Role of Tweaks in Deep Learning

A review

M. Zadem K. Abouda TH. Yen Vu

MAP670L - Generalisation properties of algorithms in ML

Academic Year 2019/2020

Dropout

- Dropout is an intuitive technique that helps neural networks generalise better
- The idea is to randomly remove a neuron with probability p during each training phase, resulting in additional noise to the model
- During the validation, the whole network is used to predict the target values

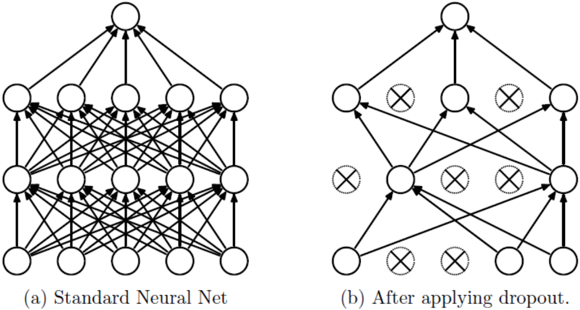


Figure: Dropout representation [SHK+14]

Experimental results

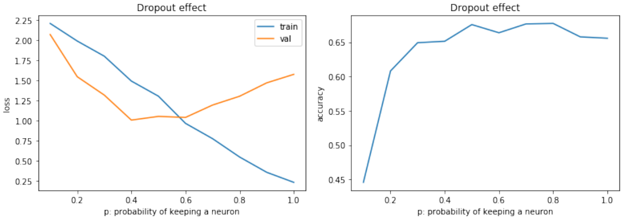


Figure: Effect of Dropout on model performance

The training performance is sacrificed in order to have better generalisation proprieties.

Large learning rate

- Neural networks almost always require a large learning rate to perform well and generalise
- The behavior of Neural nets when changing the step size is not totally understood: small learning rates allow the capture hard patterns with low noise while large learning rates are capable of capturing more general and easier patterns that feature high noise

Large learning rate

The work in [LWM19] takes a step towards explaining this behaviour. The idea is to work with a generated data composed of two components $\mathcal{P}$ and $\mathcal{Q}$.

- $\mathcal{P}$ models a high noise and easy to fit patterns
- $\mathcal{Q}$ models low noise with hard to fit pattern.

Large learning rate

Two algorithms were used to be trained on this data, each of them having different proprieties and consequently different performance.

- **Algorithm (S)** uses a small learning rate. (S) manages to fit the pattern featured in the component $\mathcal{Q}$ with a low number of observations. But struggles with $\mathcal{P}$ as the learning rate is too dependant on the high noise
- **Algorithm (LS)** uses a large initial step size combined with annealing. (LS) is able the successfully fit $\mathcal{P}$ in a initial phase using the large learning rate value. And when the annealing is introduced, the model switches to fitting $\mathcal{Q}$.

Experimental results

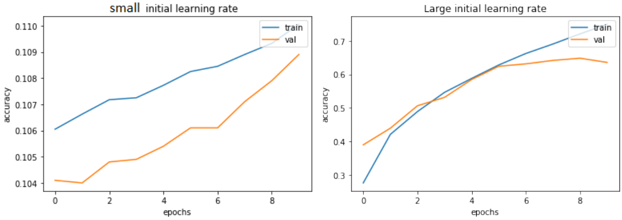


Figure: Effect of initial step size on model accuracy

Batch Norm

- Batch Normalization [SI15] is another technique that helps with generalization.
- Idea : Normalize hidden layers' output
- Normalization is performed on mini-batches and not the whole dataset resulting in introducing noise to the data and therefore a reguralization effect.
- Reducing *internal covariate shift* which speeds up the training process.

Batch Norm : How it works

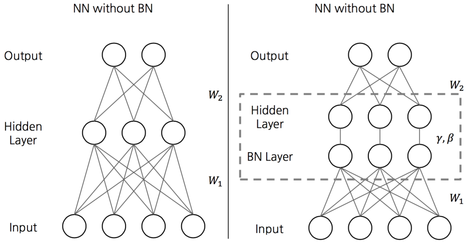


Figure: Batch Normalization

1. Normalize layers input to have zero mean and unit standard deviation with respect to the running batch
2. Add a learnable scale and bias term to allow network to still use the nonlinearity

BatchNorm Regularization effect

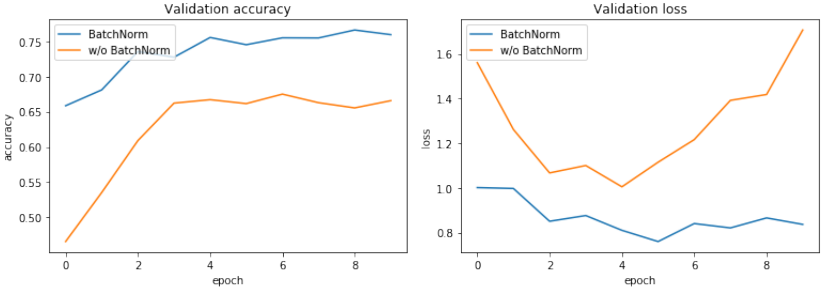


Figure: Effect of BatchNorm on model's performance

Batch Normalization effect on learning process

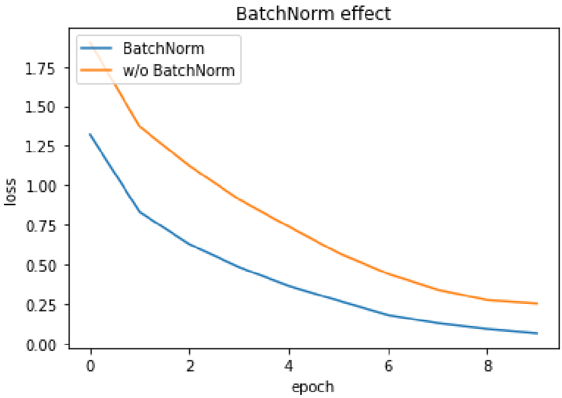


Figure: Training loss obtained with and without Batch Normalization

Weight Initialization

- The starting values of the weights can have a significant effect on the training process.
- Intuitively, initializing weights with a constant is not a good idea as the weights will remain constant during training.
- Thus, weights are initialized randomly, for example:
  - $w_i \sim \mathcal{N}(0, \sigma^2)$
  - $w_i \sim \mathcal{U}([-\sqrt{3}\sigma, \sqrt{3}\sigma])$

  for some standard deviation σ. However, what values of σ?

Weight Initialization

- **Lecun Initialization ([LBOM98]):**

$$\sigma^2 = \frac{1}{N}$$

where N is the number of input neurons at the current layer (with respect to the weight w_i).

- **Xavier/Glorot Initialization ([GB10]):**

$$\sigma^2 = \frac{2}{N_{in} + N_{out}}$$

where N_{in} and N_{out} are the number of input and output neurons of the current layer.

- **He Initialization ([HZRS15]):**

$$\sigma^2 = \frac{2}{N}.$$

Constant vs Random Initialization

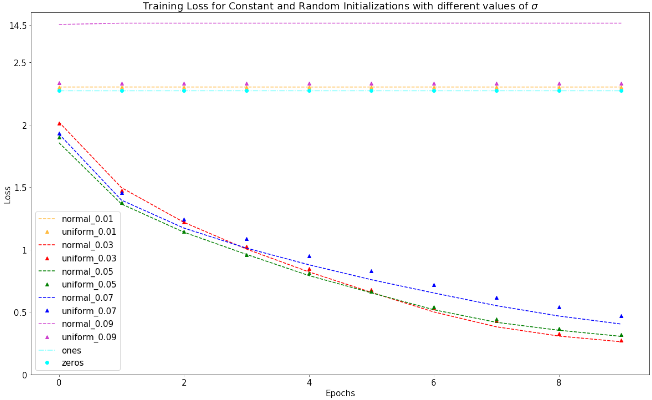
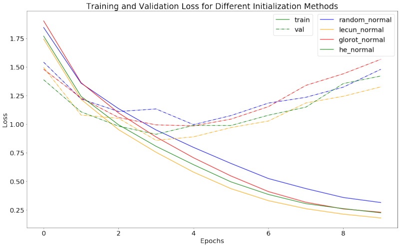
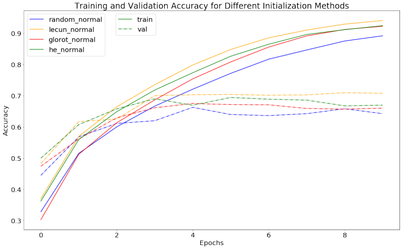


Figure: Training Loss for Constant vs Random initialization

Scaled Variance Initializations



(a) Train and Val loss



(b) Train and Val accuracy

Figure: Comparison of Scaled-Variance Initializations

Weight Decay

- Simply add a squared $L2$-norm to the loss function of the network:

$$L(w) = L_0(w) + \frac{1}{2}\lambda \sum_{i=1}^{M} w_i^2,$$

  where L_0 is the objective function.
- Restrict the network's complexity by preventing them from going too large.

Experimental Results

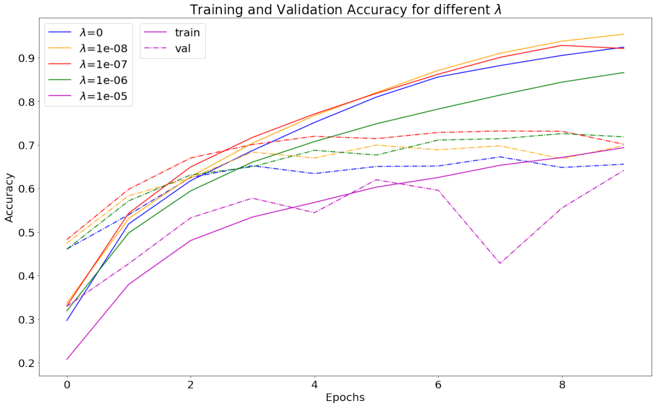


Figure: Impact of weight decay on train and validation accuracy

Experimental Results

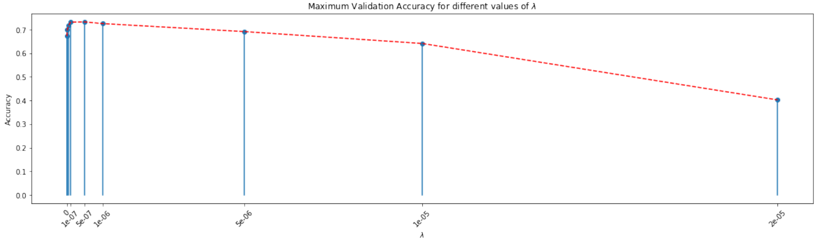


Figure: Effects of λ on validation result

Conclusion

- **Adding Noise :** Dropout, BatchNorm
- **Better ways for intializing model parameters :** Weight Initialization, Large learning rate
- **Adding regularization term :** Weight Decay

References I

- Xavier Glorot and Yoshua Bengio, *Understanding the difficulty of training deep feedforward neural networks*, In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics, 2010.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, 2015.
- Anders Krogh and John A. Hertz, *A simple weight decay can improve generalization*, NIPS, 1991.

References II

Yann LeCun, Leon Bottou, Genevieve Orr, and Klaus Müller, *Efficient backprop*, Neural Networks: Tricks of the Trade, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 1998, p. 546.

Yuanzhi Li, Colin Wei, and Tengyu Ma, *Towards explaining the regularization effect of initial large learning rate in training neural networks*, 2019.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, *Dropout: A simple way to prevent neural networks from overfitting*, Journal of Machine Learning Research **15** (2014), 1929–1958.

References III

Christian Szegedy Sergey Ioffe, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, 2015.