Assignment 2

Anonymous & Anonymous

6 April 2017

In this assignment we aim to understand, apply, and master *Deep Neural Networks* and *Deep Learning*. In the first section we will focus on quickly describing 3 types of Deep Neural Networks (Convolutional, Recurrent, and Autoencoders), plus providing a proof of understanding through experimenting with existing code and results. The second section will then introduce our DL Challenge project, followed by a detailed description about our approach, experimentation, results, and a concluding discussion.

1. Deep Neural Networks

Convolutional

A *Convolutional Neural Network (CNN)*, the first (potentially) Deep Neural Network we consider, uses insights from the organisation of the visual cortex in the brain and tries to summarize information by using receptive fields. Put simply, receptive fields consist of a small set of conditions and provide stronger output / are responsive whenever a stimulus matches these conditions. They are locally applied on the input like a moving window and produce a new layer called feature map. One example would be an edge detector in image recognition, which is resembled by a receptive field which responds more strongly whenever one side of the receptive field receives dark (e.g., represented as "-1") and the other side receives light ("+1") information.

Example: CIFAR-10

This dataset stems from a typical computer vision problem: to identify what is on the picture. It consists of 60,000 photos of 32 x 32 pixels. There are 10 different classes such as cars, birds, or airplanes, which the neural network is supposed to be able to categorize after learning from the pictures. 50,000 of the images are used for the training set in the present example, while 10,000 images are used to evaluate the overall performance. A CNN is used to catch the underlying features that make up the uniqueness of the image categories.

The researchers came up with the following structure to learn about the data:

1. Convolutional input layer, 32 feature maps with a size of 3×3, a rectifier activation function and a weight constraint of max norm set to 3.
2. Dropout set to 20%.
3. Convolutional layer, 32 feature maps with a size of 3×3, a rectifier activation function and a weight constraint of max norm set to 3.
4. Max Pool layer with size 2×2.
5. Flatten layer.
6. Fully connected layer with 512 units and a rectifier activation function.
7. Dropout set to 50%.
8. Fully connected output layer with 10 units and a softmax activation function.

A batch size of 32 was used and it was run for 25 epochs. The model contains 4,210,090 trainable parameters and the accuracy achieved was 70.85% (note that they improved it to 80.18% in a very extensive version later).

For the sake of the assignment we were interested in scaling down the problem to fewer images and see whether we can achieve something close to the accuracy above. Thus, we reduced the training set to 25,000

images by random permutation so that we ended up with 2,500 randomly chosen images per category. This should give us insight into the importance of large sample sizes for such problems (which is an issue we will face later in our DL challenge).

Result

Probing different learning rates and batch sizes (finally, we sticked with the original), the reduced dataset achieved a validation set accuracy of 55.32%, which is, on the one hand, not a good result when trying to use the model in practice. On the other hand, it still does relatively well in predicting the 10 categories given that we took away half of the information. Indeed, one should expect that a lot of examples are necessary for complex recognition tasks such as this one, and it is definitely worth the effort to obtain a large dataset when trying to achieve a high accuracy.

Autoencoders

The examples above are for supervised learning. Autoencoders instead try to learn "themselves" about the structure of the input given. When given an input vector x, the goal is to return an output vector $\hat{x}$ that resembles x as closely as possible. The identity function that was created in such a way is what the experimenter is interested in, as it might reveal new ways of summarizing the data at hand (i.e., dimensionality reduction) or reveal hidden structures in the data (i.e., data denoising). This is done by compression through forcing the network to represent a smaller and as-similar-as-possible version of the original by fewer hidden nodes than what would be needed to just push through all the information. For instance, pictures of the size 10 x 10 pixels might be compressed to 7 x 7 pixel versions. If the restructured 10 x 10 version $\hat{x}$ is close enough to $\hat{x}$ the autoencoder successfully compressed the data. A related method is *Principal Component Analysis (PCA)*, which goal is to project higher-dimensional objects into a lower-dimensional space, so that most of its information is preserved.

Example: CIFAR-10 Convolutional Autoencoder

Before we handled the CIFAR-10 problem with a convolutional net and achieved rather mediocre accuracy with our reduced dataset. We could also be interested in compressing the data using autoencoders to get a better idea about the underlying properties of the categories. This could aid future approaches to improve on the accuracy concerning the problem by giving a sense for the complexity of the data.

For that we used existing code for the MNIST handwritten digts dataset and applied it to the CIFAR-10 problem, experimenting with batch size, learning rate, and epochs.

Result

Recurrent

Recurrent Neural Networks (RNNs), in contrast to *Feedforward Neural Networks*, extend the network by including cyclic aspects between the units. Thus, instead of units depending statically on the other layers, elements inside the network may be allowed to change dynamically, so that activations over time can alter the expression of a layer.

This enables us to make use of sequential information, i.e. we do not have to assume that all inputs (and outputs) are independent of each other. We can use previous computations to alter the network on how to process new incoming information; in other words the network remembers what has happened just before. Good examples for the application of RNNs lie in text processing or *Neuro-Linguistic Programming (NLP)*. An sometimes used architecture is *Long Short-Term Memory (LSTM)*, which constructs "smart neurons" that, depending on the input, keep the old information intact or overwrite it with the new information.

Example using LSTM: the IMDB movie review sentiment classification problem

This NLP problem comes with a dataset containing movie reviews of users from IMDB.com. The goal is to find out whether they are positive or negative reviews; in other words, we need to construct a neural net which uses the aspects of a review that hint towards one or the other and learn heuristics that enable it to make an accurate classification. A trick that is being used here is Embedding, which maps each word onto an length n vector (here $n = 32$) that is supposed to resemble the unique position of a word in a n-dimensional space. (Note that the original dataset transformed the words into sorted integers from most to least common word, however the same mechanism applies). To simplify the problem, the authors decided to limit the data by using only the most frequent 5000 words and truncating or lengthening (concatenating 0s) the reviews so that their length was always 500 words.

The researchers decided on the following structure:

1. Embedded layer: uses length 32 vector to represent each word. (1.1.) Dropout: 0.2
2. LSTM layer: 100 "smart neurons" (2.1.) Dropout: 0.2
3. Dense output layer: single neuron with sigmoid activation function for good/bad classification

The bracketed layers were investigated in order to see whether dropout layers might solve overfitting issues in the model. Dropout prevents units in the neural net to adapt too much to another unit by randomly dropping them during training. A batch size of 64 reviews were used and only 3 epochs were run because of overfitting concerns. When running the model, it achieved a 86.79% validation set accuracy without, and 86.36% with dropout.

We were wondering what's the specific effect of the LSTM smart neurons on the test set accuracy and ran the model several times with differing amounts to see whether it provides any insights. Taking more than in the example greatly increased the computation time, therefore we focused on reducing the amount of smart neurons and see how much are necessary to achieve acceptable results.

Result

Table 1: Recurrent Network: Smart Neurons Necessary to Achieve Good Predictions

Neurons	Test.Accuracy
10	78.64%
20	85.82%
50	87.87%
100	86.79%

The table indicates that best accuracy is actually being achieved with 50 smart neurons instead of 100. Our first intuition is that it might be a result of overfitting in the 100 neuron case. Other reasons we came up with are that (1) more epochs might be necessary to fit appropriately on the larger set, or (2) the batch size should be increased to reduce variance in the weight updates to counteract increased noise.

2. DL Challenge

References

- http://machinelearningmastery.com/object-recognition-convolutional-neural-networks-keras-deep-learning-library/
- http://neuralnetworksanddeeplearning.com/chap6.html
- https://blog.keras.io/building-autoencoders-in-keras.html

- http://machinelearningmastery.com/predict-sentiment-movie-reviews-using-deep-learning/
- http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/