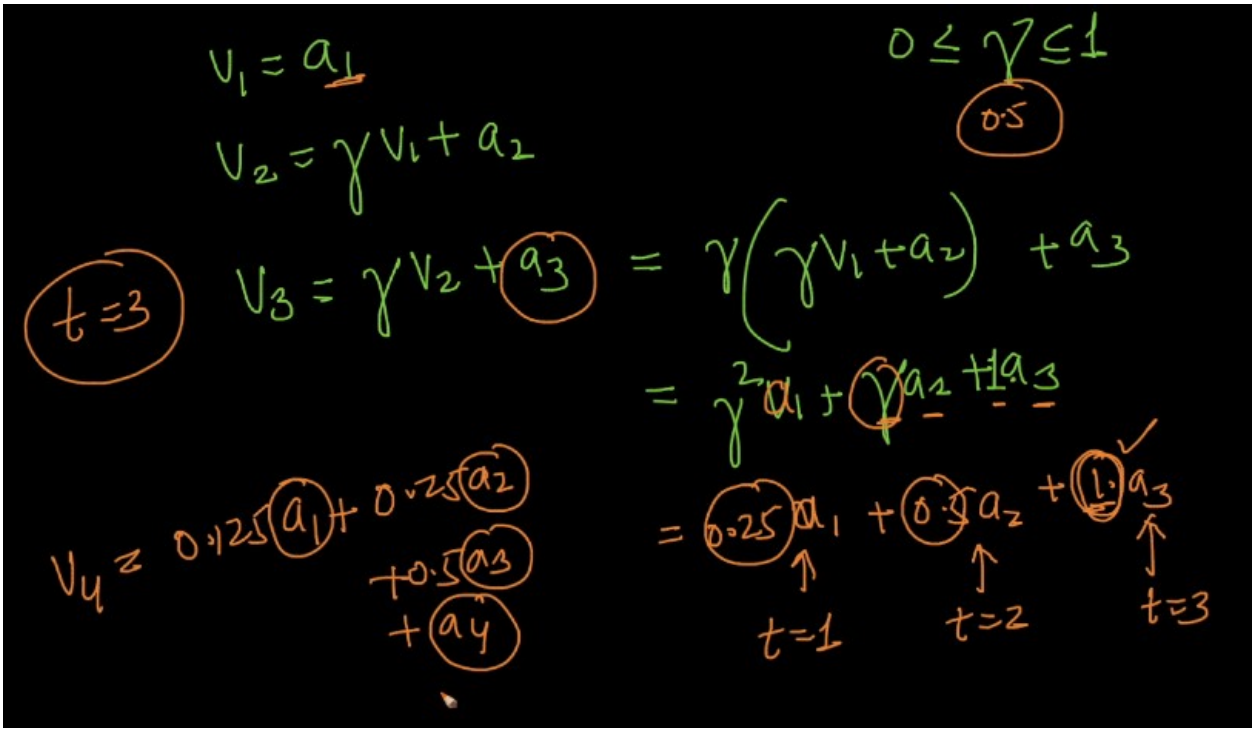
The estimate of the gradient using SGD, each of the updates completely depend of the derivative and more noisy in SGD.

Can we somehow come up with de noisy gradients from SGD.

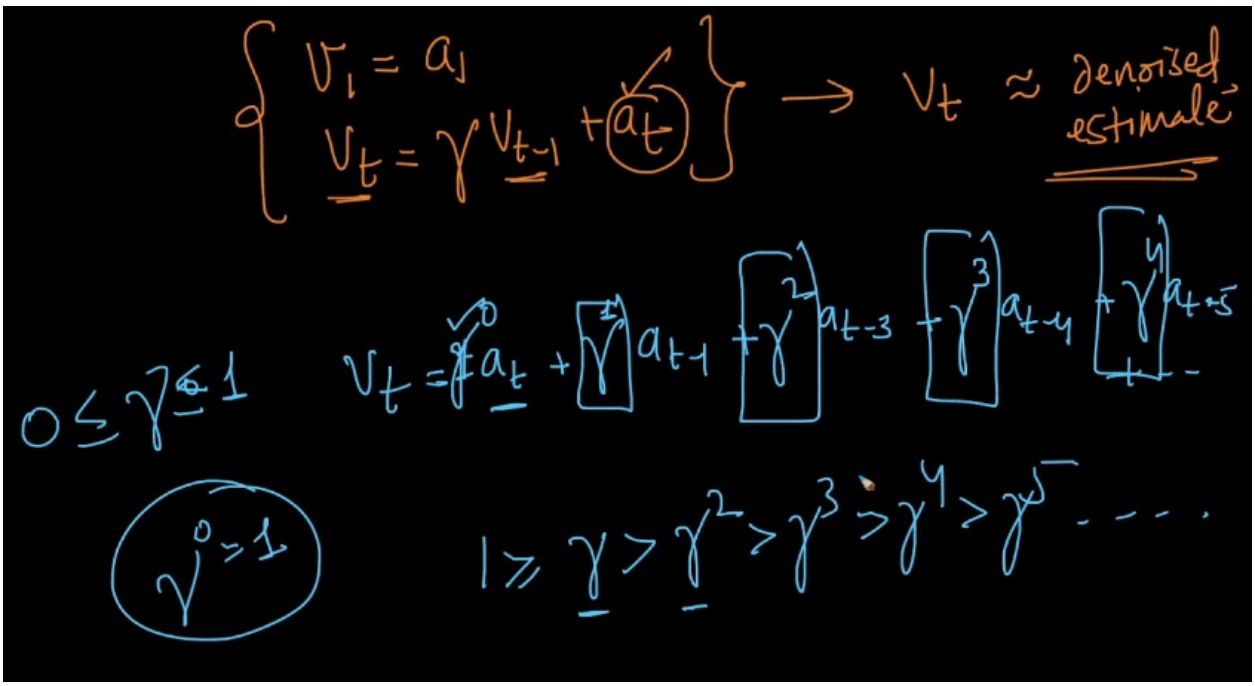
Batch SGD with momentum.

The simple way of de noising is taking averages.

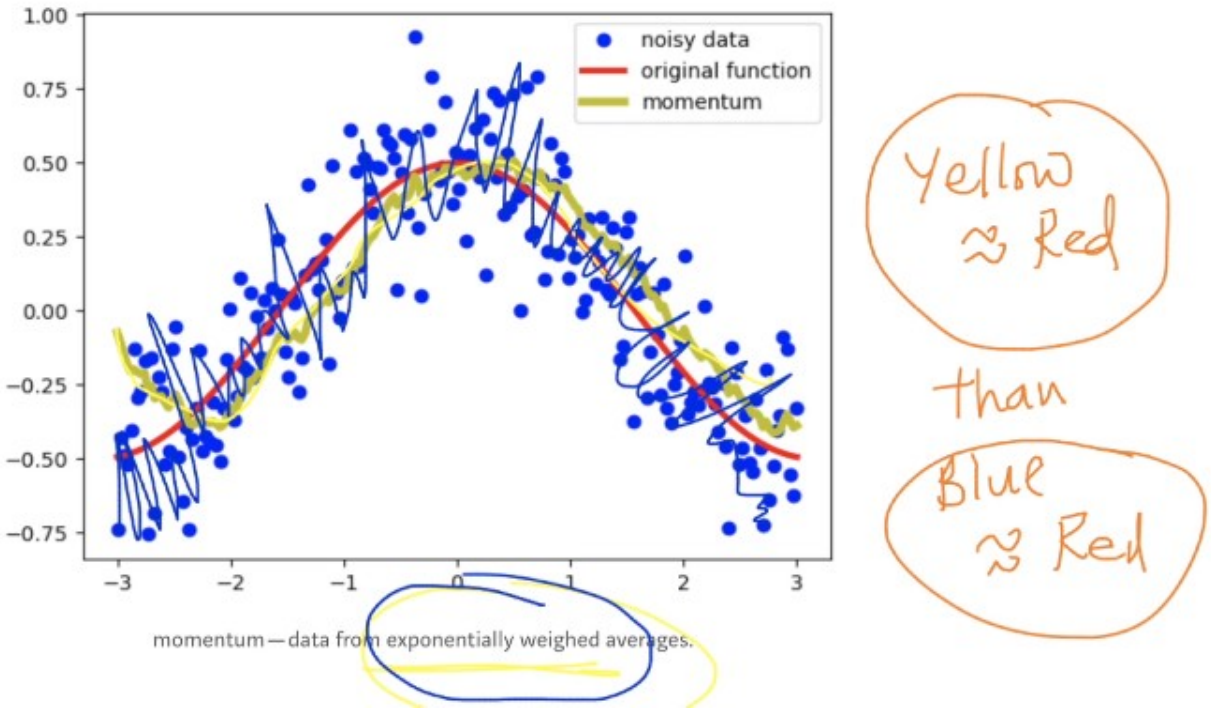




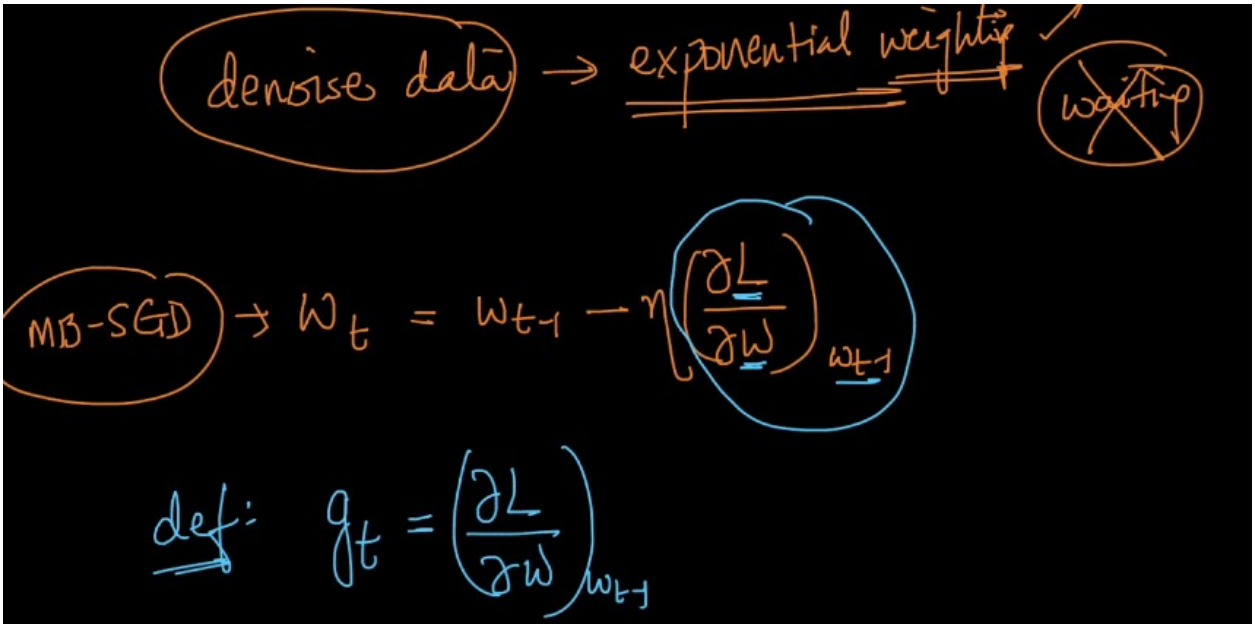
These are called the recursive statements.
As we are getting more and more data we are performing a denoising estimate of the SGD.



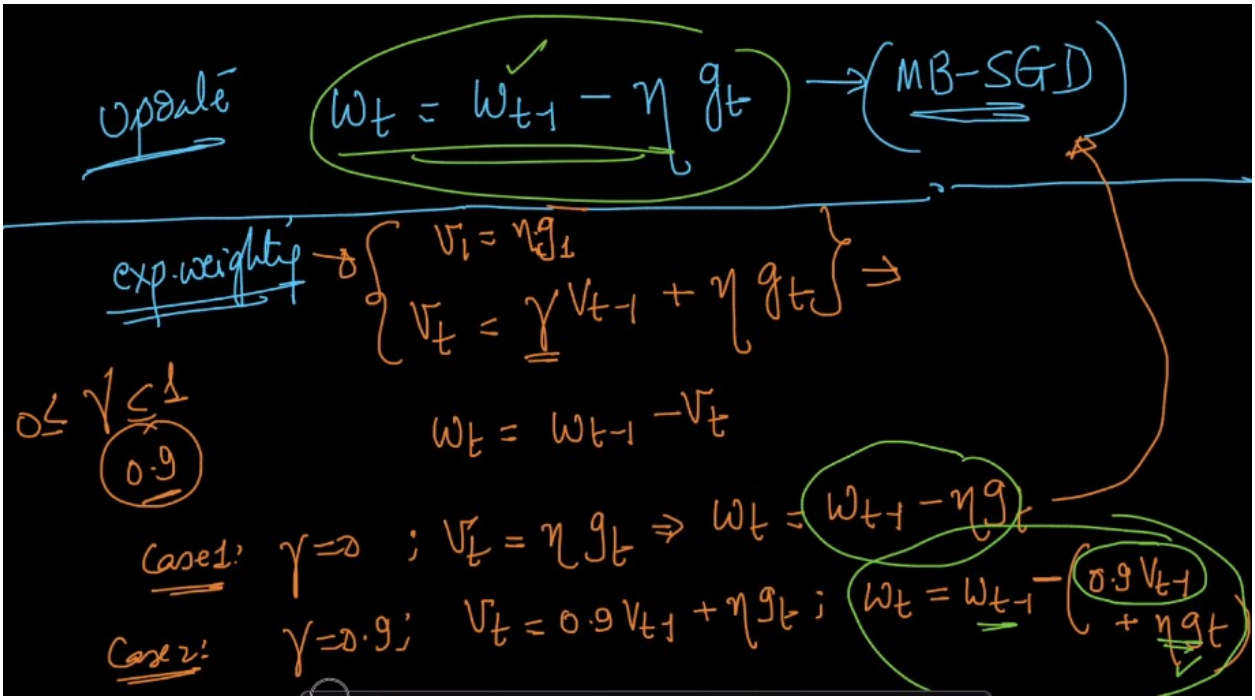
This is called the exponentially weighted average (or) sums.



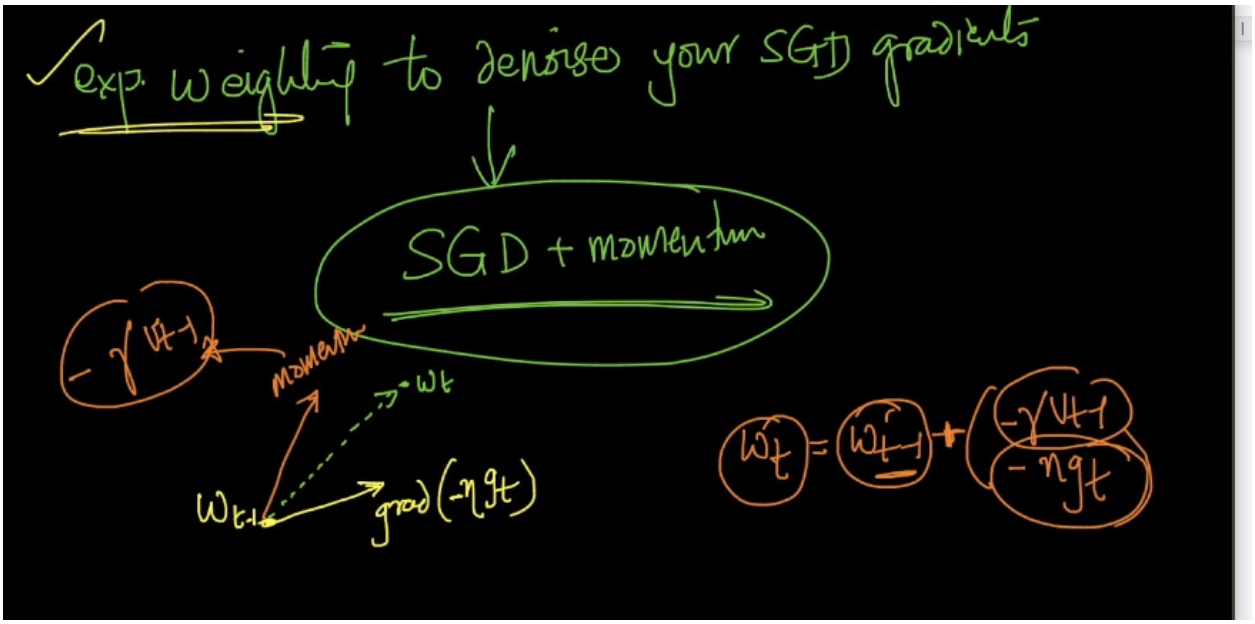
This is how we can de noise the data using exponential weighting.



Standard vs. modified SGD equation:

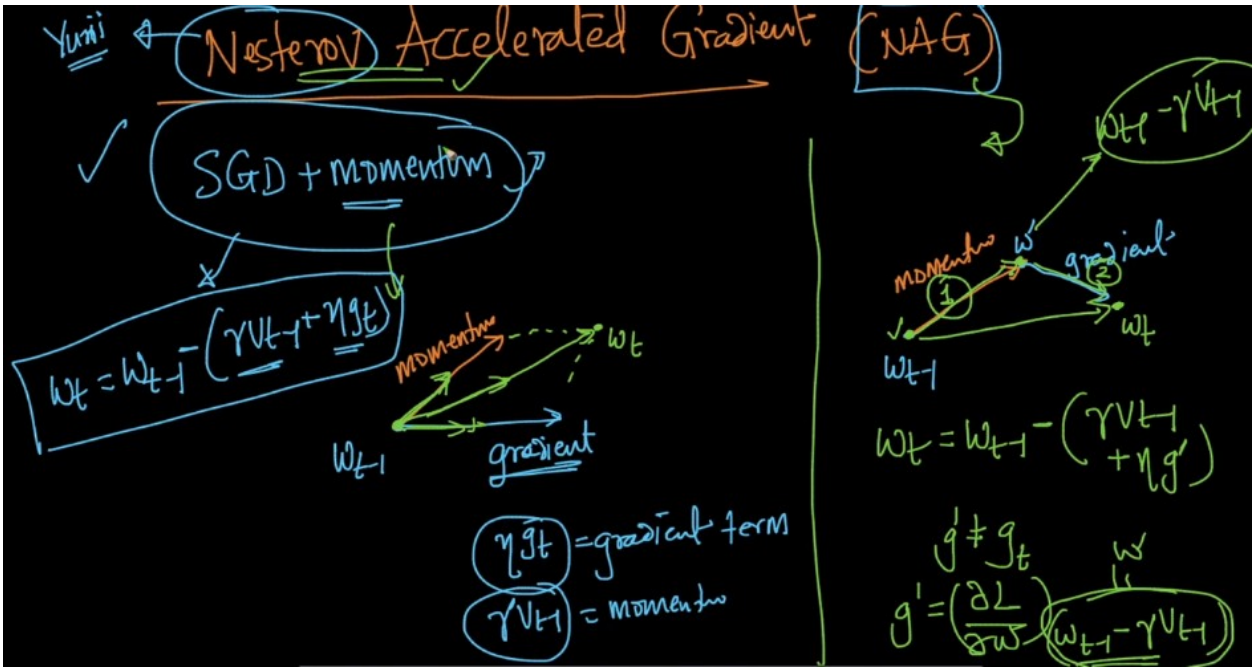


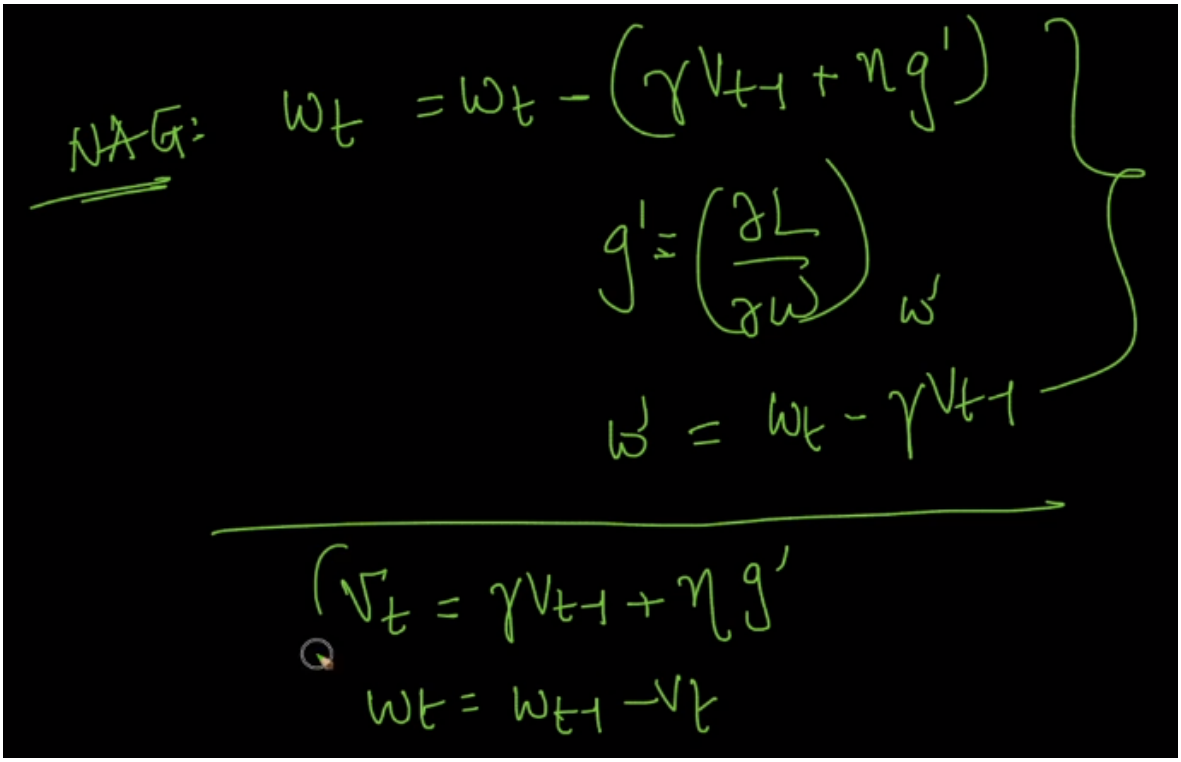
When we use exp weight then we get SGD + Momentum.



Nesterov Accelerated Gradient (NAG):

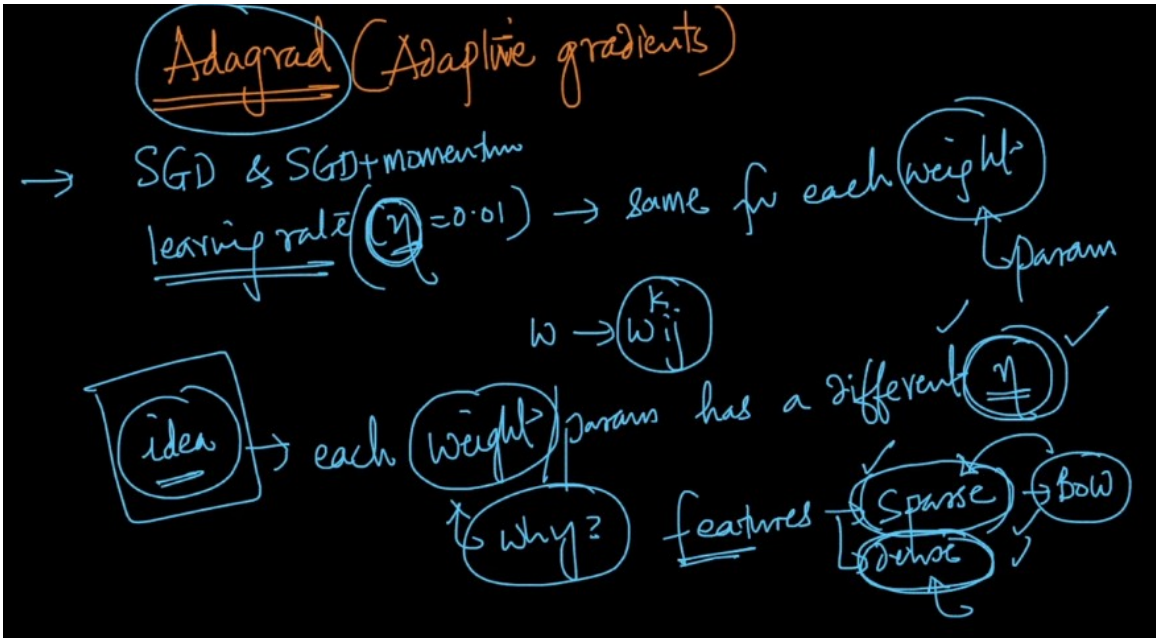
We have seen SGD with momentum, there is a related algorithm called NAG.



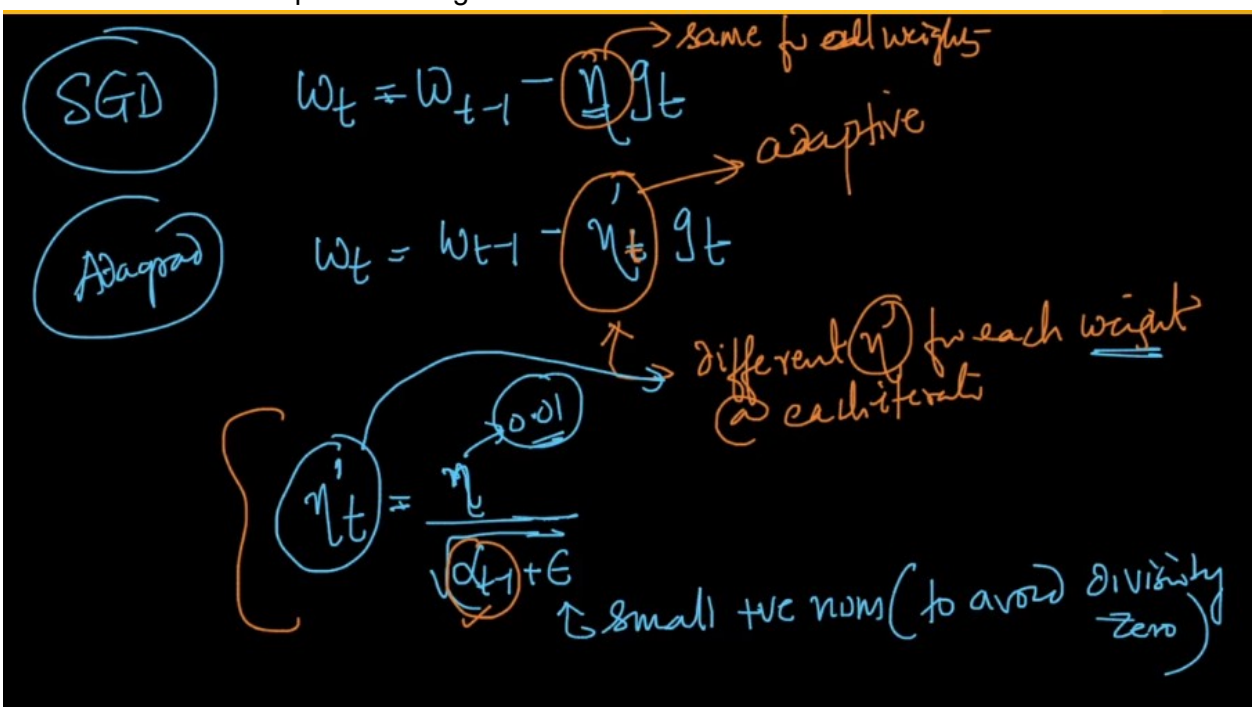


Adagrad:
Learning rate is set to small, the learning rate is same each weight in SGD.
Each weight has a different learning rate in Adagrad.

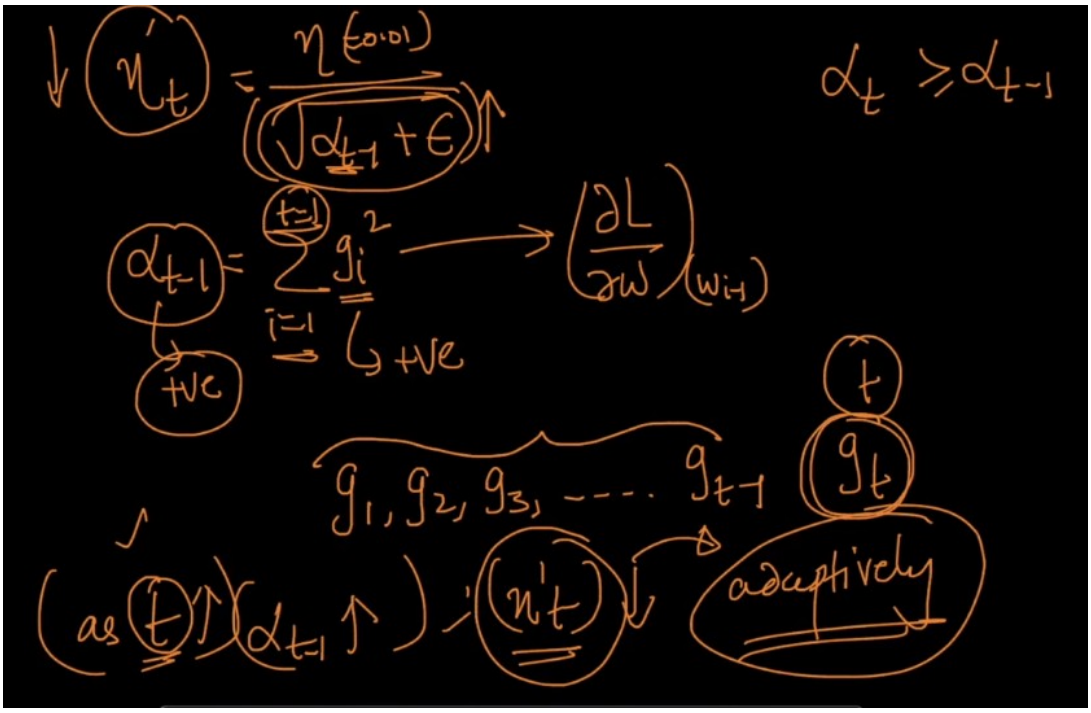
In our original datasets, some features which are dense and sparse.



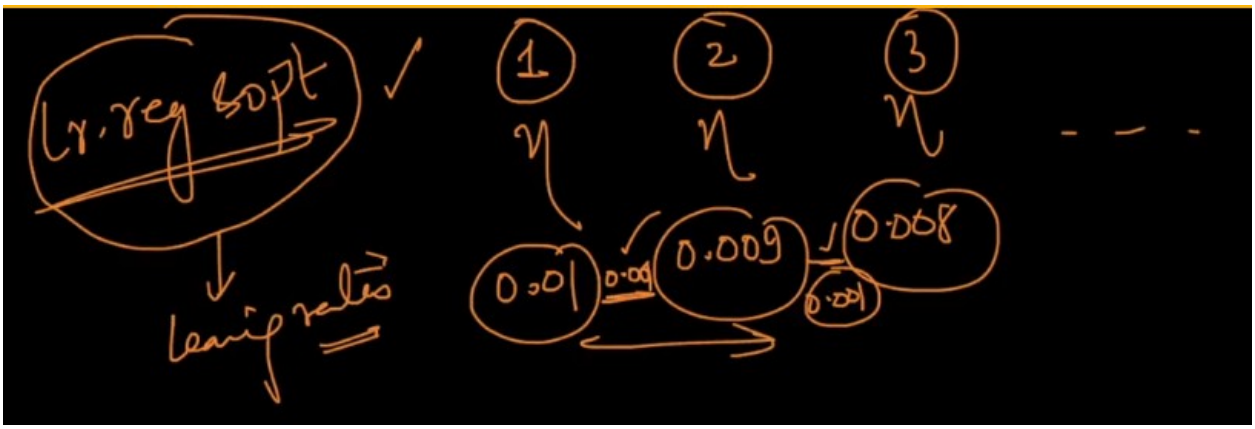
AdaGRAD - Is the adaptive learning rate.



Adaptive Learning rate formula:

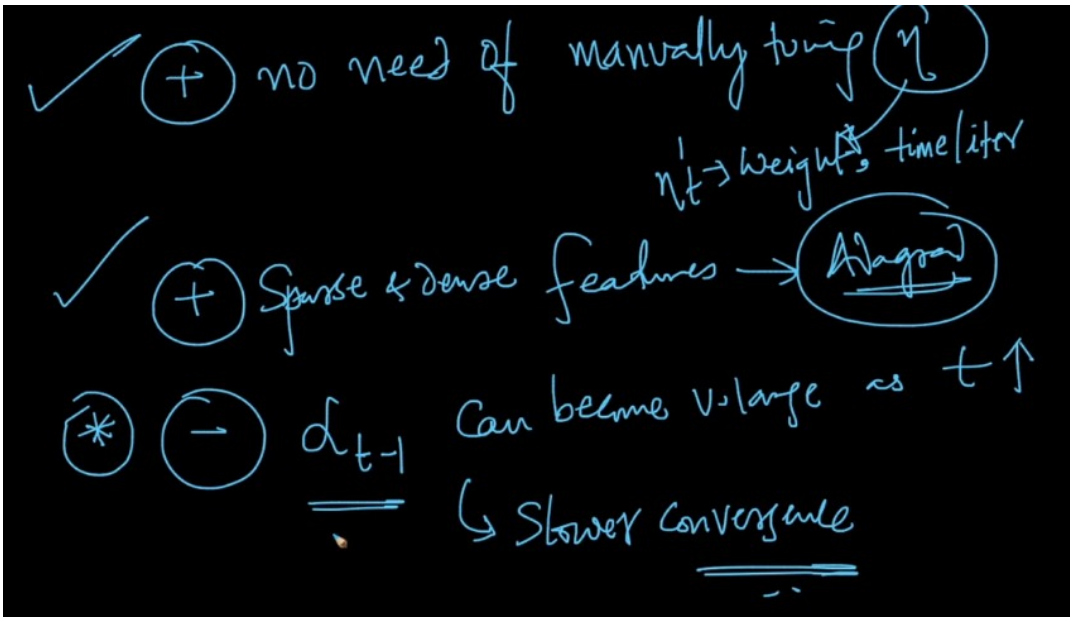


This is iteration based decay.

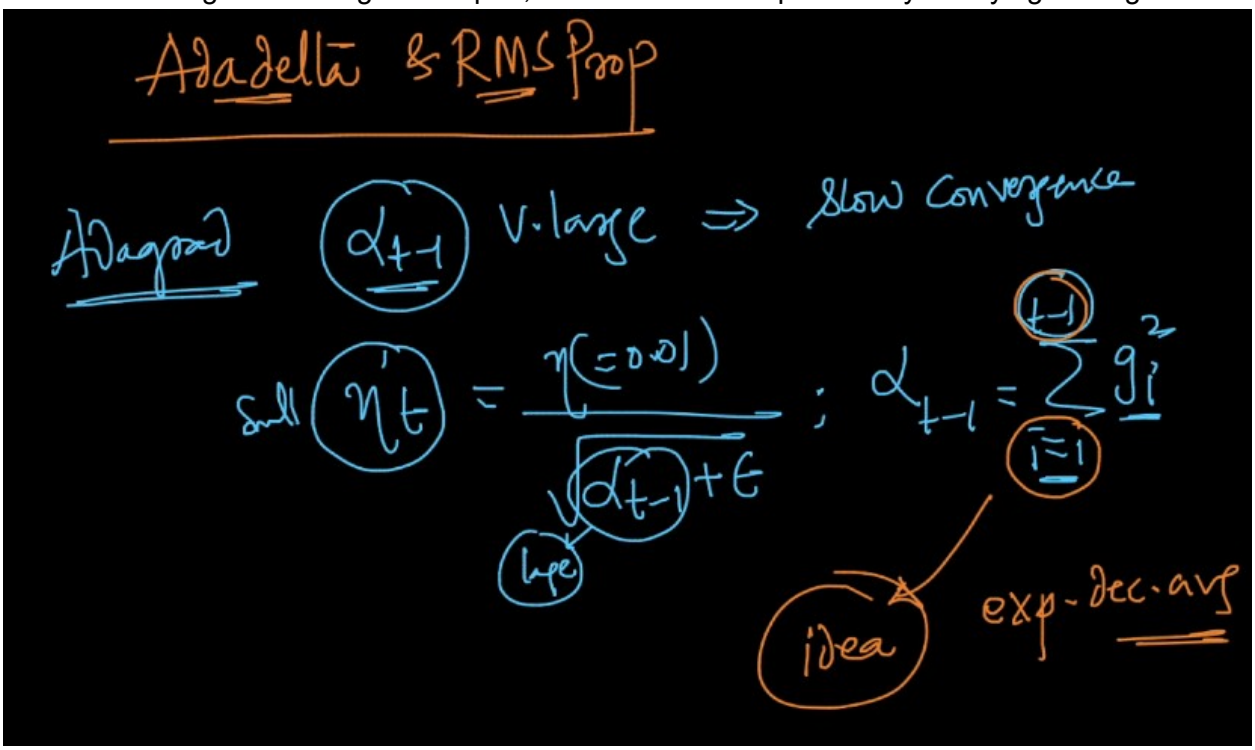


Major advantage is there is no need of manually tuning the learning rate.

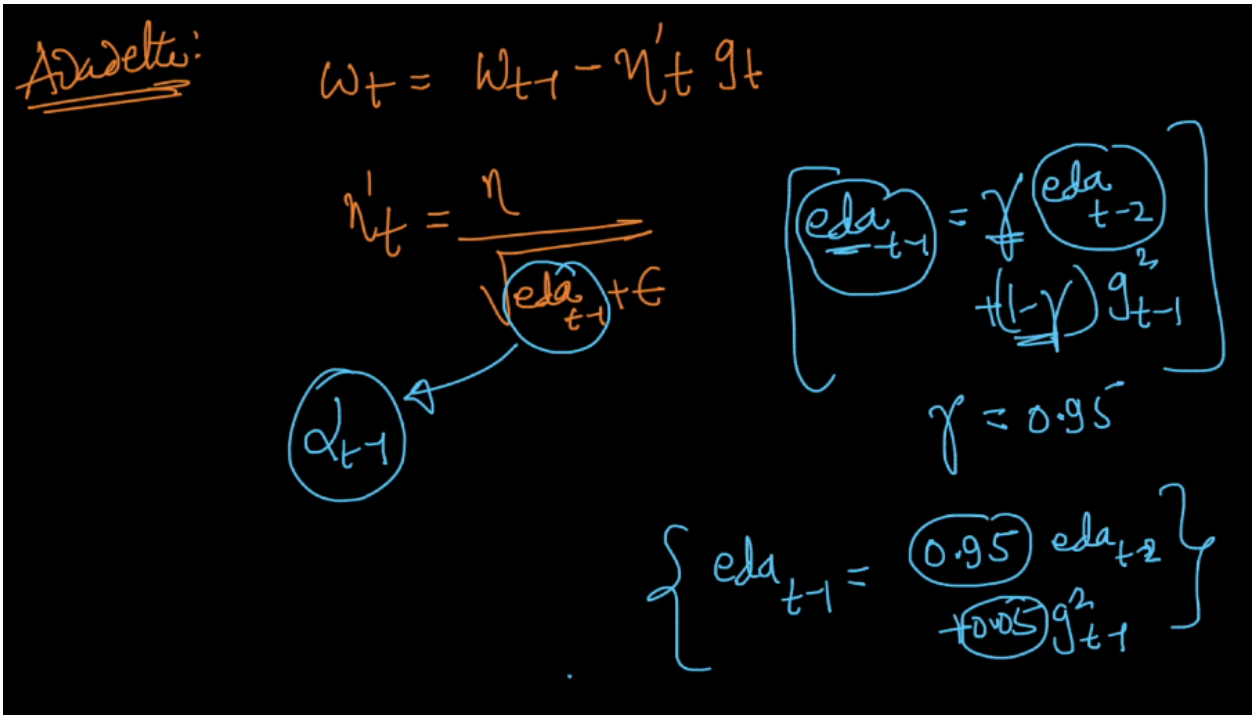
THe alpha values can become very small, then the weights cannot converge. This could take slower time to converge.



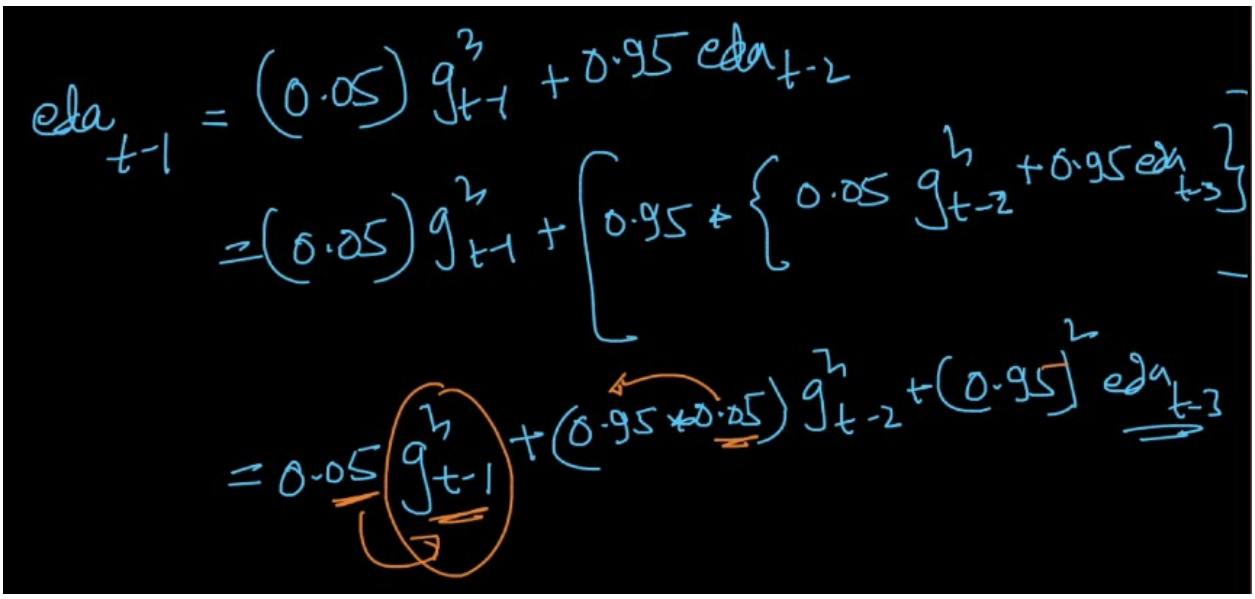
This problem is fixed in next algorithms.
Adadelta and RMSProp:
AdaDelta:
Instead of taking all the weights in alpha, we will take the exponentially decaying average.



We will replace with exponential decaying average.



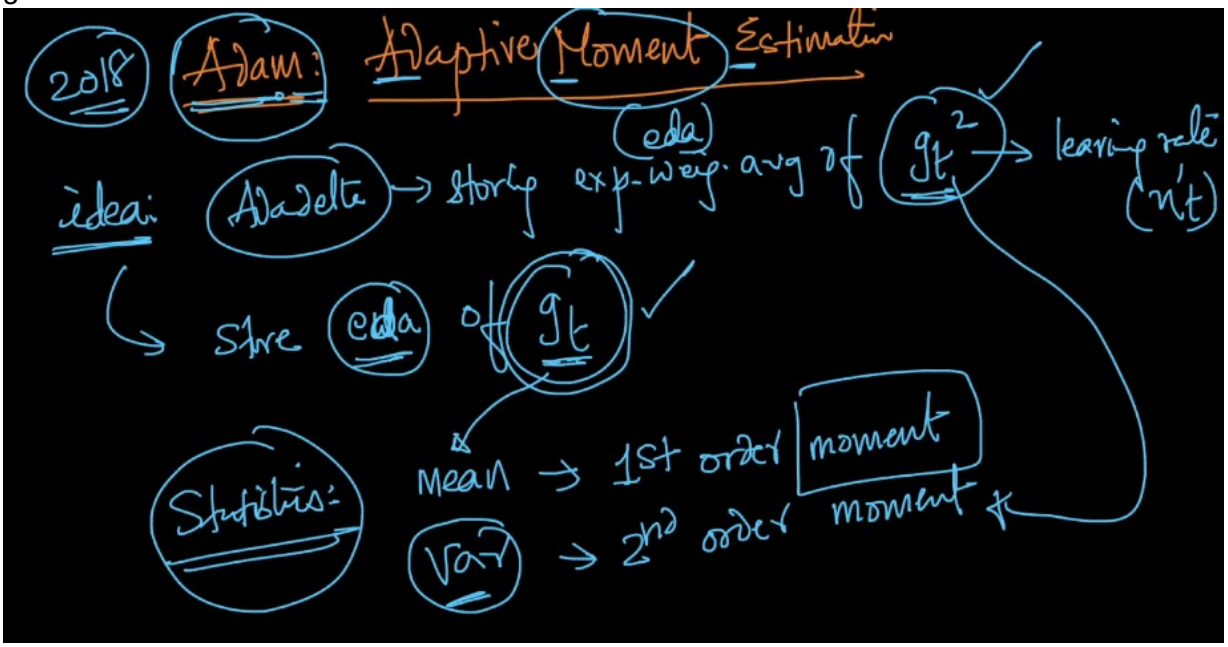
Expanding EDA (t-1):



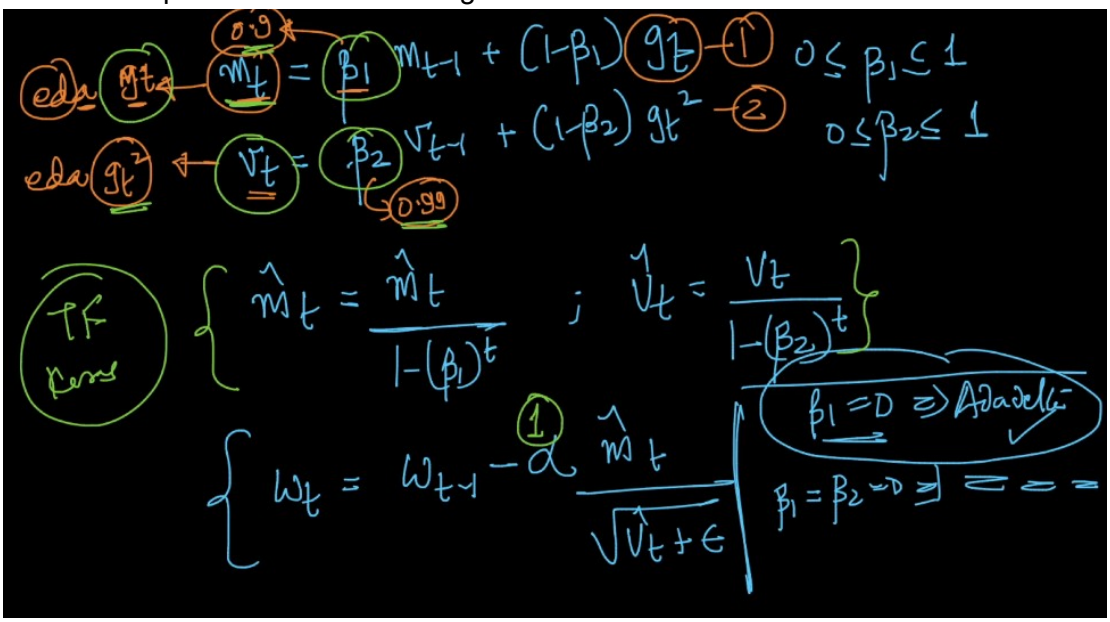
In a nutshell, take exponential weighted averages of gradient^2, rather than the sum of squares of all the gradients.

Adadelta has the faster convergence.
Adam:
It is most Popular in algorithm.

What if i store the exponential weighted average of the gradients it self, than the squares of the gradients.



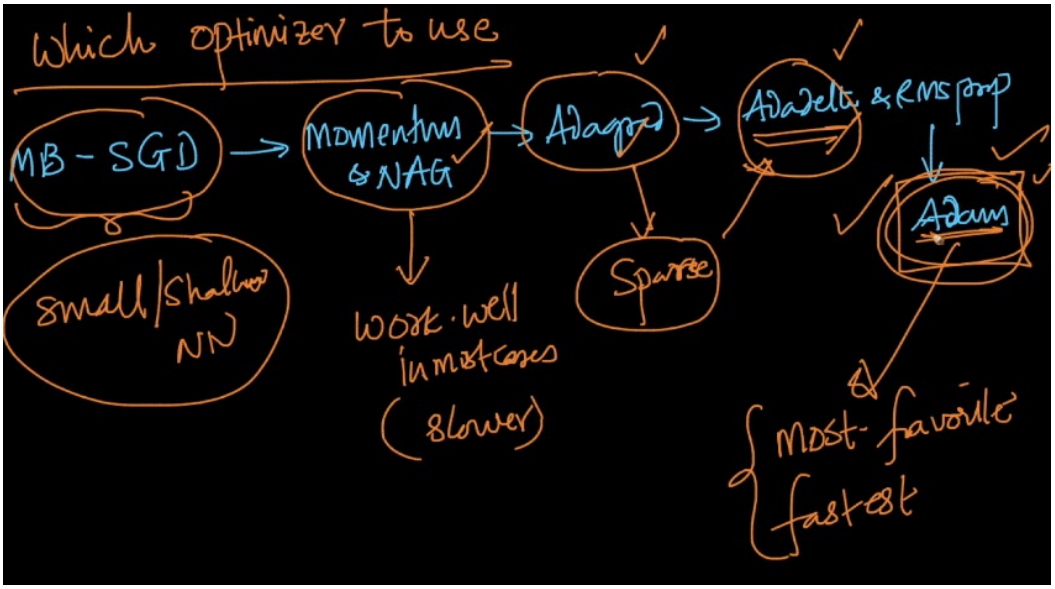
This is a moment estimation algorithm.
The three equations of the Adam algorithm.



Which algorithm to choose when ?

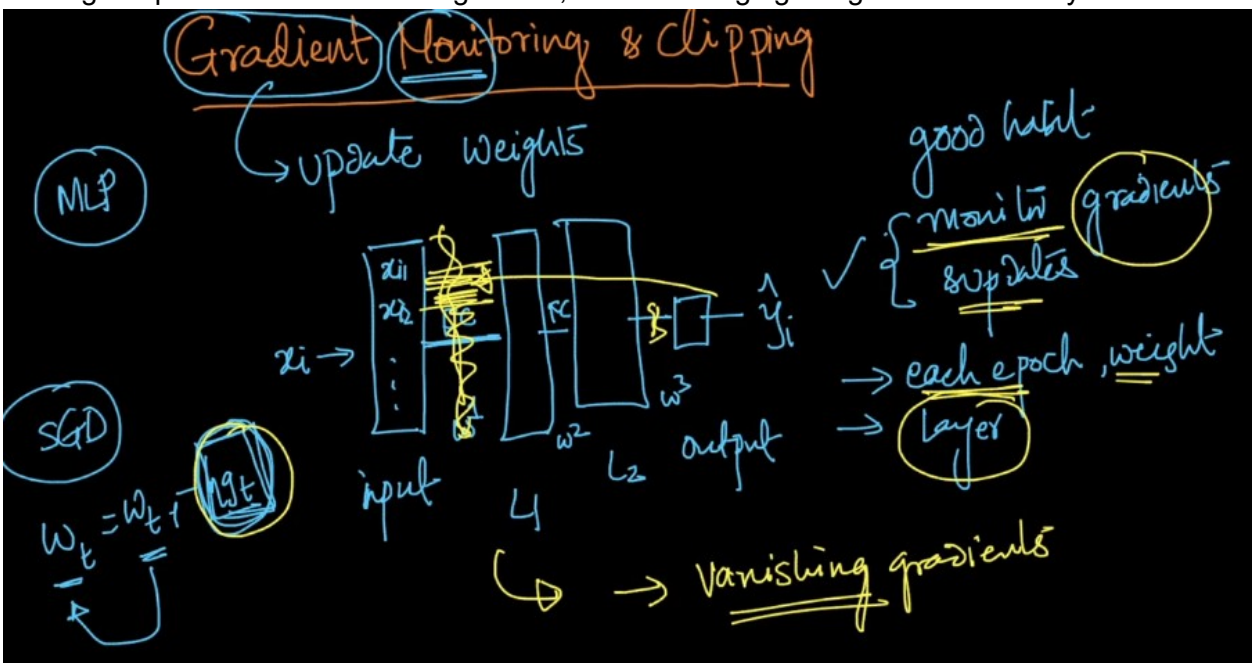
Refer to the blog below:

https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f

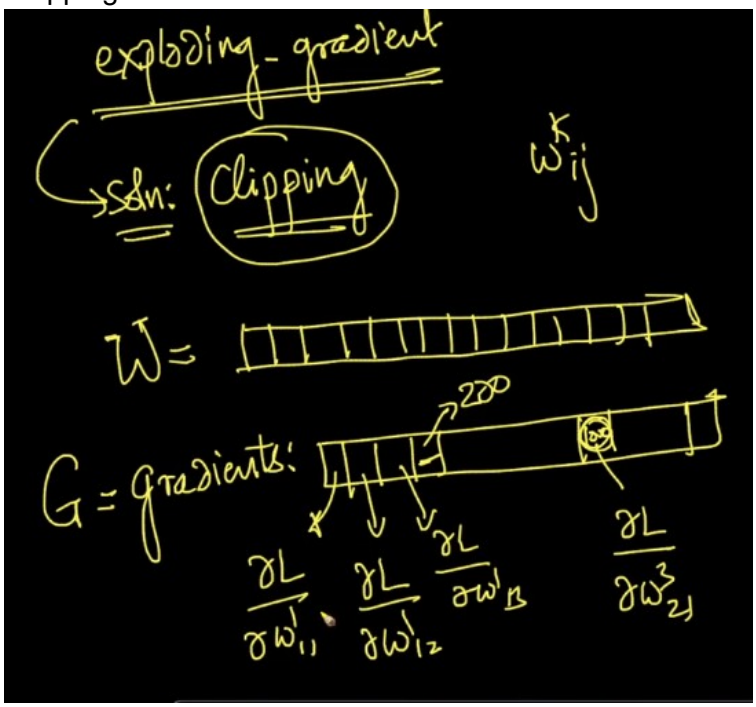


Gradient Monitoring and Clipping:

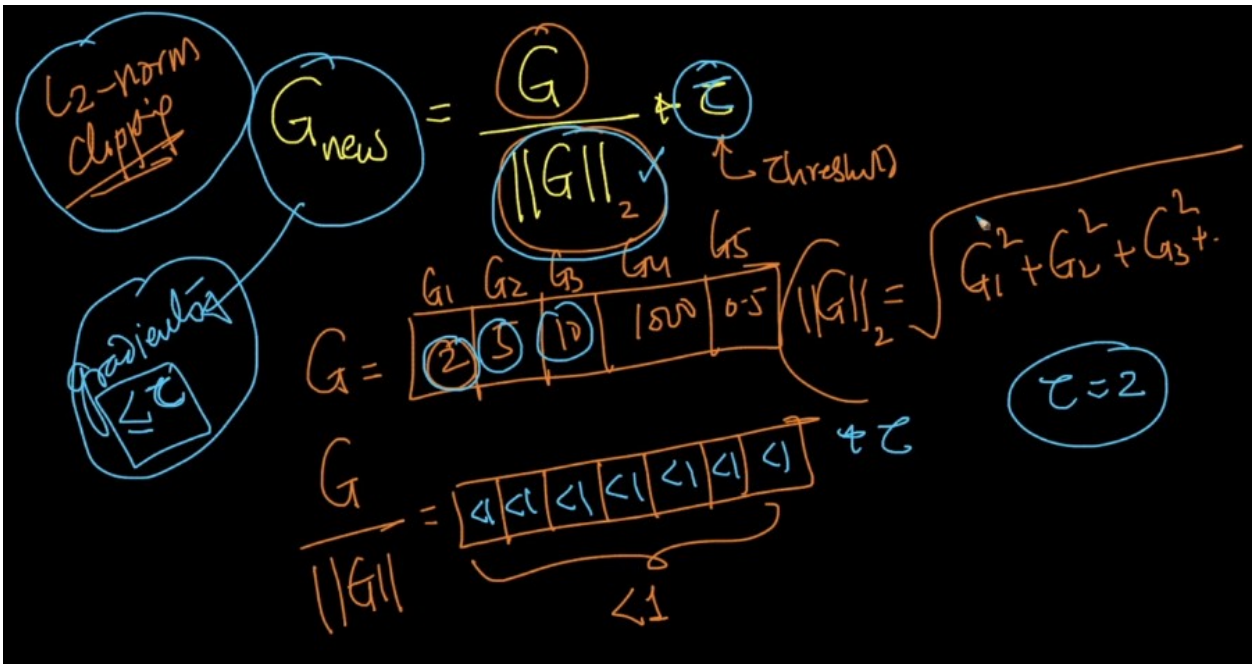
It is a good practice to monitor the gradient, we are changing the gradients at every iteration.



Solution is gradient Clipping:

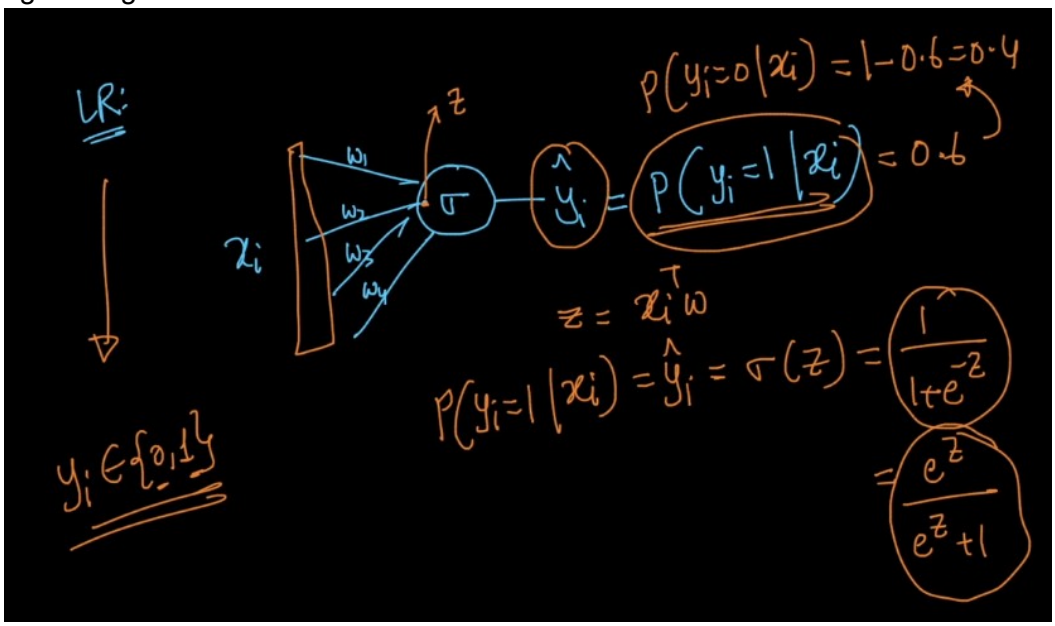


The concept is as follows:

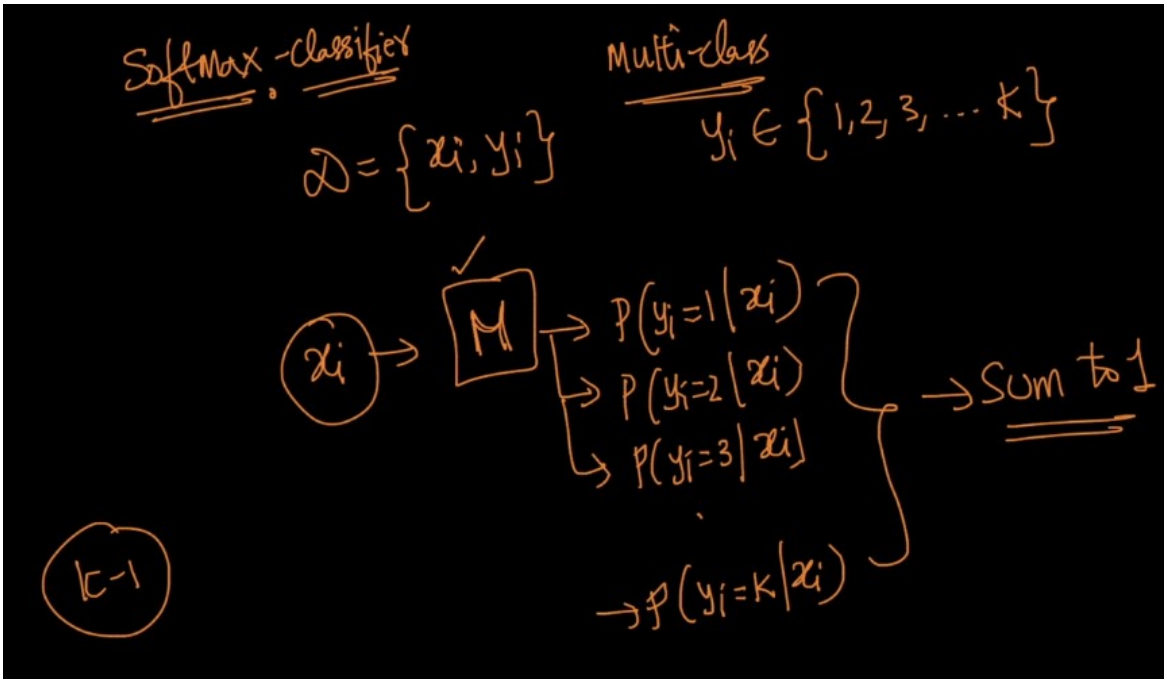


Monitoring gradients is crucial in NN.

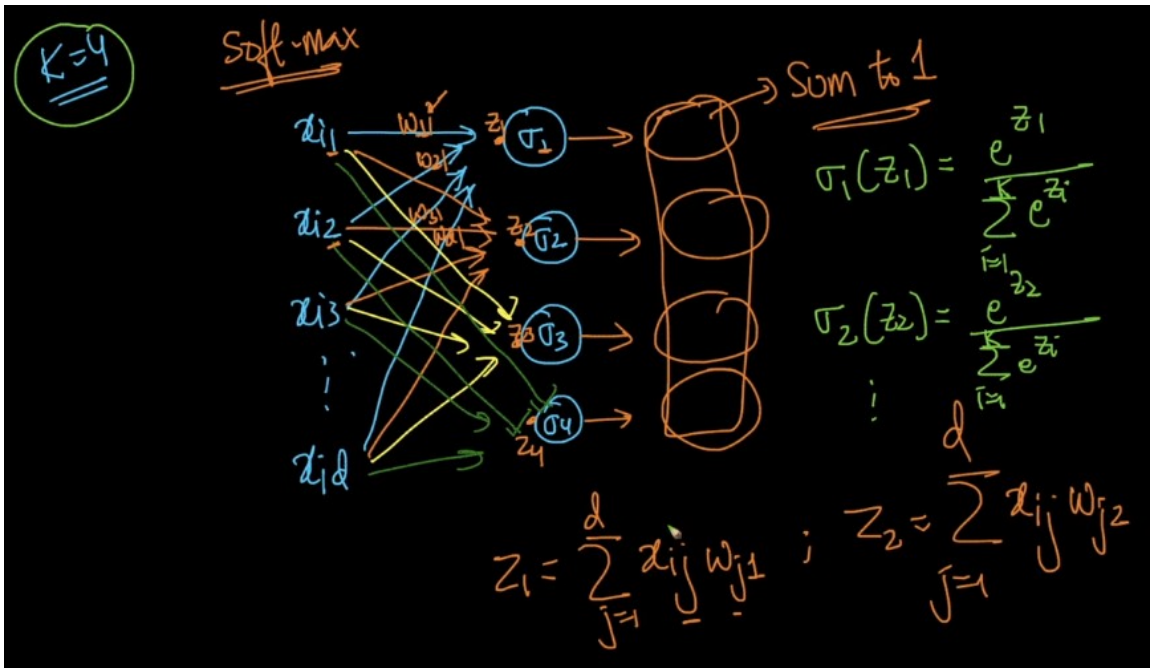
Softmax and Cross-Entropy: Logistic regression to multiclass is called softmax.
This is logistic regression:



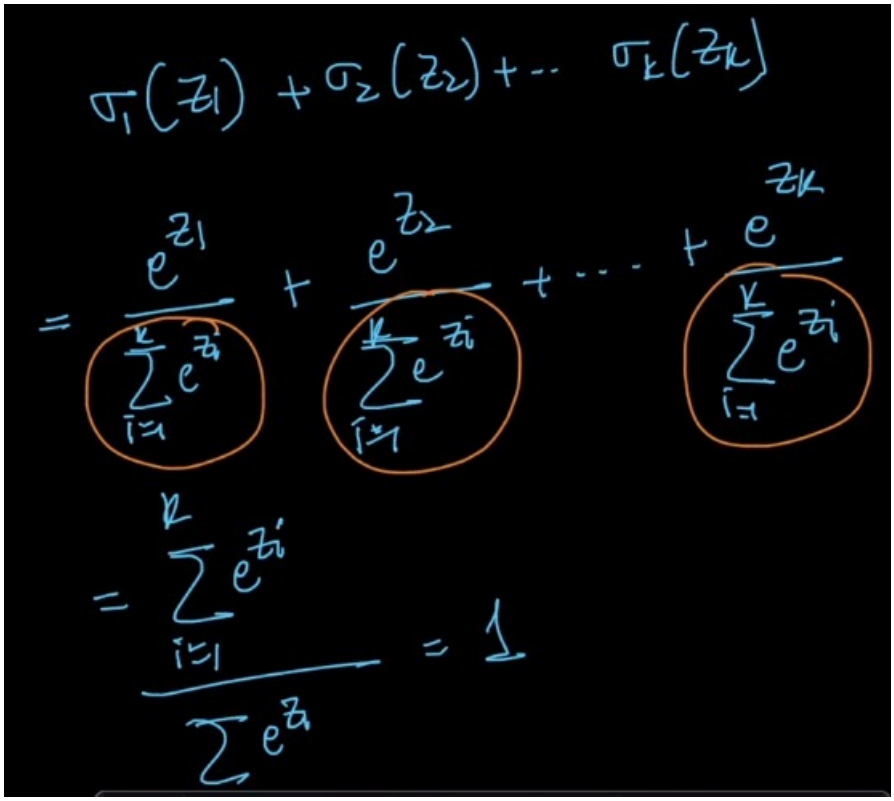
Softmax Classifier:
Here yi belongs to “K” such classes. Here the summation of all the class belongs to 1.



In the case of softmax classifer, the input that I get at each of the neuron is calculated as follows:



Formulation:
This satisfys our requirement.



Softmax is the generalization of Logistic regression to multi class setting.

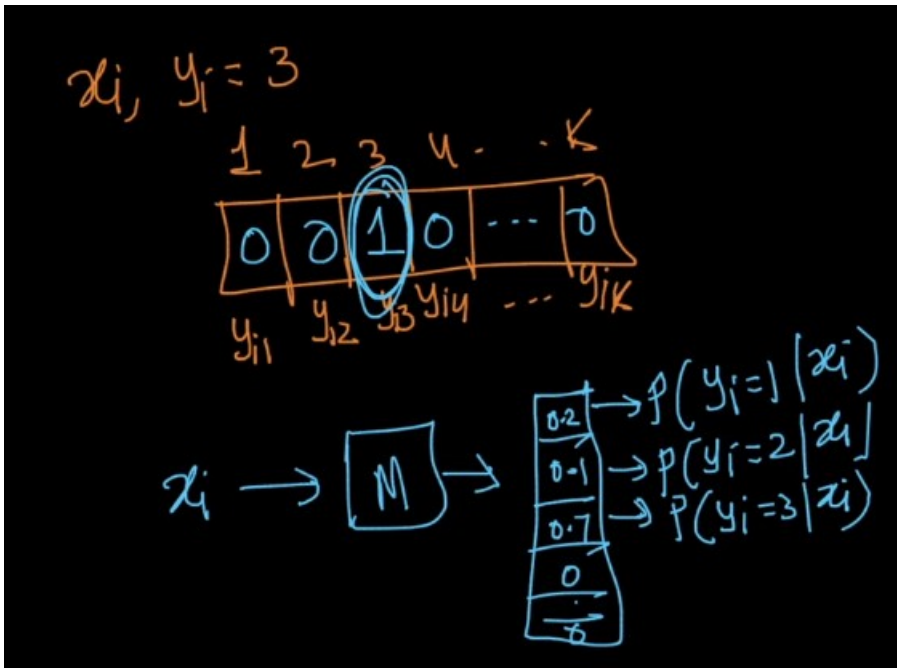
It minimizes the multi – class log loss (Or) Cross - Entropy:

moment.

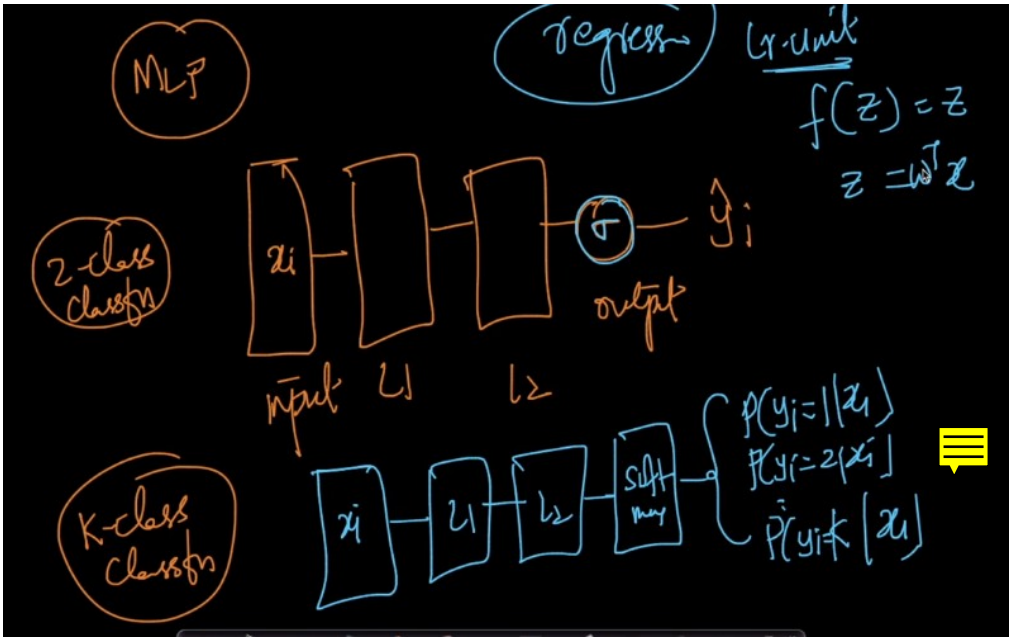
er to calculate Log Loss the classifier must assign a probability to each class rather than simply ng the most likely class. Mathematically Log Loss is defined as

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{K} y_{ij} \log p_{ij}$$

N is the number of samples or instances, M is the number of possible labels, y_{ij} is a binary indicator ether or not label j is the correct classification for instance i, and p_{ij} is the model probability of

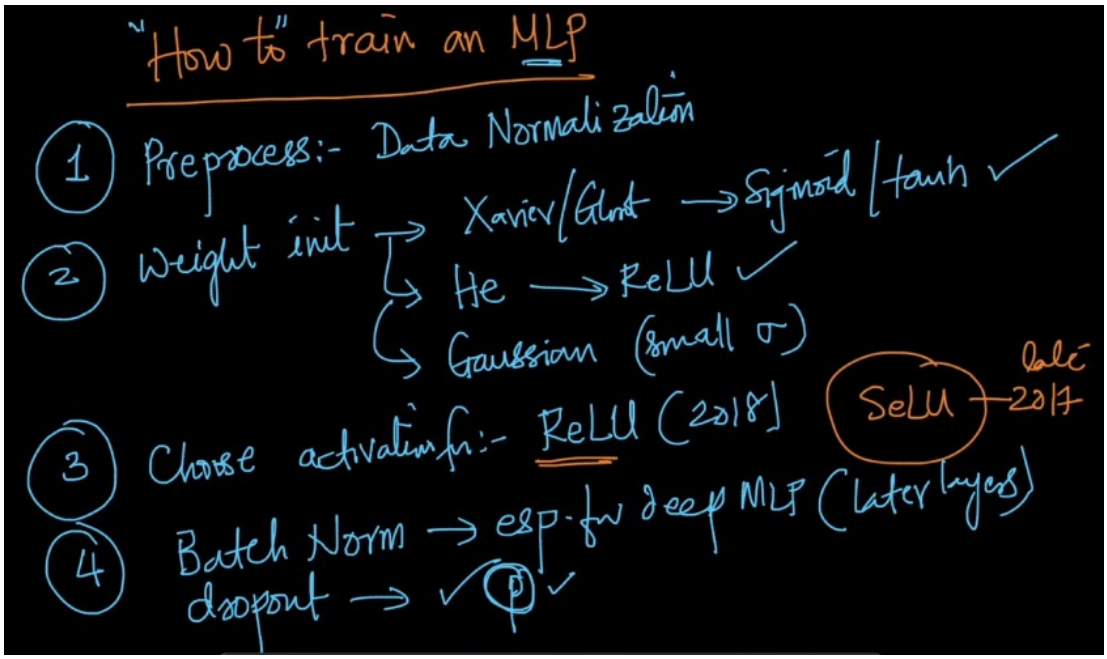


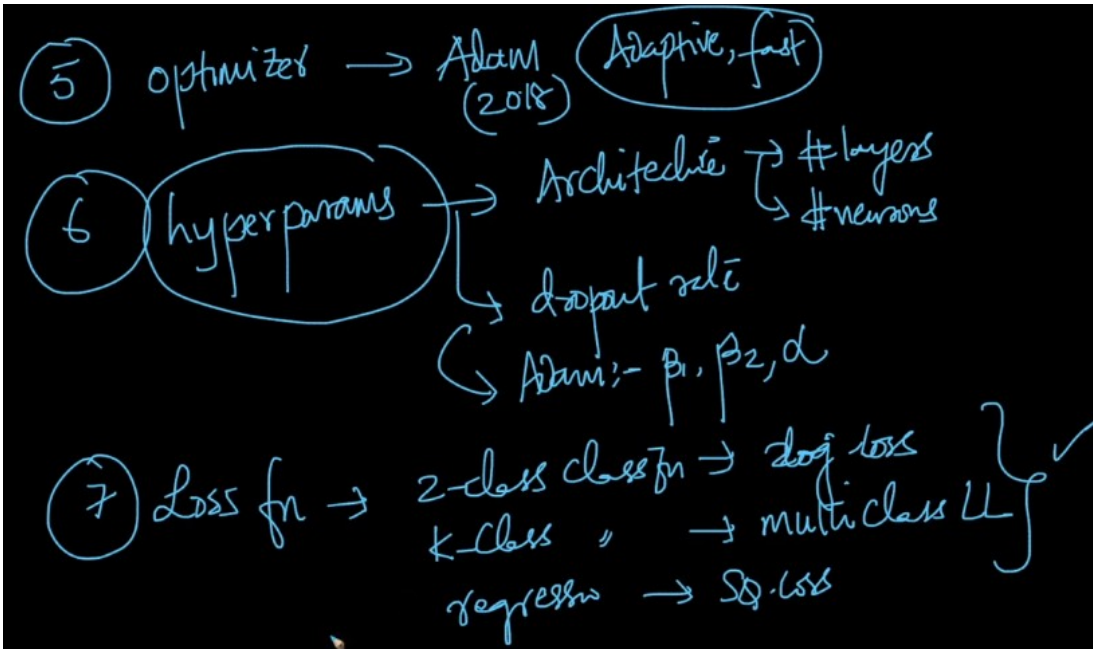
Example of usage of Softmax:
This generates the probability of the class belonging to the each class.



A linear unit looks like this for regression:
It takes the whatever the input and gives the same this is linear unit. We can use the simple squared loss.

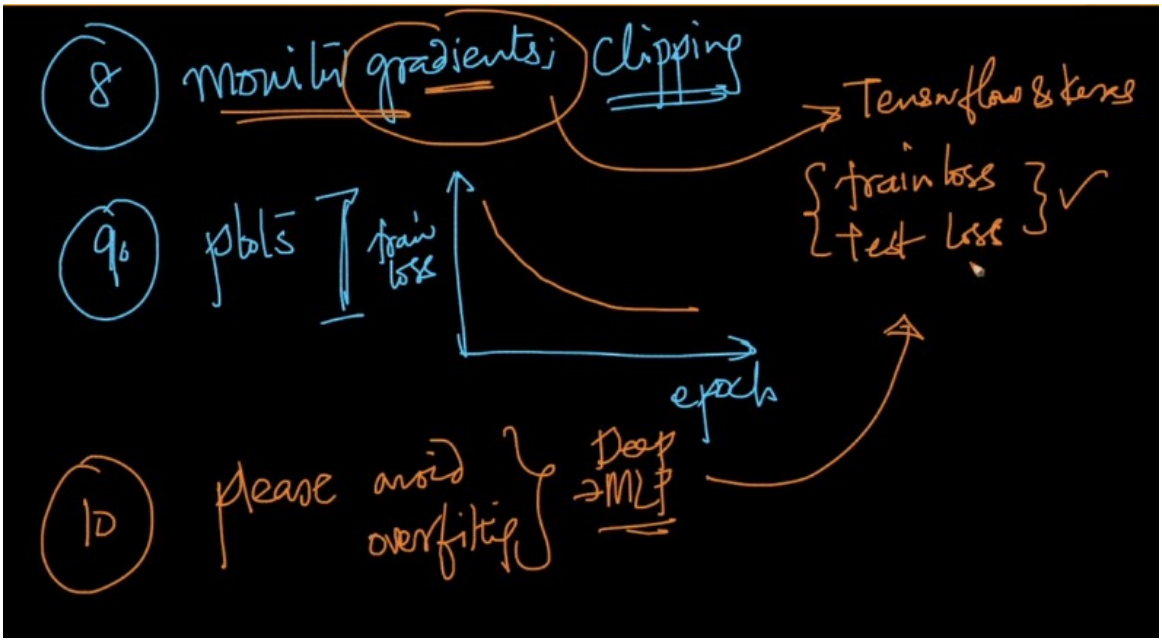
Summerize the MLP's:





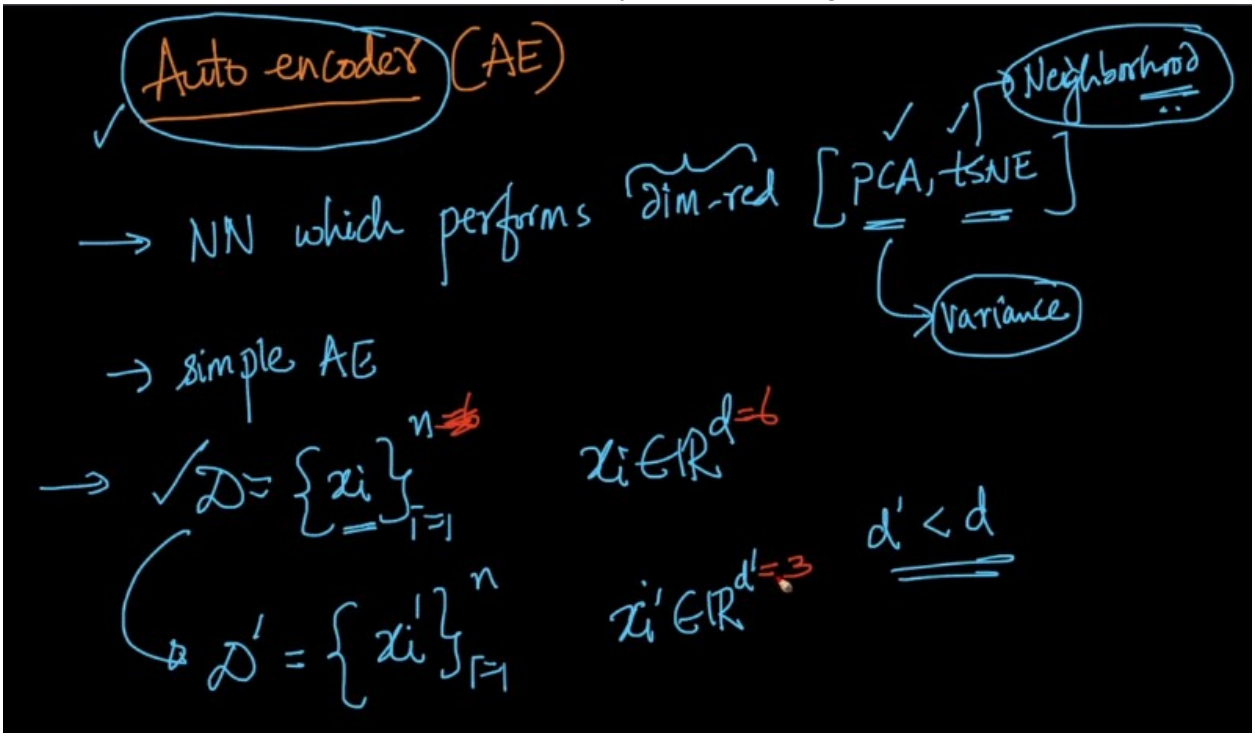
The very important is always monitor gradients, and apply gradient clipping.

We can easily overfit the neural networks.

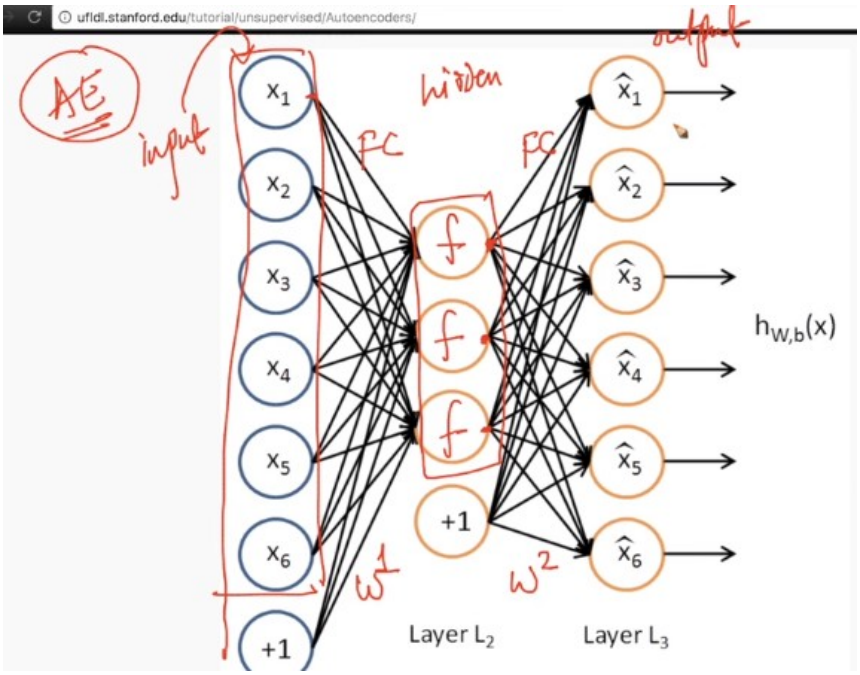


There are statergies we can overfit and forgot the any one the model can go crazy.

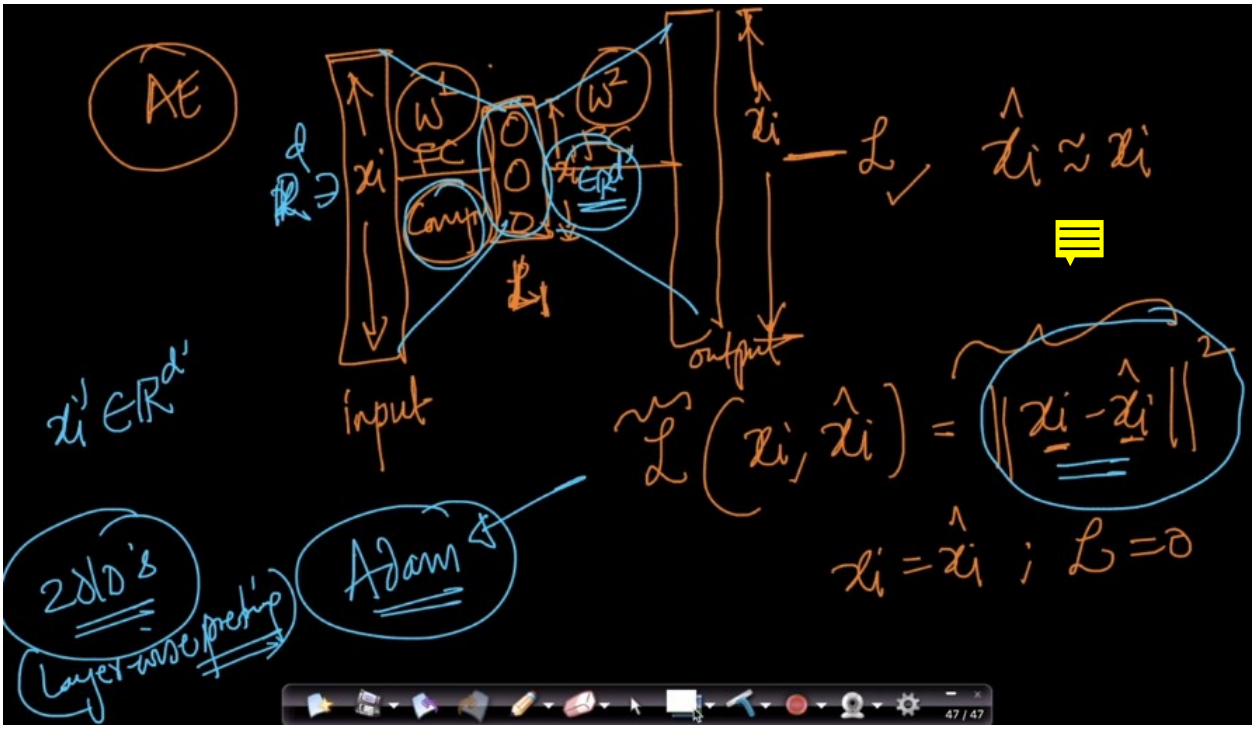
Auto Encoders: This performs the dimensionality reduction using the NN.



We need to get the three dimensional output for the six dimensional input.



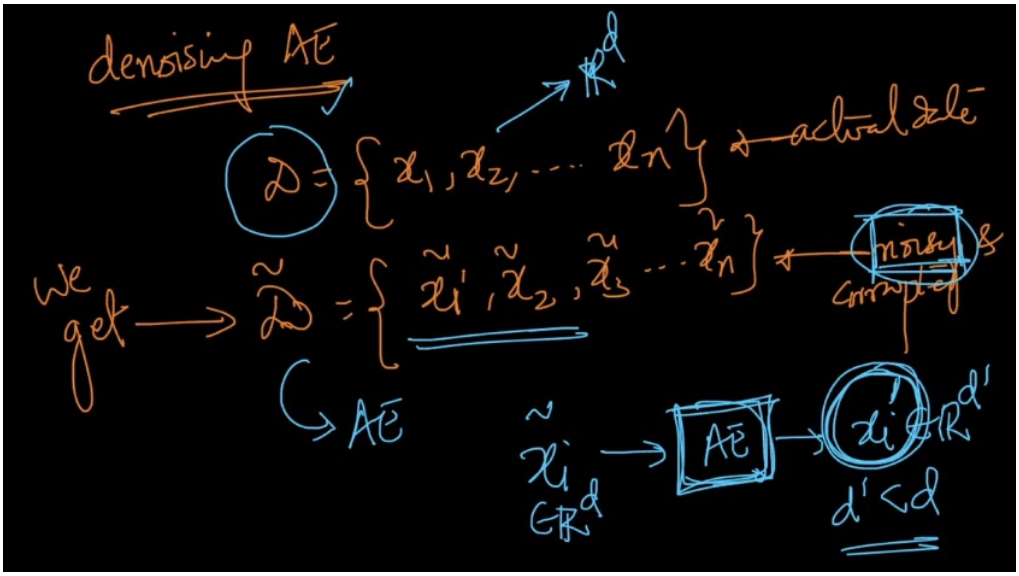
The output that we will predict the X it self. We can conclude the middle layer preserves the input data.



Auto Encoders reference:

https://en.wikipedia.org/wiki/Autoencoder

Deep AE requires the dropout layer.
There are some variations to the AE called denoising autoencoders.



Ppl intentially add noise, this learns the data and leaves the noise and makes the robust noise free encoder.
Sparse AE:
We will apply loss function with L1 regularization, If we add the L1 reg sparse autoencoder is achienved.

For better and unsupervised feature representation and extracting the important features in the data is done by Auto Encoders.

AUTO ENCODERS do the job very very well.