Multilayer Perceptrons and the Backpropagation Algorithm

Exercise T3.1: Forward and Back Propagation (tutorial)

(a) What is a feedforward multilayer perceptron (MLP)?

(b) Derive the forward and back propagation algorithm (backprop) for an MLP with L layers.

(c) Discuss the consequence of parameter space symmetries: (i) permutation of neuron indices within a layer, (ii) reversal of signs across consecutive layers.

(d) Summarize the error measures and output layers for regression and classification.

Exercise H3.1: Multilayer Perceptrons (MLP) (homework, 4 points)

For an MLP with input $x \in \mathbb{R}$ and 1 hidden layer and 1 output node. The input-output function can be computed as

$$y(x) = \sum_{i=1}^{N_{\text{hid}}} w_{1i}^{21} f(w_{i1}^{10} x - w_{i0}^{20}))$$

with output weights w_{1i}^{21} and parameters w_{i1}^{10} and w_{i0}^{20} for the i-th hidden unit. In this case, the output node has no bias.

(a) Create 50 independent MLPs with $N_{\text{hid}} = 10$ hidden units by sampling for each MLP a set of random parameters $\{w_{1i}^{21}, w_{i1}^{10}, w_{i0}^{20}\}, i = 1, ..., 10$.

- Use $f(\cdot) := \tanh(\cdot)$ as the transfer function.
- Use normally distributed $w_{1i}^{21} \sim \mathcal{N}(0, 1)$
- Use normally distributed $w_{i1}^{10} \sim \mathcal{N}(0, 2)$ and uniformly distributed $w_{i0}^{20} \sim \mathcal{U}(-2, 2)$.

(b) Plot the input-output functions (i.e. the response $y(x)$ vs. x) of these 50 MLPs for $x \in [-2, 2]$.

(c) Repeat this procedure using a different intialization scheme for the weights of the hidden neurons:
$w_{i1}^{10} \sim \mathcal{N}(0, 0.5)$. What difference can you observe?

(d) Compute the mean squared error (MSE) between each of these 2×50 (50 from each of the above two initialization procedures) input-output functions and the function $g(x) = -x$. For each of the two initialization procedures, which MLP approximates g best? Plot $y(x)$ vs. x for these two MLPs

Exercise H3.2: MLP Regression (homework, 7 points)

The task is to implement an MLP with one hidden layer and apply the backpropagation algorithm to learn its parameters for a regression task.

Training Data: The file `RegressionData.txt` from the ISIS platform contains a small training dataset $\{x^{(\alpha)}, y_T^{(\alpha)}\}$, $\alpha = 1, \ldots, p$ with $p = 10$. The input values $\{x^{(\alpha)}\}$ in the first column are random numbers drawn from a uniform distribution over the interval $[0, 1]$. The target values $\{y_T^{(\alpha)}\}$ were generated using the function $\sin\left(2\pi x^{(\alpha)}\right)$ and Gaussian noise with standard deviation $\sigma = 0.25$ was added [1].

(A) **Initialization:**

1. Construct the MLP using a single hidden layer with 3 hidden nodes ($N_1 = 3$) and an output layer with a single output neuron ($N_L = N_2 = 1$).
2. Use the $\tanh$ transfer function for the hidden neurons and the linear transfer function (i.e. the identity) for the output neuron.
3. Set the weights and biases to random values from the interval $[-0.5, 0.5]$.

(B) **Iterative learning:**

1. For each input value $x^{(\alpha)}$ of the training set, do:
   (a) **Forward Propagation:** Calculate the total input and activity of the hidden neurons and the output neuron.
   (b) Compute the **output error** $e^{(\alpha)}$ using the quadratic error cost function.
   (c) **Backpropagation:** Calculate the "local errors" δ_i^v for the output and the hidden layer for each training point.
   (d) Calculate the gradient of the error function w.r.t. the first and second layer weights $w_{ij}^{1\,0}$ and $w_{ij}^{2\,1}$ respectively [2].
2. Calculate the batch gradient in order to obtain the direction of the weight updates:

$$\Delta w_{ij}^{v'v} = -\frac{\partial E_{[\underline{\mathbf{w}}]}^T}{\partial w_{ij}^{v'v}} = -\frac{1}{p}\sum_{\alpha=1}^{p} \frac{\partial e_{[\underline{\mathbf{w}}]}^{(\alpha)}}{\partial w_{ij}^{v'v}}$$

   where $j = 0, \ldots, N_v$ and $i = 1, \ldots, N_{v'}$
3. **Weight update:** Use gradient descent with a fixed learning rate $\eta = 0.5$ to update the weights in each iteration according to

$$\underline{\mathbf{w}}^{(t+1)} = \underline{\mathbf{w}}^{(t)} + \eta \Delta \underline{\mathbf{w}}^{(t)}$$

(C) **Stopping criterion**:
Stop the iterative weight updates if the error E^T has converged, i.e. $|\Delta E^T|/E^T$ has fallen below some small value (e.g. 10^{-5}) or a maximum number of iterations $t_{max} = 3000$ has been reached.

Devliverables:

(a) (2 point) Plot the error E^T over the iterations.

[1] Normally, you would not have access to such information and your training algorithm will not make direct use of it how the data was generated.

[2] The weights in the second layer are those connecting the hidden neurons with the output node. Since we only have one output neuron, $w_{ij}^{2\,1}$ is effectively $w_{1j}^{2\,1}$

(b) (1 point) For the final network, plot the output of hidden units for all inputs.

(c) (1 point) Plot the output values over the input space (i.e. the input-output function of the network) together with the training dataset.

(d) (2 point) Plot (a)–(c) *twice* (i.e., for different initial conditions) next to each other and discuss: is there a difference, and if so, why?

(e) (1 point) What might have been the motivation for using a quadratic error function here?

Hints:

(1) Modularizing your code can help narrow down potetnial mistakes in your implementation. One way to go about it is to have separate functions for the forward propagation, calculating the error, calculating the different terms that make up the gradients (error funciton, non-linearity). This can help verifying the different parts of your code.

(2) Making use of broadcasting in numpy and turning everything into a matrix multiplication will reduce the number of lines in your code. Matrix multuplication can also speed up your training process. But some of these code optimizations could make your code less readable and harder to debug.

(3) Don't forget to update the bias weights in your network. Every hidden neuron and the output neuron has a bias weight.

(4) Keep in mind that the transfer function applied to the bias value is the identity function. Don't apply the non-linear transfer function to the bias nodes.

(5) A sudden "spike" in your error plot is ok.

Total 11 points.