# 

[**10.1: Batch Normalization**](#_lycljv128knv) **2**

[10.1.1: Normalizing inputs](#_bhidkcjzbi) 2

[10.1.2: Why should we normalize inputs](#_i6g4kfk1jerh) 3

[10.1.3: Batch Normalization](#_emtde23tti1) 4

[10.1.4: Learning Mu and Sigma](#_ofj12iwsjo8y) 6

[**10.2: Dropout**](#_a0svl7dmwsqs) **7**

[10.2.1: Ensemble Methods](#_4x6769m09b0p) 7

[10.2.2: The idea of dropout](#_bwm9bn15s85t) 8

[10.2.3: Training without dropout](#_wxlctoa8zy63) 9

[10.2.4: How does weight sharing help?](#_6xtkrbjdy2em) 10

[10.2.5: Using dropout at test time](#_9vujttgz1iah) 11

[10.2.6: How does dropout act as a regularizer?](#_gd7ddp4x1oom) 12

[10.2.7: Summary and what next?](#_ay34v41hk89r) 13

# 

# Batch Normalization

## Normalizing inputs

How do you normalize the data?

1. First, let’s have another brief look at the Deep Learning timeline
   1. In this chapter, we will be looking at Batch Normalization and Dropout, which in some sense come under the topic of **“Better Regularization”**
2. Before we talk about Batch Normalization, let us first recap on how we normalize/standardize inputs. **The terms normalization & standardization are used interchangeably**
   1. We normalize the data using the following formula:
      1. for each feature of each training sample.
      2. Where mean:
      3. And Variance/Standard deviation:

## Why should we normalize inputs

Why does normalizing inputs help?

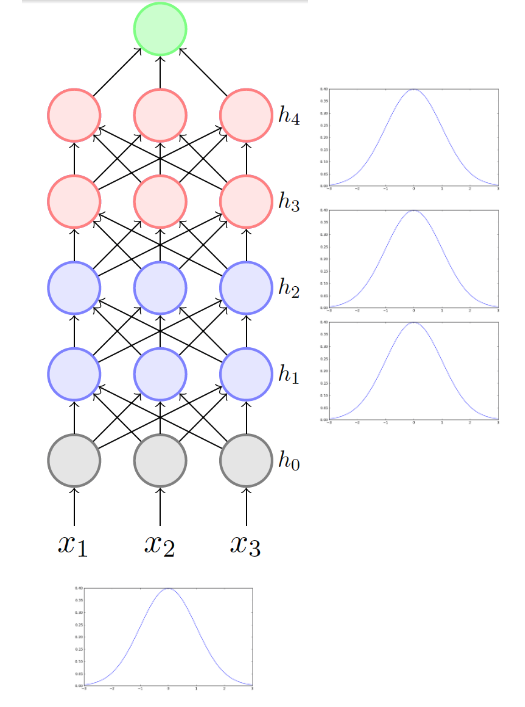
1. Let’s compare the before-after of normalization of inputs

|  |  |
| --- | --- |
| **Before Normalization** | **After Normalization** |
|  |  |
| Here, we observe that x1 has a much larger range of values than x2 | Here, x1 and x2 have both been brought into a comparable range of values. |
| When optimising the Loss function, one set of weights corresponding to the smaller feature x2 will end up being very large.  This leads to oscillations about the local minima during gradient descent as much larger updates are made due to large weights. | Here, since weights end up being smaller, we see a good reduction in the oscillation about the local minima during optimization. |
| Another point to note is that in Gradient Descent, our **updates would be very biased** to the feature corresponding to the larger weights.  Thus we may see the model becoming very sensitive to that particular feature | Here, the weights are brought in a comparable range.  This reduces the chance of updates biased to any particular feature |

## 

## Batch Normalization

We have standardized inputs but what about hidden representations?

1. When we normalize/standardize the inputs, most of the inputs lie around the mean value, in the standard deviation range for all inputs xmn where (m: no. of training samples, n: no. of features)
2. Now, let’s try applying normalizations to any particular hidden representations. Consider one particular Hidden representation H
3. Now, standardization is done on the H matrix so that all of the columns are in the same range.
4. In a sense, the H matrix acts as an input to the next layer.
5. Thus, to learn the weights effectively, Standardization is done on all the columns to bring them to a comparable range.
6. Just as we standardize the inputs, we standardize the activations at all layers
7. Now, why is it called batch normalization?
8. Let’s take a look at the formulae for Batch normalizing H
   1. for each feature of each training sample.
   2. Where mean:
   3. And Variance/Standard deviation:
9. Now, for the input normalization, we consider the entire training size (m) when computing mean and std. However, here we only consider a smaller subset of samples (k). **Mean (μ) and Std (σ) are calculated using a batch of k samples, hence it is called batch normalization.** K = 32 is commonly used.

## 

## Learning Mu and Sigma

But by normalising the activations are we enforcing some constraints?

1. By forcing the activation function outputs to lie within a particular range, are we imposing some constraints on the model?
2. Batch normalization has a solution for that, in the form of the 𝛄 and β terms.
3. Consider the matrix H:



* 1. (1)
  2. (2)
  3. **𝛄 and β are learned parameters** for each column of H. They are learned just like the weights and biases, using an update rule like SGD, Adam, NAG etc.
  4. Introduction of learned parameters **𝛄 and β** ensures we are not locked to a **μ = 0 and σ = 1**
     1. Let’s see what this means
     2. Suppose the network learns the values **𝛄j = σj** and **βj = μj**
     3. So, equation (2) can be rewritten:
     4. Rearranging equation (1):
     5. From the above two equations, we can see that if there is a particular set of values of **σ** and **μ** that cause the loss to decrease, then the network will learn **𝛄j = σj** and **βj = μj** so that
     6. This allows us to the network to opt out of normalization and use the original value in cases where normalization does not decrease the loss.

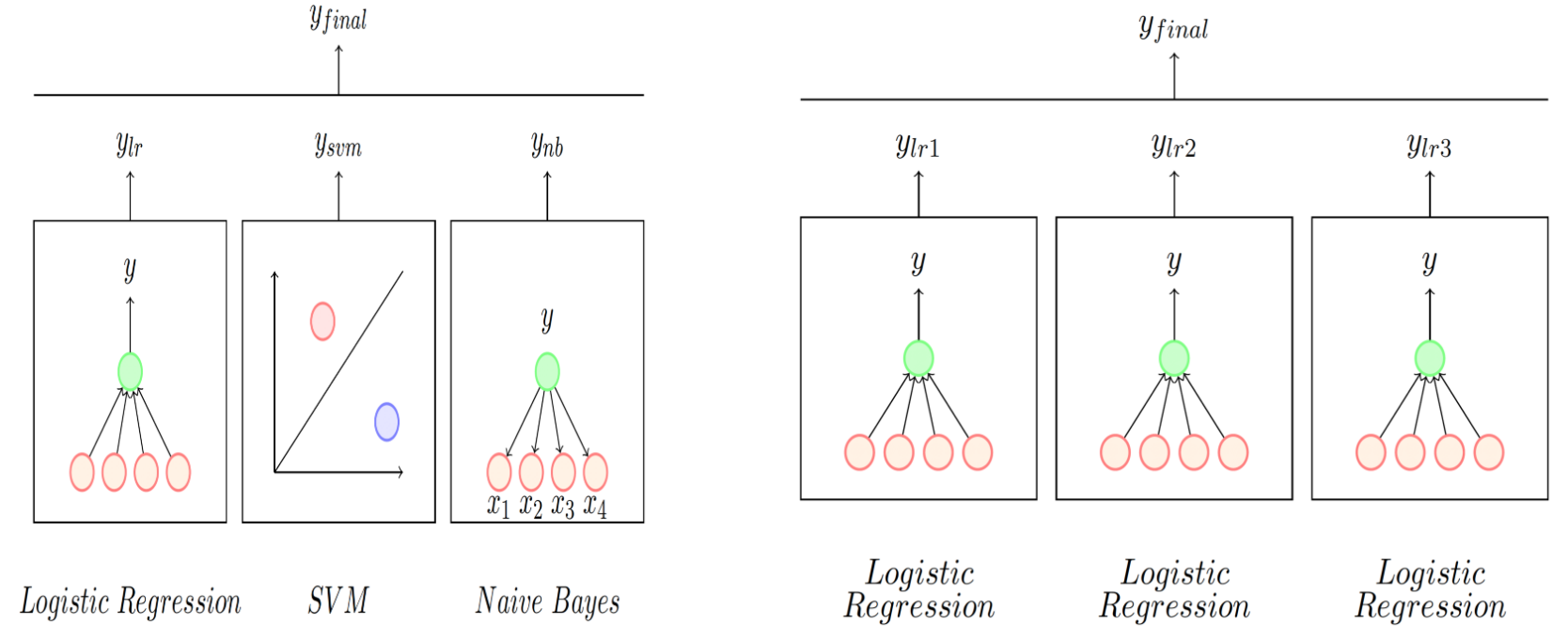
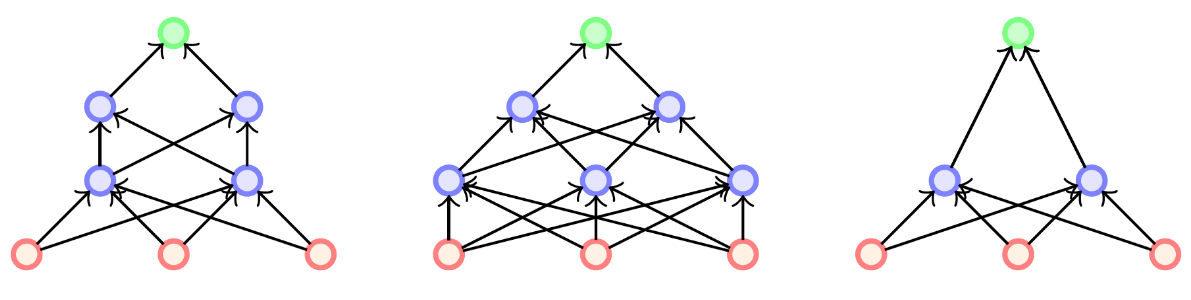
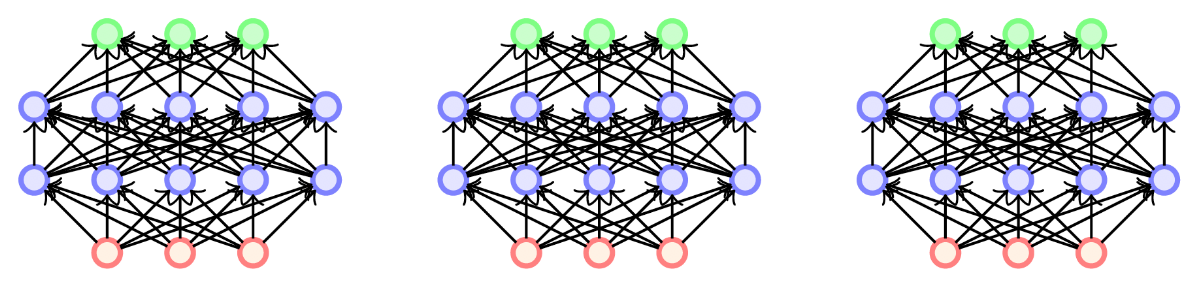
1. Another use of Batch Normalization is that it acts as a form of regularization
   1. Here, **μ and σ** are computed from a mini-batch of size k, thus they are very likely to be noisy (as they are not calculated using the entire dataset).
   2. Introducing noise to the data leads to better regularization.
   3. Hence, the network is less likely to overfit the training data, thereby making the network more robust.

## 

# Dropout

## Ensemble Methods

Does it make sense to rely on multiple models instead of a single one?

1. Ensemble methods are techniques that create multiple models and combine their outputs to produce improved results.
2. The outputs of the models are combined in various ways, such as averaging them, taking the weighted average etc.
   1. The models used can be dissimilar: Combining a Logistic Regression model with an SVM and a Naive Bayes Classifier
   2. The modes can also be similar: Combining the output of 3 Logistic Regression classifiers each trained differently (different subsets of data/features or using different hyperparameters).
3. Now, let us look at how to apply ensemble methods to Neural Networks
   1. **Method 1**: Train different architectures(models) on the same data. **Computationally Expensive**
   2. Method 2: Train the same architecture on different training data subsets. **Computationally Expensive**
4. How do we create an ensemble method for Neural Networks that does not have a significantly large training time

## The idea of dropout

Can we create multiple neural networks from a single neural network?

1. Now, let us explore the idea of creating multiple neural networks out of a single neural network.
2. Consider a network with n-nodes. We can create sub-networks which operate using a subset of these n-nodes like shown in the figure.

|  |  |
| --- | --- |
| **Original Network** | **Network with some nodes dropped out** |
|  | |

1. The excluded neurons are said to be dropout neurons and the network is called a dropped-out network
2. If we have n-neurons/nodes in a network, it is possible to create up to 2N dropped-out networks from it. These networks are much less dense than the original network and can be used in an ensemble method without drastically increasing the training time.
3. Now, even for a relatively modest n value of say 100, we have a very large number of dropped out network. The question arises, **how do we train these networks?**
4. We can use the following tricks to train the dropped-out networks:
   1. Share parameters across all these neural networks
   2. Sample a different neural network for each training instance

## 

## Training without dropout

What does the training procedure look like?

1. Let’s look at the training procedure for dropped-out networks
   1. **Initialise**:
      1. Initialise the parameters
   2. **Iterate over data**:
      1. = Current\_mini\_batch
      2. NNi = dropout Network for that mini-batch (made using coin-toss method for each neuron n)
      3. The same/relevant weights are shared between the mega-network and the dropped out network.
      4. Backpropagate(NNi)
      5. Only connected weights are updated.
   3. **End**:
      1. Repeat till satisfied

## 

## How does weight sharing help?

How are weights shared between subsequent dropped-out networks

1. Let’s take another look at the dropout training procedure
2. From the above, we can see that there are some common weights between the two dropped out networks.
   1. Let be the highlighted weight in the Mega Network
   2. In the first iteration, for , the weight gets updated
   3. In the second iteration, for , the weight gets updated
   4. We can see how the weights are not freshly calculated for each iteration, and instead all **common weights are shared between the dropped-out networks**.
3. Consider a dataset of 1-million training samples. With a possible total of 2N dropped out networks, even for a small value of n=100, it is impossible to cover all possible networks even through multiple epochs over our dataset. 2N >> 1-million. (recall that each-mini batch corresponds to 1 dropped-out network)
4. So, even with a small mini-batch size, most individual dropped-out networks will get trained only a small number of times. How do we ensure the efficient training of the entire set of dropped-out networks?
5. Now, instead of looking at it in terms of dropped-out networks getting trained, let’s look at in terms of how many times each weight gets trained.
6. In short, each weight gets updated a suitable number of times. Here is the intuition behind it:
   1. Each neuron will be present in half of all possible networks (Use truth table to verify)
   2. The weights are shared across all networks
   3. Hence each weight will get updated frequently during training.
7. Thus, we **do not need to train the network (2N x k)** times, it is **sufficient if it is trained k times**, where k is the number of training steps/iterations whereby a set of weights are updated.
8. Now, the next step is figuring out how to combine the outputs of all the networks in the ensemble.

## Using dropout at test time

How do you use the networks at test time?

1. At the end of training, we are left with the outputs of the various networks, that we combine with an ensemble method such as average, voting etc.
2. In theory, combining 2N such outputs would prove to be very computationally expensive.
3. There is a simple strategy to overcome this.
4. Consider any particular neuron from a dropped-network during training.
5. Now, during test time, instead of using the ensemble, we use the large mega-network.
6. Instead of each neuron being present with probability p, we assume that all neurons are present at all times.
7. However, we multiply the output of the neurons with the probability value p.
8. We scale the output of the neuron by p to show that it is only p% reliable.
9. So in our example of p = 0.5, it’s like saying that a particular neuron is only present in 50% of the networks, thus we approximate it by multiplying its outputs by 0.5

## 

## How does dropout act as a regularizer?

Why does dropout help?

1. Let us first talk about co-adaptation of neurons. It is the phenomenon where neurons adapt to each others’ performance and begin individual specialization.
2. The following example will illustrate how we regularize the network using dropout
3. From the above diagram, we can see how other neurons are forced to learn to compensate for **hi** being dropped from the network, thereby preventing co-adaptation.
4. Dropout acts as a regularizer by corrupting the input given to the subsequent layers. As certain neurons are dropped, this results in a less complete output being transferred to the next layer. This can also be viewed as adding noise to the input.
5. These processes add a degree of robustness to the network.

## 

## Summary and what next?

Let’s summarize what we’ve completed so far

1. Let’s have another brief look at our Deep Learning timeline
2. Here’s a brief overview of the topics covered so far
3. Feedforward Neural networks
   1. Universal approximation theorem (UAT)
   2. Backpropagation
   3. Learning Algorithms
   4. Activation functions
   5. Regularization methods
4. Convolutional Neural networks
   1. Convolutional operation
   2. CNN Architectures (AlexNet, VGGNet, ResNet etc)
   3. Batch Normalization:
      1. An interesting point to note is that batch normalization can either be performed pre-activation or post-activation (after non-linearity like ReLU is applied)
   4. Dropout:
      1. For hidden layers, we normally use probability p = 0.5
      2. For input layer, we use p = 0.8
      3. Using 50% probability in the input layer would cause too much noise to the network.
5. The next set of topics to cover are as follows:
   1. Recurrent Neural Networks (RNNs)
   2. Encoder-Decoder Models