Introduction to Deep Learning 2016-2017

Instructor: Iasonas Kokkinos, i.kokkinos-at-ucl.ac.uk

Assignment 3: Back-Propagation (2 Points/20) Deliverable: 5/3/17

1 Training Neural Networks

During this assignment you will experiment with training a neural network for classification.

You are provided with an implementation of the back-propagation algorithm for a neural network. [1] The code provided to you implements several crucial functionalities for neural network training:

- `nnet.m` implements the forward and backward pass of the back-propagation algorithm for a layer of an arbitrary number of nodes and layers.
- `checkNNgradients.m` implements numerical differentiation to verify that the analytical estimates of your derivatives are correct.
- `fmincg.m` is an implementation of conjugate-gradients for the minimization of nonlinear objective functions.

We are providing you with a running system. Your coding load is particularily light for this assignment. But you will need to understand what is happenning within the provided code - this will help you understand back-propagation without having to code every detail of it.

(0.5) Understanding the code:
Associate each of the variables below with symbols in the slides.
`d_output_d_activation,d_loss_d_activation,d_loss_d_output`
Why are we doing this?:

```
2*lambda * Weights{l}(:, 2:end);
```

and in particular why do we start from 2? (Attention: we did not talk about this in class - you will need to think about it!) (reply in no more than 3 lines of text)

Why are we doing this?:

```
d_loss_d_output_below  = Weights{l}'  *  d_loss_d_activation;
```

Which lecture slide does it correspond to, and why? (reply in no more than 3 lines of text)

(0.5) Extension: using softmax
The algorithm provided to you trains the top-layer neurons in a way that does not match with what we described in class. In particular, each top-layer neuron provides a score, that is trained with a one-vs-all objective.

[1] Starting point is here: `https://github.com/everpeace/ml-class-assignments/tree/master/ex4.Neural_Network_Learning`

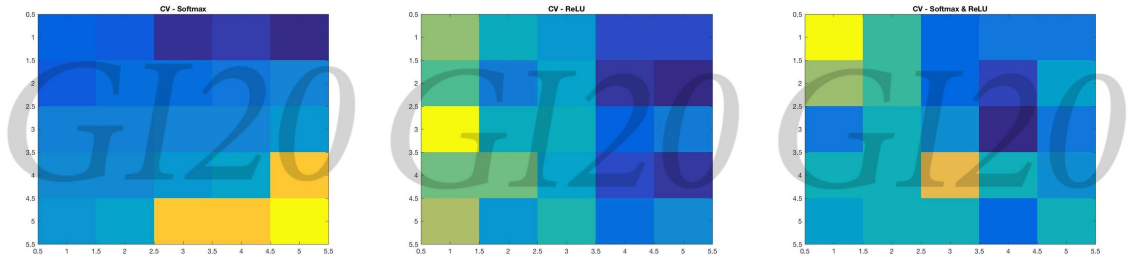

Figure 1: Cross-validation error for softmax (left), relu (middle) and softmax-relu (right) combinations.

In particular if we denote by $b_c(W)$ the score of the c-th class discriminant, as computed by the present algorithm, the data term of our present implementation would be:

$$L(W) = -\sum_{i=1}^{N}\sum_{c=1}^{C}\left[y_c^i \log(P_c) + +(1-y_c^i)\log(1-P_c)\right], \quad P_c = \frac{1}{1+\exp(-b_c(W))}, \tag{1}$$

Instead, during the class we described the more common softmax-based objective:

$$L(W) = -\sum_{i=1}^{N}\sum_{c=1}^{C}\left[y_c^i \log(P_c)\right], \quad P_c = \frac{\exp(b_c(W))}{\sum_{c=1}^{C}\exp(b_c(W))}, \tag{2}$$

Modify the existing code so as to incorporate this loss function. Verify that the gradients you are computing are correct using the checkNNgradients.m function. This is approximating numerically the value of your function's gradient and comparing it to your analytically-obtained gradient. If your code is correct, you should have the error in the order of 1e-9, or smaller.

(0.5) Extension: Using ReLUs
Modify the existing code so that rather than the sigmoidal activation function you use the Rectified Linear Unity (ReLU) function. Verify again the your code is giving the correct value of the gradient - as above.

(0.5) :Experiment
Cross-validate over λ (regularization cost) and number of intermediate neurons N.

Use the following values:

```
N_range = [5 10 20 50 100];
lambda_range  = linspace(0.00001,0.0005,N_lambdas);
```

Use the same cross-validation procedure as the one used in the previous computational assignments.

If your code is working properly you should be getting a cross-validation score similar to the one in the Fig. 1. Some minor deviations from these figures may be possible due to details that are specific to your computer's architecture. (but the overall form of your computed function needs to be the same).