Deep Learning (IST, 2025-26)

Homework 1

Ben Peters, Raul Monteiro, Beatriz Canaverde, André Martins

Deadline: Friday, December 12, 2025.

Please turn in the answers to the questions below in a PDF file, together with the code you implemented to solve them (when applicable).

IMPORTANT: Please write 1 paragraph indicating clearly what was the contribution of each member of the group in this project. A penalization of 10 points will be applied if this information is missing.

Please submit **a single zip file** in Fenix under your group's name.

Question 1 (35 points)

Handwritten letter classification with linear classifiers and neural networks. In this exercise, you will implement a linear classifier to classify images of handwritten letters using the EMNIST Letter Dataset.[1] The dataset consists of 28x28 pixel bitmap images of the 26 letters of the Roman alphabet. We provide a prepared version of the dataset which has been partitioned into training, validation, and test sets, and in which bitmap has been flattened into a single $28 \times 28 = 784$-dimensional vector of independent pixels. These vectors will be your **default feature representation**. The dataset can be downloaded from

`https://drive.google.com/file/d/1HiWdETnSyHRlVeHu_0OOth2Zip5WuLq3`

Please do not use any machine learning library such as `scikit-learn` or similar for this exercise; just plain linear algebra (the `numpy` library is fine).

1. **Perceptron** In this exercise, you will implement a multi-class perceptron and then report its results on EMNIST Letters.
   (a) (10 points) Implement the `update_weight`, `train_epoch`, `predict`, and `evaluate` methods of the `Perceptron` class in `hw1-perceptron.py`. Train the perceptron for 20 epochs and plot its train and validation accuracies as a function of the epoch number. After each epoch, save a checkpoint of the model if it has higher validation accuracy than any previous checkpoint. After training, report the test accuracy of the model with the best performance on the validation set.
2. **Logistic Regression** In this exercise, you will implement a logistic regression classifier, using stochastic gradient descent (*i.e.*, with a batch size of 1) as the training algorithm, and conduct a grid search to find a good feature representation and hyperparameters for the task.

[1] `https://www.nist.gov/itl/products-and-services/emnist-dataset`

(a) (10 points) Implement logistic regression with ℓ_2 regularization and a stochastic gradient descent (SGD) weight update rule (hint: it may be useful to model the implementation on the perceptron from the previous exercise). Report results using a learning rate of 0.0001 and an ℓ_2 penalty of 0.00001. As in the perceptron exercise, train for 20 epochs, plot training and validation accuracies for each epoch, and report the test accuracy of the best-performing checkpoint.

(b) (5 points) Propose a different feature representation other than the 784-dimension flattened vector representation of the pixels and implement it. Explain why you think this is a good feature representation ("good" can either mean that you think it will lead to better accuracy, or that it will be more efficient to train without losing accuracy). List any references you found that support your choice.

(c) (10 points) Next you will implement a grid search to find a good set of hyperparameters for logistic regression on this dataset. More concretely, you will run your implementation using all combinations of

- 3 learning rate values
- 2 ℓ_2 penalty values
- The original feature representation and the one you implemented in part (b).

In other words, you will need to run $3 \times 2 \times 2 = 12$ configurations in total. As in previous exercises, for each configuration train for 20 epochs and save the checkpoint with the best validation accuracy. After you run the grid, report

- The validation accuracy of every configuration
- The test accuracy of the configuration with the best validation accuracy

3. **Multi-Layer Perceptron** Now, you will implement a multi-layer perceptron (a feed-forward neural network), again using as input the original feature representation.

(a) (10 points) **Without using any neural network toolkit,** implement a multi-layer perceptron with a single hidden layer to solve this problem, including the gradient backpropagation algorithm needed to train the model. Use 100 hidden units, a `relu` activation function for the hidden layers, and a multinomial logistic loss (also called cross-entropy) in the output layer. Do not forget to include bias terms in your hidden units. Train the model for 20 epochs with stochastic gradient descent (batch size of 1) with a learning rate of 0.001. Initialize biases with zero vectors and values in weight matrices with $w_{ij} \sim \mathcal{N}(\mu, \sigma^2)$ with $\mu = 0.1$ and $\sigma^2 = 0.1^2$ (hint: use `numpy.random.normal`). Use the same procedure for saving the best checkpoint as in the perceptron and logistic regression exercises. Report the test accuracy of the best-performing checkpoint and include the plots of the train loss and train and validation accuracies as a function of the epoch number.

Question 2 (35 points)

In this question, you will continue working with the EMNIST Letters dataset and extend your exploration beyond manual gradient computations. You will implement feedforward neural network (FFN) models using a modern automatic differentiation framework, specifically PyTorch, and investigate both the advantages and the limitations of these architectures. Your experiments will allow you to assess how much performance can be gained by using multilayer neural networks, as well as to identify situations in which increasing model complexity does not necessarily lead to better results.

1. (5 points) **Implementation Warmup.** Fill the missing parts of the script `hw1-ffn.py` to enable training of multi-layer FFNs. Insert your implementation in all locations where `NotImplementedError` is raised, following the specifications provided in the docstrings and ensuring that your functions return the expected values.

2. (18 points) **Towards an Infinite-Width One-Layer FFN.** In this exercise, you will investigate how increasing the number of hidden units affects performance in a FFN with a single layer.

   (a) (10 points) Train a one-hidden-layer feedforward neural network for a varying number of hidden units: $\{16,\ 32,\ 64,\ 128,\ 256\}$. For each width, perform a small grid search over all combinations of the following hyperparameters:

   - Four learning rate values;
   - No dropout and one non-zero dropout value;
   - No l_2 penalty and one non-zero value (commonly known as weight decay);

   In other words, you will need $5 \times 4 \times 2 \times 2 = 80$ configurations in total. Choose one optimizer from `SGD` and `Adam` and one activation function from `relu` and `tanh`. Use a batch size of 64 and train each model for 30 epochs. Present a table summarizing the configurations you explored, including the best validation accuracy obtained during training for each configuration. Make sure to highlight the best-performing configuration for each width (according to validation accuracy). Briefly comment on how the validation accuracy changes as the layer width increases.

   (b) (4 points) From among all widths and hyperparameter settings tested in a, identify the model with the highest validation accuracy. Plot the training loss and the validation accuracy over epochs. Comment on the training dynamics observed in these plots, including convergence behavior and whether there are any signs of underfitting or overfitting. Report the test accuracy of this model and briefly comment on its generalization ability.

   (c) (4 points) Using the best-performing model for each hidden-layer width explored in a, analyze how model width affects the network's capacity to interpolate the training data. Produce a plot of the final training accuracy as a function of hidden-layer width.
   Discuss whether increased width consistently results in improved training accuracy. Is the training accuracy converging to any value when you increase the number of units? How does this relate to the Universal Approximation Theorem? What would be the expected training accuracy with an infinite-width one-layer FFN?

3. (12 points) **Effect of Depth in Vanilla FFNs** In this exercise, you will investigate how increasing network depth affects performance in a simple FFN architecture.

   (a) (4 points) Using **32 hidden units per layer**, train models with number of hidden layers $L \in \{1,\ 3,\ 5,\ 7,\ 9\}$. Regarding the remaining hyperparameters, use the best-performing hyperparameter configuration (learning rate, dropout, l_2 penalty, optimizer, activation function, etc) that you obtained for the 32-unit model in the first exercise. Train each model for 30 epochs with a batch size of 64. No additional architectural changes or regularization techniques should be introduced. Present a table with the highest validation accuracy during training for each depth. Briefly comment on how the validation accuracy changes as the model depth increases.

   (b) (4 points) Select the configuration (i.e., depth) that achieved the highest validation accuracy and plot the training loss curve and the validation accuracy curve over the 30 epochs. Based on the curves, discuss whether the model appears to have converged within 30 epochs and whether there are any signs of underfitting or overfitting. Report the corresponding test accuracy and briefly comment on the model generalization.

(c) (4 points) Using the models you trained in a, collect the training accuracy obtained in the final epoch and create a plot of training accuracy as a function of depth. Based on this plot, discuss how depth affects the network's capacity to interpolate the training data. In your reflection, consider:

- the model's ability to fit the data as depth increases;
- possible limitations or challenges that emerge when the network becomes deeper;
- why deeper architectures may or may not achieve higher levels of interpolation in this setting.

Comprehensive but short answers are appreciated.

Question 3 (30 points)

Normalized relu and sparse alternatives to softmax. The softmax activation transforms a vector of logits $\boldsymbol{z} \in \mathbb{R}^K$ into a probability vector $\boldsymbol{p} \in \triangle_K$, where $\triangle_K := \{\boldsymbol{y} \in \mathbb{R}^K : \boldsymbol{y} \geq \boldsymbol{0}, \sum_i y_i = 1\}$ is the probability simplex. In this question you will study sparse alternatives to the softmax activation. One such alternative is the **sparsemax** activation, defined as the Euclidean projection onto the simplex:

$$\mathrm{sparsemax}(\boldsymbol{z}) := \arg\min_{\boldsymbol{p}\in\triangle_K} \|\boldsymbol{p} - \boldsymbol{z}\|^2. \tag{1}$$

It is shown [Martins and Astudillo, 2016] that the solution of the optimization problem above is of the form $\mathrm{sparsemax}(\boldsymbol{z})_i = \mathrm{relu}(z_i - \tau)$, where $\tau \in \mathbb{R}$ is the (unique) scalar satisfying the normalization constraint $\sum_i \mathrm{relu}(z_i - \tau) = 1$.

1. (5 points) Show that

$$\mathrm{relu}(\boldsymbol{z}) := \arg\min_{\boldsymbol{y}\geq\boldsymbol{0}} \|\boldsymbol{y} - \boldsymbol{z}\|^2. \tag{2}$$

Therefore, sparsemax is a "normalized relu", since (1) adds the constraint $\sum_i y_i = 1$ to the constraints in (2).

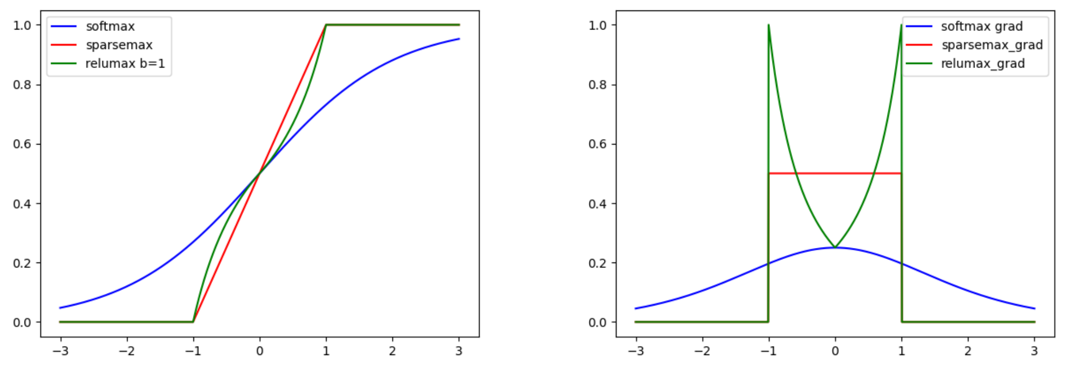
2. (5 points) Another way to normalize relu is through "post-normalization". A naive way to do this is by defining $p_i = \frac{\mathrm{relu}(z_i)}{\sum_j \mathrm{relu}(z_j)}$, however this is undefined if $\boldsymbol{z} \leq \boldsymbol{0}$ since we would get a $\frac{0}{0}$ indetermination. Consider instead, for a given choice of $b > 0$, the following activation, which we dub **relumax**:

$$\mathrm{relumax}_b(\boldsymbol{z})_i := \frac{\mathrm{relu}(z_i - \max(\boldsymbol{z}) + b)}{\sum_j \mathrm{relu}(z_j - \max(\boldsymbol{z}) + b)}. \tag{3}$$

Show that softmax, sparsemax, and relumax are all insensitive to adding a constant to $\boldsymbol{z}$ and that, assuming all entries in $\boldsymbol{z}$ are distinct, they all approach a one-hot vector in the zero temperature limit.[2] Show that, for each $\boldsymbol{z}$, there is a b such that $\mathrm{relumax}_b(\boldsymbol{z}) = \mathrm{sparsemax}(\boldsymbol{z})$.

3. (10 points) Assume $K = 2$ and $\boldsymbol{z} = [0, t]$. Derive closed form expressions for $\mathrm{softmax}(\boldsymbol{z})_2$, $\mathrm{sparsemax}(\boldsymbol{z})_2$, and $\mathrm{relumax}_b(\boldsymbol{z})_2$ as a function of t (and in the latter case, also b). Derive closed form expressions for their derivatives with respect to t. Plot the three functions and their derivatives.

[2]That is, $\lim_{T\to0^+}\mathrm{softmax}(\boldsymbol{z}/T) = \lim_{T\to0^+}\mathrm{sparsemax}(\boldsymbol{z}/T) = \lim_{T\to0^+}\mathrm{relumax}(\boldsymbol{z}/T) = \boldsymbol{e}_k$ with $k := \arg\max_j z_j$.



4. (5 points) Assume general $K \geq 2$. Compute the Jacobian of relumax, i.e., the matrix with entries $\frac{\partial \text{relumax}_b(\boldsymbol{z})_i}{\partial z_j}$. (Assume that all entries of $\boldsymbol{z}$ are distinct.)

5. (5 points) The cross-entropy loss (assuming the target class is i) is $L(\boldsymbol{z}) = -\log \text{softmax}(\boldsymbol{z})_i$. Explain why it's not a very good idea to use this loss when softmax is replaced by sparsemax or by relumax. Consider the loss $L(\boldsymbol{z}) = -\log \frac{\text{relumax}_b(\boldsymbol{z})_i + \epsilon}{1 + K\epsilon}$ for some $\epsilon > 0$. Compute the gradient of this loss with respect to $\boldsymbol{z}$.

References

[Martins and Astudillo, 2016] Martins, A. and Astudillo, R. (2016). From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International conference on machine learning*, pages 1614–1623. PMLR.