Improving Generalization for Convolutional Neural Networks

Carlo Tomasi

November 29, 2021

Stochastic Gradient Descent (SGD) minimizes the training risk $L_T(\mathbf{w})$ of neural network h over the set of all possible network parameters in $\mathbf{w} \in \mathbb{R}^m$. Since the risk is a very non-convex function of $\mathbf{w}$, the final vector $\hat{\mathbf{w}}$ of weights typically only achieves a local minimum. Even so, empirical evidence shows that at these minima the risk is often very low, sometimes even zero. In other words, deep neural networks often overfit.

Under some circumstances, overfitting has shown empirically to be less of an issue for deep neural networks than classical statistical learning theory suggests, at least when training achieves zero empirical risk [1]. However, the theory behind these "interpolating neural networks" is still in its infancy. For the purposes of this course, we still take the classical view that overfitting hurts, a fact that still holds at least for networks that are unable to interpolate the training set fully.[1]

One way to prevent overfitting is somehow to prevent the optimization algorithm from freely traversing the parameter space. In this way, the effective size of the hypothesis space is reduced, and better generalization is possible. Somehow biasing what part of the parameter space can be explored during training is called *regularization*.[2]

Here are some ways to regularize the optimization problem:

- Prevent the algorithm from falling into shallow local minima. Of the techniques discussed in this note, this is perhaps the most difficult to envision as an example of regularization. However, one can imagine the optimization landscape (that is, the plot of the function $f(\mathbf{w}) = L_T(\mathbf{w})$) as a terrain riddled with local minima with high risk, and interspersed with many fewer "good" local minima, where the risk is much lower. If shallow minima are somehow avoided, then only the part of parameter space with deeper minima is open for exploration. In this way, the size of the effective parameter space has been reduced. The momentum method described in a previous note is such a technique.

- Keep the magnitude of $\mathbf{w}$ small. This technique, called *weight decay* in the deep learning literature, is regularization in the strictest sense, and is discussed in Section 1. It amounts to imposing a penalty on parameter vectors with large magnitude, thereby encouraging exploration of the parameter space only in a small neighborhood of the origin.

- Place a time deadline on exploration of the parameter space. If descent is stopped before a minimum is reached, local or otherwise, the algorithm can only explore a part of parameter space that is not too far from the initial vector $\mathbf{w}_0$. The size of this part of space is not

[1]Recall that interpolating means fitting with zero residual risk.

[2]In applied mathematics, the term "regularization" takes on a more technical and specific meaning. The term is used more loosely in this note.

easy to predict, since it depends on how much progress is made in every iteration of SGD. Nonetheless, free-wheeling exploration of the parameter space is thereby avoided. We already saw the technique of *early termination* in a previous note: Rather than imposing a fixed deadline, this technique stops descent once a validation risk bottoms out, rather than stopping when the training risk reaches a minimum. This technique is the most data-driven of all the methods discussed here, and is perhaps the most important because of this. It is briefly revisited in Section 2.

- Average an ensemble of neural networks. We saw the effectiveness of ensemble predictors when we studied random forests. The rub with ensemble methods for neural networks is that the cost of training and deploying a whole set of networks is typically prohibitive. Instead, the technique of the *dropout*, discussed in Section 3, attempts to simulate an ensemble of networks with a single network. The theoretical motivation for this method is dubious, but empirical observation has shown its effectiveness in many situations.

Other than regularization, generalization can be helped by providing more data. The effectiveness of this approach is limited, since data space is itself very large in many deep learning problems. The curse of dimensionality then suggests that adding data indiscriminately will not help. Instead, the technique of *data augmentation*, discussed in Section 4, places new data samples near the existing ones, to make predictions less sensitive to small perturbations of the inputs.

1 Weight Decay

What is called *weight decay* in the literature of deep learning is called L_2 *regularization* in applied mathematics, and is a special case of *Tikhonov regularization* [6]. Rather than minimizing $L_T(\mathbf{w})$, one regularizes

$$g(\mathbf{w}) = L_T(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2 \; .$$

The parameter λ is ideally determined by validation. However, validation is typically too expensive for deep learning, so one ends up setting λ to some small positive real value (for instance, 0.01). With this added term, parameters vectors with large magnitude are unlikely to be visited, since the increase in $\|\mathbf{w}\|_2^2$ that would result with these is unlikely to be compensated by a greater decrease in $L_T(\mathbf{w})$. Therefore, only parameter vectors that are not too far from the origin are explored.

Tikhonov regularization is a well-studied technique in applied mathematics, where the functions being optimized are often simpler than those implemented by a neural network. In deep learning, there is little theory about this technique. Empirically, some amount of weight decay has proven beneficial, and is nearly always included.

2 Early Termination

We already discussed early termination: One monitors the zero-one risk of the classifier on a validation set while training on the cross-entropy risk of the soft-max output on the training set. Training then stops when the validation-set error bottoms out, even if the training-set risk would continue to decrease.

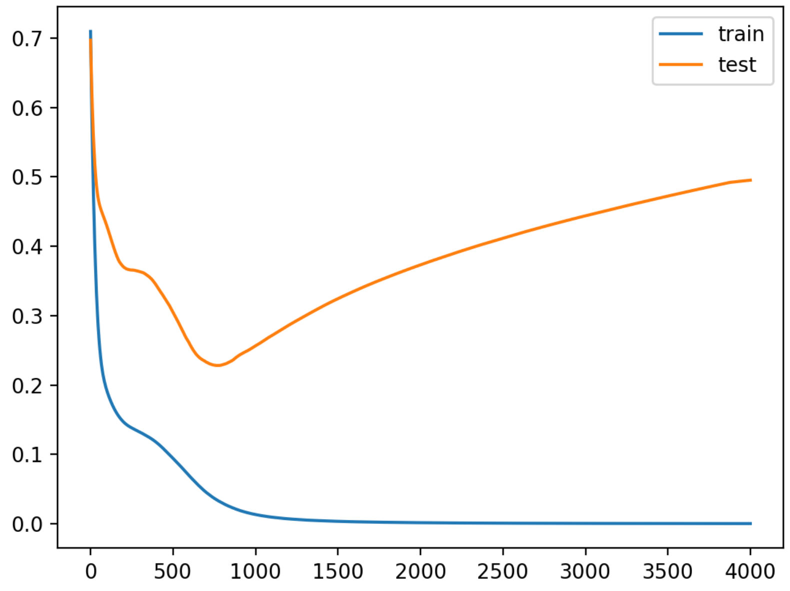

Figure 1: Training and validation risk as a function of SGD epoch number for a simple, artificial deep learning problem. The optimization algorithm is ADAM [2]. Picture from a recent blog. The training risk keeps decreasing, but the validation risk reaches a minimum after about 700 iterations. The parameters obtained at that point are returned as a result of training.

Figure 1 shows an example from a recent blog, which also contains Keras code for the example. In more complex cases, the two functions are often not as clean, and deciding when the validation risk has bottomed out can be tricky. Deep learning packages typically provide a "patience" parameter, which is some positive integer p. If the validation risk has not improved in p epochs, training is stopped, and the parameters that achieve the lowest validation loss up to that point are returned.

3 Dropout

In general, the best way to avoid overfitting in the presence of limited data would be to build an ensemble of networks. The networks are made different from each other either by different initialization or by changing their architecture. They are trained on different views of the data obtained by bagging. Finally, their predictions are averaged for regression or aggregated by majority voting for classification. This approach, which is reminiscent of building a forest of trees, is obviously infeasible for nontrivial nets because of its high computational cost.

One way to approximate this scheme in a computationally efficient way is called the *dropout* method [4]. Given a deep network to be trained, a *dropout network* is obtained by flipping a biased coin for each node of the original network and "dropping" that node if the flip turns out heads. Dropping the node means that all the weights and biases for that node are set to zero, so that the node becomes effectively inactive. Equivalently, the output of the node is clamped to zero.

One then trains the network by using mini-batches of training data, and performs one iteration of training on each mini-batch after dropping neurons independently with probability $1 - p$. When training is done, all the weights in the network are multiplied by p. This rescaling is meant to effectively average the outputs of the nets with weights that depend on how often a unit participated in training. The value of p is typically set to $1/2$.

Each dropout network can be viewed as a different network, and the dropout method is loosely equivalent to effectively sampling a large number of nets efficiently. The theoretical case for the dropout method is only partially convincing. However, empirical results show that the method works well in many circumstances.

4 Data Augmentation

Data augmentation is not a regularization technique, but improves generalization performance nonetheless. This technique is best described with an example. Suppose that you need to classify images into one of several categories (labels). For simplicity, assume that each image contains a single object, so the category for it is uncontroversial. A sample $(\mathbf{x}, y)$ from the training set T comprises and image $\mathbf{x}$ and the corresponding label y. Data augmentation generates a new set of samples $(\mathbf{a}_1(\mathbf{x}), y), \ldots, (\mathbf{a}_j(\mathbf{x}), y)$ by modifying $\mathbf{x}$ in some way and assigning the same label y to each new image.

Typical modifications are either photometric or geometric, and more often a combination of the two. A photometric variation will change the colors in the image at random. For instance, color $\mathbf{c} = (r, g, b)$ may be replaced by color $\mathbf{c}' = \mathbf{c} + \delta\mathbf{n}$ where δ is a small positive real number and $\mathbf{n}$ is a random 3-vector out of a standard distribution. To make the colors in the modified image more realistic, one often evaluates the probability distribution p of colors over the entire training set, and then generates new colors from the values of p in a small neighborhood of $\mathbf{c}$.

Geometric modifications, on the other hand, translate, rotate, shear, or scale the image, or apply some other geometric transformation to it.

Of course, it would be hopeless to "fill" data space X by data augmentation, because of the curse of dimensionality and the large dimensionality of X. Instead, data augmentation "thickens" the cloud of points $\mathbf{x}_1, \ldots, \mathbf{x}_N$ in the original training set T by surrounding each point with a small cloudlet of neighboring points that share the same label. In this way, the predictor becomes less sensitive to small variations of the inputs away from the original training samples, and therefore tends to generalize better to new data, as long as the new data is not too different from that in the training set. Since the known label of $\mathbf{x}$ is assigned to all its augmentations $\mathbf{a}_i(\mathbf{x})$, this technique requires no new data annotation, and can be applied automatically and with relatively little effort. It is used nearly always in deep networks.

References

[1] Mikhail Belkin, Daniel J Hsu, and Partha Mitra. Overfitting or perfect fitting? risk bounds for classification and regression rules that interpolate. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 2300–2311. Curran Associates, Inc., 2018.

[2] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[3] B. T. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

[4] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[5] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1139–1147, 2013.

[6] A. N. Tikhonov and V. Y. Arsenin. *Solutions of ill-posed problems*. Halsted Press, New York, NY, 1977.