PRACTICE EXAM 3

Exam will be Monday, December 13, 2021, 8:00-11:00am

- This will be a CLOSED BOOK exam.
- You will be permitted two sheets of handwritten notes, 8.5x11.
- Calculators and computers are not permitted.
- If you’re taking the exam online, you will need to have your webcam turned on. Your exam will be provided by e-mail and on zoom at exactly 8:00am; you will need to photograph and upload your answers by exactly 11:00am.
- There will be a total of 200 points in the exam. Each problem specifies its point total. Plan your work accordingly.
- You must SHOW YOUR WORK to get full credit.
1. (20 points) Suppose we’re trying to predict the sequence $\zeta_1, \ldots, \zeta_{100}$ from the sequence $x_1, \ldots, x_{100}$. We want to use some type of neural net (fully-connected, CNN, or LSTM) to compute $z_1, \ldots, z_{100}$ in order to minimize the error

$$E = \frac{1}{200} \sum_{t=1}^{100} (z_t - \zeta_t)^2$$

We only have one training sequence $(x_1, \ldots, x_{100}, \zeta_1, \ldots, \zeta_{100})$.

(a) Suppose we use a fully-connected one-layer neural net, with 10,000 trainable network weights $w_{kj}$, and 100 trainable bias terms $w_k$, such that

$$z_k = \sigma \left( w_k + \sum_{j=1}^{100} w_{kj} x_j \right)$$

where $\sigma(x) = 1/(1 + e^{-x})$ is the logistic nonlinearity. Find the derivatives of the error with respect to the weights ($dE/dw_{kj}$) and biases ($dE/dw_k$). Express your answers in terms of $x_j$, $z_k$, and $\zeta_k$ for appropriate values of $k$ and $j$; the terms $w_{kj}$ and $w_k$ should not show up on the right-hand-side of any of your equations.

**Solution:**

$$\frac{dE}{dw_k} = \frac{1}{100} (z_k - \zeta_k) z_k (1 - z_k)$$

$$\frac{dE}{dw_{kj}} = \frac{1}{100} (z_k - \zeta_k) z_k (1 - z_k) x_j$$
Suppose we use a CNN (convolutional neural net) with 99 trainable weights \( w[\tau] \) and a single scalar bias term, \( b \), i.e.,

\[
z_t = \sigma \left( b + \sum_{\tau=-49}^{49} w[\tau] x_{t-\tau} \right)
\]

where \( \sigma(x) = 1/(1 + e^{-x}) \) is the logistic nonlinearity. Find the derivatives of the error with respect to the weights \( (dE/dw[\tau]) \) and bias \( (dE/db) \). Assume that \( x_t = 0 \) for \( t \leq 0 \) or \( t \geq 101 \). Express your answers in terms of \( x_j, z_k \), and \( \zeta_k \) for appropriate values of \( k \) and \( j \); the terms \( w[\tau] \) and \( b \) should not show up on the right-hand-side of any of your equations.

Solution:

\[
\frac{dE}{db} = \frac{1}{100} \sum_{t=1}^{100} (z_t - \zeta_t) z_t (1 - z_t)
\]

\[
\frac{dE}{d w[\tau]} = \frac{1}{100} \sum_{t=1}^{100} (z_t - \zeta_t) z_t (1 - z_t) x_{t-\tau}
\]
(c) Suppose we use an **RNN (recurrent neural network)** with just one scalar memory cell whose weights and biases are $w$, $u$, and $b$:

$$ z_t = \sigma(ux_t + wz_{t-1} + b) $$

Find the derivatives of the error with respect to the weights and biases ($dE/du$, $dE/dw$, and $dE/db$). Express your answers in terms of $x_j$, $z_k$, and $\zeta_k$ for appropriate values of $k$ and $j$; **the terms** $u$, $w$ **and** $b** should not show up on the right-hand-side of any of your equations. You may express your answer recursively, or your answer may contain summation ($\sum$) and/or product ($\prod$) terms.

**Solution:**

\[
\begin{align*}
\frac{dE}{dz_t} &= \frac{\partial E}{\partial z_t} + \frac{dE}{dz_{t+1}} \frac{\partial z_{t+1}}{\partial z_t} \\
&= \frac{1}{100}(z_t - \zeta_t) + \frac{dE}{dz_{t+1}}z_t(1 - z_{t+1})w
\end{align*}
\]

\[
\begin{align*}
\frac{dE}{db} &= \sum_{t=1}^{100} \frac{dE}{dz_t} \frac{\partial z_t}{\partial b} \\
&= \sum_{t=1}^{100} \frac{dE}{dz_t} z_t(1 - z_t)
\end{align*}
\]

\[
\begin{align*}
\frac{dE}{du} &= \sum_{t=1}^{100} \frac{dE}{dz_t} \frac{\partial z_t}{\partial u} \\
&= \sum_{t=1}^{100} \frac{dE}{dz_t} z_t(1 - z_t)x_t
\end{align*}
\]

\[
\begin{align*}
\frac{dE}{dw} &= \sum_{t=1}^{100} \frac{dE}{dz_t} \frac{\partial z_t}{\partial w} \\
&= \sum_{t=2}^{100} \frac{dE}{dz_t} z_t(1 - z_t)z_{t-1}
\end{align*}
\]
(d) Suppose we use an LSTM (long-short-term memory network) whose weights and biases are pre-specified: $u_c = 1$, and all of the other weights and biases are zero:

\[ b_c = 0, u_c = 1, w_c = 0, b_f = 0, u_f = 0, w_f = 0, b_i = 0, u_i = 0, w_i = 0, b_o = 0, u_o = 0, w_o = 0 \]

\[
f[t] = \sigma(u_f x_t + w_f z_{t-1} + b_f), \quad i[t] = \sigma(u_i x_t + w_i z_{t-1} + b_i), \quad o[t] = \sigma(u_o x_t + w_o z_{t-1} + b_o) \]
\[
c[t] = f[t]c[t-1] + i[t] \sigma(u_c x_t + w_c z_{t-1} + b_c), \quad z_t = o[t]c[t] \]

Assume that $c[t] = 0$ for $t \leq 0$. Express $z_t$ in terms of $\sigma(x_t)$ for $0 \leq t \leq 100$. Your answer should NOT contain any of the variables $c[t]$, $f[t]$, $i[t]$, or $o[t]$. Your answer may contain a summation ($\sum$). You may find it useful to know that $\sigma(0) = \frac{1}{2}$.

**Solution:**

\[
c[t] = \frac{1}{2} c[t-1] + \frac{1}{2} \sigma(x_t) \]
\[
z_t = \frac{1}{2} c[t] \]
\[
= \sum_{\tau=1}^{t} \left( \frac{1}{2} \right)^{2+t-\tau} \sigma(x_{\tau}) \]
2. (20 points) In class, we have been working with the exponential softmax function, but other forms exist. For example, the polynomial softmax function transforms inputs $b_\ell$ into outputs $z_\ell$ according to

$$z_\ell = \frac{b_\ell^p}{\sum_k b_k^p},$$

for some constant integer power, $p$. The cross-entropy loss is

$$E = -\sum_\ell \zeta_\ell \ln z_\ell, \quad \zeta_\ell = \begin{cases} 1 & \ell = \ell^* \\ 0 & \text{otherwise} \end{cases}$$

Find $\frac{\partial E}{\partial b_j}$ for all $j$.

**Solution:** Define

$$\delta_{j\ell} = \begin{cases} 1 & j = \ell \\ 0 & \text{otherwise} \end{cases}$$

Then

$$\frac{\partial E}{\partial b_j} = -\sum_\ell \frac{\zeta_\ell}{z_\ell} \left(\frac{p b_j^{p-1} \delta_{j\ell}}{\sum_k b_k^p} - \frac{p b_j^p b_{j^*}^{p-1}}{\left(\sum_k b_k^p\right)^2}\right)$$

Actually, that’s an adequate solution. But if we want, we could simplify it:

$$\frac{\partial E}{\partial b_j} = -\sum_\ell \frac{p}{b_j} \zeta_\ell \left(\delta_{j\ell} - z_j\right)$$

$$= -\frac{p}{b_j} \left(\delta_{j\ell^*} - z_j\right) = -\frac{p}{b_j} \left(\zeta_j - z_j\right)$$
3. (20 points) Suppose you have a 10-pixel input image, $x[n]$. This is processed by a one-pixel “convolution” (really just multiplication by a scalar coefficient, $w$), followed by a stride-2 max pooling layer, thus:

$$a[n] = wx[n], \quad 1 \leq n \leq 10$$
$$y[k] = \max \left( 0, \max_{2k-1 \leq n \leq 2k} a[n] \right), \quad 1 \leq k \leq 5$$

Suppose you know the input $x[n]$, and you know $\epsilon[k] = \frac{\partial E}{\partial y[k]}$. Find $\frac{\partial E}{\partial w}$ in terms of $x[n]$ and $\epsilon[k]$.

**Solution:**

$$\frac{\partial E}{\partial w} = \sum_{k=1}^{5} \epsilon[k] x \left[ \arg\max_{2k-1 \leq n \leq 2k} a[n] \right]$$
4. (20 points) Suppose that you have a training dataset with \( n \) training tokens \( \{(\vec{x}_1, \zeta_1), \ldots, (\vec{x}_n, \zeta_n)\} \), where \( \vec{x}_i = [x_{i1}, \ldots, x_{ip}]^T \), and \( \zeta_i \in \{0, 1\} \). You have a one-layer neural network that tries to approximate \( \zeta_i \) with \( z_i \), computed as \( z_i = \sigma(\vec{w}^T \vec{x}_i) \), where \( \sigma(\cdot) \) is the logistic function, and \( \vec{w} \) is a weight vector. Suppose that you want to maximize the accuracy of \( z_i \), but you also want to make \( \vec{w}^T \vec{x}_i \) as small as possible. One way to do this is by using a two-part error metric,

\[
E = -\frac{1}{n} \sum_{i=1}^{n} (\zeta_i \ln z_i + (1 - \zeta_i) \ln(1 - z_i)) + \frac{1}{2n} \sum_{i=1}^{n} (\vec{w}^T \vec{x}_i)^2
\]

Find \( \nabla_{\vec{w}} E \), the gradient of \( E \) with respect to \( \vec{w} \).

Solution:

\[
\nabla_{\vec{w}} E = \frac{1}{n} \sum_{i=1}^{n} \left( (1 - \zeta_i)z_i - \zeta_i(1 - z_i) + \vec{w}^T \vec{x}_i \right) \vec{x}_i^T
\]
5. (20 points) Consider a scalar LSTM, with a scalar memory cell, input gate, output gate, and forget gate, related to one another by scalar coefficients $a_i, b_i, a_o, b_o, a_f, b_f, a_c, b_c$ as follows:

$$i[n] = \text{input gate} = \sigma(b_i x[n] + a_i c[n-1]), \quad 1 \leq n$$
$$o[n] = \text{output gate} = \sigma(b_o x[n] + a_o c[n-1]), \quad 1 \leq n$$
$$f[n] = \text{forget gate} = \sigma(b_f x[n] + a_f c[n-1]), \quad 1 \leq n$$
$$c[n] = f[n] c[n-1] + i[n] (b_c x[n] + a_c c[n-1]), 1 \leq n$$
$$y[n] = o[n] c[n], \quad 1 \leq n$$

Suppose that the network is initialized with $b_i = b_o = b_f = a_i = a_o = a_f = a_c = 0$, and $c[0] = 0$. In fact, the only nonzero coefficient is $b_c = 1$. Under this condition, find a formula for $y[n]$ in terms of the values of $x[m], \quad 1 \leq m \leq n$. No variables other than $x[m]$ should appear in your answer. HINT: $\sigma(0) = 1/2$.

**Solution:**

$$y[n] = \sum_{m=1}^{n} \left( \frac{1}{2} \right)^{n-m+2} x[m]$$
6. (20 points) In class, we have been working with nodes in layers, but a neural net can also be defined as a fully-connected graph, with every node connected to every other node. For example, suppose there is a scalar input $x$, and

$$y_0 = x$$

$$a_\ell = \sum_{k=0}^{\ell-1} w_{\ell k} y_k, \quad 1 \leq \ell \leq L$$

$$y_\ell = \sigma(a_\ell), \quad 1 \leq \ell \leq L$$

$$E = \frac{1}{2} \sum_{\ell=1}^{L} (y_\ell - y_\ell^*)^2$$

Define the back-propagation error to be $\delta_\ell = \frac{dE}{da_\ell}$. Find an algorithm that computes $\delta_\ell$ for all $1 \leq \ell \leq L$.

**Solution:**

$$\delta_\ell = \left( (y_\ell - y_\ell^*) + \sum_{k=\ell+1}^{L} \delta_k w_{kl} \right) \sigma'(a_\ell)$$

$$= \left( (y_\ell - y_\ell^*) + \sum_{k=\ell+1}^{L} \delta_k w_{kl} \right) y_\ell (1 - y_\ell)$$
7. (20 points) A convolutional layer leads to convolutional back-propagation. In the neural net literature, however, convolution is sometimes replaced (without comment!) by correlation, resulting in something like the following, where \( x[m_1, m_2] \) is the input and \( u[m_1, m_2] \) are the network weights:

\[
a[n_1, n_2] = \sum_{m_1} \sum_{m_2} u[m_1 - n_1, m_2 - n_2] x[m_1, m_2]
\]

Suppose the error, \( E \), is some function whose partial derivatives \( \epsilon[n_1, n_2] = \frac{\partial E}{\partial a[n_1, n_2]} \) are known. Define \( \delta[m_1, m_2] = \frac{\partial E}{\partial x[m_1, m_2]} \). Find \( \delta[m_1, m_2] \) in terms of \( \epsilon[n_1, n_2] \).

**Solution:**

\[
\delta[m_1, m_2] = \sum_{n_1} \sum_{n_2} \epsilon[n_1, n_2] u[m_1 - n_1, m_2 - n_2]
\]
8. (20 points) Suppose you have a dataset containing audio waveforms, $\vec{x}_i$, each matched with two different one-hot label vectors. The label vector $\vec{y}_i^* = [y_{i1}^*, \ldots, y_{iq}^*]^T$, where $y_{ij}^* \in \{0, 1\}$, is approximated by the network output $\vec{y}_i = [y_{i1}, \ldots, y_{iq}]^T$, where $y_{ij} \in (0, 1)$. The label vector $\vec{z}_i^* = [z_{i1}^*, \ldots, z_{ir}^*]^T$, where $z_{ij}^* \in \{0, 1\}$, is approximated by the network output $\vec{z}_i = [z_{i1}, \ldots, z_{ir}]^T$, where $z_{ij} \in (0, 1)$. Both $\vec{y}_i$ and $\vec{z}_i$ are functions of a hidden nodes vector $\vec{h}_i$ as

$$\vec{h}_i = g(W\vec{x}_i)$$
$$\vec{y}_i = \text{softmax}(U\vec{h}_i)$$
$$\vec{z}_i = \text{softmax}(V\vec{h}_i)$$

where $U$, $V$ and $W$ are trainable weight matrices, and $g(\cdot)$ is some scalar nonlinearity. Find an error metric $E$ such that, by minimizing $E$, you can:

- **maximize** the accuracy of $\vec{y}_i$ as an estimate of $\vec{y}_i^*$
- **minimize** the accuracy of $\vec{z}_i$ as an estimate of $\vec{z}_i^*$

**Solution:** There are many, many acceptable solutions. Most are forms of $E$ with two terms: the first term is reduced as $y_{ik}$ gets more accurate, the second term is reduced as $z_{il}$ gets more inaccurate. For example,

$$E = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{q} y_{ik}^* \ln y_{ik} + \frac{1}{n} \sum_{i=1}^{n} \sum_{\ell=1}^{r} z_{i\ell}^* \ln z_{i\ell}$$
9. (20 points) Consider an LSTM defined by

\[
\begin{align*}
\vec{i}[n] &= \text{input gate} = \sigma(B_i \vec{x}[n] + A_i \vec{c}[n - 1]) \\
\vec{o}[n] &= \text{output gate} = \sigma(B_o \vec{x}[n] + A_o \vec{c}[n - 1]) \\
\vec{f}[n] &= \text{forget gate} = \sigma(B_f \vec{x}[n] + A_f \vec{c}[n - 1]) \\
\vec{c}[n] &= \vec{f}[n] \odot \vec{c}[n - 1] + \vec{i}[n] \odot g(B_c \vec{x}[n] + A_c \vec{c}[n - 1]) \\
\vec{y}[n] &= \vec{o}[n] \odot \vec{c}[n]
\end{align*}
\]

where the vector cell is \(\vec{c}[n] = [c_1[n], \ldots, c_p[n]]^T\), and where \(\odot\) denotes the Hadamard (array) product, e.g., \(\vec{o}[n] \odot \vec{c}[n] = [o_1[n]c_1[n], \ldots, o_p[n]c_p[n]]^T\). Find the derivative \(\frac{\partial c_j[n]}{\partial c_k[n-1]}\).

**Solution:** Define \(\delta_{jk} = \begin{cases} 1 & j = k \\ 0 & \text{else} \end{cases}\). Define \(\vec{c}_j[n]\) to be the \(j\)th element of \(g(B_c \vec{x}[n] + A_c \vec{c}[n - 1])\), and

\[
\frac{\partial c_j[n]}{\partial c_k[n-1]} = \frac{\partial c_j[n]}{\partial f_j[n]} \frac{\partial f_j[n]}{\partial c_k[n-1]} + f_j[n] \delta_{jk} + \frac{\partial c_j[n]}{\partial i_j[n]} \frac{\partial i_j[n]}{\partial c_k[n-1]} + i_j[n] \frac{\partial \vec{c}_j[n]}{\partial c_k[n-1]}
\]

Now define \(\vec{c}'_j[n]\) to be the \(j\)th element of \(g'(B_c \vec{x}[n] + A_c \vec{c}[n - 1])\). We could also define \(i'_j[n]\) to be the \(j\)th element of \(\sigma'(B_i \vec{x}[n] + A_i \vec{c}[n - 1])\), but actually we don’t need to, since the derivative of the logistic function is just \(i_j[n](1 - i_j[n])\). Define \(a_{mn}'\) to be the \((m, n)\)th element of the matrix \(A_o\), and so on. Then

\[
\frac{\partial c_j[n]}{\partial c_k[n-1]} = c_j[n-1]f_j[n](1 - f_j[n])a'_{jk} + f_j[n] \delta_{jk} + \vec{c}_j[n]i_j[n][1 - i_j[n]]a'_j[k] + i_j[n] \vec{c}'_j[n]a'_c[k]
\]
10. (20 points) A particular two-layer neural network accepts a two-dimensional input vector \( \vec{x} = [x_1, x_2, 1]^T \), and generates an output \( z = h(\vec{v}^T g(U\vec{x})) \). Choose network weights \( \vec{v} \) and \( U \), and element-wise scalar nonlinearities \( h() \) and \( g() \), that will generate the following output:

\[
z = \begin{cases} 
1 & |x_1| < 2 \text{ and } |x_2| < 2 \\
-1 & \text{otherwise}
\end{cases}
\]

**Solution:** Several solutions are possible. Here’s one.

\[
U = \begin{bmatrix}
-1 & 0 & 2 \\
1 & 0 & 2 \\
0 & -1 & 2 \\
0 & 1 & 2 \\
0 & 0 & 1
\end{bmatrix}
\]

\[
g(a) = u(a)
\]

\[
\vec{v}^T = [1, 1, 1, 1, -3.5]
\]

\[
h(a) = \text{sgn}(b)
\]

For this definition to exactly match the problem statement, it’s necessary to define \( u(0) = 0 \).
11. (20 points) Your training database contains matched pairs \( \{(\vec{x}_1, \vec{y}_1), \ldots, (\vec{x}_n, \vec{y}_n)\} \) where \( \vec{x}_i \) is the \( i \)th observation vector, and \( \vec{y}_i \) is the \( i \)th label vector. For some initial weight matrix \( W = \begin{bmatrix} w_{11} & \cdots & w_{1p} \\ \vdots & \ddots & \vdots \\ w_{q1} & \cdots & w_{qp} \end{bmatrix} \), you have already computed the following two quantities:

\[
\begin{align*}
    f_\ell(\vec{x}_i, W) & \quad 1 \leq \ell \leq r, \ 1 \leq i \leq n \quad (1) \\
    \frac{\partial f_\ell(\vec{x}_i, W)}{\partial w_{kj}} & \quad 1 \leq \ell \leq r, \ 1 \leq k \leq q, \ 1 \leq j \leq p, \ 1 \leq i \leq n \quad (2)
\end{align*}
\]

You want to find a new matrix \( W' = \begin{bmatrix} w'_{11} & \cdots & w'_{1p} \\ \vdots & \ddots & \vdots \\ w'_{q1} & \cdots & w'_{qp} \end{bmatrix} \) such that \( J(W') \geq J(W) \) (that is, you want to maximize \( J \)), where

\[
J(W) = \sum_{i=1}^{n} \sum_{\ell=1}^{r} y_{\ell,i} \ln(f_\ell(\vec{x}_i, W))
\]

Give a formula for \( w'_{kj} \) in terms of \( w_{kj}, f_\ell(\vec{x}_i, W), \frac{\partial f_\ell(\vec{x}_i, W)}{\partial w_{kj}} \), and in terms of a step size, \( \eta \), such that for suitable values of \( \eta \), \( J(W') \geq J(W) \).

**Solution:**

\[
\frac{dJ(W)}{dw_{kj}} = \sum_{i} \sum_{\ell} \frac{dJ(W)}{df_\ell(\vec{x}_i, W)} \frac{\partial f_\ell(\vec{x}_i, W)}{\partial w_{kj}} = \sum_{i} \sum_{\ell} y_{\ell,i} \frac{\partial f_\ell(\vec{x}_i, W)}{\partial w_{kj}}
\]

In this case we are trying to increase \( J(W) \), rather than decreasing it, so the gradient update should be in the direction of the gradient, not in the opposite direction, thus:

\[
w'_{kj} = w_{kj} + \eta \sum_{i} \sum_{\ell} y_{\ell,i} \frac{\partial f_\ell(\vec{x}_i, W)}{\partial w_{kj}}
\]