Improving Performance

Week 4

COMP6252 (Deep Learning Technologies)

ECS, University of Southampton

28 April 2022

Introduction

- As we have already seen DNN are powerful models.
- This is mainly due to their strong expressiveness:
  - the ability to model complicated relationships between input and output
- But this same expressiveness can lead to some problems

1. Overfitting
   - Due to their power, NN can "learn" noise present in the training but not in the test datasets
   - This reduces their generalization efficacy
2. Convergence
   - Training takes long time. Sometimes doesn't even converge
   - The results highly dependent on the meta parameters

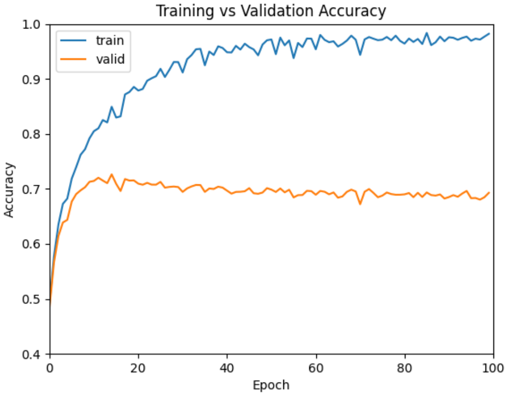
What can be done?

This week we will introduce three approaches to mitigate the problems of convergence and overfitting.

1. Early stopping
2. Dropout
3. Batch normalization

Overfitting

- An example of overfitting is shown below
- The training accuracy keeps increasing while the validation plateau and even decreases

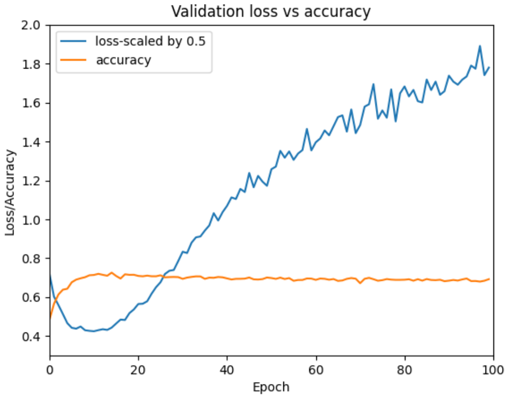


Early stopping

- In the previous figure, the max accuracy occurs at epoch 14
- If we stop training at epoch 14 or close to it not only we get better accuracy but we save lots of time
- This is exactly what is called **early stopping**
- How do we know when to stop training?
- One of the simplest methods is to monitor the **validation loss**

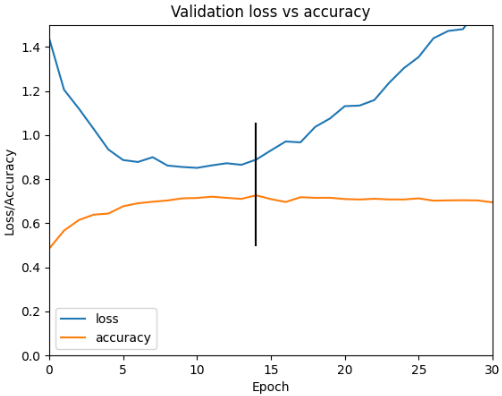
Validation loss vs accuracy

- Below is a plot of validation accuracy vs loss.
- Loss was multiplied by 0.5 to show them both on the same plot
- It is clear that accuracy plateaus after the loss starts increasing



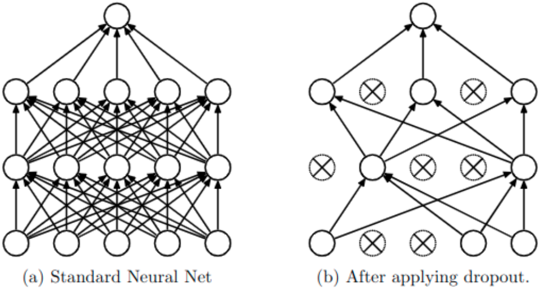
A zoomed view

- Below is a zoomed view of the previous figure
- A vertical line passing by epoch 14 (where max accuracy occurs) is shown



Dropout

- Dropout is another method used to minimize overfitting
- A dropout layer "drops" certain nodes from the NN as shown in the figure below



From: Srivastava et al.

What is happening exactly?

- Every training pass **different** nodes are dropped
- A neural network with n nodes that uses dropout, can be seen as a set of 2^n different "smaller" networks. Why?
- It means that using dropout is equivalent to averaging over an ensemble of Neural networks sharing the same weights
- Since the noise learned by each "smaller" network is random, "averaging" makes them cancel each other.

Implementation

- Using layers with an on/off switch on nodes is not practical
- More importantly, with an NN with large number of nodes it is **impossible**
- Instead, a node is turned off by multiplying its output with zero **with probability** p
- Mathematically, let $\mathbf{y}^l$ be the output of layer l in a fully connected network
- Then the output of layer $(l+1)$ is given by (* is element wise product)

$$
\begin{aligned}
\mathbf{r}^l &= Bernoulli(p) \\
\tilde{\mathbf{y}}^l &= \mathbf{r}^l * \mathbf{y}^l \\
\mathbf{z}^{l+1} &= \mathbf{W}^{l+1}\mathbf{y}^l + \mathbf{b}^{l+1} \\
\mathbf{y}^{l+1} &= \sigma(\mathbf{z}^{l+1})
\end{aligned}
$$

What about testing?

- We have seen that training a NN with dropout is is equivalent to training multiple "smaller" networks.
- The result is an average over all such networks.
- To be consistent we must use the weights learned in the training phase for testing
- This means we have to **average** over all possible networks, which is infeasible.
- We approximate this averaging by **not using dropout** during testing
- But the weights obtained in the training phase are multiplied by the probability p
- Equivalently the weights are multiplied by $\frac{1}{p}$ during training

Batch Normalization

- A Batch normalization layer transforms the distribution of the input to a distribution that has 0 mean and unit variance
- Batch normalization is a powerful method that
  1. Allows for better convergence of NN training
  2. Adding BN layers to existing NN yield better generalization results
  3. Reduces over fitting
  4. Training is less sensitive to meta-parameters (learning rate, weights initialization)

Batch Normalization- How does it work

- Given a mini-batch of tensors x_{ci} of dimension (S,C,H,W)
- where c is the channel index and i collectively refers to all other dimensions.
- Let $N = S \times H \times W$.
- Batch normalization computes the mean and variance of the batch (per channel) according to

$$\mu_c = \frac{1}{N} \sum_{i=1}^{N} x_{ci}$$

$$\sigma_c^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{ci} - \mu_c)^2$$

Batch Normalization- How does it work

The normalized inputs are computed as follows:

$$\hat{x}_{ci} = \frac{x_{ic} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}}$$

Therefore, for each channel, the $\hat{x}_{ci}$ have zero mean and unit variance. The output of the batch normalization layer is given by

$$y_{ic} = \gamma \hat{x}_{ic} + \beta$$

Where γ and β are **learnable** parameters.

Batch normalization - Why does it work?

- Considerable ongoing debate
- In the original paper of ‣ Ioffe and Szegedy it claims that it fixes the internal covariate shift
- meaning the change in the distribution due to the change of the weights during learning
- ‣ Kohler et al. propose that it separates the change in magnitude from the change of directions.
- This can be seen from the fact that the $\hat{x}_{ci}$ are normalized and all of them share the same magnitude γ

Example

Consider a mini-batch of size two, containing the tensors of size 2×2, A and B with both having a single channel.

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

- The mean is 4.5 and (biased) variance is 5.25 therefore the output is

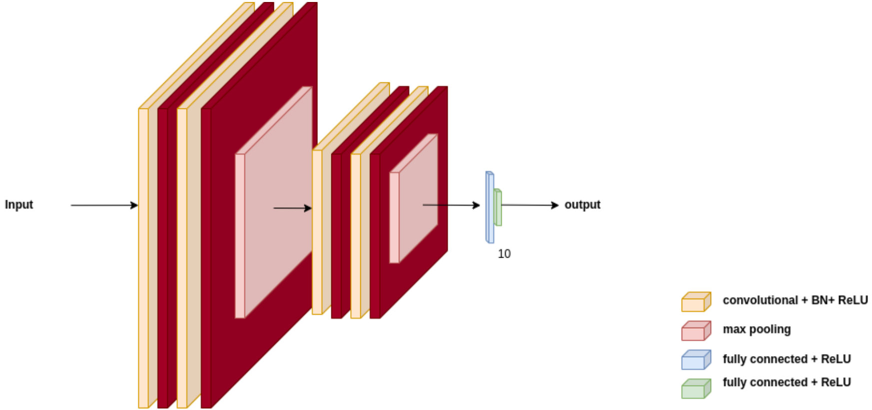
$$A = \begin{bmatrix} (1-4.5)/5.25 & (2-4.5)/5.25 \\ (3-4.5)/5.25 & (4-4.5)/5.25 \end{bmatrix} \qquad B = \begin{bmatrix} (5-4.5)/5.25 & (6-4.5)/5.25 \\ (7-4.5)/5.25 & (8-4.5)/5.25 \end{bmatrix}$$

- When the input has multiple channels, the same operation is performed for each channel independently

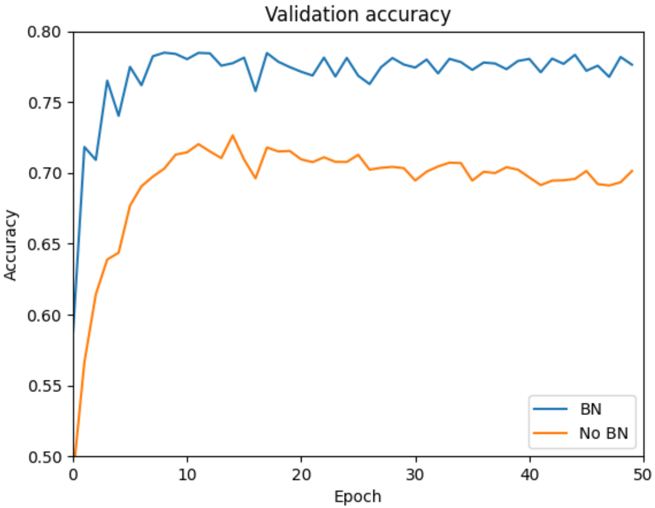
In practice

- Data: CIFAR10
- The NN has two convolutional blocks
- The first contains two sub-blocks each with
  1. 32 filters with receptive field of 3x3
  2. followed by BN layer
  3. followed by ReLU
- the sub-blocks are followed by a max pooling layer of size 2x2
- The second block is the same as the first but with 64 filter
- The conv blocks are followed by two feedforward layers with 128 and 10 nodes respectively

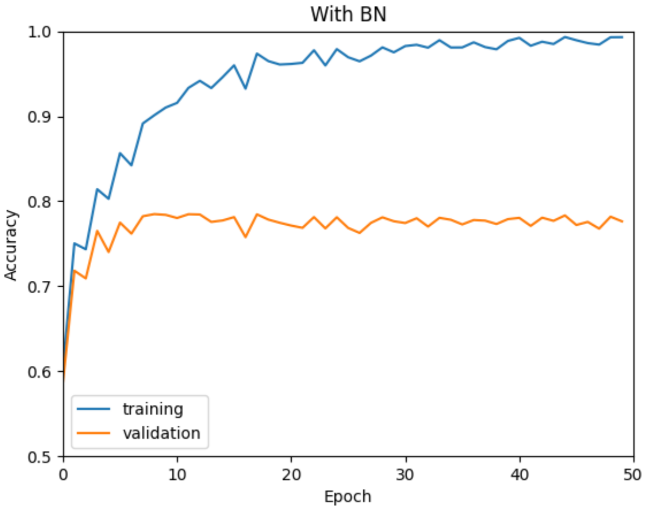
Nework Architecture



Results: validation accuracy of BN vs no BN



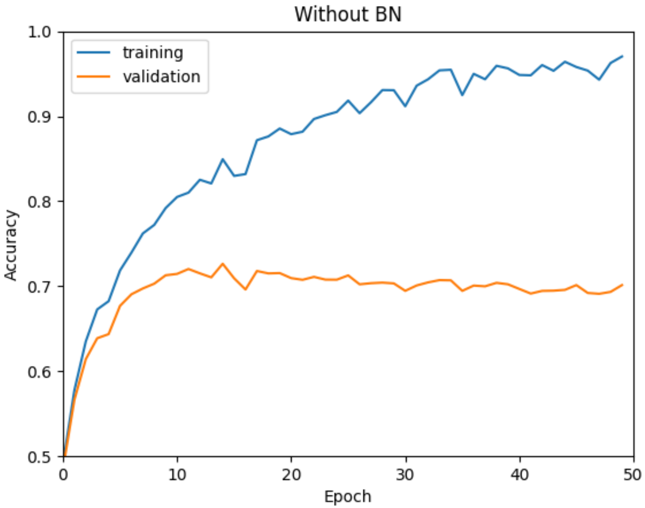
Results: train-vs-validation accuracy with BN



Analysis of results

- Batch normalization give much better accuracy.
- Peak value for the accuracy for BN is 78% vs 72%
- The average after the peak for BN is vs 75% vs 69%
- Also, with BN the peak acc is reached after 8 epochs vs 14 without BN.

Results: train-vs-validation accuracy without BN



References

- Ioffe, Sergey and Christian Szegedy (2015). "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167. arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.
- Kohler, Jonas et al. (2019). "Exponential convergence rates for batch normalization: The power of length-direction decoupling in non-convex optimization". In: *The 22nd International Conference on Artificial Intelligence and Statistics*. PMLR, pp. 806–815.
- Srivastava, Nitish et al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56, pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.