Exercises

1	2	3	4	5	6	7	8	9

Surname, First name

Deep Learning 1 (52041DEL6Y)
resit2022

1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0	1 2 3 4 5 6 7 8 9 0

Exam Cheat Sheet: Useful equations

We provide below a list of important equations:

Stochastic Gradient Descend:

$w^{(t)} = w^{(t-1)} - \eta \cdot g^{(t)}$

SGD with Momentum:

$m^{(t)} = \beta_1 m^{(t-1)} + (1-\beta_1) \cdot g^{(t)}, w^{(t)} = w^{(t-1)} - \eta \cdot m^{(t)}$

Adam:

$m^{(t)} = \beta_1 m^{(t-1)} + (1-\beta_1) \cdot g^{(t)}$

$v^{(t)} = \beta_2 v^{(t-1)} + (1-\beta_2) \cdot \left(g^{(t)}\right)^2$

$\hat{m}^{(t)} = \frac{m^{(t)}}{1-\beta_1^t}, \hat{v}^{(t)} = \frac{v^{(t)}}{1-\beta_2^t}$

$w^{(t)} = w^{(t-1)} - \frac{\eta}{\sqrt{v^{(t)}+\epsilon}} \circ \hat{m}^{(t)}$

Derivative of Sigmoid:

$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1-\sigma(x))$

Derivative of Tanh:

$\frac{\partial \tanh(x)}{\partial x} = 1 - \tanh^2(x)$

Derivative of ReLU:

$$\frac{\partial \text{relu}(x)}{\partial x} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

Derivative of ELU:

$$\frac{\partial \text{elu}(x)}{\partial x} = \begin{cases} 1 & \text{if } x > 0 \\ \exp(x) & \text{otherwise} \end{cases}$$

Xavier init:

Normal distribution: $w_{ij} \sim \mathcal{N}\left(0, \frac{2}{d_x+d_y}\right)$, Uniform distribution: $w_{ij} \sim U\left[-\frac{\sqrt{6}}{\sqrt{d_x+d_y}}, +\frac{\sqrt{6}}{\sqrt{d_x+d_y}}\right]$

Kaiming init (ReLU activation):

Normal distribution: $w_{ij} \sim \mathcal{N}\left(0, \frac{2}{d_x}\right)$, Uniform distribution: $w_{ij} \sim U\left[-\frac{2\sqrt{3}}{\sqrt{d_x}}, +\frac{2\sqrt{3}}{\sqrt{d_x}}\right]$

Variance rules for two independent random variable X and Y:

- $\text{Var}(X+Y) = \text{Var}(X) + \text{Var}(Y)$
- $\text{Var}(X \cdot Y) = \mathbb{E}(Y)^2\text{Var}(X) + \mathbb{E}(X)^2\text{Var}(Y) + \text{Var}(X)\text{Var}(Y) = \mathbb{E}(Y^2)\mathbb{E}(X^2) - \mathbb{E}(Y)^2\mathbb{E}(X)^2$

VAE ELBO:

$\log p(x) \geq \mathbb{E}_{q(z|x)}\left[\log p(x|z)\right] - KL\left(q(z|x)||p(z)\right)$

KL divergence:

$D_{\text{KL}}(q||p) = -\mathbb{E}_{q(x)}\left[\log \frac{p(x)}{q(x)}\right] = -\int q(x)\left[\log \frac{p(x)}{q(x)}\right]dx$

GAN Value Function:

$\min_G \max_D V(D,G) = \min_G \max_D \mathbb{E}_{p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{p_z(z)}[\log\left(1 - D\left(G(z)\right)\right)]$

Batch Normalization:

$\mu_j \rightarrow \frac{1}{m}\sum_{i=1}^m x_{ij}$

$\sigma_j \rightarrow \frac{1}{m}\sum_{i=1}^m (x_{ij} - \mu_j)^2$

$\hat{x_{ij}} \rightarrow \frac{(x_{ij}-\mu_j)}{\sigma_j+\epsilon}$

$\hat{x_{ij}} \rightarrow \gamma \hat{x}_{ij} + \beta$

Exam: MCQ

2p **1a** Which of the following is NOT a common approach for self-supervised learning?

- ☐ Pretraining on a large dataset and fine-tuning on a smaller task-specific dataset
- ☐ Applying self-supervision to predict the weights of a supervised network
- ☐ Using the data itself as the supervision signal
- ☐ Using hand-labeled data to train the model

2p **1b** What statements are true about contrastive learning?

- ☐ When the dataset is very large and very diverse, a higher temperature (τ) might be required.
- ☐ When the dataset is very large and very diverse, a lower temperature (τ) might be required.
- ☐ For any given sample, in self-supervised contrastive learning we define positive and negative samples. The positives are chosen in the dataloader by picking samples with the same class-label.
- ☐ When training on multiple GPUs, one should stick to a normal BatchNorm implementation.

2p **1c** What statements about generative models are true?

- ☐ The improvement in GANs largely came from using larger datasets
- ☐ Generative models can and are used for malicious purposes.
- ☐ Evaluating the quality of generative models objectively is difficult.
- ☐ The main goal of generative modelling is to learn a model that can accurately predict the value of a discrete variable

2p **1d** What is a NeRF?

- (a) A machine learning model for 3D reconstruction
- (b) A computer vision algorithm for object detection
- (c) A deep learning framework for image classification
- (d) A MLP based approach to model 3D point clouds

Exam: Quadratic Layer

For some input features $\mathbf{x} \in \mathbb{R}^M$ and output $\mathbf{y} \in \mathbb{R}^N$ let a quadratic layer be defined by:
$y_i = \lambda \mathbf{x}^\top \mathbf{A}^{(i)} \mathbf{x} + \mathbf{x}^\top \mathbf{b}^{(i)} + c^{(i)}$

Where the parameters are given by a symmetric square matrix $\mathbf{A}^{(i)} \in \mathbb{R}^{M \times M}$, a vector $\mathbf{b}^{(i)} \in \mathbb{R}^M$, and a scalar $c^{(i)} \in \mathbb{R}$. With $i \in \mathbb{N}$ and $0 \leqslant i < N$. There is also a hyperparameter $\lambda \in \mathbb{R}$. Note that one can also think of MM and NN as being the number of input and output neurons, respectively.

Hint: The number of degrees of freedom (i.e. unique numbers) in a symmetric square matrix is determined by the number of elements on or above the main diagonal.

2p **2a** What type of module is obtained when $\lambda = 0$?

- (a) Convolutional layer
- (b) BatchNorm layer
- (c) Linear activation
- (d) Max pooling

2p **2b** In the context of neural networks, how can the role of the scalar $c^{(i)}$ best be described?

- (a) Bias term
- (b) Stride
- (c) Activation function
- (d) Ensures output stays large (in order to avoid small gradients)

2p **2c** Why do the square matrices $\mathbf{A}^{(i)}$ have to be symmetric?

- (a) For the quadratic term to be well-defined (as this term is a scalar and it must equal its transpose).
- (b) To make the matrix unique (otherwise, during optimization, the parameters can get arbitrarily large).
- (c) The matrix must have real eigenvalues, or it will crash the code (which is not always optimized for complex numbers).
- (d) None of the above

We wish to compare the number of trainable parameters in this quadratic layer to the traditional linear layer. Let P_Q and P_L be the number of parameters in the quadratic and linear layers, respectively. These will depend on the number of input and output neurons.

3p **2d** What is $\frac{P_Q}{P_L}$, the ratio of the number of parameters in the quadratic layer to that of the linear layer?

- (a) $M^2 N$
- (b) N
- (c) $\frac{1}{2}M + 1$
- (d) M

A model consisting of a single quadratic layer together with a softmax output layer and categorical cross entropy loss function is used to classify MNIST (28-by-28 images of handwritten digits).

3p **2e** How many parameters does this model have?

1p **2f** Can a deep neural network consisting of quadratic layers learn to classify MNIST well?

- (a) Yes
- (b) No
- (c) It depends on whether there are activation functions

2p **2g** Below are the equations required during backpropagation for $\lambda = 1$:

$\frac{\partial L}{\partial x_i} = \sum_{j,m} \frac{\partial L}{\partial y_j}(2A_{im}^{(j)} x_m + b_i^{(j)})$

$\frac{\partial L}{\partial A_{ij}^{(k)}} = \Omega$

$\frac{\partial L}{\partial b_j^{(i)}} = \frac{\partial L}{\partial y_i} x_j$

$\frac{\partial L}{\partial c^{(i)}} = \frac{\partial L}{\partial y_i}$

What is the unknown Ω in the equations above?

- (a) $\frac{\partial L}{\partial y_k} x_i x_j$
- (b) $\frac{\partial L}{\partial y_k} x_i^2$
- (c) $\frac{\partial L}{\partial y_j} x_k x_i$
- (d) $\frac{\partial L}{\partial y_k}$

Exam: Graph Neural Networks

2p **3a** Which property should an aggregation function for graph neural networks have?

4p **3b** Give an example of two graphs $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$, and initial node embeddings H_1 and H_2, such that for two nodes $v_1 \in V_1$ and $v_2 \in V_2$ with the same initial node embeddings $h_{v_1}^{(0)} = h_{v_2}^{(0)}$ the updated features $h_{v_1}^{(1)}$, $h_{v_2}^{(1)}$ are the same if we aggregate over their neighbors with:

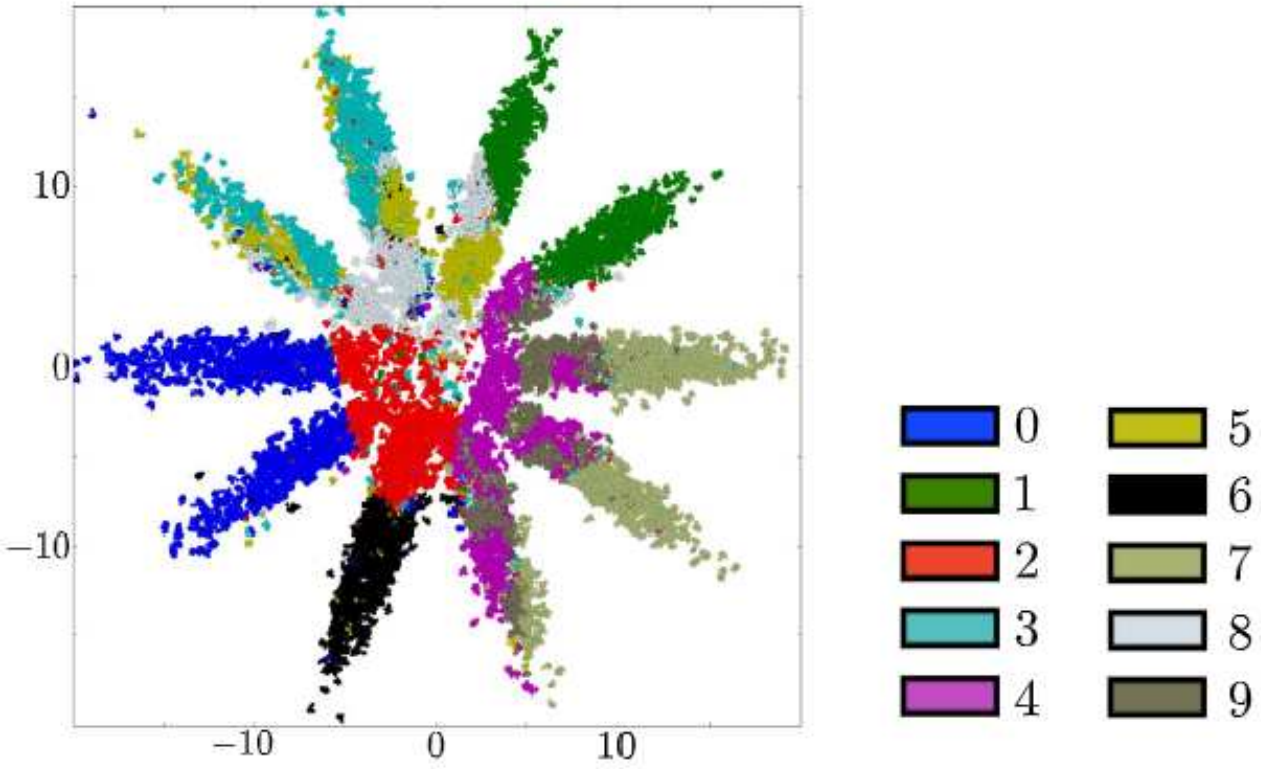
1. *mean* and *sum*, but not the same if we aggregate with *max*.
2. *max* and *mean*, but not the same if we aggregate with *sum*.

4p **3c** What is the connection between the Transformer Architecture and Graph Neural Networks? Talk about the adjacency, the GNN type and the keys and query matrices and the implied invariance/symmetry.

↳

Exam: AAE

For adversarial AutoEncoders (AAE), we are not limited to plain Gaussian priors. For instance, the latent space representation of an adversarial AutoEncoder can be a mixture of multiple Gaussians. An AAE's latent space that has been trained using a mixture of 10 2D Gaussian priors is depicted in the figure below. Here, the original label (MNIST digits ranging from 0-9) is encoded in color. From this figure, we can see that sometimes two modes are used for a single number. On the other hand, the model also shares the same Gaussian mode between different numbers.



1p **4a** Why might there be an overlap between regions that reconstruct different digits within a single Gaussian?

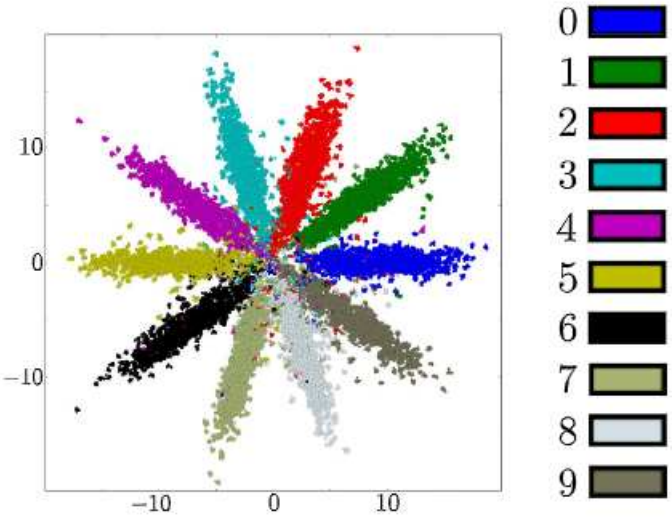
When label information is available, this can help structure the latent space of AAEs. To this end, let us consider the mixture of 10 2D-Gaussians in the previous question.

1p **4b** Why might dedicating one mode to each different digit be be more advantageous for rare categories during training time?

3p **4c** How can label information be used in the training of an AAE?

2p **4d** How can you train the AAE to benefit both from labeled and unlabelled data?

After training the network by incorporating label information for a prior distribution of a mixture of 10 2D Gaussian, we have depicted the latent presentations in the figure below:



2p **4e** Explain how the mode allocation is different from unlabelled AAE.

Exam: Classification metrics

Recall and accuracy are two different metrics used to evaluate the performance of a model. Recall is a measure of a model's ability to correctly identify all relevant instances in a dataset, while accuracy is a measure of a model's ability to correctly classify instances in a dataset. In some cases, it may be more important to have a model that has a high recall because it is better at identifying all relevant instances, even if that means that the model is not as accurate overall.

2p **5a** There are several common loss functions that are used for training a classification neural network. Very often we use the cross-entropy loss. Even in use cases where we care about having a high recall, we still use the cross-entropy loss, and not recall as the optimization metric. Why is this the case?

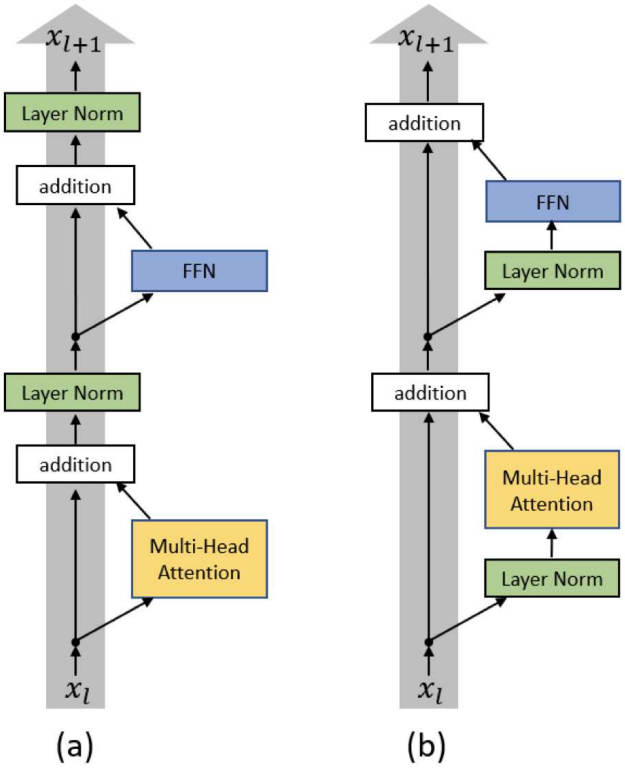
2p **5b** Suppose you have N classes, but you care about the k-th class having a high recall. How would you modify the cross-entropy loss to potentially improve the recall of the k-th class? Write down explicitly the modified version of the loss function, assuming you have a set of weights w_j for every class j. What are the appropriate values of the weight w_k? Note that weights w_j are not trainable parameters, but rather scalars used to modify the loss function.

Hint: make use of the weighted sum.

2p **5c** Now consider the following limit where the ratio $w_k/w_j \to \infty, \forall j \neq k$. In this limit, what is the final accuracy of the model? What is the final recall of this model? Express your answer in terms of the number of data points belonging to each class (for example, let N_j denote the number of data points belonging to the j-th class).

Exam: Transformer/Gradients

In this question, we will take a closer look at the Transformer architecture, specifically its interaction with the Layer Normalization. Each encoder block of the Transformer consists of a Multi-Head Attention layer and a small Feed-Forward Network (FFN) (in the form of an MLP), connected in a residual manner. After each block, a LayerNorm is applied (see figure a) below. However, there has been proposals to move the LayerNorm into the residual block (b).



(figure credit - Xiong et al., 2020; you don't need to know the paper for this question)

2p **6a** For the original Transformer architecture (a), consider an input $x_l \in \mathbb{R}^{B \times T \times d_x}$ with batch size B, T time steps, and d_x features. Assuming that each input element is sampled from a standard Gaussian, i.e. $[x_l]_{ijk} \sim \mathcal{N}(0, 1)$, which distribution do we expect the elements/features of x_{l+1} to follow?

3p **6b** Now consider the alternative architecture in (b). Which distributions do we expect x_{l+1} to follow for this architecture?

You can take the following assumptions:

- The FFN and MHA block are both initialized such that the output features have the same standard deviation as the inputs that came into the FFN and MHA block respectively
- The outputs of the FFN and MHA are statistically independent of their inputs

4p **6c** Besides the feature variance, we are also interested in the scale of the gradients we obtain for each feature vector. Consider that the gradients of x_{l+1}, i.e. g_{l+1}, are sampled from $[g_{l+1}]_{ijk} \sim \mathcal{N}(0,1)$, and the original features $x_l \sim \mathcal{N}(0,1)$. Which distribution do we expect the gradients of x_l to follow for the original Transformer architecture (a)?

You can take the following assumptions:

- The gradients through the normalization statistics μ and σ in the LayerNorm can be neglected/considered to be zero.
- The FFN and MHA have been designed such that the gradients with respect to their inputs follow the same distribution (same mean and std) as the gradients with respect to their outputs.
- The gradients with respect to the inputs of the FFN and MHA block are statistically independent of the gradients of their outputs.

↳

4p **6d** Repeat the same analysis of the previous subquestion for the architecture in (b). Take the same assumption to answer what the gradient distribution of x_l is.

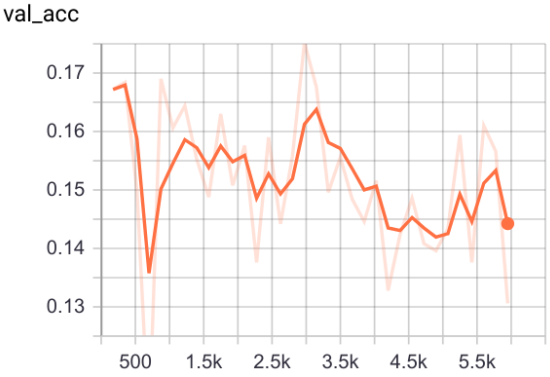
Exam: MLP-code

An assignment similar to our assignment 1, was due, yet Erin had little time to work on it. Charlie, being a friend, provided Erin some code for training a linear neural network on the CIFAR10 classification task as a very romantic "Valentine's day gift". However, it wasn't really tested properly. Now Erin is finding that the model is significantly underperforming. The code from Charlie for the network and optimiser is shown below:

```python
class MyMLP(nn.Module):

    def __init__(self, input_size=3072, output_size=10):
        super().__init__()
        self.layers = nn.Sequential(
            nn.Linear(input_size, 512),
            nn.Tanh(),
            nn.Dropout(0.1),
            nn.Linear(512, 256),
            nn.Tanh(),
            nn.Dropout(0.1),
            nn.Linear(256, 10),
            nn.Tanh()
        )
        self.softmax = nn.LogSoftmax(dim=-1)

        self._reset_parameters()

    def _reset_parameters(self):
        for n, p in self.named_parameters():
            if n.endswith("bias"):
                p.data.fill_(0)
            else:
                p.data.normal_(std=0.01)

    def forward(self, imgs):
        imgs = imgs.flatten(start_dim=1)
        logits = self.layers(imgs)
        logits = self.softmax(logits)
        return logits
```

```python
model = MyMLP()
optimizer = torch.optim.Adam(model.parameters(), lr=0.1, betas=(0.9,0.999))
```

The validation accuracy over training iterations is plotted below.



Validation performance over training iterations

4p **7a** Looking at the code above, name at least 2 reasons why the model might be underperforming. Shortly explain why, and propose a solution to solve it.
Note: You can assume that the training and test loop is coded without an error, and uses the NLL loss function. The mistakes we are looking for are not specific to PyTorch. For example, are all hyperparameters in a reasonable range?

Expected answer length: two sentences for each reason.

5p **7b** Erin and Charlie discuss the residual connections in ResNet architectures. Charlie: "In most networks I have seen, the residual connection always adds the input to the output of a layer block, i.e. $x_{l+1} = x_l + F(x_l)$. But couldn't we subtract as well instead of summing, i.e. $x_{l+1} = x_l - F(x_l)$?" Comment on Charlie's suggestion. Are we able to replace the summation with subtraction in a standard ResNet with ReLU activations on the residual connection, and train it to a similar performance than the standard ResNet? Explain your answer.

Expected answer length: 3-4 sentences.

Exam: VAE - ELBO and β-VAE

We have seen that when training a VAE we optimize the ELBO:
$\mathcal{L}_{ELBO} = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}\left[\log p_\theta(\boldsymbol{x} \mid \boldsymbol{z})\right] - KL\left(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \| p(\boldsymbol{z})\right)$

3p **8a** As seen above, $\mathcal{L}_{ELBO}$ consists out of two terms. What are these two terms typically called?

4p **8b** Briefly explain why the names of the two terms (max 5 sentences per term) provided in part a are appropriate.

Another variation of VAEs is the β-VAE in which the Kullback-Leibler divergence term is weighted with a parameter β. So that
$\mathcal{L}_\beta = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x} \mid \boldsymbol{z})] - \beta KL\left(q_\phi(\boldsymbol{z} \mid \boldsymbol{x}) \| p(\boldsymbol{z})\right)$
Note that when $\beta = 1$ is it is identical to the original VAE formulation.

4p **8c** Given that we choose the following prior: $p(\boldsymbol{z}) = \mathcal{N}(0, I_D)$
Briefly explain (max. 5 sentences) what kind of effect a high value of β would have on the trained model?

↳

4p **8d** Assume $p(x)$ is a data distribution which we wish to approximate with a parameterized distribution $q_\theta(x)$. Since the Kullback-Leibler divergence is asymmetric in its input distributions, the there are two possible expressions: $KL(p(x)\|q_\theta(x))$ and $KL(q_\theta(x)\|p(x))$.

Using the definition of the KL divergence, explain which of the above is referred to as mode-seeking and which as mean-seeking.

Exam: Optimization

Convergence Rate is defined as $max\frac{\|X_{t+1}-X^*\|}{\|X_t-X^*\|}$ in which X^* is the optimization solution.

3p **9a** For general quadratic form, which is $f(x) = \frac{1}{2}x^TAx$ calculate gradient decent for one step (learning rate is η).

5p **9b** Using the value you find in the previous section, calculate the convergence rate for $f(x)$.
Hint: You can use $A = \sum_{i=1}^{n} \lambda_i u_i u_i^T$ in which λ_is are eigenvalues of matrix A and u_is are its eigenvectors

3p **9c** What is the optimal learning rate for optimizing convergence rate (in terms of eigenvalues of the matrix A)?

2p **9d** We define the condition number for matrix A as $\kappa = \frac{\lambda_{max}}{\lambda_{min}}$, what is the optimized convergence rate of function f in terms of κ?

2p **9e** It can be proven that if we use momentum instead of gradient descent, the optimized convergence rate for the quadratic function will be $\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$.
For $\kappa = 4$, if the number of required steps for gradient descent to reach an error of δ is n, calculate the number of steps required for Momentum to reach the same error.

This page is left blank intentionally