Lecture 19: Regularization in Neural Networks

Instructor: Aditya Bhaskara *Scribe: Chitradeep Dutta Roy*

CS 5966/6966: Theory of Machine Learning

March 24th, 2017

Abstract

This lecture reviewed the back-propagation algorithm that uses stochastic gradient descent to train feed-forward neural networks from previous class. Then some strategies to control the problem of overfitting was discussed and ended with a brief introduction of convolutional neural networks.

1 Generalization in Neural Networks

So far we have seen how to train neural netwroks with back-propagation algorithm which acts fairly well in practice despite the problem of ERM in neural networks being computationally hard. But now we look at the eternal question of machine learning which is whether the network is able to generalize or simply overfit to the sample training data. Now we will see a few approaches that are frequently used to mitigate overfitting in neural networks.

Low complexity networks The first approach that comes to mind is to choose a network structure having a low VC-dimension, as we are already familiar that hypothesis classes with low VC-dimensions have limited richness in terms of what they can represent and therefore generalize better in practice. But the problem with this approach is usually the following,

- Usually the training becomes harder (it takes much longer for gradient descent to converge / find an answer), as there might be very few networks with the given structure that is optimum.
- The structure might not be rich enough to capture the optimum function for the problem. This issue is known as *Bias*.

1.1 *Regularization*

A common approach to controlling the complexity of a neural network is to add a regularization term to the objective loss function. The regularization term must capture the complexity of the network. The idea is that when the training algorithm tries to minimize the regularized objective function it will try to decrease both the original objective loss and also the complexity term related to the network. In general, the regularized objective function for a network class $\mathcal{H}_{V,E,\sigma}$ with weight function as a vector $\mathbf{w} \in \mathbb{R}^{|E|}$ looks like the following,

$$\mathcal{L}_R(\mathbf{w}) = \mathcal{L}(\mathbf{w}) + \lambda \text{Comp}(\mathbf{w}).$$

Here $\lambda \in [0, \infty)$ is a hyperparameter deciding the relative contribution of the penalty term, *Comp* and the original loss fucntion $\mathcal{L}$.

L1 regularization One common choice for the penalty term is *L*1 regularization. It looks like the following,

$$\text{Comp}_{L1}(\mathbf{w}) = ||\mathbf{w}||_1 = \sum_i |w_i|.$$

The property of this regularizer is that it generally favors *sparse* solutions, meaning most of the entries in the weight vector $\mathbf{w}$ is zero.

L2 regularization This is perhaps the most popular choice for regularizer in neural networks. It looks like the following,

$$\text{Comp}_{L2}(\mathbf{w}) = ||\mathbf{w}||_2^2 = \sum_i w_i^2.$$

The property of this regularizer is that it favors weights that are *small and well spread*.

1.2 *Data Augmentaion*

The best way to make any machine learning model to generalize better is to provide it with more and more input data, but in practice data is limited. Here we will introduce a few strategies to overcome this problem.

Injecting Noise One way to make neural networks more robust or in other words increase the networks stability is to train the network by applying random noise to its inputs. The idea is that the behavior of the network should not change much if a small amount of noise is added to the input. Noise injection can also be done to the units in the hidden layers of the network which can be seen to be doing data augmentation to multiple levels of abstraction.

Dropout Technique This is another powerful approach which can be regarded as creating new inputs by multiplying the original with random noise. As training neural networks usually is expensive in terms of runtime and memory, this approach gives an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks. Basically, dropout trains an ensemble of networks by removing randomly selected units and associated edges in each round from non-output layers of the base structure of the network and then running gradient descent on the induced network.

This technique helps to minimize the networks sensitivity to any particular subset of features which might not be desirable, and helps in overall generalization.

More details on regularization and data augmentation can be found in *Chapter 7 of the Deep Learning book by Goodfellow, Bengio and Courville*.

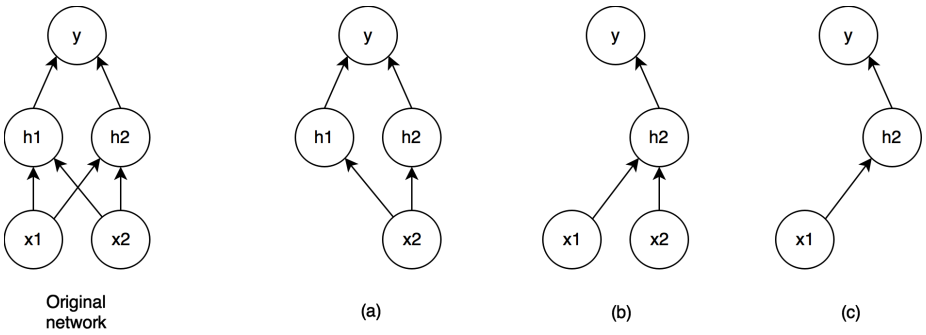

Figure 1: On the left is the original network and (a), (b), (c) are three induced networks by deleting different nodes by dropout algorithm, there can be 16 unique networks for this example

2 Convolutional Neural Networks

Convolution neural networks (CNN) are specific kind of networks that have a known grid-like topology and uses a method known as *convolution* in atleast one of its layers. Following are some kinds of situations where these are very useful,

- Features have some kind of known structure.
- Features are local.
- Features are shift invariant.

Edge detection One very simple example would be a network that detects edges in an image. A simple algorithm to detect edge can be designed as follows look at any 4 adjacent pixels in a row, lets call them p_1, p_2, p_3, p_4 in order from left to right. Now we can detect vertical edges if $|p_1 + p_2 - p_3 - p_4| > \epsilon$ for a stack of such rows. A similar logic can be used to for detecting horizontal edges using a stack of columns. Now one can detect edges this way for a whole image by just using a kernel that detects edges in a $4 * 4$ image and then tiling the whole image (let's say $m * n$) with that same kernel.

Convolutional neural networks have been hugely successful in many applications in domains like computer vision and speech.

Convolution depends on three important ideas *sparse interactions, parameter sharing and equivalent representations*. Because of these proeprties they are much more efficient both in terms of memory and runtime than traditional neural networks which looks at interactions between units that several orders of magnitude greater.

More details can be found on *Chapter 9 of the Deep Learning book by Goodfellow, Bengio and Courville*.