NEURAL NETWORKS AND DEEP LEARNING CST 395 CS 5TH SEMESTER HONORS COURSE- Dr Binu V P, 9847390760

CS 5th Semester Honors course for the Computer Science at KTU- Dr Binu V P

Practical Issues in Neural Network Training

September 07, 2022

In spite of the formidable reputation of neural networks as universal function approximators, considerable challenges remain with respect to actually training neural networks to provide this level of performance. These challenges are primarily related to several practical problems associated with training, the most important one of which is overfitting.

1.The Problem of Overfitting

The problem of overfitting refers to the fact that fitting a model to a particular training data set does not guarantee that it will provide good prediction performance on unseen test data, even if the model predicts the targets on the training data perfectly. In other words, there is always a gap between the training and test data performance, which is particularly large when the models are complex and the data set is small.

In order to understand this point, consider a simple single-layer neural network on a data set with five attributes, where we use the identity activation to learn a real-valued target variable. Therefore, the network tries to learn the following function:

$\hat{y} = \sum_{i=1}^{5} w_i . x_i$

Consider a situation in which the observed target value is real and is always twice the value of the first attribute, whereas other attributes are completely unrelated to the target.However, we have only four training instances, which is one less than the number of features (free parameters). For example, the training instances could be as follows:

x_1	x_2	x_3	x_4	x_5	y
1	1	0	0	0	2
2	0	1	0	0	4
3	0	0	1	0	6
4	0	0	0	1	8

The correct parameter vector in this case is $W = [2, 0, 0, 0, 0]$ based on the known relationship between the first feature and target. The training data also provides zero error with this solution, although the relationship needs to be learned from the given instances since it is not given to us a priori. However, the problem is that the number of training points is fewer than the number of parameters and it is possible to find an infinite number of solutions with zero error. For example, the parameter set $[0, 2, 4, 6, 8]$ also provides zero error on the training data. However, if we used this solution on unseen test data, it is likely to provide very poor performance because the learned parameters are spuriously inferred and are unlikely to generalize well to new points in which the target is twice the first attribute (and other attributes are random). This type of spurious inference is caused by the paucity of training data, where random nuances are encoded into the model. As a result, the solution does not generalize well to unseen test data. This situation is almost similar to learning by rote, which is highly predictive for training data but not predictive for unseen test data.

Increasing the number of training instances improves the generalization power of the model, whereas increasing the complexity of the model reduces its generalization power. At the same time, when a lot of training data is available, an overly simple model is unlikely to capture complex relationships between the features and target. A good rule of thumb is that the total number of training data points should be at least 2 to 3 times larger than the number of parameters in the neural network, although the precise number of data instances depends on the specific model at hand. In general, models with a larger number of parameters are said to have high capacity, and they require a larger amount of data in order to gain generalization power to unseen test data. *The notion of overfitting is often understood in the trade-off between bias and variance in machine learning.*

The key take-away from the notion of bias-variance trade-off is that one does not always win with more powerful (i.e., less biased) models when working with limited training data, because of the higher variance of these models. For example, if we change the training data in the table above to a different set of four points, we are likely to learn a

completely different set of parameters (from the random nuances of those points). This new model is likely to yield a completely different prediction on the same test instance as compared to the predictions using the first training data set. This type of variation in the prediction of the same test instance using different training data sets is a manifestation of model variance, which also adds to the error of the model; after all, both predictions of the same test instance could not possibly be correct. More complex models have the drawback of seeing spurious patterns in random nuances, especially when the training data are insufficient. One must be careful to pick an optimum point when deciding the complexity of the model.

Neural networks have always been known to theoretically be powerful enough to approximate any function . However, the lack of data availability can result in poor performance; this is one of the reasons that neural networks only recently achieved prominence.The greater availability of data has revealed the advantages of neural networks over traditional machine learning . In general, neural networks require careful design to minimize the harmful effects of overfitting, even when a large amount of data is available. Next section provides an overview of some of the design methods used to mitigate the impact of overfitting.

Regularization

Since a larger number of parameters causes overfitting, a natural approach is to constrain the model to use fewer non-zero parameters. In the previous example, if we constrain the vector W to have only one non-zero component out of five components, it will correctly obtain the solution $[2, 0, 0, 0, 0]$. Smaller absolute values of the parameters also tend to overfit less. Since it is hard to constrain the values of the parameters, the softer approach of adding the penalty $\lambda||W||^p$ to the loss function is used. The value of p is typically set to 2, which leads to **Tikhonov regularization**. In general, the squared value of each parameter (multiplied with the regularization parameter $\lambda > 0$) is added to the objective function.
The practical effect of this change is that a quantity proportional to λw_i is subtracted from the update of the parameter w_i. An example of a regularized version of equation for mini-batch S and update step-size $\alpha > 0$ is as follows:
$\overline{W} \Leftarrow \overline{W}(1 - \alpha\lambda) + \alpha \sum_{\overline{X} \in S} E(\overline{X})\overline{X}$

Here, $E[X]$ represents the current error $(y - \hat{y})$ between observed and predicted values of training instance X. One can view this type of penalization as a kind of weight decay during the updates. Regularization is particularly important when the amount of available data is limited. A neat biological interpretation of regularization is that it corresponds to gradual forgetting, as a result of which “less important” (i.e., noisy) patterns are removed.In general, it is often advisable to use more complex models with regularization rather than simpler models without regularization.

The general form of above Equation is used by many regularized machine learning models like least-squares regression , where $E(X)$ is replaced by the error-function of that specific model. Interestingly, weight decay is only sparingly used in the single-layer perceptron because it can sometimes cause overly rapid forgetting with a small number of recently misclassified training points dominating the weight vector; the main issue is that the perceptron criterion is already a degenerate loss function with a minimum value of 0 at $W = 0$ (unlike its hinge-loss or least-squares cousins). This quirk is a legacy of the fact that the single-layer perceptron was originally defined in terms of biologically inspired updates rather than in terms of carefully thought-out loss functions. Convergence to an optimal solution was never guaranteed other than in linearly separable cases. For the single-layer perceptron, some other regularization techniques, which are discussed below, are more commonly used.

Neural Architecture and Parameter Sharing

The most effective way of building a neural network is by constructing the architecture of the neural network after giving some thought to the underlying data domain. For example, the successive words in a sentence are often related to one another, whereas the nearby pixels in an image are typically related. These types of insights are used to create specialized architectures for text and image data with fewer parameters. Furthermore, many of the parameters might be shared. For example, a convolutional neural network uses the same set of parameters to learn the characteristics of a local block of the image. The recent advancements in the use of neural networks like recurrent neural networks and convolutional neural networks are examples of this phenomena.

Trading Off Breadth for Depth

As discussed earlier, a two-layer neural network can be used as a universal function approximator , if a large number of hidden units are used within the hidden layer. It turns out that networks with more layers (i.e., greater depth) tend to require far fewer units per layer because the composition functions created by successive layers make the neural network more powerful. *Increased depth is a form of regularization,* as the features in later layers are forced to obey a particular type of structure imposed by the earlier layers. Increased constraints reduce the capacity of the network, which is helpful when there are limitations on the amount of available data. The number of units in each layer can typically be reduced to such an extent that a deep network often has far fewer parameters even when added up over the greater number of layers. This observation has led to an explosion in research on the topic of deep learning.

Even though deep networks have fewer problems with respect to overfitting, they come with a different family of problems associated with ease of training. In particular, the loss derivatives with respect to the weights in different layers of the network tend to

have vastly different magnitudes, which causes challenges in properly choosing step sizes. Different manifestations of this undesirable behavior are referred to as the vanishing and exploding gradient problems. Furthermore, deep networks often take unreasonably long to converge. These issues and design choices will be discussed later

Ensemble Methods

A variety of ensemble methods like **bagging** are used in order to increase the generalization power of the model. These methods are applicable not just to neural networks but to any type of machine learning algorithm. However, in recent years, a number of ensemble methods that are specifically focused on neural networks have also been proposed. Two such methods include **Dropout and Dropconnect**. These methods can be combined with many neural network architectures to obtain an additional accuracy improvement of about 2% in many real settings. However, the precise improvement depends to the type of data and the nature of the underlying training. For example, normalizing the activations in hidden layers can reduce the effectiveness of Dropout methods, although one can gain from the normalization itself.

2.The Vanishing and Exploding Gradient Problems

While increasing depth often reduces the number of parameters of the network, it leads to different types of practical issues. Propagating backwards using the chain rule has its drawbacks in networks with a large number of layers in terms of the stability of the updates. In particular, the updates in earlier layers can either be **negligibly small (vanishing gradient)** or they can be **increasingly large (exploding gradient)** in certain types of neural network architectures.This is primarily caused by the chain-like product computation in back progation Equation which can either exponentially increase or decay over the length of the path.

In order to understand this point, consider a situation in which we have a multi-layer network with one neuron in each layer. Each local derivative along a path can be shown to be the product of the weight and the derivative of the activation function. The overall backpropagated derivative is the product of these values. If each such value is randomly distributed, and has an expected value less than 1, the product of these derivatives in Equation will drop off exponentially fast with path length. If the individual values on the path have expected values greater than 1, it will typically cause the gradient to explode. Even if the local derivatives are randomly distributed with an expected value of exactly 1, the overall derivative will typically show instability depending on how the values are actually distributed.

In other words, the vanishing and exploding gradient problems are rather natural to deep networks, which makes their training process unstable.Many solutions have been proposed to address this issue. For example, a **sigmoid** activation often encourages the vanishing gradient problem, because its derivative is less than 0.25 at all values of its argument , and is extremely small at saturation. A **ReLU** activation unit is known to be

less likely to create a vanishing gradient problem because its derivative is always 1 for positive values of the argument. More discussions on this issue are provided later. Aside from the use of the ReLU, a whole host of gradient-descent tricks are used to improve the convergence behavior of the problem. In particular, the use of adaptive learning rates and conjugate gradient methods can help in many cases. Furthermore, a recent technique called batch normalization is helpful in addressing some of these issues.

3.Difficulties in Convergence

Sufficiently fast convergence of the optimization process is difficult to achieve with very deep networks, as depth leads to increased resistance to the training process in terms of letting the gradients smoothly flow through the network. This problem is somewhat related to the vanishing gradient problem, but has its own unique characteristics. Therefore, some “tricks” have been proposed in the literature for these cases, including the use of gating networks and residual networks

4.Local and Spurious Optima

The optimization function of a neural network is highly nonlinear, which has lots of local optima. When the parameter space is large, and there are many local optima, it makes sense to spend some effort in picking good initialization points. One such method for improving neural network initialization is referred to as **pretraining**. The basic idea is to use either supervised or unsupervised training on shallow sub-networks of the original network in order to create the initial weights. This type of pretraining is done in a greedy and layerwise fashion in which a single layer of the network is trained at one time in order to learn the initialization points of that layer.
This type of approach provides initialization points that ignore drastically irrelevant parts of the parameter space to begin with. Furthermore, unsupervised pretraining often tends to avoid problems associated with overfitting. The basic idea here is that some of the minima in the loss function are spurious optima because they are exhibited only in the training data and not in the test data. Using unsupervised pretraining tends to move the initialization point closer to the basin of “good” optima in the test data. This is an issue associated with model generalization.

Interestingly, the notion of spurious optima is often viewed from the lens of model generalization in neural networks. This is a different perspective from traditional optimization.In traditional optimization, one does not focus on the differences in the loss functions of the training and test data, but on the shape of the loss function in only the training data. Surprisingly, the problem of local optima (from a traditional perspective) is a smaller issue in neural networks than one might normally expect from such a nonlinear function. Most of the time, the nonlinearity causes problems during the training process itself (e.g., failure to converge), rather than getting stuck in a local

minimum.

5.Computational Challenges

A significant challenge in neural network design is the running time required to train the network. It is not uncommon to require weeks to train neural networks in the text and image domains. In recent years, advances in hardware technology such as **Graphics Processor Units (GPUs)** have helped to a significant extent. GPUs are specialized hardware processors that can significantly speed up the kinds of operations commonly used in neural networks. In this sense, some algorithmic frameworks like **Torch** are particularly convenient because they have GPU support tightly integrated into the platform.

Although algorithmic advancements have played a role in the recent excitement around deep learning, a lot of the gains have come from the fact that the same algorithms can do much more on modern hardware. Faster hardware also supports algorithmic development, because one needs to repeatedly test computationally intensive algorithms to understand what works and what does not. For example, a recent neural model such as the **long shortterm memory(LSTM)** has changed only modestly since it was first proposed in 1997 . Yet, the potential of this model has been recognized only recently because of the advances in computational power of modern machines and algorithmic tweaks associated with improved experimentation.

One convenient property of the vast majority of neural network models is that most of the computational heavy lifting is front loaded during the training phase, and the prediction phase is often computationally efficient, because it requires a small number of operations (depending on the number of layers). This is important because the prediction phase is often far more time-critical compared to the training phase. For example, it is far more important to classify an image in real time (with a pre-built model), although the actual building of that model might have required a few weeks over millions of images. Methods have also been designed to compress trained networks in order to enable their deployment in mobile and space-constrained settings

Summary

Although a neural network can be viewed as a simulation of the learning process in living organisms, a more direct understanding of neural networks is as computational graphs. Such computational graphs perform recursive composition of simpler functions in order to learn more complex functions. Since these computational graphs are parameterized, the problem generally boils down to learning the parameters of the graph in order to optimize a loss function. The simplest types of neural networks are often basic machine learning models like least-squares regression. The real power of neural networks is unleashed by using more complex combinations of the underlying functions. The parameters of such networks are learned by using a dynamic

programming method, referred to as backpropagation. There are several challenges associated with learning neural network models, such as overfitting and training instability. In recent years, numerous algorithmic advancements have reduced these problems. The design of deep learning methods in specific domains such as text and images requires carefully crafted architectures. Examples of such architectures include recurrent neural networks and convolutional neural networks. For dynamic settings in which a sequence of decisions need to be learned by a system, methods like reinforcement learning are useful.

Popular posts from this blog

NEURAL NETWORKS AND DEEP LEARNING CST 395 CS 5TH SEMESTER HONORS COURSE NOTES - Dr Binu V P, 9847390760

October 03, 2022

About Me Syllabus Question Paper Dec 2022 Module 1 (Basics of Machine Learning) Overview of Machine Learning Machine Learning Algorithm Linear Regression Capacity, Overfitting and Underfitting Regularization Hyperparameters and Validatio ...

READ MORE

Machine Learning Algorithm

October 01, 2022

A machine learning algorithm is an algorithm that is able to learn from data. But what do we mean by learning? Mitchell (1997) provides the definition "A computer program is said to learn from experience E with respect to some class of tasks T and ...

READ MORE

Syllabus CST 395 Neural Network and Deep Learning

October 01, 2022

Syllabus Module - 1 (Basics of Machine Learning) Machine Learning basics - Learning algorithms - Supervised, Unsupervised, Reinforcement, overfitting, Underfitting, Hyper parameters and Validation sets, Estimators -Bias and Variance. Challenges ...

READ MORE

DR.BINU V P-9847390760

Associate Professor and Head Dept of Computer Science-IHRD Karunagappally.Love to share the thoughts and views .I play lot of Games and basically an Athlet in my school and college days.I never keep my studies as a second option.Stood first In MTech and BTech.I did my resea ...

VISIT PROFILE

Archive