Final exam

Please hand in the final exam via MarkUs by 9am on Friday, December 15th. You are welcome to spend any amount of time working on the exam before then. The exam is "open-course-materials": you are allowed to consult the textbook (d2l.ai), my lecture notes, class recordings, your own course notes, and past homeworks. You are welcome to check your answers by writing and running code, but this should not be necessary for any of the problems. No collaboration with other students or use of other sources is allowed.

Problem 1 (2 points)

Show that a softmax nonlinearity with two logits is equivalent to a sigmoid nolinearity. Hint: Write down $p(y=0)$ and $p(y=1)$ for the softmax and sigmoid and find an expression for the sigmoid logit in terms of the softmax logits that causes the two to be equivalent.

Answer 1

We want to classify y within two classes $\{0, 1\}$:

$$sigmoid(x) = \frac{1}{1+\exp(-x)}$$

$$softmax(\mathbf{x})_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

For softmax we have:

$$\begin{aligned} p(y=0) &= softmax(\mathbf{z})_0 \\ &= \frac{\exp(z_0)}{\sum_j \exp(z_j)} \\ &= \frac{\exp(z_0)}{\exp(z_0)+\exp(z_1)} \\ &= \frac{\exp(z_0)}{\exp(z_0)+\exp(z_1)} \frac{\exp(-z_0)}{\exp(-z_0)} \\ &= \frac{1}{1+\exp(z_1 - z_0)} \end{aligned}$$

$$\begin{aligned}
p(y=1) &= softmax(\mathbf{z})_1 \\
&= \frac{\exp(z_1)}{\sum_j \exp(z_j)} \\
&= \frac{\exp(z_1)}{\exp(z_0)+\exp(z_1)} \\
&= \frac{\exp(z_1)}{\exp(z_0)+\exp(z_1)} \frac{\exp(-z_1)}{\exp(-z_1)} \\
&= \frac{1}{1+\exp(z_0-z_1)}
\end{aligned}$$

For $sigmoid$ we consider only one logit z:

$$\begin{aligned}
p(y=0) = sigmoid(z) &= \frac{1}{1+\exp(-z)} \\
p(y=1) &= 1 - sigmoid(z) \\
&= 1 - \frac{1}{1+\exp(-z)} \\
&= \frac{1+\exp(-z)}{1+\exp(-z)} - \frac{1}{1+\exp(-z)} \\
&= \frac{\exp(-z)}{1+\exp(-z)} \\
&= \frac{1}{\exp(z)+1}
\end{aligned}$$

We can observe that $p(y=0)$ and $p(y=1)$ are equal both for sigmoid and softmax when $z = z_0 - z_1$. This holds for all values of z_0 and z_1.

Problem 2 (6 points)

Consider minimization of the function $y = \frac{x^2}{4}$ with respect to x. Assume that the initial value of x is $x_0 = 1$.

1. Find the learning rate so that gradient descent converges to the minimum in one iteration.
2. Now, consider minimization of this function with a momentum optimizer with the learning rate fixed to 1. Find the value of the momentum hyperparameter (we called it β in class) so that optimization converges to the minimum in two iterations.

Answer 2.1

By calculating the derivative of $f(x) = y$ with respect to x, we have:

$$\begin{aligned}\frac{dy}{dx} &= \frac{d}{dx}\left[\frac{x^2}{4}\right]\\ &= \frac{2x}{4}\\ &= \frac{x}{2}\end{aligned}$$

Making $f'(x) = 0$, gives us the minimum:

$$\begin{aligned}\frac{x}{2} &= 0\\ x &= 0\end{aligned}$$

The update rule for the first iteration of gradient descent is:

$$x_1 = x_0 - \eta\frac{dy}{dx}|_{x=x_0}$$

As we want to reach the minimum value at one iteration, we must have $x_1 = 0$. Since $x_0 = 1$, the value of the learning rate η that satisfies this requirement is found by:

$$\begin{aligned}0 &= 1 - \eta\frac{dy}{dx}|_{x=x_0}\\ 0 &= 1 - \eta\frac{1}{2}\\ \eta\frac{1}{2} &= 1\\ \eta &= 2\end{aligned}$$

Therefore, to reach the minimum via gradient descent in one iteration we must have a learning rate $\eta = 2$

Answer 2.2

The momentum optimizer (as defined in class/textbook) is :

$$\begin{aligned}V_t &= \beta V_{t-1} + g_t\\ x_t &= x_{t-1} - \eta V_t\\ \\ V_0 &= 0\\ g_t &= \nabla f_{x_{t-1}}(x)\end{aligned}$$

From the problem statement and Answer 2.1, we have:

$$x_0 = 1$$
$$\eta = 1$$
$$\nabla f(x) = \frac{d}{dx}\left[\frac{x^2}{4}\right] = \frac{x}{2}$$

Since we want to reach the minimum x = 0 in two iterations, we must have:

$$x_2 = 0$$

From this problem set, we formulate the following equations:

$$\begin{aligned} V_1 &= \beta V_0 + g_1 \\ &= 0 + \nabla f_{x_0}(x) \\ &= \frac{x_0}{2} \\ &= \frac{1}{2} \end{aligned}$$

Substitute $V_1 \rightarrow x_1$:

$$\begin{aligned} x_1 &= x_0 - \eta V_1 \\ &= 1 - 1\frac{1}{2} \\ &= \frac{1}{2} \end{aligned}$$

Substitute $x_1, V_1 \rightarrow V_2$:

$$\begin{aligned} V_2 &= \beta V_1 + g_2 \\ &= \beta\frac{1}{2} + \nabla f_{x_1}(x) \\ &= \frac{\beta}{2} + \frac{x_1}{2} \\ &= \frac{\beta}{2} + \frac{\frac{1}{2}}{2} \\ &= \frac{\beta}{2} + \frac{1}{4} \end{aligned}$$

Substitute $V_2, \rightarrow x_2$:

$$\begin{aligned} x_2 &= x_1 - \eta V_2 \\ 0 &= \frac{1}{2} - 1(\frac{\beta}{2} + \frac{1}{4}) \\ \frac{\beta}{2} + \frac{1}{4} &= \frac{1}{2} \\ \frac{\beta}{2} &= \frac{2}{4} - \frac{1}{4} = \frac{1}{4} \\ \beta &= \frac{2}{4} = \frac{1}{2} \end{aligned}$$

Therefore, to reach the minimum in two iterations with momentum optimizer, we must have $\beta = \frac{1}{2}$

Problem 3 (3 points)

Rewrite the following convolution

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} * \begin{bmatrix} k_1 & k_2 \end{bmatrix}$$

as a matrix-vector product. Assume there is no padding and all strides are 1.

Answer 3

The result of the given convolution is:

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} * \begin{bmatrix} k_1 & k_2 \end{bmatrix} = \begin{bmatrix} x_1k_1 + x_2k_2 & x_2k_1 + x_3k_2 & x_3k_1 + x_4k_2 \end{bmatrix}$$

Therefore, given the matrix B and the row-vector a,

$$B = \begin{bmatrix} x_1 & x_2 & x_3 \\ x_2 & x_3 & x_4 \end{bmatrix}$$
$$a = \begin{bmatrix} k_1 & k_2 \end{bmatrix}$$

we can rewrite the convolution as:

$$aB = \begin{bmatrix} k_1 & k_2 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & x_3 \\ x_2 & x_3 & x_4 \end{bmatrix} = \begin{bmatrix} x_1k_1 + x_2k_2 & x_2k_1 + x_3k_2 & x_3k_1 + x_4k_2 \end{bmatrix}$$

Problem 4 (6 points)

Consider the vanilla RNN layer:

$$h_t = \phi(W_x x_t + W_h h_{t-1} + b)$$

For the purposes of this problem, assume that $W_x = W_h = I$ (i.e. both weight matrices are the identity matrix), $b = 0$, and $\phi(x) = x$ (i.e. there is no nonlinearity).

1. Provide a closed-form expression for h_t in terms of $x_1, \ldots, x_t$.
2. Provide a closed-form expression for $\frac{\partial h_t}{\partial W_h}$ in terms of $h_0, \ldots, h_t$.

Answer 4.1

Since $b = 0$, $\phi(x) = x$, and $W_x = W_h = I$, we have:

$$h_t = x_t + h_{t-1}$$

Assumin $h_0 = 0$, we expand the expression and find the closed-form solution:

$$\begin{aligned} h_t &= x_t + x_{t-1} + h_{t-2} \\ &= x_t + x_{t-1} + x_{t-2} + h_{t-3} \\ &= x_t + x_{t-1} + x_{t-2} + x_{t-3} + h_{t-4} \\ &= x_t + x_{t-1} + x_{t-2} + x_{t-3} + \cdots + x_2 + x_1 \\ &= \sum_{i=1}^{t} x_i \end{aligned}$$

Answer 4.2

In Lecture 6 notes, we derived the following expression for a Linear Vanilla RNN:

$$h_t = W_x x_t + W_h h_{t-1}$$
$$\frac{dh_t}{dW_h} = h_{t-1} + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} W_h \right) h_{i-1}$$

Since $W_h = I$, we find the closed-form solution:

$$\begin{aligned} \frac{dh_t}{dW_h} &= h_{t-1} + \sum_{i=1}^{t-1} \left(\prod_{j=i+1}^{t} W_h \right) h_{i-1} \\ &= h_{t-1} + \sum_{i=1}^{t-1} h_{i-1} \\ &= h_0 + h_1 + \cdots + h_{t-2} + h_{t-1} \\ &= \sum_{i=0}^{t-1} h_i \end{aligned}$$

Problem 5 (2 points)

In class and in the textbook, we introduced Dropout as replacing an activation h_i with

$$h_i' = \begin{cases} 0 \text{ with probability } p \\ \frac{h_i}{1-p} \text{ with probability } 1-p \end{cases}$$

An alternative formulation is

$$h_i' = \begin{cases} 0 \text{ with probability } p \\ h_i \text{ with probability } 1-p \end{cases}$$

Assume that this alternative form of dropout is applied before a standard fully-connected (dense) layer. How should the weights of the fully-connected layer be modified so that when dropout is not applied the expected output of the layer is unchanged?

Answer 5

We first define the dense layer as a vector-matrix multiplication:

$$h^n = h^{n-1} W$$
$$(1)$$

Were h^{n-1} corresponds do the activation of the previous layer, W to the dense layer weight matrix, and h^n to the output generated by the dense layer.

In the original Dropout formulation, we have $E[h'] = h$, since the values that are not beind dropped are normalized.

In the formulation displayed in (1), the alternative Dropout is applied to h^{n-1}, that is multiplied by W resulting in h'^n. We want to modify W in order to $E[h'^n] = h^n$

To do so, we can normalize the weight matrix W by $1 - p$, resulting in:

$$W' = W \cdot \frac{1}{1-p}$$

Where $\cdot$ represents a pointwise multiplication, and W' is the modified fully connected layer weight matrix.

Problem 6 (2 points)

I decide that ReLU(x) increases so quickly so I try a new nonlinearity: $\text{LogLU}(x) = \log \text{ReLU}(x)$. I figure I'm being clever by applying the ReLU before the log so the output is always finite. I train a model using this nonlinearity and immediately get NaNs. Why?

Answer 6

The issue comes from the fact that $\log(0)$ is undefined. $ReLU(x) = max(0, x) = 0$, when $x \leq 0$. Therefore we will encouter $\log(0)$. In PyTorch, for example, `torch.log(torch.tensor(0))` results in `tensor(-inf)`

Even if we disconsidered this issue, we would also have a division by 0 on the derivative of the proposed $\text{LogLU}(x)$:

We know that for a given logarithmic base a function $\log_a(f(x))$ its derivative will be:

$$\frac{d}{dx}\left[\log_a(f(x))\right] = \frac{f'(x)}{\ln(a)f(x)}$$

In this case, $f(x) = ReLU(x)$, defined as:

$$ReLU(x) = max(0, x)$$

Therefore, we will have $ReLU(x) = 0$, when $x \leq 0$:

This will cause:

$$\frac{d}{dx}\left[\log_a(ReLU(x))\right] = \frac{ReLU'(x)}{\ln(a) \cdot 0} = \frac{ReLU'(x)}{0}$$

Resulting in a division by 0

Both issues can be visulized in the code snippet below:

```
import numpy as np
import torch
print(torch.log(torch.tensor(0)))
print(0/0)
```

```
tensor(-inf)
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-2-0a50c4651849> in <cell line: 4>()
      2 import torch
      3 print(torch.log(torch.tensor(0)))
----> 4 print(0/0)

ZeroDivisionError: division by zero
```

PESQUISAR NO STACK OVERFLOW

Problem 7 (9 points)

For this problem, consider a single-headed local self-attention mechanism we briefly discussed in class, i.e. one where the attention weight for any value more than k timesteps away from the query is set to zero.

1. Write down a modified version of standard self-attention that implements local self-attention. You should assume that standard self-attention uses the scaled dot-product comparison function and a softmax-weighted average for the reduction function. As usual, you should define what the queries, keys, values, comparison function, and reduction function are, though

you can feel free to just state that any of these these are computed the same as in standard self-attention.

2. Write down the number of times α needs to be called to compute the output of standard self-attention and local self-attention. What is the relative speedup or slowdown of using local self-attention?
3. Assume that the input to the single-headed local self-attention mechanism is a one-dimensional sequence. Modify queries, keys, and/or values for local self-attention so that it closely approximates max-pooling with a stride of 1 and a window size of $2k + 1$ around the current input. You must use the same definition for the comparison function and reduction function as in part 1.

Answer 7.1

The local self-attention can be defined as:

$$\begin{aligned}
&\mathbf{x} = x_1, x_2, \ldots, x_T, \quad x_t \in \mathbb{R}^d \\
&q_t = k_t = v_t = x_t \\
&comparison : a_{ij} = \alpha(q_i, k_j) = \frac{q_i^T k_j}{\sqrt{d_k}} \\
&reduction : softmax_{local}(a) \cdot v \\
&softmax_{local}(a_{ij}) = \frac{\exp(\alpha a_{ij})}{\sum_{l=i-k}^{i+k} \exp(\alpha a_{lj})} \\
&\alpha = 1
\end{aligned}$$

Here we substituted the original $softmax(a_{ij})$ by $softmax_{local}(a_{ij})$, making shure that the softmax average only takes into consideration the $[i-k, i+k]$ window. The comparison function is also only performed for values of $j \in [i-k, i+k]$. Also it is important to notice that we ignore boundary effects on the defined formulation (as mentined by Prof. Raffel here).

Important note

α is utilized here based on the textbook section on the softmax function. The section mentions the origins of softmax dating back to Gibbs and Boltzmann, showing that $\exp(\frac{-E}{kT})$ is parameterized by T to "favor lower or higher energy states E" Here we substitute $\frac{-1}{kT}$ by α. This will be use in Answer 7.3

Answer 7.2

For a given sequence $x_1, x_2, \ldots, x_T$, having $q_t = k_t$ we will have T^2 calls from the α function for the standard self attention.

When utilizing the proposed local self-attention, we'll have $T(2k+1)$ calls of the α function.

Therefore, the speedup from the local self-attention is equal to:

$$\frac{T^2}{T(2k+1)} = \frac{T}{2k+1}$$

We are disconsidering the boundary effects and any speedup gained from the reduction function.

Answer 7.3

We formulate queries, keys and values as:

$$V = \begin{bmatrix} x_1 \\ x_2 \\ \ldots \\ x_T \end{bmatrix}$$
$$K = I \in \mathbb{R}^{T \times T}$$
$$q_i = [x_1, x_2, \ldots, x_T], \quad q \in \mathbb{R}^{T \times T}$$

We ignore the behaviour at the boundaries again.

We have:

$$softmax_{local}(a_{ij}) = \frac{\exp(\alpha a_{ij})}{\sum_{l=i-k}^{i+k} \exp(\alpha a_{lj})}$$

As α grows , softmax will approximate the argmax function, within the $2k+1$ window.

Therefore,

$$reduction : softmax_{local}(a) \cdot v$$

will approximate the maxpooling function as α grows.