

Material/tools allowed in the exam: You are allowed to bring one A4 sheet with notes written on one side only by your own hand. No electronic devices are allowed.

Answer all five questions. Each question is worth one fifth of the total points.

Question 1 For each claim, answer either TRUE or FALSE and **justify** your answer. Half of the points are awarded for the justification, and you may get points even if the answer itself was incorrect.

1. The learning algorithm hyperparameters (learning rate, momentum, etc.) that were chosen for a particular data set are likely to be good also for a new data set if the model remains the same.
2. Stochastic gradient descent implicitly regularizes the solution in the same way as ℓ_2 regularization does, helping models avoid overfitting.
3. When re-training a model on a larger training collection, we should always expect the generalization performance to improve.
4. Adding random noise to model inputs during training can improve generalization of the model.
5. Manifold learning can be accomplished by an autoencoder.
6. An autoencoder can have exactly the same $\mathbf{x}$ both as input and as target output during training.
7. Given an unnormalized pdf q , it can be normalized by multiplying it by $\int q(\boldsymbol{\xi}) d\boldsymbol{\xi}$.
8. Noise-contrastive estimation is based on discriminating true data $\mathbf{x}$ from noisy data $\mathbf{x} + \mathbf{n}$.
9. The problem of non-identifiability in an autoencoder means that the target outputs cannot be accurately reconstructed from the input.
10. Linear ICA achieves identifiability by assuming the latent variables are Gaussian.

Question 2 A dog shelter owner gives you a labeled collection of 500 photographs $\mathbf{x}_i$ of packs of stray dogs, each with a label vector $\mathbf{y}_i \in \{0, 1\}^{20}$ that tells which of 20 different possible dog breeds are seen in the image. Your task is to create a neural network $f(\mathbf{x})$ that can identify all dog breeds in similar images.

Assume you already have a neural network $g(\mathbf{x})$ that was trained for image classification on a collection of millions of images. It takes as input images of the same size as yours and outputs a probability vector $\mathbf{p}_i \in \mathbb{R}^{1000}$ (that sums to one for each i) of class labels (objects, animal species, etc.).

- (a) Mention **two** different neural network architectures covered during the course that could be good for $f(\mathbf{x})$. Giving the name is enough; no need to explain the architecture. However, for both alternatives, characterize how the **final layer** of the network would look like in a typical case. What are the inputs and outputs of the final layer? What activation function and loss you would use for training the networks?
- (b) You want to use $g(\mathbf{x})$ to make learning a good $f(\mathbf{x})$ easier. Give a name for this idea and explain what kind of benefits you can expect. Then explain **one** concrete algorithm for doing this. Provide detailed enough instructions so that someone familiar with neural networks and machine learning could proceed to implement your method. Use equations and/or pseudo-code to explain the exact steps.

Question 3 Explain the code snippets, telling what they do. For (b) and (c), also write the function as equation(s). Does the code as written here implement what you think it should? If not, identify possible mistakes or issues, explaining how they could be resolved. There is no need to consider detailed syntax. Instead, focus on conceptual issues someone else using the code would need to be aware of.

- (a) `% Note: You can assume the data loading and model definition are correct`
`x, y = load_data_as_tensors("ImageNet_complete.data"); model = my_large_network()`
`loss_fn = torch.nn.CrossEntropyLoss()`
`opt = torch.optim.SGD(model.parameters(), lr=1); model.train()`
`for i in range(1000):`
`y_pred = model(x)`
`loss = loss_fn(y_pred, y)`
`opt.zero_grad()`
`loss.backward()`
`opt.step()`
- (b) `def process(X, Wq, Wk, Wv):`
`Q = Wq @ X; K = Wk @ X; V = Wv @ X`
`A = Q.T @ K`
`D = Q.shape[0]`
`A = A / np.sqrt(D)`
`Y = V @ A`
`return Y, A`
- (c) `% Note: layer1 and layer2 are instances of torch.nn.Linear() with suitable dimensions`
`def compute(self, x):`
`mid = self.layer1(x); out = torch.nn.ReLU(mid)`
`out = self.layer2(out); out = out + mid`
`return out`

Question 4 This question is about *autoencoders*.

1. Explain the basic principle of autoencoders. What is it that defines the concept of an autoencoder.
2. Explain two different kinds of autoencoders (select freely from the ones we covered). For each, give
 - (a) its motivation
 - (b) a verbal description of how it works
 - (c) the objective function (as an explicit mathematical formula)

As a very rough guideline, answer in approximately 500 words in total; there is no penalty for too long or short answers but this gives an idea of how much detail you are supposed to give.

Question 5 Consider the following *energy-based (unnormalized) model*:

$$\log p_{\text{un}}(x; \theta) = -\theta g(x) \quad (1)$$

for $x, \theta \in \mathbb{R}$, and some function $g: \mathbb{R} \rightarrow \mathbb{R}$.

1. Give the formula (as an integral) for the normalization constant for this model.
2. Explain why maximum likelihood estimation of θ would be computationally difficult.
3. Assume we have a sample $x_1, \dots, x_N$. Give the objective function of score matching for estimating θ .
4. Solve for the score matching estimator $\hat{\theta}$, resulting in an explicit analytical formula (defining just the optimization problem is not enough).