Regularization for Neural Networks

L. Graesser

July 31, 2016

Research into regularization techniques is motivated by the tendency of neural networks to to learn the specifics of the dataset it was trained on rather than learning general features that are applicable to unseen data. This is known as overfitting. The goal of any supervised machine learning task is to approximate a function that maps inputs to outputs, given a dataset of examples and labels. An important assumption is that models are trained on datasets which are representative of the true distribution of the data and the target function to be approximated. However, almost all data is noisy and contains some random deviations from the true distribution. Given this fact, approximating the function represented by the training data very precisely is undesirable. The algorithm will learn the noise in the training dataset and will be unlikely to perform well when applied to unseen data. Regularizing neural networks helps them to learn the true function and ignore the noise.

Neural networks are high capacity models capable of approximating very complex functions, so are particularly susceptible to overfitting. Training a neural network by programming it to minimize a loss function, a measure of how well a network matches training data labels, is the unavoidable source of the overfitting problem in supervised learning tasks. In order to make the value of the loss as small as possible, the network matches the training data as closely as possible. However, this is not actually the right measure of a network's performace, only a substitute. The real goal is to accurately label unseen data. Since it is impossible to train a neural network using the real goal, it is trained it using this substitute and designed in such a way so as to increase the chances that it performs well on the real task. How effective this is in practice depends on the capacity of the neural network (i.e. whether the architecture of the network can actually approximate the true function), how much training data there is and how representative it is of the true distribution of the data, how noisy the training data is, and whether the neural network chooses the right function to approximate. This suggests three mechanisms to improve neural network performance.

1. Increase the complexity of the neural network by adding more layers and / or more nodes per layer. This expands the family of functions that a specific network can approximate, increasing the likelihood that the true function that maps inputs to outputs is included in this family.

2. Get more and better data, giving the network more information about the true function.

3. Regularize the model to make it more likely that the network approximates the true function out of the family of functions that it is capable of approximating.

They are all related. The first point suggest that more complex networks are better since they are more likely to be able to approximate the true function. However it also increases the likelihood that the network will learn the wrong function, one that better approximates the training data. An alternative approach is to lower the complexity of the model, reducing the extent to which a network can overfit the data. However, unless the prediction problem is very well understood, this is likely to result in a model that cannot approximate the true function well and will lead to poor performance on both the training data and unseen data. A better approach is to do all three of the above. Design high capacity networks, invest in getting more and better data, and regularize them, to help the network approximate the true function and prevent overfitting.

The three most common forms of regularization are:

- Weight penalties
- Early Stopping
- Dropout

All of these have been shown to work well in practice, in particularly dropout. Interestingly, why they do is not well understood mathematically. Instead, progress has been made through the development of hypotheses as to what may work well, guided by experience, and supported with experimental evidence. The custom within the deep learning community of making results and publications freely available, as well as the availability of standardized datasets such as the ImageNet datasets, has likely helped speed up the spread and development of different regularization techniques.

Weight penalties add an additional element to the loss function, the size of which is related to the absolute size of all of the weights in a neural network. A parameter, λ, controls how high a penalty to impose since it is multiplied by the weight penalty. The larger λ is, the higher the penalty, and $\lambda = 0$ corresponds to no weight penalty at all. λ is not a learnt parameter, it is chosen by the network designer, typically through hyper-parameter tuning.

The effect of weight penalties is to reduce the size of the weights in the network on average, since large weights will result in a higher loss. If a neural network has no weight penalties then weights tend to be pushed away from zero to more extreme values, even if this results in only a marginal decrease in the loss. Consider the case where the output of a network is a probability distribution over the possible categories achieved by having the softmax activation function in the nodes of the last layer. The output values are $\in (0, 1)$ but the labels are 1 for the correct category, and 0 for all of the others. The network will never output a 1 but may continue to adjust the weights to more extreme values in order to push to output of the network closer and closer to 1s and 0s. But once a network is 99% or even 99.9% sure that an example belongs to a particular category there is little benefit to it becoming more sure, and instead increases the risk of overfitting the training data. This is because a large weight has the effect of making the output of the network sensitive to small changes in the feature that it corresponds to because changes in this feature are amplified by being multiplied by a large weight. Small changes in the value of this input feature can lead to large changes in the output. In theory this could be correct, but a more likely outcome is that the network is fitting noise in the data. One heuristic guiding the weight penalty approach is that neural networks should be relatively locally insensitive, i.e. inputs from similar local regions should produce similar outputs. Small changes in the input should not result in large changes in the output.

The weight penalty increases the hurdle for large weights. The larger the value of the λ parameter, the larger the hurdle. It will cause weights to be small except for those which have a significant effect on reducing the loss by improving the output of the network. Enough to offset the increase in the loss due to regularization. A large weight can be interpreted as a particular feature being important for predicting the correct output. Constraining the set of large weights to those which have a significant effect on the loss function increases the likelihood that features with large weights are generally important. That is, the important features learned by the model are part of the true function we want it to approximate.

Two forms of weights regularization are used most often, $L2$ and $L1$. The difference is how the weights are penalized in the loss function. Let C be any loss function used to train a neural network. Let C_i be the loss for one example. The equations below give the modification to the loss function for $L2$ and $L1$ weight penalties, computed across n examples

$$\begin{aligned} L2 : C &= \frac{1}{n}\Big(\sum_{i=1}^{n} C_i + \frac{\lambda}{2}\sum_{w \in W} w^T w\Big) \\ L1 : C &= \frac{1}{n}\Big(\sum_{i=1}^{n} C_i + \lambda \sum_{w \in W} |w|\Big) \end{aligned} \tag{1}$$

$$\begin{aligned} &L2 : \frac{\partial C}{\partial w} = \lambda w \quad \textit{(regularization component of the loss function only)} \\ &L1 : \frac{\partial C}{\partial w} = \lambda\ sign(w) \quad \textit{(regularization component of the loss function only)} \end{aligned} \tag{2}$$

For $L2$ regularization, the derivative of the regularization component of the loss function with respect to a weight matrix, w, is λw. So, the contribution from $L2$ regularization to the weight update depends on the current value of the weights. For $L1$ regularization, the derivative of the regularization component of the loss function with respect to a weight matrix, w, is $\lambda sign(w)$, where $sign(w)$ is a matrix of the same size as w in which each element is the sign of the corresponding element of w. The contribution from $L1$ regularization during the weight update step is therefore proportional to λ and independent of the current value of w. This has the effect of shrinking w towards 0, taking λ sized steps. In contrast, $L2$ shrinks w towards 0 in λw sized steps. The closer w is to 0 the smaller the update with $L2$ regularization. This is not the case for $L1$ regularization. The result is that whilst weights are typically small with $L2$ regularization, they do not tend to be 0. In contrast, $L1$ regularization tends to enforce sparsity on the model, making many weights 0. So only a few nodes (features) are active and contribute to mapping inputs to outputs. Which to choose is application dependent.

Early stopping is a very intuitive technique. It has a regularizing effect by stopping the training process before the model learns specifics of the training data. While the model is learning it is typical to see the error (percentage of incorrectly classified examples) fall at the beginning for both the training and test set. As the number of iterations increases, the training set error should continue to fall, at a decreasing rate and eventually plateau. However the test set error may plateau and then begin to increase, as shown in the figure below. This is a sign that the model has started to overfit the training data.

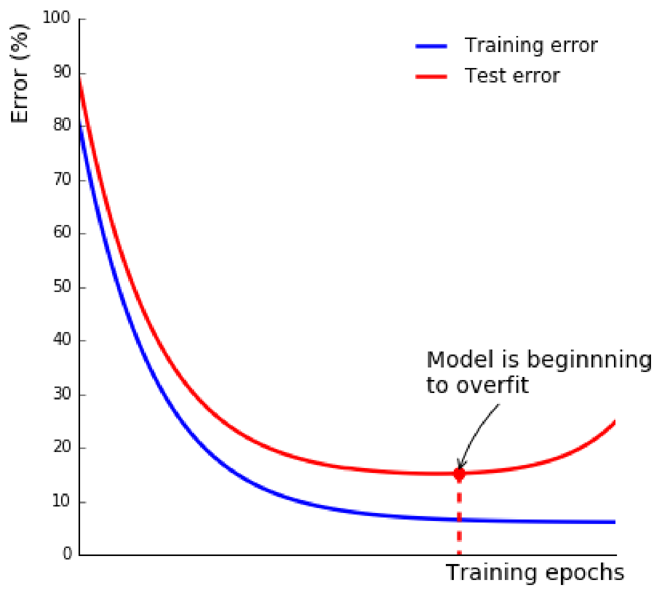

Figure 1: Training vs. test error

Training a neural network until the test error starts to increase and then returning to the model which generated the minimum error would be the most thorough way to implement early stopping. However, it is often too expensive to keep track of the network's weights for every iteration. Instead, a common approach is to monitor both the training and test error and stop training once the test error hasn't decreased for a few epochs, 10 for example.

Another approach to regularization is to use an ensemble (collection) of models to make the final prediction. The intuition behind this approach is that a number of models with the same accuracy may make different errors. Making the final prediction by selecting the class that the majority of models predict to be the correct one will decrease the error rate of the overall result and the performance of the model on both the training and the test set will increase. A similar idea is application to regularization. Different models with the same ability to generalize may learn different specific features of the training data. Taking the majority vote of these models smooths out the noise. The key is to try to build models that make errors or learn training data noise that are independent from each other. And to do this in a computationally efficient manner.

Dropout is an ingenious technique for building an ensemble of independent neural networks with the same data and network architecture in a computationally efficient way. Each time a batch is processed during stochastic gradient descent, each node in the hidden layers is turned off (output 0) with some positive probability, p, typically 50%. This has the effect of randomly generating a subnetwork from the main neural network. It is with this subnetwork that the feedforward, backpropagation and weight update steps are made. With each batch a new subnetwork is generated and therefore a different subset of the total available features are used to make a prediction. Once the model has been trained, the probability of a node being turned on needs to be reset to 1 and the weights multiplied by p before it is used on test data. The multiplication by p turns out to be a good approximation of the expected value of the output of a unit over all the possible subnetworks and is known as the *weight scaling inference rule*[1]. This is the mechanism by which the majority vote is computed. The result is equivalent to averaging the output of a very large number of neural networks which have been trained on the same data.

During training each hidden node cannot rely on a particular input being present. So when a node is detecting a feature the network must learn to be able to do this in a number of different ways. This could be by replicating feature detection. For example, if a node detects the presence of a face by identifying if a nose is present, then the previous layer may have multiple features for detecting noses, making the face detection feature robust to any one of the nose features being turned off. Alternatively, a node may learn to detect a face by identifying if one or more of an eye, mouth, or nose is present. Making the network robust to any one of the nose, mouth or eye detectors in the layer below being turned off. Understood this way, dropout has the effect of preventing nodes from overly relying on any one of its inputs to detect the presence of a feature. It can also be thought of as producing nodes that detect features that are not just good in one context, but instead are good features in multiple contexts[2]. This is exactly the goal of regularization.

A significant advantage of this approach is that it is efficient. Because parameters between these networks are shared by virtue of them being part of the parent network, it is possible to represent an exponential number of neural networks in a feasible amount of memory[3]. It is also through parameter sharing that good values for the weights and biases are reached despite the fact that only one subnetwork is being trained at each step. Computationally it only takes twice as many operations to make the feedforward, backpropagation and weight update steps as a network of the same size without dropout. The two networks are of the same order of algorithmic complexity and the multiplier for the additional steps of a network with dropout is small. This is partially offset by the need to increase the size of the network so that the average subnetwork has the same capacity as a network without dropout. For example, if $p = 50\%$ the number of nodes in the network needs to be doubled. Overall however, the efficiency gains are significant.

Many other forms of regularization are available to a network designer. Below I list just a few. If you are interested in learning more about regularization then I recommend reading chapter 7 of

[1]Deep Learning; Goodfellow, Bengio, Courville; ch 7, page 263, http://www.deeplearningbook.org/contents/regularization.html

[2]Deep Learning; Goodfellow, Bengio, Courville; ch 7, page 267, http://www.deeplearningbook.org/contents/regularization.html

[3]Deep Learning; Goodfellow, Bengio, Courville; ch 7, page 259, http://www.deeplearningbook.org/contents/regularization.html

Deep Learning by Ian Goodfellow, Yoshua Bengio, and Aaron Courville[4]

- Dataset augmentation: The dataset is expanded by applying small transformations to the training data that mimic variations that are likely to be encountered in unseen data and are unlikely to change the class of the example. This has the effect of making the network robust to these types of transformations in the data. For images it is quite easy to imagine what they could be. For example, shifting all pixels left, right, up or down by 1, changing the color saturation, rotation by a degree or two, and making multiple crops of the image. Large rotations however are not appropriate. Consider an image of the digit 9. A large rotation may change this into a 6.

- Parameter sharing: Parameters are shared across parts of the network (between groups of nodes). A widely used and extremely effective example of this is a convolutional neural network. This family of networks makes use of layers which contain small feature maps, perhaps 5×5 pixels wide. These feature maps are slid across the whole image. The 25 weights of the feature map are shared by a set of nodes in the layer above which computes the extent to which one feature appears in a particular patch on an image. The result is a set of weights that are associated with recognizing one feature, and they do so across the whole image. This makes them robust to small variations in the appearance and location of that feature, and less likely to overfit.

- Adversarial training: Images are generated which caused the network to make incorrect predictions with high confidence. These images are then used to tune the network weights to prevent it from making similar errors. This makes the network more robust to small permutations in the data within the parameter space region of a particular class[5]. However, Nguyen, Yosinski, and Clune[6], showed that it is difficult to make neural networks robust to all adversarial images.

Regularization refers to a set of techniques applied to neural networks to prevent them from overfitting the training dataset. They are an essential part of any network designer's toolkit. Often techniques are combined to good effect. A common strategy is to use early stopping, augment data sets where possible, and apply dropout to fully connected layers. Convolutional layers do not need any additional regularization since they are self regularizing. Empirically, dropout seems to have replaced weight penalties for fully connected layers. Dropout is a little trickier to apply to recurrent neural networks since dropout on recurrent connections can make it difficult for networks to learn to to store information. However, Zaremba, Sutskever, and Vinyals[7], showed that dropout applied to only the non recurrent connections in an LTSM network has an effective regularizing effect whilst still allowing the network to learn.

I hope that you've enjoyed this post and found it useful. If you have any feedback please send me an email at contact@learningmachinelearning.org

[4] http://www.deeplearningbook.org/contents/regularization.html

[5] Deep Learning; Goodfellow, Bengio, Courville; ch 7, page 269, http://www.deeplearningbook.org/contents/regularization.html

[6] Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images, https://arxiv.org/pdf/1412.1897.pdf

[7] Recurrent Neural Network Regularization, Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals, 2015, https://arxiv.org/pdf/1409.2329.pdf