Machine Learning
Exercise 7

Marc Toussaint
Machine Learning & Robotics lab, U Stuttgart
Universitätsstraße 38, 70569 Stuttgart, Germany

June 3, 2016

1 Intro to Neural Networks

Read sections 11.1-11.4 of Hastie's book http://www-stat.stanford.edu/~tibs/ElemStatLearn/ As a success criterion for learning, check if you can explain the following to others:

Let h_0 denote the input dimentionality, and $h_{1,..,L}$ the number of neurons in L hidden layers. For all layers, let $z_l = W_{l\text{-}1} x_{l\text{-}1}$ denote the inputs to the neurons, and $x_l = \sigma(z_l)$ the activation of the neurons, $z_l, x_l \in \mathbb{R}^{h_l}$.[1] Let $x_0 \equiv x$ denote the data input, or activation of the input layer. Then an L-layer NN first computes (forward propagation)

$$\forall_{l=1,..,L}: \; z_l = W_{l\text{-}1} x_{l\text{-}1} \;, \quad x_l = \sigma(z_l) \;, \quad f \equiv z_{L+1} = W_L x_L \;, \tag{1}$$

where $w = (W_0, .., W_L)$, with $W_l \in \mathbb{R}^{h_{l+1} \times h_l}$, are the parameter matrices, and $f \in \mathbb{R}^{h_L}$ the function output.

Assume a loss function $\ell(f, y)$. For a specific y, denote by

$$\delta_{L+1} \triangleq \frac{\partial \ell}{\partial f} \tag{2}$$

the partial derivative (row vector!) of the loss w.r.t. the output function values. E.g., for squared error $\ell(f, y) = (f-y)^2$, $\delta_{L+1} = 2(f-y)^\top$. We have (backward propagation)

$$\forall_{l=L,..,1}: \; \delta_l \triangleq \frac{d\ell}{dz_l} = \frac{d\ell}{dz_{l+1}} \frac{\partial z_{l+1}}{\partial x_l} \frac{\partial x_l}{\partial z_l} = [\delta_{l+1} \; W_l] \circ [x_l \circ (1 - x_l)]^\top \;. \tag{3}$$

Note that $\circ$ is an *element-wise product* (that arises from the element-wise application of the activation function). Further, the brackets indicate which order of computation is efficient. Given these, the gradient w.r.t. the weights is

$$\frac{d\ell}{dW_{l,ij}} = \frac{d\ell}{dz_{l+1,i}} \frac{\partial z_{l+1,i}}{\partial W_{l,ij}} \quad \text{or} \quad \frac{d\ell}{dW_l} = \delta_{l+1}^\top x_l^\top \;. \tag{4}$$

2 Weigth Initialization in Neural Networks

In Glorot, Bengio: *Understanding the difficulty of training deep feedforward neural networks* (AISTATS'10) the issue of weight initialization in neural networks is discussed, which is super important. Often weights are initialized uniformly from some interval $[-a, a]$. The question is how to choose a.

Consider a single layer of a neural network

$$y = \sigma(Wx)$$

where $x \in \mathbb{R}^n$ is an n-dimensional input, $W \in \mathbb{R}^{h \times n}$ is an $h \times n$ matrix of weights, $\sigma(z) = \frac{1}{e^{-z}+1}$ is the sigmoid function, and $y \in \mathbb{R}^h$ the activation vector of the h neurons of the layer. Now assume that $\mathrm{E}\{x\} = 0$ and $\mathrm{Var}\{x\} = 1$, and each weight $W_{ij} \sim \mathcal{U}(-a, a)$ is uniformly samples from $[-a, a]$.

What we want is that $\mathrm{Var}\{y\} = 1$, that is, the layer's output y should have the same variance as the layer's input x. (Why, we'll discuss in the tutorial.)

How do you have to choose a so that $\mathrm{Var}\{y\} = 1$? For simplicity you may assume that $\sigma(z) \approx z$, that is, that there is no sigmoid (or that, for small weights, the sigmoid is approximately linear).

[1] The typical activation function σ is the *logistic sigmoid function* $\sigma(z) = \frac{e^z}{1+e^z} = \frac{1}{e^{-z}+1}$ with $\sigma'(z) = \sigma(z)(1-\sigma(z))$. Alternatives would be $\sigma(z) = \tanh(z)$ or (rare) $\sigma(z) = \frac{z}{1+|z|} \in [-1, 1]$ or newer (hinge loss like) versions.

3 Programming Backprop

Consider the classification data set "data2Class_adjusted.txt", which is a non-linear classification problem. It has a three-dimensional input, where the first is a constant bias input.

Train a NN for this classification problem. As this is a binary classification problem we only need one output neuron y. If $y > 0$ we classify $+1$, otherwise we classify -1.

- Code a routine "forward(x, w)" that computes $f(x, w)$, the forward propagation of the network, for a NN with a single hidden layer of size $h_1 = 100$. Note that we have 100×3 and 100×1 parameters respectively (remember to initialize them in a smart manner, as discussed in previous exercise).
- Code a routine "backward(δ_{L+1}, x, w)", that performs the backpropagation steps and collects the gradients $\frac{d\ell}{dw_l}$.
- Let us use a hinge-loss function $\ell(f, y) = \max(0, 1 - fy)$, which means that $\delta_{L+1} = -y[1 - yf > 0]$.
- Run forward and backward propagation for each x, y in the dataset, and sum up the respective gradients.
- Code a routine which optimizes the parameters using gradient descent: $W_l = W_l - \alpha \frac{d\ell}{dW_l}$ with step size $\alpha = .05$. Run until convergence (should take a few hundred steps).
- Print out the loss function ℓ at each iteration, to verify that the parameter optimization is indeed decreasing the loss.

a) Visualize the predicion by plotting $\sigma(f(x, w))$ over a 2-dimensional grid.

b) What happens if we initialize the weights with 0? Why?