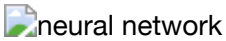
Assignment 7 - Artifical neural network (ANN)

The deadline for this assginment is 18.03 (Fri) 20:00

Being a subset of machine learning (ML) methods, **deep learning** follows the basic ML principle: find a hypothesis map out of a hypothesis space (represented by neural networks) that minimizes a chosen loss on datapoints.

Neural networks are called networks because they are typically represened by composing together many different functions, and the computed values create a network-like structure. For example, we might have three functions $f^{(1)}$, $f^{(2)}$ and $f^{(3)}$ connected in a chain, to form $f(x) = f^{(3)}(f^{(2)}(f^{(1)}))$. In this case, $f(1)$is called the first layer of the network, $f^{(2)}$ is called the second layer, and so on. The overall length of the chain gives the depth of the network. Networks with multiple layers are called deep networks, hence the name deep learning.

Typically the final layer is called the **output layer**, and represents the label that we want to predict. The training datapoints "tell" what the output layer must do at each datapoint - it should produce a value that is close to the desired label. The behaviour of other layers however, is not directly specified by the training data. Instead the learning algorithm decides how to use these layers to find the best approximation of the ideal map by minimizing (locally) the loss on the training dataset. Thus, these layers are called **hidden layers**.



In this assignment a fully connected multi-layer neural network, also called feed-forward neural network or **Multilayer Perceptron (MLP)**, is used to represent a hypothesis space that includes highly non-linear functions. MLP is the simplest type of a neural network, where each cell (neuron) is 'connected' to all the cells from the next layer, and only the next layer uses its value.

If you wish to gain understanding of how a neural network actually works and learns, this video series (https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi) by 3blue1brown provides a briliant visual explanation.

Learning goals

After successfully completing this assignment, you should be able to:

- understand the difference of hypothesis maps between MLP and linear models
- understand that activation functions are a key part of neural network design
- train MLPs to complete a regression task

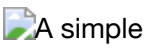
- train MLPs to complete a classification task
- have some basic understanding of gradient based learning of ANN weights
- use grid-search for adjusting multiple MLP hyper-parameters such as number of layers and learning rate parameters

In [7]:

```
%config Completer.use_jedi = False  # enable code auto-completion
import numpy as np #import numpy to work with arrays
import pandas as pd #import pandas to manipulate the dataset
from matplotlib import pyplot as plt #import the module matplotlib.pyplot to do
 visulization
from sklearn.preprocessing import PolynomialFeatures    # function to generate p
olynomial and interaction features
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, accuracy_score    # function to
 calculate mean squared error
```

Basic neural network structure

Take a look at the image below, showing a very basic neural network, with one-element input layer `x` and output layer `y`, and a single hidden layer with 3 hidden units:

 A simple neural network schema

The process of calculating the output of the network is as follows. We have the network defined as above, and a bias vector `[bias1, bias2, bias3]` corresponding to the `hidden1, hidden2, hidden3` hidden units. Then:

1. We multiply x with the first layer's weights, and add a constant bias term. We have obtained initial values of the hidden 'neuron' activations. A single value is calculated as:
$$hidden_i = x * weight_{1i} + bias_i$$
2. Simply multiplying and adding to the initial value of x would not let us represent any complex non-linear functions, so we need to introduce a non-linearity to our network. It is done via activation functions in the hidden layer, and the most common one is the rectified linear unit (ReLU):
$$ReLU(x) = max(x, 0)$$
It could be thought of as a function replacing negative values with 0s. We apply it to our hidden layer activations:
$$hidden_i = ReLU(hidden_i)$$
3. Now that we have the activation values, we multiply them with the final weights and sum the result together to obtain the final output:
$$y = hidden_1 * weight_{21} + hidden_2 * weight_{22} + hidden_3 * weight_{23}$$

Note that there is no bias in the last step. We also do not use ReLU anymore, as it is a simple regression network. However, for classification, you would use a different activation function for your output to turn it into a probability distribution - see sigmoid (https://en.wikipedia.org/wiki/Sigmoid_function) (binary) or softmax (https://en.wikipedia.org/wiki/Softmax_function) (multiclass)

Student Task A7.1

Use your understanding of how a neural network works, to set the correct weights and biases for our toy neural network example from above, so that the resulting function `f(x) = y` is a non-linear function looking like this:

 Neural Network hypothesis plot

Note that this would not be possible with linear models we studied previously, such as linear regression or SVMs.

The vertices of the triangle must be the points `(0, 0), (1, 1), (2, 0)`. Fill in the missing (`None`) values in the `weights_1`, `weights_2`, and `bias` vectors. There is no need to change the values that have already been set!

Hint: Try it out as a math problem with pen and paper if you have trouble! Try to think what are the already given functions $hidden_1(x)$ and $hidden_3(x)$, what unknowns there are in $hidden_2$, and what is the result of adding them together to obtain $y = f(x) = hidden_1(x) + hidden_2(x) + hidden_3(X)$ - e.g. with the given values:

$hidden_3(x) = 1 * x - 2$; for `x > 2` (i.e. `(1*x-2) > 0`)

$hidden_3(x) = 0$; otherwise

There are also visualizations of the intermediate plots below.

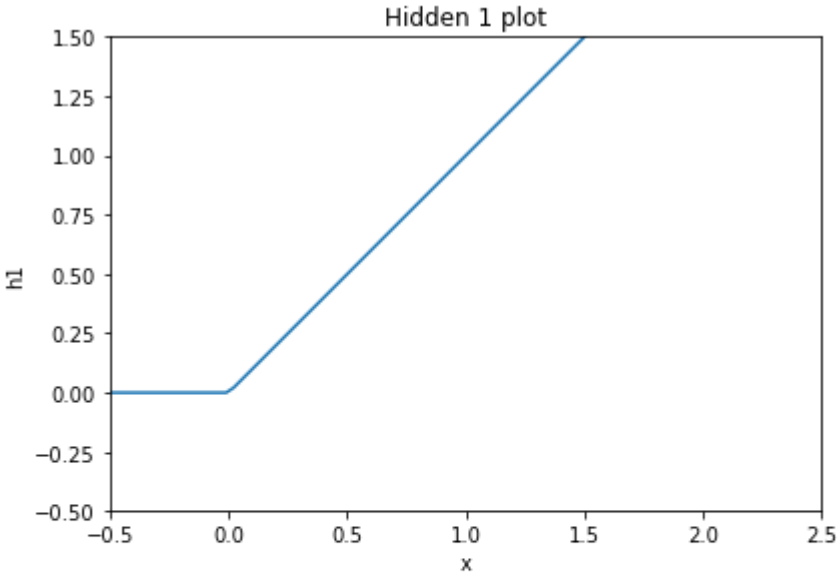
In [8]:

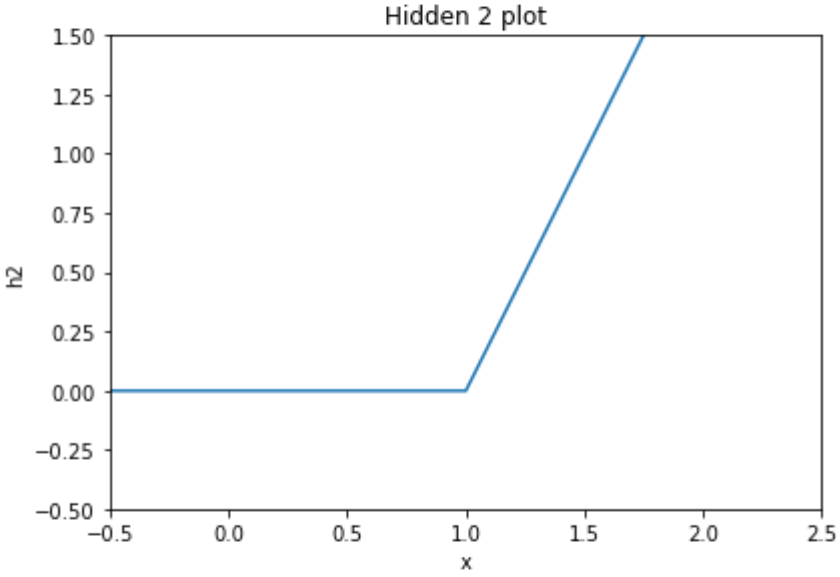
In [9]:

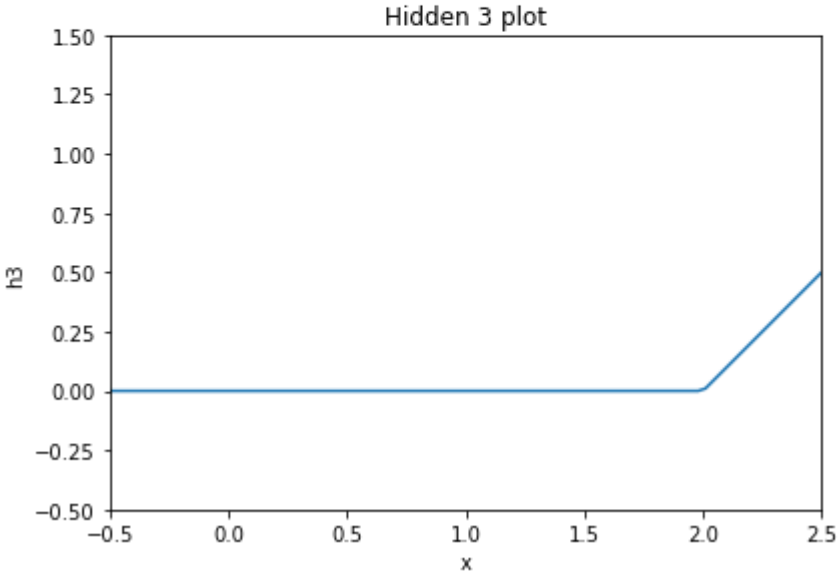
Let's check how the results look visually, starting with the hidden layer activations:

In [10]:

In [11]:


Hidden 1 plot
h1
x
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.5
0.0
0.5
1.0
1.5
2.0
2.5


Hidden 2 plot
h2
x
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

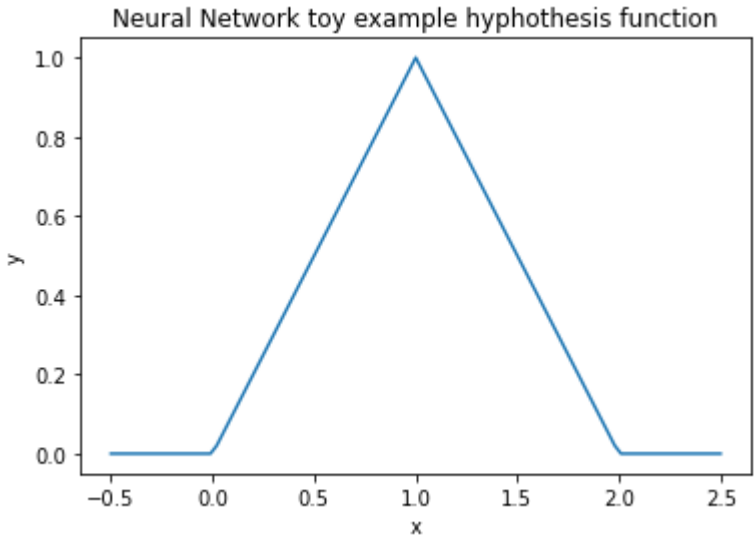

Hidden 3 plot
h3
x
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.5
0.0
0.5
1.0
1.5
2.0
2.5

Now let's add the hidden activations multiplied by `weights_2` together, and plot the final hyphothesis:

In [12]:

Out[12]:

```
[<matplotlib.lines.Line2D at 0x7f0a69f53940>]
```



In [13]:

In [14]:

Dataset for this assignment

We are going to reuse the dataset we have been working with in the previous assignments, but this time we will formulate a different ML problem: predicting air temperature by using history records, in particular, the label is air temperature at 00:00 of a given day, the features are air temperatures of previous 5 days at the same time, 00:00.

In [15]:

Out[15]:

	year	m	d	time	air temperature	pre_1	pre_2	pre_3	pre_4	pre_5
10	2020	1	6	00:00	1.5	-1.6	1.7	4.6	3.6	1.5
12	2020	1	7	00:00	4.5	1.5	-1.6	1.7	4.6	3.6
14	2020	1	8	00:00	5.3	4.5	1.5	-1.6	1.7	4.6
16	2020	1	9	00:00	2.1	5.3	4.5	1.5	-1.6	1.7
18	2020	1	10	00:00	-2.0	2.1	5.3	4.5	1.5	-1.6

Student Task A7.2 In this problem formulation, a datapoint represents a day corresponding to a row in the dataframe. The column "air temperature" stores the labels and columns 'pre_1', 'pre_2', 'pre_3', 'pre_4', 'pre_5' are used as features. Before training an MLP, let's firstly try out the old method we used before: PolynomialRegression. As usual, we create the feature matrix and label vector and then train several models with different polynomial degrees to see which one is ideal. **Your task** is: create a feature matrix and a label vector

In [18]:

```
(708, 5)
(708,)
```

In [19]:

In [20]:

In [21]:

In [22]:

Out[22]:

	poly degree	linear_train_errors	linear_val_errors
0	1	3.947190	5.547871
1	2	3.789401	5.444261
2	3	3.344839	5.818018
3	4	2.679989	18.741724

Student Task A7.3 Answer the following quiz questions by setting the corresponding variable to the index of the answer that you consider correct. Question 1: Which of the models from A7.2 would you recommend based on the above table? - Answer 1: Degree 1 - Answer 2: Degree 2 - Answer 3: Degree 3 - Answer 4: Degree 4

In [24]:

```
my answer is:  2
```

In [25]:

In [26]:

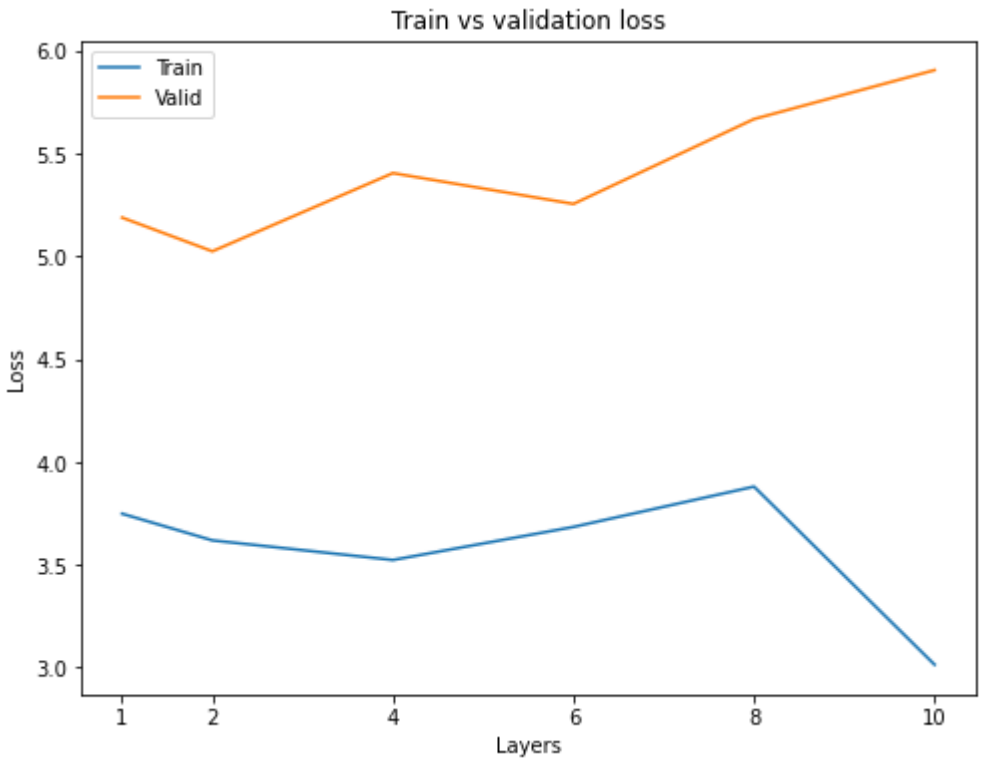
Student Task A7.4 Now, we know the performance of polynomial regression on this prediction task.

It is time to try out MLP and see whether MLP can defeat polynomial regression or not. We will still focus on Sklearn library and use Sklearn class [MLPRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html) to implement our model, but for complicated modern deep learning tasks, Python provides other easy-to-use libraries for the design and training of ANN, such as [Keras](https://keras.io/). **Hypothesis Space used in this task - MLP Structure:** - one input layer consists of the individual features (5 features) and is the entry point to the MLP. - several hidden layers with 15 neuron units in each layer and [ReLU activation function](https://en.wikipedia.org/wiki/Rectifier_(neural_networks)), we will explore the number of hidden layers in this task, find out the ideal number of hidden layers for this given ML problem. - one final output layer with 1 neuron unit. For regularization strength and learning rate, default values are used. **Loss used in this task**: MSE In the following solution cell, you will: - Initialise an MLPRegressor, please use the `hidden_layer_sizes` defined for you and set `max_iter` to 1000, `random_state` to 42. - Train the regressor on the training set. - Evaluate the regressor on the training set and validation set.

In [28]:

In [29]:

In [30]:



In [31]:

Out[31]:

	num_hidden_layers	mlp_train_errors	mlp_val_errors
0	1	3.748909	5.189208
1	2	3.619271	5.025214
2	4	3.523818	5.405671
3	6	3.684636	5.255682
4	8	3.880760	5.668580
5	10	3.014633	5.906500

Student Task A7.5 Answer the following quiz questions by setting the corresponding variable to the index of the answer that you consider correct. Question 1: Which of the models from A7.4 would you recommend based on the table above? - Answer 1: 1 hidden layer MLP - Answer 2: 2 hidden layers MLP - Answer 3: 4 hidden layers MLP - Answer 4: 6 hidden layers MLP - Answer 5: 8 hidden layers MLP - Answer 6: 10 hidden layers MLP

In [32]:

```
my answer is: Answer 2
```

In [33]:

In [34]:

In [35]:

```
training errors and validation errors of PolynomialRegression
```

Out[35]:

	poly degree	linear_train_errors	linear_val_errors
0	1	3.947190	5.547871
1	2	3.789401	5.444261
2	3	3.344839	5.818018
3	4	2.679989	18.741724

In [36]:

```
training errors and validation errors of MLP
```

Out[36]:

PolynomialRegression vs MLP

	num_hidden_layers	mlp_train_errors	mlp_val_errors
0	1	3.748909	5.180208
1	2	3.619271	5.252114
2	4	3.523818	5.405671
3	6	3.684636	5.255682
4	8	3.880760	5.668580
5	10	3.014633	5.906500

The tables above compare the performance of PolynomialRegression and MLP on this specific ML problem. We can see that their performances are similar, but MLP is a bit better than PolynomialRegression. Considering sampling randomness, this suggests that deep learning methods are not always undisputed winners, particularly for simpler tasks where a simple model would train faster, predict quicker, and achieve comparable results.

However, one significant advantage of MLP, even for simple problems, is that it is much less sensitive to model complexity than regression, so even without careful hyperparameter tuning the network always somehow 'does the job for us' with decent validation errors ~5, while regression explodes with an error of 18+ when we use a too complex model (max poly degree=4).

Student Task A7.6 **Train MLP classifiers and evaluate them on both training set and validation set.** **Problem Formulation**: just like in Assingment3, we first categorize air temperature into 4 categories, for each datapoint (a day), the label is categorized air temperature of a given day at 00:00 and the features are air temperatures of previous 5 days at 00:00. **Hypothesis Space - MLP Structure:** - one input layer consists of the individual features (5 features) and is the entry point to the MLP - several hidden layers with 10 neuron units and [ReLU activation function] (https://en.wikipedia.org/wiki/Rectifier_(neural_networks)), we will explore the number of hidden layers in this task, find out the ideal number of hidden layers for this given ML problem. - one final output layer with 4 neuron units and [softmax activation function] (https://en.wikipedia.org/wiki/Softmax_function) For regularization strength and learning rate, default values are used. **Loss**: log-loss

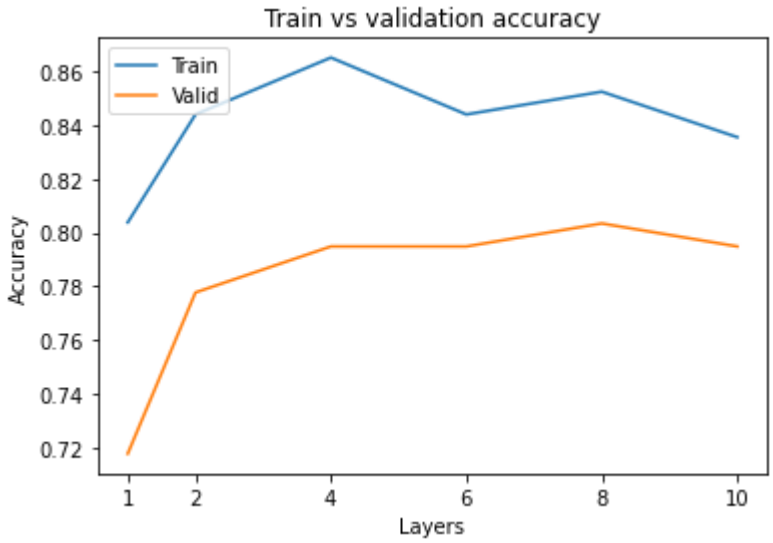
In [37]:

In the following solution cell, you will: - Initialise an MLPClassifier, please use the `hidden_layer_sizes` defined for you and set `max_iter` to 5000, `learning_rate_init` to 0.0001 and `random_state` to 0. - Train the classifier on the training set. - Evaluate the classifier on the training set and validation set. **Sklearn class**: [MLPClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.M provides the methods you need to complete model training. **Model performance evaluation**: calculate the train and validation accuracy (average **0/1 loss**). The function [accuracy_score()] (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html) has been imported for you in the beginning of this notebook.

In [38]:

In [40]:

In [41]:



In [42]:

Out[42]:

	num_hidden_layers	MLP_train_accs	MLP_val_accs
0	1	0.803797	0.717949
1	2	0.843882	0.777778
2	4	0.864979	0.794872
3	6	0.843882	0.794872
4	8	0.852321	0.803419
5	10	0.835443	0.794872

Student Task A7.7 Answer the following quiz questions by setting the corresponding variable to the index of the answer that you consider correct. Question 1: Which of the models from A7.6 would you recommend based on the table above? - Answer 1: 1 hidden layer MLP - Answer 2: 2 hidden layers MLP - Answer 3: 4 hidden layers MLP - Answer 4: 6 hidden layers MLP - Answer 5: 8 hidden layers MLP - Answer 6: 10 hidden layers MLP - Answer 7: 12 hidden layers MLP

In [43]:

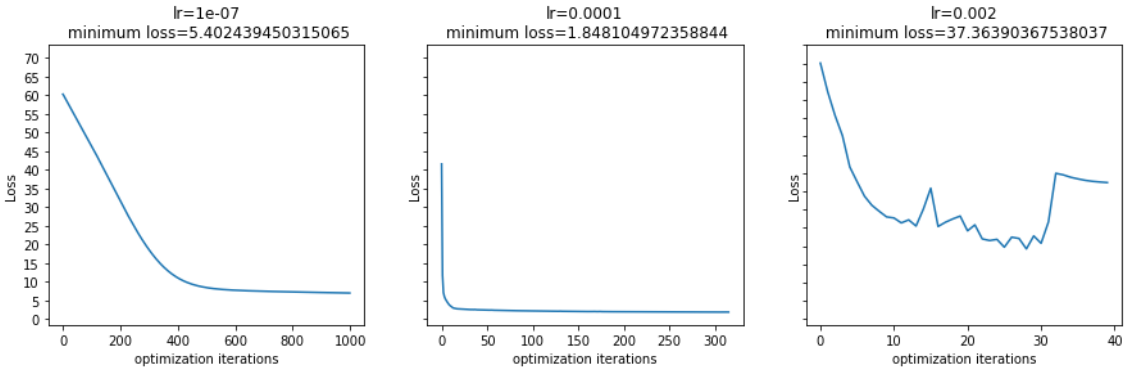
```
my answer is: Answer 5
```

In [44]:

Demo: Learning rate The main learning algorithm for deep learning is gradient descent. From calculus courses, you should be familiar with what a gradient (or a derivative for a single variable function) represents. By using it, we can learn how our function is behaving in a given datapoint (is it rising or falling?), and based on that, we know 'which direction to go' in order to minimize it. Neural networks learn by computing the gradient of the loss function, which gives them the information on how to change their weights to move towards the local minimum. Typically we would take an average direction based on a batch of datapoints, usually 16/32/64 of them at a time. However, the gradient just gives us the direction, we do not know how far in that direction we should go. We control that using a number called the **learning rate**, which lets us scale the steps we take. ``` weight_update = update_from_the_gradient * learning_rate ``` If we are too careful and set a very small learning rate, taking baby steps towards the minimum, our training would take too long. On the other hand, if the learning rate is too big, we might 'overshoot', and keep missing the optimal point. This means that learning rate is usually the most important training parameter and should be chosen carefully, and even adjusted during the training - luckily most libraries provide us with good default values and training algorithms. Gradient descent is a complex topic, and the above explanation is very simplified. We highly recommend [this video](https://www.youtube.com/watch?v=IHZwWFHWa-w) by 3blue1brown, showing the process in greater detail, with easy to understand animations. Take a look at the code below, showing what happens to the training process when we change the learning rate:

In [45]:

```
/opt/conda/lib/python3.8/site-packages/sklearn/neural_network/_mult
ilayer_perceptron.py:582: ConvergenceWarning: Stochastic Optimizer:
Maximum iterations (3000) reached and the optimization hasn't conve
rged yet.
  warnings.warn(
```



Student task A7.8 Answer the following quiz questions by setting the corresponding variable to the index of the answer that you consider correct: Question1: 'Larger learning rate is always better, since it lets our model converge faster, reducing the length of the trainig.' - is this sentence true or false? - Answer 1: True - Answer 2: False

In [46]:

```
my answer is: Answer 2
```

In [ ]:

Demo: Grid search By now, you have experienced model selections based on different parameters, such as max poly degree for Polynomial Regression, alpha for Ridge Regression, number of hidden layers for MLP, etc. These parameters are called hyperparameters, because they are not directly learnt within regressors/classifiers, but rather a property of the training process selected by humans. In sklearn they are passed as arguments to the constructor of the regressor/classifier instances or feature transforming classes. In previous assignments and tasks, We used for loops to iterate a list of candidate hyperparameters to train corresponding models and compare the train/validation errors to do final model selection/adjusting. But most models, especially complicated models such as ANNs, have multiple hyper-parameters. Using (deeply) nested for-loops to search for a best combination from a large hyper-parameter space will make the code quite tedious, so sklearn provides a grid search method [GridSearchCV](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection to exhaustively search candidates from a grid of parameter values specified with the `param_grid` parameter. For instance, the following param_grid will be used in this demo of hyperparameters for tuning/adjusting the MLPClassifier: `param_grid = {`

```
    'hidden_layer_sizes': [(15,15),(10)],
    'learning_rate_init':[0.001,0.01]
}
```

We first define the model (mlp_grids). GridSearchCV method is used to `fit()` the model for different combinations of the hyper-parameters specified in `param_grid` and give the best combination based on the validation accuracies. `cv` is the parameter to specify how many folds will be used for cross validation. For more details, please read the documentation. Note: Do not worry about ConvergenceWarnings - since we try various hyperparameters, it is normal that not all of the models converge properly.

In [47]:

In [48]:

```
Best hyper-parameters found:

 {'hidden_layer_sizes': (15, 15), 'learning_rate_init': 0.001}
```

Student Task A7.9 Please carefully read the documentation of [GridSearchCV](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection figure out how to obtain the attribute: best score (it is the best validation accuracy) for the classifiers trained in the demo and assign it to the variable `best_accuracy`.

In [50]:

```
Best validation accuracy:

 0.8037858355808611
```

In [51]:

Student Task A7.10 Answer the following quiz questions by setting the corresponding variable to the index of the answer that you consider correct. Question 1: "If the activation functions are all linear

functions, Neural Network only acts as a linear hypothesis mapper." Is this statement correct? - Answer 1: Yes, it is correct. - Answer 2: No, it is not correct. Question 2: "Neural Networks can approximate any continuous function." Is this statement correct? - Answer 1: Yes, it is correct. - Answer 2: No, it is not correct.

In [54]:

```
My answer for Question1 is: 1
My answer for Question2 is: 1
```

In [55]:

In [56]: