Machine Learning
Exercise 5

Marc Toussaint
TAs: Janik Hager, Philipp Kratzer
Machine Learning & Robotics lab, U Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany

May 14, 2019

In these two exercises you'll program a NN from scratch, use neural random features for classification, and train it too. Don't use tensorflow yet, but the same language you used for standard regression & classification. Take slide 04:14 as reference for NN equations.

(DS BSc students may skip 2 b-c, i.e. should at least try to code/draft also the backward pass, but ok if no working solutions.)

1 Programming your own NN – NN initialization and neural random features (5 Points)

(Such an approach was (once) also known as Extreme Learning.)

A standard NN structure is described by $h_{0:L}$, which describes the dimensionality of the input (h_0), the dimensionality of all hidden layers ($h_{1:L\text{-}1}$), and the dimensionality of the output h_L.

a) Code a routine "forward(x, β)" that computes $f_\beta(x)$, the forward propagation of the network, for a NN with given structure $h_{0:L}$. Note that for each layer $l = 1, .., L$ we have parameters $W_l \in \mathbb{R}^{h_L \times h_{L\text{-}1}}$ and $b_l \in \mathbb{R}^{h_l}$. Use the leaky ReLu activation function. (2 P)

b) Write a method that initializes all weights such that for each neuron, the $z_i = 0$ hyperplane is located randomly, with random orientation and random offset (follow slide 04:21). Namely, choose each $W_{l,i\cdot}$ as Gaussian with sdv $1/\sqrt{h_{l\text{-}1}}$, and choose the biases $b_{l,i} \sim \mathcal{U}(-1, 1)$ uniform. (1 P)

c) Consider again the classification data set `data2Class.txt`, which we also used in the previous exercise. In each line it has a two-dimensional input and the output $y_i \in \{0, 1\}$.

Use your NN to map each input x to features $\phi(x) = x_{L\text{-}1}$, then use these features as input to logistic regression as done in the previous exercise. (Initialize a separate β and optimize by iterating Newton steps.)

First consider just $L = 2$ (just one hidden layer and $x_{L\text{-}1}$ are the features) and $h_1 = 300$. (2 P)

Extra) How does it perform if we initialize all $b_l = 0$? How would it perform if the input would be rescaled $x \leftarrow 10^5 x$? How does the performance vary with h_1 and with L?

2 Programming your own NN – Backprop & training (5 Points)

We now also train the network using backpropagation and hinge loss. We test again on `data2Class.txt`. As this is a binary classification problem we only need one output neuron $f_\beta(x)$. If $f_\beta(x) > 0$ we classify 1, otherwise we classify 0.

Reuse the "forward(x, β)" coded above.

a) Code a routine "backward(δ_{L+1}, x, w)", that performs the backpropagation steps and collects the gradients $\frac{d\ell}{dw_l}$.

For this, let us use a hinge loss. In the binary case (when you use only one output neuron), it is simplest to redefine $y \in \{-1, +1\}$, and define the hinge loss as $\ell(f, y) = \max(0, 1 - fy)$, which has the loss gradient $\delta_L = -y[1 - yf > 0]$ at the output neuron.

Run forward and backward propagation for each x, y in the dataset, and sum up the respective gradients. (2 P)

b) Code a routine which optimizes the parameters using gradient descent:

$$\forall_{l=1,..,L}: \; W_l \leftarrow W_l - \alpha \frac{d\ell}{dW_l} \;, \quad b_l \leftarrow b_l - \alpha \frac{d\ell}{db_l}$$

with step size $\alpha = .01$. Run until convergence (should take a few thousand steps). Print out the loss function ℓ at each 100th iteration, to verify that the parameter optimization is indeed decreasing the loss. (2 P)

c) Run for $h = (2, 20, 1)$ and visualize the prediction by plotting $\sigma(f_\beta(x))$ over a 2-dimensional grid. (1 P)