Problem set 1
Deep Learning E1394

Carmen Garro, Kai Foerster

Out on Sept 18
Due on Oct 2, 23:59

Submit your written answers as a pdf typed in LaTeXtogether with your code. Submit one answer per group (as assigned on Moodle) and include names of all group members in the document. Round answers to two decimal places as needed. Include references to any external sources you have consulted (points are deducted if those were used but not cited). See "Submission" at the bottom of the problem set for more details on how to submit using Github classroom.

1 Theoretical part

These are some recap and refresher problems to get you up to speed with the mathematics and statistical learning that is behind deep learning, as well as problems that help you work out how neural networks are trained.

1.1 Optimization

Let $f : \mathbb{R}^2 \to \mathbb{R}$ be defined by $f(x, y) = x^2y + xy^2 - 6xy$. Identify and classify all critical points of f.

Solution:

$$f(x, y) = x^2y + xy^2 - 6xy$$

$$\frac{\partial f(x,y)}{\partial x} = 2xy + y^2 - 6y = y(2x + y - 6)$$

$$\frac{\partial f(x,y)}{\partial y} = x^2 + 2xy - 6x = x(x + 2y - 6)$$

$$\frac{\partial^2 f(x,y)}{\partial x^2} = 2y$$

$$\frac{\partial^2 f(x,y)}{\partial y^2} = 2x$$

$\frac{\partial^2 f(x,y)}{\partial x \partial y} = 2x + 2y - 6$

$\nabla f(x, y) = (2xy + y^2 - 6y, x^2 + 2xy - 6x)$

$$H = \begin{pmatrix} 2y & 2x + 2y - 6 \\ 2x + 2y - 6 & 2x \end{pmatrix}$$

Setting both derivatives to zero:

$$\begin{cases} 2xy + y^2 - 6y = y(2x + y - 6) = 0 \\ x^2 + 2xy - 6x = x(x + 2y - 6) = 0 \end{cases}$$

Because in both terms of the gradient, there are and x or y multiplying, then we know that with either being zero we will get a critical point.

1.$(x, y) = (0, 0)$

Also, by setting $y = o$ and $x = 0$, separately we also get more critical points.

2.$(x, y) = (6, 0)$ This is obtained by replacing y with zero in the second term of the derivative with respect to x.

3.$(x, y) = (0, 6)$ This is obtained by replacing x with zero in the second term of the derivative with respect to y.

4.$(x, y) = (2, 2)$ This is obtained with the system of equations when x and y are not zero:

$$\begin{cases} y(2x + y - 6) = 0 \\ x(x + 2y - 6) = 0 \end{cases}$$

$$\begin{cases} 2x + y = 6 \\ x + 2y = 6 \end{cases}$$

The first term minus the second times 2 will give:

$$\begin{cases} 2x + y = 6 \\ -2(x + 2y) = -2(6) \end{cases}$$

$$-3y = -6$$

$$y = 2$$

Then we substitute in $2x + y = 6$ we get $2x + 2 = 6$ and $2x = 4$, $x = 2$

Now, with regard to what type of points these are, we have to evaluate the determinant in the Hessian at each critical point:

$$D = f_{xx} * f_{yy} - f_{xy}^2$$
$$= 2y2x - (2x + 2y - 6)^2$$
$$= 4xy - (2x + 2y - 6)^2$$

2

1.$(x, y) = (0, 0)$

$$D_{(0,0)} = 4 * 0 * 0 - (2 * 0 + 2 * 0 - 6)^2$$
$$= 0 - (-6)^2$$
$$= -36$$

Because $-36 < 0$ then it is a saddle point.
2.$(x, y) = (6, 0)$

$$D_{(6,0)} = 4 * 6 * 0 - (2 * 6 + 2 * 0 - 6)^2$$
$$= 0 - (12 - 6)^2$$
$$= -36$$

And $f_{(x,x)}(6, 0) = 2y = 0$

Because $-36 < 0$ then it is a saddle point.
3.$(x, y) = (0, 6)$

$$D_{(0,6)} = 4 * 0 * 6 - (2 * 0 + 2 * 6 - 6)^2$$
$$= 0 - (12 - 6)^2$$
$$= -36$$

Because $-36 < 0$ then it is a saddle point.
3.$(x, y) = (2, 2)$

$$D_{(2,2)} = 4 * 2 * 2 - (2 * 2 + 2 * 2 - 6)^2$$
$$= 16 - (4 + 4 - 6)^2$$
$$= 16 - (4)$$
$$= 12$$

And $f_{xx}(2, 2) = 2y = 4$

Because $12 > 0$ and the second derivative with respect to x is greater than 0, then it is a local minimum.

1.2 Activation functions

Compute the gradient of the function

$$f(b, w) = ReLU(b + xw) = \begin{cases} 0 & \text{if } b + xw < 0 \\ b + xw & \text{if } b + xw \geq 0 \end{cases} \tag{1}$$

with respect to the parameters b and w, with $x, b, w \in \mathbb{R}$.

Solution:

The gradient for the ReLU function will be dependent on the value that $b + xw$ takes, when $b + xw < 0$ then $\frac{\partial f(b,w)}{\partial b} = 0$ and $\frac{\partial f(b,w)}{\partial w} = 0$. This happens because we are taking the derivative of the constant 0.

$\nabla f(b, w) = (0, 0)$ when $b + wx < 0$

When $b + xw \geq 0$ then the derivatives become $\frac{\partial f(b,w)}{\partial b} = 1$ and $\frac{\partial f(b,w)}{\partial w} = x$

$\nabla f(b, w) = (1, x)$ when $b + wx \geq 0$

Note however that when $b + wx = 0$, the derivative of ReLu is undefined because left-hand limit (0) is different from the right-hand limit.

1.3 Overfitting

This question will test your general understanding of overfitting as it relates to model complexity and training set size. Consider a continuous domain and a smooth joint distribution over inputs and outputs, so that no test or training case is ever duplicated exactly. *The graphs in your solutions should be drawn on paper and then included as a picture. In your answers, describe how your graphs should be read.*

1.3.1 Error rate versus dataset size

Sketch a graph of the typical behavior of training error rate (y-axis) as a function of training set size (x-axis). Add to this graph a curve showing the typical behavior of test error rate versus training set size, on the same axes. (Assume that we have an infinite test set drawn independently from the same joint distribution as the training set). Indicate on your y-axis where zero error is and draw your graphs with increasing error upwards and increasing training set size rightwards.
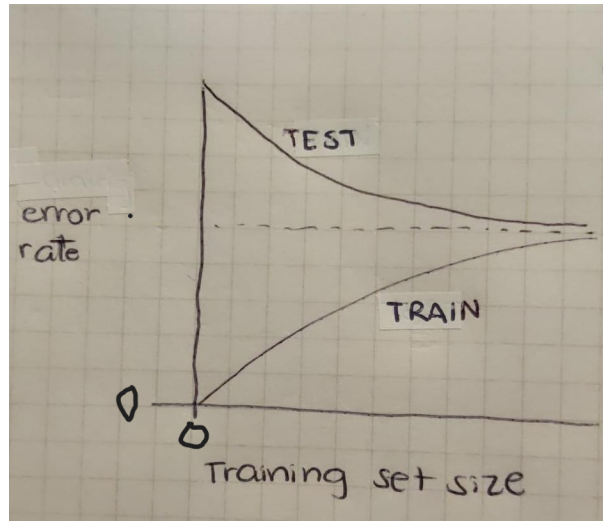
As the number of data points increases, the train error will increase from the point where there are more data points than model features. It will not increase indefinitely, instead it will plateau at some point where the training set size is sufficiently large. On the other hand, the test error will be larger if there are fewer observations in the training set size because the model overfits the data, but it will decrease as more data points are added.

### 1.3.2   Error rate versus model complexity

For a fixed training set size, sketch a graph of the typical behavior of training error rate (y-axis) versus model complexity (x-axis). Add to this graph a curve showing the typical behavior of the corresponding test error rate versus model complexity, on the same axes (again on an IID infinite test set). Show where on the x-axis you think is the most complex model that your data supports (mark this with a vertical line). Choose the x-range so that this line is neither on the extreme left nor on the extreme right. Indicate on your vertical axis where zero error is and draw your graphs with increasing error upwards and increasing complexity rightwards.
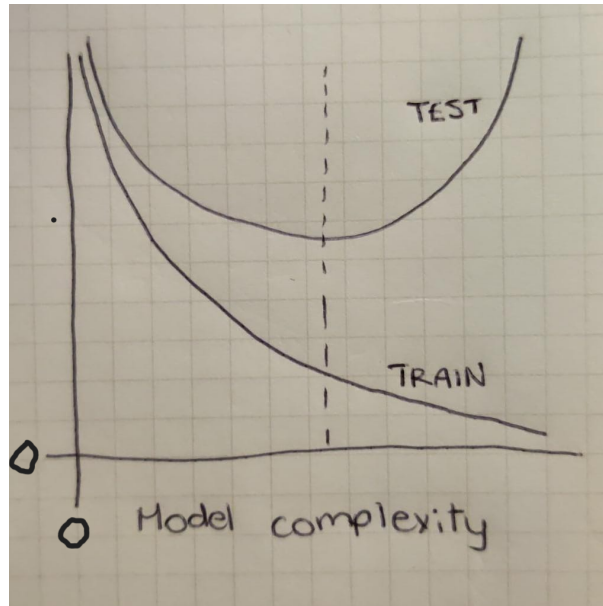
As the model becomes more complex it fits the data better and better, but at a lower growth rate given there is less and less training error to minimise as the model becomes more complex, until finally a complex model fits the training data perfectly. The test error decreases at first with model complexity because the reduction in error coming from a reduced bias is larger than the increase in error coming from an increased variance. As the complexity of the model increases, there is a point where the reduction in bias reduces error by less than the increase in variance. This is the minimum of the error curve, after which it starts sloping up again.

1.3.3 Training epochs

Use a similar graph to illustrate the error rate as a function of training epochs in neural networks. One of the commonly used regularization methods in neural networks is early stopping. Describe (also using the graph) how early stopping is applied, and argue qualitatively why (or why not) early stopping is a reasonable regularization metric.
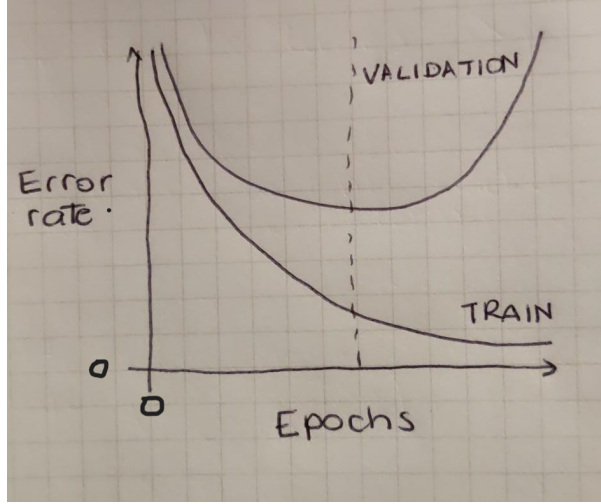
Early stopping works by setting a threshold on how close the gradient of the validation loss function can go to zero (potentially specifying the number of repetitions (consecutively) that the gradient can be below a threshold) before we stop training. It is a reasonable regularization method because when the validation loss plateaus, the gradient reaches a close to zero point. At this point, we can assume that reduction in bias through training further will decrease validation loss by no more than the effect of further training increasing variance. Hence, early stopping prevents over fitting, while allowing for sufficient training epochs to avoid under-fitting.

The drawback of the early stopping method is that it might result in stopping the training in a local minimum.

1.4 Capacity of neural network

We have a single hidden-layer neural network with two hidden units, and data with 2 features $X = (x_1, x_2)$. Design an activation function $g(\cdot)$ applied in the hidden layer, and an output activation function $o(\cdot)$ applied in the output layer, as well as a set of values of the weights that will create a logistic regression model where the input to the logistic function is of the form: $\beta_0 + \beta_1 x_1 x_2$.

Solution:

A single hidden-layer neural network with two hidden units and data with two features can be expressed as following

$$f(X) = \sigma(b^{[2]} + \sum_i^{H=2} w_i^{[2]} g(b_i^{[1]} + \sum_j^{d=2} w_{ij}^{[1]} x_j)) \tag{2}$$

Where:
$g(z) = (z)^2$ (quadratic activation function)
$b_1^{[1]} = 0, w_{11}^{[1]} = 1, w_{12}^{[1]} = 1$

7

$b_2^{[1]} = 0, w_{21}^{[1]} = 1, w_{22}^{[1]} = -1$
$\sigma(z) = \frac{1}{1+e^{-z}}$ (sigmoid / logistic activation function)
$b^{[2]} = \beta_0, w_1^{[2]} = \frac{1}{4}\beta_1, w_2^{[2]} = -\frac{1}{4}\beta_1$

Putting it together gives us:

$$
\begin{aligned}
\text{f(x)} &= \sigma(\beta_0 + \tfrac{1}{4}\beta_1(0 + 1 \times x_1 + 1 \times x_2)^2 - \tfrac{1}{4}\beta_1(0 + 1 \times x_1 - 1 \times x_2)^2) \\
&= \sigma(\beta_0 + \tfrac{1}{4}\beta_1(x_1 + x_2)^2 - \tfrac{1}{4}\beta_1(x_1 - x_2)^2) \\
&= \sigma(\beta_0 + \tfrac{1}{4}\beta_1(x_1^2 + 2x_1x_2 + x_2^2) - \tfrac{1}{4}\beta_1(x_1^2 - 2x_1x_2 + x_2^2)) \\
&= \sigma(\beta_0 + \tfrac{1}{4}\beta_1 x_1^2 + \tfrac{1}{2}\beta_1 x_1 x_2 + \tfrac{1}{4}\beta_1 x_2^2 - \tfrac{1}{4}\beta_1 x_1^2 + \tfrac{1}{2}\beta_1 x_1 x_2 - \tfrac{1}{4}\beta_1 x_2^2) \\
&= \sigma(\beta_0 + \beta_1 x_1 x_2) \\
&= \tfrac{1}{1+e^{-(\beta_0 + \beta_1 x_1 x_2)}}
\end{aligned}
$$

which is a logistic regression of the form: $\beta_0 + \beta_1 x_1 x_2$.

1.5 Neural network theory

Derive the weight updates in gradient descent for a neural network with 2 hidden layers (superscripts [1] and [2]) that each have $H^{[1]}$ and $H^{[2]}$ hidden units respectively. The output layer has superscript [3], and we want to classify the data into K classes.

The output of the network for classes $k = 1, 2, ..., K$ and one data point $x \in \mathbb{R}^d$ can be written as such:

$$f_k(\theta; X) = o(a_k^{[3]}) = o(b_k^{[3]} + \sum_{m=1}^{H^{[2]}} w_{km}^{[3]} h_m^{[2]}) \tag{3}$$

$$h_m^{[2]} = \sigma\left(a_m^{[2]}\right) = \sigma\left(b_m^{[2]} + \sum_{i=1}^{H^{[1]}} w_{mi}^{[2]} h_i^{[1]}\right) \tag{4}$$

$$h_i^{[1]} = \sigma\left(a_i^{[1]}\right) = \sigma\left(b_i^{[1]} + \sum_{j=1}^{d} w_{ij}^{[1]} x_j\right), \tag{5}$$

where θ indicates the vector of all weights and biases.

While this is typically not recommended, in this exercise we use a a quadratic loss function for classification. We use a softmax activation function at the output layer and a ReLU activation function at the hidden layers. Derive the the weight update for the weights of the first hidden layer $w_{ij}^{[1]}$. Start by stating the gradient descent weight update for the $(r+1)^{th}$ iteration as a function of the $(r)^{th}$ iteration, and then compute the partial derivative needed in the update.

Show all the steps in your derivation. You may use the derivatives for activation functions introduced in class. There is no need to show the derivations

of those.

Solution:

$$w_{ij}^{[1](r+1)} = w_{ij}^{[1](r)} + \eta \Delta w_{ij}^{[1](r)} = w_{ij}^{[1](r)} + \eta \left(\frac{\partial L(f(x), y)}{\partial w_{ij}^{[1]}} \right)^{(r)}$$

Quadratic loss function: $L(f(x), y) = \sum_{k=1}^{k} (y_{jk} - f_k(x_j))^2$ (loss for one data point)

$$\sum_{j=1}^{N} \sum_{k=1}^{N} (x_{jk} - f_k(x_j))^2 \quad \text{(for N data point)}$$

$$f_k(X) = \sigma_1 \left(b_k^{[3]} + \sum_{m=1}^{H^{[2]}} w_{km}^{[3]} \sigma_2 \left(b_m^{[2]} + \sum_{i=1}^{H^{[1]}} w_{mi}^{[2]} \sigma_3 \left(b_i^{[1]} + \sum_{j=1}^{d} w_{ij}^{[1]} x_j \right) \right) \right)$$

$$\left(\frac{\partial L(f(X), y)}{\partial w_{ij}^{[1]}} \right) = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial a_k^{[3]}} \cdot \frac{\partial a_k^{[3]}}{\partial h_m^{[2]}} \cdot \frac{\partial h_m^{[2]}}{\partial a_m^{[2]}} \cdot \frac{\partial a_m^{[2]}}{\partial h_i^{[1]}} \cdot \frac{\partial h_i^{[1]}}{\partial a_i^{[1]}} \cdot \frac{\partial a_i^{[1]}}{\partial w_{ij}^{[1]}}$$

$$\frac{\partial L}{\partial f} = -2 \sum_{k=1}^{K} (y_k - f_k(x))$$

Using Kronechurdelta and $o_1(a_k^{[3]}) = \frac{e^{a_k^{[3]}}}{\sum_{n=1}^{k} e^{a_k^{[3]}}}$:

$$\frac{\partial f}{\partial a_k^{[3]}} = \sigma_1(a_k^{[3]})(\delta_{kn} - \sigma_1(a_n^{[3]}))$$

$$\frac{\partial a_k^{[3]}}{\partial h_m^{[2]}} = \frac{\partial}{\partial h_m^{[2]}} \left(b_k^{[3]} + \sum_{m=1}^{H^{[2]}} w_{km}^{[3]} h_m^{[2]} \right) = w_{km}^{[3]}$$

$$\frac{\partial h_m^{[2]}}{\partial a_m^{[2]}} = \frac{\partial \sigma_2(a_m^{[2]})}{\partial a_m^{[2]}} =$$

$$\begin{cases} 0 & \text{if} \quad a_m^{[2]} < 0 \\ a_m^{[2]} & \text{otherwise} \end{cases}$$

Note: if $a_m^{[2]} = 0 \implies NDF$

$$\frac{\partial a_m^{[2]}}{\partial h_i^{[1]}} = \frac{\partial}{\partial h_i^{[1]}} \left(b_m^{[2]} + \sum_{i=1}^{H^{[1]}} w_{mi}^{[2]} h_i^{[1]} \right) = w_{mi}^{[2]}$$

$$\frac{\partial h_i^{[1]}}{\partial a_i^{[1]}} = \frac{\partial \sigma_3(a_i^{[1]})}{a_i^{[1]}} =$$

9

$$\begin{cases} 0 & \text{if} \quad a_i^{[1]} < 0 \\ a_i^{[1]} \text{ otherwise} \end{cases}$$

Note: if $a_i^{[1]} = 0 \implies NDF$

$$\frac{\partial a_i^{[1]}}{\partial w_{ij}^{[1]}} = \frac{\partial}{\partial w_{ij}^{[1]}}\left(b_i^{[1]} + \sum_{j=1}^{d} w_{ij}^{[1]} x_j\right) = x_j$$

So,

$$\left(\frac{\partial L(f(X), y)}{\partial w_{ij}^{[1]}}\right) = -2\sum_{k=1}^{K}(y_k - f_k(x))\sigma_1(a^{[3]})_k(\delta_{kn} - \sigma_1(a^{[3]})_n)w_{km}^{[3]}a_m^{[2]}w_{mi}^{[2]}a_i^{[1]}x_j$$

for $a_m^{[2]} > 0$ and $a_i^{[1]} > 0$, this is for a data point.

2   Neural network implementation

You will implement different classes representing a fully connected network for image classification problems. There are two classes of neural networks: one using only basic packages and one using PyTorch. In addition to that, a third class of neural network has been implemented using Tensorflow and requires some fixing in order to function correctly.
As you work through this problem, you will see how those machine learning libraries abstract away implementation details allowing for fast and simple construction of deep neural networks.

For each approach, a Python template is supplied that you will need to complete with the missing methods. Feel free to play around with the rest, but please do not change anything else for the submission. All approaches will solve the same classification task, which will help you validate that your code is working and that the network is training properly.
For this problem, you will work with the MNIST dataset of handwritten digits, which has been widely used for training image classification models. You will build models for a multiclass classification task, where the goal is to predict what digit is written in an image (to be precise, this is a k-class classification task where in this case $k = 10$). The MNIST dataset consists of black and white images of digits from 0 to 9 with a pixel resolution of 28x28. Therefore, in a tensor representation the images have the shape 28x28x1. The goal is to classify what digit is drawn on a picture using a neural network with the following characteristics:

- an arbitrary amount of hidden layers, each with arbitrary amount of neurons

- sigmoid activation function for all hidden layers
- softmax activation function for the output layer
- cross entropy loss function. *

* For the implementation from scratch (Part (a)) we use a mean squared error (MSE) loss function. This is not recommended for a classification task, but we use it to simplify the implementation. In the future please consider using cross entropy / log loss instead.


Figure 1: Example images from MNIST dataset with 28x28 pixel resolution.

The training set with n data points is of the form

$$(x^{(i)}, y^{(i)}), \ i = 1, ..., n, \tag{6}$$

with $y^{(i)} \in \{0, 1\}^k$, where $y_j^{(i)} = 1$ when j is the target class, and 0 otherwise (i.e., the output is one-hot encoded). The softmax output function $\hat{y} = h_\theta(x)$ outputs vectors in $\mathbb{R}^k$, where the relative size of the $\hat{y}_j$ corresponds roughly to how likely we believe that the output is really in class j. If we want the model to predict a single class label for the output, we simply predict class j for which $\hat{y}_j$ takes on the largest value.

Tasks

1. Complete the implementation of the neural network classes marked by TODO comments:

   (a) From scratch (`network_scratch.py`)
   (b) Using PyTorch (`network_pytorch.py`)

   A repository with the template files will automatically be created for your team when registering for the GitHub classroom assignment at `https://classroom.github.com/a/XgELWE_C`.

2. In `network_tensorflow.py` a Neural Network class has been implemented using Tensorflow and Keras.

   (a) There are three mistakes in the code preventing the network from working correctly. Comment the mistakes out and repair the class.

   (b) Implement a time-based learning rate class `TimeBasedLearningRate`: it should be initialized with a positive integer as initial learning rate and have the learning rate reduced by 1 at each step, until a learning rate of 1 is reached.

3. In addition to the implementation, show how your networks perform on the MNIST classification task in `MNIST_classification.ipynb`.

    (a) Plot the accuracy of each neural network on the training and on the validation set as a function of the epochs

    (b) Please make sure that the uploaded Notebook shows printed training and validation progress.

    (c) Please add team number, team member names and matriculation numbers as a comment at the top of the notebook.

Please note that the classifier does not need to get anywhere close to 100% accuracy. Especially, with the small amount of training data this is difficult. Instead, the training output should indicate that the model learns something, i.e., the accuracy increases over the epochs on the training data and is significantly better than random guessing.

References

A general note: This worksheet could not have been solved without the discussions and help of other classmates, ideas generated on ChatGPT and threads found on StackOverflow. All these resources have helped the learning process and have contributed at different parts, however, are hard to disentangle. Below, a list of non-exhaustive resources:

$$https://www.youtube.com/watch?v = UID893EosM8$$
$$https://sassafras13.github.io/Bias_variance/: : text = As$$

Submission

The submission of the whole problem set is done via GitHub classroom. You have to register for the assignment with your GitHub account at `https://classroom.github.com/a/XgELWE_C`. Please make sure that your GitHub account profile includes your real name. When starting the assignment, please create or join a team according to the teams assigned in Moodle (use the exact team name from Moodle).

    Please upload / push all solutions to the GitHub repository which was created for your team. Please upload your solutions for the theoretical part (1) as a PDF to GitHub. If you upload multiple PDF files, please indicate in the file name to which subtask they are corresponding (we much prefer one single pdf). For the practical part (2), just push your code changes to the existing files.

    Anybody in your team can push as often as they want. Once the deadline has passed, you are not able to push to your repository anymore and all changes on your **main** branch will be considered for grading. See the README.md in the repository for more details on how to push your changes to GitHub.