CS 475/675 Machine Learning: Homework 3
Analytical Questions

30 Points Total Version 1.0

Due: Monday March 21, 2022 11:59 pm

Cassandra Parent, Alexandra Szewc (cparent5, aszewc1)

Instructions

We have provided this LaTeX document for turning in this homework. We give you one or more boxes to answer each question. The question to answer for each box will be noted in the title of the box.

Other than your name, do not type anything outside the boxes. Leave the rest of the document unchanged.

Do not change any formatting in this document, or we may be unable to grade your work. This includes, but is not limited to, the height of textboxes, font sizes, and the spacing of text and tables. Additionally, do not add text outside of the answer boxes. Entering your answers are the only changes allowed.

We strongly recommend you review your answers in the generated PDF to ensure they appear correct. We will grade what appears in the answer boxes in the submitted PDF, NOT the original latex file.

Notation

$\mathbf{x}$ One input data vector. $\mathbf{x_i}$ is d dimensional. $\mathbf{x} \in \mathbb{R}^d$.

$\mathbf{W}$ The weight matrix. We ignore bias terms (i.e. $\mathbf{b}$) in this assignment. $\mathbf{W} \in \mathbb{R}^{d \times d}$.

σ Activation functions. It can be any non-linear *element-wise* functions.

$\odot$ Element-wise product, also named as Hadamard product.
If $\mathbf{c} = \mathbf{a} \odot \mathbf{b}$, then $c_i = a_i \cdot b_i$.

$\mathrm{d}\mathbf{x}$ The differential of the variable $\mathbf{x}$, where $\mathbf{x} \in \mathbb{R}^d$.
$\mathrm{d}\mathbf{x} = [\mathrm{d}x_1, \mathrm{d}x_2, \dots]^\top$.

$\frac{\partial \ell}{\partial \mathbf{x}}$ The partial derivative of ℓ over $\mathbf{x} \in \mathbb{R}^d$.
$\frac{\partial \ell}{\partial \mathbf{x}} = \left[\frac{\partial \ell}{\partial x_1}, \frac{\partial \ell}{\partial x_2}, \dots\right]^\top$.
In general, we have $\mathrm{d}\ell = \left(\frac{\partial \ell}{\partial \mathbf{x}}\right)^\top \mathrm{d}\mathbf{x}$.

Notes: In general, a lowercase letter (not boldface), a, indicates a scalar.
A boldface lowercase letter, $\mathbf{a}$, indicates a vector. a_i indicates its i-th element.
A boldface uppercase letter, $\mathbf{A}$, indicates a matrix.
Any additions between vectors are *element-wise*.

1) Activation Functions (15 points)

Various activation functions are used in neural networks. We will consider a few common ones.

For each of the following activation functions, write the first derivative. You do not need to write the derivation.

(1) (2 points) ReLU:

$$\sigma_1(x) = \max(x, 0) \tag{1}$$

$\frac{\partial \sigma_1}{\partial x} = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{if } x > 0 \end{cases}$

If $x = 0$, the derivative is undefined

(2) (2 points) Softplus ($\beta > 0$):

$$\sigma_2(x) = \frac{1}{\beta} \log(1 + \exp(\beta x)) \tag{2}$$

$\frac{\partial \sigma_2}{\partial x} = \frac{e^{\beta x}}{1+e^{\beta x}}$

(3) (2 points) ELU ($\alpha > 0$):

$$\sigma_3(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(\exp(x) - 1), & \text{if } x \leq 0 \end{cases} \tag{3}$$

$$\frac{\partial \sigma_3}{\partial x} = \begin{cases} 1, & \text{if } x > 0 \\ \alpha * e^x, & \text{if } x < 0 \end{cases}$$

If $\alpha \neq 1$, then the derivative is not defined at $x = 0$.
If $\alpha = 1$, then the derivative at $x = 0$ is 1.

(4) (2 points) LogSigmoid:

$$\sigma_4(x) = \log\left(\frac{1}{1 + \exp(-x)}\right) \tag{4}$$

$\frac{\partial \sigma_4}{\partial x} = \frac{1}{e^x + 1}$

(5) (3 points) Softplus is often treated as a "soft" version of the ReLU function. Based on the equations above, and your derivative, explain why this is the case? What are the advantages of Softplus over ReLU?

Softplus is treated as the soft version of ReLU as it softplus is differentiable everywhere and has no conditional change at x = 0. This means that while ReLU has no derivative defined at x=0, there is a derivative at x=0 for softplus. This could be advantageous in backpropogation as the derivative for softplus is defined everywhere. Using softplus could also help eliminate the possibility of vanishing gradients as there is no clear "off" as there is in ReLU. Softplus has a slower descent to zero.

(6) (4 points) Smoothness is an important property for activation functions. We say a function is of class C^0 if it is *continuous everywhere*, and C^1 if it is *differentiable everywhere*. For the activation function we mentioned above, which of them are in C^0? Which of them are in C^1? Briefly explain the reason of possible non-smoothness.

ReLU, Softplus, ELU, and LogSigmoid are C^0

Softplus and LogSigmoid are C^1. ELU will be in C^1 only if $\alpha = 1$.

ReLU is not differentiable everywhere, (not in C^1). The reason this is not smooth is because Relu has a steep change at x=0: it goes from zero to linear immediately creating a "corner." This can be helpful to compute derivatives quickly, but it also means ReLU does not have a derivative at x = 0. ELU (when the parameter $\alpha \neq 1$) has this same corner effect where at zero, the function changes it's definition. This smooth to linear effect once again makes the function not differentiable at the point where the function makes the change (at x = 0).

2)Back Propagation (15 points)

The "skip-connection" structure for neural networks has been proposed as a solution to problems caused by vanishing gradients. Instead of stacking multi-layer perceptrons (MLPs), we impose some "shortcuts" to the network. For example, we can define a network as follows:

$$\begin{aligned}
&\mathbf{z}_1 = \sigma(\mathbf{W}_1\mathbf{x}) \\
&\mathbf{y}_1 = \mathbf{z}_1 + \mathbf{x} \\
&\mathbf{z}_2 = \sigma(\mathbf{W}_2\mathbf{y}_1) \\
&\mathbf{y}_2 = \mathbf{z}_2 + \mathbf{y}_1, \\
&\dots
\end{aligned} \tag{5}$$

where $\mathbf{W}_* \in \mathbb{R}^{d\times d}$, $\mathbf{x} \in \mathbb{R}^d$ are the inputs and $\mathbf{y}_*, \mathbf{z}_* \in \mathbb{R}^d$ are intermediate variables. We have omitted the bias terms for simplicity. Fig 1 shows the structure of this network.

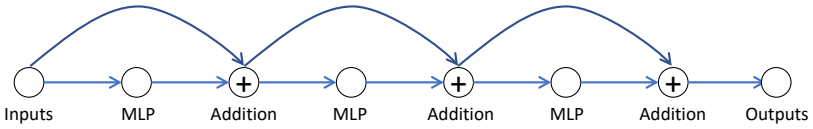

Figure 1: In this picture, "Addition" is element-wise addition between two vectors. "MLP" refers to a multi-layer Perceptron, but can be replaced any complex neural networks.

In this question we will analyze skip-connection network structures to understand why they are effective at addressing the vanishing gradient problem.

(1) (6 points) Let's start by considering a traditional neural network (no skip-connections.) In this network we have a standard MLP, and use a sigmoid function for σ:

$$\mathbf{z} = \sigma\left(\mathbf{W}\mathbf{x}\right). \tag{6}$$

We will use matrix calculus mathematical representations. For scalars ℓ and x, we have

$$\mathrm{d}\ell = \frac{\partial \ell}{\partial x}\mathrm{d}x. \tag{7}$$

If we change x to a vector $\mathbf{x}$, we have

$$\mathrm{d}\ell = \left(\frac{\partial \ell}{\partial \mathbf{x}}\right)^\top \mathrm{d}\mathbf{x}, \tag{8}$$

where we define $\frac{\partial \ell}{\partial \mathbf{x}} = \left[\frac{\partial \ell}{\partial x_1}, \frac{\partial \ell}{\partial x_2}, \dots\right]^\top$ and $\mathrm{d}\mathbf{x} = [\mathrm{d}x_1, \mathrm{d}x_2, \dots]^\top$.

Let $\hat{\mathbf{z}} = \frac{\partial \ell}{\partial \mathbf{z}}$, where ℓ is the value of the loss function. Note that $\hat{\mathbf{z}} \in \mathbb{R}^d$ and $\hat{z}_i = \frac{\partial \ell}{\partial z_i}$. Let's suppose we're doing back-propagation and we have obtained $\hat{\mathbf{z}}$.

Derive the form of $\frac{\partial \ell}{\partial \mathbf{x}}$ as a function of $\mathbf{x}$, $\mathbf{W}$, $\mathbf{z}$, and $\hat{\mathbf{z}}$. [1] The follows may help:

- Because the sigmoid function is element-wise, we have $\mathrm{d}\sigma(\mathbf{x}) = \sigma(\mathbf{x}) \odot (1 - \sigma(\mathbf{x})) \odot \mathrm{d}\mathbf{x}$, where the operator $\odot$ is element-wise production.
- $\mathbf{a}^\top(\mathbf{b} \odot \mathbf{c}) = (\mathbf{a} \odot \mathbf{b})^\top \mathbf{c}$.

[1] The arguments are not independent and it's not necessary to include all of them in your answer.

$$\partial l = \frac{\partial l}{\partial \mathbf{z}}^T \partial \mathbf{z}$$
$$= \hat{\mathbf{z}}^T \partial \mathbf{z}$$
$$= \hat{\mathbf{z}}^T \partial \sigma(\mathbf{W}\mathbf{x})$$
$$= \hat{\mathbf{z}}^T (\mathbf{W} \partial \sigma(\mathbf{x}))$$
$$= \hat{\mathbf{z}}^T (\mathbf{W}((\sigma(\mathbf{x}) \odot (1 - \sigma(\mathbf{x}))) \odot \partial \mathbf{x}))$$
$$= \hat{\mathbf{z}}^T (\mathbf{W} \odot (\sigma(\mathbf{x}) \odot (1 - \sigma(\mathbf{x}))))^T \partial \mathbf{x}$$
$$\frac{\partial l}{\partial \mathbf{x}} = \hat{\mathbf{z}}^T (\mathbf{W} \odot (\sigma(\mathbf{x}) \odot (1 - \sigma(\mathbf{x}))))^T$$

(2) (6 points) We can now consider the "skip-connections" by adding them to the above MLP. Suppose that

$$\mathbf{z} = \sigma(\mathbf{W}\mathbf{x}) \tag{9}$$
$$\mathbf{y} = \mathbf{x} + \mathbf{z}, \tag{10}$$

where σ is again the sigmoid function. $\mathbf{y}$ becomes the input to the next layer of the MLP. Assume we have obtained $\hat{\mathbf{y}} = \frac{\partial \ell}{\partial \mathbf{y}}$ during back-propagation. Derive the form of $\frac{\partial \ell}{\partial \mathbf{x}}$ as a function of $\hat{\mathbf{y}}$, $\mathbf{W}$, $\mathbf{z}$, and $\mathbf{x}$. [2]

[2]The arguments are not independent, and it's not necessary to include all of them in your answer.

$$\partial l = \frac{\partial l}{\partial \mathbf{y}}^T \partial \mathbf{y}$$
$$= \hat{\mathbf{y}}^T \partial \mathbf{y}$$
$$= \hat{\mathbf{y}}^T \partial (\mathbf{x} + \mathbf{z})$$
$$= \hat{\mathbf{y}}^T \partial \mathbf{x} + \hat{\mathbf{y}}^T \partial \mathbf{z}$$

Using the answer from part 1 since pieces of the two computational graphs are equivalent...

$$= \hat{\mathbf{y}}^T \partial \mathbf{x} + \hat{\mathbf{y}}^T (\mathbf{W} \odot (\sigma(\mathbf{x}) \odot (1 - \sigma(\mathbf{x}))))^T \partial \mathbf{x}$$
$$\frac{\partial l}{\partial \mathbf{x}} = \hat{\mathbf{y}}^T + \hat{\mathbf{y}}^T (\mathbf{W} \odot (\sigma(\mathbf{x}) \odot (1 - \sigma(\mathbf{x}))))^T$$
$$\frac{\partial l}{\partial \mathbf{x}} = \hat{\mathbf{y}}^T (1 + (\mathbf{W} \odot (\sigma(\mathbf{x}) \odot (1 - \sigma(\mathbf{x}))))^T)$$

(3) (3 points) Based on your answers showing the gradients to both types of networks, explain why skip-connection networks have become an effective way to handle the vanishing gradient problem.

By imposing some "shortcuts" in the network, we skip places where the gradient would be multiplied. In a normal network, these multiplications would result in smaller and smaller numbers until it reached close to zero causing a "vanishing gradient." Using addition between skipping layers instead of this prevents vanishing gradients. In the answer for 2.2, it is clearly seen that even if the latter half of the sum (the answer to part 2.1) was zero, the update to $\mathbf{x}$ would still be $\hat{\mathbf{y}}^T$. This maintains an update to the network when otherwise it would be zero (as in 2.1).