Imperial College
London

CBC: Neural Networks

Imperial College London

Department of Computing

Machine Learning

Author:
Dimitrogiannis Kontogouris, Michael Tarasiou (CID: 00650270)

Date: March 2, 2018

1 Linear and Relu layers

1.1 Linear forward pass

The linear forward function performs a linear transformation to the incoming data matrix $X \in \mathbb{R}^{N \times D}$, transforming it into a matrix $y \in \mathbb{R}^{N \times B}$ where B is the number of neurons in the next layer of the network. Mathematically this is shown below:

$$y = XW + b \tag{1}$$

The matrix b representing the biases of each neuron is a matrix of size $N \times B$. The rows are all the same, and in each column the bias of the corresponding neuron is stored.

1.2 Linear backward pass

For the backward pass, we are asked to return the derivatives of the loss function with respect to the weights and biases in the layer and the input X. We have:

$$dw = \frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \times \frac{\partial y}{\partial w} \tag{2}$$

$$\frac{\partial y}{\partial w} = \frac{\partial (XW + b)}{\partial w} = X$$

We know $\frac{\partial L}{\partial y}$ since it is a function input (dout) and thus we can calculate dw:

$$dw = \frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \times X = dout \times X \tag{3}$$

Similarly, the gradient with respect to the biases is:

$$db = \frac{\partial L}{\partial b} = \frac{\partial L}{\partial y} \times \frac{\partial y}{\partial b} = \frac{\partial L}{\partial y} \times 1 = \frac{\partial L}{\partial y} = dout$$

Finally, the gradient with respect to the input X:

$$dX = \frac{\partial L}{\partial X} = \frac{\partial L}{\partial y} \times \frac{\partial y}{\partial X} = \frac{\partial L}{\partial y} \times W = dout \times W \tag{4}$$

The dimensions of the gradients are the following:

- $dw \in \mathbb{R}^{D \times B}$
- $db \in \mathbb{R}^{N \times B}$
- $dX \in \mathbb{R}^{N \times D}$

1.3 Relu forward pass

The relu forward pass is simply defined as:

$$y = max(0, X) \tag{5}$$

The output of the relu forward pass is the input X, if X is greater than 0, otherwise it is 0. The dimensions of matrices y and X are the same $N \times D$ for some N and D.

1.4 Relu backward pass

The relu backward pass is simply 0 if X less than 0, otherwise it is equal to dout.

2 Dropout

Dropout is a method to prevent over-fitting in the neural network by ”disabling” neurons in the layers of the network. Each neuron has a probability p of being dropped.

2.1 Dropout forward pass

To perform the dropout forward pass, we create a matrix called *mask* which is of the same shape as the input X, Each element of *mask*, is a random draw from a Bernoulli distribution, with p probability of being 0, and $1-p$ of being 1. The output of the layer is an element-wise multiplication of matrix *mask* and the input X, and thus the result is a matrix equal to X, with some of its elements being equal to 0. In addition, the resulting matrix is scaled with a factor of $1/(1-p)$ since we are performing inverted dropout. This is only done during the training of the network.

2.2 Dropout backward pass

For the backward pass, we perform the same operation as for the forward pass in the opposite direction. We multiply the scaling factor, with the mask (the same as before!) and with the derivatives from previous layers which is an input in the functions. This is during the training phase. In the testing phase, the derivatives simply pass through, without a mask.

3 Softmax

The output of the neural network produces a number for each of the classes we have in our classification problem. These numbers can be interpreted as un-normalized probabilities. Using a softmax function, we can normalize them, so that their sum adds up to 1. Mathematically the softmax function is defined as:

$$\sigma_j(y_i) = \frac{e^{y_i[j]}}{\sum_{k=1}^{D} e^{y_i[k]}} \tag{6}$$

where, D is the number of classes and $y_i \in \mathbb{R}^D$ is a vector with the outputs of the network for each class (of the i_{th} sample). For numerical stability, we multiply (6) both the numerator and denominator by a constant K which improves stability without changing the results.

3.1 Gradient of softmax

MIKE

4 Fully connected Network

4.1 Architecture

4.2 L2-regularization

4.3 Overfitting

For the overfitting task we selected the top 50 images and label from CIFAR10 and used just those to fit a model. Since the objective was to fit the training data as good as possible with no consideration of generalization capabilities of our network we had the following strategy:

- use a network with many nodes in the hidden layers. We selected a network with two hidden layers with 1024 and 512 nodes respectively

- we did not apply any type of regularization as this would penalize the complexity of the model

The selected model achieves 100% accuracy over the training set at the sixth epoch as can be shown in Fig.1 below.

To get 50% accuracy over the validation set, we selected a network with two hidden layers, 512 and 256 nodes per respective layer. We didn't use dropout, but we did use a 0.2 L2 regularization multiplier. The model achieved > 50% over the validation set after 4 epochs as can be seen in Fig.2 below.

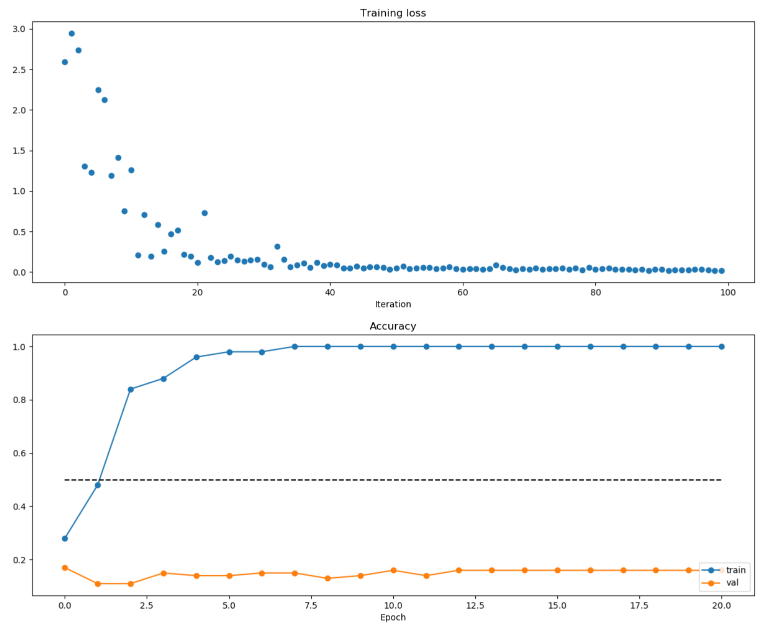

Figure 1: Network overfitting a subset of CIFAR10 dataset

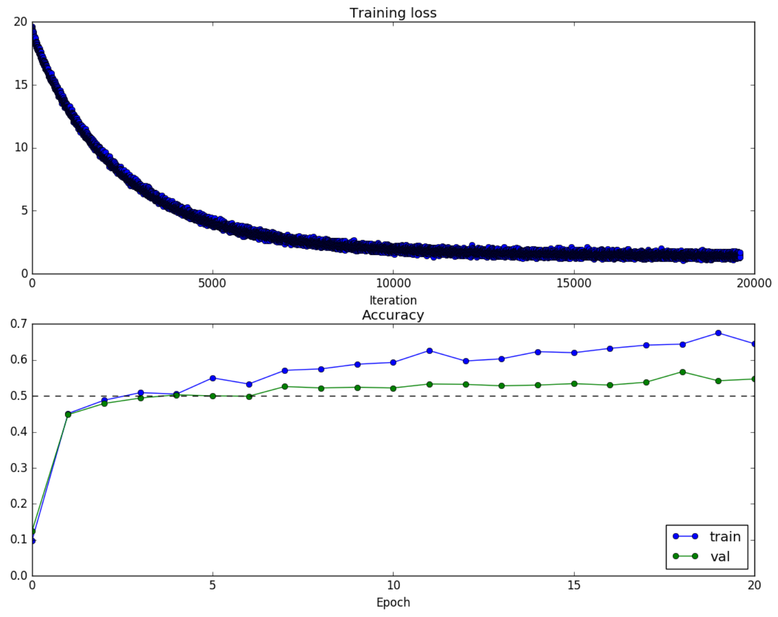

Figure 2: Network achieving > 50% validation accuracy in CIFAR10 dataset

4.4 Question 5

Select a reasonable network architecture as starting point and explain the motivation to choose so. Define a stopping criterion and a momentum and learning rate update schedule. You needto apply the stopping criterion to the solver (src/utils/solver.py). Optimise the learning rate (disable regularisation). Explain how you found a good initial valuefor learning rate. Include the plot for training and validation loss and report training and vali-dation classification error. Use dropout and report if there is any improvement in the validation performance. You have implemented dropout and L2 as ways of regularisation. Use L2 and compare the performance with dropout. Optimise the topology of the network, i.e. the number of hidden layers and the number of neurons in each hidden layer.

4.5 Question 6

For this task we used Tensorflow and built three different models. The first model we built is ha sthe following layers:

- input layer of dimensions Nx48x48x1 (batch size x image width x image height x number of channels)
- convolutional layer 1, 16x5x5x1 filters (number of filters x kernel width x kernel height x number of channels) at stride 1 and 2 zero padding at all dimensions, followed by ReLu activation. The 5x5 filters were chosen as a commonly used filter type and the stride and padding were selected so that the convolution layer does not modify the size of the features
- max pool layer of size 2x2 (width x height) at stride 2 that performs dimensionality reduction. After this step the each feature's dimensions are halved leading to a tensor of dimensions Nx24x24x16 (batch size x image width x image height x number of features)
- convolutional layer 2, 32x5x5x16 filters (number of filters x kernel width x kernel height x number of channels) at stride 1 and 2 zero padding at all dimensions, followed by ReLu activation. The 5x5 filters were chosen as a commonly used filter type and the stride and padding were selected so that the convolution layer does not modify the size of the features
- max pool layer of size 2x2 (width x height) at stride 2 that performs dimensionality reduction. After this step the each feature's dimensions are halved leading to a tensor of dimensions Nx12x12x32 (batch size x image width x image height x number of features). This layer is reshaped to a vector of dimension 4608 and dropout is applied
- fully connected layer with 1024 nodes and dropout
- output layer of dimension 7 as this is the number of classes we want to predict

The other two networks we used are variants of ResNet presnted in [1], [2], specifically the 50-layer and the 101-layer bottleneck networks. Residual networks are one of the most significant advances in terms of convolutional neural network architecture over the last decade. They use what the authors call skip connections between layers that pass identity mappings between the layers they connect. This is shown in Fig.3 below. As a result the convolutional layers between skip connections have to model the residual function which is an easier task for the model. This architecture option results in more efficient backpropagation and the capacity to train far deeper models that was though achievable before.

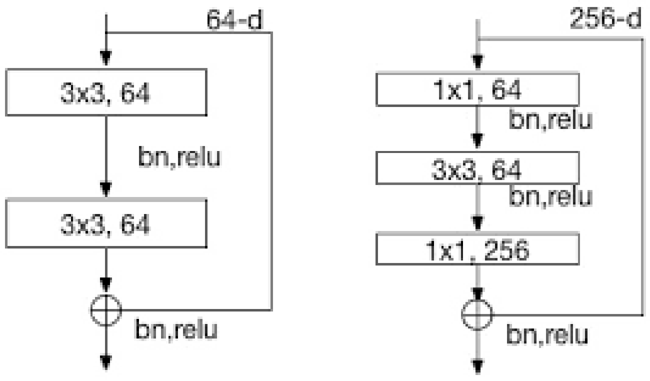

Figure 3: Residual networks building blocks. taken from [1]

5 A1

We could not claim one to be a better algorithm than the other in general for the following reasons stemming from the fact that the two methods are very different qualitatively:

- neural networks can be seen as black box function approximators while trees are highly interpretable. In cases where interpretability is crucial we could not use neural networks
- fitting and doing inference with the two models scales differently depending on the software and hardware architecture used. The computing architecture of today might allow one model to be run more efficiently and outperform the other but that will not necessarily hold for tommorow's architectures

Finally, the no free lunch theorem of [3] states that no apriori distinctions can be made between any two learning algorithms. that is to say that while the inductive principle used by one model might allow it to overperform another at a given problem, when examined under the scope of all possible problems, any two learning algorithms are equivalent

6 A2

For the classifcation trees, we will need to retrain the whole model as the addition of one class, because the space previously occupied by some classes wil need to be assigned to the new class.

For the case of neural networks a step we would definitely need to take is to add another node at the output layer of th enetwork allowing the classification of $n+1$ classes where the previous network was capable of classifying n classes. We will need to initialize the wieghts and bias connecting the new node with the last layer and we will need to train at least the last layer of the network. most possibly we will need to retrain the other layers but the previously trained network will provide a good initialization, so we will not need to start from scratch.

7 Instructions for executing

All code was written in Python3. All scripts were tested and run in the DOC Lab environment. To run the submitted scripts open a terminal window in the project directory and follow the instructions below.

8 References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun
Deep Residual Learning for Image Recognition. arXiv:1512.03385

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun
Identity Mappings in Deep Residual Networks. arXiv: 1603.05027

[3] Wolpert, David (1996), "The Lack of A Priori Distinctions between Learning Algor