Machine Learning I: Foundations
Exercise Sheet 5

Prof. Marius Kloft TA:Billy Joe Franks

09.06.2021
Deadline: 08.06.2021

1) (MANDATORY) 10 Points

In this exercise, we will be deriving everything necessary for the gradient of a neural network. Later in the lecture you will see these terms in action. Calculate the following derivatives for $a, y \in \mathbb{R}, \mathbf{x}, \mathbf{b}, \mathbf{w} \in \mathbb{R}^d, W \in \mathbb{R}^{d \times d}$:

a) $\frac{\partial}{\partial W_{i,j}}(W\mathbf{x} + \mathbf{b})$.

$$\frac{\partial}{\partial W_{i,j}}(W\mathbf{x} + \mathbf{b}) = \frac{\partial}{\partial W_{i,j}} W\mathbf{x} = \frac{\partial}{\partial W_{i,j}} \begin{bmatrix} \sum_k W_{1,k}x_k \\ \vdots \\ \sum_k W_{i,k}x_k \\ \vdots \\ \sum_k W_{d,k}x_k \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ x_j \\ \vdots \\ 0 \end{bmatrix}$$

b) $\frac{\partial}{\partial b_i}(W\mathbf{x} + \mathbf{b})$.

$$\frac{\partial}{\partial b_i}(W\mathbf{x} + \mathbf{b}) = \frac{\partial}{\partial b_i}\mathbf{b} = \frac{\partial}{\partial b_i} \begin{bmatrix} b_1 \\ \vdots \\ b_i \\ \vdots \\ b_d \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

c) $\frac{\partial}{\partial x_i}(W\mathbf{x} + \mathbf{b})$.

$$\frac{\partial}{\partial x_i}(W\mathbf{x} + \mathbf{b}) = \frac{\partial}{\partial x_i} W\mathbf{x} = \frac{\partial}{\partial x_i} \begin{bmatrix} \sum_k W_{1,k}x_k \\ \vdots \\ \sum_k W_{d,k}x_k \end{bmatrix} = \begin{bmatrix} W_{1,i} \\ \vdots \\ W_{d,i} \end{bmatrix}$$

d) $\frac{\partial}{\partial a}\sigma(a) = \frac{\partial}{\partial a}\left(1+e^{-a}\right)^{-1}$.

$$\frac{\partial}{\partial a}\left(1+e^{-a}\right)^{-1} = \left(1+e^{-a}\right)^{-2} e^{-a}$$

In addition we note

$$\frac{e^{-a}}{(1+e^{-a})^2} = \sigma(a)\frac{1+e^{-a}-1}{1+e^{-a}} = \sigma(a)\left(1-\frac{1}{1+e^{-a}}\right) = \sigma(a)\left(1-\sigma(a)\right)$$

e) $\frac{\partial}{\partial a}\log\left(1+e^{-ya}\right)$.

$$\frac{\partial}{\partial a}\log\left(1+e^{-ya}\right) = \frac{-ye^{-ya}}{1+e^{-ya}} = -ye^{-ya}\sigma(ya)$$

f) Use the above to calculate $\frac{\partial}{\partial W_{i,j}}\log\left(1+\exp(-y\mathbf{w}^T\hat{\sigma}(W\mathbf{x}+\mathbf{b}))\right)$. $\hat{\sigma}$ is the elementwise application of σ, i.e.

$$\hat{\sigma}\left(\begin{bmatrix}x_1\\x_2\end{bmatrix}\right) = \begin{bmatrix}\sigma(x_1)\\\sigma(x_2)\end{bmatrix}$$

In this exercise we will use the chain rule to simplify the derivation of derivatives.

$$\begin{aligned}
&\frac{\partial}{\partial W_{i,j}}\log\left(1+\exp(-y\mathbf{w}^T\hat{\sigma}(W\mathbf{x}+\mathbf{b}))\right)\\
=&\frac{\partial\log\left(1+\exp(-y\mathbf{w}^T\hat{\sigma}(W\mathbf{x}+\mathbf{b}))\right)}{\partial\mathbf{w}^T\hat{\sigma}(W\mathbf{x}+\mathbf{b}))}\frac{\partial\mathbf{w}^T\hat{\sigma}(W\mathbf{x}+\mathbf{b}))}{\partial\hat{\sigma}(W\mathbf{x}+\mathbf{b}))}\frac{\partial\hat{\sigma}(W\mathbf{x}+\mathbf{b}))}{\partial W\mathbf{x}+\mathbf{b}}\frac{\partial W\mathbf{x}+\mathbf{b}}{\partial W_{i,j}}\\
=&f_e(\mathbf{w}^T\hat{\sigma}(W\mathbf{x}+\mathbf{b}))\mathbf{w}^T\mathrm{diag}(\hat{f}_d(W\mathbf{x}+\mathbf{b}))\begin{bmatrix}0\\\vdots\\x_j\\\vdots\\0\end{bmatrix}
\end{aligned}$$

with

$$f_e(a) := -ye^{-ya}\sigma(ya)$$

$$f_d(a) := \sigma(a)\left(1-\sigma(a)\right)$$

And as above $\hat{f}_d(\mathbf{x})$ is the elementwise application of f_d to $\mathbf{x}$. We note that for optimization we would also have to calculate the derivative according to $\mathbf{b}$, however, this is very simple to adjust now. We just have to replace the last derivative with the one found in b).

2) We will explore activation functions for neural networks. For each function do the following, plot it, state the range of its output (i.e., $f(\mathbb{R})$), derive its gradient, and state an advantage and disadvantage of using it.

a) **TanH**: $f(x) = \tanh(x)$

b) **Error function**: $f(x) = \text{erf}(x)$

c) **ReLU**: $f(x) = \max(0, x)$

d) **Leaky-ReLU**: $f(x) = \max(0.01x, x)$

e) **PLU**: $f(x) = \max(\alpha(x + c) - c, \min[\alpha(x - c) + c, x])$

f) **Sinusoid**: $f(x) = \sin(x)$

g) **Gaussian**: $f(x) = e^{-x^2}$

a) $f(x) = \tanh(x) \in (-1, 1)$

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$
$$f'(x) = 1 - \tanh(x)^2 = \frac{(e^x + e^{-x})^2 - (e^x - e^{-x})^2}{(e^x + e^{-x})^2}$$

In the past this function was very common in deep learning, it is quite similar to the sigmoid. However the sigmoid is in a different range, specifically $\sigma(x) \in (0, 1)$. Overall tanh has the same properties as the sigmoid, except for its range.

b) $f(x) = \text{erf}(x) \in (-1, 1)$ The derivative of the error function follows from its definition.

$$f(x) = \text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-x^2}$$
$$f'(x) = \frac{2}{\sqrt{\pi}} e^{-x^2}$$

This function is extremely similar to the tanh, however in general it is easier to compute, tanh can be computed using sinusoidal tables and approximation, which takes long, or it can be computed using its exponential definition, which takes the approximation of multiple exponentials, which takes long. The error function is rather hard to compute itself, i.e. it is harder to compute than tanh, however its gradient is rather simple and easier to compute than tanh. However tanh is still typically preferred.

c) $f(x) = \max(0, x) \in [0, \infty)$

$$f'(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases}$$

This function is preferred nowadays, because it is easy to compute and has a very simple derivative. The downside is, it is not continuous.

d) $f(x) = \max(0.01x, x) \in (-\infty, \infty)$

$$f'(x) = \begin{cases} 0.01, & \text{if } x < 0 \\ 1 & \text{otherwise} \end{cases}$$

Soon after the ReLU the Leaky-ReLU followed. The 0.01 is replacable by basically any small constant. It solves a problem the ReLU has, which is if a ReLU unit is ever 0 no gradient is propagated through this unit, thus the Leaky-ReLU can be seen as a strict upgrade to the ReLU function.

e) $f(x) = \max(\alpha(x + c) - c, \min[\alpha(x - c) + c, x]) \in (-\infty, \infty)$, if $\alpha \neq 0$

$$f'(x) = \begin{cases} \alpha, & \text{if } x < -c \\ 1 & \text{if } -c < x < c \\ \alpha & \text{otherwise} \end{cases}$$

Generally, this function has similar properties to the ReLU, however it is more similar to the sigmoidal functions. If $\alpha > 0$, then this function is a mix of Leaky-ReLU with sigmoidal functions. This function is not used often, as the non-linearity of ReLU is typically satisfactory. However if the range of ReLU is problematic, then this function can be used.

f) $f(x) = \sin(x) \in [-1, 1]$

$$f'(x) = cos(x)$$

This is a rather odd choice of activation function. Overall it has the advantage that it is scale invariant. Some issues of sigmoidal functions are fixed, i.e. for sigmoidal functions gradients get small as $x \to \infty$, for this function this is not the case. An issue is that gradients should never be too big, because the gradient is only accurate in a very small region.

g) $f(x) = e^{-x^2} \in (0, 1)$

$$f'(x) = -2xe^{-x^2}$$

This is in some sense an inverted activation function, as it activates close to 0, instead of away from 0. As such it can be an interesting choice as an activation function. Overall it is again similar to sigmoidal functions in its properties.

3) The aim of this question is to show the similarities between the hinge loss and logistic loss in optimization. For the SVM problem we considered using gradient descent in slide 10 (Lecture 3.3). We proposed two such methods one using the hinge loss $h(\mathbf{x}_i) = \max(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i))$ and the other using the logistic loss function:

$$l(x_i) = \ln(1 + e^{-\mathbf{w}^\top \Phi(\mathbf{x}_i)}),$$

where $\Phi(\mathbf{x}_i)$ is the output of the output layer.

a) Let $a_1, a_2, \cdots, a_n \in \mathbb{R}^+$, where $a_i \neq a_j$, $a^* = \max(\{a_1, a_2, \cdots, a_n\})$ and $a^{**} = \max(\{a_1, a_2, \cdots, a_n\} \backslash \{a^*\})$. Show that:

$$\lim_{(a^{**}-a^*)\to-\infty} \ln(e^{a_1} + e^{a_2} + \cdots + e^{a_n}) = \max(\{a_1, a_2, \cdots, a_n\}) = a^*$$

$$\begin{aligned}
& \lim_{(a^{**}-a^*)\to-\infty} \ln(e^{a_1} + e^{a_2} + \cdots e^{a_n}) \\
= & \lim_{(a^{**}-a^*)\to-\infty} \ln(\frac{e^{a^*}}{e^{a^*}}(e^{a_1} + e^{a_2} + \cdots + e^{a^*} + \cdots + e^{a_n})) \\
= & \lim_{(a^{**}-a^*)\to-\infty} \ln(e^{a^*}(\frac{e^{a_1}}{e^{a^*}} + \frac{e^{a_2}}{e^{a^*}} + \cdots + \frac{e^{a^*}}{e^{a^*}} + \cdots + \frac{e^{a_n}}{e^{a^*}})) \\
= & \lim_{(a^{**}-a^*)\to-\infty} \ln(e^{a^*}(e^{a_1-a^*} + e^{a_2-a^*} + \cdots + e^{a^*-a^*} + \cdots + e^{a_n-a^*})) \\
=^1 & \ln(e^{a^*}(0 + 0 + \cdots + 1 + \cdots + 0)) \\
= & \ln e^{a^*} \\
= & a^* = \max\{a_1, a_2, \cdots, a_n\}
\end{aligned}$$

The equals sign with [1] is not reasonable as written. In fact it is multiple steps in one. Using the limit of compositions the limit is drawn past ln and then it is pulled into each summand, where the limit is evaluated. The reason for this simplification, is simply space of this sheet.

b) Using item (a), show that the hinge loss can be approximated asymptotically by the calculated limit.

From item (a) we have:

$$\max(a_1, a_2, \cdots, a_n) = \lim_{(a^{**}-a^{*})\to-\infty} \ln\left(e^{a_1} + e^{a^2} + \cdots + e^{a_n}\right)$$

.

We can write the hinge loss as:

$$h(t) = \max(0, t).$$

If $t \to \infty$ then $(0 - t) \to -\infty$.Therefore, asymptotcally

$$h(t) = \ln(e^0 + e^t)$$

In the SVM case:

$$\begin{aligned} h^*(t) &= \ln(e^0 + e^{1-y_i(\mathbf{w}^T x_i)}) \\ &= \ln(1 + e^{1-y_i(\mathbf{w}^T x_i)}) \end{aligned}$$

.

c) You probably noticed that the hinge loss can be approximated by the function $h^*(t) = \ln(1 + e^{1-t})$, where $t = y_i(\mathbf{w}^\top \mathbf{x}_i)$. Compare $h^*(t)$ to $l(t) = \ln(1 + e^{-t})$ (logistic loss).

Essentially $\ln(1+e^{1-t})$ and the hinge-loss fit each other perfectly. The logistic loss on the other hand is shifted to the right in relation to the hinge-loss. This exercise was supposed to show you that indeed the logistic loss is an approximation of the hinge-loss.

4) Solve programming task 5.