Institut for Theoretical Physics
University Leipzig

Prof. Dr. B. Rosenow
M. Kühn

Statistical Mechanics of Deep Learning - Problem set 14

Winter Term 2024/25

Hand in Python code: Before **Monday 03.02.2025, 9:15**, only submit the Python code you have written. Share a Google Colab Notebook with your code and send the link via email to itpleipzig@gmail.com.

25. Wide Networks and Gaussian Processes *3 + 3 + 3 Points*

In the lecture, the equivalence between noisy gradient descent training with L_2-regularization of an infinitely wide neural network and a Gaussian process is established. The purpose of this problem is to numerically validate the equivalence between training a wide neural network and predicting the test data with a Gaussian process.

(a) We consider a two-layer neural network defined by

$$f_{\mathbf{w}}(\mathbf{x}) \;=\; \sum_{i=1}^{N} w_i^{(2)}\, g\left(\sum_{j=1}^{d} w_{ij}^{(1)}\, x_j\right) \quad ,$$

with ReLU transfer function $g(z) = \theta(z)\, z$ and input $\mathbf{x} \in \mathbb{R}^d$. The corresponding Gaussian process is defined by the kernel

$$K(\mathbf{x}, \mathbf{x}') = \langle f_{\mathbf{w}}(\mathbf{x}) f_{\mathbf{w}}(\mathbf{x}') \rangle_w$$

where the average is performed over the set of weights $\{w_i^{(2)}, w_{ij}^{(1)}\}$. To make the coding simpler later on, we choose the variances $\sigma_1^2 = \sigma_2^2 = 1/\sqrt{dN}$. Numerically, the average can be performed by initializing many, e.g. $M = 400$, different networks using Pytorch, and then computing

$$K(\mathbf{x}, \mathbf{x}') \;=\; \frac{1}{M} \sum_{i=1}^{M} f_{\mathbf{w}_i}(\mathbf{x}) f_{\mathbf{w}_i}(\mathbf{x}') \quad .$$

Perform the average for $N = 1000$ hidden units, and for 200 pairs of vectors $\mathbf{x}, \mathbf{x}' \in \mathbb{S}^d$ for $d = 10$. Compare the numerically averaged kernel with the analytic result

$$K_{\text{analytic}}(\mathbf{x}, \mathbf{x}') \;=\; \sigma_1^2 \sigma_2^2 N \frac{1}{2\pi} \left(\sin\theta + (\pi - \theta)\cos\theta\right) \qquad \text{where } \cos\theta = (\mathbf{x}, \mathbf{x}')$$

by plotting the set of 200 pairs of data points $\{K(\mathbf{x}, \mathbf{x}'), \arccos(\mathbf{x}, \mathbf{x}')\}$ together with the theoretical prediction $K_{\text{analytic}}(\theta)$.

(b) Draw $p = 40$ training data points $\mathbf{x}_i \in \mathbb{S}^d$ and generate labels y_i via the target function $h(\mathbf{x}) = \sqrt{d} \cdot |\mathbf{x} \cdot \mathbf{T}|$ with a fixed $\mathbf{T} \in \mathbb{S}^d$ which has randomly chosen components and is then normalized to unity. Now $K_{\text{analytic}}(\mathbf{x}_i, \mathbf{x}_j) \equiv K_{ij}$ defines a $p \times p$ matrix. Compute K_{ij}. The main appeal of Gaussian processes is that Bayesian Inference with Gaussian process priors is tractable. In Gaussian process inference we use the mean of the Gaussian process distribution conditioned on the data (posterior) as the predictor g^*, and it is given by

$$g^*(\mathbf{x}_{\text{test}}) = \sum_{n,m=1}^{p} K_{\text{analytic}}(\mathbf{x}_{\text{test}}, \mathbf{x}_n) \left(\underline{\underline{K}} + \sigma^2 \mathbb{I}\right)^{-1}_{n,m} y_m \quad .$$

Here, $\mathbb{I}$ denotes the identity matrix, $\sigma^2 = 0.1$ acts as a regulator of the prediction and can be understood as assuming the labels have a Gaussian noise with variance σ^2. $(\underline{\underline{K}}+\sigma^2\mathbb{I})^{-1}_{n,m}$ must be understood as inverting the matrix $(\underline{\underline{K}}+\sigma^2\mathbb{I})$ and evaluating the inverse for indices n, m. A good check for a correct implementation is that for $\sigma = 0$ you must find $g^*(\mathbf{x}_i) = y_i$. Next, draw 40 test vectors $\mathbf{x}_{i,\text{test}} \in \mathbb{S}^d$, and compute the **full** squared error (loss) as

$$\sum_{i=1}^{40} \left[g^*(\mathbf{x}_{i,\text{test}}) - h(\mathbf{x}_{i,\text{test}})\right]^2$$

over the test vectors. Similarly, calculate the train loss.

(c) Build a neural network as described in (a) for N=1000 and train it on the data set prepared in (b). Use gradient descent with **full** squared error as loss function and, after each update step, add a gaussian random vector $\boldsymbol{\xi}$ with zero mean and standard deviation of $2 * \sqrt{\text{learning\_rate}} * \sigma$ to the weight vector to achieve noisy gradient descent. Use L_2-regularization with strength $\lambda = 2\sigma^2/\sigma_1^2$. Train for 10000 update steps with a learning rate of 0.001. What is the train and test loss of the final network on the data set prepared in (b)? What is the train and test loss if you save the corresponding outputs the network returns for the examples every 100 update steps and then calculate the loss for the average of these outputs? Compare the results of (b) and (c) for multiple randomly initialized data sets.

Hint: To add the random vector to the weights you can use the following code:

```
with torch.no_grad():
    for param in network.parameters():
        random_vector = 2*sigma*np.sqrt(lr)*torch.randn_like(param)
        param.add_(random_vector)
```

Furthermore, to use full squared error, pass the argument 'reduction="sum"' to the Pytorch function:

```
criterion = nn.MSELoss(reduction="sum")
```