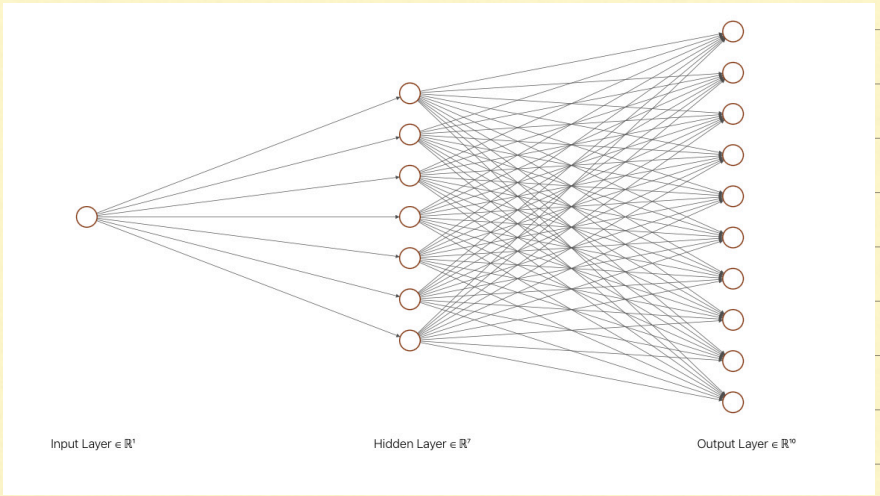
Gengcong Yan
1009903



Schematic network

1: m=1

$$E = -t \cdot \log y = -t \cdot \log(G(Wx)) = -t \cdot \log\left(\frac{1}{1+e^{-Wx}}\right)$$

$$\therefore \frac{\partial E}{\partial t} = \log\left(\frac{1}{1+e^{-Wx}}\right) \qquad \frac{\partial E}{\partial x} = -t \cdot \frac{1}{G(Wx)} \cdot \frac{\partial G(Wx)}{\partial x}$$

$$= -t \cdot (1+e^{-Wx}) \cdot W \frac{1}{1+e^{-Wx}}\left(1-\frac{1}{1+e^{-Wx}}\right)$$

$$= -tW \frac{e^{-Wx}}{1+e^{-Wx}}$$

2: There are only 2 layers in network, we can know:

$z^{(1)} = W^{(1)}x \qquad y^{(1)} = G(z^{(1)})$,

$z^{(2)} = W^{(2)}y^{(1)} \qquad y^{(2)} = S(z^{(2)})$,

$y = y^{(2)}$

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

3: From (2) we can know: $\frac{\partial E}{\partial z^{(2)}} = (y^{(2)} - t)^T$

$\because z^{(2)} = W^{(2)} y^{(1)} \Rightarrow \frac{\partial z^{(2)}}{\partial y^{(1)}} = W^{(2)}$

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

4: $\frac{\partial E}{\partial W_{uv}^{(2)}} = \frac{\partial E}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial W_{uv}^{(2)}} = (y_u^{(2)} - t_u)^T \cdot y_v^{(1)}$

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

5:

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

$$= \sigma(z)(1 - \sigma(z))$$

$$\therefore \frac{\partial y^{(1)}}{\partial z^{(1)}} = \frac{\partial \sigma(z^{(1)})}{\partial z^{(1)}} = \sigma(z^{(1)}) \cdot (1 - \sigma(z^{(1)})) = y^{(1)} \cdot (1 - y^{(1)})$$

$$= \mathrm{diag}(y^{(1)} \cdot (1 - y^{(1)}))$$

6:

From the computation in (3) and (5), we can get:

$$\frac{\partial E}{\partial z^{(1)}} = \frac{\partial E}{\partial y^{(1)}} \cdot \frac{\partial y^{(1)}}{\partial z^{(1)}} = (y^{(2)} - t)^T W^{(2)} \mathrm{diag}(y^{(1)} * (1 - y^{(1)}))$$

7:

$$\frac{\partial E}{\partial W^{(l)}} = \frac{\partial E}{\partial Z^{(l)}} \cdot \frac{\partial Z^{(l)}}{\partial W^{(l)}} = \frac{\partial E}{\partial Z^{(l)}} \cdot X$$

$$\frac{\partial E}{\partial W_{uv}^{(l)}} = \frac{\partial E}{\partial Z_u^{(l)}} \cdot \frac{\partial Z_u^{(l)}}{\partial W_{uv}^{(l)}} = \frac{\partial E}{\partial Z_u^{(l)}} \cdot \frac{\partial (\sum_i W_{ui} X_i)}{\partial W_{uv}^{(l)}} = \frac{\partial E}{\partial Z_u^{(l)}} \cdot X_v$$

ExerciseRound09

November 16, 2021

[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
```

[32]:
```
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>
   ####target or taegets?????
    # 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--#

    # we need to compute the gradients in (4) and (7)  in exercise 1

    nums=len(data['inputs'][0])

    w1= model['input_to_hid']
    w2= model['hid_to_class']

    z1=np.dot(w1,data["inputs"])
    y1=logistic(z1)

    z2=np.dot(w2,y1)

    #softmax
    class_normalizer = log_sum_exp_over_rows(z2) # 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 = z2 - np.tile(class_normalizer, (z2.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>

    y2= class_prob
    t=data["target"]

    ret['hid_to_class']= np.dot((y2-t), y1.T)/nums

    part1=np.dot((y2-t).T , w2)

    for i in range(part1.shape[0]):
        row=np.dot(part1[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.dot(dEdz1.T , data["inputs"].T) / nums;

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

[6]:
```
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
```

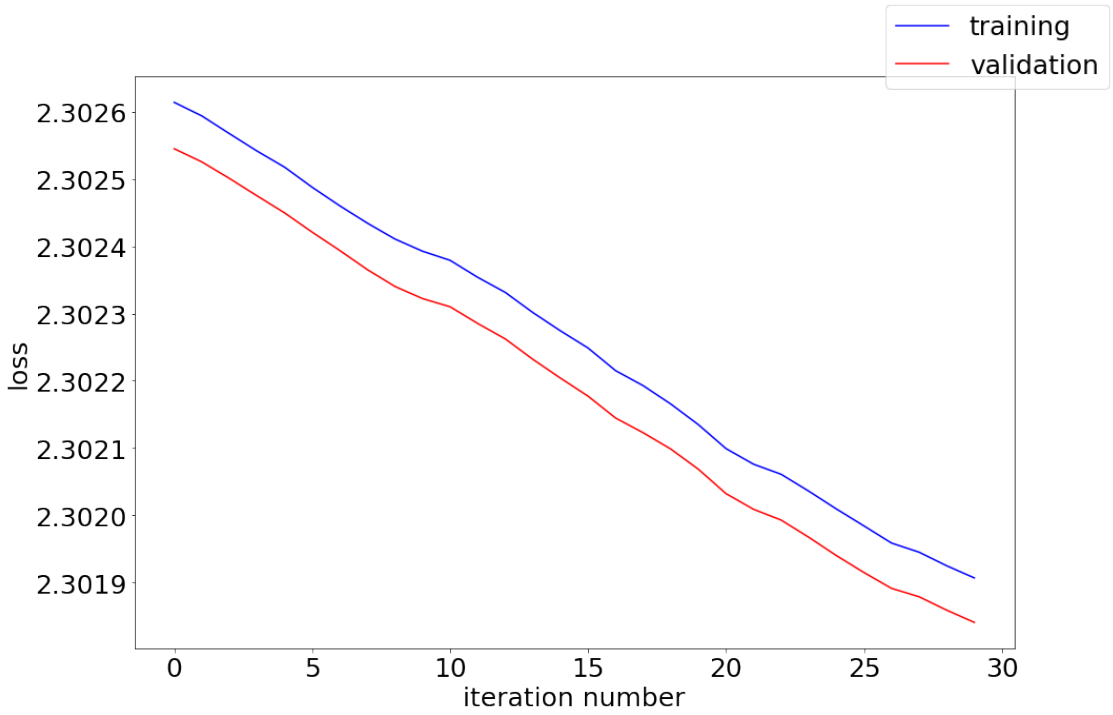
```
        best_so_far['validation_loss'] = np.Inf
        best_so_far['after_n_iters'] = -1

    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)))
```

[34]:
```
# 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: 2.301906765216956

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/

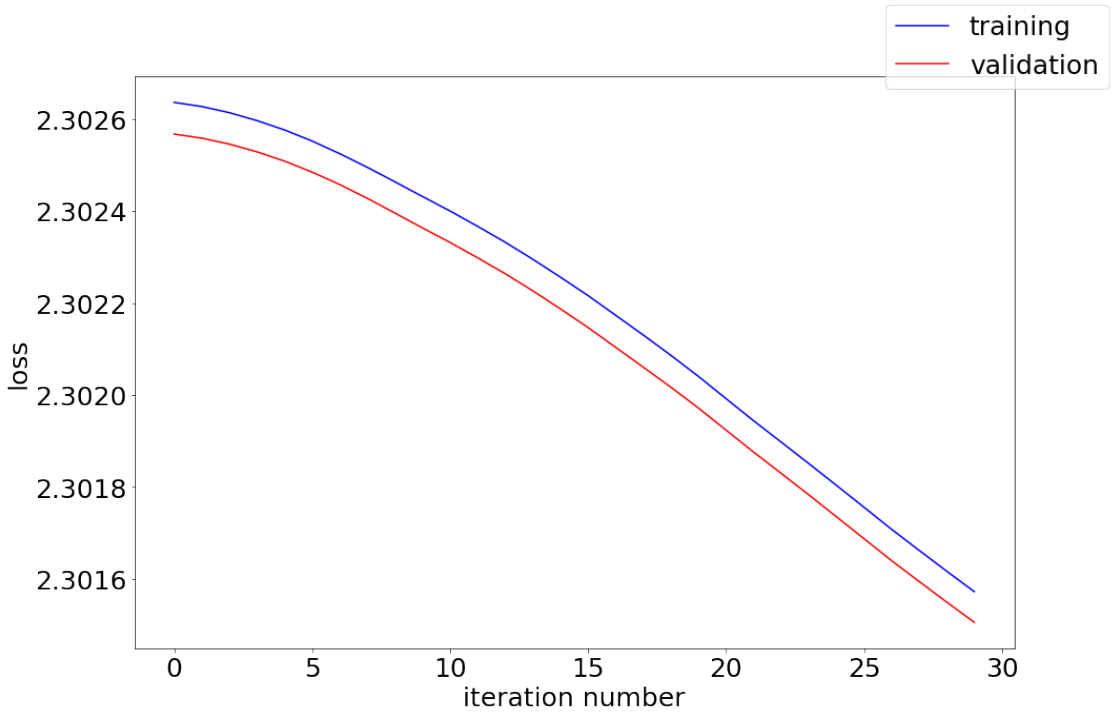
The task is to experiment with the example code given above and report your findings. There is no need to program anything in this part but completing it requires that Part 2 is successfully solved.

[37]:
```
#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_rate = [0.002, 0.01, 0.05, 0.2, 1.0, 5.0,20.0]

for lr in learning_rate:
    print(f"This round: learning rate={lr} , with momentum_multiplier=0.9")
    a2(0, 10, 30, lr, 0.9, False, 10)


for lr in learning_rate:
```

```
    print(f"This round: learning rate={lr} , without momentum_multiplier")
    a2(0, 10, 30, lr, 0, False, 10)
```

```
This round: 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 3 optimization iterations, training data loss is 2.3026136468874983, and
validation data loss is 2.302545302729936

After 6 optimization iterations, training data loss is 2.302551879810409, and
validation data loss is 2.302484355683604

After 9 optimization iterations, training data loss is 2.3024632914890106, and
validation data loss is 2.3023954195680694

After 12 optimization iterations, training data loss is 2.302366432450772, and
validation data loss is 2.3022980645303726

After 15 optimization iterations, training data loss is 2.302256013736331, and
validation data loss is 2.302187070301092

After 18 optimization iterations, training data loss is 2.302130290435607, and
validation data loss is 2.302060429324925

After 21 optimization iterations, training data loss is 2.3019920297971157, and
validation data loss is 2.301923606147474

After 24 optimization iterations, training data loss is 2.3018508437638956, and
validation data loss is 2.3017828115920884

After 27 optimization iterations, training data loss is 2.3017079531935756, and
validation data loss is 2.3016398479106055

After 30 optimization iterations, training data loss is 2.301572377661204, and
validation data loss is 2.3015055669676623

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 2.301572377661204

The classification loss (i.e. without weight decay) on the training data is 2.301572377661204

The classification error rate on the training data is 0.883

The total loss on the validation data is 2.3015055669676623

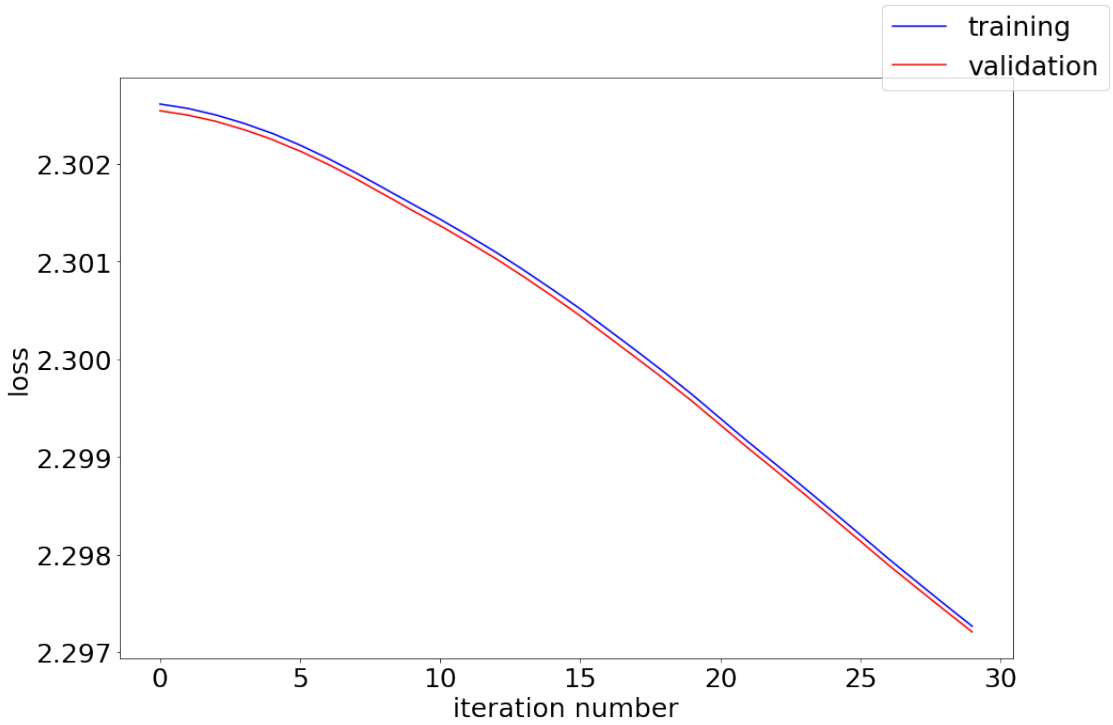
The classification loss (i.e. without weight decay) on the validation data is 2.3015055669676623

The classification error rate on the validation data is 0.89

The total loss on the test data is 2.3015293133436328

```
The classification loss (i.e. without weight decay) on the test data is
2.3015293133436328

The classification error rate on the test data is 0.8807777777777778

This round: 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 3 optimization iterations, training data loss is 2.3025018693168438, and
validation data loss is 2.3024357571061653

After 6 optimization iterations, training data loss is 2.3021932852394476, and
validation data loss is 2.3021312778651177

After 9 optimization iterations, training data loss is 2.301751003154958, and
validation data loss is 2.30168726914556

After 12 optimization iterations, training data loss is 2.3012676188131276, and
validation data loss is 2.3012014231074223

After 15 optimization iterations, training data loss is 2.3007164972983336, and
validation data loss is 2.3006474463535933

After 18 optimization iterations, training data loss is 2.3000883011902715, and
validation data loss is 2.3000146853896766

After 21 optimization iterations, training data loss is 2.299395322309504, and
validation data loss is 2.2993288877846463

After 24 optimization iterations, training data loss is 2.29868425234578, and
validation data loss is 2.2986198079080813

After 27 optimization iterations, training data loss is 2.297958726417039, and
validation data loss is 2.297894023166338

After 30 optimization iterations, training data loss is 2.2972643734895173, and
validation data loss is 2.2972063004250187

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 2.2972643734895173

The classification loss (i.e. without weight decay) on the training data is 2.2972643734895173

The classification error rate on the training data is 0.812

The total loss on the validation data is 2.2972063004250187

The classification loss (i.e. without weight decay) on the validation data is 2.2972063004250187

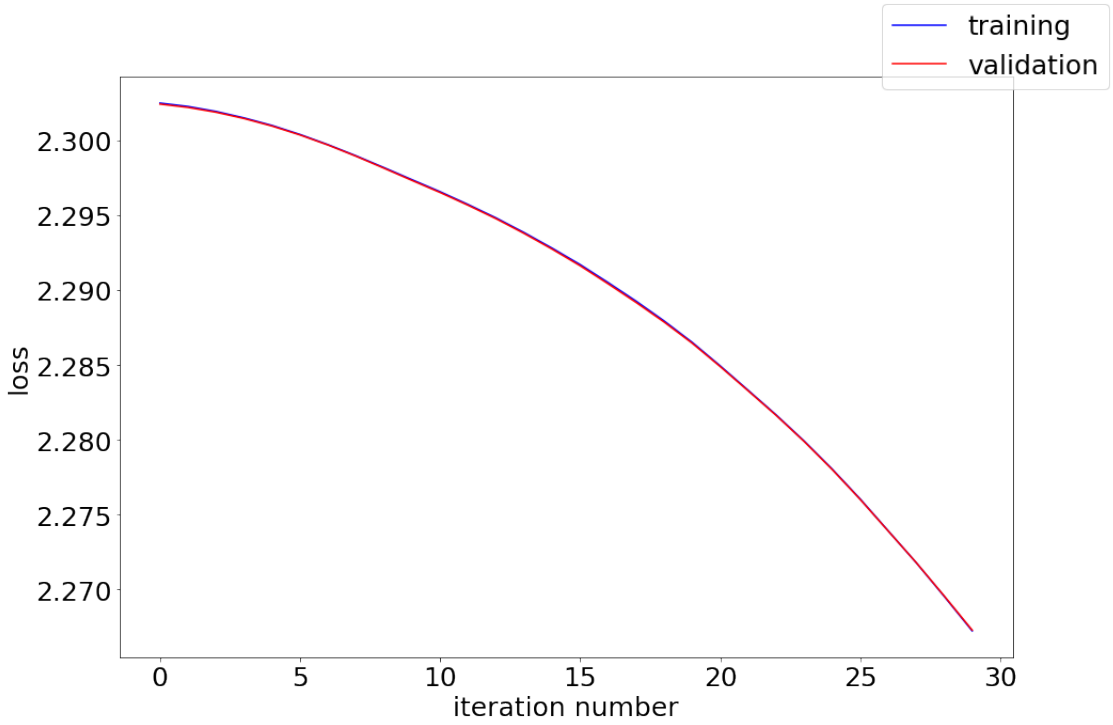
The classification error rate on the validation data is 0.799

The total loss on the test data is 2.297203913600871

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

```
2.297203913600871

The classification error rate on the test data is 0.8

This round: 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 3 optimization iterations, training data loss is 2.3019434603062483, and
validation data loss is 2.301888526104278

After 6 optimization iterations, training data loss is 2.3004042232332496, and
validation data loss is 2.300369959359394

After 9 optimization iterations, training data loss is 2.2981885003312033, and
validation data loss is 2.2981457690627103

After 12 optimization iterations, training data loss is 2.295726208312568, and
validation data loss is 2.2956714521757577

After 15 optimization iterations, training data loss is 2.2928098009364146, and
validation data loss is 2.292740839425913

After 18 optimization iterations, training data loss is 2.289267425378407, and
validation data loss is 2.2891763877689084

After 21 optimization iterations, training data loss is 2.284946978529712, and
validation data loss is 2.2848960001157255

After 24 optimization iterations, training data loss is 2.2799185937923494, and
validation data loss is 2.2798860235860494

After 27 optimization iterations, training data loss is 2.273919308276727, and
validation data loss is 2.2739050779018166

After 30 optimization iterations, training data loss is 2.267227675155303, and
validation data loss is 2.2672794811661507

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 2.267227675155303

The classification loss (i.e. without weight decay) on the training data is 2.267227675155303

The classification error rate on the training data is 0.711

The total loss on the validation data is 2.2672794811661507

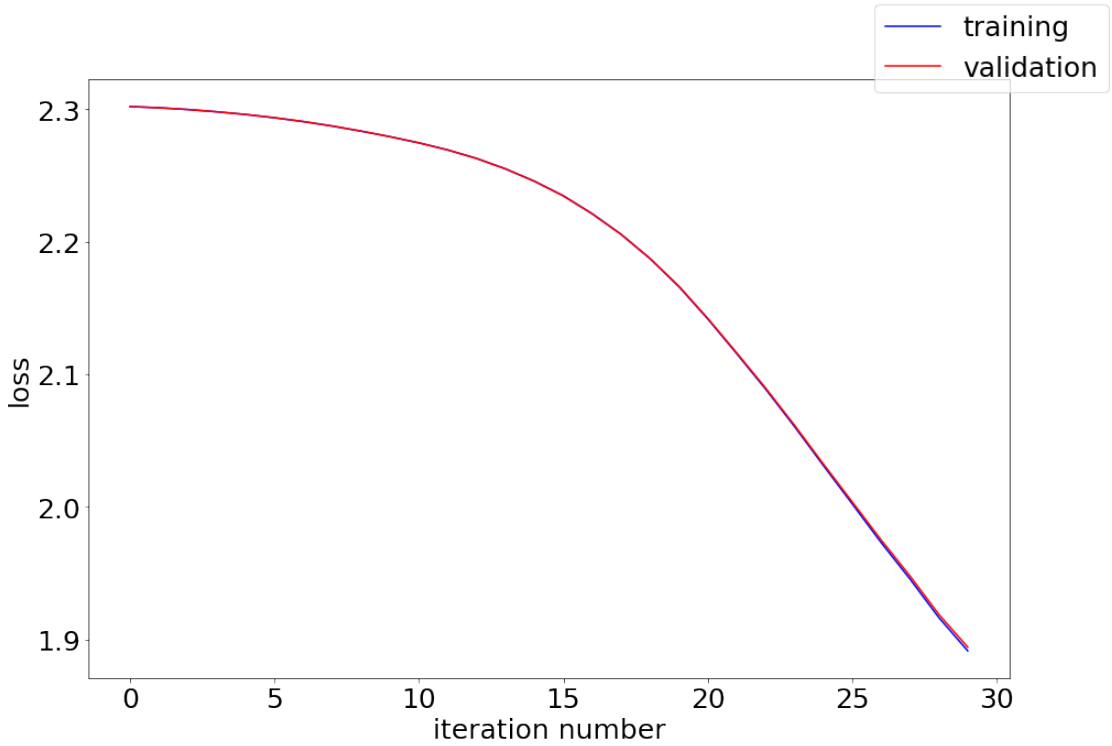
The classification loss (i.e. without weight decay) on the validation data is 2.2672794811661507

The classification error rate on the validation data is 0.713

The total loss on the test data is 2.2669681418821948

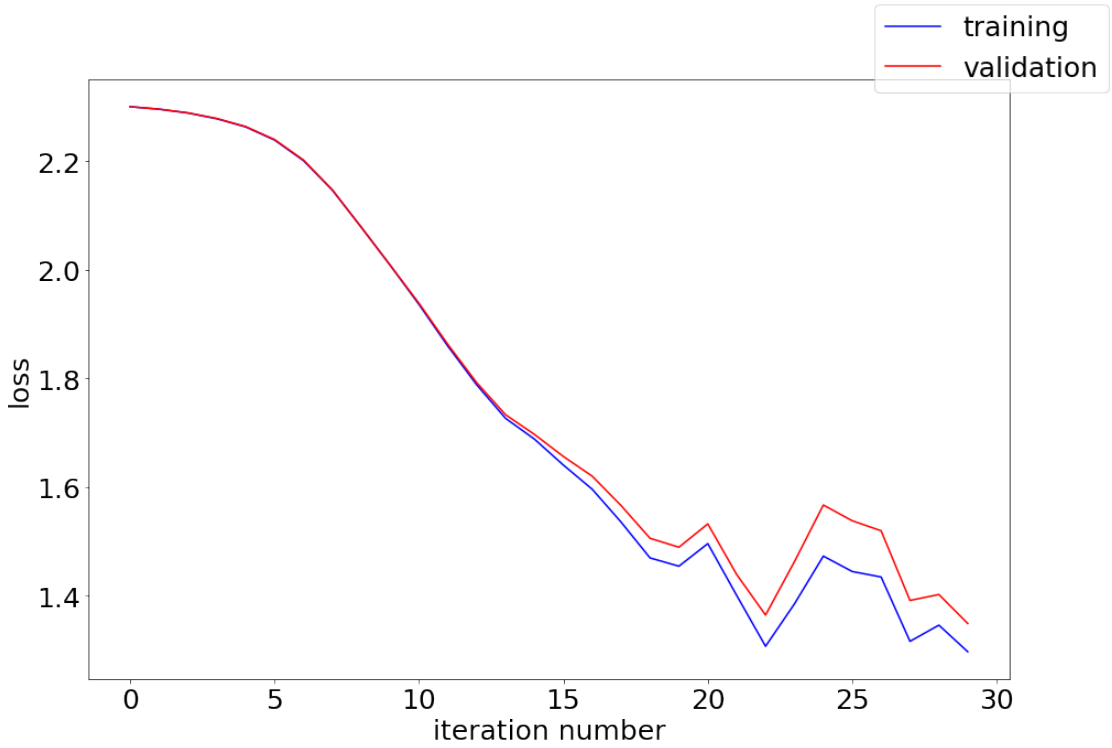
The classification loss (i.e. without weight decay) on the test data is 2.2669681418821948

```
The classification error rate on the test data is 0.7065555555555556

This round: 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 3 optimization iterations, training data loss is 2.299852934579726, and
validation data loss is 2.2998403207588756

After 6 optimization iterations, training data loss is 2.2936023006225863, and
validation data loss is 2.293676247728148

After 9 optimization iterations, training data loss is 2.283520768390833, and
validation data loss is 2.2835548797608363

After 12 optimization iterations, training data loss is 2.2693131305936776, and
validation data loss is 2.2692969015326216

After 15 optimization iterations, training data loss is 2.2457139404359503, and
validation data loss is 2.2456412328537443

After 18 optimization iterations, training data loss is 2.205622627388271, and
validation data loss is 2.2055549873744975

After 21 optimization iterations, training data loss is 2.1417599666439924, and
validation data loss is 2.14217820473946

After 24 optimization iterations, training data loss is 2.060595528355866, and
validation data loss is 2.061567783204283

After 27 optimization iterations, training data loss is 1.9730047463832652, and
validation data loss is 1.9749804215569886

After 30 optimization iterations, training data loss is 1.8912081884000205, and
validation data loss is 1.8940060201343036

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).
```



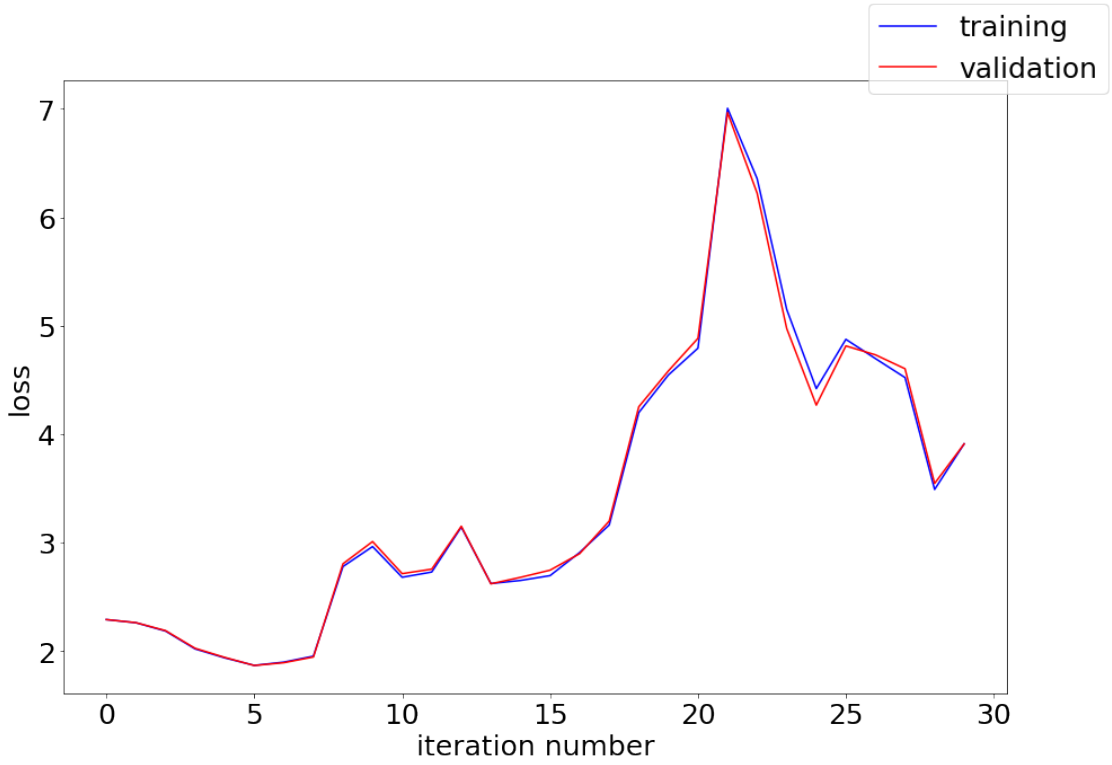

```
The total loss on the training data is 1.8912081884000205

The classification loss (i.e. without weight decay) on the training data is
1.8912081884000205

The classification error rate on the training data is 0.697


The total loss on the validation data is 1.8940060201343036

The classification loss (i.e. without weight decay) on the validation data is
1.8940060201343036

The classification error rate on the validation data is 0.692


The total loss on the test data is 1.8843840244531893

The classification loss (i.e. without weight decay) on the test data is
1.8843840244531893
```

```
The classification error rate on the test data is 0.6914444444444444

This round: 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 3 optimization iterations, training data loss is 2.288344186374662, and
validation data loss is 2.288585312649096

After 6 optimization iterations, training data loss is 2.2386056310145164, and
validation data loss is 2.239536432455113

After 9 optimization iterations, training data loss is 2.0784676950074545, and
validation data loss is 2.078066552559328

After 12 optimization iterations, training data loss is 1.8593415148785692, and
validation data loss is 1.862528556198688

After 15 optimization iterations, training data loss is 1.6881542144915904, and
validation data loss is 1.6968623318980158

After 18 optimization iterations, training data loss is 1.5354482581660787, and
validation data loss is 1.5658330445544706

After 21 optimization iterations, training data loss is 1.4959654343040083, and
validation data loss is 1.53226712577859

After 24 optimization iterations, training data loss is 1.3850558459148055, and
validation data loss is 1.4627980089578552

After 27 optimization iterations, training data loss is 1.4345331680943605, and
validation data loss is 1.5196291482691995

After 30 optimization iterations, training data loss is 1.2970068723054315, and
validation data loss is 1.348941780984258

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).
```



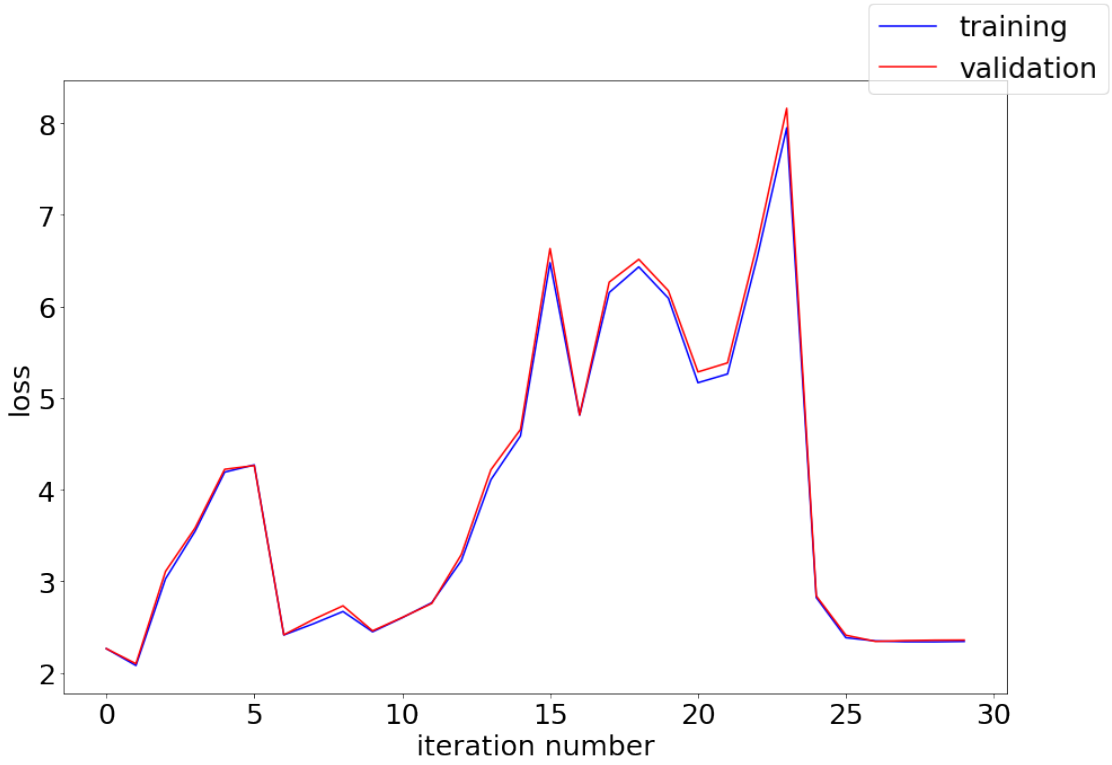

```
The total loss on the training data is 1.2970068723054315

The classification loss (i.e. without weight decay) on the training data is
1.2970068723054315

The classification error rate on the training data is 0.485


The total loss on the validation data is 1.348941780984258

The classification loss (i.e. without weight decay) on the validation data is
1.348941780984258

The classification error rate on the validation data is 0.514


The total loss on the test data is 1.3215141541942848

The classification loss (i.e. without weight decay) on the test data is
1.3215141541942848

The classification error rate on the test data is 0.49044444444444446
```

```
This round: 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 3 optimization iterations, training data loss is 2.183587087366446, and
validation data loss is 2.18766191191302

After 6 optimization iterations, training data loss is 1.8669674544071577, and
validation data loss is 1.8659573629601125

After 9 optimization iterations, training data loss is 2.7787603096438125, and
validation data loss is 2.8045056572488574

After 12 optimization iterations, training data loss is 2.7278693016157542, and
validation data loss is 2.754754141363878

After 15 optimization iterations, training data loss is 2.650289853293139, and
validation data loss is 2.679121169857526

After 18 optimization iterations, training data loss is 3.162010465127951, and
validation data loss is 3.198338745085088

After 21 optimization iterations, training data loss is 4.792963229438734, and
validation data loss is 4.883655748327477

After 24 optimization iterations, training data loss is 5.1503744221975305, and
validation data loss is 4.974199687483536

After 27 optimization iterations, training data loss is 4.696757218547823, and
validation data loss is 4.731345144274401

After 30 optimization iterations, training data loss is 3.9108041223157635, and
validation data loss is 3.9102058677708276

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).
```



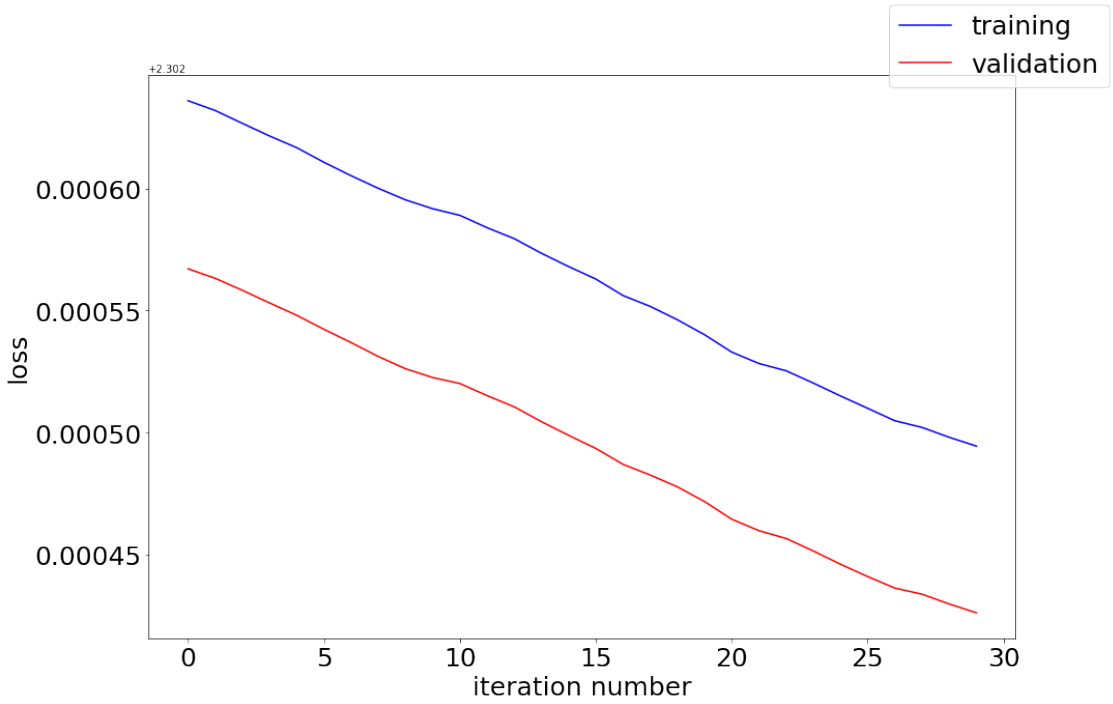

```
The total loss on the training data is 3.9108041223157635

The classification loss (i.e. without weight decay) on the training data is
3.9108041223157635

The classification error rate on the training data is 0.809


The total loss on the validation data is 3.9102058677708276

The classification loss (i.e. without weight decay) on the validation data is
3.9102058677708276

The classification error rate on the validation data is 0.791


The total loss on the test data is 3.957383294890199

The classification loss (i.e. without weight decay) on the test data is
3.957383294890199

The classification error rate on the test data is 0.8052222222222222
```

```
This round: 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 3 optimization iterations, training data loss is 3.026491009580142, and
validation data loss is 3.1073743977536763

After 6 optimization iterations, training data loss is 4.2698381343813905, and
validation data loss is 4.262568141490711

After 9 optimization iterations, training data loss is 2.6693135454750507, and
validation data loss is 2.731238033325344

After 12 optimization iterations, training data loss is 2.76519449333519, and
validation data loss is 2.7571752324108036

After 15 optimization iterations, training data loss is 4.586126998934177, and
validation data loss is 4.657121395944497

After 18 optimization iterations, training data loss is 6.151767555780349, and
validation data loss is 6.265425585971607

After 21 optimization iterations, training data loss is 5.167012182251127, and
validation data loss is 5.284753736460934

After 24 optimization iterations, training data loss is 7.94988607641146, and
validation data loss is 8.163203168156207

After 27 optimization iterations, training data loss is 2.3480795300931314, and
validation data loss is 2.3429237297820174

After 30 optimization iterations, training data loss is 2.3423985871265036, and
validation data loss is 2.3574707096066008

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).
```



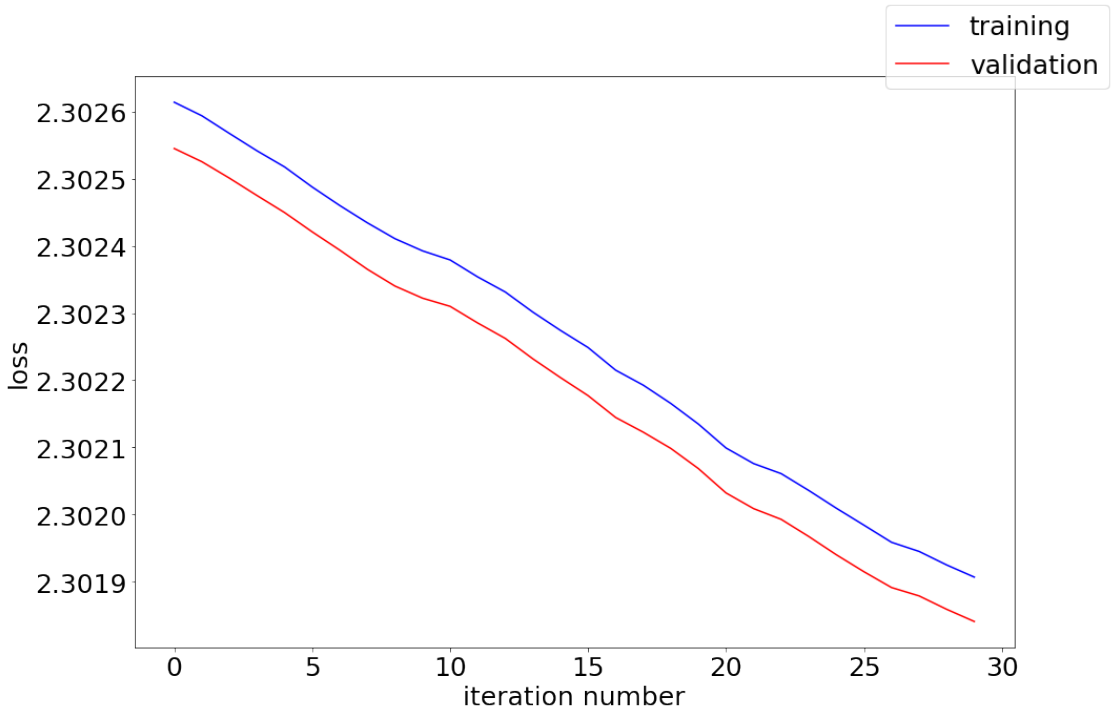

```
The total loss on the training data is 2.3423985871265036

The classification loss (i.e. without weight decay) on the training data is
2.3423985871265036

The classification error rate on the training data is 0.827


The total loss on the validation data is 2.3574707096066008

The classification loss (i.e. without weight decay) on the validation data is
2.3574707096066008

The classification error rate on the validation data is 0.835


The total loss on the test data is 2.3323692799479225

The classification loss (i.e. without weight decay) on the test data is
2.3323692799479225

The classification error rate on the test data is 0.8447777777777777
```

```
This round: 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 3 optimization iterations, training data loss is 2.3026267549161643, and
validation data loss is 2.3025583201679964

After 6 optimization iterations, training data loss is 2.3026107991005134, and
validation data loss is 2.3025423222513535

After 9 optimization iterations, training data loss is 2.3025953990752974, and
validation data loss is 2.302526167226052

After 12 optimization iterations, training data loss is 2.3025839957392673, and
validation data loss is 2.302515156414411

After 15 optimization iterations, training data loss is 2.302568056339131, and
validation data loss is 2.3024988774861317

After 18 optimization iterations, training data loss is 2.3025517264950315, and
validation data loss is 2.3024826069427027

After 21 optimization iterations, training data loss is 2.302532997577334, and
validation data loss is 2.3024644844506863

After 24 optimization iterations, training data loss is 2.3025203137483956, and
validation data loss is 2.3024514853124947

After 27 optimization iterations, training data loss is 2.302504856196062, and
validation data loss is 2.3024362601562625

After 30 optimization iterations, training data loss is 2.3024944574567052, and
validation data loss is 2.3024261216394932

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).
```



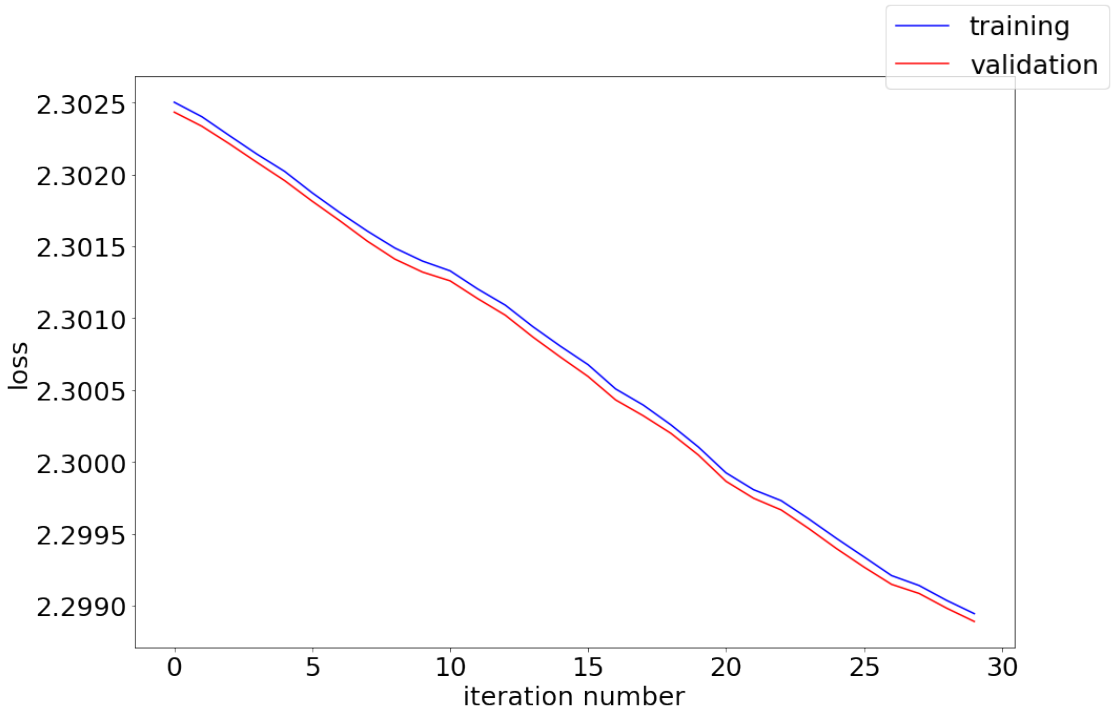

```
The total loss on the training data is 2.3024944574567052

The classification loss (i.e. without weight decay) on the training data is
2.3024944574567052

The classification error rate on the training data is 0.898


The total loss on the validation data is 2.3024261216394932

The classification loss (i.e. without weight decay) on the validation data is
2.3024261216394932

The classification error rate on the validation data is 0.899


The total loss on the test data is 2.3024550566668154

The classification loss (i.e. without weight decay) on the test data is
2.3024550566668154

The classification error rate on the test data is 0.8982222222222223

This round: 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 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).
```



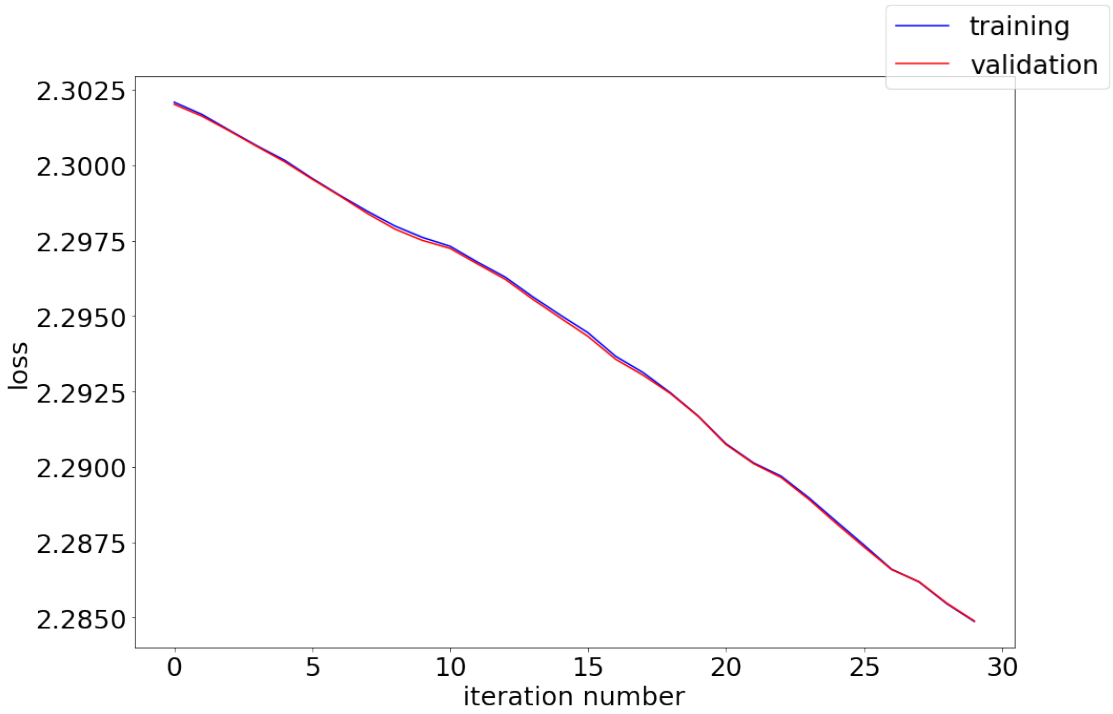

```
The total loss on the training data is 2.301906765216956

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
```

```
This round: 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 3 optimization iterations, training data loss is 2.302270722280603, and
validation data loss is 2.3022135077310604

After 6 optimization iterations, training data loss is 2.3018727323998753, and
validation data loss is 2.3018144795450715

After 9 optimization iterations, training data loss is 2.301488302952635, and
validation data loss is 2.3014111882587676

After 12 optimization iterations, training data loss is 2.301203644992165, and
validation data loss is 2.301136393797902

After 15 optimization iterations, training data loss is 2.300805182463726, and
validation data loss is 2.300729448040962

After 18 optimization iterations, training data loss is 2.300395983334581, and
validation data loss is 2.3003217145043204

After 21 optimization iterations, training data loss is 2.299924480966561, and
validation data loss is 2.299865339025977

After 24 optimization iterations, training data loss is 2.2996035585602, and
validation data loss is 2.299536568856648

After 27 optimization iterations, training data loss is 2.299209513515346, and
validation data loss is 2.299148553376364

After 30 optimization iterations, training data loss is 2.2989445467240364, and
validation data loss is 2.2988900834160577

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).
```



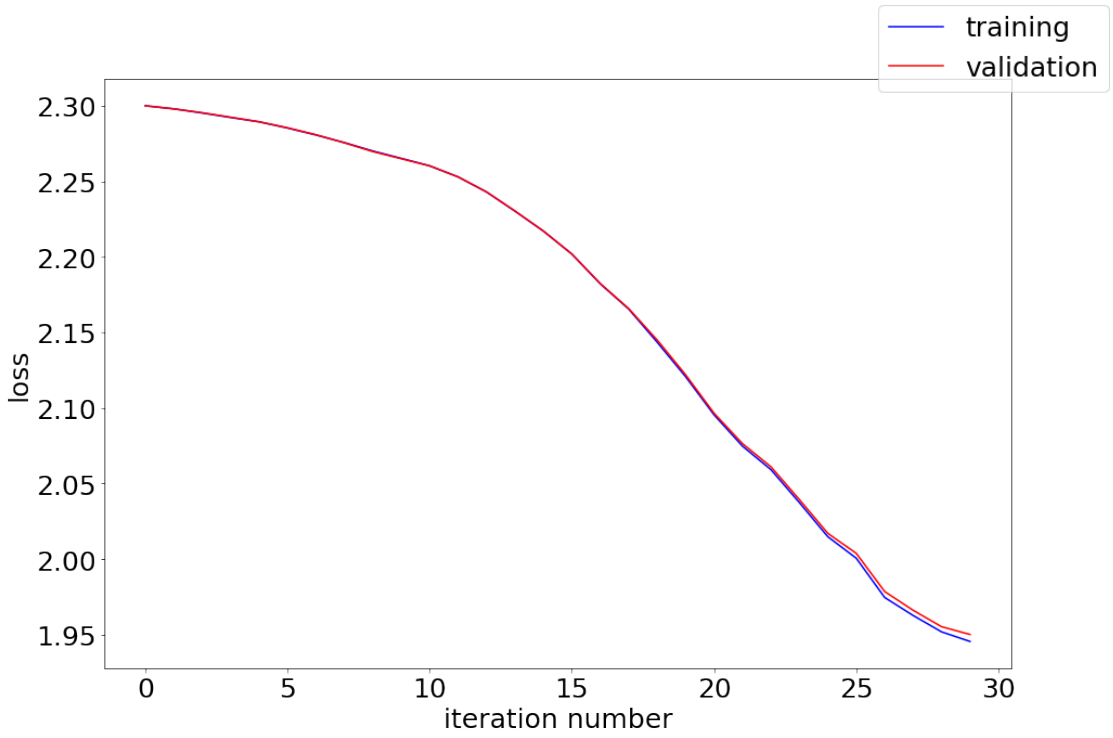

```
The total loss on the training data is 2.2989445467240364

The classification loss (i.e. without weight decay) on the training data is
2.2989445467240364

The classification error rate on the training data is 0.84


The total loss on the validation data is 2.2988900834160577

The classification loss (i.e. without weight decay) on the validation data is
2.2988900834160577

The classification error rate on the validation data is 0.827


The total loss on the test data is 2.2988914790482253

The classification loss (i.e. without weight decay) on the test data is
2.2988914790482253

The classification error rate on the test data is 0.8353333333333334
```

```
This round: 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 3 optimization iterations, training data loss is 2.3011589716760366, and
validation data loss is 2.3011369525276026

After 6 optimization iterations, training data loss is 2.299562965510114, and
validation data loss is 2.299537042734266

After 9 optimization iterations, training data loss is 2.297983840657499, and
validation data loss is 2.297881487040219

After 12 optimization iterations, training data loss is 2.296782770377463, and
validation data loss is 2.2967216485986213

After 15 optimization iterations, training data loss is 2.2950285268862114, and
validation data loss is 2.2949332947659298

After 18 optimization iterations, training data loss is 2.293123352458696, and
validation data loss is 2.293034838858313

After 21 optimization iterations, training data loss is 2.2907649853656578, and
validation data loss is 2.2907394945163317

After 24 optimization iterations, training data loss is 2.2889776602991603, and
validation data loss is 2.288923352668444

After 27 optimization iterations, training data loss is 2.2866066717358593, and
validation data loss is 2.2865945240766288

After 30 optimization iterations, training data loss is 2.284872226355331, and
validation data loss is 2.2848896629018163

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).
```



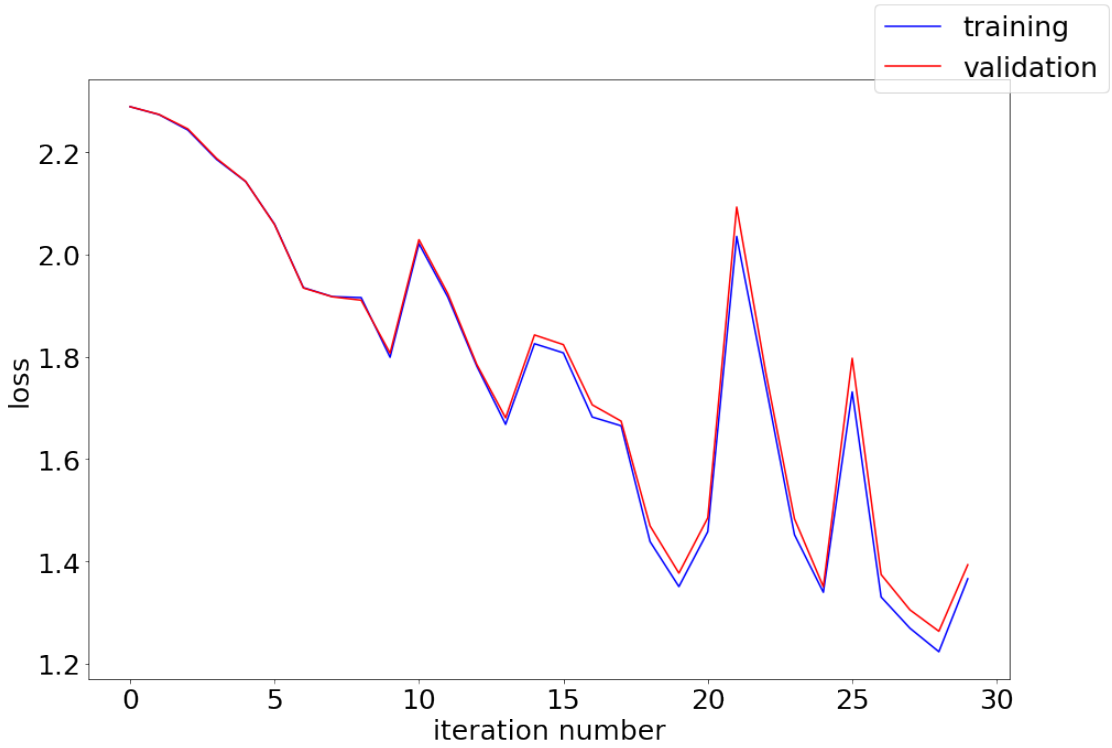

```
The total loss on the training data is 2.284872226355331

The classification loss (i.e. without weight decay) on the training data is
2.284872226355331

The classification error rate on the training data is 0.704


The total loss on the validation data is 2.2848896629018163

The classification loss (i.e. without weight decay) on the validation data is
2.2848896629018163

The classification error rate on the validation data is 0.704


The total loss on the test data is 2.2847466066806836

The classification loss (i.e. without weight decay) on the test data is
2.2847466066806836

The classification error rate on the test data is 0.7117777777777777
```

```
This round: 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 3 optimization iterations, training data loss is 2.2950895847339607, and
validation data loss is 2.295264991791344

After 6 optimization iterations, training data loss is 2.2851513266697707, and
validation data loss is 2.285322549067907

After 9 optimization iterations, training data loss is 2.2699576084183057, and
validation data loss is 2.2695759144747663

After 12 optimization iterations, training data loss is 2.252789841697498, and
validation data loss is 2.2528758637790736

After 15 optimization iterations, training data loss is 2.2169817095260567, and
validation data loss is 2.217005185198009

After 18 optimization iterations, training data loss is 2.165391502573485, and
validation data loss is 2.165721037763765

After 21 optimization iterations, training data loss is 2.09545616215241, and
validation data loss is 2.096550361483649

After 24 optimization iterations, training data loss is 2.0374530284827874, and
validation data loss is 2.039437416822139

After 27 optimization iterations, training data loss is 1.9746790018874145, and
validation data loss is 1.9786153686442194

After 30 optimization iterations, training data loss is 1.9456141566076735, and
validation data loss is 1.950216998025011

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).
```



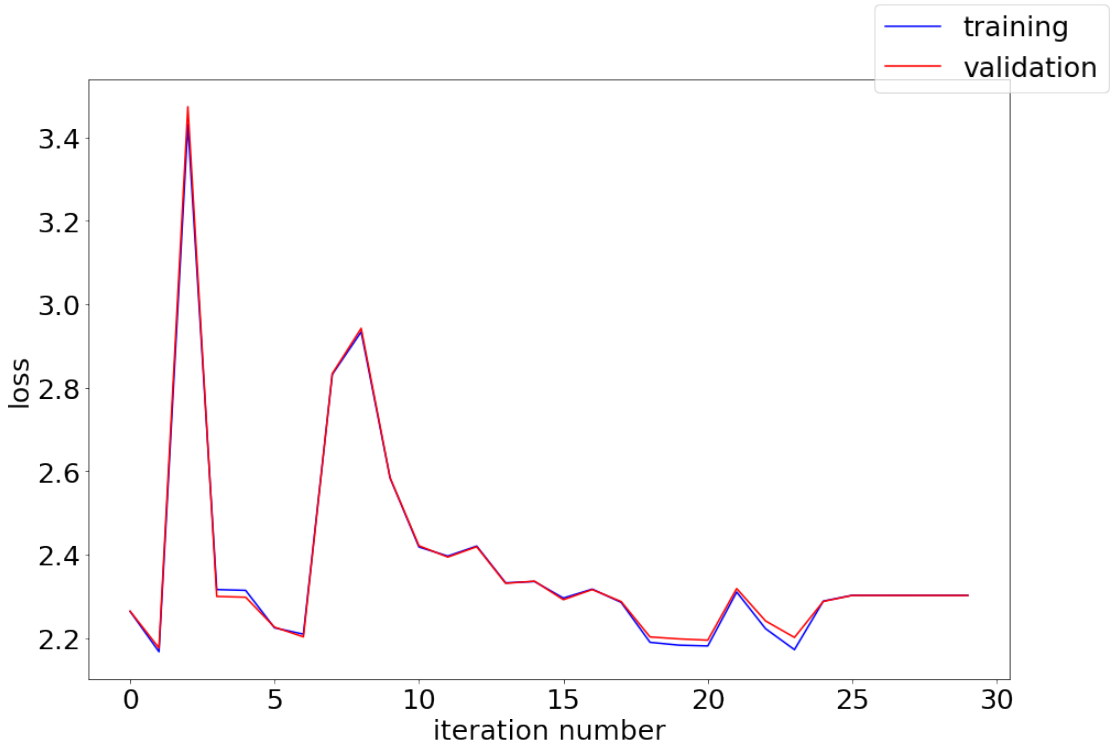

```
The total loss on the training data is 1.9456141566076735

The classification loss (i.e. without weight decay) on the training data is
1.9456141566076735

The classification error rate on the training data is 0.684


The total loss on the validation data is 1.950216998025011

The classification loss (i.e. without weight decay) on the validation data is
1.950216998025011

The classification error rate on the validation data is 0.683


The total loss on the test data is 1.942886481857062

The classification loss (i.e. without weight decay) on the test data is
1.942886481857062

The classification error rate on the test data is 0.6884444444444444
```

```
This round: 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 3 optimization iterations, training data loss is 2.243813910582931, and
validation data loss is 2.2460767063158524

After 6 optimization iterations, training data loss is 2.059900384510835, and
validation data loss is 2.0590744560574685

After 9 optimization iterations, training data loss is 1.9160914989944218, and
validation data loss is 1.9109828942304525

After 12 optimization iterations, training data loss is 1.917048302319422, and
validation data loss is 1.9237889694326196

After 15 optimization iterations, training data loss is 1.82587265631088, and
validation data loss is 1.8430004813461842

After 18 optimization iterations, training data loss is 1.665288344906398, and
validation data loss is 1.6744178453020688

After 21 optimization iterations, training data loss is 1.4582551993124988, and
validation data loss is 1.4852528003885392

After 24 optimization iterations, training data loss is 1.452320903638464, and
validation data loss is 1.4834138146568732

After 27 optimization iterations, training data loss is 1.3303922763623655, and
validation data loss is 1.3743191539017028

After 30 optimization iterations, training data loss is 1.3660074442569756, and
validation data loss is 1.3933947351036167

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 1.3660074442569756

The classification loss (i.e. without weight decay) on the training data is
1.3660074442569756

The classification error rate on the training data is 0.457


The total loss on the validation data is 1.3933947351036167

The classification loss (i.e. without weight decay) on the validation data is
1.3933947351036167

The classification error rate on the validation data is 0.493


The total loss on the test data is 1.3946544214152699

The classification loss (i.e. without weight decay) on the test data is
1.3946544214152699

The classification error rate on the test data is 0.478
```

```
This round: 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 3 optimization iterations, training data loss is 3.4311396698165475, and
validation data loss is 3.473669712540846

After 6 optimization iterations, training data loss is 2.2240985480813564, and
validation data loss is 2.22572254358168

After 9 optimization iterations, training data loss is 2.9338680804341877, and
validation data loss is 2.9423197867415056

After 12 optimization iterations, training data loss is 2.3961606161811972, and
validation data loss is 2.3937550552744713

After 15 optimization iterations, training data loss is 2.3351731623704155, and
validation data loss is 2.3359991910847664

After 18 optimization iterations, training data loss is 2.2851343405280784, and
validation data loss is 2.2871555992045955

After 21 optimization iterations, training data loss is 2.180694086161679, and
validation data loss is 2.194521785779183

After 24 optimization iterations, training data loss is 2.1718315181282803, and
validation data loss is 2.201099578485292

After 27 optimization iterations, training data loss is 2.302196175444347, and
validation data loss is 2.301808990746424

After 30 optimization iterations, training data loss is 2.302083198274314, and
validation data loss is 2.3017187263527137

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 2.302083198274314

The classification loss (i.e. without weight decay) on the training data is
2.302083198274314

The classification error rate on the training data is 0.881


The total loss on the validation data is 2.3017187263527137

The classification loss (i.e. without weight decay) on the validation data is
2.3017187263527137

The classification error rate on the validation data is 0.875


The total loss on the test data is 2.301919568249493

The classification loss (i.e. without weight decay) on the test data is
2.301919568249493

The classification error rate on the test data is 0.8791111111111111
```

[ ]: