IFT 6390 Fundamentals of Machine Learning
Ioannis Mitliagkas

Homework 3 - Theoretical part

- This homework must be done and submitted to Gradescope and can be done in groups of up to 2 students. You are welcome to discuss with students outside of your group but the solution submitted by a group must be its own.
- The theoretical part should be submitted as a pdf file. It should preferably be done in LaTeX by copying this project (`Menu -> Copy Project`) and working off of this exact file. Still all **legible** solutions submitted as a pdf will be accepted. We reserve the right to penalize solutions that are very hard to read, even if they are correct.
- Only one student should submit the homework and you should add your group member on the submission page on gradescope

1. **Derivatives and relationships between basic functions** [15 points]

   We define:

   - The "**logistic sigmoid**" function: $x \mapsto \sigma(x) = \frac{1}{1+\exp(-x)}$.
   - The "**hyperbolic tangent**" function: $x \mapsto \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.
   - The "**softplus**" function: $x \mapsto \text{softplus}(x) = \ln(1 + \exp(x))$
   - The "**sign**" function $x \mapsto \text{sign}(x)$ which returns +1 if its argument is positive, -1 if negative and 0 if 0.
   - The "**Heaviside** step function" function $x \mapsto \text{H}(x)$ which returns +1 if its argument is positive, 0 if negative and $\frac{1}{2}$ if 0.
   - The "**indicator**" function: $x \mapsto \mathbf{1}_S(x)$ which returns 1 if $x \in S$ (or x respects condition S), and otherwise returns 0.
   - The "**rectifier**" function which keeps only the positive part of its argument: $x \mapsto \text{rect}(x)$ returns x if $x \geq 0$ and returns 0 if $x < 0$. It is also named RELU (rectified linear unit): $\text{rect}(x) = \text{RELU}(x) = [x]_+ = \max(0, x) = \mathbf{1}_{\{x>0\}}(x)$
   - The "**diagonal**" function: $\mathbf{x} \in \mathbb{R}^n \mapsto diag(\mathbf{x}) \in \mathbb{R}^{n \times n}$ such that $diag(\mathbf{x})_{ij} = \mathbf{x}_i$ if $i = j$ and $diag(\mathbf{x})_{ij} = 0$ if $i \neq j$. Here $\mathbb{R}^{n \times n}$ is the set of square matrices of size n.
   - The "**softmax**" function $\mathbf{x} \in \mathbb{R}^n \mapsto S(\mathbf{x}) \in \mathbb{R}^n$ such that $S(\mathbf{x})_i = \frac{e^{\mathbf{x}_i}}{\sum_j e^{\mathbf{x}_j}}$

   Please note that in this homework we will sometimes use the partial derivative symbol to differentiate with respect to a vector, $\mathbf{x}$. In those cases, this notation denotes a gradient: $\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \nabla f(\mathbf{x})$.

   (a) [2 points] Write the derivative of the rectified linear unit (ReLU) $g(x) = \max\{0, x\}$, **wherever it exists**. Note: its derivative at 0 is not defined, but your function rect' can return 0 at 0.

(b) [1 points] Write the derivative of the sigmoid function, σ', using the σ function only.

(c) [1 points] Show that $\sigma(x) = \frac{1}{2}\left(\tanh(\frac{1}{2}x) + 1\right)$.

(d) [1 points] Show that $\ln \sigma(x) = -\text{softplus}(-x)$.

(e) [1 points] Show that $\text{softplus}(x) - \text{softplus}(-x) = x$.

(f) [1 points] Write the sign function $\text{sign}(x)$ using only indicator functions.

(g) [1 points] Let the squared L_2 norm of a vector be: $\|\mathbf{x}\|_2^2 = \sum_i \mathbf{x}_i^2$. Write the the gradient of the square of the L_2 norm function, $\frac{\partial \|\mathbf{x}\|_2^2}{\partial \mathbf{x}}$, in vector form.

(h) [1 points] Let the norm L_1 of a vector be: $\|\mathbf{x}\|_1 = \sum_i |\mathbf{x}_i|$. Write the gradient of the L_1 norm function, $\frac{\partial \|\mathbf{x}\|_1}{\partial \mathbf{x}}$, in vector form.

(i) [1 points] Show that softmax is not invariant under scalar multiplication. Let $S_c(\mathbf{x}) = S(c\mathbf{x})$ where $c \geq 0$.

(j) [1 points] Show that softmax is translation-invariant, that is: $S(\mathbf{x} + c) = S(\mathbf{x})$, where c is a scalar constant.

(k) [1 points] Show that the partial derivatives of the softmax function are given by: $\frac{\partial S(\mathbf{x})_i}{\partial \mathbf{x}_j} = S(\mathbf{x})_i \mathbf{1}_{i=j} - S(\mathbf{x})_i S(\mathbf{x})_j$.

(l) [1 points] Express the Jacobian matrix of the softmax function $\frac{\partial S(\mathbf{x})}{\partial \mathbf{x}}$ using matrix-vector notation. Use the *diag* function.

Remember that $\frac{\partial S(\mathbf{x})}{\partial \mathbf{x}}$ is a $n \times n$ matrix, and for all $i, j \in \{1, \ldots, n\}$, the (i, j) entry of the matrix is $\left(\frac{\partial S(\mathbf{x})}{\partial \mathbf{x}}\right)_{i,j} = \frac{\partial S(\mathbf{x})_i}{\partial \mathbf{x}_j}$.

(m) [2 points] Let $\mathbf{y}$ and $\mathbf{x}$ be $n-$dimensional vectors related by $\mathbf{y} = f(\mathbf{x})$, L be a differentiable loss function. According to the chain rule of calculus, $\nabla_{\mathbf{x}} L = (\frac{\partial \mathbf{y}}{\partial \mathbf{x}})^\top \nabla_{\mathbf{y}} L$, which takes up $O(n^2)$ computational time in general (as it requires a matrix-vector multiplication). Show that if $f(\mathbf{x}) = \sigma(\mathbf{x})$ or $f(\mathbf{x}) = S(\mathbf{x})$, the above matrix-vector multiplication can be simplified to a $O(n)$ operation.

Note that here, we used the sigmoid function for a vector input $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_n) \in \mathbb{R}^n \mapsto \sigma(\mathbf{x}) = (\sigma(\mathbf{x}_1), \ldots, \sigma(\mathbf{x}_n))$.

2. **Gradient computation for parameter optimization in a neural net for multiclass classification** [34 points]

Let $D_n = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\}$ be the dataset with $x^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \{1, \ldots, m\}$ indicating the class within m classes. **For vectors and matrices in the following equations, vectors are by default considered to be column vectors.**

Consider a neural net of the type Multilayer perceptron (MLP) with only one hidden layer (meaning 3 layers total if we count the input and output layers). The hidden layer is made of d_h neurons fully connected to the input layer. We shall consider a simple periodic nonlinearity ϕ as an activation function, defined as follows:

$$\phi(x) = \sin(x).$$

The output layer is made of m neurons that are fully connected to the hidden layer. They are equipped with a **softmax** non linearity. The output of the j^{th} neuron of the output layer gives a score for the j-th class which can be interpreted as the probability of x being of class j.

It is highly recommended that you draw the neural net as it helps understanding all the steps.

(a) [2 points]Let $\mathbf{W}^{(1)}$ be a $d_h \times d$ matrix of weights and $\mathbf{b}^{(1)}$ the bias vector be the connections between the input layer and the hidden layer. What is the dimension of $\mathbf{b}^{(1)}$? Give the formula of the pre-activation vector (before the non linearity) of the neurons of the hidden layer $\mathbf{h}^a$ given $\mathbf{x}$ as input, first in a matrix form ($\mathbf{h}^a = \ldots$), and then details on how to compute one element $\mathbf{h}^a_j = \ldots$. Write the output vector of the hidden layer $\mathbf{h}^s$ with respect to $\mathbf{h}^a$.

(b) [2 points] Let $\mathbf{W}^{(2)}$ be a weight matrix and $\mathbf{b}^{(2)}$a bias vector be the connections between the hidden layer and the output layer. What are the dimensions of $\mathbf{W}^{(2)}$ and $\mathbf{b}^{(2)}$? Give the formula of the activation function of the neurons of the output layer $\mathbf{o}^a$ with respect to their input $\mathbf{h}^s$ in a matrix form and then write in a detailed form for $\mathbf{o}^a_k$.

(c) [2 points] The output of the neurons at the output layer is given by:

$$\mathbf{o}^s = \text{softmax}(\mathbf{o}^a)$$

Give the precise equation for $\mathbf{o}^s_k$ as a function of $\mathbf{o}^a_j$. **Show** that the $\mathbf{o}^s_k$ are positive and sum to 1. Why is this important?

(d) [2 points] The neural net computes, for an input vector $\mathbf{x}$, a vector of probability scores $\mathbf{o}^s(\mathbf{x})$. The probability, computed by a neural net, that an observation $\mathbf{x}$ belong to class y is given by the $y^{th} output \mathbf{o}^s_y(\mathbf{x})$. This suggests a loss function such as:

$$L(\mathbf{x}, y) = \text{-log } \mathbf{o}^s_y(\mathbf{x})$$

Find the equation of L as a function of the vector $\mathbf{o}^a$. It is easily achievable with the correct substitution using the equation of the previous question.

(e) [2 points] The training of the neural net will consist of finding parameters that minimize the empirical risk $\hat{R}$ associated with this loss function. What is $\hat{R}$? What is precisely the set θ of parameters of the network? How many scalar parameters n_θ are there? Write down the optimization problem of training the network in order to find the optimal values for these parameters.

(f) [2 points] To find a solution to this optimization problem, we will use gradient descent. What is the (batch) gradient descent equation for this problem?

(g) [3 points]We can compute the vector of the gradient of the empirical risk $\hat{R}$ with respect to the parameters set θ this way

$$\begin{pmatrix} \frac{\partial \hat{R}}{\partial \theta_1} \\ \vdots \\ \frac{\partial \hat{R}}{\partial \theta_{n_\theta}} \end{pmatrix} = \frac{1}{n} \sum_{i=1}^{n} \begin{pmatrix} \frac{\partial L(\mathbf{x}_i, y_i)}{\partial \theta_1} \\ \vdots \\ \frac{\partial L(\mathbf{x}_i, y_i)}{\partial \theta_{n_\theta}} \end{pmatrix}$$

This hints that we only need to know how to compute the gradient of the loss L with an example$(\mathbf{x}, y)$ with respect to the parameters, defined as followed:

$$\frac{\partial L}{\partial \theta} = \begin{pmatrix} \frac{\partial L}{\partial \theta_1} \\ \vdots \\ \frac{\partial L}{\partial \theta_{n_\theta}} \end{pmatrix} = \begin{pmatrix} \frac{\partial L(\mathbf{x}, y)}{\partial \theta_1} \\ \vdots \\ \frac{\partial L(\mathbf{x}, y)}{\partial \theta_{n_\theta}} \end{pmatrix}$$

We shall use **gradient backpropagation**, starting with loss L and going to the output layer $\mathbf{o}$ then down the hidden layer $\mathbf{h}$ then finally at the input layer $\mathbf{x}$.

Show that

$$\frac{\partial L}{\partial \mathbf{o}^a} = \mathbf{o}^s - onehot_m(y)$$

Note: Start from the expression of L as a function of $\mathbf{o}^a$ that you previously found. Start by computing $\frac{\partial L}{\partial \mathbf{o}^a_k}$ for $k \neq y$ (using the start of the expression of the logarithm derivative). Do the same thing for $\frac{\partial L}{\partial \mathbf{o}^a_y}$.

IMPORTANT: From now on when we ask to "compute" the gradients or partial derivatives, you only need to write them as function of previously computed derivatives (do not substitute the whole expressions already computed in the previous questions)!

(h) [3 points] Compute the gradients with respect to parameters $\mathbf{W}^{(2)}$ and $\mathbf{b}^{(2)}$ of the output layer. Since L depends on $\mathbf{W}^{(2)}_{kj}$ and $\mathbf{b}^{(2)}_k$ only through $\mathbf{o}^a_k$ the result of the chain rule is:

$$\frac{\partial L}{\partial \mathbf{W}^{(2)}_{kj}} = \frac{\partial L}{\partial \mathbf{o}^a_k} \frac{\partial \mathbf{o}^a_k}{\partial \mathbf{W}^{(2)}_{kj}}$$

and

$$\frac{\partial L}{\partial \mathbf{b}^{(2)}_k} = \frac{\partial L}{\partial \mathbf{o}^a_k} \frac{\partial \mathbf{o}^a_k}{\partial \mathbf{b}^{(2)}_k}$$

(i) [2 points] Write down the gradient of the last question in matrix form and define the dimensions of all matrix or vectors involved. (For the gradient with respect to $\mathbf{W}^{(2)}$, we want to arrange the partial derivatives in a matrix that has the same shape as $\mathbf{W}^{(2)}$, and that's what we call the gradient)

What are the dimensions?

Take time to understand why the above equalities are the same as the equations of the last question.

(j) [2 points] What is the partial derivative of the loss L with respect to the output of the neurons at the hidden layer? Since L depends on $\mathbf{h}^s_j$ only through the activations of the output neurons $\mathbf{o}^a$ the chain rule yields:

$$\frac{\partial L}{\partial \mathbf{h}^s_j} = \sum_{k=1}^{m} \frac{\partial L}{\partial \mathbf{o}^a_k} \frac{\partial \mathbf{o}^a_k}{\partial \mathbf{h}^s_j}$$

(k) [2 points]Write down the gradient of the last question in matrix form and define the dimensions of all matrix or vectors involved.

What are the dimensions?

Take time to understand why the above equalities are the same as the equations of the last question.

(l) [2 points] What is the partial derivative of the loss L with respect to the activation of the neurons at the hidden layer? Since L depends on the activation $\mathbf{h}^a_j$ only through $\mathbf{h}^s_j$ of this neuron, the chain rule gives:

$$\frac{\partial L}{\partial \mathbf{h}^a_j} = \frac{\partial L}{\partial \mathbf{h}^s_j} \frac{\partial \mathbf{h}^s_j}{\partial \mathbf{h}^a_j}$$

Note $\mathbf{h}^s_j = \phi(\mathbf{h}^a_j)$: the activation function is applied element-wise. Start by writing the derivative of the function $\frac{\phi(z)}{\partial z} = \phi'(z) = \ldots$.

(m) [2 points] Write down the gradient of the last question in matrix form and define the dimensions of all matrix or vectors involved.

(n) [2 points] What is the gradient with respect to the parameters $\mathbf{W}^{(1)}$ and $\mathbf{b}^{(1)}$ of the hidden layer? **Hint: same logic as a previous question.**

(o) [2 points] Write down the gradient of the last question in matrix form and define the dimensions of all matrix or vectors involved.

Hint: same logic as a previous question.

(p) [2 points] How can you write the partial derivatives of the loss L with respect to $\mathbf{x}$ as the forward pass of a network of the same architecture but different parameters?

Hint: same logic as a previous question.

3. **Convolutional Neural Networks** [11 points]

A Convolutional Neural Network (ConvNet/CNN) is a neural network model which can take in an input image, assign learnable weights and biases to various aspects/objects in the image and be able to recognize different features in the input. The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels). After passing an image through a convolutional layer, it becomes abstracted to a feature map, with shape (number of images) $\times$ (feature map height) $\times$ (feature map width) $\times$ (feature map channels). Convolutions can be represented as a sparse matrix multiply, here are the concepts you need for this question:

- **Kernel**: It's usually known as a feature map or a convolutional filter defined by a width and height.
- **Stride** (s_i): Distance between two consecutive positions of the kernel along axis i.
- **Zero-padding** (p_i): Number of zeros concatenated at the beginning and at the end of an axis) along axis i.
- **Dilation** (d_i): It's used in Dilated convolutions which "inflate" the kernel, inserting spaces between the kernel elements along axis i.

Consider a convolutional neural network. Assume the input is a black and white image of size 32×32. The first layer convolves 6, 5×5 kernels with the input, using a stride of 1 and no padding. The second layer downsamples the output of the first layer with a 2×2 non-overlapping average pooling. The third layer convolves 16, 5×5 kernels with a stride of 1 and a zero-padding of size 0 on each border.

(a) [3 points] What is the dimensionality (scalar) of the output of the last layer?

(b) [2 points] Not including the biases, how many parameters are needed for the last layer?

Assume we are given data of size $64 \times 64 \times 3$. Provide a correct configuration of a convolutional neural network layer that satisfies the specified assumption. Answer with the window size of kernel (k), stride (s), padding (p), and dilation (d, with convention $d = 1$ for no dilation).Use square windows only (e.g. same k for both width and height). The output shape of the last layer is $(64, 6, 6)$.

(c) [2 points] Assume we are not using padding or dilation.

(d) [2 points] Assume $d = 2$, $p = 1$.

(e) [2 points] Assume $p = 1$, $d = 1$.