Practical work 03 – 4/10/2018
Shallow Networks

Objectives

The main objectives of this Practical Work for Week 3 are the following :

a) Implement MBGD and Softmax and and learn what it means to choose hyper-parameters such as learning rate, batch size or number of epochs.

b) Deepen your understanding of the Universal Representation Theorem.

c) Further deepen your skills in python and numpy.

Submission

— **Deadline** : Wednesday 17 October, 12am

— **Format** :

— Exercise 1 (MBGD and Softmax)

— iPython notebook `MNIST_softmax_stud.ipynb` completed with your solutions.

— Small pdf-report with plots and the answers to the questions.

— Exercise 2 (Universal Representation Theorem) :

— pdf with your calculation (handwritten) of the gradient of the MSE cost.

— iPython notebook `universal_representation_learning_stud.ipynb` completed with your solutions.

— Small pdf report with your findings.

Exercice 1 MBGD and Softmax

Implement Batch Gradient Descent and Mini-Batch Gradient Descent for Softmax. Do this on the basis of the iPython notebook `MNIST_softmax_stud.ipynb`. As in PW 02, do this by only using numpy functionality (scikit learn used only for loading the data and splitting it into train and test sets). Look at the suitably marked sections that you need to implement.

Note that you will train softmax for the original MNIST. Since this will take more CPU and RAM, you need to be more careful in efficiently implementing the code. Make sure to properly use numpy array arithmetics ! All the training runs should complete in at most a couple of minutes.

Proceed as follows :

(a) Implement the update rules for a softmax layer - both, for BGD and MBGD. As in PW 02, keep an eye on the shapes of the numpy arrays defined in the input and to be provided as output.

(b) Plot the learning curves when optimising with the cross-entropy cost : Cost, Error Rate, Learning Speed - both for the training dataset and the test dataset and both for BGD and MBGD. Play with the hyper-parameters

- learning rate
- number of epochs
- batch size (for MBGD)

Specify your choice of these parameters and justify why your choice is best suited.

Exercice 2 Learning Function Representation

In this exercise, you train a single (hidden) layer perceptron to represent a given function $f : [0, 1] \to \mathbb{R}$.

The MSE cost for a perceptron with a single 1d input x, a single hidden layer with n units and a linear output layer is given by

$$J_{\mathrm{MSE}}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(y^{(i)} - (\sum_{k=1}^{n} w_{2,k} \sigma(w_{1,k} \cdot x^{(i)} + b_{1,k}) + b_2) \right)^2 \tag{1}$$

The dataset is given by suitable x-values and associated function values $f(x)$, i.e. $\{(x^{(i)}, y^{(i)} = f(x^{(i)}))|i = 1, \ldots, m\}$.

a) Compute the formulas for gradient descent for this problem, i.e. compute the derivatives w.r.t. parameters $w_{1,k}, w_{2,k}, b_{1,k}, b_2$ and formulate the according update rules.

b) Implement mini-batch gradient descent for this model by using the iPython notebook `universal_representation_learning_stud.ipynb`. Apply input and output normalisation. With the settings provided in the notebook (learning rate, batchsize, etc.) and the given dataset generated for the Beta-function (with $\alpha = \beta = 2.0$ and $m = 1000$ samples) the learning should work quite well - see the plot in the section 'Check the Trained Model' and the final MSE cost ($J_{\mathrm{MSE}}(\theta_{\mathrm{trained}}) \approx 4 \times 10^{-4}$).

c) Now study the impact of input and output normalisation : Do the training without input or output normalisation or without both. Is it still properly learning ? Explain !

d) Now study the impact of different settings by looking at the learning curves (as a diagnostic tool) and the cost (obtained at the end of the last epoch) as performance measure. Consider :

   - number of epochs : How many epochs are needed to see a reasonable fit ?
   - learning rate : How large can you choose the learning rate ?
   - number of neurons : For given
   - batch size : What happens when you increase the batch size ?

   Can you improve the approximation as compared to the initial settings ?

e) (**Optional**) Now study different functions by generating new data with a different underlying function. Try e.g. the sine function with different frequencies. Start with a frequency of $\omega = 1$ then proceed and investigate to how many frequencies you can go (the frequency ω corresponds to the number of periods the function describes in the interval $[0, 1]$. Possibly, also consider generating a larger dataset. Try to give an interpretation for why the learning breaks down at larger frequencies.

Exercice 3 Optional : Universal Representation Theorem

Study the material on http ://neuralnetworksanddeeplearning.com/chap4.html with "A visual proof that neural nets can compute any function" (the section in the lecture on the universal representation theorem has been largely inspired by this).

Implement the examples presented there. Specifically, implement an approximation of the the tower function, i.e.

$$f(x_1, x_2) = \begin{cases} 1 & (0 \leq x1 \leq 1) \\ 0 & (\text{otherwise}) \end{cases} \tag{2}$$

Can you construct it with a single hidden layer and just a *linear* output layer ?

Exercice 4 Optional : Review Questions

a) What is the purpose of the softmax layer ? Where is it typically used ?

b) Why is softmax beneficial over solving m binary classification problems (one for each of the classes) ?

c) Mention indicators for the situation where the training has not yet converged.

d) What are the factors that drive the performance of MNIST classification with a single (hidden) layer perceptron.

e) What kind of mappings can be represented by neural networks with just linear activation functions?

f) Describe what statement is made by the Universal Approximation Theorem.

g) Describe an approximation scheme for a given $1d$ function.

h) Describe why the Universal Approximation Theorem is only of limited value in practice.