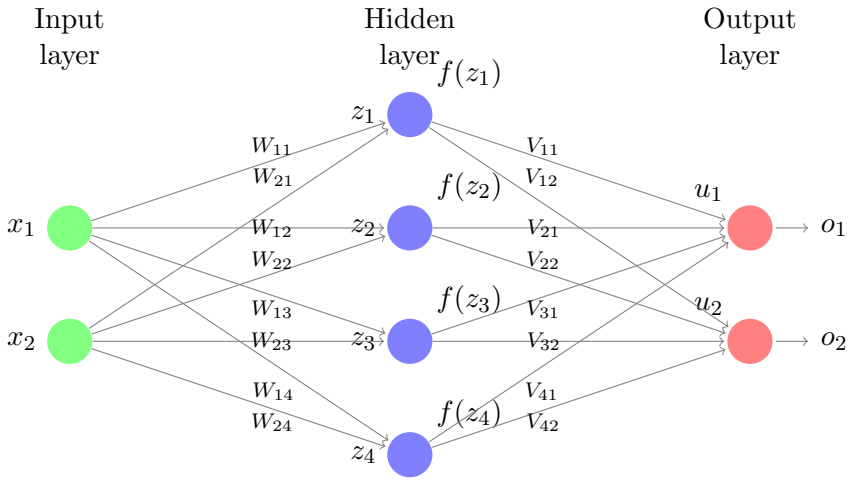
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.036—Introduction to Machine Learning
Spring 2017

Assignment 4
Issued: Friday 3/10 Due: Friday 3/17 at 9am

1. **Neural networks**

In this problem we will analyze a simple neural network to understand its classification properties. Consider the neural network given in the figure below, with ReLU activation functions (denoted by f) on all neurons, and a softmax activation function in the output layer:



Given an input $x = [x_1, x_2]^T$, the hidden units in the network are activated in stages as described by the following equations:

$$z_1 = x_1 W_{11} + x_2 W_{21} + W_{01} \qquad f(z_1) = \max\{z_1, 0\}$$
$$z_2 = x_1 W_{12} + x_2 W_{22} + W_{02} \qquad f(z_2) = \max\{z_2, 0\}$$
$$z_3 = x_1 W_{13} + x_2 W_{23} + W_{03} \qquad f(z_3) = \max\{z_3, 0\}$$
$$z_4 = x_1 W_{14} + x_2 W_{24} + W_{04} \qquad f(z_4) = \max\{z_4, 0\}$$

$$u_1 = f(z_1)V_{11} + f(z_2)V_{21} + f(z_3)V_{31} + f(z_4)V_{41} + V_{01} \qquad f(u_1) = \max\{u_1, 0\}$$
$$u_2 = f(z_1)V_{12} + f(z_2)V_{22} + f(z_3)V_{32} + f(z_4)V_{42} + V_{02} \qquad f(u_2) = \max\{u_2, 0\}.$$

The final output of the network is obtained by applying the *softmax* function to the last hidden layer,

$$o_1 = \frac{e^{f(u_1)}}{e^{f(u_1)} + e^{f(u_2)}}$$
$$o_2 = \frac{e^{f(u_2)}}{e^{f(u_1)} + e^{f(u_2)}}.$$

In this problem, we will consider the following setting of parameters:

$$\begin{bmatrix} W_{11} & W_{21} & W_{01} \\ W_{12} & W_{22} & W_{02} \\ W_{13} & W_{23} & W_{03} \\ W_{14} & W_{24} & W_{04} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ -1 & 0 & -1 \\ 0 & -1 & -1 \end{bmatrix},$$

$$\begin{bmatrix} V_{11} & V_{21} & V_{31} & V_{41} & V_{01} \\ V_{12} & V_{22} & V_{32} & V_{42} & V_{02} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 \\ -1 & -1 & -1 & -1 & 2 \end{bmatrix}.$$

(a) Consider the input $x_1 = 3$, $x_2 = 14$. What is the final output (o_1, o_2) of the network?

(b) Draw the "decision boundaries" in x-space, corresponding to the four hidden units. These are the regions where the input to the units z_1, z_2, z_3, z_4 are exactly zero. Please label the sides of the boundaries with "0" and "+" to indicate whether the unit's output would be exactly 0 or positive, respectively.

(c) Plot (in a new figure) the region in x-space where

$$f(z_1) + f(z_2) + f(z_3) + f(z_4) = 0.$$

What is the output value of the neural network in this case?

Repeat this for

$$f(z_1) + f(z_2) + f(z_3) + f(z_4) = 1$$

and

$$f(z_1) + f(z_2) + f(z_3) + f(z_4) = 3.$$

You can show all these plots in a single figure, if you prefer.

(d) Now, suppose we modify the network's softmax function as follows:

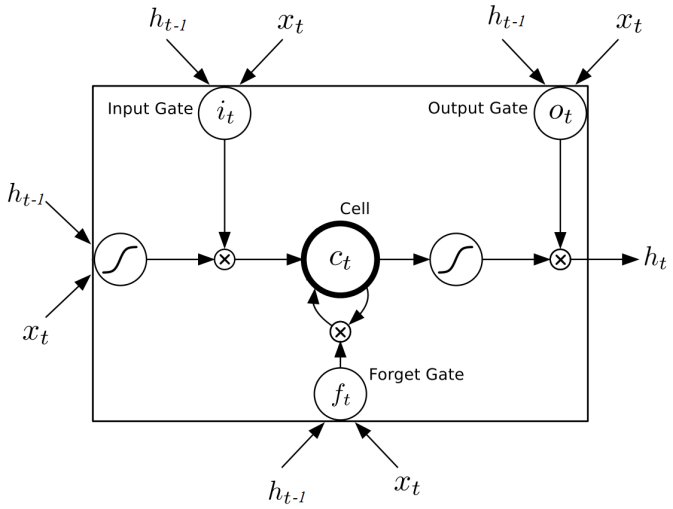
$$o_1 = \frac{e^{\beta f(u_1)}}{e^{\beta f(u_1)} + e^{\beta f(u_2)}}$$
$$o_2 = \frac{e^{\beta f(u_2)}}{e^{\beta f(u_1)} + e^{\beta f(u_2)}},$$

where $\beta > 0$ is a parameter. Note that our previous setting corresponded to the special case $\beta = 1$. What happens to the size of the region where $o_2 \geq \frac{1}{1000}$ as we increase β from 1 to 3?

(e) **(Optional)** The parameter β above is usually referred to in the literature as the "inverse temperature." What do you think is the origin of this terminology? Investigate what happens to the neural network as you take $\beta \to 0$ or $\beta \to +\infty$.

2. **Recurrent Neural Networks (LSTM)**

The diagram below shows a single LSTM unit that consists of Input, Output, and Forget gates.



The behavior of such a unit as a recurrent neural network is specified by a set of update equations. These equations define how the gates, "memory cell" c_t and the "visible state" h_t are updated in response to input x_t and previous states c_{t-1}, h_{t-1}. For the LSTM unit,

$$
\begin{aligned}
f_t &= \text{sigmoid}(W^{f,h}h_{t-1} + W^{f,x}x_t + b_f) \\
i_t &= \text{sigmoid}(W^{i,h}h_{t-1} + W^{i,x}x_t + b_i) \\
o_t &= \text{sigmoid}(W^{o,h}h_{t-1} + W^{o,x}x_t + b_o) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W^{c,h}h_{t-1} + W^{c,x}x_t + b_c) \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

where symbol $\odot$ stands for element-wise multiplication. The adjustable parameters in this unit are matrices $W^{f,h}$, $W^{f,x}$, $W^{i,h}$, $W^{i,x}$, $W^{o,h}$, $W^{o,x}$, $W^{c,h}$, $W^{c,x}$, as well as the offset parameter vectors b_f, b_i, b_o, and b_c. By changing these parameters, we change how the unit evolves as a function of inputs x_t.

To keep things simple, in this problem we assume that x_t, c_t, and h_t are all scalars. Concretely, suppose that the parameters are given by

$$
\begin{aligned}
W^{f,h} &= 0 & W^{f,x} &= 0 & b_f &= -100 & W^{c,h} &= -100 \\
W^{i,h} &= 0 & W^{i,x} &= 100 & b_i &= 100 & W^{c,x} &= 50 \\
W^{o,h} &= 0 & W^{o,x} &= 100 & b_o &= 0, & b_c &= 0
\end{aligned}
$$

We run this unit with initial conditions $h_{-1} = 0$ and $c_{-1} = 0$, and in response to two different input sequences

$$
\begin{aligned}
&\texttt{input sequence 1:} \quad [0, 0, 1, 1, 1, 0] \\
&\texttt{input sequence 2:} \quad [1, 1, 0, 1, 1].
\end{aligned}
$$

For example, for sequence 1, $x_0 = 0$, $x_1 = 0$, $x_2 = 1$, and so on. The unit is run separately for each different input sequence.

(a) For each input sequence, calculate the values h_t at each time-step, and write them in the table below. For ease of calculation, you can assume that you round h_t to the closest integer in every time-step. E.g., assume sigmoid$(50) \approx 1$ and $\tanh(-50) \approx -1$.

input	h_0	h_1	h_2	h_3	h_4	h_5
1						
2						X

(b) Can you interpret what information is carried in the visible state h_t?

(c) Modify the weights and/or offsets in such a way that new sequence of h_t's is the *negative* of the old h_t's for the same binary (0/1) input sequence x_t.

(d) **Optional:** Suppose that x_t is either 1 or -1, and you want to train the network to output h_t as the *sign* of the sum of -1's and 1's seen so far, or 0 if this sum is 0. For ease of calculation, assume that you round h_t to the closest integer in every time step. For instance, these are allowed input-output pairs:

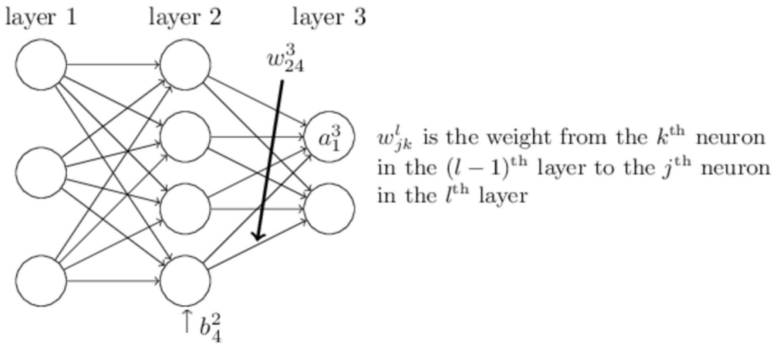
$$[-1, -1, -1] \longrightarrow -1$$
$$[-1, 1] \longrightarrow 0$$
$$[1, -1, 1] \longrightarrow 1$$

Based on your understanding of the functionality of the Input, Output, and Forget gates, can you design weights and offsets for such a network? (Assume that $h_{-1} = h_{-2} = \cdots = 0$, and $c_{-1} = c_{-2} = \cdots = 0$)

Hint: It should be enough to have h_t be only -1, 0, or 1, because the LSTM is able to model this behavior exactly, achieving zero error.

3. **Backpropagation**

We have seen during the lecture how to train multi-layer neural networks as classifiers using stochastic gradient descent (SGD). One of the key steps in the SGD method is the evaluation of the gradient of the loss function with respect to the model parameters. We saw this in detail in the lecture, but only for models that have at most one hidden layer. In this exercise, you will derive the backpropagation method, now for a general L-layer neural network.



We will use the following notation: b^l_j is the bias of the j^{th} neuron in the l^{th} layer, a^l_j is the activation of the j^{th} neuron in the l^{th} layer, and w^l_{jk} denotes the weight for the connection from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer. If the activation function is f and the loss function to be minimized is C, then the equations describing the network are:

$$a^l_j = f\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right), \qquad \text{for } l = 1, \ldots, L$$
$$\text{Loss} = C(a^L).$$

(a) Let's define the weighted input to the neurons in layer l by $z^l \equiv w^l a^{l-1} + b^l$ (as a result, $a^l = f(z^l)$), and the error δ^l_j of neuron j in layer l by $\delta^l_j \equiv \frac{\partial C}{\partial z^l_j}$. Let δ^l denote the vector of errors associated with layer l.

The backpropagation algorithm will give us a way of computing δ^l for every layer, and then relating those errors to the quantities of real interest, the partial derivatives of the loss $\frac{\partial C}{\partial w^l_{jk}}$ and $\frac{\partial C}{\partial b^l_j}$.

i. Show that

$$\delta^L_j = \frac{\partial C}{\partial a^L_j} f'(z^L_j), \qquad \delta^l_j = \sum_k w^{l+1}_{kj} \delta^{l+1}_k f'(z^l_j).$$

Hint: Use the chain rule.

ii. Show that the partial derivatives of the loss C satisfy

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j, \qquad \frac{\partial C}{\partial b^l_j} = \delta^l_j.$$

Hint: Use the chain rule, again...

iii. Explain in detail how to use these expressions to run the SGD algorithm.

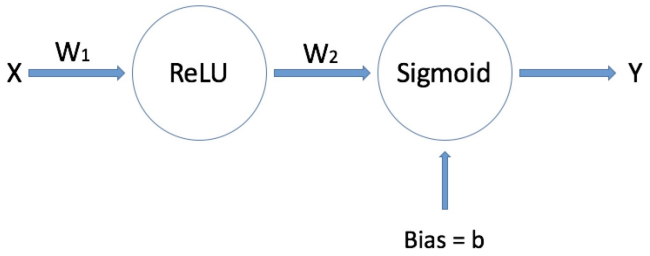
(b) Recall from the lecture that there are several different possible choices of activation functions f (e.g., tanh, ReLU). Let's get more familiar with one of them: the sigmoid function $\sigma(z) := \frac{1}{1+e^{-z}}$.

i. Sketch both the ReLU and sigmoid functions.

ii. Compute the derivative of the sigmoid function, and sketch it.

iii. Explain why the sigmoid may be a suitable choice for an activation function.

Now, we will compare the ReLU and sigmoid functions, and see how they work differently as activation functions. Consider a simple 2-layer neural network with a single neuron in each layer. The loss function is the quadratic loss $C = \frac{1}{2}(y-t)^2$, where y is the predicted value and t is the given target output value (or label value).



(c) Assume the operation of the network is given by the following expressions:

$$z_1 = w_1 x, \qquad a_1 = \text{ReLU}(z_1), \qquad z_2 = w_2 a_1 + b, \qquad y = \sigma(z_2).$$

Give symbolic expressions for the partial derivatives (i.e., the gradient) of the loss C with respect to w_1 and w_2: $\frac{\partial C}{\partial w_1}$ and $\frac{\partial C}{\partial w_2}$.

(d) Consider now a target output value $t = 1$ and input value $x = 3$. The weights and bias are initialized as $w_1 = 0.01$, $w_2 = -5$ and $b = -1$, respectively.

i. Evaluate the expressions for the partial derivatives you computed in the previous item, for the given values of the parameters.

ii. Write down an explicit update rule for the parameter w_1 in the SGD algorithm, when using a stepsize η.

(e) Assume that the input value x or the stepsize η are very large. In this case, after one iteration, will w_1 ever be updated again? Based on your answer, explain whether using a ReLU activation function at each layer (including the output layer) may or may not be a good idea. Would using a sigmoid be more useful?