Closed book exam, no books, papers, notes, phones etc. allowed. The exam has 8 coding questions (labs), 8 open questions (lectures) and 14 multiple choice questions (papers). **Answer on the separate answer sheets.** Explain all answers, i.e. explicitly show intermediate steps to clarify if needed for coding / calculations / motivations / etc. Good luck!

We scan and crop answer boxes; please do not write outside answer boxes.

1 Lab assignments (38pts)

1. **Question** (4pts) Implement the forward pass of a shallow network with an input layer, one hidden layer, followed by a Sigmoid activation function. The forward pass equations are given below:

$$\mathbf{h} = \mathbf{x}\mathbf{W_1} + \mathbf{b_1}$$
$$\mathbf{y} = \mathbf{h}\mathbf{W_2} + \mathbf{b_2}$$
$$\mathbf{y} = \text{Sigmoid}(\mathbf{y})$$

Reminder:

$$\text{Sigmoid}(\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{x})}$$

Here $\mathbf{x}$ has dimensions [batch_size, input_channels] and $\mathbf{h}$ has dimensions [batch_size, hidden_dims] . Use the code template provided below. You may use elementary operations from the PyTorch library only, i.e. no predefined layers from torch.nn. Sub-questions:

A) Placeholder for layer weight and bias (2pts)

B) Forward pass (2pts)

```python
import torch
class ShallowNet(object):
    """
    Args:
        in_feat: number of input features
        hidden_dims: number of hidden neurons
        out_feat: number of output features
    """

    def __init__(self, in_feat, out_feat):
        super(Linear, self).__init__()
        ########################################
        # A) Your code goes here.
        ########################################

        self.init_params() # Init. parameters

    def init_params(self):
        self.weight_1 = torch.randn_like(
            self.weight_1)
        self.bias_1 = torch.rand_like(
            self.bias_1)
        self.weight_2 = torch.randn_like(
            self.weight_2)
        self.bias_2 = torch.rand_like(
            self.bias_2)

    def forward(self, x):
        ########################################
        # B) Your code goes here.
        ########################################
        return y
```

2. **Question** (4pts) Implement the backward passes for the LeakyReLU and Tanh non-linearities. The non-linearities are defined as follows:

$$\text{LeakyReLU}(\mathbf{x}) = \max(0.01\mathbf{x}, \mathbf{x})$$
$$\text{Tanh}(\mathbf{x}) = \frac{\exp(\mathbf{x}) - \exp(-\mathbf{x})}{\exp(\mathbf{x}) + \exp(-\mathbf{x})}$$

You may use elementary operations from PyTorch library only, i.e. no predefined layers from torch.nn. Sub-questions:

A) LeakyReLU backward (2pts)

B) Tanh backward (2pts)

```python
import torch
class LeakyReLU(object):

    def __init__(self):
        super(LeakyReLU, self).__init__()
        self.cache = None

    def forward(self, x):
        y = torch.clamp(x, min=0.01*x)
        self.cache = y
        return y

    def backward(self, dupstream):
        dupstream = dupstream.clone()
        ########################################
        # A) Your code goes here.
        ########################################
        return dx

class Tanh(object):

    def __init__(self):
        super(Tanh, self).__init__()
        self.cache = None

    def forward(self, x):
        y = (torch.exp(x) - torch.exp(-x))
            /(torch.exp(x) + torch.exp(-x))
        self.cache = y
        return y

    def backward(self, dupstream):
        ########################################
        # B) Your code goes here
        ########################################
        return dx
```

3. **Question** (6pts) Implement the following pooling methods for 2D inputs: You may use elementary operations from PyTorch library only, i.e. no predefined layers from torch.nn. Sub-questions:

A) Max Pooling (3pts)

B) Average Pooling (3pts)

```
import torch
class MaxPool2d(object):

    def __init__(self, kernel_size, stride=1, padding=0):
        self.kernel_size = kernel_size
        self.stride = stride
        self.padding = padding

    def forward(self, x):
        """
        Args:
            x: input tensor with shape of (N, C, H, W)
        Returns:
            y: output tensor with shape of (N, C, H', W') where
                H' = 1 + (H + 2 * padding - kernel_size) / stride
                W' = 1 + (W + 2 * padding - kernel_size) / stride
        """
        # Pad the input
        x_padded = torch.nn.functional.pad(x, [self.padding] * 4)
        # Unpack the needed dimensions
        N, C, H, W = x.shape
        KS = self.kernel_size
        # Calculate output height and width
        Hp = 1 + (H + 2 * self.padding - KS) // self.stride
        Wp = 1 + (W + 2 * self.padding - KS) // self.stride
        # Create an empty output to fill in.
        # We combine first and second dim to speed up as we need no loop for each
        # channel.
        y = torch.empty((N*C, Hp, Wp), dtype=x.dtype, device=x.device)

        ######################################
        # A)    Your Code Here               #
        ######################################


        # Reshape output to seperate sample dim from channel dim since we
        # combined them
        y = y.reshape(N, C, Hp, Wp)

        # Cache padded input to use in backward pass
        self.cache = x

        return y

class AveragePool2d(object):

    def __init__(self, kernel_size, stride=1, padding=0):
        self.kernel_size = kernel_size
        self.stride = stride
        self.padding = padding

    def forward(self, x):
        """
        Args:
            x: input tensor with shape of (N, C, H, W)

        Returns:
            y: output tensor with shape of (N, C, H', W') where
                H' = 1 + (H + 2 * padding - kernel_size) / stride
                W' = 1 + (W + 2 * padding - kernel_size) / stride
        """

        # Pad the input
        x_padded = torch.nn.functional.pad(x, [self.padding] * 4)

        # Unpack the needed dimensions
        N, C, H, W = x.shape
        KS = self.kernel_size

        # Calculate output height and width
        Hp = 1 + (H + 2 * self.padding - KS) // self.stride
        Wp = 1 + (W + 2 * self.padding - KS) // self.stride

        # Create an empty output to fill in.
        # We combine first and second dim to speed up as we need no loop for each
        # channel.
        y = torch.empty((N*C, Hp, Wp), dtype=x.dtype, device=x.device)

        ######################################
        # B)    Your Code Here               #
        ######################################

        # Reshape output to seperate sample dim from channel dim since we
        # combined them
        y = y.reshape(N, C, Hp, Wp)

        # Cache padded input to use in backward pass
        self.cache = x

        return y
```

4. **Question** (4pts) Implement the gradient update of the Adam optimizer, given by:

rho prev.

$$v_i = \rho_1 v_{i-1} + (1 - \rho_1)\,\nabla_\theta \quad \text{grad}$$

$$\hat{v_i} = \frac{v_i}{1 - \rho_1^i}$$

$$r_i = \rho_2 r_{i-1} + (1 - \rho_2)\,\nabla_\theta^2$$

$$\hat{r_i} = \frac{r_i}{1 - \rho_2^i}$$

$$\theta' = \theta - \epsilon \frac{\hat{v_i}}{\sqrt{\hat{r_i} + \delta}} \quad \text{delta}$$

```
def adam(X, rhos, learning_rate, prev_values, index, Grad=Grad_f):
    """
    Adam optimization step.
    Args:
        X: Current value of objective function.
        rhos: Optimization hyperparameter - see formula above.
        learning_rate: Optimization step size.
        prev_value: Momentum parameter from previous iteration.
        index: Optimization step counter.
        Grad: Gradient of quadratic function.
    """

```

```
    delta = 1e-5                    # Tiny amount to prevent division by zero
    gradient = Grad(*X)             # Gradient of current values
    rho_v, rho_r = rhos             # Rho values for momentum & rmsProp part of Adam
    v_prev, r_prev = prev_values # Adam parameters from previous iterations

    v = r = 0                       # Adam paramters for momentum & rmsProp
    v_bc = r_bc = 0                 # Bias corrected adam parameters

    #####################################
    #   TODO: Create gradient update    #
    #with Adam: update v, v_bc, r, r_bc #
    #and X.                             #
    #####################################
    #         Your Code Here            #
    #####################################
    return X,(v,r)
```

5. **Question** (4pts) Implement the following regularization methods:

   - L2 Regularization
   - Early Stopping

```
def train_wd(train_loader, net, optimizer, criterion, wd):
    """
    Args:
        train_loader: Data loader.
        net: Neural network model.
        optimizer: Optimizer (e.g. SGD).
        criterion: Loss function
        wd: Weight decay (L2 penalty)
    """

    avg_loss = 0
    correct = 0
    total = 0

    for i, data in enumerate(train_loader):
        inputs, labels = data
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        ######################################
        #       A) Your Code Here            #
        ######################################
        loss.backward()
        optimizer.step()
        avg_loss += loss
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

    return avg_loss/len(train_loader), 100 * correct / total






    #########EARLY STOPPING#########
net = FCNet()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=5e-1, weight_decay=3e-3)

# Set the number of epochs to for training
epochs = 100

# Patience - how many epochs to keep training after accuracy has not improved
patience = 0

# Initialize early stopping variables
val_acc_best = 0
patience_cnt = 0

for epoch in tqdm(range(epochs)): # loop over the dataset multiple times
    train_loss, train_acc = train(train_loader, net,optimizer,criterion)
    val_loss, val_acc = test(val_loader,net, criterion)

    writer.add_scalars("Loss", {'Train': train_loss, 'Val': val_loss}, epoch)
    writer.add_scalars('Accuracy', {'Train': train_acc, 'Val': val_acc}, epoch)

    #####################################
    #       B) Your Code Here           #
    #####################################
```

6. **Question** (6pts) Implement the GRU. [Hint]: think about all the architectural hyperparameters and especially sizes of the input, hidden states, weights and outputs.

   The update rule is given as:

$$r_t = \sigma(W_{xr}x_t + b_{xr} + W_{hr}h(t-1) + b_{hr})$$
$$z_t = \sigma(W_{xz}x_t + b_{xz} + W_{hz}h(t-1) + b_{hz})$$
$$n_t = \tanh(W_{xn}x_t + b_{xn} + r_t \odot (W_{hn}h(t-1) + b_{hn}))$$
$$h_t = (1-z_t) \odot n_t + z_t \odot h(t-1)$$

   Sub-questions:

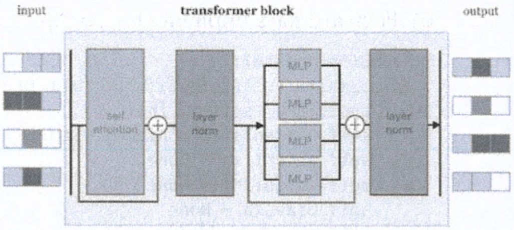
   A) Parameter initialization (2pts)

   B) Forward pass implementation (4pts)

```
class GRU(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(GRU, self).__init__()
        self.hidden_size = hidden_size
        self.weight_xh = None
        self.weight_hh = None
        self.bias_xh = None
        self.bias_hh = None

        ####################################
        # YOUR CODE HERE
        ####################################

        # Initialize parameters
        self.reset_params()

    def reset_params(self):
        std = 1.0 / math.sqrt(self.hidden_size)
        self.weight_xh.data.uniform_(-std, std)
        self.weight_hh.data.uniform_(-std, std)
        self.bias_xh.data.uniform_(-std, std)
        self.bias_hh.data.uniform_(-std, std)
```

```
    def forward(self, x):
        """
        Args:
            x: input with shape (N, T, D)
            where N is number of samples,
            T is number of timestep and
            D is input size which must be equal
            to self.input_size.

        Returns:
            y: output with a shape of (N, T, H)
            where H is hidden size
        """

        # Transpose input for efficient vectorized calculation. After transposing the input will have shape(T, N, D).
        x = x.transpose(0, 1)
        T, N, H = x.shape[0], x.shape[1], self.hidden_size
        h0 = torch.zeros(N, H, device=x.device)

        # Define a list to store outputs. We will then stack them.
        y = []

        ########################################
        # TODO: Implement GRU forward pass     #
        ########################################

        # Stack the outputs. After this operation, output will have shape of
        # (T, N, H)
        y = torch.stack(y)

        # Switch time and batch dimension, (T, N, H) -> (N, T, H)
        y = y.transpose(0, 1)
        return y
```

7. **Question** (4pts) The transformer block consists of a self-attention layer, followed by layer norm, a MLP applied on each vector individually and another layer norm. Note the residual connections in the self-attention and MLP layer. The architecture is given below:



Now implement the transformer block as a PyTorch layer. All different components are already defined in init function - your task is to connect them in the proper way in the forward method. Ima Sub-questions:

A) Perform the forward pass of a transformer block as depicted in the image.

```
import torch.nn as nn
class TransformerBlock(nn.Module):
    def __init__(self, k, heads):
        """
        Args:
            k: embedding dimension
            heads: number of heads (k mod heads must be 0)
        """
        super(TransformerBlock, self).__init__()

        self.att = MultiHeadAttention(k, heads=heads)
        self.norm1 = nn.LayerNorm(k)
        self.ff = nn.Sequential(
            nn.Linear(k, 4 * k),
            nn.ReLU(),
            nn.Linear(4 * k, k))
        self.norm2 = nn.LayerNorm(k)

    def forward(self, x):
        """
        Args:
            x: input with shape of (b, k)
        Returns:
            y: output with shape of (b, k)
        """
        ##############################
        # TODO: Perform the forward  #
        # pass of a transformer block#
        # as depicted in the image.  #
        ##############################
        return y
```

8. **Question** (6pts) Implement a Variational Autoencoder (VAE) for the following code block.

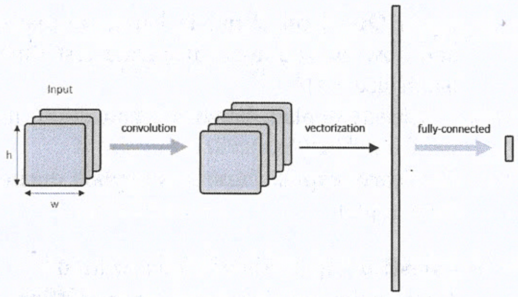
- KL Loss (2pt)
- Reparametrization (2pt)
- Forward function: creating mean and variance (2pt)

```
#encoder
class VarEncoder(nn.Module):
    def __init__(self, latent_dims, s_img, hdim):
        super(VarEncoder, self).__init__()

        #layers for g1
        self.linear1_1 = nn.Linear(s_img*s_img, hdim[0])
        self.linear2_1 = nn.Linear(hdim[0], hdim[1])
        self.linear3_1 = nn.Linear(hdim[1], latent_dims)

        #layers for g2
        self.linear1_2 = nn.Linear(s_img*s_img, hdim[0])
        self.linear2_2 = nn.Linear(hdim[0], hdim[1])
        self.linear3_2 = nn.Linear(hdim[1], latent_dims)

        self.relu    = nn.ReLU()

        #distribution setup
        self.N = torch.distributions.Normal(0, 1)
        self.N.loc = self.N.loc.to(try_gpu()) # hack to get sampling on the GPU
        self.N.scale = self.N.scale.to(try_gpu())
        self.kl = 0

    ######################################
```

```
    #    TODO: Define function for:     #
    #  A) the Kullback-Leibner          #
    #      loss "kull_leib"             #
    #  B) the Reparameterization trick  #
    #####################################
    #         Your code here            #
    #####################################

    def forward(self, x):

        #####################################
        # TODO: Create mean and variance    #
        #####################################
        # C) Forward Function your code here
        #####################################

        #reparameterize to find z
        z = self.reparameterize(mu, sig)

        #loss between N(0,I) and learned
    distribution
        self.kl = self.kull_leib(mu, sig)

        return z

#decoder: same as before

#autoencoder
class VarAutoencoder(nn.Module):
    def __init__(self, latent_dims, s_img, hdim
    = [100, 50]):
        super(VarAutoencoder, self).__init__()

        self.encoder = VarEncoder(latent_dims,
    s_img, hdim)
        self.decoder = Decoder(latent_dims,
    s_img, hdim)

    def forward(self, x):

        z = self.encoder(x)
        y = self.decoder(z)

        return y
```

2 Lectures (38pt)

1. **Question** (4pts) Consider a 2-layered network. The first layer is convolutional with a kernel size of 5×5, with padding = 1, stride = 2. The second layer is a fully connected layer. The number of input channels is 3, the number of hidden channels is 7. The number of output features is 2. The network uses bias terms. The size of an input image is: $h \times w = 12 \times 12$. The image below clarifies the network further. **What is the total number of learnable parameters?** Motivate your answer with a detailed step-by-step approach. (Don't forget the bias terms).



2. **Question** (6pts) Calculate MLE (Maximum Likelihood Estimator) for a Possion distribution.

   Consider that we are given a Poisson probability distribution given by:

$$P_{model}(X = x) = \frac{\lambda^x e^{-\lambda}}{x!}$$

   We draw m samples from the probability distribution namely : $\{x_1, x_2 \dots x_m\}$. The objective is to find the maximum likelihood estimate for the parameter λ as a function of the data samples.

   Sub-questions:
   1) (1pt) Write the likelihood function for the given probability distribution.
   2) (2pt) Write the log-likelihood function by using the logarithm operator on the function obtained in the previous step. Make sure to simplify the terms as much as possible.
   3) (2pt) Calculate the derivative of the natural log likelihood function with respect to λ
   4) (1pt) Set the derivative equal to zero and solve for λ

   Hint: use the log-likelihood, defined as

$$MLE(X) = \arg\max_{\theta} \prod_{i=1}^{m} \log P_{model}(x_i, \theta)$$

   where $X = \{x_1 \dots x_m\}$

3. **Question** (6pts)

   Calculate the receptive field of a feature/pixel in the output of the architecture given by the table below. Provide a step-by-step explanation.

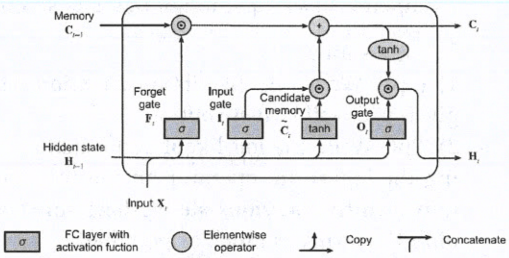
   Note: the receptive field here refers to the number of pixels in the input image that a particular feature ("pixel") in the output of Conv4 is looking at, i.e. the answer should be a single integer.

Layer	**Kernel size**	**Stride**
Conv1	3	1
Pool1	2	2
Conv2	3	1
Pool2	2	2
Conv3	3	1
Conv4	3	1

4. (4pts) **Question** What is batch normalization and how do you apply it during test time (e.g. batch size = 1)?
   1. Clearly explain batch normalization, including formulas. (2pts)
   2. Clearly explain how it is applied during test time. (2pts)

5. **Question** (4pts) About regularization.
   1. Draw the learning curve for overfitting. (1pt)
   2. Name and explain one type of parameter norm regularization (include its equation). (1pt)
   3. How is dropout performed? Explain both training and evaluation. (2pt)

6. **Question** (6pts) Long Short Term Memory (LSTM).
   Using equations, explain the working of the LSTM, i.e. current hidden state based on the current memory cell state (C_t), previous hidden state (h_{t-1}) and the current input (X_t).



   1. Equation of the forget gate (f_t). (1pt)
   2. Equation of the input gate (i_t). (1pt)
   3. Equation of the candidate cell gate (c_t). (2pt)
   4. Equation of current hidden state (h_t). (2pt)

7. **Question**(4pts) There are two types of positional information used in self-attention: embeddings or encodings.
   1. Why do we need to explicitly include positional information in self-attention? (2pt)
   2. What is the difference between positional embeddings and positional encodings? (2pt)

8. **Question** (4pts) Contractive auto-encoder.
   1. What is an auto-encoders and what is it used for? (2pt)

   A contractive auto-encoder is given by

$$l(g(f(x)), x) + \lambda \left\| \frac{\partial f(x)}{\partial x} \right\|_F^2$$

   2. Explain the equation (terms, operators), and its goal. (2pt)

3 Papers (14pts)

A 1pt multiple-choice question per paper. Select the single best fitting answer per question.

1. Paper: *A Step Toward Quantifying Independently Reproducible Machine Learning Research.* What statement matches the paper best:
   A) Releasing code is very important to reproduce ML research.
   B) Looking at the released code by the authors tends to improve reproduction quality.
   C) Paper reproduction rates have not significantly changed over the past 35 years
   D) More equations tends to improve reproduction quality.

2. Paper: *Troubling Trends in Machine Learning Scholarship.* The paper mentions "Failure to identify the sources of empirical gains". What exactly is meant by that?
   A) Gains come from using additional data instead of the proposed method
   B) Gains are not tested statistical for significance
   C) Gains come from hyper-parameter tuning instead of the proposed method
   D) Gains are tuned on the test set

3. Paper: *Do ImageNet Classifiers Generalize to ImageNet?* What is their main motivation?
   A) That repeated re-use of the test set leads to overfitting
   B) That the data collection procedure is not reproducible
   C) That the 1,000 ImageNet classes are too specialized
   D) That the concept of generalization is too narrowly defined

4. Paper: *Scaling down deep learning.* What do they scale down?
   A) The number of layers
   B) The dataset
   C) The weights in a layer
   D) The training epochs

5. Paper: *Highway and Residual Networks learn Unrolled Iterative Estimation.* Whats the difference between highway vs residual?
   A) Highway simplifies the residual
   B) Highway is a gated variant of Residual

C) Residual is a different name for Highway

D) Residual is specially adapted for images

6. Paper: *ResNet strikes back: An improved training procedure in timm.* What does the author hope to achieve?

   A) A better ResNet baseline

   B) Better understanding of the ResNet

   C) An improved ResNet architecture

   D) Much faster training times

7. Paper: *Deep Image Prior.* What statement fits best with the paper:

   A) Neural network optimizers are a form of prior knowledge

   B) ConvNets are useful for images, even without training

   C) Optimization is less important than the network architecture

   D) ConvNets can learn to denoise images

8. Paper: *Approximating CNNs with Bag-of-local-Features models works surprisingly well on ImageNet.* What statement best describes the paper?

   A) Its based on shallow networks

   B) The receptive field is artificially enlarged

   C) The class evidence used for explainability is locally bagged

   D) The receptive field is artificially reduced

9. Paper: *Group Normalization.* The statistics are computed over groups of pixels. The grouping is done by:

   A) Grouping inside a single featuremap

   B) Grouping activations

   C) Grouping parameters

   D) Grouping residuals

10. Paper: *Torch.manual_seed(3407) is all you need: On the influence of random seeds in deep learning architectures for computer vision.* What is the author surprised about?

    A) That with not very large variance its easy to find an outlier that performs much better or much worse than the average.

    B) That even though the variance is small, that the differences are still statistically significant.

    C) That the random seed plays such a big role in finding the best results.

    D) That the random seed influences so many things (initialization, batch elements, gradient steps, etc.) yet does account for so little differences.

11. Paper: *Attention Is All You Need.* Select the best answer.

    A) For each token, the query, key and value parameters are identical

    B) The connections between the losses are gated, similar to a GRU

    C) One token is best seen as a set.

    D) Without positional embeddings, it would exploit the absolute token position, similar to a MLP.

12. Paper: *Perceiver IO: A General Architecture for Structured Inputs & Outputs.*

    A) Uses standard self-attention, and it is outperformed by specialized solutions

    B) Uses a special form of self-attention, and it is outperformed by specialized solutions

    C) Uses standard self-attention and it outperforms specialized solutions

    D) Uses a special form of self-attention and it outperforms specialized solutions

13. Paper: *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks.* What does the "cycle" refer to?

    A) Cyclic featuremap padding

    B) Training data cycles

    C) Cycling between two annotations of an image

    D) Cycling between domains

14. Paper: *GANORCON: Are Generative Models Useful for Few-shot Segmentation?* So, are GANs useful for few-show segmentation?

    A) Yes, more useful than contrastive learning.

    B) Yes, but not as useful as contrastive learning

    C) No, but contrastive learning is

    D) No, and contrastive learning also is not