A deep learning library implemented using *NumPy*

M. Becker[a)]

(Dated: 13 September 2018)

We deploy a software library for developing deep learning models. Support is provided for dense, convolution, and max pooling layers with softmax and rectifier activation functions. Training is performed using backpropagation and AdaGrad optimization.

Keywords: deep convolutional net, MNIST, NumPy

TensorFlow provides support for a wide range of deep learning models, and a number of different loss functions and optimization routines. However, usage-without-thinking of TensorFlow prevents a detailed understanding of the underlying mechanisms of deep models. As an exercise, we implement a rudimentary software library using *NumPy* with support for deep convolutional networks.

I. ARCHITECTURE OF THE LIBRARY

A model class abstraction provides a place to hold layers. Available layers include dense, convolutional, and max pooling. A layer object is instantiated by providing a set of initial weights and biases, as well as an activation function. Some layers have additional parameters, like padding for max pooling. An optimizer object is instantiated with a model, a loss function, and an initial learning rate. The optimize() function performs forward and backward propagation on the batch, storing the weight and bias gradients in memory for adjusting the weights and biases. An additional set of utility functions supports batch sampling, activation functions, and padding.

II. THE LIBRARY APPLIED TO MNIST

To test the functionality of the library, we apply a network with a single hidden layer and a deep convolutional network to the task of classifying digit images from MNIST. The architecture of the deep model is shown below.

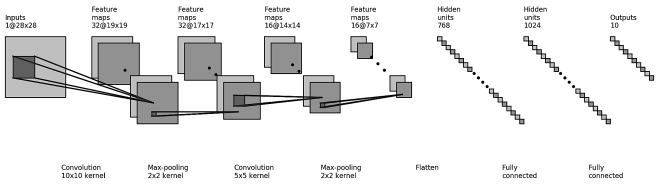

FIG. 1. The model architecture. ReLu activation on the convolutional layers and first dense layer. Softmax on the last. Cross-entropy loss.

The shallow network consists of a single hidden layer with 1024 neurons. We train the model over 10 epochs, with 200 training iterations and batches of size 32. The training loss and accuracy is shown below.

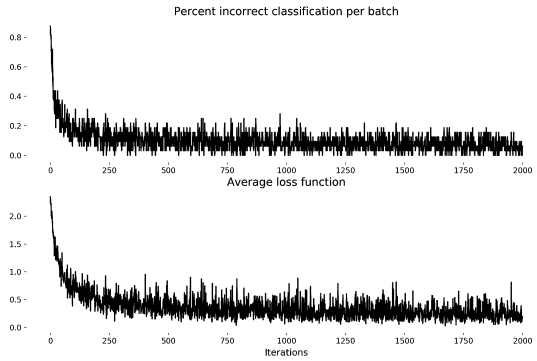

FIG. 2. Average loss and accuracy while training the shallow model. Accuracy on the test set approximately 93%.

For the deep network, we train the model over 10 epochs, with 200 training iterations and batches of size 16. In training, we encountered a number of issues related to exploding gradients and ReLu death. Nonetheless, this did not drastically affect the outcome of training, only the training time.

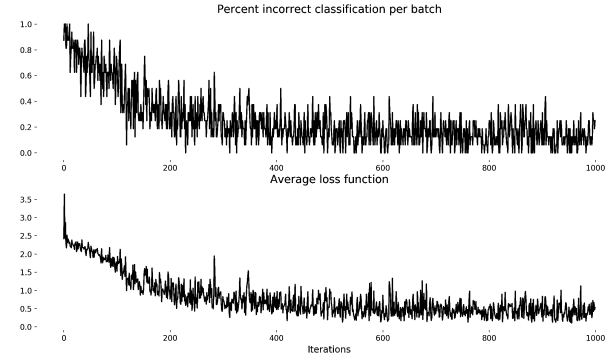

FIG. 3. Average loss and accuracy while training the deep model. Accuracy on the test set approximately 95%.

a)Electronic mail: mccoybecker@gmail.com