NNDL2024: Learning tasks and architectures
Session 3 (11 Apr), Deadline 15 Apr

Mathematical exercises

1. (15 pts) Convolutional layer as a special case of a fully connected layer.

   (a) Consider a problem where inputs $\mathbf{x} \in \mathbb{R}^D$ are processed by a single layer of M *one-dimensional convolutions* of width W, with stride of one and no dilation (dilation= 1 in PyTorch terminology), but using zero-padding. How many outputs and weights does this layer have?

   (b) Characterize a fully connected layer that can perform the same computation. How many neurons and hidden weights do you have now?

   (c) Invent an explicit regularizer for the fully connected layer that encourages the learnt weights to resemble the convolutional layer. The regularizer should favor solutions that have the same sparsity structure and parameter sharing and penalize for all other solutions, by an amount that can be controlled by some regularization parameter(s).

2. (15 pts) Residual connections.

   (a) Consider a simple residual network of

   $$\begin{aligned}\mathbf{h}_1 &= \mathbf{x} + \mathbf{f}_1(\mathbf{x}, \mathbf{W}_1)\\ \mathbf{h}_2 &= \mathbf{h}_1 + \mathbf{f}_2(\mathbf{h}_1, \mathbf{W}_2)\\ \mathbf{y} &= \mathbf{h}_2 + \mathbf{f}_3(\mathbf{h}_2, \mathbf{W}_3)\end{aligned}$$

   where each $\mathbf{f}_k(\cdot,\cdot)$ is a small neural network that takes the first argument as input and the second denotes its parameters. Express $\mathbf{y}$ as a function of only $\mathbf{x}$ and the functions $\mathbf{f}_k$, without using the intermediate variables $\mathbf{h}$.

   (b) Compute the number of alternative paths from $\mathbf{x}$ to $\mathbf{y}$. On how many of these paths $\mathbf{f}_2(\cdot)$ is used?

   (c) Express the derivative $\frac{\partial \mathbf{y}}{\partial \mathbf{f}_1}$ and compare it qualitatively to what we would have for a similar network $\mathbf{y} = \mathbf{f}_3(\mathbf{f}_2(\mathbf{f}_1(\mathbf{x})))$ without the residual connections.

Computer Assignments

1. (30 pts) Hyperparameter optimization.

✓ (a) Implement a neural network with the following structure for the MNIST data. Apart from the hyperparameters explicitly stated here, you can use the PyTorch's defaults.

- a convolutional layer with 16 output channels and the kernel size given as a free parameter,
- ReLU activation,
- a max pooling layer with kernel size 2 and stride 2 (for reducing spatial dimensionality by two),
- a fully connected layer with 32 output features,
- ReLU activation,
- a fully connected layer with 10 output (for class probabilities),
- log softmax activation.

(b) Use Bayesian optimization (the `BoTorch` method implemented in the `Ax` library) for simultaneously tuning the learning rate (in the range between $1e-7$ and $1e-3$) and momentum (in the range between 0.0 and 1.0) of the optimizer, and the kernel size (amongst the choices of $\{3, 5, 7\}$). Run the method for 75 rounds. Note that the notebook already includes most of the code needed for this.

(c) Implement grid search for the same three parameters, again covering in total 75 alternatives.

(d) **Report:** For both methods report the final optimal parameters, and also the progress of hyperparameter optimization as a function of round (the notebook already has plotting code for this). Brief analysis of the observations. How many hyperparameters you think you could tune simultaneously with these methods?

✓ 2. (20 pts) Transfer learning. The notebook provides a pre-trained digit classification model, trained on the MNIST data. Your task is to adapt this model to work well in a new classification problem where the inputs are images of same size, but the classes are now object categories (Fashion-MNIST data). You have only 64 training examples for this target task.

✓ (a) Training on the target data only: Ignore the pre-trained weights and train the model on the small target data data. Plot the training and test losses as a function of epoch.

✓ (b) Fine-tune the pre-trained source model on the target data so that you fix the weights of all other layers and only train the last layer. Plot the losses as before.

✓ (c) Fine-tune all weights of the pre-trained model on the new data. Again plot the losses.

✓ (d) **Report**: The final (based on your choice of convergence) training and test losses and classification accuracies for all three variants. Analysis of the results.

3. (20 pts) Few-shot learning. Use the same model and data as in previous exercise. Read the `SimpleShot` few-shot learning approach from paper `https://arxiv.org/pdf/1911.04623.pdf`. Use the pre-trained MNIST model as the feature extractor and use few-shot learning to classify Fashion-MNIST images.

✓ (a) Write down the algorithm on a level of mathematical expressions (probably easiest with pen and paper). What is being computed, how is the classification decision made etc.

(b) Implement the algorithm.

(c) Evaluate the classification accuracy for $K = 1$ and $K = 5$ (K is the number of examples per class), using one of the proposed feature normalization strategies. Use the results of the original paper to justify your choice of normalization.

(d) **Report**: The classification accuracies for the two choices of K. Analysis of the results.