NNDL2024: Neural network optimization and generalization
Session 2 (4 Apr), Deadline 8 Apr

Mathematical Exercises

✓ 1. (10 pts) Apply reverse-mode automatic differentiation (with pen and paper) for the function $z(a,b) = \frac{\log(b)}{1+e^{a^2+b^2}}$

   (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 computation

   (d) Evaluate the partial derivatives $\frac{\partial z}{\partial a}$ and $\frac{\partial z}{\partial b}$ at $a = 2, b = 3$.

✓ 2. (20 pts) Consider a fully connected feedforward neural network from I inputs ($\mathbf{x} \in \mathbb{R}^I$) to O outputs ($\mathbf{y} \in \mathbb{R}^O$), with K hidden layers each consisting of L neurons. The output layer has linear activation, all other activation functions are identical ($\psi : \mathbb{R} \to \mathbb{R}$), and the loss is MSE.

   (a) Write down the forward pass computation for a single $\mathbf{x}$ as mathematical expressions, indicating clearly the intermediate variables. Be explicit about the size of each matrix/vector.

   (b) Write down the backward computation for a single example, using $\mathbf{e}$ to denote the difference between the model output $\hat{\mathbf{y}}$ and the desired output $\mathbf{y}$. It is okay to build on some source material (instead of working out all derivatives yourself), but tell which source you used.

   (c) Count the number of operations required for the forward and backward passes. For the individual operations you can use the computational complexity of a naive method (e.g. $\mathbf{AB}$ for $\mathbf{A} \in \mathbb{R}^{i\times j}$ and $\mathbf{B} \in \mathbb{R}^{j\times k}$ has complexity of $\mathcal{O}(ijk)$) but you should explicitly compute how many times the operations are carried out (e.g. three such products is counted as $3\mathcal{O}(ijk)$).

3. (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) Derive the second moment of the neuron after a ReLU activation. Hint: Think about the effect of ReLU; you may not need direct integration. (March 28 update: The previous version asked to derive the variance, but this is unnecessarily difficult and not needed.)

   ✓ (c) Now consider a case where we have a layer of M such neurons, used as inputs for a neuron on a next level. What are the corresponding means and variances for the second layer neurons?

   ✓ (d) Solve for σ^2 that is needed for retaining the variance over the layers. What would happen if we initialize the weights with σ^2 considerably smaller or larger than that threshold?

   (e) Consider a network with a scalar input and a scalar output, with a single hidden layer of M units with the same activation function. Show that the initialization variance influences the initial smoothness of the function (but no need for a formal proof), for instance using a Taylor expansion. Does larger or smaller σ^2 result in smoother functions

Computer Assignments

1. Double descent (25 pts)

   (a) Use the data in the notebook, with $N = 100$ training instances for learning $\mathbf{g}(\mathbf{x}) : \mathbb{R}^8 \to \mathbb{R}^8$.

(b) Use a fully connected architecture with two hidden layers of M units in both and the ReLU activation, with linear activation for the output layer. Optimize the training MSE for M in $\{10, 50, 100\}$. You can use any optimizer, but make sure you optimize the problem well. Note that you may need a lot of epochs.

Report:

- The theoretical optimal test error R_t, figured out based on the data generation process.
- Plot that shows progress of the training loss for each M (all in the same plot, with a clear legend), using log-scale for both the epochs and the loss.
- For which M you can reach MSE that is below 10^{-3}? Is it faster or slower (in terms of epochs) to train larger models?

(c) Consider a denser grid of M from $M = 2$ to $K = 200$ and empirically validate the double descent principle. Use $B = N$ (so no stochasticity in optimization) to keep the results more consistent across runs.

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?

(d) 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 $N = 200$ samples from the same distribution.

Report: The test losses for $N = 100$ and $N = 200$ for both models, with an explanation.

2. Implicit regularization of SGD (25 pts)

(a) Use the same data and model as in the previous exercise, but add a few more hidden layers (we used five). Use standard SGD (with constant learning rate) for optimising the model and MSE as the optimimzation criterion.

(b) Optimize the model with some choice of M (neurons per layer) in the overparameterized regime, some mini-batch size B, and a few alternative choices for the learning rate μ to investigate the effect of the learning rate. We used $M = 50$, $B = 20$ and $\mu \in \{0.02, 0.04, 0.08, 0.16\}$, but you may need to use other values to see the main effects depending on your exact implementation. Run the optimizer long enough to ensure (more or less) global convergence, with MSE at least below 10^{-5} if possible.

Report: Convergence plot (all choices in the same plot, with a clear legend) confirming you optimize the problem well. Tell also which parameters (M, B, number of layers) you used.

(c) Write code that computes $\|\nabla J(\boldsymbol{\theta})\|^2$ and $\frac{1}{N_b}\sum_{b=1}^{N_b}\|\nabla J_b(\boldsymbol{\theta}) - \nabla J(\boldsymbol{\theta})\|^2$, where $\nabla J(\boldsymbol{\theta})$ is the full data gradient over all network parameters, $\nabla J_b(\boldsymbol{\theta})$ is the gradient for the bth mini-batch (both computed as averages, not sums, over the samples), and N_b is the number of mini-batches. These measure the gradient norm and its variation over the mini-batches.

Report:

- A plot showing the relationship between the learning rate μ and the final gradient norms of a converged solution.
- A plot showing the relationship between the test error and the final gradient norms.
- Explanation of the observations, in light of the implicit regularization results (Lecture 5).

Hint: If you want more reliable results, you may average the plots over several runs with newly sampled training data.

(d) Run SGD with explicit regularization for the weights (implemented easily as `weight_decay` parameter of the optimizer), trying to find a tess loss that is better than what you got with implicit regularization only. Do not change the model itself.

Report: The settings (μ, B and the regularization rate), together with the training and test losses. Conclusion on whether implicit regularization is here sufficient or not.