Examination questions for FYTN14 / EXTQ40 2019-01-16 (14.00-19.00)

Approximately 17.5 points are required for passing the exam.

1. No derivations or explanations are needed for the following 5 questions!

a) You have a categorical input variable that can take the following "values": 'winter', 'spring', 'summer' and 'fall'. Give a one-hot encoding of 'winter'. (1p)

b) A smooth version of the ReLU activation is given by $\log(1+e^x)$. True or false? The derivative of this activation function is the logistic activation function. (1p) (Note: log() is the natural logarithm.)

c) True of false? The dropout regularization technique works by randomly removing nodes (input and/or hidden) in the network for every pattern used during training. (1p)

d) How many distinct patterns can a Hopfield model with N nodes express? (1p)

e) You have an MLP with two hidden layers and three **linear** nodes per hidden layer. This MLP is used for binary classification problems. Can the decision boundary be non-linear? (1p)

Answers:

1. Typical answer: (0,0,0,1)
2. The question was a bit misleading, but the true or false question concerned whether or not the derivative of the smooth ReLU function was given by the logistic activation function. The answer was True! The derivative of $\log(x)$ is $1/x$. This gives $e^x/(1+e^x)$ as the derivative for the smooth ReLU. This is exactly the logistic.
3. True! Dropout is working on nodes and new nodes are randomly "removed" for each pattern.
4. Each nodes can take two values (-1,1). This gives 2^N possible patterns.
5. No! With linear activation functions the MLP reduces to a simple perceptron.

2. Figure 1 shows an MLP with one hidden layer and a single output. The activation function for the

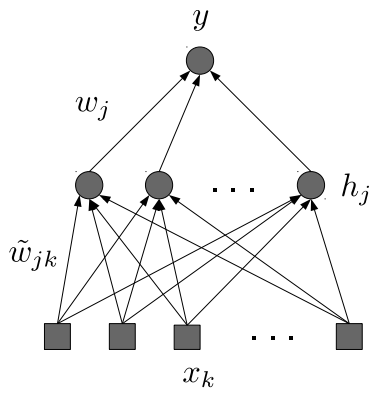

Figure 1:

hidden layer is φ_h and the activation function for the output layer is φ_o.

a) Write down the function this network implements, i.e. the output y as a function of inputs $\mathbf{x}$ and the weights. Let the bias to the output activation function be called θ, and the bias to hidden node j be called θ_j. (1p)

b) Given a dataset of inputs $\{\mathbf{x}\} = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \ldots)$ and targets $\{d\} = (d_1, d_2, d_3 \ldots)$, we want to train our network so that outputs $y(\mathbf{x}_n)$ are "as similar as possible" to targets d_n. One approach is to minimize an error function $E(\{y\}, \{d\})$. Give examples of two commonly used error functions and describe what kind of targets they are suitable for. (2p)

c) Using gradient descent as the approach to minimize E, derive the update $\Delta\theta_j$ for the biases to the hidden nodes. Assume all derivatives $\frac{\partial E}{\partial y_n}$ to be known. (2p)

Answers:

a) The mapping is

$$y = \varphi_o(\sum_j h_j \omega_j + \theta)$$

where

$$h_j = \varphi_h(\sum_k \tilde{\omega}_{jk} x_k + \theta_j)$$

b) Two common error functions are mean square error for regression

$$E = \frac{1}{N} \sum_n^N (y_n - d_n)^2$$

and the cross-entropy error for e.g. binary classification error.

$$E = -\frac{1}{N} \sum_n^N [d_n \log(y_n) + (1 - d_n) \log(1 - y_n)]$$

c)

$$
\begin{aligned}
\Delta\theta_j &= -\eta \frac{\partial E}{\partial \theta_j} \\
&= -\eta \sum_n \frac{\partial E_n}{\partial y_n} \frac{\partial y_n}{\partial \theta_j} \\
&= -\eta \sum_n \frac{\partial E_n}{\partial y_n} \frac{\partial y_n}{\partial h_{nj}} \frac{\partial h_{nj}}{\partial \theta_j} \\
&= -\eta \sum_n \frac{\partial E_n}{\partial y_n} \varphi_o'(\cdot) \omega_j \varphi_h'(\cdot)
\end{aligned}
$$

3. Suppose you have split a dataset into a training part that you use to train an MLP, and a validation part that you use to examine the MLP's performance on previously unseen data.

You train different MLP:s, all with one hidden layer but with different number of hidden nodes, and record the error (value of your error function). The results are shown in the table below:

Number of hidden nodes	2	4	8
Training E	3.4	3.1	2.3
Validation E	3.5	3.3	3.9

a) In the table we can compare training and validation error for each number of hidden nodes. We can also see how each error evolves as the number of hidden nodes increases. Discuss the observed relations. Are they reasonable? Why? In particular, introduce the concept of overtraining in your discussion. (2p)

b) Assume we are provided more data, so that we can train our networks with a larger training set. We keep the validation set unchanged. We train new MLP:s on the larger training set, trying the same number of hidden nodes as above. How do you expect the training and validation errors to change? The error function is of the form $E = \frac{1}{N}\sum_n E_n$, where N is the number of samples and E_n a function depending only on sample n. (1p)

c) Restricting the number of hidden nodes is a way to *regularize* the network, in order to avoid overtraining. Give two other examples of regularization and describe how they work. (2p)

Answers:

a) It reasonable that the training decreases when the number of hidden nodes increases, since the capacity of the network increases. The behavior of the validation error is typical of overfitting, meaning that the validation starts to increase when the training error decreases.

b) The validation error should decrease since we have have a better model (trained on more data) whit possible less overtraining. The training error should increase, since with more data it is more difficult to reach the same error as before with no change in the capacity.

c) Examples of other techniques:

- L2 or L1 regularization, meaning adding terms like $\alpha\sum_i \omega_i^2$ or $\alpha\sum_i |\omega_i|$ to the error function, respectively.
- Dropout. Here nodes in the network (both input and hidden or either of them) are randomly removed in the networks during training (random selection for each pattern presented to the network.
- Early stop. Stop training when validation error have reached a minimum.
- Ensemble of networks.

4. A single validation dataset may be too small to give a reliable estimate of how well a MLP performs on unseen data (the "generalization" performance).

a) Describe the bootstrap method to get a more reliable estimate of the generalization performance. (2p)

b) As in problem 3, the validation dataset can be used to perform *model selection*, i.e., determine *hyper-parameters* such as hidden nodes, learning rate in gradient descent, etc.. When optimal hyperparameters have been found, we can use them as we train a new network on the entire dataset (training+validation). Explain why the validation result from the model selection step is not a good estimator of the performance of this final MLP on new data! (1p)

c) Describe a way to get an unbiased estimate of the generalization performance of an MLP trained with optimized hyperparameters. (2p)

Answers:

a) Train on a bootstrap sample of the original dataset (i.e. random with replacement) and validation on the samples that were not selected for training. Repeat many times and take the average validation result.

b) The validation data is used to decide values of hyperparameters meaning that we have optimized the model also based on the validation data. It is not an independent sample anymore. The validation performance is therefore too optimistic.

c) In short we need an additional independent dataset for testing. This could be accomplished using three different datasets (training, validation and test), but it can also be done using two nested cross-validation loops.

5. Recurrent networks!

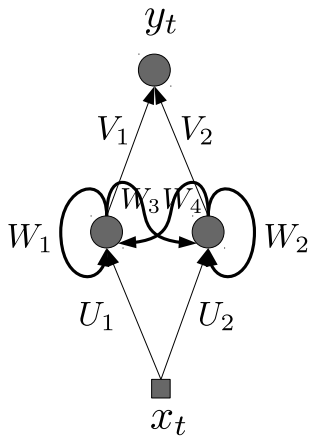

Figure 2:

a) We can deal with sequence data using an ordinary MLP without any feedback connections. We call this network a *time delay network*. Explain how the time delay network is trained for the task of predicting the next value of a simple one dimensional time series $x(t), t = 1, ..., T$. (2p)

b) Figure 2 shows a recurrent network with one input and one output node. It has two hidden nodes and four feed-back connections, marked W_1, W_2, W_3, W_4. Unfold this network one time step in time. Mark all weights in the unfolded network with the correct labels, i.e. $(U_1, U_2, W_1, W_2, W_3, W_4, V_1, V_2)$ (2p)

c) Recurrent networks, as the one shown in figure 2, can be used to model sequence data. When dealing with long sequences it can however be tricky to train the unfolded network because of the vanishing gradient problem. Why can gradients "vanish"? (1p)

Answers:

a) The time delay network is using a fixed history of the input sequence in order to make predictions on future ones. By using a sliding window we can create a training dataset of inputs and targets. For example if we use a history of $t-1, t-2, t-5$ and $t-10$ to predict $x(t)$ then we get the following input/target patterns: $\mathbf{x}_1 = (x(1), x(6), x(9), x(10)), d_1 = x(11)$, $\mathbf{x}_2 = (x(2), x(7), x(10), x(11)), d_2 = x(12)$, etc. Having this dataset you can train an ordinary MLP for this regression task.

b) See figure.

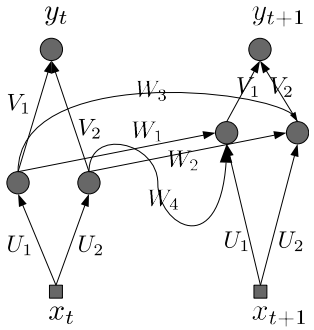

Figure 3:

c) The unfolded network becomes an MLP with many hidden layers. When computing the gradients in such a network, needed for the gradient descent minimization method, it results in multiplications of many terms of the type

$$g'(\cdot)\omega$$

where $g(\cdot)$ is the activation function and ω is the weight. If these terms are small the we can easily get vanishing gradients.

6. Convolutional neural networks (CNN)!

a) We can define hyper-parameters to be all "settings" (except the weights) that defines the network and how to train it. CNNs have many hyper-parameters. Assume you should construct a CNN for an image classification task. List **different** kinds of hyper-parameters when defining and training your CNN. (0.5 points per hyper-parameter, max 3p)

b) Assume you have an input image (4x4) as below.

$$\begin{bmatrix} 1 & 2 & -2 & 3 \\ -1 & 3 & 1 & -1 \\ 5 & 0 & -2 & 1 \\ -1 & 1 & -3 & 2 \end{bmatrix}$$

Your convolutional filter looks like this:

$$\begin{bmatrix} 1 & 3 \\ 3 & 0 \end{bmatrix}$$

Use stride = 2, no zero-padding and assume a ReLU activation function. Compute the resulting filtered image (after activation, but before a possible max-pooling). You can assume a zero bias weight! (1p)

c) You have the following setup for a binary image classification task using CNNs. Image size: 10x10. One convolutional layer: 1 kernel, size 3x3 (stride = 1, no zero-padding) + max-pooling 2x2 (stride = 2). Dense layer: 2 hidden nodes, Output layer: 1 single output node. How many weights (including bias weights) does this CNN have? (1p)

Answers:

a) Kernel size, number of kernels, stride, number of layers, pooling or not, regularization type, regularization parameters, "optimization methods parameters", ...

b)

$$\begin{aligned} 1 + 6 - 3 + 0 &= 4 \\ -2 + 9 + 9 + 0 &= 10 \\ 5 + 0 - 3 + 0 &= 2 \\ -2 + 3 - 9 + 0 &= -8 \end{aligned}$$

which results in the filtered image

$$\begin{bmatrix} 4 & 10 \\ 2 & -8 \end{bmatrix}$$

and after ReLU this becomes

$$\begin{bmatrix} 4 & 10 \\ 2 & 0 \end{bmatrix}$$

c) The 3x3 kernel gives us $3 * 3 + 1$ weights for the convolution. After maxpooling we end up with a 4x4 image. The flatten operation make this a vector of 16 inputs that feeds into 2 hidden nodes. This means $2 * (16 + 1)$ weights. The single output node then contributes with $1 * (2 + 1)$ weights. In total 47 weights.

7. Decision boundaries!

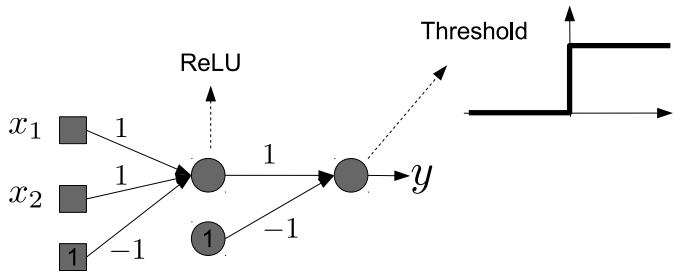

Figure 4:

Figure 4 shows a network with two inputs, a single ReLU hidden node and a single output node with a threshold activation function. This network is used for a binary classification. The values of the weights and bias-weights are shown in the figure.

Figure 5 shows two simple perceptrons with logistic output activation functions, also used for binary classification.

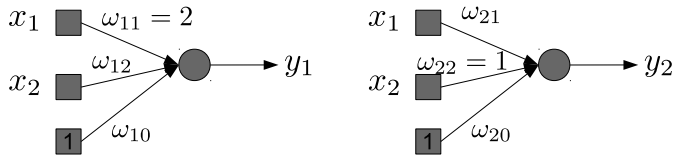

Figure 5:

They form an ensemble according to,

$$y_{\text{ens}} = \frac{1}{2}(y_1 + y_2),$$

and the cut value 0.5 is used to define the two classes for y_{ens}. Find one set of weights $(\omega_{12}, \omega_{10}, \omega_{21}, \omega_{20})$, such that the ensemble y_{ens} and the first network implements the same decision boundary. (5p) **Answer:** The mapping of the first network is,

$$y = \Theta(\text{ReLU}(x_1 + x_2 - 1) - 1)$$

which means that the boundary is,

$$\text{ReLU}(x_1 + x_2 - 1) - 1 = 0$$

Taking the ReLU into account this means the following decision line,

$$x_1 + x_2 - 2 = 0$$

Now let us look at the ensemble network. The ensemble is,

$$y_{\text{ens}} = \frac{1}{2}(y_1 + y_2)$$

Using the cut value of 0.5 to define the boundary we get the following boundary,

$$\frac{1}{2}(y_1 + y_2) = \frac{1}{2} \quad \Rightarrow$$
$$y_1 + y_2 = 1$$

The y_1 and y_2 network looks like,

$$y_1 = \frac{1}{1+e^{-(2x_1+\omega_{12}x_2+\omega_{10})}} \equiv \frac{1}{1+e^{-f_1}}$$
$$y_2 = \frac{1}{1+e^{-(\omega_{21}x_1+\omega_{+}x_2+\omega_{20})}} \equiv \frac{1}{1+e^{-f_2}}$$

The boundary equation then becomes,

$$\begin{aligned} \frac{1}{1+e^{-f_1}} + \frac{1}{1+e^{-f_2}} = 1 \quad &\Rightarrow \\ e^{-(f_1+f_2)} = 1 \quad &\Rightarrow \\ f_1 + f_2 = 0 \quad &\Rightarrow \\ x_1(\omega_{21}+2) + x_2(\omega_{12}+1) + (\omega_{10}+\omega_{20}) = 0 & \end{aligned}$$

Now compare this with the boundary for the first network. One solution is therefore, $\omega_{12} = 0, \omega_{21} = -1, \omega_{10} = -1, \omega_{20} = -1$. But we also realize that any multiplicative factor can be used for the first boundary-equation and it will still be the same boundary. E.g. the following solution also work, $\omega_{12} = 2, \omega_{21} = 1, \omega_{10} = -2, \omega_{20} = -4$.

Good Luck!

Mattias & Patrik