Chair of Visual Computing & Artificial Intelligence
Department of Informatics
Technical University of Munich

Ecorrection
Place student sticker here

Note:

- During the attendance check a sticker containing a unique code will be put on this exam.
- This code contains a unique number that associates this exam with your registration number.
- This number is printed both next to the code and to the signature field in the attendance check list.

Introduction to Deep Learning

Exam:	IN2346 / Endterm	**Date:**	Tuesday 13th July, 2021
Examiner:	Prof. Dr. Matthias Nießner	**Time:**	17:30 – 19:00

Working instructions

- This exam consists of **16 pages** with a total of **5 problems**.
  Please make sure now that you received a complete copy of the exam.
- The total amount of achievable credits in this exam is 91 credits.
- Detaching pages from the exam is prohibited.
- Allowed resources: None
- Do not write with red or green colors

Problem 1 Multiple Choice (18 credits)

Below you can see how you can answer multiple choice questions.

Mark correct answers with a cross ☒
To undo a cross, completely fill out the answer option ■
To re-mark an option, use a human-readable marking ✕■

- For all multiple choice questions any number of answers, i.e. either zero (!), one or multiple answers can be correct.
- For each question, you'll receive 2 points if all boxes are answered correctly (i.e. correct answers are checked, wrong answers are not checked) and 0 otherwise.

1.1 Which of the following models are unsupervised learning methods?

- ☒ Auto-Encoder
- ☐ Maximum Likelihood Estimate
- ☒ K-means Clustering
- ☐ Linear regression

1.2 In which cases would you usually reduce the learning rate when training a neural network?

- ☒ When the training loss stops decreasing
- ☐ To reduce memory consumption
- ☐ After increasing the mini-batch size
- ☒ After reducing the mini-batch size

1.3 Which techniques will typically decrease your **training** loss?

- ☐ Add additional training data
- ☒ Remove data augmentation
- ☒ Add batch normalization
- ☐ Add dropout

1.4 Which techniques will typically decrease your **validation** loss?

- ☒ Add dropout
- ☒ Add additional training data
- ☐ Remove data augmentation
- ☐ Use *ReLU* activations instead of *LeakyReLU*

1.5 Which of the following are affected by multiplying the loss function by a constant positive value when using SGD?

- ☐ Memory consumption during training
- ☒ Magnitude of the gradient step
- ☐ Location of minima
- ☐ Number of mini-batches per epoch

1.6 Which of the following functions are not suitable as activation functions to add non-linearity to a network?

- ☐ $\sin(x)$
- ☒ $ReLU(x) - ReLU(-x)$
- ☐ $\log(ReLU(x) + 1)$
- ☒ $\log(ReLU(x + 1))$

1.7 Which of the following introduce non-linearity in the neural network?

- ☒ LeakyReLU with $\alpha = 0$
- ☐ Convolution
- ☒ MaxPool
- ☐ Skip connection

1.8 Compared to the L1 loss, the L2 loss...

- ☐ is robust to outliers
- ☐ is costly to compute
- ☒ has a different optimum
- ☐ will lead to sparser solutions

1.9 Which of the following datasets are NOT i.i.d. (independent and identically distributed)?

- ☐ A sequence (toss number, result) of 10,000 coin flips using biased coins with $p(\text{toss result} = 1) = 0.7$
- ☒ A set of (image, label) pairs where each image is a frame in a video and each label indicates whether that frame contains humans.
- ☒ A monthly sample of Munich's population over the past 100 years
- ☐ A set of (image, number) pairs where each image is a chest X-ray of a different human and each number represents the volume of their lungs.

Problem 2 Short Questions (29 credits)

0 1 2 3

2.1 Explain the idea of data augmentation (1p). Specify 4 different data augmentation techniques you can apply on a dataset of RGB images (2p).

Improve generalization (prevent overfitting) by applying various transformations or modifications to the existing dataset. (1p)

~~Improve generalization by adding more data and preventing overfitting (1p)~~

Rotation, cropping, color jittering, salt/paper, flipping, translation jitter ... (0.5p for each)

0.5 -> make training set larger
1.0 -> generalization/prevent overfitting

0 1 2

2.2 You are training a deep neural network for the task of binary classification using the Binary Cross Entropy loss. What is the expected loss value for the first mini-batch with batch size $N = 64$ for an untrained, randomly initialized network? Hint: $BCE = -\frac{1}{N}\sum(y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i))$

$-log(0.5)$ or $log(2)$

-0.5 -> for 1/64
-0.5 -> for minus

0 1 2

2.3 Explain the differences between *ReLU*, *LeakyReLU* and *Parametric ReLU*.

ReLU: constant 0 for negative values (0.5p) LeakyReLU: pre-defined slope for negative values (0.5p) Parametric ReLU: learnable value for slope, either 1 for all channels or 1 for each channels. (1p)

for relu and leaky relu -> full points to formula / drawing
for parametric relu -> learnable slope

0 1 2

2.4 How will weights be initialized by Xavier initialization? Which mean and variance will the weights have? Which mean and variance will the output data have?

With Xavier initialization we initialize the weights to be Gaussian with zero mean and variance $Var(w) = 1/n$ where n is the amount of neurons in the input.
As a result, the output will have zero mean, and similar variance as the input

weights
0.5 -> zero mean(with mentioning Gaussian)
0.5 -> variance
output
0.5->mean
0.5-> variance(same/similar)

Answers based on uniform distribution for weight initialization are also accepted.

2.5 Why do we often refer to L2-regularization as "weight decay"? Derive a mathematical expression that includes the weights W, the learning rate η, and the L2 regularization hyperparameter λ to explain your point.

☐ 0
☐ 1
☐ 2
☐ 3

$Reg = 0.5 \cdot \lambda \cdot ||W||^2$ The 0.5 factor is only added for convenience when differentiating. Omitting it is also ok

Upon a gradient update:

$$W_{new} = W - \eta \cdot \nabla Reg = W - \eta \cdot \lambda W = (1 - \eta \cdot \lambda)W$$

0.5/3.0 -> no formula, just correct explanation

Including the loss gradient from the GD update step also ok.

1.0 -> formula of regularization
1.0 -> for gradient, inserting reg
1.0 -> weight decay

2.6 Given a Convolution Layer in a network with 6 filters, kernel size 5, a stride of 3, and a padding of 2. For an input feature map of shape $28 \times 28 \times 28$, what are the dimensions/shape of the output tensor after applying the Convolution Layer to the input?

☐ 0
☐ 1
☐ 2

Output width/height = (28 + 2 * 2 - 5) / 3 + 1 = 10 (1 pt, it's ok if they do not have the calculation).
Output shape of channels x height x width = 6 x 10 x 10 or 10 x 10 x 6 (1 pt).

0.5 -> correct formula and wrong calculation
1.0 -> output shape

2.7 You are given a Convolutional Layer with: number of input channels 3, number of filters 5, kernel size 4, stride 2, padding 1. What is the total number of trainable parameters for this layer? Don't forget to consider the bias.

☐ 0
☐ 1
☐ 2

$(3 \times (4 \times 4)) \times 5$ for weights + 5 for bias = 240 + 5 = 245
(1pt for weights without correct bias)

1.0 for each correct calculation
2.0 for correct number(245)
1.5 -> wrong addition

2.8 You are given a fully-connected network with 2 hidden layers, the first ~~of~~ has 10 neurons, and the second hidden layer contains 5 neurons. Both layers use dropout with probability 0.5. The network classifies gray-scale images of size 8×8 pixels as one of 3 different classes. All neurons include a bias. Calculate the total number of trainable parameters in this network.

☐ 0
☐ 1
☐ 2

Weights: $(8 \times 8) \times 10 + 10 \times 5 + 5 \times 3 = 705$
Biases: $10 + 5 + 3 = 18$
Total: $705 + 18 = 723$

1 p -> weights
1 p -> bias
1.5 -> wrong addition

0 ☐ 1 ☐ 2 ☐

2.9 *"Breaking the symmetry"*: Why is initializing all weights of a fully-connected layer to the same value problematic?

All neurons will learn the same thing / Gradient update will be the same for all neurons / they won't take on different values.

2p-> mentioning they all compute the same function/learn the same thing/ same gradient update

0 ☐ 1 ☐ 2 ☐

2.10 Explain the difference between *Auto-Encoders* and *Variational Auto-Encoders*.

A variational Autoencoder emposes (optional: Gaussian / KL-Divergence loss) constraints on the distribution of the bottleneck

no points for explanation of autoencoder
2p -> constraint on latent space/distribution

0 ☐ 1 ☐ 2 ☐

2.11 Generative Adversarial Networks (GANs): What is the input to the generator network (1 pt)? What are the two inputs to the discriminator (1 pt)?

Generator: The input is a random noise vector (1 pt). [Wrong: labels] Discriminator: The inputs to the Discriminator are fake/generated images (0.5 pt) and real images (0.5 pt).

random input is fine, not mentioning fake is fine

0 ☐ 1 ☐ 2 ☐

2.12 Explain how LSTM networks often outperform traditional RNNs. What in their architecture enables this?

It is difficult for traditional RNNs to learn long-term dependencies due to vanishing gradients (1p).
The cell state (1p) in LSTMs improve the gradient flow and thereby allows the network to learn longer dependencies.

0.5 -> vanishing gradient
0.5 p -> long term dep
0.5 p -> highway for gradient/ improved gradient flow
1.0-> cell state

0 ☐ 1 ☐ 2 ☐ 3 ☐

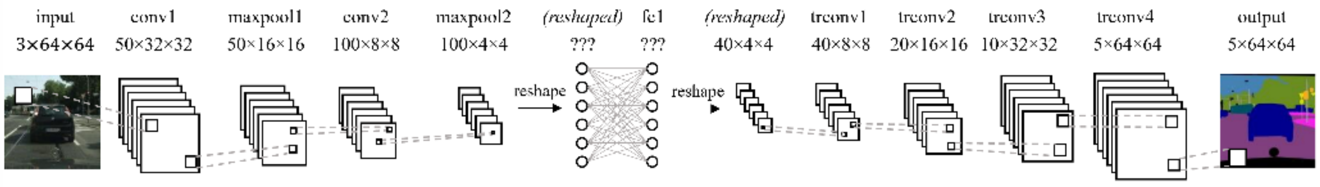
2.13 Explain how batch normalization is applied differently between a fully connected layer and a convolutional layer (1 pt). How many learnable parameters does batch normalization contain following (a) a single fully-connected layer (1 pt), and (b) a single convolutional layer with 16 filters (1 pt)?

Mini-batch is normalized over all neurons in a fully connected layer, while it is normalized over each channel in a convolutional layer (1 pt).
2 for a fully-connected layer (1 pt)
$2 \times 16 = 32$ for a convolutional layer (1 pt).

for the first point -> 0.5 p for the first part 0.5 for the second

Problem 3 Convolutions (13 credits)

You are asked to perform **per-pixel** semantic segmentation on the Cityscapes dataset, which consists of RGB images of European city streets, and you want to segment the images into 5 classes (vehicle, road, sky, nature, other). You have designed the following network, as seen in the illustration below:
For clarification of notation: The shape **after** having applied the operation 'conv1' (the first convolutional layer in the network) is 50x32x32.
You are using 2D convolutions with: `stride` = 2 , `padding` = 1 , and `kernel_size` = 4 .
For the MaxPool operation, you are using: `stride` = 2 , `padding` = 0 , and `kernel_size` = 2 .



3.1 What is the shape of the weight matrix of the fully-connected layer 'fc1'? (Ignore the bias)

0
1
2

input: $100 \times 4 \times 4 = 1600$
output: $40 \times 4 \times 4 = 640$
weight matrix: 1600×640

3.2 Explain the term 'receptive field' (1p). What is the receptive field of one pixel of the activation map. after performing the operation 'maxpool1'(1p)? What is the receptive field of a single neuron in the output of layer 'fc1' (1p)?

0
1
2
3

the size of the region in the input space that a pixel in the output space is affected by.
maxpool1: 6x6. One pixel after maxpool1 is affected by 4 pixels (2x2) in conv1. with 4x4 kernel and stride 2, a 2x2 output comes from a 6x6 grid.
f1: whole image (64x64) (accept answer that takes into account padding)

0 1

3.3 You now want to be able to classify finer-grained labels, which comprise of 30 classes. What is the **minimal** change in network architecture needed in order to support this without adding any additional layers?

1in - change output channels of trconv4 to 30 - NO: add 1x1 conv (with 30 output channels) - NO: or any conv that preserves the size (with 30 output channels)

0 1 2

3.4 Luckily, you found a pre-trained version of this network, which is trained on the original 5 labels. (It outputs a tensor of shape $5 \times 64 \times 64$). How can you make use of/build upon this pre-trained network (as a black-box) to perform segmentation into 30 classes.

- add 1x1 conv at the end (with 30 output channels) - or any conv that preserves the size (with 30 output channels)

0 1 2 3

3.5 Luckily, you have gained access to a large dataset of city street images. Unfortunately, these images are not labelled, and you do not have the resources to annotate them. However, how can you still make use of these images to improve your network? Explain the architecture of any networks that you will use and explain how training will be performed. (Note: This question is independent of (3.3) and (3.4))

transfer learning, pre-train an AutoEncoder with the unlabeled / all images, use encoder or entire network (except last layer/layers) to initialize the segmentation network. Freeze (some) weights, change/add last layer to output segmentation.

0 1 2

3.6 Instead of taking 64×64 images as input, you now want to be able to train the network to segment images of arbitrary size > 64. List, explicitly, two different approaches that would allow this. Your new network should support varying image sizes in run-time, without having to be re-trained.

- Resize layer/operation to downsample images to 64x64 (e.g bilinear)
- Make it fully convolutional by replacing the FC layer with convolutions

Wrong: explanation with RNNs (eigenvalues etc)

Problem 4 Optimization (13 credits)

4.1 Explain the idea behind the RMSProp optimizer. How does it enable faster convergence than standard SGD? How does it make use of the gradient?

0
1
2
3

RMSProp is an adaptive learning rate method - It scales the learning rate (1 pt) based on (an exponentially decaying average of) magnitude/element-wise squared gradient. This enables faster convergence by e.g skipping through saddle points with high learning rates. Also possible arguments from the lecture:

- Dampening the oscillations for high-variance directions
- Can use faster learning rate because it is less likely to diverge: Speed up learning speed, Second moment RMSProp does not have momentum!

4.2 What is the *bias correction* in the ADAM optimizer? Explain the problem that it fixes.

0
1
2

When accumulating gradients in a weighted average fashion, the first gradient is initialized to zero. This biases all the accumulated gradients down towards zero. The Bias correction normalizes the magnitude of the accumulated gradient for early steps.

4.3 You read that when training deeper networks, you may suffer from the *vanishing gradients* problem. Explain what are vanishing gradients in the context of deep convolutional networks and the underlying cause of the problem.

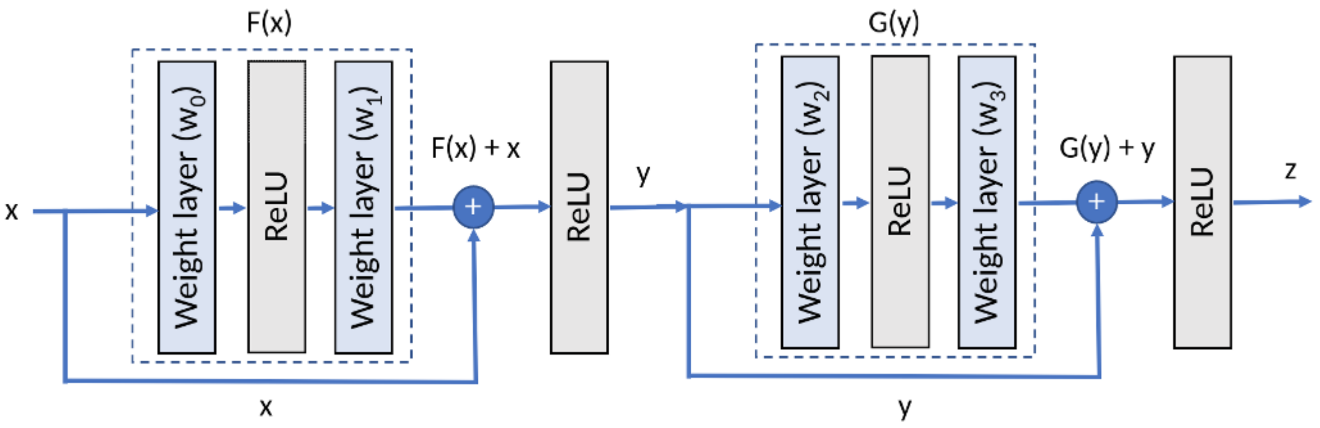
0
1
2
3

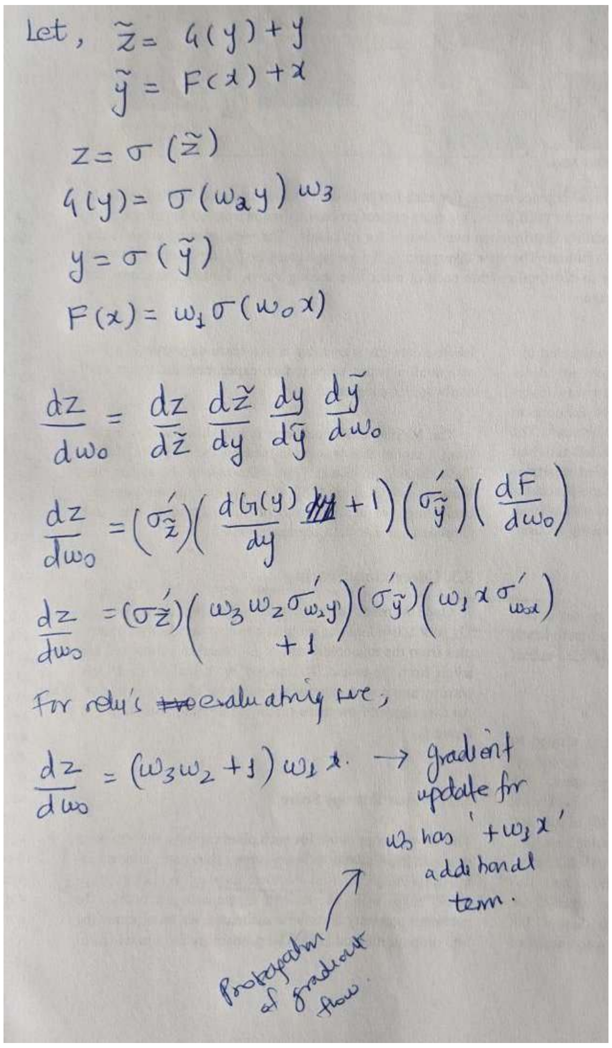
vanishing gradients are gradients with a very small magnitude (causing a meaningless update step) (1 pt) Caused by:

- saturated activations (e.g., tanh)
- chain rule - many multiplications on numbers in $(0, 1)$ goes to 0. during backpropagation, the gradient of early layers (layers near to the input layer) are obtained by multiplying the gradients of later layers

0 1 2 3 4 5

4.4 In the following image you can see a segment of a very deep architecture that uses residual connections. How are residual connections helpful against vanishing gradients? Demonstrate this mathematically by performing a weight update for w_0. Make sure to explain how this reduces the effect of vanishing gradients. Hint: Write the mathematical expression for $\frac{\partial z}{\partial W_0}$ w.r.t all other weights.





Let, $\tilde{z} = G(y) + y$

$\tilde{y} = F(x) + x$

$z = \sigma(\tilde{z})$

$G(y) = \sigma(w_2 y)\, w_3$

$y = \sigma(\tilde{y})$

$F(x) = w_1 \sigma(w_0 x)$

$$\frac{dz}{dw_0} = \frac{dz}{d\tilde{z}} \frac{d\tilde{z}}{dy} \frac{dy}{d\tilde{y}} \frac{d\tilde{y}}{dw_0}$$

$$\frac{dz}{dw_0} = (\sigma'_{\tilde{z}})\left(\frac{dG(y)}{dy} + 1\right)(\sigma'_{\tilde{y}})\left(\frac{dF}{dw_0}\right)$$

$$\frac{dz}{dw_0} = (\sigma'_{\tilde{z}})\left(w_3 w_2 \sigma'_{w_2 y} + 1\right)(\sigma'_{\tilde{y}})\left(w_1 x \sigma'_{w_0 x}\right)$$

For relu's ~~[illegible]~~ evaluating we;

$$\frac{dz}{dw_0} = (w_3 w_2 + 1)\, w_1 x.$$

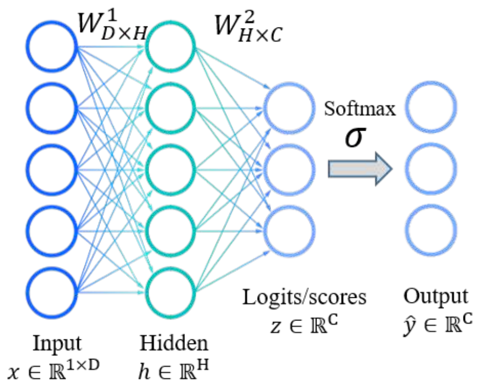
$\rightarrow$ gradient update for w_0 has '$+w_1 x$' additional term.

Propagation of gradient flow

Problem 5 Multi-Class Classification (18 credits)

Note: If you cannot solve a sub-question and need its answer for a calculation in following sub-questions, mark it as such and use a symbolic placeholder (i.e., the mathematical expression you could not explicitly calculate + a note that it is missing from the previous question.)

Assume you are given a labeled dataset $\{X, y\}$, where each sample x_i belongs to one of $C = 10$ classes. We denote its corresponding label $y_i \in \{1, ..., 10\}$. In addition, you can assume each data sample is a row vector.
You are asked to train a classifier for this classification task, namely, a 2-layer fully-connected network. For a visualization of the setting, refer to the following illustration:



5.1 Why does one use a **Softmax** activation at the end of such a classification network? What property does it have that makes it a common choice for a classification task?

0 1 2

It normalizes the logits/scores to sum up to 1 / a probability distribution
Wrong: its derivative can be expressed in terms of the softmax function itself. This is not special for classification, say only output between [0,1] (0 pt)

5.2 For a vector of logits $\vec{z}$, the Softmax function $\sigma : \mathbb{R}^C \to \mathbb{R}^C$, is defined:

0 1

$$\hat{y}_i = \sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$

where C is the number of classes and z_i is the i-th logit.
A special property of this function is that its derivative can be expressed in terms of the Softmax function itself. How could this be advantageous for training neural networks?

calculation of the backward pass is quick, immediate from saving the forward cache

$$= \frac{\sum_{j=1}^{C} e^{z_j} \left(\frac{\partial}{\partial z_i} e^{z_i}\right) - e^{z_i} \cdot \frac{\partial}{\partial z_i}\left(\sum_{j=1}^{C} e^{z_j}\right)}{\left(\sum_{j=1}^{C} e^{z_j}\right)^2}$$

0 ☐ 1 ☐ 2 ☐ 3 ☐

5.3 Show explicitly how this can be done, by writing $\frac{\partial \hat{y}_i}{\partial z_i}$ in terms of $\hat{y}_i$.

$$\frac{\partial}{\partial(z_i)}(\hat{y}_i) = \frac{e^{z_i} \cdot \sum_j e^{z_j} - e^{z_i} \cdot e^{z_i}}{\left(\sum_j e^{z_j}\right)^2} =$$

$$= \frac{e^{z_i} \cdot \left[\left(\sum_j e^{z_j}\right) - e^{z_i}\right]}{\left(\sum_j e^{z_j}\right) \cdot \left(\sum_j e^{z_j}\right)} = \frac{e^{z_i}}{\sum_j e^{z_j}} \cdot \frac{\sum_j e^{z_j} - e^{z_i}}{\sum_j e^{z_j}} =$$

$$= \hat{y}_i \cdot \left(\frac{\sum_j e^{z_j}}{\sum_j e^{z_j}} - \frac{e_i^{z}}{\sum_j e^{z_j}}\right) = \hat{y}_i \cdot (1 - \hat{y}_i)$$

0 ☐ 1 ☐ 2 ☐

5.4 Similarly, show explicitly how this can be done, by writing $\frac{\partial \hat{y}_i}{\partial z_j}$ in terms of $\hat{y}_i$ and $\hat{y}_j$, for $i \neq j$.

$$\frac{\partial}{\partial(z_j)}(\hat{y}_i) = \frac{0 \cdot \sum_j e^{z_j} - e^{z_j} \cdot e^{z_i}}{\left(\sum_j e^{z_j}\right)^2} =$$

$$= \frac{-e^{z_j} \cdot e^{z_i}}{\left(\sum_j e^{z_j}\right) \cdot \left(\sum_j e^{z_j}\right)} = -\frac{e^{z_i}}{\sum_j e^{z_j}} \cdot \frac{e^{z_j}}{\sum_j e^{z_j}} = -\hat{y}_i \hat{y}_j$$

5.5 Using the Softmax activation, what loss function $\mathcal{L}(y, \hat{y})$ would you want to *minimize*, to train a network on such a multi-class classification task? Name this loss function (1 pt), and write down its formula (2 pt), for a single sample x, in terms of the network's prediction $\hat{y}$ and its true label y. Here, you can assume the label $y \in \{0,1\}^C$ is a one-hot encoded vector:

$$y_i = \begin{cases} 1, & \text{if } i == \text{true class index} \\ 0, & \text{otherwise} \end{cases}$$

☐ 0 ☐ 1 ☐ 2 ☐ 3

(not Binary! (0 pt for binary) Cross Entropy loss / softmax loss (colloquial term) .

$$CE(y, \hat{y}) = -\sum_{j=1}^{C} y_j \log \hat{y}_j$$

or, since labels are one-hot vectors here:

$$CE(y, \hat{y}) = -\log \hat{y}_j$$

Comments:

- forget minus - lose 0.5 pt
- normalize by $1/C$ - OK
- formula has another sum over all data - OK

5.6 Having done a forward pass with our sample x, we will back-propagate through the network. We want to perform a gradient update for the weight $w^2_{j,k}$ (the weight which is in row j, column k of the second weights' matrix W^2). First, use the chain rule to write down the derivative $\frac{\partial \mathcal{L}}{\partial w_{j,k}}$ as a product of 3 partial derivatives (no need to compute them). For convenience, you can ignore the bias and omit the ^{2} superscript.

☐ 0 ☐ 1 ☐ 2

First, we write the Chain rule:

$$\frac{\partial \mathcal{L}}{\partial w_{j,k}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial w_{j,k}}$$

(the weight which is in row 3 , column 1 of the second weights' matrix W2).

0 1 2 3 4 5

5.7 Now, compute the gradient for the weight: $w^2_{3,1}$. For this, you will need to compute each of the partial derivatives you have written above, and perform the multiplication to get the final answer. You can assume the ground-truth label for the sample was `true_class = 3` . **Hint:** The derivative of the logarithm is $(\log t)' = \frac{1}{t}$.

For CE loss, the loss only depends on the prediction of $\hat{y}_{\text{true}}$, that is $\hat{y}_3$ in this case.

$$\frac{\partial \mathcal{L}}{\partial \hat{y}_3} = \frac{\partial(-\log \hat{y}_3)}{\partial \hat{y}_3} = -\frac{1}{\hat{y}_3}$$

$\hat{y}_3$ is affected by all of the entries of the vector z, because of the softmax. Note that $w_{3,1}$ only affects z_1 ($z = h \cdot W$), and from previous subquestions,

$$\frac{\partial \hat{y}_3}{\partial z_1} = -\hat{y}_3\hat{y}_1 \quad \leftarrow \text{Already derived in 5.4}$$

We are only missing $\frac{\partial z_1}{\partial w_{3,1}}$. That comes from matrix multiplication.

$$\frac{\partial z_1}{\partial w_{3,1}} = \frac{\partial \sum_{k=1}^{H} h_k w_{k,1}}{\partial w_{3,1}}$$

so $\frac{\partial z_1}{\partial w_{3,1}} = h_3$.

Finally, combining everything yields:

$$\frac{\partial \mathcal{L}}{\cancel{\partial w_{1,3}} \; \partial w_{3,1}} = -\frac{1}{\hat{y}_3} \cdot -\hat{y}_3\hat{y}_1 \cdot h_3 = \hat{y}_1 h_3$$

wrong sign (lose 0.5 pt)

Additional space for solutions–clearly mark the (sub)problem your answers are related to and strike out invalid solutions.