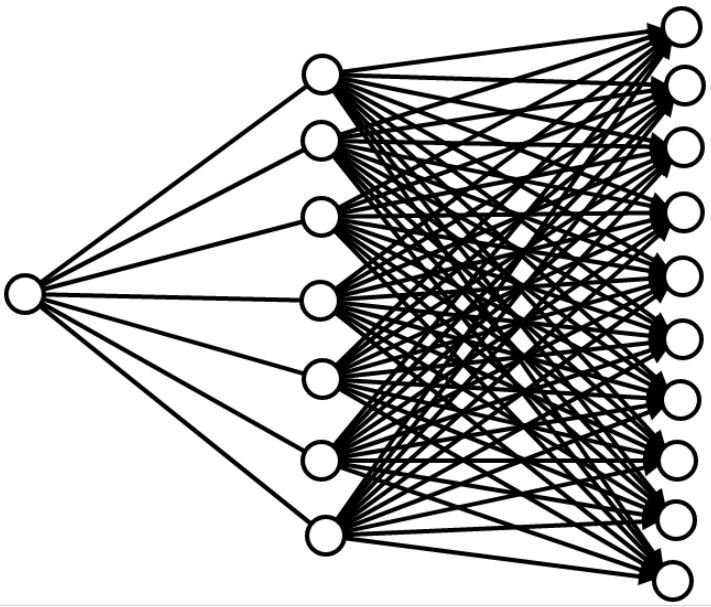
# CS-E4850 Computer Vision
# Exercise Round 9

Alex Herrero Pons

Spring Term Course 2020-2021

**Exercise 1.** Neural networks and backpropagation. (Pen & paper problem)
**Task:** Your task is to compute the expressions for the partial derivates $\partial E/\partial W_{uv}^{(l)}$ in a special case, where we have two layers in the network, the hidden layer has 7 neurons, and the softmax layer has 10 neurons, i.e. $L = 2$, $n_1 = 7$ and $n_2 = 10$. Draw a schematic picture of the network (Figure 1) and proceed with the following stages.



Input layer $\in \mathbb{R}^1$ Hidden layer $\in \mathbb{R}^7$ Output layer $\in \mathbb{R}^{10}$

Figure 1: Schematic picture of the network.

1. Use the cross-entropy loss of Eq. (1) and assume $m = 1$. That is, first compute the partial derivatives for single training sample $\mathbf{x}$, $\mathbf{t}$.

   **Solution.** Having $m = 1$ we obtain,

   $$E = \frac{1}{m}\sum_{j=1}^{m} -\mathbf{t}_j \cdot \log(\mathbf{y}_j) = -t \cdot \log(\mathbf{y}) = -t \cdot \log(\sigma(\mathbf{W}x)) = -t \cdot \log\left(\frac{1}{1+e^{-\mathbf{W}x}}\right).$$

   Then the partial derivatives are,

$$\frac{\partial E}{\partial x} = -\frac{t}{\sigma(\mathbf{W}x))} \cdot \frac{\partial \sigma(\mathbf{W}x)}{\partial x} = -t(1+e^{-\mathbf{W}x}) \cdot \mathbf{W}\frac{1}{1+e^{-\mathbf{W}x}}\left(1 - \frac{1}{1+e^{-\mathbf{W}x}}\right)$$

$$\frac{\partial E}{\partial x} = -t\mathbf{W}\frac{e^{-\mathbf{W}x}}{1+e^{-\mathbf{W}x}}$$

$$\frac{\partial E}{\partial t} = -\log\left(\frac{1}{1+e^{-\mathbf{W}x}}\right).$$

2. Show that $\partial E/\partial \mathbf{z}^{(2)} = (\mathbf{y}^{(2)} - \mathbf{t})^\top$.

**Solution.** Using the chain rule we know that,

$$\frac{\partial E}{\partial z_i^{(2)}} = \sum_{j=1}^{n} \frac{\partial E_j}{\partial y_j^{(2)}} \frac{\partial y_j^{(2)}}{\partial z_i^{(2)}}$$

$$\frac{\partial E_j}{\partial y_j^{(2)}} = \frac{\partial(-t_j \cdot \log(y_j))}{\partial z_i^{(2)}} = -\frac{t_j}{y_i}.$$

So, if $i = j$:

$$\frac{\partial y_j^{(2)}}{\partial z_i^{(2)}} = \frac{\partial \sigma(Z_j^{(2)})}{\partial z_i^{(2)}} = -y_j y_i,$$

and if $i \neq j$:

$$\frac{\partial y_j^{(2)}}{\partial z_i^{(2)}} = \frac{\partial \sigma(z_i^{(2)})}{\partial z_i^{(2)}} = y_i(1 - y_i).$$

Now assuming that $\sum_{j=1}^{m} t_j = 1$ and deriving from the previous equations we see that,

$$\frac{\partial E}{\partial z_i^{(2)}} = \sum_{j=1}^{n} \frac{\partial E_j}{\partial y_j^{(2)}} \frac{\partial y_j^{(2)}}{\partial z_i^{(2)}} = \sum_{j=i} \frac{\partial E_j}{\partial y_j^{(2)}} \frac{\partial y_j^{(2)}}{\partial z_i^{(2)}} + \sum_{j\neq i} \frac{\partial E_j}{\partial y_j^{(2)}} \frac{\partial y_j^{(2)}}{\partial z_i^{(2)}} =$$

$$= \sum_{i\neq j}\left(-\frac{t_j}{y_i}(-y_i y_j)\right) - \frac{t_j}{y_i} y_i(1 - y_i) =$$

$$= \sum_{i\neq j}(t_j y_i) + t_i y_i - t_i = y_i - t_i.$$

Hence,

$$\frac{\partial E}{\partial \mathbf{z}^{(2)}} = (\mathbf{y}^{(2)} - \mathbf{t})^\top.$$

3. Show that $\partial E/\partial \mathbf{y}^{(1)} = (\mathbf{y}^{(2)} - \mathbf{t})^\top \mathbf{W}^{(2)}$.

**Solution.** Having,

$$\frac{\partial E}{\partial \mathbf{z}^{(2)}} = (y^{(2)} - \mathbf{t})^\top,$$

we can obtain that,

$$\frac{\partial E}{\partial y^{(1)}} = \frac{\partial E}{\partial \mathbf{z}^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial \mathbf{y}^{(2)}} = (y^{(2)} - \mathbf{t})^\top \cdot \mathbf{W}^{(2)}.$$

4. Show that $\partial E/\partial W_{uv}^{(2)} = (y_u^{(2)} - t_u)y_v^{(1)}$ and hence $\partial E/\partial \mathbf{W}^{(2)} = (\mathbf{y}^{(2)} - \mathbf{t})\mathbf{y}^{(1)\top}$.

   **Solution.** We can express,

$$\frac{\partial E}{\partial W_{uv}^{(2)}} = \frac{\partial E}{\partial \mathbf{z}^{(2)}}\frac{\partial \mathbf{z}^{(2)}}{\partial W_{uv}^{(2)}} = \left(\frac{\partial E}{\partial \mathbf{z}^{(2)}}\right)_u y_v^{(1)} = (y_u^{(2)} - t_u)y_v^{(1)}$$

   So,

$$\frac{\partial E}{\partial \mathbf{W}^{(2)}} = (\mathbf{y}^{(2)} - \mathbf{t})\mathbf{y}^{(1)\top}$$

5. Show that $\partial \mathbf{y}^{(1)}/\partial \mathbf{z}^{(1)} = \text{diag}(\mathbf{y}^{(1)}. * (\mathbf{1} - \mathbf{y}^{(1)}))$, where $.*$ denotes element-wise multiplication, $\mathbf{1}$ is a vector of ones and $\text{diag}(\cdot)$ converts the input vector to a diagonal matrix (similar to Matlab function `diag`).

   **Solution.** Having,

$$\frac{\partial \sigma(\mathbf{z})}{\partial \mathbf{z}} = \frac{e^{-\mathbf{z}}}{(1+e^{-\mathbf{z}})^2} = \frac{1}{1+e^{-\mathbf{z}}} \cdot \frac{e^{-\mathbf{z}}}{1+e^{-\mathbf{z}}} = \sigma(\mathbf{z})(1-\sigma(\mathbf{z})),$$

   we obtain,

$$\frac{\partial \mathbf{y}^{(1)}}{\partial \mathbf{z}^{(1)}} = \mathbf{y}^{(1)}(1-y^{(1)}) = \text{diag}(\mathbf{y}^{(1)}. * (\mathbf{1} - \mathbf{y}^{(1)})).$$

6. Show that $\partial E/\partial \mathbf{z}^{(1)} = (\mathbf{y}^{(2)} - \mathbf{t})^\top \mathbf{W}^{(2)} \text{diag}(\mathbf{y}^{(1)}. * (\mathbf{1} - \mathbf{y}^{(1)}))$.

   **Solution.**

$$\frac{\partial E}{\partial \mathbf{z}^{(1)}} = \frac{\partial E}{\partial \mathbf{y}^{(1)}}\frac{\partial \mathbf{y}^{(1)}}{\partial \mathbf{z}^{(1)}} = (\mathbf{y}^{(2)} - \mathbf{t})^\top \mathbf{W}^{(2)} \text{diag}(\mathbf{y}^{(1)}. * (\mathbf{1} - \mathbf{y}^{(1)}))$$

7. Show that $\partial E/\partial W_{uv}^{(1)} = \frac{\partial E}{\partial z_u^{(1)}} x_v$ and hence $\partial E/\partial \mathbf{W}^{(1)} = \left(\frac{\partial E}{\partial \mathbf{z}^{(1)}}\right)^\top \mathbf{x}^\top$, where vector $\frac{\partial E}{\partial \mathbf{z}^{(1)}}$ is computed above.

   **Solution.**

$$\frac{\partial E}{\partial W_{uv}^{(1)}} = \frac{\partial E}{\partial \mathbf{z}^{(1)}}\frac{\partial \mathbf{z}^{(1)}}{\partial W_{uv}^{(1)}} = \frac{\partial E}{\partial z_u^{(1)}}\frac{\partial z_u^{(1)}}{\partial W_{uv}^{(1)}} = \frac{\partial E}{\partial z_u^{(1)}}\frac{\partial \sum_i W_{ui}x_i}{\partial W_{uv}^{(1)}} = \frac{\partial E}{\partial z_u^{(1)}}x_v$$

   So,

$$\frac{\partial E}{\partial \mathbf{W}^{(1)}} = \left(\frac{\partial E}{\partial \mathbf{z}^{(1)}}\right)^\top \mathbf{x}^\top$$

8. Finally, if we have multiple training samples (i.e. $m > 1$) we can average the elements of the corresponding matrices $\partial E/\partial \mathbf{W}^{(2)}$ and $\partial E/\partial \mathbf{W}^{(1)}$ that contain the partial derivatives of the network weights for each sample.

   **Solution.** I don't know.

9. Further, if we use weight decay as in Eq. (2), its contribution to each partial derivative is simply $\lambda$ times the value of the corresponding weight (i.e. $\lambda W_{uv}^{(l)}$) and it can be added in the end.

   **Solution.** I don't know.

# ExerciseRound09

November 13, 2020

[1]:
```
# This cell is used for creating a button that hides/unhides code cells to␣
 ↪quickly look only the results.
# Works only with Jupyter Notebooks.

from IPython.display import HTML

HTML('''<script>
code_show=true;
function code_toggle() {
if (code_show){
$('div.input').hide();
} else {
$('div.input').show();
}
code_show = !code_show
}
$( document ).ready(code_toggle);
</script>
<form action="javascript:code_toggle()"><input type="submit" value="Click here␣
 ↪to toggle on/off the raw code."></form>''')
```

[1]:
```
<IPython.core.display.HTML object>
```

[2]:
```
# Description:
#   Exercise09 notebook.
#
# Copyright (C) 2018 Santiago Cortes, Juha Ylioinas
#
# This software is distributed under the GNU General Public
# Licence (version 2 or later); please refer to the file
# Licence.txt, included with the software, for details.

# Preparations
import os
import numpy as np
import matplotlib.pyplot as plt
```

```
from utils import from_data_file, theta_to_model, model_to_theta,␣
↪initial_model, logistic, \
    log_sum_exp_over_rows, classification_performance

# Select data directory

if os.path.isdir('/coursedata'):
    # JupyterHub
    course_data_dir = '/coursedata'
elif os.path.isdir('../../../coursedata'):
    # Local installation
    course_data_dir = '../../../coursedata'
else:
    # Docker
    course_data_dir = '/home/jovyan/work/coursedata/'

print('The data directory is %s' % course_data_dir)
data_dir = os.path.join(course_data_dir, 'exercise-09-data')
print('Data stored in %s' % data_dir)
```

```
The data directory is /coursedata
Data stored in /coursedata/exercise-09-data
```

# 1 CS-E4850 Computer Vision Exercise Round 9

The problems should be solved before the exercise session and solutions returned via MyCourses. Upload the files: (1) a pdf file containing your written answers to Exercise 1, and (2) both the pdf and .ipynb versions of the notebook containing your answers to Exercises 2 and 3. Scanned, **neatly** handwritten, solutions are ok for problem 1. If possible, combine your pdf solution of Exercise 1 with the notebook pdf into a single pdf and return that with the notebook .ipynb file.

Notice also that the last two problems can be done without solving Exercise 1 since the solutions are already written out in the subtasks of Exercise 1 (i.e. only the derivations are missing and asked in Exercise 1).

If you have not studied basics of neural networks in previous courses and the problem context of these exercises is not clear, it may be helpful to check the slides of the first four lectures of prof. Hinton's course "Introduction to neural networks and machine learning":
http://www.cs.toronto.edu/~hinton/coursera_slides.html
http://www.cs.toronto.edu/~hinton/coursera_lectures.html (lecture videos).

## 1.1 Exercise 1 - Neural networks and backpropagation

This is a pen-&-paper problem. See Exercise09penandpaper.pdf for the questions.

## 1.2 Exercise 2 - Image classification using a neural network

The first exercise problem above gives the solution to Part 2 of the second programming assignment of professor Hinton's course "Introduction to neural net-

works and machine learning". The assignment and related material are available at https://www.cs.toronto.edu/~tijmen/csc321/assignments/a2/.

Check out the contents of the above web page and complete the programming task of Part 2 according to the instructions given there. The code template for the python version is below. The solution for the pen and paper part of the task is already given above in **Exercise 1**. Hence, the programming part is a relatively straightforward implementation and can be done without carrying out the derivations since the results of the derivations are already given in **Exercise 1** above.

[3]:
```
def test_gradient(model, data, wd_coefficient):
    import sys
    base_theta = model_to_theta(model)
    h = 1e-2
    correctness_threshold = 1e-5
    analytic_gradient_struct = d_loss_by_d_model(model, data, wd_coefficient)

    analytic_gradient = model_to_theta(analytic_gradient_struct);
    if True in np.isnan(analytic_gradient) or True in np.
↪isinf(analytic_gradient):
        sys.exit('Your gradient computation produced a NaN or infinity. That is␣
↪an error.')
    # We want to test the gradient not for every element of theta, because␣
↪that's a
    # lot of work. Instead, we test for only a few elements. If there's an␣
↪error, this
    # is probably enough to find that error.
    # We want to first test the hid_to_class gradient, because that's most␣
↪likely
    # to be correct (it's the easier one).
    # Let's build a list of theta indices to check. We'll check 20 elements of
    # hid_to_class, and 80 elements of input_to_hid (it's bigger than␣
↪hid_to_class).
    input_to_hid_theta_size = model['input_to_hid'].size
    hid_to_class_theta_size = model['hid_to_class'].size
    big_prime = 1299721; # 1299721 is prime and thus ensures a somewhat␣
↪random-like selection of indices.
    hid_to_class_indices_to_check = np.mod(big_prime * np.arange(20),␣
↪hid_to_class_theta_size) \
                                    + input_to_hid_theta_size
    input_to_hid_indices_to_check = np.mod(big_prime * np.arange(80),␣
↪input_to_hid_theta_size)
    a = hid_to_class_indices_to_check[np.newaxis,:]
    b = input_to_hid_indices_to_check[np.newaxis,:]
    indices_to_check = np.ravel(np.hstack((a,b)))

    for i in range(100):
        test_index = indices_to_check[i]
```

```
        analytic_here = analytic_gradient[test_index]
        theta_step = base_theta * 0
        theta_step[test_index] = h
        contribution_distances = np.array([-4.,  -3.,  -2.,  -1.,   1.,   2.,
↪3.,   4.])
        contribution_weights = np.array([1/280., -4/105., 1/5., -4/5., 4/5., -1/
↪5., 4/105., -1/280.])
        temp = 0;
        for contribution_index in range(8):
            temp = temp + loss(theta_to_model(base_theta + theta_step * \
                                                ␣
↪contribution_distances[contribution_index]), data, wd_coefficient) * \
                                                ␣
↪contribution_weights[contribution_index]
        fd_here = temp / h
        diff = np.abs(analytic_here - fd_here)

        if True in (diff > correctness_threshold) and \
            True in (diff / (np.abs(analytic_here) + np.abs(fd_here)) >␣
↪correctness_threshold):
            part_names = ['input_to_hid', 'hid_to_class']
            sys.exit('Theta element #{} (part of {}), with value {}, has finite␣
↪difference gradient {} but analytic gradient {}. That looks like an error.
↪\n'.format(test_index, part_names[(i<=20)], base_theta[test_index], fd_here,␣
↪analytic_here))
        if i==20:
            print('Gradient test passed for hid_to_class. ')
        if i==100:
            print('Gradient test passed for input_to_hid. ')
    print('Gradient test passed. That means that the gradient that your code␣
↪computed is within 0.001%% of the gradient that the finite difference␣
↪approximation computed, so the gradient calculation procedure is probably␣
↪correct (not certainly, but probably).\n')

def forward_pass(model, data):
    # This function does the forward pass through the network: calculating the␣
↪states of all units, and some related data.
    # This function is used in functions loss() and d_loss_by_d_model().

    # model.input_to_hid is a matrix of size <number of hidden units> by␣
↪<number of inputs i.e. 256>. It contains the weights from the input units to␣
↪the hidden units.
    # model.hid_to_class is a matrix of size <number of classes i.e. 10> by␣
↪<number of hidden units>. It contains the weights from the hidden units to␣
↪the softmax units.
```

```
    # data.inputs is a matrix of size <number of inputs i.e. 256> by <number of data cases>. Each column describes a different data case.
    # data.targets is a matrix of size <number of classes i.e. 10> by <number of data cases>. Each column describes a different data case. It contains a one-of-N encoding of the class, i.e. one element in every column is 1 and the others are 0.

    hid_input = np.dot(model['input_to_hid'], data['inputs']) # input to the hidden units, i.e. before the logistic. size: <number of hidden units> by <number of data cases>
    hid_output = logistic(hid_input) # output of the hidden units, i.e. after the logistic. size: <number of hidden units> by <number of data cases>
    class_input = np.dot(model['hid_to_class'], hid_output) # input to the components of the softmax. size: <number of classes, i.e. 10> by <number of data cases>

    # The following three lines of code implement the softmax.
    # However, it's written differently from what the lectures say.
    # In the lectures, a softmax is described using an exponential divided by a sum of exponentials.
    # What we do here is exactly equivalent (you can check the math or just check it in practice), but this is more numerically stable.
    # "Numerically stable" means that this way, there will never be really big numbers involved.
    # The exponential in the lectures can lead to really big numbers, which are fine in mathematical equations, but can lead to all sorts of problems in Matlab
    # Matlab isn't well prepared to deal with really large numbers, like the number 10 to the power 1000. Computations with such numbers get unstable, so we avoid them.

    class_normalizer = log_sum_exp_over_rows(class_input) # log(sum(exp of class_input)) is what we subtract to get properly normalized log class probabilities. size: <1> by <number of data cases>
    log_class_prob = class_input - np.tile(class_normalizer, (class_input.shape[0], 1)) # log of probability of each class. size: <number of classes, i.e. 10> by <number of data cases>
    class_prob = np.exp(log_class_prob) # probability of each class. Each column (i.e. each case) sums to 1. size: <number of classes, i.e. 10> by <number of data cases>

    return hid_input, hid_output, class_input, log_class_prob, class_prob

def loss(model, data, wd_coefficient):
    hid_input, hid_output, class_input, log_class_prob, class_prob = forward_pass(model, data);
```

```
    classification_loss = -np.mean(np.sum(np.multiply(log_class_prob,␣
↪data['target']), 0)) # select the right log class probability using that sum;
↪ then take the mean over all data cases.
    wd_loss = (np.sum(np.ravel(model['input_to_hid']) ** 2 ) + np.sum(np.
↪ravel(model['hid_to_class']) ** 2 )) / 2. * wd_coefficient; # weight decay␣
↪loss. very straightforward: E = 1/2 * wd_coeffecient * parameters^2
    ret = classification_loss + wd_loss
    return ret
```

[13]:
```
def d_loss_by_d_model(model, data, wd_coefficient):
    # model.input_to_hid is a matrix of size <number of hidden units> by␣
↪<number of inputs i.e. 256>
    # model.hid_to_class is a matrix of size <number of classes i.e. 10> by␣
↪<number of hidden units>
    # data.inputs is a matrix of size <number of inputs i.e. 256> by <number of␣
↪data cases>
    # data.targets is a matrix of size <number of classes i.e. 10> by <number␣
↪of data cases>

    # The returned object <ret> is supposed to be exactly like parameter␣
↪<model>, i.e. it has fields ret.input_to_hid and ret.hid_to_class, and those␣
↪are of the same shape as they are in <model>.
    # However, in <ret>, the contents of those matrices are gradients (d loss␣
↪by d weight), instead of weights.
    ret = dict()
    # This is the only function that you're expected to change. Right now, it␣
↪just returns a lot of zeros, which is obviously not the correct output. Your␣
↪job is to change that.
    #--your-code-starts-here--#
    W1 = model['input_to_hid'];
    W2 = model['hid_to_class'];

    z1 = np.dot(W1,data['inputs'])
    y1 = logistic(z1)
    z2 = np.dot(W2,y1)

    normlog = log_sum_exp_over_rows(z2)
    n_rows = z2.shape[0]
    logprob = z2 - np.tile(normlog, (n_rows,1))
    prob = np.exp(logprob)

    n_ins = len(data['inputs'][0])
    term1 = np.dot(np.subtract(prob, data['target']).T, W2)
    N = term1.shape[0]
    for i in range(N):
```

```
        row = np.dot(term1[i], np.diag(np.multiply(y1[:,i], np.subtract(np.
↪ones(y1[:,i].shape), y1[:,i]))))
        if i == 0:
            dEdz1 = row
        else:
            dEdz1 = np.vstack((dEdz1, row))

    ret['input_to_hid'] = np.divide(np.dot(dEdz1.T, data['inputs'].T), n_ins)
    ret['hid_to_class'] = np.divide(np.dot(np.subtract(prob,data['target']), y1.
↪T), n_ins)

    #--your-code-ends-here--#
    return ret
```

[5]:
```
def a2(wd_coefficient, n_hid, n_iters, learning_rate, momentum_multiplier,␣
↪do_early_stopping, mini_batch_size):
    model = initial_model(n_hid)
    datas = from_data_file(data_dir)

    n_training_cases = datas['train']['inputs'].shape[1]
    if n_iters != 0:
        print("Now testing the gradient on the whole training set...")
        test_gradient(model, datas['train'], wd_coefficient)

    # optimization
    training_batch = dict()
    best_so_far = dict()
    theta = model_to_theta(model)
    momentum_speed = theta * 0.
    training_data_losses = []
    validation_data_losses = []
    if do_early_stopping:
        best_so_far['theta'] = -1 # this will be overwritten soon
        best_so_far['validation_loss'] = np.Inf
        best_so_far['after_n_iters'] = -1

    for optimization_iteration_i in range(1, n_iters+1):
        model = theta_to_model(theta)
        training_batch_start = np.mod((optimization_iteration_i-1) *␣
↪mini_batch_size, n_training_cases);
        training_batch['inputs'] = datas['train']['inputs'][:,␣
↪training_batch_start : training_batch_start + mini_batch_size]
        training_batch['target'] = datas['train']['target'][:,␣
↪training_batch_start : training_batch_start + mini_batch_size]
        gradient = model_to_theta(d_loss_by_d_model(model, training_batch,␣
↪wd_coefficient))
```

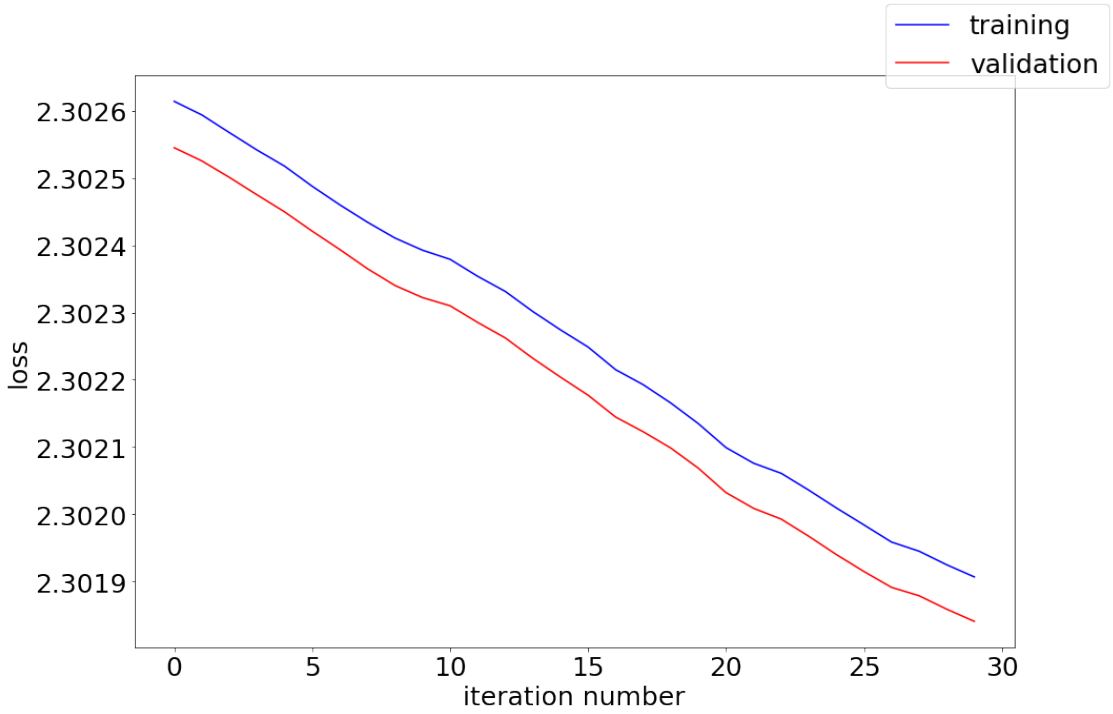
```
        momentum_speed = np.multiply(momentum_speed, momentum_multiplier) -␣
↪gradient;
        theta = theta + momentum_speed * learning_rate;
        model = theta_to_model(theta);
        training_data_losses.append(loss(model, datas['train'], wd_coefficient))
        validation_data_losses.append(loss(model, datas['val'], wd_coefficient))

        if do_early_stopping and validation_data_losses[-1] <␣
↪best_so_far['validation_loss']:
            best_so_far['theta'] = theta; # this will be overwritten soon
            best_so_far['validation_loss'] = validation_data_losses[-1]
            best_so_far['after_n_iters'] = optimization_iteration_i

        if np.mod(optimization_iteration_i, np.round(n_iters / 10.)) == 0:
            print('After {} optimization iterations, training data loss is {},␣
↪and validation data loss is {}\n'.format(optimization_iteration_i,␣
↪training_data_losses[-1], validation_data_losses[-1]))

        if optimization_iteration_i == n_iters: # check gradient again, this␣
↪time with more typical parameters and with a different data size
            print('Now testing the gradient on just a mini-batch instead of the␣
↪whole training set... ')
            test_gradient(model, training_batch, wd_coefficient)

    if do_early_stopping:
        print('Early stopping: validation loss was lowest after {} iterations.␣
↪We chose the model that we had then.\n'.format(best_so_far['after_n_iters']))
        theta = best_so_far['theta']

    # the optimization is finished. Now do some reporting.
    model = theta_to_model(theta)
    if n_iters != 0:
        ax = plt.figure(1, figsize=(15,10))
        plt.plot(training_data_losses, 'b')
        plt.plot(validation_data_losses, 'r')
        plt.tick_params(labelsize=25)
        ax.legend(('training', 'validation'), fontsize=25)
        plt.ylabel('loss', fontsize=25);
        plt.xlabel('iteration number', fontsize=25);
        plt.show()

    datas2 = [datas['train'], datas['val'], datas['test']]
    data_names = ['training', 'validation', 'test'];
    for data_i in range(3):
        data = datas2[data_i]
        data_name = data_names[data_i]
```

```
        print('\nThe total loss on the {} data is {}\n'.format(data_name,␣
↪loss(model, data, wd_coefficient)))
        print('The classification loss (i.e. without weight decay) on the {}␣
↪data is {}\n'.format(data_name, loss(model, data, 0)));
        print('The classification error rate on the {} data is {}\n'.
↪format(data_name, classification_performance(model, data)))
```

[15]:
```
# Start training the neural network
#a2(0, 10, 70, 20.0, 0, False, 4)
a2(0, 10, 30, 0.01, 0, False, 10)
```

```
Now testing the gradient on the whole training set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).

After 3 optimization iterations, training data loss is 2.3025673934869757, and
validation data loss is 2.3025008278520307

After 6 optimization iterations, training data loss is 2.3024876530303207, and
validation data loss is 2.3024208773255115

After 9 optimization iterations, training data loss is 2.3024106986962787, and
validation data loss is 2.302340149057484

After 12 optimization iterations, training data loss is 2.3023537336404405, and
validation data loss is 2.302285147694466

After 15 optimization iterations, training data loss is 2.3022741261667745, and
validation data loss is 2.302203842915163

After 18 optimization iterations, training data loss is 2.302192586688048, and
validation data loss is 2.3021225988199605

After 21 optimization iterations, training data loss is 2.3020990682386935, and
validation data loss is 2.3020321098244865

After 24 optimization iterations, training data loss is 2.302035768066352, and
validation data loss is 2.301967233398054

After 27 optimization iterations, training data loss is 2.301958598374691, and
validation data loss is 2.301891225388429
```

```
After 30 optimization iterations, training data loss is 2.301906765216956, and
validation data loss is 2.301840691197619

Now testing the gradient on just a mini-batch instead of the whole training
set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).
```




```
The total loss on the training data is 2.301906765216956

The classification loss (i.e. without weight decay) on the training data is
2.301906765216956

The classification error rate on the training data is 0.889


The total loss on the validation data is 2.301840691197619

The classification loss (i.e. without weight decay) on the validation data is
2.301840691197619
```

```
The classification error rate on the validation data is 0.895


The total loss on the test data is 2.3018651012099185

The classification loss (i.e. without weight decay) on the test data is
2.3018651012099185

The classification error rate on the test data is 0.8873333333333333
```

When you have completed the programming part, run a2(0, 10, 30, 0.01, 0, False, 10) and report the resulting training data classification loss here:

Training data classification loss: 2.3018651012099185

### 1.3 Exercise 3 - Optimisation using backpropagation

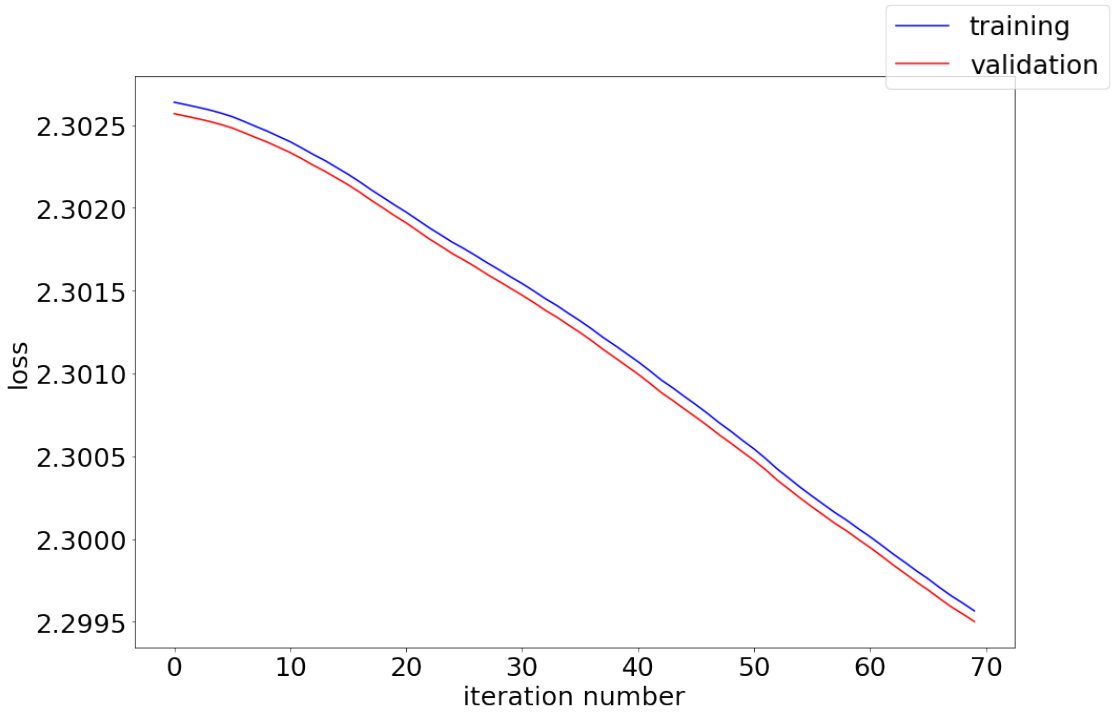
Do Part 3 of the assignment as described at http://www.cs.toronto.edu/~tijmen/csc321/assignments/a2/

The task is to experiment with the example code given above and report your findings. There is no need to program anything in this part but completing it requires that Part 2 is successfully solved.

[17]:
```
#try with learning_rate = 0.002, 0.01, 0.05, 0.2, 1.0, 5.0, and 20.0, and with␣
↪and without momentum_multiplier=0.9

learning_rates = [0.002,0.01,0.05,0.2,1.0,5.0,20.0]
momentum_multiplier=0.9
for lr in learning_rates:
    print("[EXECUTION: learning_rate =",lr, "(with momentum_multiplier␣
↪=",momentum_multiplier,")]")
    a2(0, 10, 70, lr, momentum_multiplier, False, 4)
    print("[EXECUTION: learning_rate =",lr, "(without momentum_multiplier)]")
    a2(0, 10, 70, lr, 0, False, 4)
```

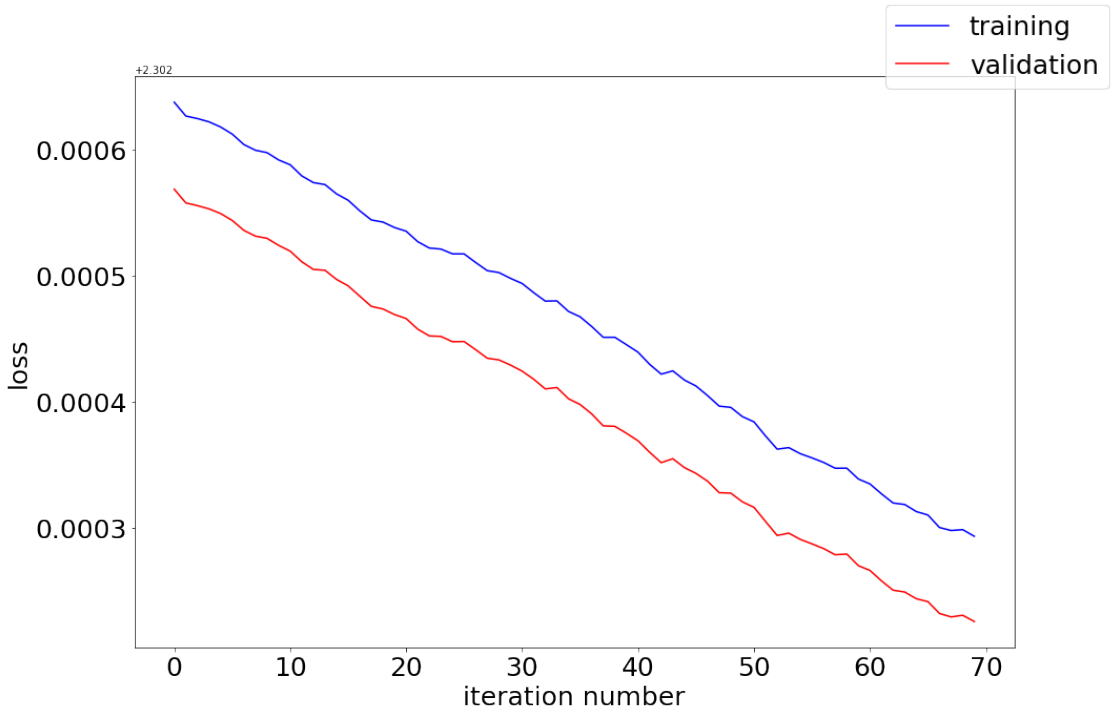
```
[EXECUTION: learning_rate = 0.002 (with momentum_multiplier = 0.9 )]
Now testing the gradient on the whole training set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).

After 7 optimization iterations, training data loss is 2.3025220073253787, and
validation data loss is 2.3024543319222457

After 14 optimization iterations, training data loss is 2.3022850269062314, and
validation data loss is 2.3022204310423118
```

```
After 21 optimization iterations, training data loss is 2.301973937768185, and
validation data loss is 2.301908906063454

After 28 optimization iterations, training data loss is 2.3016673234667206, and
validation data loss is 2.3015975800942

After 35 optimization iterations, training data loss is 2.301362642120604, and
validation data loss is 2.301292370391952

After 42 optimization iterations, training data loss is 2.301016759188716, and
validation data loss is 2.300941506610587

After 49 optimization iterations, training data loss is 2.300650985578013, and
validation data loss is 2.3005793371911767

After 56 optimization iterations, training data loss is 2.3002596286331856, and
validation data loss is 2.3001943676292913

After 63 optimization iterations, training data loss is 2.2999095721273934, and
validation data loss is 2.299843602349599

After 70 optimization iterations, training data loss is 2.2995666217165343, and
validation data loss is 2.299502143963026

Now testing the gradient on just a mini-batch instead of the whole training
set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).
```




```
The total loss on the training data is 2.2995666217165343

The classification loss (i.e. without weight decay) on the training data is
2.2995666217165343

The classification error rate on the training data is 0.848


The total loss on the validation data is 2.299502143963026

The classification loss (i.e. without weight decay) on the validation data is
2.299502143963026

The classification error rate on the validation data is 0.839


The total loss on the test data is 2.29951033761895

The classification loss (i.e. without weight decay) on the test data is
2.29951033761895

The classification error rate on the test data is 0.8507777777777777
```

```
[EXECUTION: learning_rate = 0.002 (without momentum_multiplier)]
Now testing the gradient on the whole training set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).

After 7 optimization iterations, training data loss is 2.3026037735735647, and
validation data loss is 2.302535720632007

After 14 optimization iterations, training data loss is 2.3025720131272993, and
validation data loss is 2.30250405879389

After 21 optimization iterations, training data loss is 2.302535038202373, and
validation data loss is 2.302465773152903

After 28 optimization iterations, training data loss is 2.3025037337757612, and
validation data loss is 2.302434289157999

After 35 optimization iterations, training data loss is 2.302471417748011, and
validation data loss is 2.3024021893657527

After 42 optimization iterations, training data loss is 2.302429461368185, and
validation data loss is 2.3023597450783813

After 49 optimization iterations, training data loss is 2.302395320339885, and
validation data loss is 2.3023273136661047

After 56 optimization iterations, training data loss is 2.302355289106732, and
validation data loss is 2.302287095990645

After 63 optimization iterations, training data loss is 2.3023194864578405, and
validation data loss is 2.302250331078315

After 70 optimization iterations, training data loss is 2.302293108148208, and
validation data loss is 2.3022255410002384

Now testing the gradient on just a mini-batch instead of the whole training
set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).
```




The total loss on the training data is 2.302293108148208

The classification loss (i.e. without weight decay) on the training data is 2.302293108148208

The classification error rate on the training data is 0.897

The total loss on the validation data is 2.3022255410002384

The classification loss (i.e. without weight decay) on the validation data is 2.3022255410002384

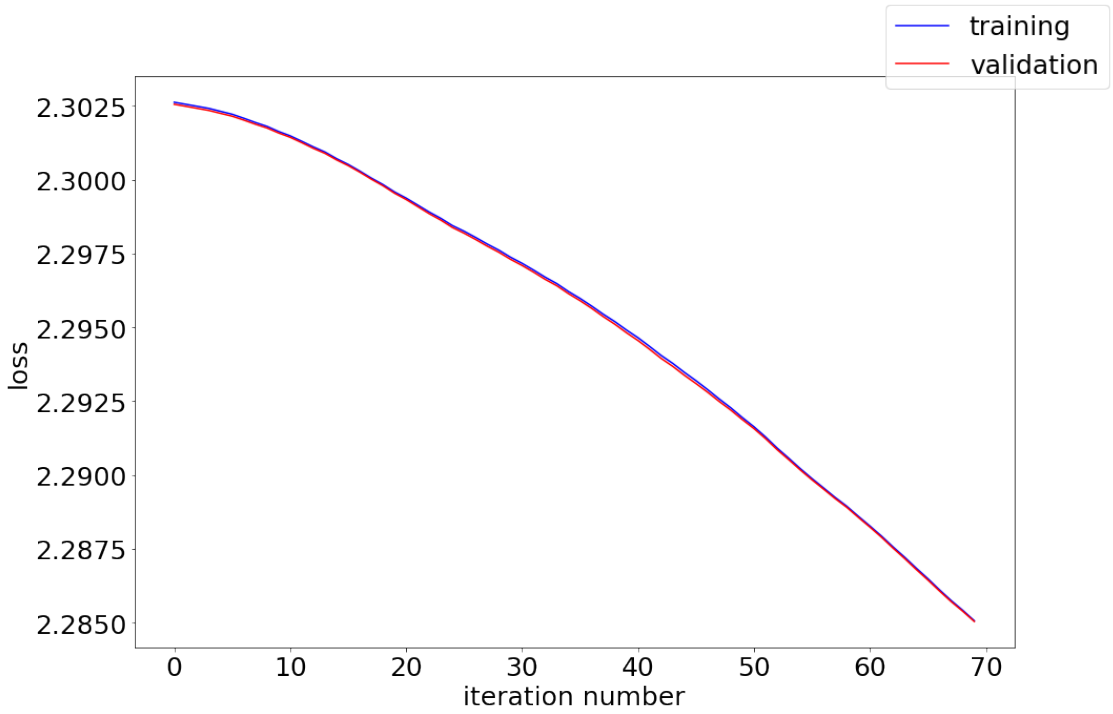
The classification error rate on the validation data is 0.898

The total loss on the test data is 2.3022526843851967

The classification loss (i.e. without weight decay) on the test data is 2.3022526843851967

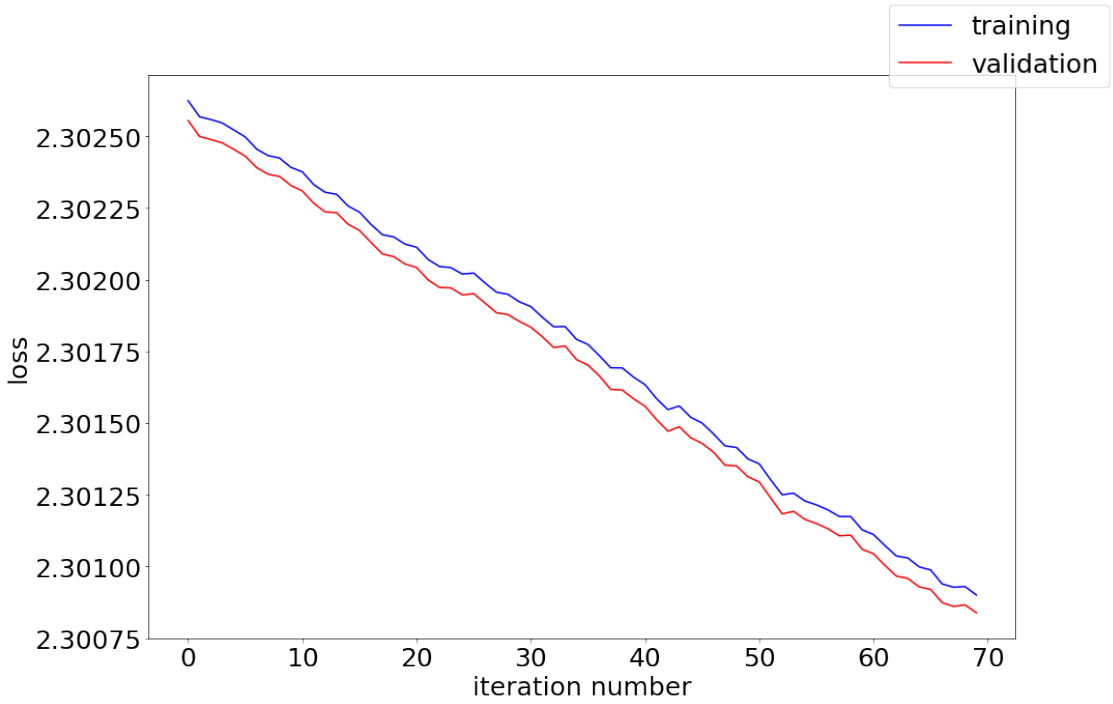
The classification error rate on the test data is 0.895

```
[EXECUTION: learning_rate = 0.01 (with momentum_multiplier = 0.9 )]
Now testing the gradient on the whole training set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).

After 7 optimization iterations, training data loss is 2.3020813945993357, and
validation data loss is 2.3020187410306927

After 14 optimization iterations, training data loss is 2.3009380114989293, and
validation data loss is 2.3008911765811164

After 21 optimization iterations, training data loss is 2.299377220861987, and
validation data loss is 2.2993287631006223

After 28 optimization iterations, training data loss is 2.2978194866837964, and
validation data loss is 2.2977479808271046

After 35 optimization iterations, training data loss is 2.2962135486738218, and
validation data loss is 2.2961398013653374

After 42 optimization iterations, training data loss is 2.294351957179535, and
validation data loss is 2.294253983382684

After 49 optimization iterations, training data loss is 2.292273720128319, and
validation data loss is 2.292195676733541

After 56 optimization iterations, training data loss is 2.2898782825276793, and
validation data loss is 2.2898348258583265

After 63 optimization iterations, training data loss is 2.287560557562744, and
validation data loss is 2.287517928063843

After 70 optimization iterations, training data loss is 2.285062439744801, and
validation data loss is 2.285036193485089

Now testing the gradient on just a mini-batch instead of the whole training
set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).
```




```
The total loss on the training data is 2.285062439744801

The classification loss (i.e. without weight decay) on the training data is
2.285062439744801

The classification error rate on the training data is 0.751


The total loss on the validation data is 2.285036193485089

The classification loss (i.e. without weight decay) on the validation data is
2.285036193485089

The classification error rate on the validation data is 0.756


The total loss on the test data is 2.284912589785349

The classification loss (i.e. without weight decay) on the test data is
2.284912589785349

The classification error rate on the test data is 0.7616666666666667
```

```
[EXECUTION: learning_rate = 0.01 (without momentum_multiplier)]
Now testing the gradient on the whole training set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).

After 7 optimization iterations, training data loss is 2.3024552683451076, and
validation data loss is 2.3023906286460742

After 14 optimization iterations, training data loss is 2.302297582234424, and
validation data loss is 2.302233443721875

After 21 optimization iterations, training data loss is 2.3021129528245314, and
validation data loss is 2.302042276395394

After 28 optimization iterations, training data loss is 2.30195621918477, and
validation data loss is 2.3018846557884483

After 35 optimization iterations, training data loss is 2.301791772390124, and
validation data loss is 2.301721302195465

After 42 optimization iterations, training data loss is 2.3015853406613815, and
validation data loss is 2.3015124422278097

After 49 optimization iterations, training data loss is 2.3014150373714997, and
validation data loss is 2.301350680390506

After 56 optimization iterations, training data loss is 2.3012148850962446, and
validation data loss is 2.301149618073701

After 63 optimization iterations, training data loss is 2.3010365412147777, and
validation data loss is 2.3009664718489464

After 70 optimization iterations, training data loss is 2.3009006724646897, and
validation data loss is 2.3008385536916567

Now testing the gradient on just a mini-batch instead of the whole training
set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).
```




The total loss on the training data is 2.3009006724646897

The classification loss (i.e. without weight decay) on the training data is 2.3009006724646897

The classification error rate on the training data is 0.886

The total loss on the validation data is 2.3008385536916567

The classification loss (i.e. without weight decay) on the validation data is 2.3008385536916567

The classification error rate on the validation data is 0.887

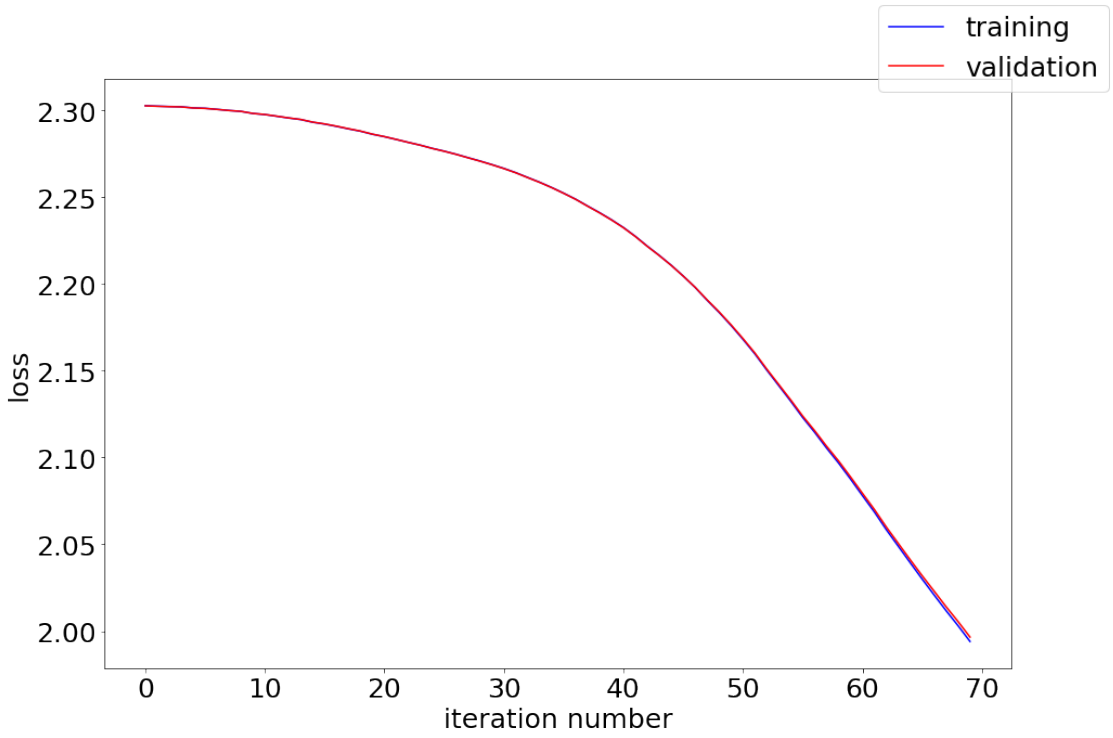
The total loss on the test data is 2.300854001904353

The classification loss (i.e. without weight decay) on the test data is 2.300854001904353

The classification error rate on the test data is 0.8816666666666667

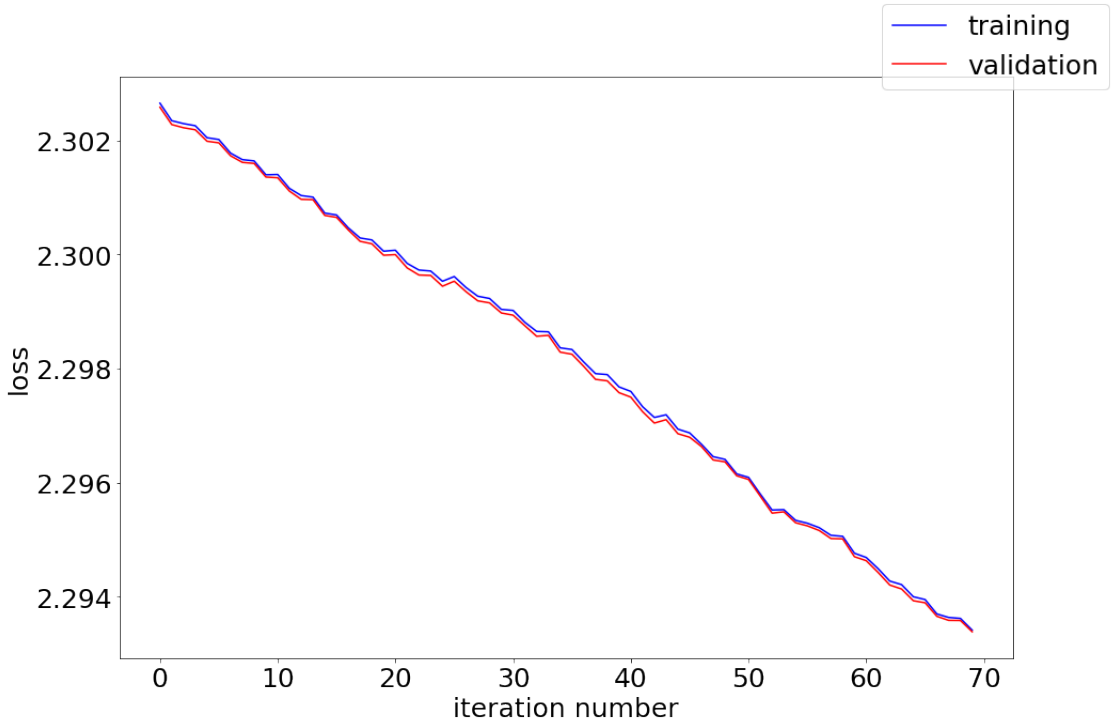
[EXECUTION: learning_rate = 0.05 (with momentum_multiplier = 0.9 )]

```
Now testing the gradient on the whole training set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).

After 7 optimization iterations, training data loss is 2.3005679758421396, and
validation data loss is 2.3005343196843526

After 14 optimization iterations, training data loss is 2.294690290846375, and
validation data loss is 2.2947468636705968

After 21 optimization iterations, training data loss is 2.2848258595613196, and
validation data loss is 2.284874532284197

After 28 optimization iterations, training data loss is 2.2727373216064093, and
validation data loss is 2.2726641075461638

After 35 optimization iterations, training data loss is 2.25555407462529, and
validation data loss is 2.2554664120483583

After 42 optimization iterations, training data loss is 2.227435187398898, and
validation data loss is 2.2273050892212876

After 49 optimization iterations, training data loss is 2.183651335804252, and
validation data loss is 2.1839613369615654

After 56 optimization iterations, training data loss is 2.123048056013555, and
validation data loss is 2.1239245408160095

After 63 optimization iterations, training data loss is 2.0584067828771593, and
validation data loss is 2.059957243385048

After 70 optimization iterations, training data loss is 1.9940517127340163, and
validation data loss is 1.9965592899672242

Now testing the gradient on just a mini-batch instead of the whole training
set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).
```



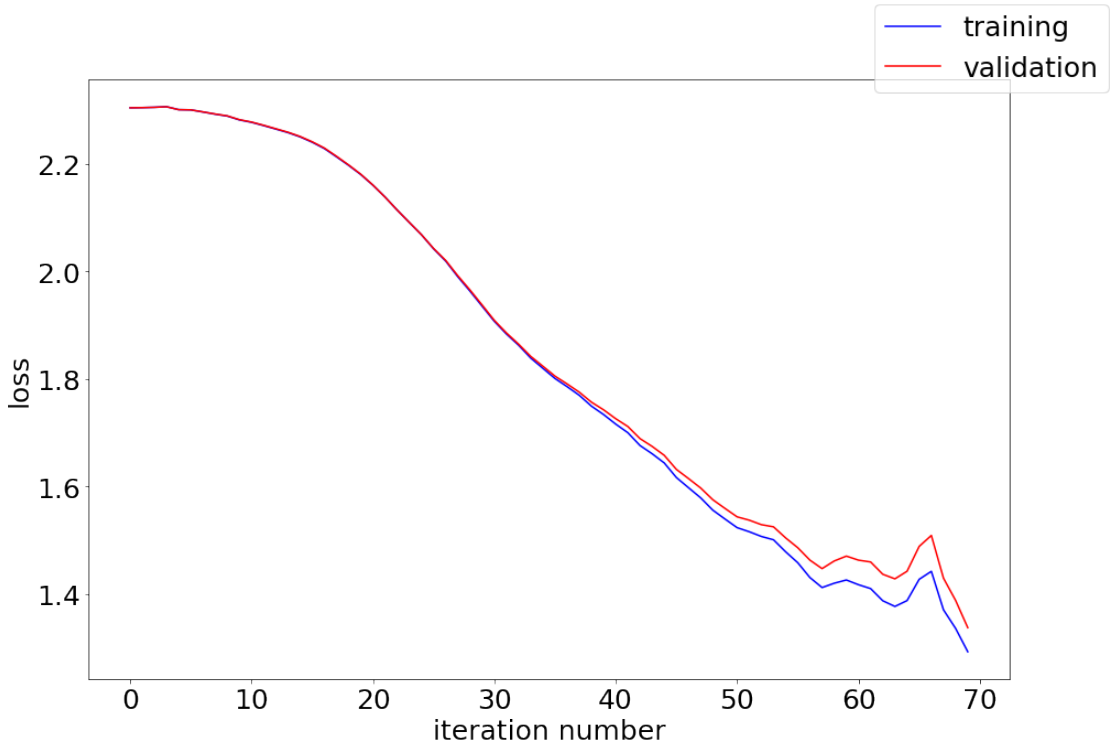

```
The total loss on the training data is 1.9940517127340163

The classification loss (i.e. without weight decay) on the training data is
1.9940517127340163

The classification error rate on the training data is 0.708


The total loss on the validation data is 1.9965592899672242

The classification loss (i.e. without weight decay) on the validation data is
1.9965592899672242

The classification error rate on the validation data is 0.7


The total loss on the test data is 1.9891214271086306

The classification loss (i.e. without weight decay) on the test data is
1.9891214271086306

The classification error rate on the test data is 0.6998888888888889
```

```
[EXECUTION: learning_rate = 0.05 (without momentum_multiplier)]
Now testing the gradient on the whole training set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).

After 7 optimization iterations, training data loss is 2.3017738673111223, and
validation data loss is 2.3017267523295453

After 14 optimization iterations, training data loss is 2.301002353575769, and
validation data loss is 2.300957756628203

After 21 optimization iterations, training data loss is 2.3000716047410164, and
validation data loss is 2.299994828079743

After 28 optimization iterations, training data loss is 2.299265340148979, and
validation data loss is 2.299184558841718

After 35 optimization iterations, training data loss is 2.298360684957857, and
validation data loss is 2.2982852385950183

After 42 optimization iterations, training data loss is 2.2973286018142134, and
validation data loss is 2.2972416728022447

After 49 optimization iterations, training data loss is 2.296406009560148, and
validation data loss is 2.296362261705057

After 56 optimization iterations, training data loss is 2.2952863962740557, and
validation data loss is 2.2952398915093526

After 63 optimization iterations, training data loss is 2.2942723433523486, and
validation data loss is 2.29420266941243

After 70 optimization iterations, training data loss is 2.2934118640834638, and
validation data loss is 2.2933835722400375

Now testing the gradient on just a mini-batch instead of the whole training
set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).
```



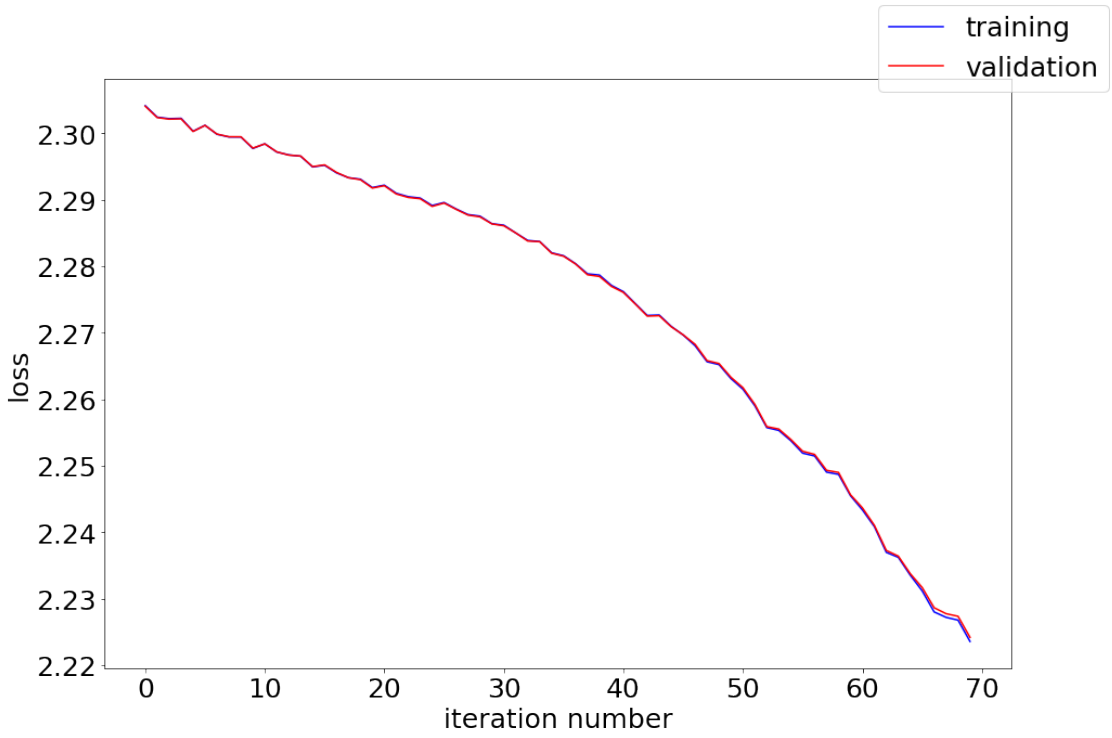

```
The total loss on the training data is 2.2934118640834638

The classification loss (i.e. without weight decay) on the training data is
2.2934118640834638

The classification error rate on the training data is 0.765


The total loss on the validation data is 2.2933835722400375

The classification loss (i.e. without weight decay) on the validation data is
2.2933835722400375

The classification error rate on the validation data is 0.766


The total loss on the test data is 2.2933294102198447

The classification loss (i.e. without weight decay) on the test data is
2.2933294102198447

The classification error rate on the test data is 0.7576666666666667
```

```
[EXECUTION: learning_rate = 0.2 (with momentum_multiplier = 0.9 )]
Now testing the gradient on the whole training set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).

After 7 optimization iterations, training data loss is 2.2962604475303907, and
validation data loss is 2.296427744922369

After 14 optimization iterations, training data loss is 2.2579364788991034, and
validation data loss is 2.2586294562618088

After 21 optimization iterations, training data loss is 2.1601215746963978, and
validation data loss is 2.1604465477872683

After 28 optimization iterations, training data loss is 1.9896511913758645, and
validation data loss is 1.9915902358021196

After 35 optimization iterations, training data loss is 1.8196056237510985, and
validation data loss is 1.8235035235068375

After 42 optimization iterations, training data loss is 1.6998341654758864, and
validation data loss is 1.7113373685413482

After 49 optimization iterations, training data loss is 1.5557933603105996, and
validation data loss is 1.5751828655649058

After 56 optimization iterations, training data loss is 1.4579829078523043, and
validation data loss is 1.485891949135234

After 63 optimization iterations, training data loss is 1.3872728587256975, and
validation data loss is 1.4365642633107423

After 70 optimization iterations, training data loss is 1.292495578954033, and
validation data loss is 1.3372647971195857

Now testing the gradient on just a mini-batch instead of the whole training
set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).
```




```
The total loss on the training data is 1.292495578954033

The classification loss (i.e. without weight decay) on the training data is
1.292495578954033

The classification error rate on the training data is 0.409


The total loss on the validation data is 1.3372647971195857

The classification loss (i.e. without weight decay) on the validation data is
1.3372647971195857

The classification error rate on the validation data is 0.423


The total loss on the test data is 1.3015061753787025

The classification loss (i.e. without weight decay) on the test data is
1.3015061753787025

The classification error rate on the test data is 0.407
```

```
[EXECUTION: learning_rate = 0.2 (without momentum_multiplier)]
Now testing the gradient on the whole training set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).

After 7 optimization iterations, training data loss is 2.2998186272412102, and
validation data loss is 2.299845785723998

After 14 optimization iterations, training data loss is 2.296523212630369, and
validation data loss is 2.296548875548235

After 21 optimization iterations, training data loss is 2.292173208378639, and
validation data loss is 2.2920876391449765

After 28 optimization iterations, training data loss is 2.287752802332703, and
validation data loss is 2.2876742660413685

After 35 optimization iterations, training data loss is 2.2820078100912022, and
validation data loss is 2.2819382701810915

After 42 optimization iterations, training data loss is 2.274370686336385, and
validation data loss is 2.2742967218075636

After 49 optimization iterations, training data loss is 2.265194570580782, and
validation data loss is 2.2653619324217233

After 56 optimization iterations, training data loss is 2.2518845971525456, and
validation data loss is 2.2521737288357193

After 63 optimization iterations, training data loss is 2.236970156868212, and
validation data loss is 2.237258560331139

After 70 optimization iterations, training data loss is 2.223581358966002, and
validation data loss is 2.224201123725473

Now testing the gradient on just a mini-batch instead of the whole training
set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).
```




The total loss on the training data is 2.223581358966002

The classification loss (i.e. without weight decay) on the training data is 2.223581358966002

The classification error rate on the training data is 0.801

The total loss on the validation data is 2.224201123725473

The classification loss (i.e. without weight decay) on the validation data is 2.224201123725473

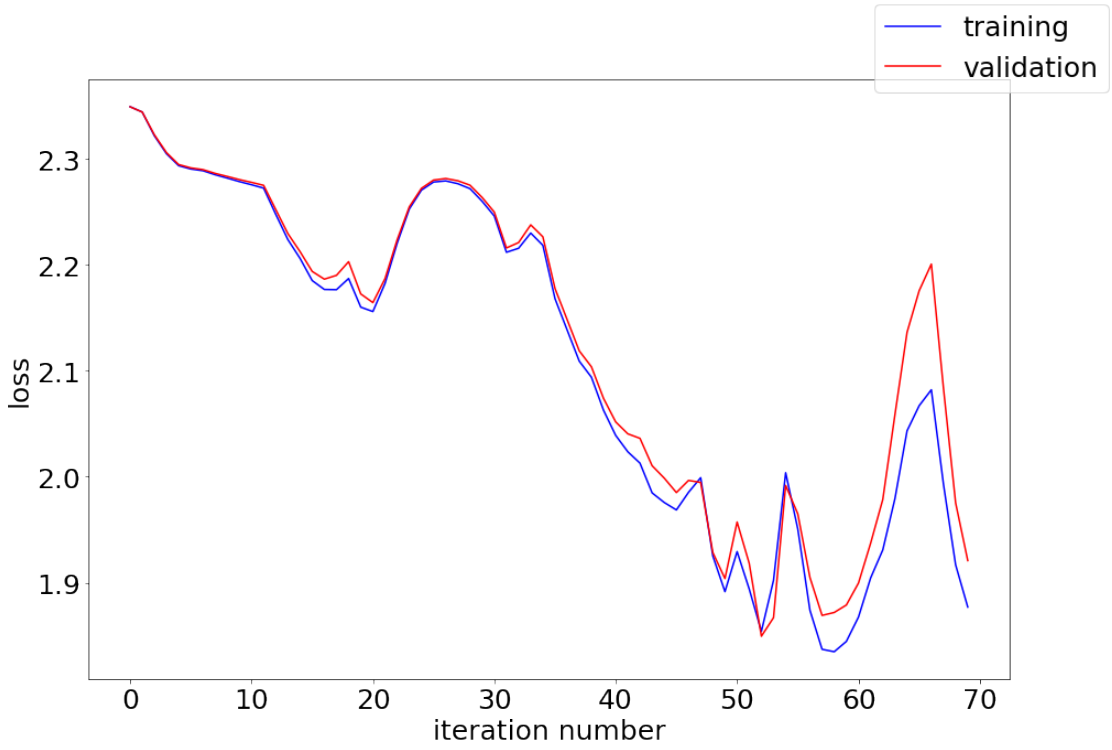
The classification error rate on the validation data is 0.797

The total loss on the test data is 2.222849187609189

The classification loss (i.e. without weight decay) on the test data is 2.222849187609189

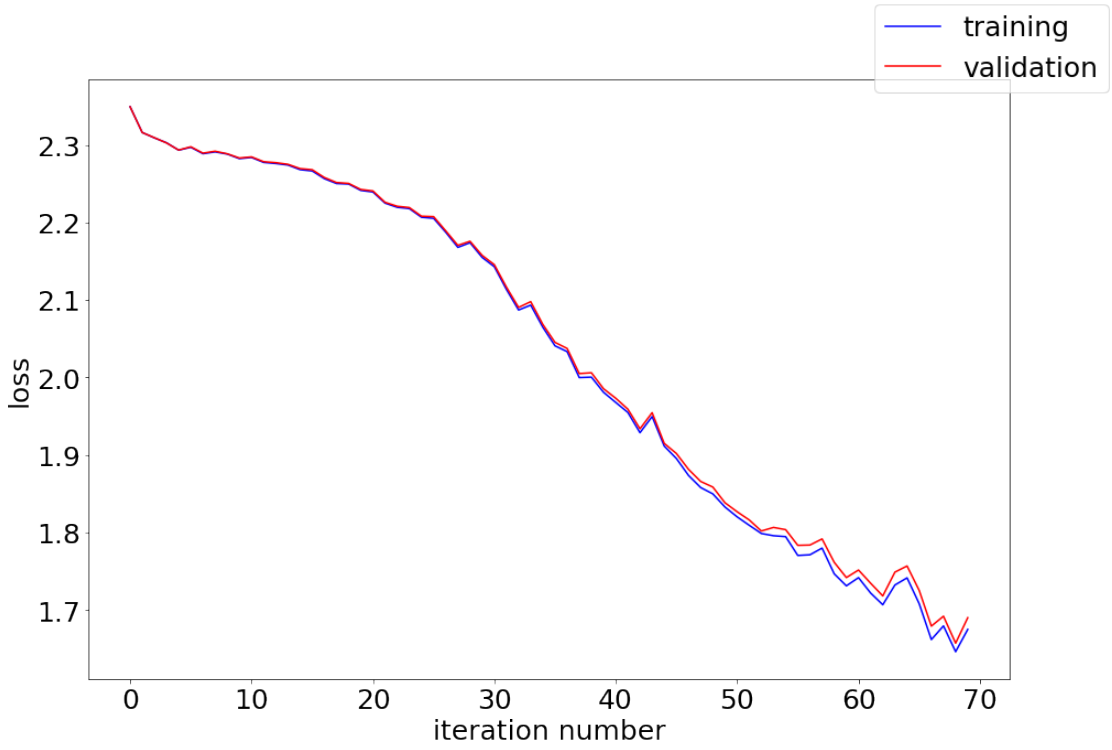
The classification error rate on the test data is 0.7956666666666666

```
[EXECUTION: learning_rate = 1.0 (with momentum_multiplier = 0.9 )]
Now testing the gradient on the whole training set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).

After 7 optimization iterations, training data loss is 2.2888414340263377, and
validation data loss is 2.2899073907965075

After 14 optimization iterations, training data loss is 2.223790743518183, and
validation data loss is 2.229455494094939

After 21 optimization iterations, training data loss is 2.1559251945785114, and
validation data loss is 2.164376997320488

After 28 optimization iterations, training data loss is 2.2765877201020124, and
validation data loss is 2.2793687668534575

After 35 optimization iterations, training data loss is 2.2184360338093594, and
validation data loss is 2.226489978348415

After 42 optimization iterations, training data loss is 2.0235643717010214, and
validation data loss is 2.0405350308902075

After 49 optimization iterations, training data loss is 1.9252529625878758, and
validation data loss is 1.9285357662948723

After 56 optimization iterations, training data loss is 1.9506926312533905, and
validation data loss is 1.9650248878282632

After 63 optimization iterations, training data loss is 1.9309661048698727, and
validation data loss is 1.9785417420801112

After 70 optimization iterations, training data loss is 1.8770514360278114, and
validation data loss is 1.9209129479145275

Now testing the gradient on just a mini-batch instead of the whole training
set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).
```



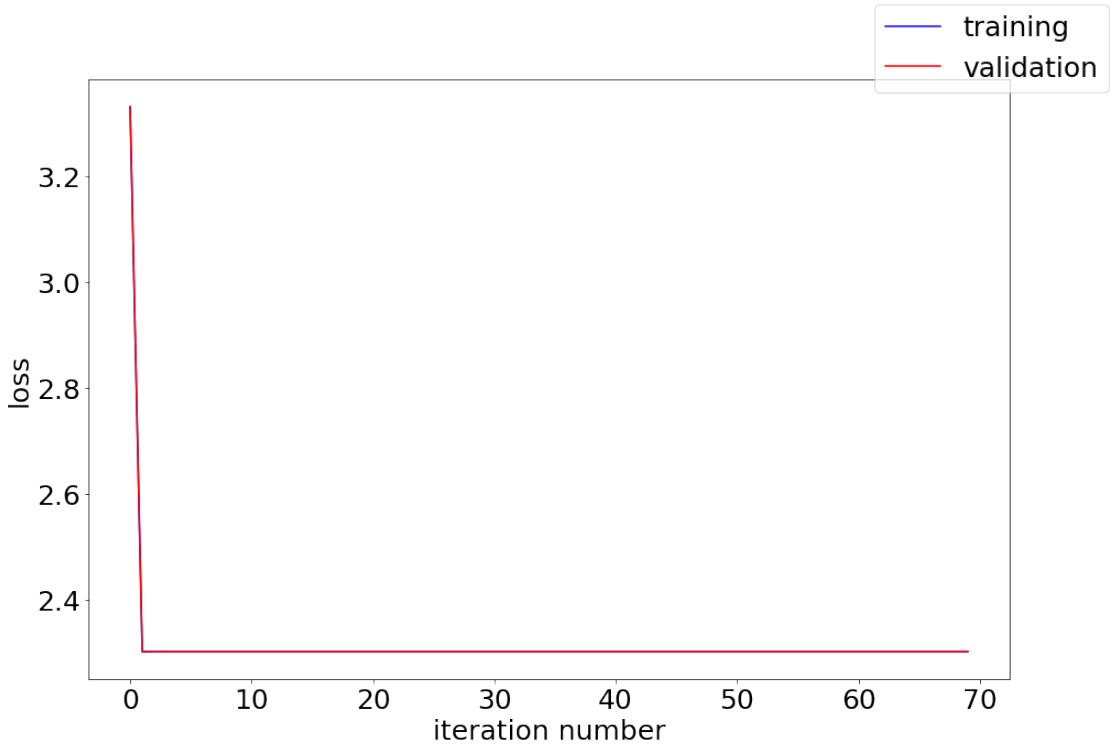

```
The total loss on the training data is 1.8770514360278114

The classification loss (i.e. without weight decay) on the training data is
1.8770514360278114

The classification error rate on the training data is 0.721


The total loss on the validation data is 1.9209129479145275

The classification loss (i.e. without weight decay) on the validation data is
1.9209129479145275

The classification error rate on the validation data is 0.737


The total loss on the test data is 1.8974511412915498

The classification loss (i.e. without weight decay) on the test data is
1.8974511412915498

The classification error rate on the test data is 0.7242222222222222
```

```
[EXECUTION: learning_rate = 1.0 (without momentum_multiplier)]
Now testing the gradient on the whole training set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).

After 7 optimization iterations, training data loss is 2.288632771565589, and
validation data loss is 2.2894001227767964

After 14 optimization iterations, training data loss is 2.2740065933806797, and
validation data loss is 2.2749978891062344

After 21 optimization iterations, training data loss is 2.239245624394231, and
validation data loss is 2.2406088548602

After 28 optimization iterations, training data loss is 2.1677529659752914, and
validation data loss is 2.1703832013196314

After 35 optimization iterations, training data loss is 2.064690677036345, and
validation data loss is 2.0683542079341324

After 42 optimization iterations, training data loss is 1.9548602774911705, and
validation data loss is 1.9591103260327536

After 49 optimization iterations, training data loss is 1.8497568912452225, and
validation data loss is 1.858573464810318

After 56 optimization iterations, training data loss is 1.7703130911939102, and
validation data loss is 1.7833624668563843

After 63 optimization iterations, training data loss is 1.706710503793043, and
validation data loss is 1.7180966243310214

After 70 optimization iterations, training data loss is 1.6749332144612648, and
validation data loss is 1.6900252783448697

Now testing the gradient on just a mini-batch instead of the whole training
set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).
```



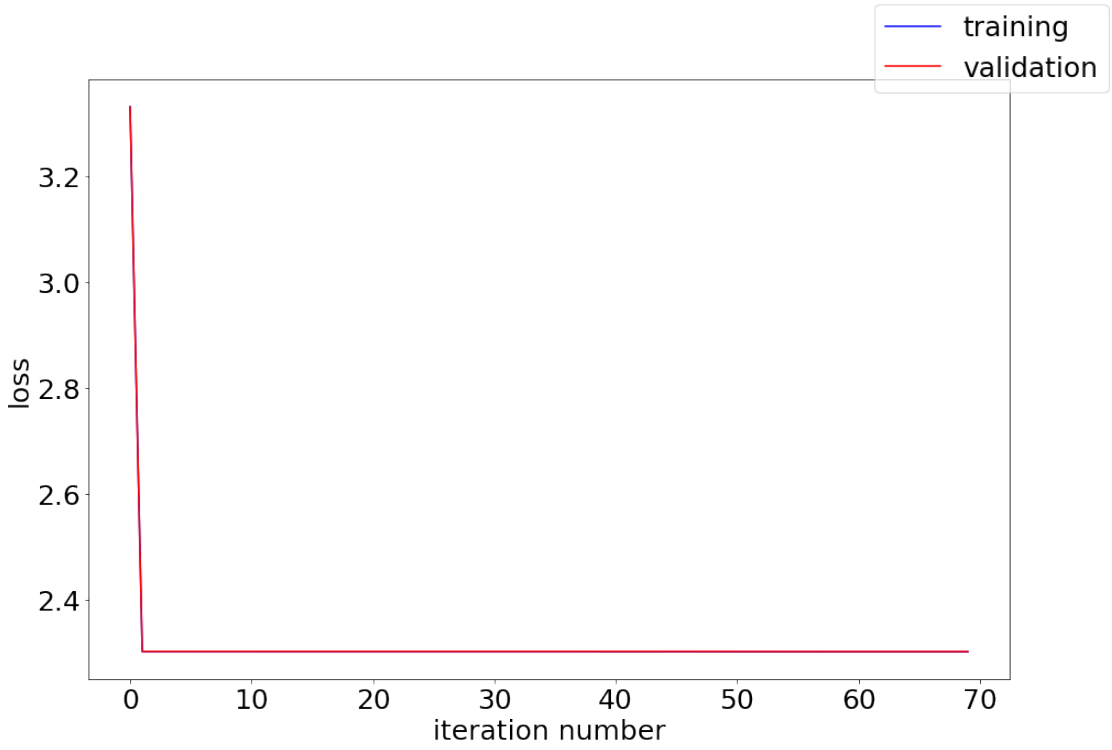

```
The total loss on the training data is 1.6749332144612648

The classification loss (i.e. without weight decay) on the training data is
1.6749332144612648

The classification error rate on the training data is 0.697


The total loss on the validation data is 1.6900252783448697

The classification loss (i.e. without weight decay) on the validation data is
1.6900252783448697

The classification error rate on the validation data is 0.714


The total loss on the test data is 1.676162605073363

The classification loss (i.e. without weight decay) on the test data is
1.676162605073363

The classification error rate on the test data is 0.7032222222222222
```

```
[EXECUTION: learning_rate = 5.0 (with momentum_multiplier = 0.9 )]
Now testing the gradient on the whole training set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).

After 7 optimization iterations, training data loss is 2.3025850783281085, and
validation data loss is 2.302585023871894

After 14 optimization iterations, training data loss is 2.302585092986651, and
validation data loss is 2.3025850929239517

After 21 optimization iterations, training data loss is 2.3025850929938483, and
validation data loss is 2.3025850929915532

After 28 optimization iterations, training data loss is 2.302585092994011, and
validation data loss is 2.302585092993543

After 35 optimization iterations, training data loss is 2.302585092994031, and
validation data loss is 2.3025850929938123

After 42 optimization iterations, training data loss is 2.302585092994036, and
validation data loss is 2.302585092993884

After 49 optimization iterations, training data loss is 2.3025850929940375, and
validation data loss is 2.30258509299391

After 56 optimization iterations, training data loss is 2.302585092994038, and
validation data loss is 2.3025850929939207

After 63 optimization iterations, training data loss is 2.302585092994039, and
validation data loss is 2.302585092993926

After 70 optimization iterations, training data loss is 2.302585092994039, and
validation data loss is 2.3025850929939278

Now testing the gradient on just a mini-batch instead of the whole training
set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).
```



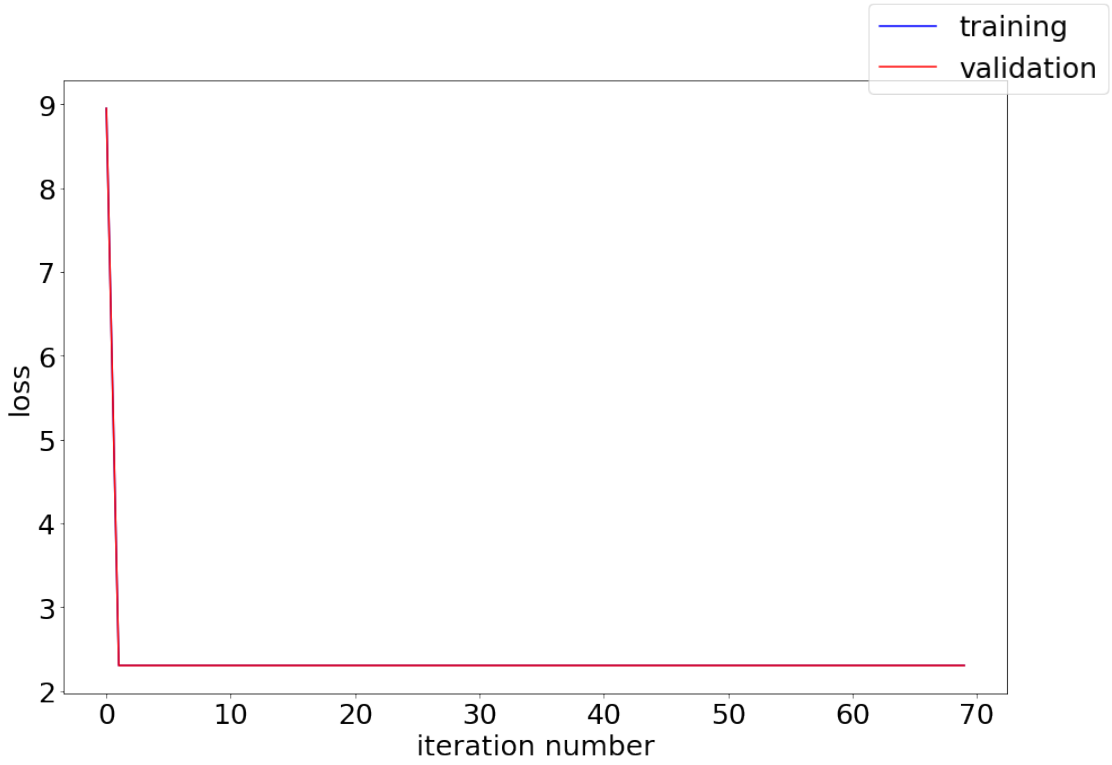

```
The total loss on the training data is 2.302585092994039

The classification loss (i.e. without weight decay) on the training data is
2.302585092994039

The classification error rate on the training data is 0.886


The total loss on the validation data is 2.3025850929939278

The classification loss (i.e. without weight decay) on the validation data is
2.3025850929939278

The classification error rate on the validation data is 0.897


The total loss on the test data is 2.3025850919707307

The classification loss (i.e. without weight decay) on the test data is
2.3025850919707307

The classification error rate on the test data is 0.8877777777777778
```

```
[EXECUTION: learning_rate = 5.0 (without momentum_multiplier)]
Now testing the gradient on the whole training set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).

After 7 optimization iterations, training data loss is 2.302067196784302, and
validation data loss is 2.302340910584947

After 14 optimization iterations, training data loss is 2.3020191850189793, and
validation data loss is 2.3023001127502645

After 21 optimization iterations, training data loss is 2.302014482280857, and
validation data loss is 2.302296675017002

After 28 optimization iterations, training data loss is 2.3019893530648567, and
validation data loss is 2.3022820933145955

After 35 optimization iterations, training data loss is 2.301980512310046, and
validation data loss is 2.302281410337269

After 42 optimization iterations, training data loss is 2.301902179137929, and
validation data loss is 2.302240807694258

After 49 optimization iterations, training data loss is 2.3018942353043887, and
validation data loss is 2.3022192579508376

After 56 optimization iterations, training data loss is 2.301630568304788, and
validation data loss is 2.302088602304471

After 63 optimization iterations, training data loss is 2.3016214258765095, and
validation data loss is 2.3020443490933222

After 70 optimization iterations, training data loss is 2.3016188038504537, and
validation data loss is 2.3020434221578814

Now testing the gradient on just a mini-batch instead of the whole training
set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).
```



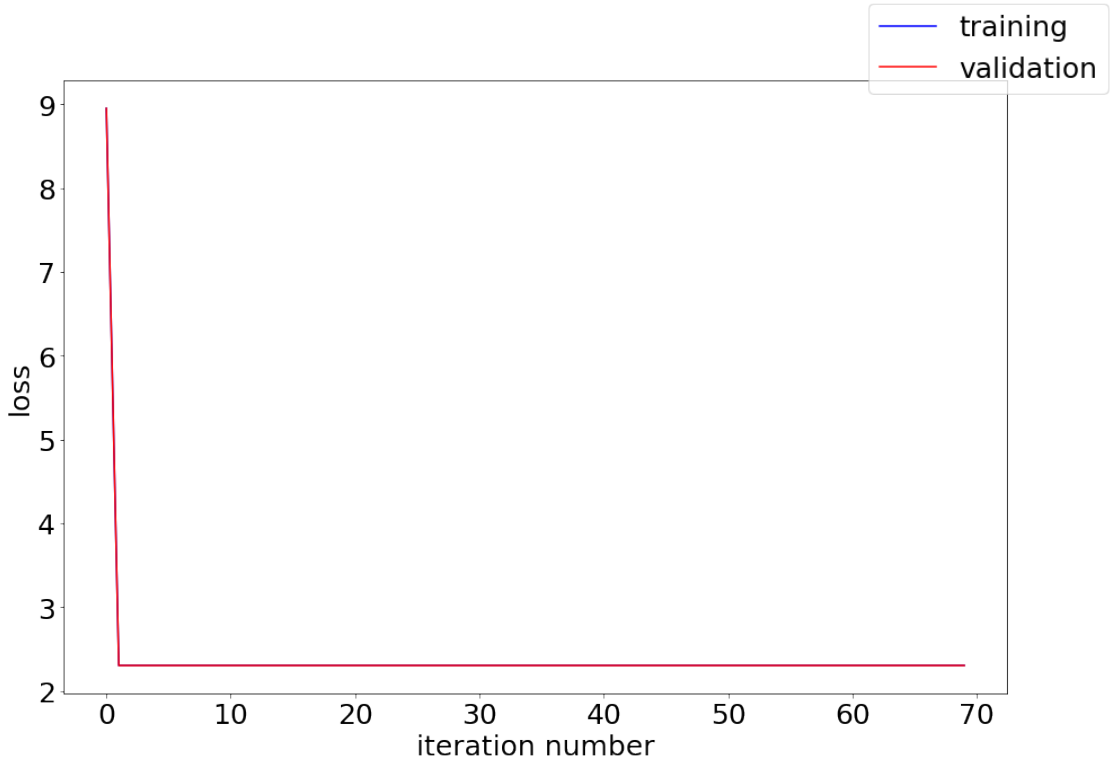

```
The total loss on the training data is 2.3016188038504537

The classification loss (i.e. without weight decay) on the training data is
2.3016188038504537

The classification error rate on the training data is 0.899


The total loss on the validation data is 2.3020434221578814

The classification loss (i.e. without weight decay) on the validation data is
2.3020434221578814

The classification error rate on the validation data is 0.9


The total loss on the test data is 2.302442636292494

The classification loss (i.e. without weight decay) on the test data is
2.302442636292494

The classification error rate on the test data is 0.8993333333333333
```

```
[EXECUTION: learning_rate = 20.0 (with momentum_multiplier = 0.9 )]
Now testing the gradient on the whole training set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).

After 7 optimization iterations, training data loss is 2.3025850929940463, and
validation data loss is 2.3025850929940463

After 14 optimization iterations, training data loss is 2.3025850929940463, and
validation data loss is 2.3025850929940463

After 21 optimization iterations, training data loss is 2.3025850929940463, and
validation data loss is 2.3025850929940463

After 28 optimization iterations, training data loss is 2.3025850929940463, and
validation data loss is 2.3025850929940463

After 35 optimization iterations, training data loss is 2.3025850929940463, and
validation data loss is 2.3025850929940463

After 42 optimization iterations, training data loss is 2.3025850929940463, and
validation data loss is 2.3025850929940463

After 49 optimization iterations, training data loss is 2.3025850929940463, and
validation data loss is 2.3025850929940463

After 56 optimization iterations, training data loss is 2.3025850929940463, and
validation data loss is 2.3025850929940463

After 63 optimization iterations, training data loss is 2.3025850929940463, and
validation data loss is 2.3025850929940463

After 70 optimization iterations, training data loss is 2.3025850929940463, and
validation data loss is 2.3025850929940463

Now testing the gradient on just a mini-batch instead of the whole training
set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).
```




```
The total loss on the training data is 2.3025850929940463

The classification loss (i.e. without weight decay) on the training data is
2.3025850929940463

The classification error rate on the training data is 0.874


The total loss on the validation data is 2.3025850929940463

The classification loss (i.e. without weight decay) on the validation data is
2.3025850929940463

The classification error rate on the validation data is 0.882


The total loss on the test data is 2.302585092994046

The classification loss (i.e. without weight decay) on the test data is
2.302585092994046

The classification error rate on the test data is 0.8778888888888889
```

```
[EXECUTION: learning_rate = 20.0 (without momentum_multiplier)]
Now testing the gradient on the whole training set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).

After 7 optimization iterations, training data loss is 2.302585091837178, and
validation data loss is 2.3025850910767875

After 14 optimization iterations, training data loss is 2.302585091837178, and
validation data loss is 2.3025850910767875

After 21 optimization iterations, training data loss is 2.302585091837178, and
validation data loss is 2.3025850910767875

After 28 optimization iterations, training data loss is 2.302585091837178, and
validation data loss is 2.3025850910767875

After 35 optimization iterations, training data loss is 2.302585091837178, and
validation data loss is 2.3025850910767875

After 42 optimization iterations, training data loss is 2.302585091837178, and
validation data loss is 2.3025850910767875

After 49 optimization iterations, training data loss is 2.302585091837178, and
validation data loss is 2.3025850910767875

After 56 optimization iterations, training data loss is 2.3025850918371775, and
validation data loss is 2.302585091076785

After 63 optimization iterations, training data loss is 2.3025850918371775, and
validation data loss is 2.302585091076785

After 70 optimization iterations, training data loss is 2.3025850918371775, and
validation data loss is 2.302585091076785

Now testing the gradient on just a mini-batch instead of the whole training
set…
Gradient test passed for hid_to_class.
Gradient test passed. That means that the gradient that your code computed is
within 0.001%% of the gradient that the finite difference approximation
computed, so the gradient calculation procedure is probably correct (not
certainly, but probably).
```




```
The total loss on the training data is 2.3025850918371775

The classification loss (i.e. without weight decay) on the training data is
2.3025850918371775

The classification error rate on the training data is 0.9


The total loss on the validation data is 2.302585091076785

The classification loss (i.e. without weight decay) on the validation data is
2.302585091076785

The classification error rate on the validation data is 0.9


The total loss on the test data is 2.3025850220361597

The classification loss (i.e. without weight decay) on the test data is
2.3025850220361597

The classification error rate on the test data is 0.9
```