Exercises

1	2	3	4	5	6	7	8	9	10
11									

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

Surname, First name

Deep Learning 1 (52041DEL6Y)
resit2024

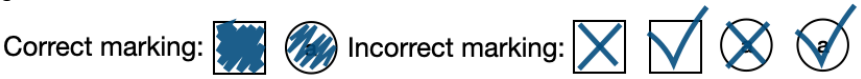
Deep Learning 1 - resit exam 2024

In order to have a smooth exam, please read the following instructions:

- Do not turn the page until instructed to do so.
- Please silence any electronic devices and put them away.
- Please put your backpack on the side of your desk.
- Do not cheat or engage in any misconduct during the exam.
- Follow all instructions carefully and read each question carefully before answering.
- If you need to leave the room for any reason, inform a proctor first.
- Once the time is up:
  - Please put down your exams on one side of the table
  - Please bring your exam to the front

Some further instructions regarding the answers in the paper:

- Write your name and student number on the front page (don't forget to mark the digits)
- Write the number of pages you're handing in
- Write all your answers on this exam booklet
- Write your answers inside the boxes (it is ok, to slightly go over the margins though).
- Round boxes are "single-correct-choice" questions and squared boxes can have any amount of correct answers.
- You should fill in the boxes and not just check them (regarding multiple-choice and adding your ID). E.g.



- In case you want to correct your answer, clearly indicate this (e.g., by filling all boxes and using another indicator such as an arrow to indicate your choice). We can then resolve this during grading.
- The grading of the multiple-choice questions is based on partial grading; not giving all correct options will give you some points, but not all. Marking incorrect options whilst also marking correct options also reduces the nr of points!
- We work with partial grading: you can get points even if you don't manage to solve the full question.
- **If you need to empty the answer box in order to start over, ask the invigilator for a blank sticker.**
- ***Please scan over the questions before you start to get an impression of the content.***

The best of luck, you got this.

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:

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

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

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

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

MCQ Softmax

2p **1** Let $f(x)$ be the softmax operation usually used in classification problems. According to the definition of the softmax, which of these changes still gives you the same loss? $a \in \mathbb{R}$.

- ☐ $e^{-a} f(x+a)$
- ☐ $f(a \cdot x)$
- ☐ $f(x) + a$
- ☐ $\frac{1}{a} f(x)^a$

1x1 conv

1x1 convolutions are essential in deep learning. Let's explore some of their features

2p **2a** Jeremy wants to apply two convolutions with kernel sizes 3 and 5 after each other. What do you recommend him do to save compute?

3p **2b** Let's define a "1x1 *k*-MLP" a module that applies *k* linear layers each followed by a non-linearity. What is true?

- ☐ This 1x1 *k*-MLP can be applied to learn deep features from the raw images and should be quite efficient at it.
- ☐ the 1x1 *k*-MLP can be used to downsample deep features by applying strides >1
- ☐ if it has hidden dimension (incoming and outgoing) of *d,* overall it either has learnable parameters of d^k or $d^2k + k$
- ☐ Imagine applying this 1x1 *k*-MLP (let's call it *m)* on a set of features *f* multiple times: m(m(f)). This would not raise issues with pytorch's backprop engine.

2p **2c** Hypernetworks are neural networks that output or modify a neural network's weights based on e.g. some domain signal *x*. Imagine a hypernetwork *h* has been trained to output the weights of the last two layers of network pretrained to generate images of satellite images. Consider a 1x1 conv *c(),* that we apply after the hypernetwork *c(h(x))*.

- ☐ Provided a learning signal, the 1x1 conv can only linearly affect the output of the generative model
- ☐ Given data that only consists of domains *x*, and corresponding satellite images *I*, you also need to know what architecture the generative model should have to train c()
- ☐ One advantage of learning *c()* only for the last layer is that we can interpret the values of *c()* in terms of how they modify the final generated image's RGB values.

2p **2d** Imagine the following architecture: You first apply a large 32x32 convolution (+ nonlinearity) on an image, then you repeat reshaping the output to be of size (dim, $N\hat{2}$) and applying a 1x1 convolution (+ nonlinearity) and reshaping it back to (N,N,dim) to apply a different 1x1 convolution (+ nonlinearity).

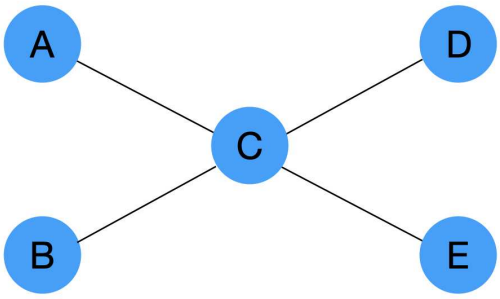
- ☐ This model achieves a global receptive field by the reshaping operation
- ☐ This model is faster or equal compared to a CNN at an equal FLOP count
- ☐ If we change the first 1x1 conv (which is applied on the (dim, $N\hat{2}$) features) to be a grouped 1x1 convolution with $N\hat{2}$ groups, this operation is the same as an attention-operation with k=q and v=1

Understanding the Graph Laplacian

In this question, we will explore the concept of the graph Laplacian and its implications in different graph structures. The graph Laplacian is a matrix representation that captures the structure of a graph and is crucial in various applications, including deep learning on graph-structured data.

1p **3a Basic Structure in Graphs**

Consider the graph provided below, which consists of four nodes labeled A, B, C, and D.



which adjacency matrix is correct (ordering is A,B,C,D,E)?

(a)
$$\begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix}$$

(b)
$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

(c)
$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \end{bmatrix}$$

(d)
$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

1p **3b** Compute M^2. What is entry $M^2_{A,D}$ + $M^2_{C,D}$?

(a) 1 (b) 2 (c) 0 (d) 3

2p **3c** What do the entries of M^k mean?

1p **3d** Now consider a graph with self-connections (going from one node to itself). Name an example real-life application where these would be useful?

In the next sections, we'll explore the graph Laplacian in three different graph configurations: a fully connected graph with uniform features, a linearly connected graph with increasing features, and a circular graph structure. These scenarios will help us understand how the Laplacian responds to various types of connectivity and feature distributions in graphs, which is crucial for deep learning applications involving graph-structured data.

2p **3e Fully Connected Graph with Uniform Features**
Now, assume the graph is fully connected and the features at each node are identical. What values does the graph Laplacian yield for this configuration, and why?

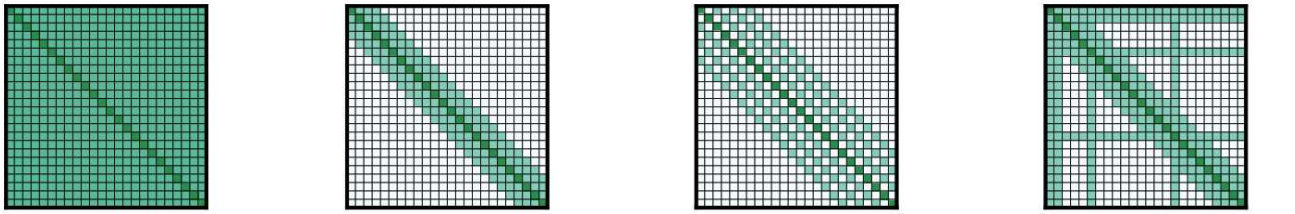
2p **3f Linearly Connected Nodes with Linearly Increasing Features**

Consider a new graph where the nodes are connected in a 1-D line (A → B → C → D) and the features at each node increase linearly. The features are given by [10,1], [20,2], [30,3], [40,4]. Discuss the result of the Laplacian in this scenario.

3p **3g Circular Graph Structure**

Finally, modify the graph from part (c) by connecting the last node (D) back to the first node (A), forming a circle. Explain what happens to the Laplacian due to this change and why.

Attention and transformers



(a) Full n^2 attention (b) Sliding window attention (c) Dilated sliding window (d) Global+sliding window

Figure: Attention Mechanism. Comparing the full self-attention pattern and the configuration of attention patterns where we deal with 1D sequences.

The original Transformer model has a self-attention component with $O\left(n^2\right)$ time and memory complexity where n is the input sequence length. To address this challenge, several attention mechanisms, e.g. Sliding window attention, Dilated sliding window attention, and Global+sliding window attention, are introduced by sparsifing the full self-attention matrix according to an “attention pattern” specifying pairs of input locations attending to one another. Unlike full self-attention, these proposed attention patterns scale linearly with the input sequence, making it efficient for longer sequences.

2p **4a** Compare Full n^2 attention with causal attention used in the GPT-style model. Draw a sketch, as in the above figure, illustrating the causal attention matrix for an input with length $n = 2$.

2p **4b** **Sliding window attention:** Given a fixed window size w, each token attends to $\left(\frac{1}{2}\right) w$ tokens on each side. In a transformer with l layers, what is the receptive field size at the top layer? (assuming w is fixed for all layers).

(a) *lw*
(b) *2lw*
(c) *0.5lw*
(d) w

2p **4c** **Dilated sliding window attention:** Assuming a fixed d, referred to dilation size, and w for all layers, what is the receptive field size at the top layer?

- ☐ $(d + w)\, l$
- ☐ $(d + 0.5w)l$
- ☐ dwl
- ☐ $(lw)^d$

2p **4d** What is true?

- ☐ (b) does not reduce the number of learnable parameters
- ☐ (d) corresponds to the vision transformer (ViT)'s attention pattern
- ☐ the attention values of (b) yield a circulant matrix and can therefore be computed efficiently
- ☐ the values of the entries at *(i,j)* for the attention matrix from (a) and (b) differ by a factor of $\frac{N}{\sqrt{(w)}}$, where *N* is the number of inputs

Consider auto-regressive generation using a transformer, like GPT-2. To speed up computation, we can store a memory of previous k,v features ("kv-caching")

3p **4e** How does that save compute? And why are we not saving the query values?

2p **4f** How much compute does this save when generating a sentence with *s* words? Consider a full-forward pass to be of compute cost *C*

2p **4g** Consider the case of you working at OpenAI and hosting a language model that answers the queries of multiple users on a server. What problems would you encounter with this approach if the batch-size is larger than 1?

Initialization

Consider the computation between adjacent pre-activations $\mathbf{f}$ (input layer) and $\mathbf{f}'$ (output layer) that are expressed as follows:

$$\mathbf{h} = a(\mathbf{f})$$
$$\mathbf{f}' = \boldsymbol{\beta} + \mathbf{W}\mathbf{h}$$

with $a()$ the activation function and **W**, $\boldsymbol{\beta}$ represents the weights and biases.

Assume the pre-activations f_j in the input layer $\mathbf{f}$ have variance σ_f^2. Consider initializing the biases β_i to zero and the weights W_{ij} as normally distributed with mean zero and variance σ_W^2.

2p **5a** Derive the mean expression of the pre-activations f_i' in the output layer.

Assume that the distributions between the hidden units and the network weights are independent. The dimensionality of the hidden layer can be denoted as D_h.

2p **5b** Derive the variance $\sigma_{f'}^2$, for the pre-activations f_i' in the output layer.

Assume that the distributions between the hidden units and the network weights are independent. The dimensionality of the hidden layer can be denoted as D_h.

Hint; make use of the variance identity:

$$\boldsymbol{\sigma}_z^2 = Var[z] = E[(z - E[z])^2] = E[z^2] - E[z]^2$$

.

↳

2p **5c** Assuming that the input distribution of pre-activations f_i is symmetric about zero, half of these pre-activations will be clipped by the ReLU function, thus, $\mathbb{E}[h_j^2]$ will be half the variance of f_j. Find an expression that connects the variances from f and f'.

2p **5d** Given the previous expression how you would initialize the weights $\mathbf{W}$ for forward and backward pass?

2p **5e** What would you expect to happen if we initialize all the weights and the biases of a neural network to zero?

Optimization basics

2p **6a** Name two advantages of stochastic gradient descent over vanilla gradient descent.

3p **6b** Describe (i) what a **pathological curve** (i.e. ravine) is, (ii) why they may be problematic during optimization using vanilla (stochastic) gradient descent, and (iii) a way to address this problem.

2p **6c** Despite the popularity of the Adam optimizer, there are scenarios in which it is preferable to use SGD over Adam. Give one possible reason why it may be preferable to use SGD over Adam in some scenarios.

2p **6d** Second-order optimization method such as Newton's method take the curvature around a point into account in order to update the weights in more promising directions. Why are second-order methods nevertheless rarely used in practice?

Theory: Init & Bn

You are training a deep, dense neural network on some flat input X with feature dimensionality of D. Your network has L hidden layers, where $L > 50$, and you use *ReLU* activation and Kaiming-initialization.

2p **7a** At *i*-th layer, your activation output is 32x128, where 32 is the batch size and 128 is the hidden features' dimensionality. You would like to add a *BatchNormalization* layer here. How many parameters in your *BatchNormalization* layer will be trainable, and how many will be not?

2p **7b** Assume you managed to implement the *BatchNormalization* layer, and you also extended the idea to all other layers, so now each layer block is a combination of $Linear \rightarrow Bn \rightarrow ReLU$ respectively. What change do you expect in your training and why?

2p **7c** We typically subtract the mean and average from our input data. What happens if we instead apply a BatchNorm layer as the very first layer?

2p **7d** Now consider applying a LayerNorm as the first layer in a transformer (e.g. a ViT). What would you expect to happen?

2p **7e** Different norm layers: which statement(s) are true?

- ☐ LayerNorm and BatchNorm are linear layers, ie can be expressed as an affine transformation of the incoming features.
- ☐ InstanceNorm and LayerNorm do not provide the same gradient after re-scaling the activations with $\frac{1}{\sqrt{(d)}}$, where *d* is the dimensionality
- ☐ Larger batch sizes improve the mean and variance estimates of LayerNorm
- ☐ It is better to initialise BatchNorm layers' running estimate values with random values than with mean=0 and variance=1

CNNs

2p **8a** Consider applying five 3x3 convolutions with padding=1, stride=1, one after another (with some nonlinearity in between). What is the resulting feature size for an incoming feature of size 224x224x*D*? Assume that the hidden-layer dimensionality *D* stays constant.

2p **8b** What is the size of the receptive field of a 11x11 convolution (stride=4, padding=2) followed by maxpooling of size 3x3 (padding=2)?

(a) 14 (b) 12 (c) 19 (d) 21

2p **8c** ConvNets

- ☐ After training, it is always possible to turn a ResNet (with only common hidden dimensionality size) into a network without any "skip-connections" without any loss in accuracy
- ☐ Depthwise convolutions allow for higher throughput and latency by splitting a convolution into the channel and spatial domains
- ☐ One difference between faster R-CNN and R-CNN lies in the former model's faster computation of region proposals straight from the RGB image
- ☐ Starting from a detection model (e.g. faster R-CNN), semantic segmentation for *K* classes (e.g. Mask R-CNN) is usually obtained by learning a fully convolutional head which outputs *K+1* dimensional maps for each box

LoRA

Consider a fully connected layer without bias, consisting of a single matrix multiply $W \in \mathbb{R}^{a,b}$, with incoming dimensionality *a,* and outgoing dimensionality *b.* We call L=AB a low-rank adaptation (LoRA), where $A \in \mathbb{R}^{a,r}$, $B \in \mathbb{R}^{r,b}$, if *r«min(a,b).* This is used in a way to modify the output as *y' = Wx + ABx,* where *x* is the input feature vector and *W* stays frozen.

2p **9a** What is the number of parameters that need to be fine-tuned with LoRA (p) compared to the number of parameters that are require gradients when normally fine-tuning *W (P)?*

- ☐ $\frac{p}{P} = \frac{r\left(1+\frac{b}{a}\right)}{b}$
- ☐ $\frac{p}{P} = \frac{r\left(1+\frac{a}{b}\right)}{b}$
- ☐ $\frac{p}{P} = r^2$
- ☐ $\frac{p}{P} = \frac{r\left(a^2-b^2\right)}{ab(a-b)}$

4p **9b** Imagine applying this adaptation at every fully connected layer of a pretrained transformer model. Describe the advantage of adapting a pretrained model to conduct a new task this way, compared to linear probing and full-finetuning. Name 1 relative advantage and 1 relative disadvantage for each.

↳

1p **9c** Consider another case of fine-tuning a pretrained model by inserting learnable 1x1 convolutions *c()*, such that *y'=c(σ(Wx))*, where σ is a non-linearity. Why can a LoRA-adapted model be faster at inference time than this adaptation method? You can assume applying the non-linearity does not cost any time/compute.

MQA

6p **10** In a typical transformer block, we have H heads and for each head, we learn a key, query and value matrix. More recently, researchers started using a "Multi-query attention" (MQA) formulation, where only a single key and value matrix are used, no matter how many heads. While this performed not as well as the previous method, a novel approach called "Grouped Multi-query" attention (GQA) is now standard and used in models like Llama-2. The idea is similar to grouped convolutions, whereby not all dimensions are in communication with each other.

Say GQA-G refers to grouped-query with G groups. GQA-1, with a single group and therefore single key and value head, is equivalent to MQA, while GQA-H, with groups equal to number of heads, is equivalent to MHA. Give an expression for the number of multiplication and addition operations for a single transformer block with H heads and G groups and dimensionality D for GQA-H.

↷

blank space

11