Quick, Draw! Image Classification

Group 3

December 7, 2018

1 Introduction

Many recent industry innovations have included neural networks. An important research area within neural networks has been image classification. One of the best techniques to classify images is Convolutional Neural Networks (CNN). We have sought to better understand industry best practices by exploring different image classification techniques with neural networks. Our exploration starts by manually performing image classification and noting the level of human accuracy. We then use a fully connected neural network to perform image classification before implementing a CNN. We implemented options for tuning and debugging the neural network while implementing each method during our process. We were able to develop an intuition for image classification best practices using neural networks after our research.

Many supervised machine learning methods often differ from those employed when constructing neural networks. These supervised machine learning methods, such as logistic regression or decision tree classification, require feature engineering to successfully classify an image. Models requiring feature engineering rely on considerable effort by subject matter experts to achieve reasonable accuracy. Neural networks allow for an approach which does not require feature engineering to achieve similar or better accuracy. Industry engineering efforts regarding feature engineering can be shifted to neural networks in some cases such as image classification. Moreover, the availability of cheap computing resources sped up a shift towards the use of neural networks.

When considering modeling approaches, we examine both fully connected and convolutional neural networks. A fully connected neural network is known to be inefficient at classifying large images. These inefficiencies in a fully connected neural network partly arise because every neuron in the $\ell - 1$ layer must be connected to the ℓth layer. This requires a series of matrix multiplications and additions across each layer ℓ. Alternatively, a CNN allows for more efficient computations by: (1) efficient and automatic feature generation using local connectivity and parameter sharing of convolution operations, (2) dimensionality reduction using pooling layers. In this project, we attempt to understand these topics in depth from an empirical stanpoint in relation to image classification using Google's "Quick, Draw!" dataset.

2 Methods

Our methodology consists of reviewing statistical methodology, presenting theoretically informed hypotheses, and providing an overview of our data.

2.1 Overview of Statistical Modeling Techniques

We use human classification, fully connected neural networks, and a Convolutional Neural Network (CNN). We discuss the CNN in more detail than was covered in class so an overview is provided here.

2.1.1 Convolutional Neural Network (CNN)

TODO: explain how we used a CNN or anything else not covered in class

2.2 Hypotheses

We have generated the following hypotheses based on theoretical expectations:

2.2.1 Addressing model component of performance

$H1_a$ Fully Connected Neural Network will struggle to get reasonable accuracy.

$H2_a$ CNN should perform better than fully connected neural network.

2.2.2 Addressing data component of performance

$H3_a$ Small amounts of data will lead to overfitting.

$H4_a$ Imbalanced class prediction can be improved using data augmentation.

$H5_a$ Addition of a category to the trained model will reduce the performance of the model.

$H6_a$ Adding more training data will improve the performance of the model.

TODO: more context would be helpful here, maybe some citations

2.3 Data

Google originally collected the data from users by explicitly asking users to draw objects like forks, hotdogs, etc. The original dataset is 70 GB in size but Google also provides a simplified version of the dataset- consisting only of the final images. We used these '.npy' files to train all of our models. These numpy bitmap files consist of more than 100,000 rows each and 784 features, since they are basically 28×28 images. Samples of these images are shown in Figure 1. The subset of this dataset sampled varies from experiment to experiment. More details can be found in the experiments mentioned below.

Figure 1: Sample images from Quick, Draw! dataset

3 Results

We report our results by examining the validity of each hypothesis.

3.1 Addressing model component of performance

In order to perform this experiment we built a fully connected neural network using tensorflow. Our dataset sample selected for this experiment had 10 categories: fish, fork, hotdog, flamingo, airplane, alarm clock, baseball bat, bicycle, dolphin, elephant. We had 100,000 examples in total, with 10,000 example for each of the aforementioned categories. Our train:test split was chosen to be 80:20, an industry standard.

Our initial objective was to build a model that overfit our given dataset, with the idea that once our model is complex enough to overfit the training data, we could regularize it in order to achieve a more optimal fit.

After experimenting with various architectures, we narrowed down to a 3 layer model as specified in Figure 2.

The training set accuracy observed was 93%, while the test accuracy was at 73%, which clearly indicated an overfit in the model, as designed.

The model took 1500 epochs to converge taking around 45 minutes on a machine with 13 GB of RAM, Intel Xeon CPU 2.3 GHz (1 core, 2 threads), 1xTesla K80 GPU having 2496 CUDA cores with 12 GB GDDR5 VRAM. In order to manage variance, we attempted regularizing

$$Z_1 = \underset{25\times784}{W_1} X + \underset{25\times1}{b_1} \Rightarrow A_1 = \text{ReLU}(Z_1)$$

$$Z_2 = \underset{12\times25}{W_2} A_1 + \underset{12\times1}{b_2} \Rightarrow A_2 = \text{ReLU}(Z_2)$$

$$Z_3 = \underset{10\times12}{W_3} A_2 + \underset{10\times1}{b_3} \Rightarrow \text{output layer} = \text{SoftMax}(Z_3)$$

$$(\underbrace{25 \times 784 + 25 \times 1}_{Z_1}) + (\underbrace{12 \times 25 + 12 \times 1}_{Z_2}) + (\underbrace{10 \times 12 + 10 \times 1}_{Z_3}) = 20,067 \text{ parameters}$$

Figure 2: Fully Connected Neural Network Architecture

28 x 28 x 1 [Input Layer]	3 x 3 x 4 [ReLU] [Convolution]	2 x 2 x 8 [ReLU] [Convolution]	64 [Relu] [Dense]	10 [Softmax] [Output Layer]

Figure 3: CNN Architecture

using dropouts. Unfortunately, that effort was futile. We then doubled the amount of data sampled to train the model, eventually helping us achieve training set accuracy of 86% and test set accuracy of 82%. Our learning rate was specified as 0.001.

While our initial hypothesis assumed that a regular fully connected network would struggle to achieve a reasonable accuracy, our experiment proved our hypothesis as invalid by achieving a commendable accuracy. Having said that, the model did take a longer to converge than expected.

As for the CNN, we started off with the same approach with attempting to overfit the model using convolution operations and a fully connected layer. Since our objective is to categorise 10 categories, softmax was chosen as the activation function for the final layer, while every other neuron had ReLU as the activation function owing to ReLU's faster convergence.

3.1.1 Understanding the Loss Function

Most modern neural networks are trained using maximum likelihood. This means that the cost function is the negative log-likelihood, also described as the cross-entropy between the training data and the model distribution [1]. In our use case, we are interested in correctly classifying 10 categories of drawings. Our output unit use a Softmax function in our output layer because they are most often used for classifiers representing the probability distribution over n different classes. Given features h, weights W, and bias b, a linear layer first predicts unnormalized log probabilities:

$$z = W^T h + b \tag{1}$$

where $z_i = \log \tilde{P}(y = i|x)$. The softmax function can then exponentiate and normalize z to obtain the desired $\hat{y}$. The softmax function is given by

$$\text{softmax}(z)_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}. \tag{2}$$

As a cost function, we wish to maximize $\log P(y = i; z) = \log \text{softmax}(z)_i$. We can do this formally as

$$\log \text{softmax}(z)_i = z_i - \log \sum_j \exp(z_j). \tag{3}$$

which gives us the intuition for a single example. Overall, we want to learn parameters that drive the softmax to predict the fraction of counts of each outcome observed in the training set [1]:

$$\text{softmax}(z(x; \theta))_i \approx \frac{\sum_{j=1}^{m} 1_{y^{(j)}=i, x^{(j)}=x}}{\sum_{j=1}^{m} 1_{x^{(j)}=x}}. \tag{4}$$

This is how the categorical_crossetropy(y_true, y_pred) function is constructed in Keras [2] as well as the softmax_cross_entropy_with_logits_v2(labels, logits) in Tensorflow [3].

3.1.2 Understanding the Optimization Technique

There are a number of different optimization methods such as Stochastic and Batch Gradient descent. In practice, we chose to use a faster variant of gradient descent called Adam. The Adam optimization is able to use an exponentially weighted average of the gradients, a technique called momentum, and then use that gradient to update your weights instead. We also want to slow down gradient descent in the horizontal direction, a technique called RMSprop), to reduce oscillations from batch gradient descent. The Adam optimization method is combining momentum and RMSprop to perform gradient descent more quickly [4].

Number of Trainable Parameters: 320,890

The number of clearly being much greater than the number of observations used to train the model, we need to reduce the number of parameters vis-a-vis number of observations. From the available strategies to do so, we incorporated MaxPooling [TODO: why?] post each convolution operation, in order to extract only the most significant features. As as a consequence, this reduced the volume of the output, giving the model a smaller memory footprint as well as a smaller number of trainable parameters.

The observable training accuracy is 0.90, whereas the validation accuracy is 0.89. The model converged after 10 epochs, which took 3 minutes.

Some of the other unsuccessful strategies used to decrease the number of parameters and to regularize the network were strides and dropouts which sacrificed way too much in the way of accuracy and thus had to be dropped. Since most of our drawings had most of

28 x 28 x 1 [Input Layer]	3 x 3 x 4 [ReLU] [Convolution] [MaxPool - 2x2]	2 x 2 x 8 [ReLU] [Convolution] [MaxPool - 2x2]	64 [Relu] [Dense]	10 [Softmax] [Output Layer]

Figure 4: CNN Architecture with Reduced Parameters

their information in the center of the image, padding, in theory wouldn't be of much help. However, actually implementing padding improved our accuracy by 1% at a cost of 6000 parameters.

Conclusion

A slight (5%) improvement in the accuracy of a CNN when compared to a fully connected network, keeping the number of parameters constant, was observed. However, the CNN happened to be around 13 times faster in practice. This delta in accuracy and speed is likely to be amplified as we scale the model up with respect to model complexity or data.

3.2 Addressing data component of performance

ok

4 Discussion

Talk about our final slide and tie the hypotheses together

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[2] François Chollet et al. Keras. https://keras.io, 2015.

[3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[4] Andrew Ng. Deep learning — coursera. https://www.coursera.org/specializations/deep-learning, 2018. (Accessed on 12/07/2018).