Exam for ML for Physicists, PHYS-467, academic year 2024/25

Solution

Instructions:

- Total number of points is 73.

Contents

1 Match tools to tasks. [5 points]

Match one of the tools (a)-(e) to each task 1.-5. in the list below (1 point for each correct match):

Tools:

(a) Supervised regression, prediction

(b) Supervised regression, estimation

(c) Supervised classification

(d) Unsupervised clustering

(e) Unsupervised generative models

Tasks:

1. A music streaming service groups songs into playlists by analyzing similarities in acoustic features. One aims to create groups without prior labels.
2. In signal processing, the frequency and amplitude of a sinusoidal signal are estimated by fitting a regression model to sampled data.
3. A weather app predicts the temperature for tomorrow based on historical weather data and current atmospheric conditions. The focus is on forecasting precise future values for continuous outcomes.
4. An AI creates photorealistic images of non-existent landscapes by learning patterns from thousands of real photos. It learns to synthesize new, plausible examples.
5. An email filter determines whether a new message is 'spam' or 'not spam' by analyzing patterns from labeled examples.

Solution:

1, (d)
2, (b)
3, (a)
4, (e)
5, (c)

2 Classification with Linear Discriminant Analysis [9 points]

Consider a data matrix $X \in \mathbb{R}^{n\times d}$ and labels $\mathbf{y} \in \{-1,+1\}^n$. Our goal is to learn a binary classifier from this data. To do so, we assume the following probabilistic model: for a given vector $\mathbf{x} \in \mathbb{R}^d$ and sign $s = \pm 1$,

$$p\left(\mathbf{x}|y=s\right) \sim \mathcal{N}\left(\mu_s, \Sigma\right) \tag{1}$$

for vectors $\mu_{-1}, \mu_{+1} \in \mathbb{R}^d$ and a covariance matrix $\Sigma \in \mathbb{R}^{d\times d}$. In other words, the distribution of $\mathbf{x}$ is a Gaussian whose mean depends on the label y, and with the same covariance matrix for both classes. This model is called *Linear Discriminant Analysis*, and in this exercise we want to learn the parameters $\mu_{-1}, \mu_{+1} \in \mathbb{R}^d$, $\Sigma \in \mathbb{R}^{d\times d}$ from the data.

1. (4 points) Write the log-likelihood $\log p\left(X, \mathbf{y}|\mu_{+1}, \mu_{-1}, \Sigma\right)$ and deduce the maximum likelihood estimators for $\hat{\mu}_{+1}, \hat{\mu}_{-1}, \hat{\Sigma}$.

Solution: By independence of the samples, the log-likelihood is

$$p(X, y|\mu_{\pm 1}, \Sigma) = \prod_{i=1}^{n} p(y_\mu) \times p(\mathbf{x}_i|y_i, \mu_{\pm 1}, \Sigma) \tag{2}$$

$$= \frac{1}{2^n} \prod_{i=1}^{n} \frac{1}{(2\pi)^{d/2}\sqrt{det(\Sigma)}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu_{\mathbf{y_i}})^\top \Sigma^{-1}(\mathbf{x}-\mu_{\mathbf{y_i}})\right) \tag{3}$$

Such that, up to an additive constant that is irrelevant, the log likelihood is

$$\log p = -\frac{n}{2}\log(det(\Sigma)) - \frac{1}{2} \times \sum_{i=1}^{n} (\mathbf{x_i}-\mu_{\mathbf{y_i}})^\top \Sigma^{-1}(\mathbf{x_i}-\mu_{\mathbf{y_i}}) \tag{4}$$

We derive the estimators by cancelling the derivative of $\log p$. For $\hat{\mu}_1$,

$$\Sigma^{-1} \sum_{i=1, y_i=1}^{n} \mathbf{x_i} - \hat{\mu}_\mathbf{1} = \mathbf{0} \Rightarrow \hat{\mu}_1 = \frac{1}{N_1} \sum_{i=1, y_i=1}^{n} \mathbf{x_i}$$

i.e $\hat{\mu}_s$ is the empirical mean of the samples with label s. Canceling the derivative with respect to Σ, and knowing (non trivial!) that the derivative of $\log\det\Sigma$ is Σ^{-1}

$$\hat{\Sigma} = \frac{1}{n}\sum_{i=1}^{n} \left(\mathbf{x}_i - \hat{\mu}_{y_i}\right)\left(\mathbf{x}_i - \hat{\mu}_{y_i}\right)^T$$

i.e the empirical covariance matrix.

2. (2 points) Given the estimated parameters $\hat{\mu}_{+1}, \hat{\mu}_{-1}, \hat{\Sigma}$, write the probability $p\left(y=1|\mathbf{x}\right)$ that the label is 1 for a previously unseen test sample $\mathbf{x} \in \mathbb{R}^d$.

Solution: We use the Bayes rule to write

$$p(y=1|\mathbf{x}) = \frac{p(\mathbf{x}|y=1) \times p(y=1)}{p(\mathbf{x}|y=1) \times p(y=1) + p(\mathbf{x}|y=-1) \times p(y=-1)} \tag{5}$$

$$= \frac{p(\mathbf{x}|y=1)}{p(\mathbf{x}|y=1 + p(\mathbf{x}|y=-1)} \tag{6}$$

$$= \frac{\exp\left(-\frac{1}{2}(\mathbf{x}-\mu_\mathbf{1})^\top\Sigma^{-1}(\mathbf{x}-\mu_\mathbf{1})\right)}{\exp\left(-\frac{1}{2}(\mathbf{x}-\mu_\mathbf{1})^\top\Sigma^{-1}(\mathbf{x}-\mu_\mathbf{1})\right) + \exp\left(-\frac{1}{2}(\mathbf{x}-\mu_{-\mathbf{1}})^\top\Sigma^{-1}(\mathbf{x}-\mu_{-\mathbf{1}})\right)} \tag{7}$$

3. (2 points) Manipulating the expression for $p\left(y=1|\mathbf{x}\right)$ you will see that the decision boundary between the two classes is linear, i.e. there is a vector $\mathbf{w}$ that defines a hyperplane separating the two classes. What is the expression of $\mathbf{w}$ in terms of $\hat{\mu}_{+1}, \hat{\mu}_{-1}, \hat{\Sigma}$?

Solution: Simplifying the expression above, we have

$$p(y=1|\mathbf{x}) = \frac{1}{1+\exp\left(\mathbf{x}^\top\Sigma^{-1}(\mu_{-1}-\mu_1)+\frac{1}{2}(\mu_1+\mu_{-1})\Sigma^{-1}(\mu_1-\mu_{-1})\right)} \tag{8}$$

such that the separating vector is $\mathbf{w} = \Sigma^{-1}\left(\mu_1 - \mu_{-1}\right)$ Full points are awarded only if the correct expression is given (up to a sign or a multiplicative factor)

4. (1 point) What model seen in class has (up to constants) the same form of $p\left(y=1|\mathbf{x}\right)$?

Solution: The expression is the same as the logistic regression (with a bias added).

3 Three-State Spin Card Game [12 points]

We study the following statistical inference problem, we can call it a three-state spin card game, in which each player $i \in \{1, \ldots, N\}$ holds a *three-valued* card, $S_i^* \in \{-1, 0, +1\}$. We do *not* assume these three values to be equally likely among players; instead a prior parameter $\rho \in [0, 1/2]$ controls how often ± 1 appear, while 0 is another state with its own probability. Precisely:

$$P(S_i^* = +1) \;=\; P(S_i^* = -1) \;=\; \frac{\rho}{2}, \quad P(S_i^* = 0) \;=\; 1 - \rho.$$

We observe a measurement $Y \in \mathbb{R}^{N \times N}$, symmetric ($Y_{ij} = Y_{ji}$), such that

$$Y_{ij} \;=\; \begin{cases} \dfrac{1}{\sqrt{N}} \Big(2\, \delta_{S_i^*,\, S_j^*} - 1 \Big) \;+\; \sqrt{\Delta}\, W_{ij}, & i < j, \\ 0, & i = j, \\ Y_{ji}, & i > j, \end{cases} \tag{9}$$

$$W_{ij} \sim \mathcal{N}(0, 1) \text{ i.i.d.},$$
$$\Delta > 0 \text{ (noise variance)}.$$

(The factor $\delta_{S_i^*,\, S_j^*}$ is 1 if $S_i^* = S_j^*$ and 0 otherwise. Thus if two cards match, $2\delta_{S_i^*, S_j^*} - 1 = +1$, else -1.)

1. (3 points) Write the posterior probability distribution $P(S|Y)$ over card configurations $S = (S_1, \ldots, S_N) \in \{-1, 0, +1\}^N$. Simplify the expression in a way that terms that do not depend on the configuration S are included in the normalization.

Solution:

- The prior on each card: $P(S_i) = \rho/2$ for $S_i = \pm 1$, and $1 - \rho$ for $S_i = 0$. Thus

$$P(S) = \prod_{i=1}^{N} \big[\tfrac{\rho}{2} \delta_{S_i,+1} \;+\; \tfrac{\rho}{2} \delta_{S_i,-1} \;+\; (1-\rho) \delta_{S_i,0} \big].$$

- The likelihood for Y_{ij}, for $i < j$, up to constants independent of S, is $\exp\big(-\frac{1}{2\Delta}(\frac{2\delta_{S_i,S_j}-1}{\sqrt{N}} - Y_{ij})^2\big)$. Expanding the exponent and ignoring S-independent parts yields $\exp\big(\frac{1}{\Delta\sqrt{N}}(2\delta_{S_i,S_j} - 1) Y_{ij}\big)$.

- Multiplying over all $i < j$ and dividing by the normalization factor $\tilde{Z}$ leads to

$$P(S \mid Y) = \frac{1}{Z(Y, \Delta)} \prod_{i=1}^{N} \Big[\tfrac{\rho}{2}\, \delta_{S_i,+1} + \tfrac{\rho}{2}\, \delta_{S_i,-1} + (1-\rho)\, \delta_{S_i,0} \Big] \;\times\; \prod_{i<j} \exp\Big[\tfrac{1}{\sqrt{N}\,\Delta} \big(2\, \delta_{S_i,S_j} - 1 \big)\, Y_{ij} \Big].$$

□

2. (2 points) Write the estimator for the values of cards $\hat{S}_i : Y \to \in \{\pm 1, 0\}$ that maximizes the number of cards for which the value is correctly estimated, i.e. $\hat{S}_i = S_i^*$ (assume that you compare $\hat{S}$ to S^*, but also to $-S^*$ so that you can ignore issues arising because of ± 1 symmetry.).

Solution: Argmax of the marginals of the posterior.

3. (3 points) Describe a Metropolis MCMC to sample from the posterior $P(S|Y)$. Recall that in the Metropolis MCMC we consider a given current configuration S and propose a new configuration S' that is then accepted with a certain acceptance probability involving the ratio $\frac{P(S')}{P(S)}$. For your proposed Metropolis MCMC write the ratio $\frac{P(S')}{P(S)}$ as explicitly as possible and write explicitly the rule for acceptance of S'.

Solution: Metropolis Steps:

(a) From current $S = (S_1, \ldots, S_N)$, pick an index i uniformly in $\{1, \ldots, N\}$.

(b) Propose to change S_i to one of the other 2 states in $\{-1, 0, +1\} \setminus \{S_i\}$, each with probability 1/2.

(c) Compute ratio

$$r \;=\; \frac{P(S')}{P(S)} \;=\; \frac{P(S' \mid Y)}{P(S \mid Y)},$$

where S' differs from S only at site i. Since the prior factor changes from $\rho/2$ to $(1-\rho)$ or vice versa, and the likelihood factor changes in i-th row/column only, we get

$$r = \frac{\text{(new prior factor at site } i)}{\text{(old prior factor at site } i)} \;\times\; \exp\Big[\, \tfrac{1}{\sqrt{N}\,\Delta} \sum_{j \neq i} \Big((2\,\delta_{S_i', S_j} - 1) \;-\; (2\,\delta_{S_i, S_j} - 1) \Big)\, Y_{ij} \Big].$$

(d) Accept with probability $\min(1,\, r)$.

4. (2 points) Verify the proposed Metropolis MCMC satisfies the detailed balance condition.

Solution: Detailed Balance: If $P_{eq}(\cdot) = P(S|Y)$ is the equilibrium measure, we check $P_{eq}(S)\,T(S \to S') = P_{eq}(S')\,T(S' \to S)$, where $T(S \to S')$ is the product of (i) probability $\frac{1}{N}$ to pick site i (ii) the $\frac{1}{2}$ to pick that new state, (iii) accept probability $\min(1,\, r)$. So we have $P_{\text{eq}}(S) = P(S \mid Y)$ and $T(S \to S') = \frac{1}{N}\,\frac{1}{2}\,\min(1, r)$. Meanwhile, $r = \frac{P(S')}{P(S)}$. Standard Metropolis arguments show that

$$P(S)\,\frac{1}{N}\,\tfrac{1}{2}\,\min(1,\, r) \;=\; P(S')\,\frac{1}{N}\,\tfrac{1}{2}\,\min(1,\, 1/r).$$

Hence $P(S)T(S \to S') = P(S')T(S' \to S)$, so the chain obeys detailed balance.

5. (2 points) Another type of MCMC that allows to sample the posterior is *Gibbs sampling*. Write the main idea of Gibbs sampling and its main steps for the present problem.

Solution: Holding S_{-i} fixed, the posterior factorizes with respect to the site i via

$$P(S_i = s \mid S_{-i}, Y) \;=\; \frac{P(S \mid Y)}{\sum_{s'} P(S_i = s', S_{-i} \mid Y)}.$$

But $P(S \mid Y)$ factorizes into the prior piece at site i and the likelihood terms for pairs (i, j). Summation over $s' \in \{-1, 0, +1\}$ in the denominator normalizes the distribution.

4 Kernel feature maps [4 points]

Consider the following kernel in one dimension $K(x, y) = e^{-xy}$, $x, y \in \mathbb{R}$.

1. (3 points) Write a feature map $\phi(x)$ associated to the kernel K ($\phi(x)$ can be complex-valued). Hint: Taylor-expand the exponential in the definition of K.

Solution: Let us expand the exponential as

$$K(x, y) = \sum_{k=0}^{\infty} \frac{(-1)^k}{k!} x^k y^k = \phi(x)^\top \phi(y) \tag{10}$$

where $\phi(x) = \left(\frac{i^k}{\sqrt{k!}} x^k \right)_{k \in \mathbb{N}}$

2. (1 point) What is the dimensionality of the feature space you proposed?

Solution: The feature space is infinite-dimensional.

5 One Hidden Layer Neural Network Approximation [4 points]

From the lectures you know that 1-NNs are universal approximators for any sufficiently smooth function.

A simple one-hidden-layer neural network with output dimension 1 can be written as:

$$f(x) \;=\; v^T\,\varphi\big(Wx + b\big) \;+\; c,$$

where:

- $x \in \mathbb{R}^d$ is the input vector.
- $W \in \mathbb{R}^{h\times d}$ is the weight matrix into the hidden layer, where h is the number of hidden neurons.
- $b \in \mathbb{R}^h$ is the bias vector for the hidden layer.
- $v \in \mathbb{R}^h$ is the output weight vector.
- $c \in \mathbb{R}$ is the output bias (scalar).
- φ is an activation function applied elementwise, e.g., the ReLU activation $\mathrm{ReLU}(z) = \max\{0, z\}$.

Given the scalar function

$$f^*(x) = x$$

where $x \in \mathbb{R}$, construct a one-hidden-layer neural network $f(x)$ with a ReLU activation $\varphi = \mathrm{ReLU}$, such that $f(x) = f^*(x)$ exactly. Explicitly provide:

1. The full weight matrix W, bias vector b, output weight vector v, and scalar bias c.
2. The number of hidden neurons (h).
3. A demonstration of how the neural network computes $f(x) = x$.

Solution:

The key idea is to use the identity $x = \max\{0, x\} - \max\{0, -x\}$. That is, $x = \mathrm{ReLU}(x) - \mathrm{ReLU}(-x)$.

Let $x \in \mathbb{R}$ (scalar). Then a one-hidden-layer network with 2 neurons suffices:

$$\varphi(z) = \max\{0, z\}, \quad \text{Hidden layer has two units: } h_1 = \mathrm{ReLU}(x),\; h_2 = \mathrm{ReLU}(-x).$$

We can represent $h_1 = \mathrm{ReLU}(w_1 x + b_1)$ with $w_1 = 1, b_1 = 0$, and $h_2 = \mathrm{ReLU}(w_2 x + b_2)$ with $w_2 = -1, b_2 = 0$. The output layer is $v = [1,\; -1]^T$ (so that $v_1 h_1 + v_2 h_2 = h_1 - h_2$) and $c = 0$. Thus

$$f(x) \;=\; 1 \cdot \mathrm{ReLU}(x) \;+\; (-1) \cdot \mathrm{ReLU}(-x) \;+\; 0 \;=\; \max\{0, x\} - \max\{0, -x\} \;=\; x,$$

exactly. This confirms $f(x) = x$ is representable by a 2-neuron ReLU net.

Details of parameter shapes:

- $W \in \mathbb{R}^{2\times 1}$ can be $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$, $b \in \mathbb{R}^2$ is (0, 0).
- φ is ReLU applied componentwise, giving $\big(\mathrm{ReLU}(x),\, \mathrm{ReLU}(-x)\big)$ in the hidden layer.
- $v \in \mathbb{R}^2$ is (1, −1), $c = 0$.

Hence $f(x) = v^T\,\varphi(Wx + b) + c = x$.

6 Convolutional neural networks [5 points]

Consider the 1-d convolution of an input $\mathbf{x} \in \mathbb{R}^D$ with a filter $\mathbf{h} \in \mathbb{R}^K$. Define a convolution operation as

$$(\mathbf{x} * \mathbf{h})_k = \sum_{i=1}^{K} x_{K+k-i} h_i \quad \text{for } k \in \{1, ..., D-K+1\} \tag{11}$$

For fixed $\mathbf{h}$ we can thus view $\mathbf{x} * \mathbf{h}$ as a 1-d convolutional layer in a neural network

$$f_{\boldsymbol{h}} : \mathbb{R}^D \to \mathbb{R}^{D-K+1}, \quad \text{with} \quad f_{\boldsymbol{h}}(\boldsymbol{x}) = \boldsymbol{x} * \boldsymbol{h} \tag{12}$$

1. (3 points) For $D = 5$ and $K = 3$, rewrite this operation as a fully connected layer, namely find a matrix $\mathbf{W_h} \in \mathbb{R}^{(D-K+1)\times D}$ such that you can write

$$f_{\mathbf{h}}(\boldsymbol{x}) = \boldsymbol{W_h x} \quad \forall \boldsymbol{x} \tag{13}$$

Solution:

$$\boldsymbol{W_h} = \underbrace{\begin{pmatrix} - & \bar{\boldsymbol{h}} & - & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & - & \bar{\boldsymbol{h}} & - & 0 & \dots & 0 & 0 & 0 & \\ 0 & 0 & - & \bar{\boldsymbol{h}} & - & \dots & 0 & 0 & 0 & \\ 0 & 0 & 0 & - & \bar{\boldsymbol{h}} & - & 0 & \dots & 0 & 0 \\ \vdots & & & & & \ddots & & & \vdots & \\ \vdots & & & & & & \ddots & & \vdots & \\ \vdots & & & & & & & \ddots & & \vdots \\ 0 & 0 & 0 & \dots & 0 & 0 & 0 & - & \bar{\boldsymbol{h}} & - \end{pmatrix}}_{D}$$

where $\bar{\boldsymbol{h}} \in \mathbb{R}^D$ such that $h_i = h_{K+1-i}$. Consider $\boldsymbol{h} = \begin{pmatrix} h_1 \\ h_2 \\ h_3 \end{pmatrix} \in \mathbb{R}^3$ and $\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \in \mathbb{R}^5$.

We can express the first entries of $\boldsymbol{x} * \boldsymbol{h}$ as
1. $(\boldsymbol{x} * \boldsymbol{h})_1 = \sum_{i=1}^{3} x_{3+1-i} h_i = x_3 h_1 + x_2 h_2 + x_1 h_3$
2. $(\boldsymbol{x} * \boldsymbol{h})_2 = \sum_{i=1}^{3} x_{3+2-i} h_i = x_4 h_1 + x_3 h_2 + x_2 h_3$
3. $(\boldsymbol{x} * \boldsymbol{h})_3 = \sum_{i=1}^{3} x_{3+3-i} h_i = x_5 h_1 + x_4 h_2 + x_3 h_3$

Or,more compactly: $\boldsymbol{x} * \boldsymbol{h} = \underbrace{\begin{pmatrix} h_3 & h_2 & h_1 & 0 & 0 \\ 0 & h_3 & h_2 & h_1 & 0 \\ 0 & 0 & h_3 & h_2 & h_1 \end{pmatrix}}_{\boldsymbol{W_h}} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}$

2. (2 points) For general D and K, compare the layer $f_{\boldsymbol{h}} : \mathbb{R}^D \to \mathbb{R}^{D-K+1}$, $f_{\boldsymbol{h}}(\boldsymbol{x}) = \boldsymbol{W_h x}$ with a general fully connected layer $f_{\text{FC}} : \mathbb{R}^D \to \mathbb{R}^{D-K+1}$, $f_{\text{FC}}(\boldsymbol{x}) = \boldsymbol{W x}$ for some matrix $\boldsymbol{W} \in \mathbb{R}^{(D-K+1)\times D}$. What are the numbers of different learnable parameters in f_{FC} and in $f_{\mathbf{h}}$?

Solution: For a general fully connected layer parametrized by $\mathbf{W} \in \mathbb{R}^{(D-K+1)\times D}$, we have $D(D-K+1)$ free parameters by definition (the number of entries in $\mathbf{W}$). For $\mathbf{W_h}$, observe that it may have the same number of entries as $\mathbf{W}$ but we fill all of those entries by the entries of $\mathbf{h}$ and some zeros. The parameters are "shared" because we re-use them multiple times. We hence only have $dim(\mathbf{h}) = K$ parameters.

7 Coding and Course Questions: [8 points]

1. (2 points) *(Random number generators)* Complete the sequence of numbers output by the random generator in numpy. Put a question mark where it is not possible to predict the output.

```
rs = np.random.RandomState(18)
print(rs.randint(1, 11))
print(rs.randint(1, 11))
rs2 = np.random.RandomState(12)
print(rs2.randint(1, 11))
rs3 = np.random.RandomState(18)
print(rs3.randint(1, 11))
print(np.random.randint(1, 11))
print(rs3.randint(1, 11))
print(rs3.randint(1, 11))
```

```
>> 3 9 _ _ _ _ _
```

Solution: `>> 3 9 ?  3 ?  9 ?`

2. (2 points) *(Word/Token embeddings)* You defined the following code for your language model. What is the result of the print statement, i.e. the dimensions of the input and the embedding?

```
x = torch.tensor([[1,3,5,1]],dtype=torch.long)
model_dim = 32
T = 12
f = nn.Embedding(T, model_dim)
embd = f(x)
print(x.shape, embd.shape)
```

Solution: It would print `(1,4),(1,4,32)`.

3. (2 points) *(Attention matrices)* Given a data matrix $X \in \mathbb{R}^{L \times d}$ and three $d \times d$ real matrices W_q, W_k and W_v, write the expression for queries, keys and values and then write the expression of the (single-head) self-attention mechanism. Which part of this expression is the attention matrix and what is its size?

Solution: The queries, keys and values can be calculated as $Q = XW_q$,$K = XW_k$,$V = XW_v$. The attention mechanism is then obtained by: $\text{softmax}(\frac{QK^T}{\sqrt{d}})V$. Attention matrix $\text{softmax}(\frac{QK^T}{\sqrt{d}})$ is $L \times L$.

4. (2 points) *(Research community practices)* Imagine a global competition to achieve the best test score on the BildNetz dataset, focused on classifying objects in images. Each year, research groups develop innovative neural network components, training only on the training data. They submit their models and new components to a consortium which has private access to some test data. The component that lead to the highest test score each year is adopted by everyone else for the next iteration of the competition.

   After 10 years of this process, the community has discovered ten highly effective components, and test scores found by the consortium are now nearly 100%. The consortium then creates a completely new dataset and tests the model composed of all ten innovative components on this new data.

   Would you expect the test score on the new dataset to be similar to the original test score? Explain your reasoning.

Solution: The score on the new dataset is unlikely to be as high as on the original test dataset. Over time, the model components became implicitly optimized for the patterns in the original test data, since they were selected based on the test scores. As a result, the 10 components we found overfit to the original test dataset and are less effective on new, unseen data.

8 Improving outputs of ChatGPT-like generative language models [13 points]

You set up an example regression problem in your code as below, where you define a neural network with a single hidden layer and ReLU activation, as well as the dataloaders needed for subsequent training.

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import TensorDataset, DataLoader

hidden_neurons = 100  # Number of neurons in the hidden layer
input_dim = 10  # Input dimension (e.g., 10 features)
batch_size = 32  # Batch size
learning_rate = 0.01  # Learning rate
epochs = 20  # Number of epochs


model = nn.Sequential(
        nn.Linear(input_dim, hidden_neurons),
        nn.ReLU(),
        nn.Linear(hidden_neurons, output_dim)
    )

# Generate random data
train_samples = 1000  # Number of training samples
test_samples = 200  # Number of testing samples

# Create random features and targets
X_train = torch.randn(train_samples, input_dim)
y_train = torch.randn(train_samples, 1)
X_test = torch.randn(test_samples, input_dim)
y_test = torch.randn(test_samples, 1)

# Create datasets
train_dataset = TensorDataset(X_train, y_train)
test_dataset = TensorDataset(X_test, y_test)

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

⇒ exercise continues on next page

You now prompt a ChatGPT-like generative model with the following prompt: *Create a function 'train' for training the model via SGD with a given learning rate and with batches from the given training loader for the given number of epochs using an MSE loss. Every epoch the model should print the epoch and the average loss on the training and test data respectively for that epoch. It should return both train and test loss at the end of training.*

It spits out the following code:

```
# Training function
def train(model, train_loader, test_loader, epochs, learning_rate):

    assert learning_rate > 0, "Learning rate must be positive"

    criterion = nn.MSELoss()  # Mean Squared Error loss for regression
    optimizer = optim.SGD(model.parameters(), lr=learning_rate)
    # Stochastic Gradient Descent optimizer

    for epoch in range(epochs):
        model.train() # Put model in train mode
        train_loss = 0.0

        # Training loop
        # (A)
        for inputs, targets in train_loader:
            # (B)
            outputs = model(inputs)
            # (C)
            loss = criterion(outputs, targets)
            # (D)
            loss.backward()
            # (E)
            optimizer.step()
            # (F)
            train_loss += loss.item()
        # (G)

        train_loss = float(train_loss) # convert to float

        # Evaluate on test data
        model.eval() # Put model in evaluation mode
        test_loss = 0.0
        for inputs, targets in test_loader:
            outputs = model(inputs)
            loss = criterion(outputs, targets)
            test_loss += loss.item()

        test_loss = float(test_loss)  # convert to float

        # Print progress
        print(f"Epoch {epoch + 1}/{epochs}
        - Train Loss: {train_loss/len(train_loader.dataset)}
        - Test Loss: {test_loss/len(test_loader)}")

    # print('Inside:',model[0].bias[0])
    return train_loss, test_loss
```

⇒ exercise continues on next page

You run the code but it does not work, and also you realize that you could make a couple of improvements

1. (2 points) Start by removing obsolete stuff: Which five lines (excluding empty lines and comments) are not absolutely necessary to execute the task that you outlined in your prompt? (you can only remove full lines and not parts of them)

   Solution: 4, 11, 29, 32, 39

2. (2 points) You realize that the generative model did not put the `optimizer.zero_grad()` line that you need to avoid accumulating gradients in the computational graph. You want to insert this line back into the code to fix it. In which of the lines (A)-(G) would it fix the code? (Hint: There is more than one possibility, and you need to give **all** of them.)

   Solution: (B), (C), (D), (F)

3. (2 points) Check lines 42-44. Is there a mistake? If yes, explain what is wrong and why, and propose a new line that works.

   Solution: Yes, there is a mistake, since the train and test loss are not normalized in the same way. Since the MSELoss is already averaging over the batch size, we only need to normalize by the number of batches, i.e. we should replace the `len(train_loader.dataset)` with `len(train_loader)`.

4. (3 points) You execute the code below, where you print the first item of the bias parameter of the first linear layer *before* and *after* the training. Assume you also uncomment line 46 in the train function, so you also print the parameter in the `train` function itself.

```
print('Before:',model[0].bias[0])
train_loss, test_loss = train(model, train_loader, test_loader, epochs, learning_rate)
print('After:',model[0].bias[0])
```

You first observe

```
Before:0.43212
Inside:-0.11901
```

Which line would follow, `After:0.43212` or `After:-0.11901`? Explain how you reached this conclusion.

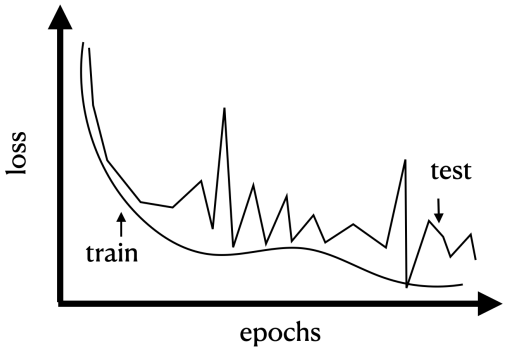
Solution: We will see `After:-0.11901`. This is because we changed the model's parameters inside of the train function. When calling it again after the function, we follow the same reference, so we get the same value that is the one updated by using stochastic gradient descent.

5. (2 points) You want to add early stopping to the train function instead of fixing the number of epochs beforehand, i.e. stopping immediately when test loss of one epoch is larger than that of the previous one. Describe in words how you would implement it, and in which lines you would add/remove code.

Solution: Early stopping means that we continue to train as long as the test error does not increase. We would therefore

- add a variable that keeps track of the previous epochs test loss, initialized to infinity
- change the for loop to a while loop, and exit it on the condition that the previous test loss is smaller than the current test loss

6. (2 points) Your early stopping implementation gives you models with very high losses compared to what you had for fixed epochs. To find out why, you check a plot of the initial 100 epochs of your train and test losses. Can you explain why your implementation of the early stopping did not lead to satisfying results? How could you change your implementation to handle test losses with such high variability?



Solution: Because of the variability in the test loss my early stopping implementation would have stopped right after a few epochs, because it often happens that the test loss increases for this function. In order to handle test losses with high variability, we should take a weighted average of the past losses and see if it is much lower than the current average.

9 Boltzmann Machines [13 Points]

We define a Boltzmann Machine that models strings of lengths L of binary variables $\mathbf{x} \in \{0,1\}^L$. The energy function is given by

$$E_{\mathbf{b},W}(\mathbf{x}) = -\sum_{i=1}^{L} b_i x_i \; - \sum_{1 \le i < j \le L} W_{ij}\, x_i x_j.$$

Here, $\mathbf{b}$ are local fields and W_{ij} are pairwise coupling terms.

1. (1 point) Which of the following statements about a Boltzmann Machine is **correct**? Only one is correct.
   (A) It assumes each spin is fully independent of every other.
   (B) It can capture pairwise interactions among all variables, but requires sampling from a possibly complex probability distribution.
   (C) It is guaranteed to converge to the global optimum by iterating gradient descent.
   (D) It cannot represent binary variables, only real-valued ones.

   Solution: (B) is correct.

2. (3 points) We train the Boltzmann machine by maximizing the likelihood of observed data $\tilde{X} \in \{0,1\}^{n \times L}$. Write the likelihood $\mathscr{L}(b, W)$.

   Solution: Define the normalization $Z(b,W) = \sum_v e^{-E_{\mathbf{b},W}(x)}$ the likelihood is then

   $$\mathscr{L}(b,W) = \frac{1}{Z^n(b,W)} \prod_{\mu=1}^{n} e^{-E_{\mathbf{b},W}(\mathbf{x}_\mu)} \tag{14}$$

3. (1 point) Gradient updates on this likelihood with respect to the parameters b_i and W_{ij} lead to the following expressions

   $$\frac{\partial \mathscr{L}}{\partial b_i} \propto \langle x_i \rangle_{\text{data}} - \langle x_i \rangle_{\text{model}}, \quad \frac{\partial \mathscr{L}}{\partial W_{ij}} \propto \langle x_i x_j \rangle_{\text{data}} - \langle x_i x_j \rangle_{\text{model}}.$$

   So

   $$b_i \leftarrow b_i + \eta\big(\langle x_i \rangle_{\text{data}} - \langle x_i \rangle_{\text{model}}\big), \quad W_{ij} \leftarrow W_{ij} + \eta\big(\langle x_i x_j \rangle_{\text{data}} - \langle x_i x_j \rangle_{\text{model}}\big).$$

   Write how do you compute the terms $\langle x_i \rangle_{\text{data}}$ and $\langle x_i x_j \rangle_{\text{data}}$. Write their expressions in terms of the data $\tilde{X}$ and $\mathbf{b}, W$.

   Solution: $\langle x_i \rangle_{\text{data}} = \frac{1}{n} \sum_{\mu=1}^{n} \tilde{x}_{\mu i}$ and $\langle x_i x_j \rangle_{\text{data}} = \frac{1}{n} \sum_{\mu=1}^{n} \tilde{x}_{\mu i} x_{\mu j}$

4. (2 points) Still referring to the gradient updates above, write how you compute the terms $\langle x_i \rangle_{\text{model}}$and $\langle x_i x_j \rangle_{\text{model}}$. Describe in words the algorithm you would use to compute them.

> **Solution:** $\langle x_i \rangle_{\text{model}}$ is obtained by *sampling* from the model distribution $p_{\text{b},W}(x) = \frac{1}{Z(\mathbf{b},W)} \exp(-E_{\mathbf{b},W}(x))$ in order to approximate $\langle \cdot \rangle_{\text{model}}$, in particular to compute $\langle x_i \rangle_{\text{model}}$ and $\langle x_i x_j \rangle_{\text{model}}$. Monte Carlo Markov Chain for the sampling.

Below is a partial code snippet for computing the energy of a Boltzmann Machine in PyTorch. The function contains errors and inefficiencies and for this reason it is called `energy_wrong`. In the next questions we ask you to identify the issues and propose corrected/optimized code.

```
1  import torch
2
3  def energy_wrong(x, b, W):
4      """
5      x: shape (batch_size, L), binary {0,1}
6      b: shape (L,)
7      W: shape (L, L), upper-triangle is the one that is relevant
8      returns energy for each sample in x
9      """
10     E = torch.zeros(x.size(0))
11
12     # (I) local fields:
13     for n in range(x.size(0)):
14         for i in range(x.size(1)):
15             E[n] += b[i] * x[n,i]
16
17     # (II) couplings:
18     for n in range(x.size(0)):
19         for i in range(x.size(1)):
20             for j in range(x.size(1)):
21                 E[n] -= W[i,j] * x[n,i] * x[n,j]
22
23     return E
```

5. (2 points) There is an error in the computation of the local fields summation in the box (I), from line 12 to 15. Indicate what the error is. What equation should you implement here? Write the code that corrects it.

> **Solution:**
>
> - The code `E[n] += b[i]*x[n,i]` has the *wrong sign* for a BM, which should add `- b[i] * x[n,i]` to E.
>
> A correct fix (assuming we want $E[n]$ to contain the $-\sum_i b_i x_i$ part):
>
> ```
> for n in range(x.size(0)):
>     for i in range(x.size(1)):
>         E[n] -= b[i] * x[n,i]
> ```
>
> This implements $E[n] \leftarrow E[n] - \sum_i b[i]x[n,i]$.

6. (2 points) The computation of the interaction term in the box (II), from line 17 to 21, is very inefficient. Replace the nested loops implementation with a line or two of code that is more efficient, e.g. using `torch.einsum` or matrix multiplication?

> **Solution:** `0.5 * torch.einsum('bi,ij,bj->b', x, W, x)`. Then *subtract* or *add* it with correct sign for the total. For instance:
>
> ```
> E = - (x * b).sum(dim=1) \
>     - 0.5*torch.einsum('bi,ij,bj->b', x, W, x)
> ```
>
> This is more efficient than triple nested loops and avoids double counting with the factor 0.5 .

7. (2 points) In the gradient descent implementation, that we do not show explicitly since it is not important to the scope of the question, assume we have a bug: the update

```
b += gamma*(pos_mean - neg_mean)
W += gamma*(pos_corr - neg_corr)
```

updates all `W[i,j]` for both `i<j` and `i>j`. The couplings are meant to be *symmetrical* in `(i,j)`. Suggest an easy fix to ensure $W_{ij} = W_{ji}$ remains consistent, and write the line of code that achieves this.

Solution: One easy approach is to enforce symmetry after the update. For instance:

```
W = 0.5*(W + W.transpose((1,0)))
```

This ensures $W_{ij} = W_{ji}$ is satisfied.