UE21CS343BB2

Topics in Deep Learning

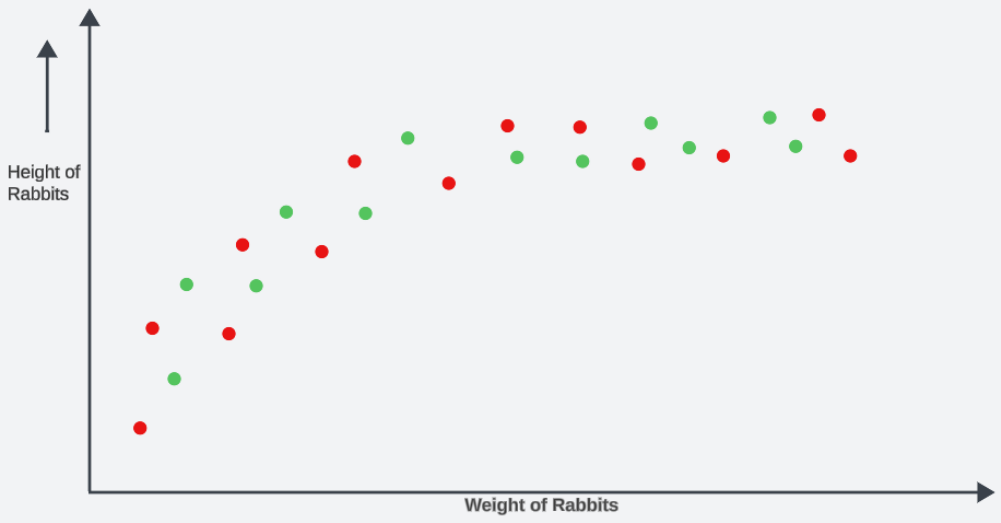
Dr. Shylaja S S
Director of Cloud Computing & Big Data (CCBD), Centre for Data Sciences & Applied Machine Learning (CDSAML)
Department of Computer Science and Engineering
shylaja.sharath@pes.edu

Ack: Anashua Dattidar, Teaching Assistant

A Recap of Bias Variance Tradeoff using an example

Say we have some data on heights and weights of rabbits and we wish to predict the height of an rabbit given its weight . After plotting the data we realise that the weight and height values have a linear relationship upto a point after which the rabbits don't get any taller but only fatter .

. Thus we go through the steps of our usual machine learning pipeline :

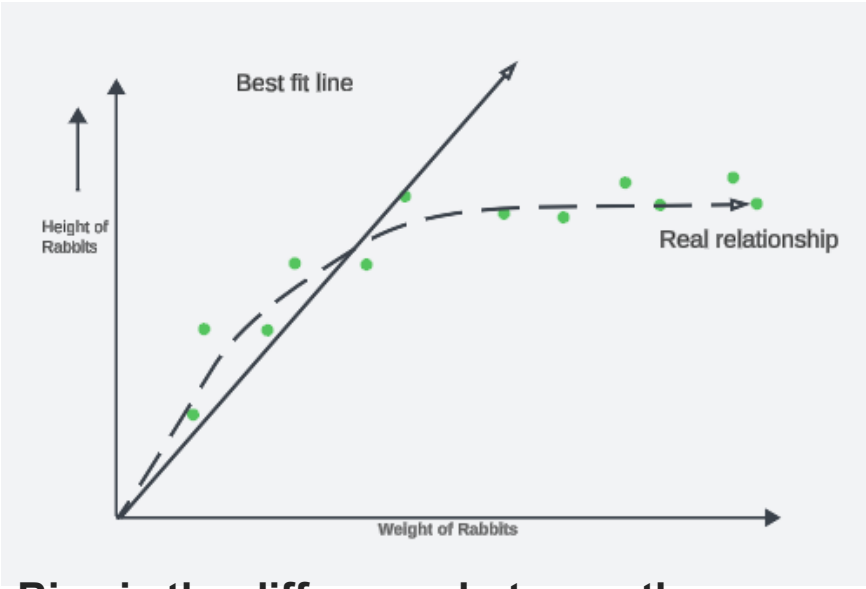


1. Data Preprocessing

1. Split into training and test set (Assume here the green points are training data and the red points are testing data)

A Recap of Bias Variance Tradeoff using an example

Now we would like to use some machine learning methods to approximate this relationship. So let's use Linear Regression and fit a straight line to the data



A clear problem with this approach is that the straight line will never be able to capture the true relationship between weight and height

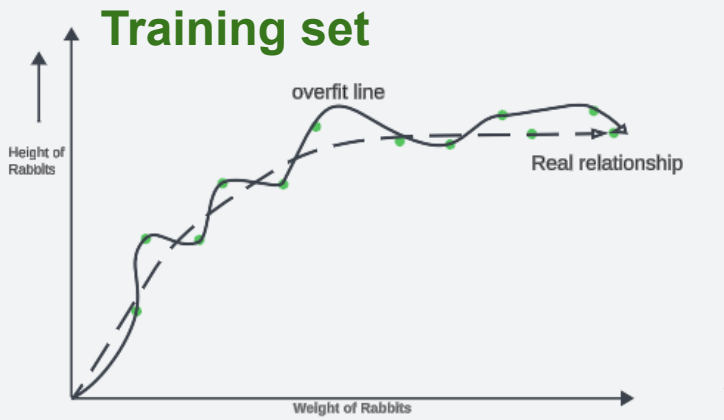
Thus the issue here lies in the assumption that the relationship can be modeled using a straight line and we are **highly "biased"** in our model selection and we have chosen a model which is too simple

Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model.

A Recap of Bias Variance Tradeoff using an example

Now in search of a more complex model for our data we have landed on the following function . This line passes through each and every data point on the training set (low bias) and does a great job of fitting the training set but it does an equally terrible job of fitting the testing set . In this case we have landed on a model which is too complex for the given data and thus has **high variance.**

Such a model is also said to be **overfit** as it perfectly fits the training data but fails to generalize.

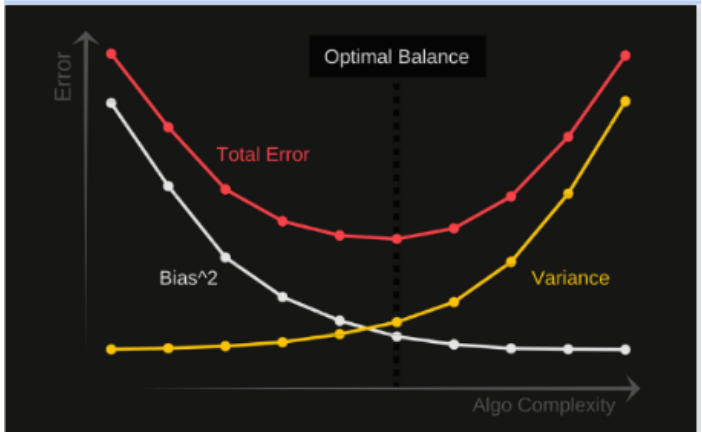


Variance is the variability of model prediction for a given data point or a value which tells us spread of our data.

A Recap of Bias Variance Tradeoff using an example

If our model is too simple and has very few parameters then it may have high bias and low variance. On the other hand if our model has large number of parameters then it's going to have high variance and low bias. So we need to find the right balance without overfitting and underfitting the data.

This is known as the bias variance tradeoff in machine learning as a model cannot be too complex or too simple but must be perfect for each problem.



To build a good model, we need to find a good balance between bias and variance such that it minimizes the total error.

Total Error = Bias^2 + Variance + Irreducible Error

Say , we have encountered an overfit model such as in the previous example . How do we improve the model ?

One way to get a better model is to use some form of **regularization** to get a more generalised model.

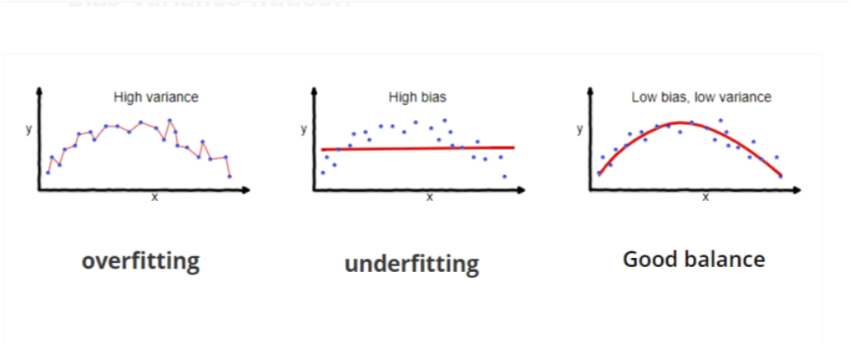
You could alternatively also use bagging or boosting.

What is Regularization?

Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.

Machine learning faces a key challenge: ensuring algorithms perform well not just on training data but on new inputs. To address this, various strategies, collectively known as regularization, aim to reduce test error, sometimes leading to higher training error. This field constantly explores newer, more effective regularization methods.
In this course, we'll delve into several such strategies which are:

1. L1 regularization
2. L2 regularization
3. Dropout
4. Early Stoppage



Parameter Norm Penalties

Regularization techniques have been much prevalent before neural networks came into the picture . The L1 and L2 family of regularization techniques are based on adding a parameter norm penalty (let's call it $\Omega(\theta)$; where θ represents the parameters) . To the objective function J (also known as cost function) .
Thus J' the new cost function becomes:

$$J'(\theta, X, y) = J(\theta, X, y) + \lambda^* \Omega(\theta)$$

λ [0, ∞) is a hyperparameter that weights the relative contribution of the norm penalty term

When our training algorithm minimizes the regularized objective function J' . It will decrease both the original objective J on the training data and some measure of the size of the parameters θ(or some subset of the parameters). Different choices for the parameter norm Ω can result in different solutions being preferred.

L^2 Parameter regularization

The L2 parameter norm penalty is commonly known as weight decay . This regularization strategy drives the weights closer to the origin by adding a regularization term to the objective function such as :

$$\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2 \qquad \|\mathbf{w}\|_2 = \left(|w_1|^2 + |w_2|^2 + \ldots + |w_N|^2\right)^{\frac{1}{2}}$$

2-norm (also known as L2 norm or Euclidean norm)

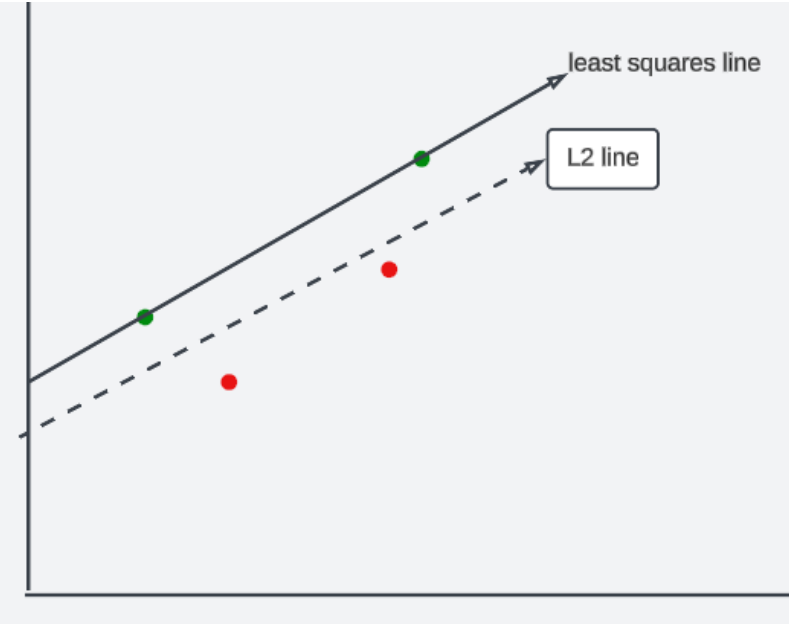
L2 regularization is also known as ridge regression (when used in linear regression) or Tikhonov regularization.

Let us understand L2 regularization using an linear regression example:

Consider a scenario in which we have 2 points in our training dataset and 2 points in our testing dataset , thus using linear regression we get a the best fit line which has a $SSE_{training}$ (Sum of square error) =0 but $SSE_{testing}$ = large !!

Ridge regression example

The main idea behind techniques like ridge regression is to find a new line with a little more bias such that there is a significant drop in variance. So with a slightly worse fit we get a more generalized line.



So as we have a straight line our equation here is :

$$y = m*x + c$$

were we estimate m and c by minimising the Sum of Square error or (SSE).

If we use ridge regression instead we shall be minimising

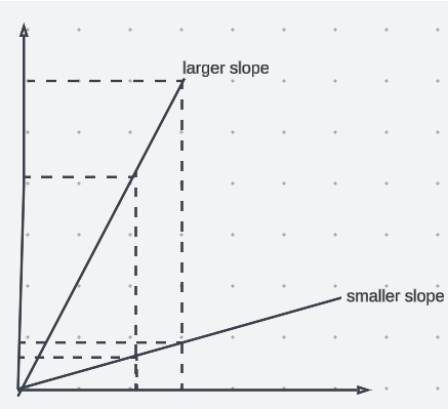
$$SSE + \lambda * slope^2$$

Adds penalty to the traditional least squares line

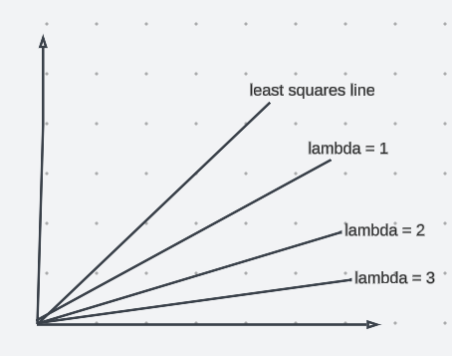
What is λ ?

λ [0, ∞) is a hyperparameter that weights the relative contribution of the norm penalty term . So when **λ** =0 we get the same line as the least squares line.

We know that a line with higher slope (steeper) y is more sensitive to small changes in x. While on a line with smaller slope y value changes slowly compared to large changes in x and is less sensitive to the value of x.



- As we increase the successive values of **λ** we get a less steeper line thus the predicted variable gets successively less sensitive to the parameter weights.
- Thus ridge regression can reduce the value of the slope to be asymptotically close to zero.
- One can use techniques like cross validation to get the best value for **λ**



L1 parameter regularisation and lasso regression

Although , L2 form of weight decay is most commonly used another option which may be used is L1 regularization . L1 regularization on the model parameter w is defined as :

$$\Omega(\boldsymbol{\theta}) = ||\boldsymbol{w}||_1 \qquad ||\mathbf{w}||_1 = |w_1| + |w_2| + ... + |w_N|$$

1-norm (also known as L1 norm)

The L1 regularization model with respect to linear regression is also known as lasso regression. As you may recall in ridge regression the cost function used was :

$$SSE + \boldsymbol{\lambda} * slope^2$$

Here we use ,

$$SSE + \boldsymbol{\lambda} * |\ slope|$$

Difference between L1 and L2

- The main difference between ridge and lasso regression is while ridge regression can shrink the slope asymptotically close to zero lasso regression can actually reduce the slope to zero .

This is useful while eliminating parameters which have no effect on the prediction.
For example :

size= a* weight + b* height + c* no_of_friends +d* horoscope + so on…4

- In the above equation although L2 regularization can reduce the value of c and d it will never be equal to zero.
- But using lasso regression c and d can be made zero so that the effects of such variables is completely removed.
- Thus Lasso regression is useful in cases where there are useless variables while ridge regression is a better option when most variables are useful.

L2 regularization for Neural networks

For a NN with L layers and n training examples the new cost function J would be denoted as:

$$J(W^1, b^1, W^L, b^L) = \sum_{i=1}^{n} L(Y'^{(i)}, Y^{(i)}) + \lambda^* \left(\sum_{l=1}^{L} \| W^{[l]} \|^2 \right)$$

$$\left(\sum_{l=1}^{L} \| W^{[l]} \|^2 \right) \Rightarrow \text{Frobenius norm}$$

$$\left(\sum_{l=1}^{L} \| W^{[l]} \|^2 \right) = \sum_{i=1}^{n[l-1]n[l]} \sum_{j=1} (W_{ij}^{l})^2$$

Sum of losses over n training examples

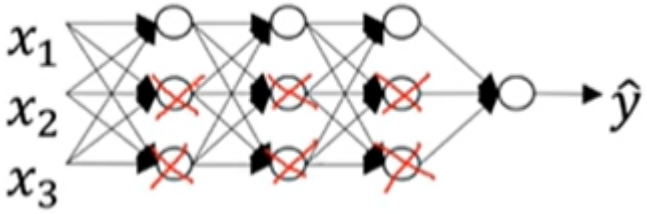
Intuition on how regularization reduce overfitting in NN

$$J(W^1, b^1, W^L, b^L) = \sum_{i=1}^{n} L(Y'^{(i)}, Y^{(i)}) + \lambda^* \left(\sum_{l=1}^{L} || W^{[l]} ||^2 \right)$$

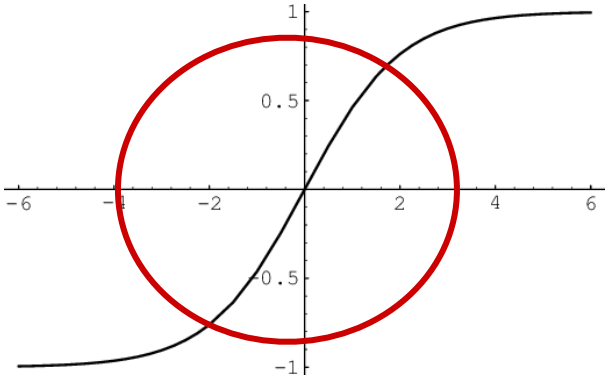
The Frobenius norm penalizes the weight matrices for being too large

- Here if the **λ** value is large then the W matrix values are close to 0 . This reduces the impact of a lot of hidden units.

- The the remaining NN is a much smaller And sparse network greatly simplifying the complexity of the current network And thus reducing overfitting.



Intuition on how regularization reduce overfitting in NN



Say, we have Tanh as our activation function

$$g(z) = \text{Tanh}(z)$$

For small values of z we almost have a linear curve, thus for values in which λ is large the weights W^{L} are small . So as

$$Z^{[L]} = W^{[L]} * a^{[L-1]} + b^{[L]}$$

Thus is the weights are small then Z is also relatively small.

As g(z) ends up being roughly linear , every layer in the NN ends up being roughly linear and so the model is made much simpler.

Why Dropout ?

Why Regularization using Dropout?

- One approach to reduce overfitting is to **fit all possible different neural network architectures on the same dataset and to average the predictions from each model.**

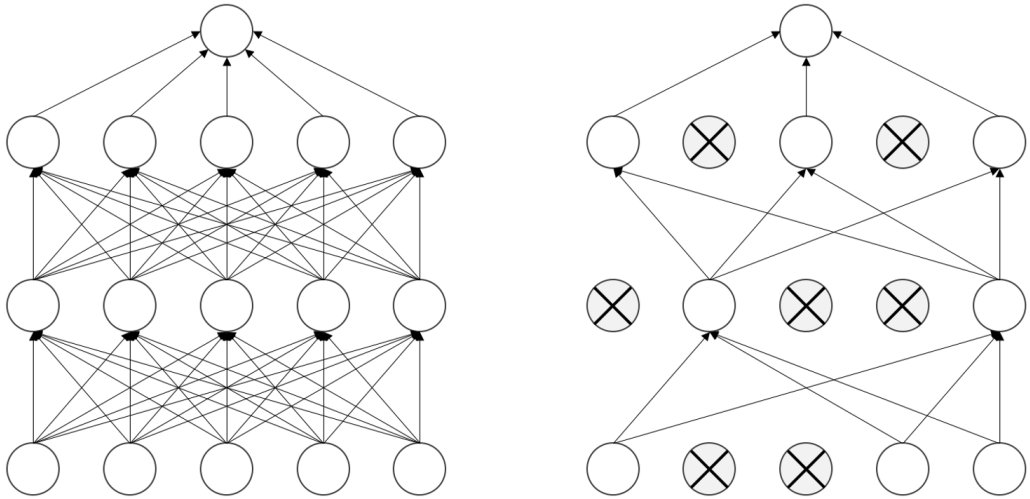
  This is not feasible in practice, and can be approximated using a small collection of different models, called an ensemble. However even with the ensemble approximation, multiple models need to be fit and stored, which can be a challenge if the models are large, requiring days or weeks to train and tune.

- Another approach could be **training multiple instances of the same neural network using different subsets of the training data**. Each instance learns from a different subset, which helps reduce overfitting. However, this approach also requires significant computational resources.

To overcome these expensive options, **dropout** emerges as a solution.

Dropout

The term "dropout" refers to dropping out the nodes in a neural network. All the forward and backwards connections with a dropped node are temporarily removed, thus creating a new network architecture out of the parent network.



Standard Neural Net

After Dropout

Dropout

For instance, if the hidden layers have 1000 neurons (nodes) and a dropout is applied with drop probability = 0.5, then 500 neurons would be randomly dropped in every iteration (batch).

A neural network with n nodes, the total number of thinned networks that can be formed are 2^n

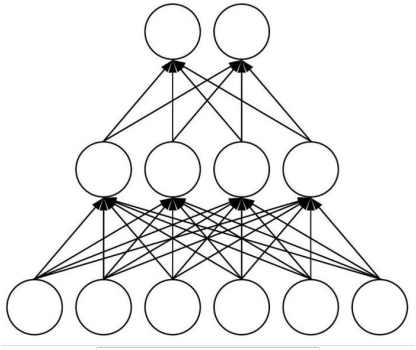
Of course, this is prohibitively large and we cannot possibly train 2^n networks. So we

- share the weights across all the networks
- sample a different network for each training instance

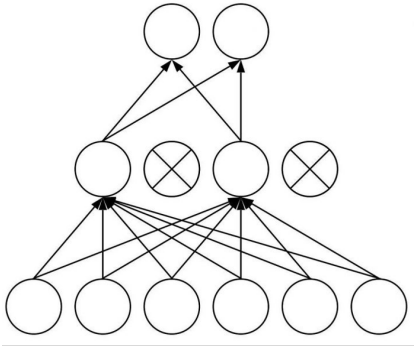
PES UNIVERSITY

More on Dropout

- While training, layers might co-adapt to correct mistakes from prior layers. This results in **complex co-adaptations which do not generalize unseen data resulting in overfitting**. Performing a dropout prevents the co-adaptations from occurring.
- A side-effect of applying dropout is that the activations of the hidden units become **sparse**, even when no sparsity inducing regularizers are present.
- Because the outputs of a layer under dropout are randomly subsampled, it has the effect of reducing the capacity or thinning the network during training. As such, a wider network, e.g. more nodes, may be required when using dropout.



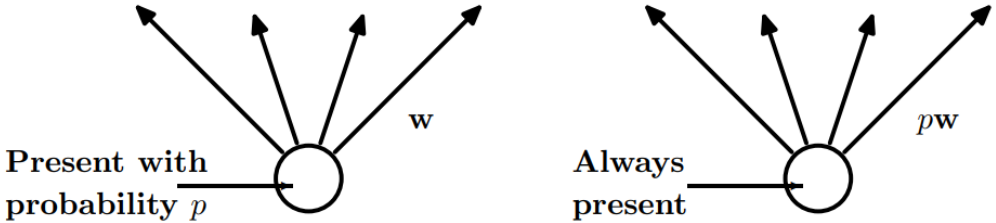
Standard Neural Net



After Dropout

Dropout Implementation

- Dropout is typically applied to hidden and input layers, and not on nodes of the output layer.
- A hyperparameter defines the **probability at which outputs of the layer are dropped out**, or inversely, the probability at which outputs of the layer are retained.
- Dropout is **not used after training** when making predictions. The **weights of the network will be larger than normal because of dropout**. Therefore, before finalizing the network, the weights are **first scaled by the chosen dropout rate**.
- **The rescaling of the weights can also be performed at training time instead, after each weight update at the end of the mini-batch.** This is called "**inverse dropout**" and does not require any modification of weights during training. (Both Keras & PyTorch implement dropout in this way)

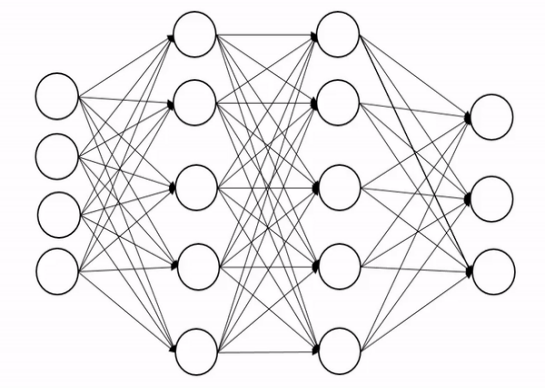


(a) At training time (b) At test time

Dropout Training

Steps:

- for the first training instance (or mini-batch), the weights are initialized and dropout is applied
- loss is computed and backpropagation occurs (only the active parameters get updated)
- dropout is applied again for the second training instance, resulting in in a different thinned network
- we again compute the loss and backpropagate to the active weights
- if the weight was active for both the training instances then it receives a total of two updates and if the weight was active for only one of the training instances then it receives only one update
- each thinned network gets trained rarely (or even never) but the parameter sharing ensures that no model has untrained parameters

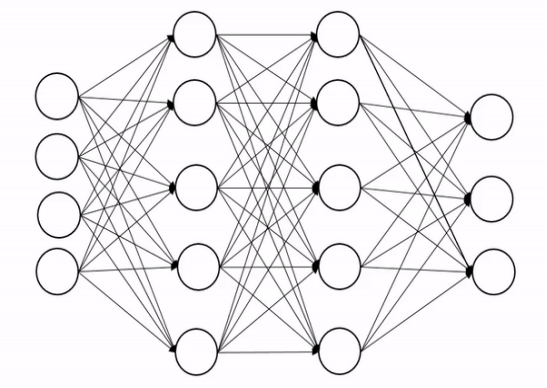


Tips for Using Dropout Regularization

- dropout is a generic approach and can be used with all network types (MLPs, CNNs, LSTMs etc.)
- a good value for dropout in a hidden layer is between 0.5 and 0.8 and for units in the input layers a larger dropout rate, such as of 0.8.
- usually applied in larger networks as they overfit easily
- testing of different rates is important as it results in the best dropout rate for the problem; along with this it also indicates how sensitive the network is to dropout and an unstable network can benefit from an increase in size
- dropout is more effective where there is a limited amount of training data and the model is likely to overfit the training data

Summary

- in dropout, units in the neural network are dropped
- each iteration of the training process has a different set of nodes, resulting in different outputs each time; resembles ensemble training
- the probability with which a unit is dropped is defined by the hyperparameter of the dropout function
- a common value for probability is 0.5 for retaining the output of each node in a hidden layer and a value close to 1.0, such as 0.8, for retaining inputs from the visible layer.
- the weights of the network are rescaled when dropout is applied
- dropout is used to prevent overfitting and is a popular method of regularization



Data Augmentation

The word “**augmentation**” literally means “the action or process of making or becoming greater in size or amount”, summarizes the outcome of this technique.

But another important effect is that it increases or augments the diversity of the data. The increased diversity means, at each training stage the model comes across a different version of the original data.

For images, some common methods of data augmentation are taking cropped portions, zooming in/ out, rotating along the axis, vertical/horizontal flips, adjusting the brightness and sheer intensity. Data augmentation for audio data involves adding noise, changing speed and pitch.

While data augmentation prevents the model from overfitting, **some augmentation combinations can actually lead to underfitting.**

Data Augmentation

The simplest way to prevent overfitting is to increase the size of the training data but collecting more labelled data is costly. In case of images, as mentioned earlier there are a few ways such as rotating the image, flipping, scaling, shifting, etc.

In the above image, some transformations have been applied on the MNIST dataset. This provides a big leap in improving the accuracy of the model. **It can be considered as a mandatory trick in order to improve our predictions.**

In Keras, we can perform all of these transformations using ImageDataGenerator. It has a big list of arguments which you you can use to pre-process your training data.

Early Stopping

Too little training results in an underfit model and too much training results in an overfit model

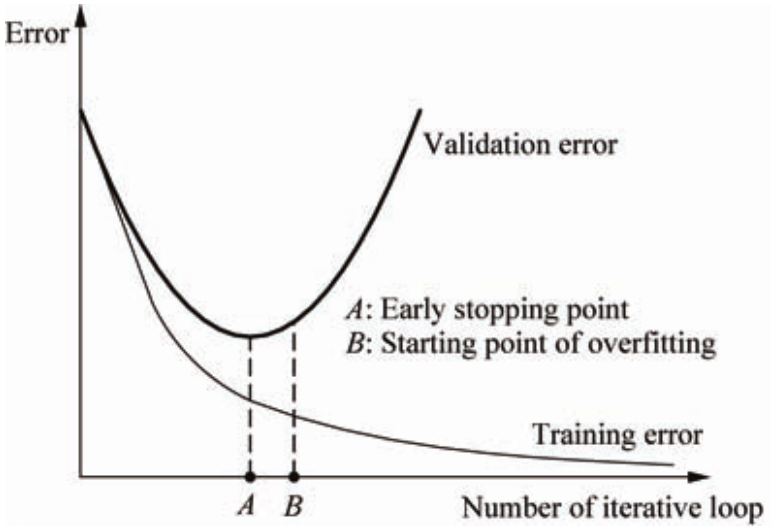
If the performance of the model on the validation dataset starts to **degrade** (e.g. loss begins to increase or accuracy begins to decrease), then the training process is **stopped**. The model at the time that training is stopped is then used and is known to have good generalization performance.

This simple, effective, and widely used approach to training neural networks is called early stopping.

When training the network, a larger number of training epochs is used than may normally be required, to give the network plenty of opportunity to fit, then begin to overfit the training dataset.

Early Stopping

- To prevent overfitting and avoid driving the validation error to a high value, it is important to track the validation error during training.
- A patience parameter (p) is used to determine the number of steps to wait for improvement in the validation error.
- At each step (k), the validation error is checked. If there is no improvement in the validation error for the previous p steps, training is stopped, and the model stored at step (k - p) is returned.
- By stopping the training early, we prevent the model from overfitting and avoid the situation where the training error approaches 0 while the validation error increases.



References

- https://towardsdatascience.com/understanding-the-bias-variance-tradeoff-165e6942b229
- https://www.youtube.com/watch?v=SjQyLhQIXSM&list=PLkDaE6sCZn6Hn0vK8co82zjQtt3T2Nkqc&index=2
- https://www.youtube.com/watch?v=EuBBz3bI-aA
- https://www.youtube.com/watch?v=Q81RR3yKn30&t=891s
- https://www.youtube.com/watch?v=D8PJAL-MZv8&list=PLkDaE6sCZn6Hn0vK8co82zjQtt3T2Nkqc&index=6
- https://towardsdatascience.com/intuitions-on-l1-and-l2-regularisation-235f2db4c261

UE21CS343BB2

Topics in Deep Learning

Dr. Shylaja S S
Director of Cloud Computing & Big Data (CCBD), Centre for Data Sciences & Applied Machine Learning (CDSAML)
Department of Computer Science and Engineering
shylaja.sharath@pes.edu

Ack: Anashua Dattidar, Teaching Assistant