# Dropout and Batch Normalization

Pournami P N

NITC

March 31, 2024

## Table of Contents

## Dropout

- Dropout is a regularization technique used in neural networks to prevent overfitting. [Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, *Dropout: a simple way to prevent neural networks from overfitting*, The journal of machine learning research, Volume 15, Issue 1, Pages 1929-1958].
- During training, a fraction of neurons are randomly dropped out with a probability $p$.
- This prevents the network from relying too heavily on any individual neuron.

## Dropout

- The term "dropout" refers to dropping out the nodes (input and hidden layer) 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.

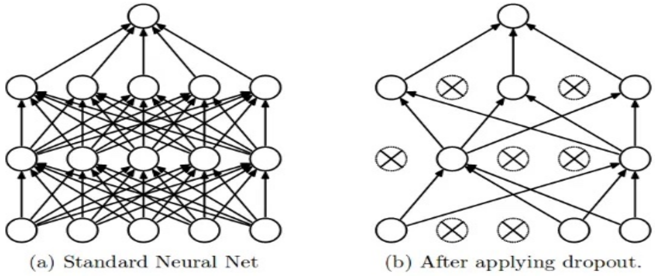

Figure 1: Dropout applied to a Standard Neural Network

## How does it solve the Overfitting problem?

- In the overfitting problem, the model learns the statistical noise. To be precise, the main motive of training is to decrease the loss function, given all the units (neurons). So in overfitting, a unit may change in a way that fixes up the mistakes of the other units. This leads to complex co-adaptations, which in turn leads to the overfitting problem because this complex co-adaptation fails to generalise on the unseen dataset.
- Now, if we use dropout, it prevents these units to fix up the mistake of other units, thus preventing co-adaptation, as in every iteration the presence of a unit is highly unreliable. So by randomly dropping a few units (nodes), it forces the layers to take more or less responsibility for the input by taking a probabilistic approach.
- This ensures that the model is getting generalised and hence reducing the overfitting problem.

## Mathematical explanation of the dropout

- In the original implementation of the dropout layer, during training, a neuron in a layer is selected with a keep probability (1-$p$).
- This creates a thinner architecture in the given training batch, and every time this architecture is different.

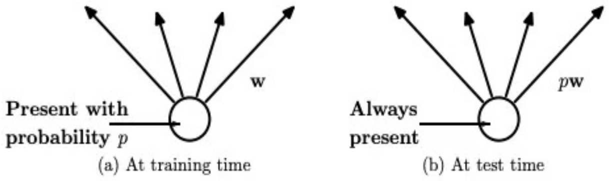

Figure 2: (a) A unit (neuron) during training is present with a probability $p$ and is connected to the next layer with weights '$w$' ; (b) A unit during inference/prediction is always present and is connected to the next layer with weights, '$pw$'

## Mathematical explanation of the dropout

- In the standard neural network, during the forward propagation we have the following equations:

$$\begin{aligned} z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)}\mathbf{y}^l + b_i^{(l+1)}, \\ y_i^{(l+1)} &= f(z_i^{(l+1)}), \end{aligned}$$

- $z$: denote the vector of output from layer $(l+1)$ before activation
  $y$: denote the vector of outputs from layer $l$
  $w$: weight of the layer $l$
  $b$: bias of the layer $l$
- $z$ is transformed into the output for layer $(l+1)$ with the activation function.

## Mathematical explanation of the dropout

- Now, if we have a dropout, the forward propagation equations change in the following way:

$$\begin{aligned} r_j^{(l)} &\sim \text{Bernoulli}(p), \\ \widetilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\ z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \widetilde{\mathbf{y}}^{l} + b_i^{(l+1)}, \\ y_i^{(l+1)} &= f(z_i^{(l+1)}). \end{aligned}$$

- So before we calculate $z$, the input to the layer is sampled and multiplied element-wise with the independent Bernoulli variables.
- $r$ denotes the Bernoulli random variables each of which has a probability p of being 1.
- $r$ acts as a mask to the input variable, which ensures only a few units are kept according to the keep probability of a dropout.
- This ensures that we have thinned outputs "$y(bar)$", which is given as an input to the layer during feed-forward propagation.

## Mathematical explanation of the dropout

During inference:

- No dropout is applied, but the output is scaled by the dropout probability $p$.
- if a unit is retained with probability $p$ during training, the outgoing weights of that unit are multiplied by $p$ during the prediction stage.
- This means that all the units are considered during the prediction step. But, because of taking all the units/neurons from a layer, the final weights will be larger than expected
- Hence, weights are first scaled by the chosen dropout rate. With this, the network would be able to make accurate predictions.

## Batch Normalization

- Batch Normalization is a technique used to stabilize and accelerate training in neural networks.
- It normalizes the activations of each layer across mini-batches during training.
- This helps in reducing internal covariate shift and allows for higher learning rates.

## Input Data Normalization

- When inputting data to a deep learning model, it is a standard practice to normalize the data to zero mean and unit variance. What does this mean and why do we do this?
- Suppose there are $n$ features $x_1, x_2, \ldots x_n$ as input to a neural network. Each feature might have a different range of values.
- For instance, values for feature $x_1$ might range from 1 through 5, while values for feature $x_2$ might range from 1000 to 99999.
- So, for each feature column separately, we take the values of all samples in the dataset and compute the mean and the variance. And then normalize the values using the formula below.

$$\hat{X}_i = \frac{X_i - Mean_i}{StdDev_i}$$

## Implementation of Batch Normalization

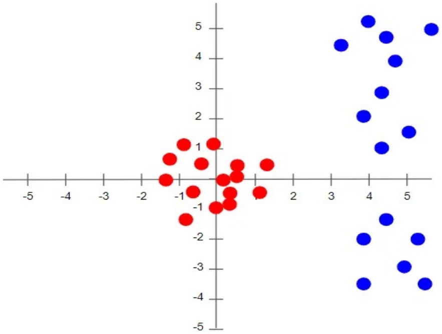

Figure 3: Before normalization (Blue dots), After Normalization (Red dots), the feature values are centered around zero.

# Implementation of Batch Normalization

- The same logic that requires us to normalize the input for the first layer will also apply to each of these hidden layers.
- if we are able to somehow normalize the activations from each previous layer then the gradient descent will converge better during training. This process is called Batch Normalisation.
- Batch Norm is just another network layer that gets inserted between a hidden layer and the next hidden layer.

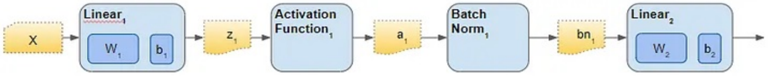

Figure 4: Parameters of a Batch Norm layer

- Its job is to take the outputs from the first hidden layer and normalize them before passing them on as the input of the next hidden layer.

## Implementation of Batch Normalization

- Just like the parameters (eg. weights, bias) of any network layer, a Batch Norm layer also has parameters of its own:
  - Two learnable parameters called $\beta$ and $\gamma$.
  - Two non-learnable parameters (Mean Moving Average and Variance Moving Average) are saved as part of the 'state' of the Batch Norm layer.

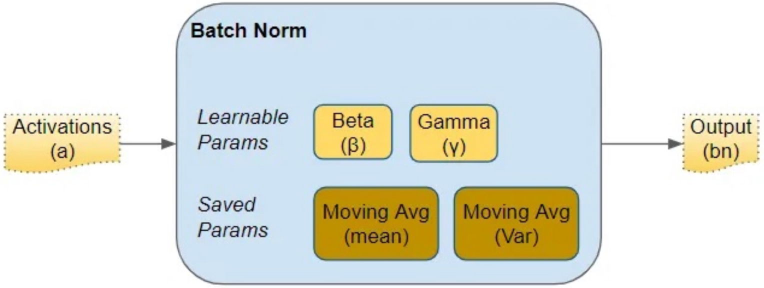

Figure 5: Parameters of a Batch Norm layer

## BatchNorm during Training I

- During training, one mini-batch of data is fed into the network at a time.
- During the forward pass, each layer of the network processes that mini-batch of data.
- The Batch Norm layer processes its data as follows:
  1. **Activations** - The activations from the previous layer are passed as input to the Batch Norm. There is one activation vector for each feature in the data.
  2. **Calculate Mean and Variance** - For each activation vector separately, calculate the mean and variance of all the values in the mini-batch.
  3. **Normalize** - Calculate the normalized values for each activation feature vector using the corresponding mean and variance. These normalized values now have zero mean and unit variance.

## BatchNorm during Training II

4. **Scale and Shift** - This step is the huge innovation introduced by Batch Norm that gives it its power. Unlike the input layer, which requires all normalized values to have zero mean and unit variance, Batch Norm allows its values to be shifted (to a different mean) and scaled (to a different variance). It does this by multiplying the normalized values by a factor, $\gamma$, and adding to it a factor, $\beta$ using an element-wise multiplication.
5. **Moving Average** - Batch Norm also keeps a running count of the Exponential Moving Average (EMA) of the mean and variance. During training, it simply calculates this EMA but does not do anything with it. At the end of training, it simply saves this value as part of the layer's state, for use during the Inference phase.

## BatchNorm during Training III

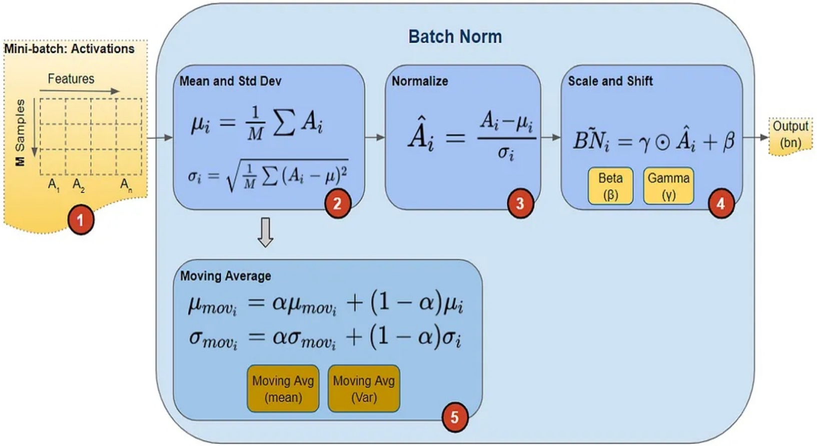
Mini-batch: Activations
Features
M Samples
$A_1$ $A_2$ $A_n$
1
Batch Norm
Mean and Std Dev
$\mu_i = \frac{1}{M}\sum A_i$
$\sigma_i = \sqrt{\frac{1}{M}\sum(A_i - \mu)^2}$
2
Normalize
$\hat{A}_i = \frac{A_i - \mu_i}{\sigma_i}$
3
Scale and Shift
$\tilde{BN}_i = \gamma \odot \hat{A}_i + \beta$
Beta (β)
Gamma (γ)
4
Output (bn)
Moving Average
$\mu_{mov_i} = \alpha\mu_{mov_i} + (1-\alpha)\mu_i$
$\sigma_{mov_i} = \alpha\sigma_{mov_i} + (1-\alpha)\sigma_i$
Moving Avg (mean)
Moving Avg (Var)
5

Figure 6: Calculations performed by Batch Norm layer

## Batch Norm during Inference

- During Inference, we have a single sample, not a mini-batch.
- Here is where the two Moving Average parameters come in — the ones that we calculated during training and saved with the model. Those saved mean and variance values are used for the Batch Norm during Inference.

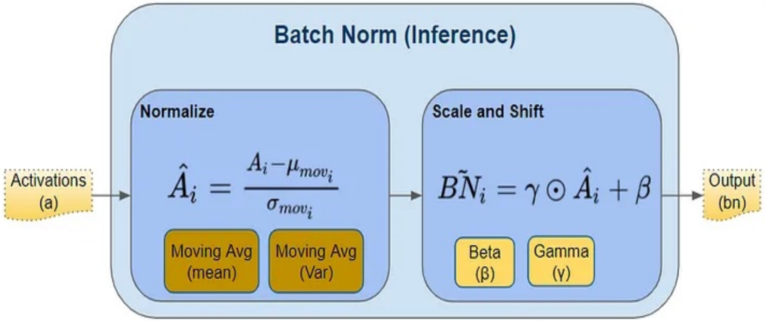

Figure 7: Calculations performed by Batch Norm layer during Inference

## Internal covariate shift

- **Internal Covariate Shift** is defined as the change in the distribution of network activations due to the change in network parameters during training.
- In neural networks, the output of the first layer feeds into the second layer, the output of the second layer feeds into the third, and so on. When the parameters of a layer change, so does the distribution of inputs to subsequent layers.
- These shifts in input distributions can be problematic for neural networks, especially deep neural networks that could have a large number of layers.
- Batch normalization is a method intended to mitigate internal covariate shift for neural networks.

# Internal covariate shift

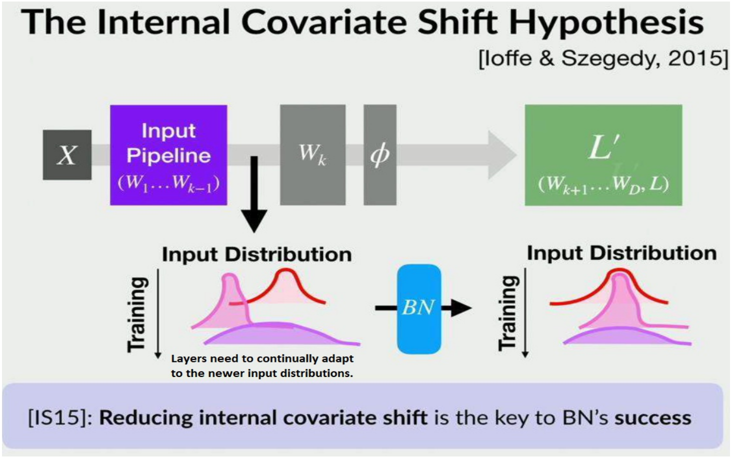
The Internal Covariate Shift Hypothesis
[Ioffe & Szegedy, 2015]
$X$
Input Pipeline
$(W_1 \dots W_{k-1})$
$W_k$
$\phi$
$L'$
$(W_{k+1} \dots W_D, L)$
Input Distribution
Training
Layers need to continually adapt to the newer input distributions.
$BN$
Input Distribution
Training
[IS15]: Reducing internal covariate shift is the key to BN's success

Figure 8: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

## Conclusion

- Dropout and Batch Normalization are powerful techniques for improving the performance and robustness of neural networks.
- Dropout prevents overfitting by regularizing the network and encouraging robustness.
- Batch Normalization stabilizes and accelerates training by reducing internal covariate shift.
- They are commonly used in practice to train deep neural networks effectively.

## References

1. https://towardsdatascience.com/dropout-in-neural-networks-47a162d621d9
2. https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739
3. Sergey Ioffe, Christian Szegedy, Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, https://doi.org/10.48550/arXiv.1502.03167
4. Andriy Burkov, The Hundred-Page Machine Learning Book, ISBN 1999579518, 9781999579517.