Practical task 14

Neural networks as a universal approximator

In this task you need to approximate the function $f(x) = x^3$ on the $[-3, 2]$ interval with a neural network.

1. Using the library `pybrain` define a neural architecture consisting from the following layers: *input layer (`pybrain.structure.LinearLayer`)* $\rightarrow$ *hidden layer (`pybrain.structure.TanhLayer`)* $\rightarrow$ *output layer (`pybrain.structure.LinearLayer`)*. Input and output layers must have dimensionality 1.
2. Prepare a train set (`pybrain.datasets.SupervisedDataSet`) from 2000 uniformly distributed points on the $[-3, 2]$ interval, and two test sets of 1000 points - one on the same interval and another on the $[-3, 3]$ interval.
3. Plot the mean squared error (`pybrain.tools.validation.ModuleValidator`) on both test sets depending on the number of neurons in the hidden layer (from 1 to 100 with 10 step). Use not less than 100 epochs during the training process, use the weight decay (L2) regularization with weight 10^{-5}.
   (a) What is the minimum number of neurons in hidden layer enough to approximate the given function on $[-3, 2]$ interval?
   (b) What is the difference between errors on the two test sets?
4. Plot on the same graph function $f(x)$ and its neural approximation varying the number of neurons in $1, 5, 10, 20, 50, 100, 200$ on the interval $[-3, 3]$.
5. Repeat 2-4 using a train set from the domain $[-3, 1] \cup [2, 3]$. Describe the difference in approximation quality of this model and the prevous one on unknown intervals.

Classification and representation learning with a perceptron

1. Load the MNIST dataset (files *mnist_ train* and *mnist_ test*) using `loadFromFile` method of `pybrain.datasets.ClassificationDataSet` class.
2. Using the `pybrain` library define a neural architecture for classification of written digits images, consisting from the following layers: *input layer (`pybrain.structure.LinearLayer`)* $\rightarrow$ *hidden layer (`pybrain.structure.SigmoidLayer`)* $\rightarrow$ *output layer (`pybrain.structure.SoftmaxLayer`)*. Each layer must have a full set of connections with neurons from a previous layer (`pybrain.structure.FullConnection`). Hidden layer must consist from not less than 30 neurons.
3. Learn the model using the `pybrain.supervised.trainers.BackpropTrainer` class, use not less than a 25 epochs and weight decay (L2) regularization with weight 10^{-3}.
4. For each neuron in the hidden layer visualize weights on its connections from the input layer as 28×28 image (this corresponds to a row in the weight matrix `FullConnection.params`). How one can interpret these images?
5. Implement a function that transfroms an input image into a set of activations of the hidden layer neurons. Use this function to transform both train and test sets into the new feature space.
   (a) Visualize the new feature representations for objects of any new classes as stacked matrix rows. Can you separate objects of these two classes by just examining such a visualization?

(b) Using PCA (`sklearn.decomposition.PCA`) project new feature representations of these objects on a plane, draw objects of different classes with different colours. Describe the class separation quality of this 2-dimensional projection. What could cause the imperfect distribution of classes?

(c) Compare classification accuracy on the test of the following approaches: 1) the trained neural network, 2) kNN classifier (with $k = 3$) using the original pixel intensities as input features, 3) the similar kNN classifier using the new feature representations (hidden neurons activations).

    i. Which approach is the best? How big is the gap between the best and the second best approaches and how can this be explained?

    ii. Does any of these algorithms have a practical advantage on this task of digits classification?

6. Comment on the new feature space. What are its advantages and shortcomings?