Autonomous Vision Group
Prof. Dr.-Ing. Andreas Geiger

University of Tübingen
MPI for Intelligent Systems
Autonomous Vision Group

Deep Learning

Exercise 1 – Intro to EDF

Release date: November 4, 2020 **Submission deadline: November 22, 2020**

This assignment may be submitted as an individual or in groups of up to 2 students. Write down the names and student IDs of all group members on top of the first page which you are submitting. However, note that each group member has to submit the solution separately to ILIAS. **By submitting, all group members affirm that they have solved this assignment on their own.**

Please upload your solutions as an archive named `firstname-lastname.zip` to ILIAS. The archive should contain a single `.pdf` file with all the solutions to the handwritten pen and paper exercises. You can use a smartphone app like CamScanner to bundle all sheets into a single PDF document. Alternatively, you can also use Latex to answer the pen and paper questions. The archive should further contain the jupyter notebooks `vector_derivative_product.ipynb`, `linear_regression.ipynb` and `logistic_regression.ipynb` with your solutions. Other files (e.g. datasets) should not be part of the archive. **Please save the notebooks after executing all cells.** This saves us a lot of time, because like this we do not need to rerun your notebooks.

1.1 Derivatives (1 + 1 + 1 + 1 + 1 + 1 + 2 + 2 = 10 Points)

a) Let $\mathbf{f}(\mathbf{x}) = (e^{x_1}\sin(x_3), \sqrt{x_2 + x_3}), \mathbf{f}: \mathbb{R}^3 \mapsto \mathbb{R}^2$. Calculate the matrix of partial derivatives $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x})$.

b) Let $a: \mathbb{R}^1 \mapsto \mathbb{R}^1$ and define $\mathbf{f}(\mathbf{x}) = (a(x_1), \ldots, a(x_n)), \mathbf{f}: \mathbb{R}^n \mapsto \mathbb{R}^n$. Calculate the matrix of partial derivatives $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x})$. What property has the resulting matrix?

c) Let $\sigma(x) = \frac{1}{1+e^{-x}}, \sigma: \mathbb{R}^1 \mapsto \mathbb{R}^1$. Calculate $\sigma'(x)$.

d) Let $\mathbf{f}$ be like in **b)** with $a = \sigma$. Calculate the matrix of partial derivatives $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x})$.

e) Let $\mathbf{W} \in \mathbb{R}^{m \times n}$, $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} = \mathbf{W}\mathbf{x}$.

i) Calculate the matrix of partial derivatives $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$.

ii) Calculate $\frac{\partial y_k}{\partial w_{ij}}$ for $k \in \{1, \ldots, m\}, i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, n\}$.

iii) Let $\mathbf{A} \in \mathbb{R}^{1 \times m}$. Simplify $\hat{w}_{ij} = \mathbf{A}\frac{\partial \mathbf{y}}{\partial w_{ij}}$.

iv) Implement the function `vector_derivative_product` in the jupyter notebook `vector_derivative_product.ipynb` that takes $\mathbf{A}$ and $\mathbf{x}$ as numpy arrays as input and returns the matrix $\hat{\mathbf{W}}$ (with elements $\hat{w}_{ij}$) as numpy array. Find a solution based on broadcasting instead of using `for` loops.

1.2 Linear Regression (2 + 3 + 4 = 9 Points)

a) The loss function $\mathcal{L}$ of a Linear Regression model with a scalar input and scalar output is given by:

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N}(w_0 + w_1 x_i - y_i)^2$$

In this simple 1D case, we are fitting a straight line to the dataset. This line is determined by its offset along the y axis w_0 and its slope w_1. Compute the partial derivative of the loss wrt. the y intercept $\frac{\partial \mathcal{L}}{\partial w_0}$ and the partial derivative of the loss wrt. the slope $\frac{\partial \mathcal{L}}{\partial w_1}$.

b) In the jupyter notebook `linear_regression.ipynb` an almost complete code for loading a dataset and training a Linear Regression model is given. Merely the function `compute_derivatives(x, y, slope, offset_y)` needs to be implemented. Here, `x` and `y` are equally sized numpy arrays containing the inputs $(x_1, x_2, \ldots)$ and outputs $(y_1, y_2, \ldots)$. `slope` corresponds to w_1 and `offset_y` corresponds to w_0. The function should return a tuple of float values containing the derivatives wrt. to the parameters: $(\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_0})$. Implement the function and verify it by training a Linear Regression model and checking whether the plotted line fits the dataset.

c) `linear_regression.ipynb` also contains almost complete code for training a Linear Regression model with help of the Educational Framework (EDF). Merely the computational node `L2Loss` needs to be implemented and the computational graph needs to be defined. The node `L2Loss` computes the *elementwise* squared error between its two input nodes `y` and `y_hat`. Since EDF operates on whole batches, which means that multiple inputs are processed simultaneously, `y` and `y_hat` are $B \times 1$ matrices, where B is the batch size. For this exercise, we consider the entire dataset as one batch, thus $B = N$. Since this node computes the elementwise squared error, the node's output should also be a $B \times 1$ matrix. In contrast to exercise **b)** it is not necessary to compute the mean along the dataset/batch dimension, which is implicitly taken care of by the EDF. Implement the `L2Loss` node including the forward and the backward pass and define the computation graph for the Linear Regression model. Again you can verify your solution by training the model and checking whether the plotted line fits the dataset.

1.3 Computational Graphs and Backpropagation (2 + 2 + 2 + 2 = 8 Points)

Complete the following exercises for each of these expressions:

$$\mathcal{A} = \frac{e^a}{\sqrt{b}} log(a + b)$$

$$\mathcal{B} = \sqrt{a + b + c^2} + \log(a + b + c^2) + \frac{a + b + c^2}{bc^2}$$

$$\mathcal{C} = \sum_{i=1}^{3} (w_0 + w_1 x_i - y_i)^2$$

a) Introduce intermediate variables $(t_1, t_2, t_3, \ldots)$ such that there is a single mathematical operation per assignment. Reuse intermediate variables whenever possible.

b) Draw the corresponding computation graph based on the variables you introduced in **a)**.

c) For each node in your computation graph, write down the products of partial derivatives (e.g. $\frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial x}$) that are computed by this node during backpropagation. For each such product indicate to which node the results are sent to (e.g., by writing down the products close to the corresponding graph edges as we did it in the lecture). Note: you do not need to symbolically calculate or simplify the derivative for specific operations.

d) At each node, write down the sum of partial derivative products that calculates the derivative of the output node wrt. to this node. Your expressions only need to contain the direct children for the node and again you do not need to symbolically calculate or simplify the derivatives.

1.4 Logistic Regression (4 + 2 + 6 + 1 = 13 Points)

a) Using the chain rule, show that

$$\nabla_{\mathbf{w}} \frac{1}{B} \sum_{i=1}^{B} \mathcal{L}(y_i, \hat{y}_i) = \frac{1}{B} \sum_{i=1}^{B} (\hat{y}_i - y_i) \mathbf{x}_i$$

where

$$\mathcal{L}(y, \hat{y}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$
$$\hat{y}_i = \sigma(\mathbf{w}^\top \mathbf{x}_i)$$

b) In the jupyter notebook `logistic_regression.ipynb` an almost complete code for loading the MNIST handwritten digit dataset and training a Logistic Regression model is given. Merely the function `compute_derivatives(x, y, y_hat)` needs to be implemented. This function should return the gradient wrt. to the parameters $\nabla_{\mathbf{w}} \frac{1}{B} \sum_{i=1}^{B} \mathcal{L}(y_i, \hat{y}_i)$ as a numpy array. The argument `x` is a $B \times D$ matrix, where row `x[i]` contains the input vector $\mathbf{x}_{i+1}$, B is the batch size and D is the number of input features. `y` and `y_hat` are vectors of size B. `y[i]` and `y_hat[i]` contain the scalars y_{i+1} and $\hat{y}_{i+1}$, respectively. Verify your solution by training a Logistic Regression model and checking whether the test error drops during the course of training.

Remark: In the preprocessing part of `logistic_regression.ipynb`, four different versions of the MNIST dataset are created. Beside the original dataset, which has a resolution of 28×28 pixels, downscaled versions with a resolution of 1×1 (`x_train_mean`, `x_test_mean`), 4×4 (`x_train_4x4`, `x_test_4x4`) and 8×8 (`x_train_8x8`, `x_test_8x8`) pixels are created. After downscaling, the pixel matrices are flattened into a vector. The code for this exercise is set up to use the 1×1 pixel version of the dataset. Your implementation of the function `compute_derivatives` is not affected by this as it should handle a variable number of input features D.

c) `logistic_regression.ipynb` also contains almost complete code for training a Logistic Regression model with help of the EDF. EDF already implements the required `CrossEntropyLoss` node. For binary classification this node expects as input a matrix `p` of size $B \times 2$ where each entry `p[i]` corresponds to a probability vector with entries `p[i][0]` $= P(Y = 0|X = \mathbf{x}_i)$ and `p[i][1]` $= P(Y = 1|X = \mathbf{x}_i)$. In contrast, we defined our logistic regression model to only output a single probability $P(Y = 1|X = \mathbf{x}_i)$, which is sufficient since the other probability can be deduced via the relationship $P(Y = 0|X = \mathbf{x}_i) = 1 - P(Y = 1|X = \mathbf{x}_i)$. However, to make use of EDF's `CrossEntropyLoss` we need to provide the other probability as well. To this end, we define the computation node `SingleProbToProbVector` that takes as input a $B \times 1$ matrix containing only the probabilities $P(Y = 1|X = \mathbf{x}_i)$ and returns a $B \times 2$ matrix containing both $P(Y = 1|X = \mathbf{x}_i)$ and $P(Y = 0|X = \mathbf{x}_i)$ for each $i \in \{1, \ldots, b\}$.
More concretely, let `z` be the $B \times 1$ matrix that the node receives as input and let `o` be the $B \times 2$ matrix the node outputs, then `o[i][0]` $=$ `z[i][0]` and `o[i][1]` $= 1$ - `z[i][0]` for each `i` $\in \{0, \ldots, B-1\}$. The code for this exercise is also set up to use the 1×1 version of the dataset. Implement the forward and backward pass of the `SingleProbToProbVector` node and define the computation graph. Again you can verify your implementation by training a Logistic Regression model for the 1×1 dataset. A similar test error like in the previous exercise **b)** should be achieved.

d) Use the EDF to train Logistic Regression models with the 4×4, 8×8 and 28×28 pixel versions of the dataset. Use the cells at the end of `logistic_regression.ipynb` for these additional experiments.