Regularization of Neural Networks - Mini-Project

Oliver Vipond

January 3, 2018

1 Introduction

Neural networks constructed to solve a machine learning problem typically have sufficient complexity to overfit training data. As such, one must implement regularization techniques to improve the generalization power of the neural network. There are a multiplicity of techniques one can employ to regularize a network including but not limited to: modifying the network architecture, adding a penalty of the parameter weights to the loss function, and enforcing that the network constructs similar embeddings of similar data [Mur12]. In this Mini-Project I aim to survey several of these techniques used to control overfitting in feed forward neural networks. I will discuss the advantages and disadvantages of each technique. Moreover I shall provide some examples of these techniques in action on a neural network to aid heuristic understanding and guide explanations as to why each of the techniques is effective in reducing overfitting.

2 L^1, L^2 - Regularisation

The first technique I shall look at is L^1, L^2-regularisation (L^2-regularisation is also known as Tikhonov or ridge regularization [GBC16]). To implement this method one introduces an L^1-penalty, an L^2-penalty, or a combination of the two to the cost function of the network. The penalty is not typically applied to the bias terms. This technique introduces the headache of a hyperparameter(s) to determine. One can choose a global hyperparameter for all weights in the network or a different parameter for each layer.

The general principle behind this regularization technique is to introduce a driving force to drag the weights towards zero, so that in the stochastic gradient descent updates we shall only increase weights which have a large contribution to decreasing the cost function. Let us consider why it might be undesirable to have many large weights in our neural network.

If we consider the non-linearity perceptrons in the network (e.g. $\tanh$, sigmoids), then for small parameter values these non-linear functions are operating in a roughly linear fashion, since for small inputs these functions are almost linear. Thus a network with almost every weight small is not a very complex model and acts approximately like a linear SVM. Permitting a few weights at a time to increase introduces complexity and the non-linearity functionality. A network with small weights gives rise to a smooth model in which the output changes slowly with a small change in input. In contrast many large weights can give rise to a sharp chaotic model, behaviour typically associated with overfitting.

The choice of penalty function gives rise to distinct behaviours. Consider for instance the situation where two inputs into a neuron encode very similar information. The L^2 penalty will

want to distribute the optimal weighting of this information to make the parameter weights equal, whereas the L^1 penalty will not drive this distribution of weights. As such the L^1 penalty often gives rise to sparse networks with many weights exactly zero, whereas the L^2 penalty has a smoothing effect on weights.

We can examine precisely the effect of L^2 regularisation, by seeing how a single gradient descent update changes when adding the regularisation. Moreover we can consider how the optimal weight values change in a simple model [GBC16]. Let $\mathbf{X}, \mathbf{y}, \mathbf{w}$ denote our input data, output data and parameter weights respectively. Let $J(\mathbf{w}; \mathbf{X}, \mathbf{y})$ denote our unregularized objective function. Our L^2-regularized objective function becomes:

$$J'(\mathbf{w}; \mathbf{X}, \mathbf{y}) = J(\mathbf{w}; \mathbf{X}, \mathbf{y}) + \frac{\alpha}{2}\mathbf{w}^t\mathbf{w}$$

where α is a hyperparameter to be determined and for simplicity we do not distinguish connection weights and biases.

The derivative of the regularized objective function hence becomes:

$$\nabla_{\mathbf{w}} J'(\mathbf{w}; \mathbf{X}, \mathbf{y}) = \nabla_{\mathbf{w}} J(\mathbf{w}; \mathbf{X}, \mathbf{y}) + \alpha\mathbf{w}$$

Consequently an ϵ-step update of parameter weights gives new parameter values:

$$\mathbf{w}_{\text{new}} = (1 - \epsilon\alpha)\mathbf{w} - \epsilon\nabla_{\mathbf{w}} J(\mathbf{w}; \mathbf{X}, \mathbf{y})$$

This formula makes it explicit that the update step will only increase the modulus of the parameter values that have a significant effect on the original objective function, and those without a significant effect on the original object function will have their modulus shrunk by the scaling factor $(1 - \epsilon\alpha)$.

Let us consider how the optimal value of an objective function changes under L^2 regularization. Let $\mathbf{w}^*$ denote the parameter values for which the original objective function is minimal. Using Taylor's Theorem a 2nd order approximation of the original objective function about $\mathbf{w}^*$ is given by the following expression:

$$J(\mathbf{w}) \approx J(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^t H(\mathbf{w} - \mathbf{w}^*)$$

Here the Hessian H is positive semi-definite since we are centred at a minimum. Moreover the Hessian is real and symmetric so has an orthogonal-diagonal decomposition $H = P\Lambda P^t$. Therefore an approximation of the gradient of the regularized objective function is given by:

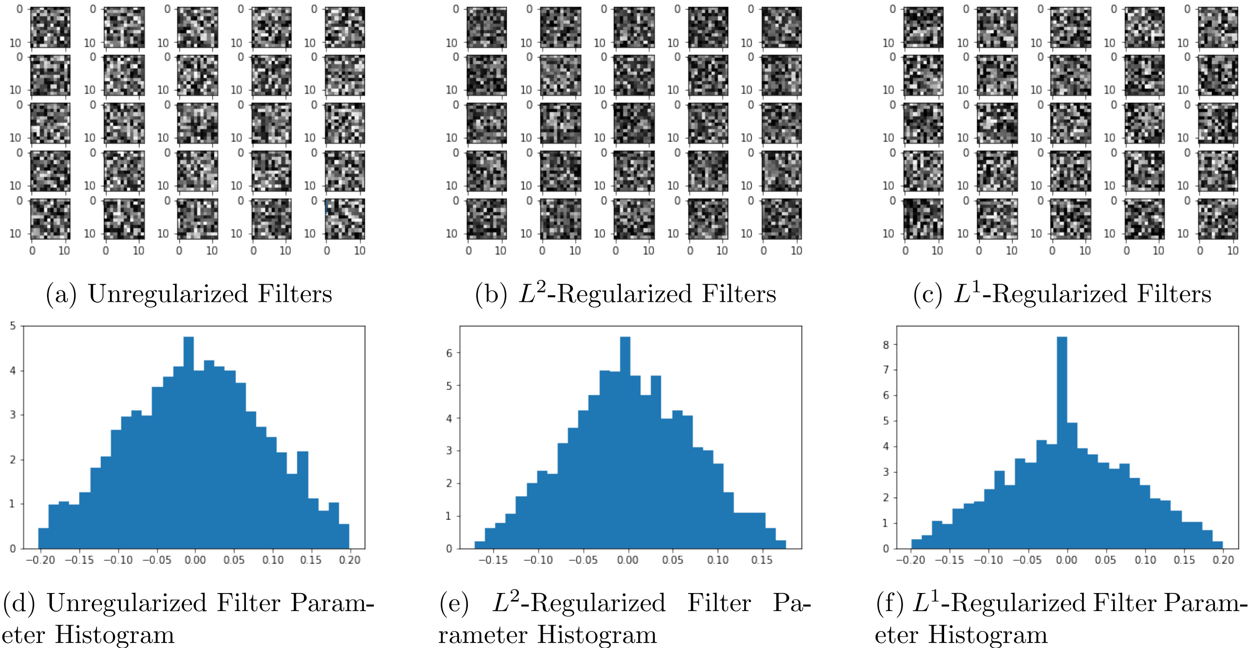
$$\nabla_{\mathbf{w}} J'(\mathbf{w}) \approx H(\mathbf{w} - \mathbf{w}^*) + \alpha\mathbf{w}$$

Setting the gradient to zero to find the optimal parameter values $\mathbf{w}_{\text{reg}}$ of the regularized objective we attain:

$$\mathbf{w}_{\text{reg}} = (H + \alpha I)^{-1}H\mathbf{w}^* = P(\Lambda + \alpha I)^{-1}\Lambda P^t\mathbf{w}^*$$

This expression makes it clear that the regularized optimal parameter values are attained by rescaling the original optimal parameter values by a factor of $\frac{\lambda_j}{\lambda_j+\alpha}$ along the direction of the jth eigenvector of the Hessian. Note that eigenvectors with large eigenvalues indicate directions in which a small parameter change will significantly increase the original objective function. The rescaling factor in these directions will be approximately 1 since λ_j will dominate α.

A similar analysis may be performed to see the precise effect of introducing an L^1-penalty term.



(a) Unregularized Filters

(b) L^2-Regularized Filters

(c) L^1-Regularized Filters

(d) Unregularized Filter Parameter Histogram

(e) L^2-Regularized Filter Parameter Histogram

(f) L^1-Regularized Filter Parameter Histogram

Figure 1: The effect of L^1, L^2 weight decay on the first convolutional layer of filters in a neural network

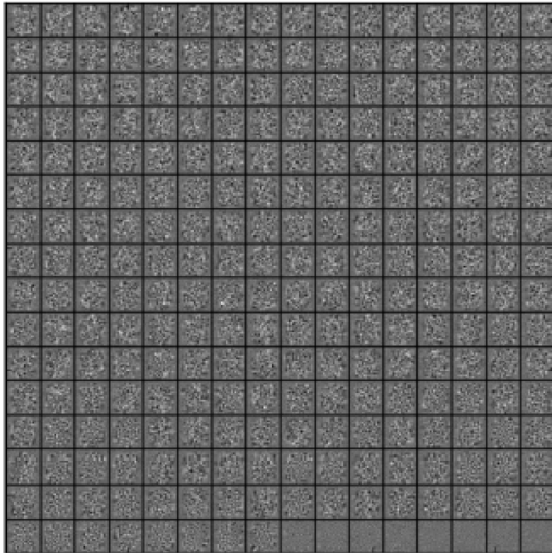
I have trained the convolutional neural network described in the final practical of the course with and without regularization to see the effect on weights in the network. Figure 1 shows the 25 12×12 filters from the first convolutional layer, and a histogram of the parameter values in these filters. We see the effect that each regularisation penalty has had on the weights in the filters. In Figure 1(f) the sparsity induced by the L^1 penalty is particularly apparent with the histogram of the weights sharply peaked at zero.

3 Dropout

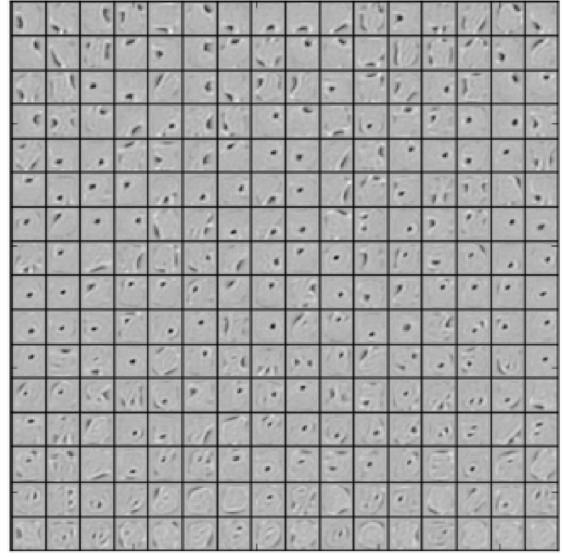
Dropout is now a common technique used for regularizing deep neural networks, and describes the process of randomly dropping connections in the network at the time of training [SHK+14]. Whilst there is still active research to determine precisely why this technique is an effective regularisation tool, there are various heuristic explanations that indicate why this technique works. A significant advantage of using dropout is the ease with which it may be applied to a network.

Traditional dropout acts on the units of a network and introduces dynamic sparsity in the model. A probability p of keeping unit in the network is chosen, and then at each stage of training only a thinned network is trained with only a fraction of the units present. If there are N total units then at each stage we are training one of 2^N possible networks. The final network then has its weights scaled by p and is applied to the test data with all units present. Sometimes one can improve performance by selecting distinct probablities of a unit being present for distinct layers of the network, typically a high probability ~ 0.8 for the input layer and ~ 0.5 in the rest of the network.

The process can be thought of as a computationally efficient way to simulate the effect of



(a) Filters trained without dropout ($p = 1$)



(b) Filters trained with dropout ($p = 0.5$)

Figure 2: The first layer of convolutional filters on a neural network trained with and without dropout [SHK+14]

training multiple networks and averaging their predictions. Moreover the dropout technique also prevents too much co-adaptation between units, since any given unit cannot always rely on the presence of any other unit. Consequently it is thought that it will be more likley that units develop their own independent function that will generalize with greater accuracy.

[SHK+14] describes a motivation for dropout from the field of biology. The motivation is in essence inspired by the advantages of sexual reproduction over asexual reproduction in the evolution of organisms. One may form an analogy between natural selection *optimising* genes in a species and the training of a neural network optimising the weights of the network. It is worth remarking that the most complex organisms have evolved through sexual reproduction rather than asexual reproduction.

Asexual reproduction slightly mutates the genes of the child of an individual and as such co-adaptation between genes is not likely to be interupted. In contrast to asexual reproduction, the random combination of genes from two individuals in sexual reproduction mean that large co-adapted sets of genes are likely to be split up. This pressure will likely cause individual genes to become more robust to the presence of other sets of genes. This is replicated in the dropout technique, where omitting units in training reduces co-adaptation between units. It is hoped that since dropout mimics sexual reproduction in training neural networks it shall encourage the robustness of an individual unit in performing an independent function.

[SHK+14] provides empirical evidence of this behaviour in action. A network is trained without dropout and then with dropout probability ($p = 0.5$) on the MNIST data set. Examining the first layer of learned features in Figure 2 we see that it appears the filters in the network trained without dropout have not individually learned meaningful features of the MNIST data set, whereas each filter in the network trained with dropout seems to detect an individual feature of a written character. It is worth remarking that in the previous assertion we are reliant on our intuition of

what constitutes a *meaningful* feature of the MNIST character set being correct.

A new paradigm for dropout has been proposed in [WZZ+13] and is called DropConnect. Rather than dropping units with some probability, instead individual connections are dropped. It is shown that DropConnect outperforms traditional dropout in some data sets, exhibiting both faster convergence and higher test accuracy.

4 Semi-Supervised Embedding

In order to generalize well, we expect a neural network to learn embeddings of objects that reflects their similarity i.e. similar objects should have similar embedding in the network [Mur12]. Semi-supervised embedding describes the regularisation technique of enforcing similar embeddings of similar objects. Typically one augments the cost function to include a cost that rewards embedding similar objects in a similar fashion.

A clear advantage of this method is that one can use unlabelled data to optimise the embedding. An example of this approach is presented in the paper [CW08], in which data from English Wikipedia pages is used to learn coherent English phrases. The network trained on this data can then be used as a starting point for other tasks that may not have sufficient labeled training data.

[Mur12] describes a possible implementation of the semi-supervised embedding technique. Let $\{\mathbf{x}_i : i \in \Lambda\}$ denote training data which may or may not be labelled, and let $f(\mathbf{x}_i)$ denote some embedding of the data $\mathbf{x}_i$, e.g. the values attained at each unit in some chosen layer of the network when $\mathbf{x}_i$ is parsed into the network. For each pair of indices let $S_{ij} = \mathbf{1}_{\{\mathbf{x}_i \text{ and } \mathbf{x}_j \text{ are } \textit{similar}\}}$ for some determined similarity measure. For each pair of indices we define the embedding loss as:

$$L_{ij} = S_{ij}\|f(\mathbf{x}_i) - f(\mathbf{x}_j)\| + (1 - S_{ij}) \max\{0, \delta - \|f(\mathbf{x}_i) - f(\mathbf{x}_j)\|\}$$

If objects $\mathbf{x}_i$ and $\mathbf{x}_j$ are deemed similar they contribute the norm of the difference between their embeddings. If they $\mathbf{x}_i$ and $\mathbf{x}_j$ are deemed disimilar then the embedding loss pushes the embeddings to differ by some determined margin δ.

The loss function of the network is augmented to include the sum of the embedding losses of each pair of data. The contribution of the embedding loss is scaled by some hyperparameter. Optimising the weights of the network via stochastic gradient descent will thus also encourage the network to give similar objects a similar embedding. There is clearly a multitude of choices one can make in employing this technique, e.g. norm, margin, hyperparameter, embedding to optimise, similarity measure.

Another interesting example in which semi-supervised learning is used to improve network performance is presented in the paper [MCW09]. The task is visual object recognition and in this setting the unlabeled data is given by video recordings.The underlying assumption of the model is that frames of a video file which are close in time are likely to contain similar content, subject to subtle changes in position and lighting. The similarity measure used in the paper deems frames to be similar if they are consecutive. The embedding of the data $(f(\mathbf{x}_i))$ that is trained to be similar is the representation at some depth of the network, and the L^1 norm is chosen. Training their convolutional network with unlabeled video data sees an improvement in performance on a variety of data sets.

5 Conclusion

I have briefly surveyed a few regularization techniques for avoiding overfitting in neural networks. Whilst there is no silver bullet for regularization in neural networks, it is common for a combination of dropout and L^2-regularization to be used. In practice the most effective regularization tool is likely to be task specific, so it is useful to have a varied toolkit of regularization techniques. The regularization techniques I have discussed in this short essay are by no means exhaustive, further techniques include: max-norm constraints, addition of artificial noise, and soft weight sharing [Mur12].

References

[CW08] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 160–167, New York, NY, USA, 2008. ACM.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[MCW09] Hossein Mobahi, Ronan Collobert, and Jason Weston. Deep learning from temporal coherence in video. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 737–744, New York, NY, USA, 2009. ACM.

[Mur12] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[SHK+14] 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*, 15:1929–1958, 2014.

[WZZ+13] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1058–1066, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.