Lecture 23: Group Presentations

Instructor: Aditya Bhaskara　　*Scribe: Christopher Mertin*

CS 5966/6966: Theory of Machine Learning

April 17th, 2017

Abstract

There were 4 different presentations over various topics which included Adversarial examples for neural nets, Saddle point problem in deep nural networks, Active learning, and ImageNet Classification with Deep Convolutional Nural Networks.

1 Adversarial examples for neural nets

`https://arxiv.org/pdf/1412.6572.pdf`

This paper explored the use of neural networks misclassifying adversarial examples. Adversarial examples are defined as having random perturbations applied to the inputs such that the perturbed input forces the model to make a false prediction.

This new paper disreguards the previous attempts to explain with nonlinearity and overfitting. Instead, they propose that the cause is inherent with the neural networks, and is due to their linearity. This explination comes about from new results and allowed them a way to quickly develop adversarial examples.

They state that LSTMs, ReLUs, and maxout networks are intentionally designed to behave linearly to make it easy to optimize. Even the sigmoid networks are carefully tuned to spend their predictions in the linear section of the output to also help with optimization.

This linearity allows for cheap, analytical perturbations of a linear model, though this can bring about problmes with linear perturbations.

The way that the adversarial examples are generatedis by taking some input image x and adding $\epsilon \times r$ where r is a random perturbation. This noise r is a small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input $\text{sign}\left(\nabla_x J(\theta, x, y)\right)$. ϵ is the magnitude of the smallest bit of an 8 bit image encoding after using GoogLeNet to convert it to real numbers.

This method causes a wide variety of models to misclassify their input. They propose methods to fix this in deep networks. They propose a change of the cost function to add on a regularization term to help, turning the cost function into $J(\theta, x + \epsilon\text{sign}(\nabla_x J(\theta, x, y)), y)$. Doing this allowed for the models to help combat the adversarial inputs, but they could not get it to be reduced to a 0% error rate.

Generalizing the adversarial examples across different models can be explained as a result of adversarial perturbations being highly aligned with the weight vectors of a model, and different models learnign similar functions when trained to perform the same task.

They were able to come to the conclusion that Radial Basis Functions were resistant to adversarial examples, and might be better utilized.

2 Saddle point problem in deep nural networks

`https://arxiv.org/pdf/1405.4604.pdf`

In high dimensional spaces, you are likely to run into saddle points which can prevent you from reaching a minimum. There are 4 aspects to these spaces, all relative to the Hessian of some function $f(\theta)$

- If all the eigenvalues are non-zero and positive, then the critical point is a local minimum
- If all the eigenvalues are non-zero and negative, than the critical point is a local maximum
- If the eigenvalues are non-zero and we have both positive and negative eigenvalues, then the critical point is a saddle point with a *min-max* structure
- If the Hessian matrix is singular, then the *degenerate* critical point can be a saddle point or a monkey saddle.

This plateaus are poor in reaching the critical point. The most common method to be used is Newton's method, which is also known to get stuck at plateau's in all aspects of the sciences, and are more common than any other critical point.

To deal with sattle points, Newton's method rescales by $1/\lambda_i$ in the given eigendirection. However, this leads to some issues, if $\lambda_i < 0$, then it can cause problems in reaching the critical point and get stuck on a saddle point.

The authors propose a fix, scaling by $1/|\lambda_i|$, which they denote as the "saddle-free Newton method." This method worked on a scaled-down version of MNIST, though might be unfeasible for larger problems due to the computational complexity. However, it did bring about significant improvement in how long it took to actually learn.

3 Active learning

`http://www.cs.northwestern.edu/~pardo/courses/mmml/papers/active_learning/improving_generalization_with_active_learning_ML94.pdf`

"Active learning" means that the learning algorithm has some control over the order in which it receives examples. This allows for the network to give a better generalization over standard implementations as it doesn't need to utilize all of the input to get the same result.

This paper explores the use of active learning in the binary field *without* noise. This form is known as *selective sampling* and describes how to approximately implement it in regards to a neural network.

In the sequential process, you would draw examples one after the other. So, for example, if you have a binary classification problem to be split, it will take

$\mathcal{O}(n)$ examples to learn the region dividing the two sets of data. However, with active learning, you can *query* the data to get the most relevant points to speed up thsi learning process. This allows for you to get $\mathcal{O}(\log(n))$ data points to learn the boundary.

They go on to suggest a naïve neural network querying algorithm, such that you divide the domain into three classes:

- "1" for a value ≥ 0.9
- "0" for a value ≤ 0.1
- "uncertain" for everything else

This allows them to query values in these three categories so that they can actively learn. However, they go on to explore different orders in which they learn with the above query categories and they note that this method is prone to failure.

They go on to explore the use of a *version-space* approach, which provides a partial ordering in generality of the concepts being learned. Therefore, if some concept c_1 is "more general" than another c_2, and if and only if $c_2 \subset c_1$, then we can use a version-space approach.

However, even though this proved to be better, the authors go on to mention that this only works with relatively simple concept classes. As the classes become more complex, it becomes difficult to predict what should be learned next. Though in the tests, selective sampling did provide a significant improvement over passive, random sampling techniques. However, as the number of concepts increase, the cost teo identify the "next best" unlabeled example increases drastically.

4 ImageNet Classification with Deep Convolutional Neural Networks

`https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-network`
`pdf`

The ImageNet challenge has over 15 million labeled high-resolution images of variable resolutions, with over 22,000 categories. The authors used one of the largest CNNs on ImageNet, which made it composed of 60 million parameters and 650,000 neurons.

This network was one of the first to implement a convolutional neural network on the ImageNet data set. Most previous implementations were using Scale-Invariant Feature Transform (SIFT) and Fisher Vectors (FVs) to identify objects. However, rather than predefining the required features with SIFT Features and FVs, this work used CNNs in such a way that the CNNs learned them themselves.

The network consisted of 5 convolutional layers, some of which are followed by a maxpooling layer, and 3 fully connected layers with a final 1000-categorical softmax. In order to decrease overfitting, dropout was utilized in the fully connected layers.

The use of convolutional layers are important as fully-connected layers for real world (large) images are computationally expensive. Convolutional layers exploit recurring and sparse features available in an image or signal. This allows us to share weights and convolutional layers.

The components of a CNN are that they are composed of filters. These filters identify features by using a "sliding window" on the image to identify these features. They also utilize a max pooling layer to reduce the full size of the input.

ReLU layers are used to zero-out negative values and to give neurons in the network an "activation" while the fully connected layers in a NN usually appear in the final layers.

The convolution on an image i is composed of a filter f giving

$$\frac{\sum_{x,y} f(x,y)i(x,y)}{9}$$

Which is repeatedly applied over the entire image.

Max pooling selectes a subsection of the image and represents those values with a maximum value to reduce the number of parameters. For example, if you have a 2×2 square with the values $[[0.77, -0.11], [-0.11, 1.00]]$, if you apply max pooling on the layer you reduce the 2×2 values down to one, giving $[1.00]$.

The use of ReLU allowed them to reach a 25% training error rate 6 times faster than other implementations. The optimization function that they used was SGD with a batch size of 128, momentum of 0.9, and a weight decay of 0.0005.

By utilizing these concepts, they were able to perform 10% better than the other implementations at the time, achieving a Top-5 test accuracy of 15.3%, while the SIFT and Feature Vector implementations that were being used at the time got 26.2%.