NNDL2025: Neural network optimization and generalization Session 2 (27 Mar)

Mathematical Exercises

1. (20 pts) Apply reverse-mode automatic differentiation (with pen and paper) for the function $z(a, b) = \frac{a^2}{a^2+b^2+1}$. Follow the algorithm as closely as possible. Do not, for example, write out a general expression for the derivative even though it would be easy in this case.
   (a) Draw the computational graph and label all intermediate results
   (b) Provide the forward pass computation, including the partial derivatives for intermediate steps
   (c) Provide the backward pass and evaluate the partial derivatives $\frac{\partial z}{\partial a}$ and $\frac{\partial z}{\partial b}$ at $a = 2, b = 1$.
2. (20 pts) NN initialization. Consider an individual neuron with M inputs, with weights w_m initialized for random values drawn from $\mathcal{N}(0, \sigma^2)$ (and no bias for simplicity). Further assume that the inputs x_m are independently distributed as $\mathcal{N}(0, 1)$ (for instance, we used z-score normalization).
   (a) Derive the mean and variance of the neuron before the activation function.
   (b) Consider a case where we have a layer of M such neurons, used as inputs for a neuron on a next level. Assume the activation function is identity. What is the mean and variance for the second layer neurons?
   (c) Solve for σ^2 that is needed for retaining the variance over the layers, irrespective of how many layers we stack. What would happen if we initialize the weights with σ^2 considerably smaller or larger than that threshold?
   (d) Consider again an individual neural, but now assume we have the ReLU activation. What can you say about the mean and variance after the activation? Do you think we could still arrive at an initialization rule that would retain the variance?
   Hint: You can figure out second moments quite easily without direct integration.

Computer Assignments

1. Optimization (30 pts). The purpose of this exercise is to learn how to implement a simple neural network and optimize it well, so that you reach perfect accuracy on the training data.
   (a) Finalize the network specification, a fully connected neural network with 3 hidden layers, having 400, 200 and 100 neurons. Use the ReLU activation function for the hidden layers and linear for the output.
   (b) Write optimization loop (over epochs and batches). You can follow any tutorials you want to, but write the optimization loop explicitly (instead of using some high-level functions that perform the whole optimization) and tell what material you relied on, if any.
   (c) Select an optimizer and find (by manual exploration) hyperparameters (batch size, step length etc.; the set depends on the optimizer you chose) such that you get the training error below 10^{-6}. That is, train the model so that it fits exactly to the training samples, within numerical accuracy.
   (d) Write code that evaluates the following quantities after the model has been trained:
      i. The squared norm of the gradient over all network parameters, as average over the training samples: $\|\nabla_\theta \mathcal{L}(\theta)\|^2 = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{P} \|\nabla_{\theta_j}\mathcal{L}_i(\theta)\|^2$, where P is the number of model parameters, N the number of samples, and $\mathcal{L}_i$ refers to the loss for the ith sample.
      ii. The variation of the gradient norm over mini-batches: Compute the above expression separately for each batch and then evaluate the variance over the batches.

These are the quantities studied in SGD regularization theory, but here we just practice how to extract gradient information and do not use them for anything.

(e) **Report:**

i. Plot that shows the progress of the training error over the epochs, using log-scale for the y-axis so that we see the small diferences.

ii. The choice of optimizer and all of its parameters you used.

iii. How many epochs it took to reach the threshold of 10^{-4}

iv. The gradient norm and its variation over the minibatches, computed after training the model.

2. Double descent (30 pts). The purpose of this exercise is to understand the double descent phenomenon.

(a) Use the data in the notebook, with $N = 100$ training instances for learning $\mathbf{g}(\mathbf{x}) : \mathbb{R}^{10} \to \mathbb{R}^5$. The notebook provides the model architecture with the parameter M controlling the model complexity (number of neurons per layer). There is also a separate test set with much more samples, for evaluating generalization.

(b) Empirically validate the double descent principle. Optimize the model until convergence for a range of M from $M = 2$ to $K = 60$. You can use $B = N$ (so no stochasticity in optimization) to keep the results more consistent across runs. Remember to check that you optimize the model well enough!

Report:

- Plot the final training loss and the final test loss as a function of K, aiming for a plot that looks like the one in the lecture slides.
- What is the *interpolation threshold* in terms of M? How many parameters does your model have for that M?

(c) Validate empirically that having more data can hurt. Pick two M, one at the interpolation threshold and one in the overparameterized regime. Re-train the models with $N = 500$ samples from the same distribution.

Report: The test losses for $N = 100$ and $N = 500$ for both models, with an explanation.