Due Date : February 12th, 2018

Instructions

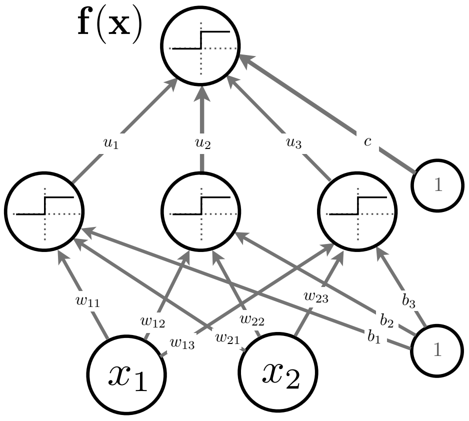
- *For all questions, show your work!*
- *Use a document preparation system such as LaTeX.*
- *Submit your answers electronically via the course studium page.*
- *Bonus questions are optional, but will fetch extra points.*

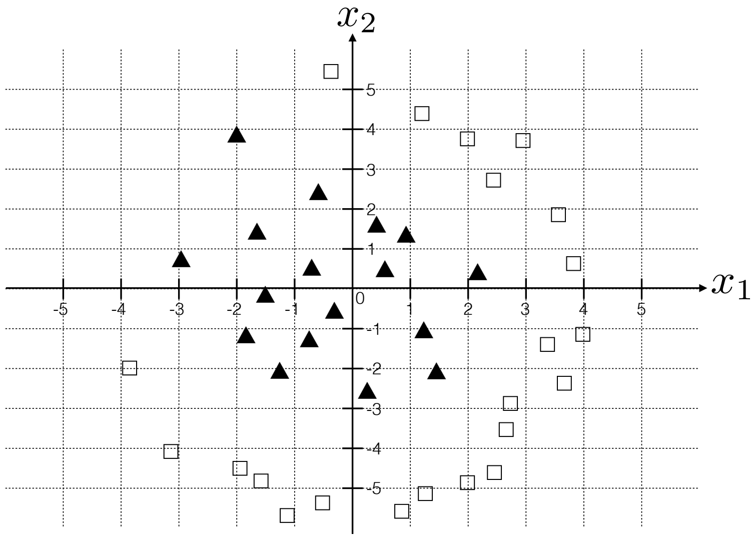
1. **(10 points) Neural Networks Classification**
Consider training a standard feed-forward neural network. For the purposes of this question we are interested in a single iteration of SGD on a single training example : $(\boldsymbol{x}, y)$. We denote $f(\boldsymbol{x}, \boldsymbol{\theta})$ as the output of the neural network with model parameters $\boldsymbol{\theta}$. Now let's say g is the output activation function and $a(\boldsymbol{x}, \boldsymbol{\theta})$ is the pre-activation network output such that $f(\boldsymbol{x}, \boldsymbol{\theta}) = g(a(\boldsymbol{x}, \boldsymbol{\theta}))$.
   (a) Assuming the network's goal is to do binary classification (with the detailed structure above), what would be an appropriate activation function for the output layer, i.e. what would be an appropriate function g.
   (b) What does the output represent under this activation function ?
   (c) Let $L_{CE}(f(\boldsymbol{x}, \boldsymbol{\theta}), y)$ be cross-entropy loss, express it as a function of $f(\boldsymbol{x}, \boldsymbol{\theta})$ and y.
   (d) Compute the partial derivative $\frac{\partial L_{CE}(f(\boldsymbol{x},\boldsymbol{\theta}),y)}{\partial a(\boldsymbol{x},\boldsymbol{\theta})}$.
   (e) Let $L_{MSE}(f(\boldsymbol{x}, \boldsymbol{\theta}), y)$ be the mean-squared error, express it as a function of $f(\boldsymbol{x}, \boldsymbol{\theta})$ and y.
   (f) Compute the partial derivative $\frac{\partial L_{MSE}(f(\boldsymbol{x},\boldsymbol{\theta}),y)}{\partial a(\boldsymbol{x},\boldsymbol{\theta})}$.
   (g) Based on your answers to the above questions, what would argue would be the more appropriate loss function for binary classification and why ?

2. **(10 points) Neural Network Representation**
Consider the binary classification problem described in the figure below (right), with the two classes being represented by solid triangles and empty squares. Let us refer to the squares as Class 0 and the triangles as Class 1. In this problem we will consider a classifier using the network in the figure (right). All activation functions in the hidden layer being Heavyside step functions :

$$H[x] = \begin{cases} 0, & x < 0, \\ 1, & x \geq 0, \end{cases} \tag{1}$$

Fill-in the values of the model parameters (w_{11}, w_{12}, w_{13}, w_{21}, w_{22}, w_{23}, u_1, u_2, u_3, b_1, b_2, b_3 and c) that result in the correct class prediction for the data presented in the figure.





3. **(10 point) Activation Functions.**
Using the definition of the derivative and the definition of the Heaviside step function :

$$\frac{d}{dx}f(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon}$$

$$H(x) = \begin{cases} 1 & \text{if } x > 0 \\ \frac{1}{2} & \text{if } x = 0 \\ 0 & \text{if } x < 0 \end{cases}$$

(a) Show that the derivative of the rectified linear unit ($relu(x) = max(0, x)$), where it exists, is given by the Heaviside step function. Where the derivative is not defined, assign a value to it.

(b) Give two alternative definitions of the $relu$ using $H(x)$.

(c) Give an asymptotic expression for $H(x)$ using the sigmoid function, $\sigma(x) = 1/(1 + e^{-x})$.

(d) Using the same technique as in a), show that the derivative of H(x) is given by the dirac delta function $\delta(x)$, defined in section 3.9.5 of the deep learning textbook.

4. **(10 point) Gradients and Networks.**

(a) Compute the jacobian matrix of the softmax function, $S(x_i) = \frac{e^{x_i}}{\sum_k e^{x_k}}$.

(b) Express it as a matrix equation.

(c) Compute the jacobian matrix of the logistic sigmoid function, applied element-wise to the vector x , $\sigma(x) = 1/(1 + e^{-x})$.

(d) Let y and x be vectors related by $y = f(x)$, L be an unspecified loss function. Let g_x and g_y be the gradient of L with respect to x and y, and let $J_y(x)$ be the jacobian of y with respect to x. From the Deep Learning textbook, Eq. 6.46 tells us that :

$$g_x = J_y(x) \cdot g_y$$

. Show that if $f(x) \equiv \sigma(x)$, the above can be rewritten as

$$g_x = g_y \odot \sigma'(x).$$

If $f(x) \equiv S(x)$, can g_x be defined the in the same way ?

5. **(10 point) Softmax activation function**

(a) Show that the softmax function is invariant under translation. In other words that

$$\text{softmax}(\boldsymbol{x} + c) = \text{softmax}(\boldsymbol{x})$$

where the softmax function is defined by

$$\text{softmax}(\boldsymbol{x}) = \left(\frac{e^{x_i}}{\sum_{k=1}^{|\boldsymbol{x}|} e^{x_k}} \right)_{i=1,\ldots,|\boldsymbol{x}|}$$

c is a constant added to all elements of $\boldsymbol{x}$ and $|\boldsymbol{x}|$ is the length of $\boldsymbol{x}$.

(b) It's obvious that the softmax is not invariant under multiplication. What is the effect when you multiply the vector by a scalar larger than 1 ? Smaller than 1 ? In terms of the output of the softmax as the certitude of the network in its prediction, what effect would this multiplication have.

(c) Show that a 2-class softmax function can be rewritten as sigmoid function. In other words that

$$\text{softmax}(\boldsymbol{x}) = \begin{pmatrix} \sigma(z) & 1 - \sigma(z) \end{pmatrix}^\top$$

where the softmax function is defined by

$$\text{softmax}(\boldsymbol{x}) = \begin{pmatrix} \frac{e^{x_1}}{e^{x_1}+e^{x_2}} & \frac{e^{x_2}}{e^{x_1}+e^{x_2}} \end{pmatrix}^\top$$

and the sigmoid function is defined by

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

(d) Use the previous result to show that it's possible to write a k-class softmax function as a function of $k - 1$ variables.

6. **(10 point) Using cross-entropy cost for real-valued data** Cross-entropy cost is a popular cost function used to estimate distributions over binary support, such as binarized MNIST. It's defined as :

$$ce(p, x) = -x \log(p) - (1 - x) \log(1 - p)$$

This derivation assumes that x is binary, i.e. $x \in \{0, 1\}$, such as binarized MNIST. However, the same cost function is often also used with real-valued $x \in (0, 1)$, such as real-valued MNIST without any binarization.

(a) Derive the cross-entropy cost function using the maximum likelihood principle for $x \in \{0, 1\}$.

(b) Suggest a probabilistic interpretation of the cross-entropy cost function when $x \in (0, 1)$. (HINT : KL divergence between two distributions)

7. **(20 point) Deriving the Glorot initialization scheme** The pre-2010 phase of deep learning research made extensive use of model pre-training, since training deep models was thought to be very hard. This changed in 2010 due in part to a paper by Xavier Glorot and Yoshua Bengio who showed that deep models can be trained by just ensuring good initializations. The key insight by Xavier was that a layer in a deep network should ensure that data passed through it maintains the same variance, since if it does not, deeper networks will have a multiplicative effect and change the variance even more.

In the following questions, we want the pre-activation of each layer (a_l) in the forward pass to have $\mathbb{E}[a_l] = \mathbf{0}$ and $\text{Var}(a_l) = \mathbf{1}$. We can assume the input is standardized. You may use the the following formula to compute the variance of product of independent random variables :

$$\text{Var}(XY) = \mathbb{E}[X^2Y^2] - (\mathbb{E}[XY])^2 = \text{Var}(X)\text{Var}(Y) + \text{Var}(X)(\mathbb{E}[Y])^2 + \text{Var}(Y)(\mathbb{E}[X])^2$$

(a) (15 points) Derive the Glorot initialization scheme for the weight W_l of the l'th hidden layer. Suppose the previous hidden layer has d_{l-1} neurons. Also suppose the neurons are all distributed by a standard normal.

(b) (5 points) Remove the last assumption of the last question. Now assume the pre-activations are all distributed by normal distributions. Derive an initialization scheme to satisfy the first and second moment constraints ($\mathbb{E}[a_l] = \mathbf{0}$ and $\text{Var}(a_l) = \mathbf{1}$) if the *relu* activation function is used.

8. **(BONUS, 10 point) Interpolation capacity of neural nets**
As seen in class, neural networks are known to be **universal approximators** : loosely speaking, under mild assumptions, any function between two finite-dimensional spaces can be approximated by a 2-layer neural network with arbitrary precision. The goal here is to illustrate the capacity of neural networks for the much simpler problem of **interpolation** of functions on finite samples. The problem is as follows : given $N \in \mathbb{N}^*$, (i) is there a two-layer network such that for any $f : \mathbb{R}^n \to \mathbb{R}^m$ and any sample $S \subset \mathbb{R}^n$ of size N, there is a set of weights such that the output $y(x)$ matches $f(x)$ for all $x \in S$? (ii) if there is, what is the minimum number of hidden units of such a network ?

We show below that any function can be interpolated on finite samples of size N by a 2-layer network with $(N - 1)$ hidden units, with linear output and both with RELU and sigmoid activations.

(a) Write the generic form of the function $y : \mathbb{R}^n \to \mathbb{R}^m$ defined by a 2-layer network with $N - 1$ hidden units, with linear output and activation function ϕ, in terms of its weights $(W^{(1)}, b^{(1)})$ and $(W^{(2)}, b^{(2)})$.
In what follow we will restrict $W^{(1)}$ to be $W^{(1)} = [w, \cdots, w]^T$ where $w \in \mathbb{R}^n$.

(b) Let $f : \mathbb{R}^n \to \mathbb{R}^m$; and pick a sample $S = \{x^{(1)}, \cdots x^{(N)}\} \subset \mathbb{R}^d$ of size N. Show that the interpolation problem on the sample S can be reduced to solving a matrix equation :

$$M\tilde{W}^{(2)} = F \qquad (2)$$

where $\tilde{W}^{(2)}$ and F are $N \times m$ matrices respectively given by

$$\tilde{W}^{(2)} = [W^{(2)}, b^{(2)}]^T, \qquad F = [f(x^{(1)}), \cdots f(x^{(N)})]^T \tag{3}$$

Write the $N \times N$ matrix M explicitly in terms of w, $b^{(1)}$, the activation ϕ and the sample elements $x^{(i)}$.

For the following subquestions, we may assume without loss of generality that $\langle w, x^{(1)} \rangle < \cdots < \langle w, x^{(N)} \rangle$. Note that triangular matrix with non-zero diagonal elements is sufficient for proving existence of the solution to a linear system, but ordering here does not matter, as permutation does not change the volume (determinant), so it is not necessary to assume the order.

(c) **Proof with Relu activation.** Note that the $x^{(i)}$ are all distinct. Choose ω such that the scalar products $\langle w, x^{(j)} \rangle$ are also all distinct Set $b_j^{(1)} = -\langle w, x^{(j)} \rangle + \epsilon$, where $\epsilon > 0$. Find a value of ϵ such that M is triangular with non-zero diagonal elements. Conclude.

(d) **Proof with sigmoid-like activations**. Here we consider sigmoid activations. (In fact we may only assume that ϕ is continuous, bounded, and that $\phi(-\infty) = 0$ and $\phi(0) > 0$.) Set $b_j^{(1)} = -\langle \lambda u, x^{(j)} \rangle$ and redifine $w = \lambda u$, where $\lambda > 0$. Show that $\lim_{\lambda \to +\infty} M$ is triangular with non-zero diagonal elements. Conclude.

9. **(BONUS, 10 point) Mixture Density Network (6.2.2.4)**
Consider a univariate-output Mixture Density Network, with n specifying the number of latent variables :

$$p(y|x) = \sum_{i=1}^{n} p(c = i|x) p(y|x; \theta_i) \tag{4}$$

where $\theta = \{\theta_i\}_{i=1:n}$ are the set of class conditional parameters. In the following questions, assume the "prior" probability distributions $p(c = i|x)$ form a multinoulli distribution, parameterized by a softmax function (a one hidden-layer network) mapping from the input, i.e.

$$p(c = i|x; \zeta) = s_i(x; \zeta) = \frac{\exp(\zeta_i^T x)}{\sum_{i'} \exp(\zeta_{i'}^T x)} \tag{5}$$

(a) Suppose Y is continuous, and let $p(y; \theta^{(i)}(x)) = \mathcal{N}(y; x^T \beta_i, \sigma_i^2)$. To do prediction, use the expected conditional as a point estimate of the output. Derive $\mathbf{E}[y|x]$ and $\mathbf{Var}[y|x]$.

(b) Holding the class conditional parameters (θ_i) fixed, derive a stochastic (i.e. for one data point) gradient ascent expression for the softmax weight parameters ζ_i using maximum likelihood principle. (Hint : M-step of the EM algorithm)

(c) Now devise a prediction mapping function $h : \mathcal{X} \to \mathcal{Y}$ defined as $h(x) = \sum_i^n \sigma_i(x) \mu_i(x)$, where generally $\sigma(\cdot)$ is a MLP and $\mu(\cdot)$ is a prediction function depending on the input x. Now let $\sigma(x)$ be a softmax regression of n classes and μ be a set of n linear mapping functions, i.e. $h(y|x) = \sum_i^n \{s_i(x; \zeta)(\sum_j^p x_j \beta_{ij})\}$. If we want to minimize the quadratic loss $l(y_n, x_n) = (y_n - h(y|x_n))^2$ for each data point n, what is the gradient descent update expression for parameter ζ_i if β is fixed ?

(d) Comment on the difference between the previous two training objectives.

10. **(BONUS, 10 point) Transformation of Activation Function.** Consider a two-layer neural network function of the form :

$$y_k(x,w) = \sigma(\sum_{j=1}^{M} \omega_{kj}^{(2)} g(\sum_{i=1}^{D} \omega_{ji}^{(1)} x_i + \omega_{j0}^{(1)}) + \omega_{k0}^{(2)}) \tag{6}$$

in which the hidden-unit nonlinear activation functions g(·) are given by logistic sigmoid functions of the form :

$$\sigma(a) = (1 + exp(-a))^{-1} \tag{7}$$

Show that there exists an equivalent network, which computes exactly the same function, but with hidden unit activation functions given by $tanh(a)$ where the tanh function is defined by :

$$tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \tag{8}$$