6_Backpropagation_students

July 13, 2019

1 HaMLeT

1.1 Session 6: Backpropagation

by Leon Weninger and Raphael Kolk

1.1.1 Goal of this Session

In this session you will, step by step, implement a the backpropagation algorithm yourself without using any deep learning libraries. You should already be familiar with Python as well as NumPy (a package for scientific computing with Python).

1.1.2 Given code

Task 0: Familiarize yourself briefly with the given code. Pay particural attention to the `Layer` and `Cost` classes, from which you will derive the classes you implement, and the `Sigmoid` layer which is predefined as an example. You'll also need to execute the cells in this section once.

The following code loads the data and trains the network in a similar fashion as in the last session.

In [9]:
```
import numpy as np
from tqdm import tqdm
from load_mnist import load_mnist


def vectorize(j):
    label_vector = np.zeros((1, 10))
    label_vector[0, int(j)] = 1.0
    return label_vector


def load_data():
    images, labels = load_mnist()

    image_size = images.shape[1]
    label_size = labels.shape[1]

    random_permutation = np.random.permutation(images.shape[0])
```

```
    images = images[random_permutation, :]
    labels = labels[random_permutation, :]

    return images, labels, image_size, label_size


def train(net, cost_function, number_epochs, batch_size, learning_rate):
    images, labels, image_size, label_size = load_data()
    training_images, validation_images = images[:50000], images[50000:]
    training_labels, validation_labels = labels[:50000], labels[50000:]

    for e in range(number_epochs):
        cost = train_epoch(e, net, training_images, training_labels, cost_function, bat
        accuracy = validate_epoch(e, net, validation_images, validation_labels, batch_s
        print('cost=%5.6f, accuracy=%2.6f' % (cost, accuracy), flush=True)


def train_epoch(e, net, images, labels, cost_function, batch_size, learning_rate):
    epoch_cost = 0

    for i in tqdm(range(0, len(images), batch_size), ascii=False, desc='training,   e=%
        batch_images = images[i:min(i + batch_size, len(images)), :]
        batch_labels = labels[i:min(i + batch_size, len(labels)), :]

        # zero the gradients
        net.zero_gradients()

        # forward pass
        prediction = net.forward(batch_images)
        cost = cost_function.estimate(batch_labels, prediction)

        # backward pass
        dprediction = cost_function.gradient(cost)
        net.backward(dprediction)

        # update the parameters using the computed gradients via stochastic gradient d
        net.update_parameters(learning_rate)

        epoch_cost += np.mean(cost)

    return epoch_cost


def validate_epoch(e, net, images, labels, batch_size):
    n_correct = 0
    n_total = 0

    for i in tqdm(range(0, len(images), batch_size), ascii=False, desc='validation, e=%
```

```
            batch_images = images[i:min(i + batch_size, len(images)), :]
            batch_labels = labels[i:min(i + batch_size, len(labels)), :]

            # compute predicted probabilities.
            predictions = net.forward(batch_images)

            # find the most probable class label.
            n_correct += sum(np.argmax(batch_labels, axis=1) == np.argmax(predictions, axis
            n_total += batch_labels.shape[0]

        return n_correct / n_total
```

Remember the sigmoid function and its derivative you implemented in the previous session.

```
In [10]: def sigmoid_function(var):
             return 1.0 / (1.0 + np.exp(-var))


         def sigmoid_derivative(z):
             return sigmoid_function(z) * (1 - sigmoid_function(z))
```

The following abstract classes should serve as parent classes for all the different layers and cost functions which you will implement.

```
In [11]: class Layer:
             def __init__(self):
                 # Initialize all member variables of the layer.
                 pass

             def forward(self, x_in):
                 # Implemets for forward pass of the layer and returns x_out.
                 pass

             def backward(self, d_out):
                 # Implements the backward pass of the layer and returns d_in.
                 pass

             def zero_gradients(self):
                 # Sets all gradients of the layer to zero.
                 pass

             def update_parameters(self, learning_rate):
                 # Update the parameters of the layer with the help of the gradients stored du
                 pass


         class Cost:
             def __init__(self):
                 # Initialize all member variables of the cost function.
```

```
            pass

        def estimate(self, target, prediction):
            # Estimates and return the cost with respect to the predicted label and a tar
            pass

        def gradient(self, cost):
            # Calculates and returns the gradient with respect to the cost.
            pass
```

The following class derived from the Layer class implements the forward and backward pass of the sigmoid activation function alread known from the previous session and serves as an example for you. Since it does not have learnable parameters, no `update_parameters` or `zero_gradients` function needs to be implemented.

```
In [12]: class Sigmoid(Layer):
            def __init__(self):
                self.x_in = None

            def forward(self, x_in):
                self.x_in = x_in
                x_out = sigmoid_function(x_in)
                return x_out

            def backward(self, d_out):
                d_in = d_out * sigmoid_derivative(self.x_in)
                return d_in

            def zero_gradients(self):
                pass

            def update_parameters(self, learning_rate):
                pass
```

1.1.3 Theoretical Foundation

Task 1: Take a peace of paper and a pencil, use your knowledge from the preparation material and the introduction slides and fill in the gaps in the preparation material. Keep in mind that the inputs may be batched. Having written the formulas down, please check with the tutor if they are correct.

1.1.4 Practical Implementation

Task 2a: Implement the `forward` function for the `Linear` layer. Remember to store the `x_in` for use in the backward pass.

Task 2b: Implement the `estimate` function for the `MeanSquareError` cost, which estimates the cost after a ground truth target is set. Remember to store the `prediction` for calculating the gradient.

Task 2c: Implement the `gradient` function for the `MeanSquareError` cost, which calculates the gradient with respect to the cost. Use the `prediction` stored during the forward pass.

Task 2d: Implement the `backward` function for the `Linear` layer. The function should also calculate and accumulate the gradient of `w` and `b` with regard to the error.

Task 2e: Implement the `update_parameters` function for the `Linear` layer, i.e., use the gradients `dw` and `db` together with a given `learning_rate` to update the parameters `w` and `b` accordingly.

Task 2f: Test your implementation by propagating random input through a linear layer followed by a sigmoid layer, estimating the mean square error to a random target, calculating the gradient and propagating it back through the sigmoid and linear layer. Afterwards update the parameters of the linear layer using the function you implemented.

Task 3a: Implement the `Network` class which can encapsulate multiple layers. It offers the same interface as a layer and is therefore derived from the `Layer` parent as well. Make sure to implement all member functions needed. The `forward` function propagates a given input through all encapsulated layers and returns the final prediction of the network, whereas the `backward` function propagates a given gradient through all layers in reversed order. `zero_gradients` and `update_parameters` invoke the respective functions of the encapsulated layers.

Task 3b: Test your implementation analogous to task 2f but using the `Network` class to encapsulate the linear and sigmoid layer.

Task 4: Train the network you just implemented using the dataloader and train function given above and the hyperparameter given below.

Task 5: Come up with a more sophisticated network structure and adjust the hyperparameter in order to increase the accuracy.

```
In [13]: class Linear(Layer):
             def __init__(self, n_in, n_out, initial_sigma=0.1):
                 self.n_in = n_in
                 self.n_out = n_out

                 self.w = initial_sigma * np.random.randn(n_out, n_in)
                 self.b = np.zeros((1, n_out))

                 self.zero_gradients()

                 self.x_in = None

             def forward(self, x_in):
                 # ----- Add code for task 2a between comments -----
                 self.x_in =x_in

                 x_out = np.dot(self.x_in,self.w.T) +self.b

                 # -------------------------------------------------
                 return x_out

             def backward(self, d_out):
                 # ----- Add code for task 2d between comments -----

                 self.db += np.sum(d_out,axis=0,keepdims=True)
```

```
        self.dw += np.dot(d_out.T,self.x_in)

        self.d_in = np.dot(d_out,self.w)
        # ------------------------------------------------
        return self.d_in

    def zero_gradients(self):
        self.dw = np.zeros((self.n_out, self.n_in))
        self.db = np.zeros((1, self.n_out))
        self.dx = np.empty((0, self.n_in))

    def update_parameters(self, learning_rate):
        # ----- Add code for task 2e between comments -----

        self.w = self.w-(learning_rate*self.dw)
        self.b = self.b-(learning_rate*self.db)
        # ------------------------------------------------


class MeanSquareError(Cost):
    def __init__(self):
        self.prediction = None
        self.target = None

    def estimate(self, target, prediction):
        self.target = target
        self.prediction=prediction
        # ----- add code for task 2b between comments -----
        cost = 1/2*(target-prediction)**2
        # ------------------------------------------------
        return cost

    def gradient(self, cost):
        # ----- add code for task 2c between comments -----

        gradient =self.prediction- self.target

        #gradient = np.sprt(2*cost)

        # ------------------------------------------------
        return gradient


class Network(Layer):
    def __init__(self, layers):
        self.layers = layers
```

```
        # ----- add code for task 3a between comments -----
        def forward(self,x):
            for layer in self.layers:
                x=layer.forward(x)
            return x

        def backward(self,d):
            for layer in reversed(self.layers):
                d=layer.backward(d)
            return d

        def update_parameters(self,alpha):
            for layer in self.layers:
                layer.update_parameters(alpha)

        def zero_gradients(self):
            for layer in self.layers:
                layer.zero_gradients()

        # -------------------------------------------------
```

In [6]:
```
# define hyperparameters
input_size = 28**2
label_size = 10
batch_size = 600
learning_rate = 0.0001
number_epochs = 100

random_input = np.random.rand(batch_size, input_size)
random_label = np.random.rand(batch_size, label_size)

linear_layer = Linear(input_size, label_size)
sigmoid_layer = Sigmoid()
cost_function = MeanSquareError()

# ----- add code for task 2f between comments -----
act=linear_layer.forward(random_input)
pred=sigmoid_layer.forward(act)

cost=cost_function.estimate(random_label,pred)
grad=cost_function.gradient(cost)

d=sigmoid_layer.backward(grad)
d_in=linear_layer.backward(d)

linear_layer.update_parameters(learning_rate)

# -------------------------------------------------
```

```
        # ----- add code for task 3b between comments -----

        linear_layer_1 = Linear(input_size, label_size)
        sigmoid_layer = Sigmoid()
        net = Network([linear_layer_1,sigmoid_layer])
        train(net, cost_function, number_epochs, batch_size, learning_rate)




        # -------------------------------------------------

        # ----- add code for task 4 between comments ------


        # -------------------------------------------------

        # ----- add code for task 5 between comments ------


        # -------------------------------------------------
```

```
training,   e=0:   0%|          | 0/84 [00:00<?, ?it/s]/usr/local/anaconda3/lib/python3.6/site-

training,   e=0: 100%|| 84/84 [00:00<00:00, 98.33it/s]
validation, e=0: 100%|| 17/17 [00:00<00:00, 185.19it/s]

cost=4.938776, accuracy=0.309500



training,   e=1: 100%|| 84/84 [00:00<00:00, 104.64it/s]
validation, e=1: 100%|| 17/17 [00:00<00:00, 185.28it/s]

cost=3.624176, accuracy=0.306500



training,   e=2: 100%|| 84/84 [00:00<00:00, 105.60it/s]
validation, e=2: 100%|| 17/17 [00:00<00:00, 185.08it/s]

cost=3.551397, accuracy=0.320900



training,   e=3: 100%|| 84/84 [00:00<00:00, 106.68it/s]
validation, e=3: 100%|| 17/17 [00:00<00:00, 185.09it/s]
```

```
cost=3.511281, accuracy=0.333500


training,   e=4: 100%|| 84/84 [00:00<00:00, 105.62it/s]
validation, e=4: 100%|| 17/17 [00:00<00:00, 185.08it/s]

cost=3.461046, accuracy=0.335800


training,   e=5: 100%|| 84/84 [00:00<00:00, 86.33it/s]
validation, e=5: 100%|| 17/17 [00:00<00:00, 141.62it/s]

cost=3.331616, accuracy=0.396200


training,   e=6: 100%|| 84/84 [00:00<00:00, 102.03it/s]
validation, e=6: 100%|| 17/17 [00:00<00:00, 185.37it/s]

cost=3.168227, accuracy=0.413300


training,   e=7: 100%|| 84/84 [00:00<00:00, 107.01it/s]
validation, e=7: 100%|| 17/17 [00:00<00:00, 185.24it/s]

cost=3.058832, accuracy=0.417100


training,   e=8: 100%|| 84/84 [00:00<00:00, 104.91it/s]
validation, e=8: 100%|| 17/17 [00:00<00:00, 185.32it/s]

cost=2.994231, accuracy=0.431600


training,   e=9: 100%|| 84/84 [00:00<00:00, 98.41it/s]
validation, e=9: 100%|| 17/17 [00:00<00:00, 185.79it/s]

cost=2.963683, accuracy=0.428900


training,   e=10: 100%|| 84/84 [00:00<00:00, 108.27it/s]
validation, e=10: 100%|| 17/17 [00:00<00:00, 185.33it/s]

cost=2.921293, accuracy=0.438800
```

```
training,   e=11: 100%|| 84/84 [00:00<00:00, 108.18it/s]
validation, e=11: 100%|| 17/17 [00:00<00:00, 187.51it/s]

cost=2.894430, accuracy=0.441400



training,   e=12: 100%|| 84/84 [00:00<00:00, 92.79it/s]
validation, e=12: 100%|| 17/17 [00:00<00:00, 141.94it/s]

cost=2.871689, accuracy=0.451400



training,   e=13: 100%|| 84/84 [00:00<00:00, 108.81it/s]
validation, e=13: 100%|| 17/17 [00:00<00:00, 184.29it/s]

cost=2.854031, accuracy=0.442300



training,   e=14: 100%|| 84/84 [00:00<00:00, 108.94it/s]
validation, e=14: 100%|| 17/17 [00:00<00:00, 185.11it/s]

cost=2.836971, accuracy=0.443800



training,   e=15: 100%|| 84/84 [00:00<00:00, 106.09it/s]
validation, e=15: 100%|| 17/17 [00:00<00:00, 184.23it/s]

cost=2.827713, accuracy=0.444200



training,   e=16: 100%|| 84/84 [00:00<00:00, 92.51it/s]
validation, e=16: 100%|| 17/17 [00:00<00:00, 139.15it/s]

cost=2.833228, accuracy=0.458200



training,   e=17: 100%|| 84/84 [00:00<00:00, 109.96it/s]
validation, e=17: 100%|| 17/17 [00:00<00:00, 184.68it/s]

cost=2.813443, accuracy=0.459700



training,   e=18: 100%|| 84/84 [00:00<00:00, 110.14it/s]
validation, e=18: 100%|| 17/17 [00:00<00:00, 184.95it/s]
```

```
cost=2.804464, accuracy=0.465800


training,   e=19: 100%|| 84/84 [00:00<00:00, 110.13it/s]
validation, e=19: 100%|| 17/17 [00:00<00:00, 185.00it/s]

cost=2.798534, accuracy=0.467900


training,   e=20: 100%|| 84/84 [00:00<00:00, 102.17it/s]
validation, e=20: 100%|| 17/17 [00:00<00:00, 183.10it/s]

cost=2.784008, accuracy=0.468700


training,   e=21: 100%|| 84/84 [00:00<00:00, 102.78it/s]
validation, e=21: 100%|| 17/17 [00:00<00:00, 141.85it/s]

cost=2.779173, accuracy=0.467800


training,   e=22: 100%|| 84/84 [00:00<00:00, 100.15it/s]
validation, e=22: 100%|| 17/17 [00:00<00:00, 185.21it/s]

cost=2.776501, accuracy=0.467600


training,   e=23: 100%|| 84/84 [00:00<00:00, 109.24it/s]
validation, e=23: 100%|| 17/17 [00:00<00:00, 184.87it/s]

cost=2.763745, accuracy=0.469400


training,   e=24: 100%|| 84/84 [00:00<00:00, 106.26it/s]
validation, e=24: 100%|| 17/17 [00:00<00:00, 184.94it/s]

cost=2.756209, accuracy=0.468500


training,   e=25: 100%|| 84/84 [00:00<00:00, 85.54it/s]
validation, e=25: 100%|| 17/17 [00:00<00:00, 181.22it/s]

cost=2.757379, accuracy=0.472200
```

```
training,   e=26: 100%|| 84/84 [00:00<00:00, 102.88it/s]
validation, e=26: 100%|| 17/17 [00:00<00:00, 184.70it/s]

cost=2.744545, accuracy=0.472800



training,   e=27: 100%|| 84/84 [00:01<00:00, 78.79it/s]
validation, e=27: 100%|| 17/17 [00:00<00:00, 143.03it/s]

cost=2.740545, accuracy=0.474200



training,   e=28: 100%|| 84/84 [00:00<00:00, 109.50it/s]
validation, e=28: 100%|| 17/17 [00:00<00:00, 179.70it/s]

cost=2.733786, accuracy=0.478800



training,   e=29: 100%|| 84/84 [00:00<00:00, 108.12it/s]
validation, e=29: 100%|| 17/17 [00:00<00:00, 184.78it/s]

cost=2.719958, accuracy=0.479400



training,   e=30: 100%|| 84/84 [00:00<00:00, 107.57it/s]
validation, e=30: 100%|| 17/17 [00:00<00:00, 184.67it/s]

cost=2.707361, accuracy=0.487400



training,   e=31: 100%|| 84/84 [00:00<00:00, 106.98it/s]
validation, e=31: 100%|| 17/17 [00:00<00:00, 185.06it/s]

cost=2.707886, accuracy=0.478700



training,   e=32: 100%|| 84/84 [00:00<00:00, 107.55it/s]
validation, e=32: 100%|| 17/17 [00:00<00:00, 184.66it/s]

cost=2.678279, accuracy=0.485100



training,   e=33: 100%|| 84/84 [00:00<00:00, 91.85it/s]
validation, e=33: 100%|| 17/17 [00:00<00:00, 184.20it/s]
```

```
cost=2.666325, accuracy=0.494200


training,   e=34: 100%|| 84/84 [00:00<00:00, 87.82it/s]
validation, e=34: 100%|| 17/17 [00:00<00:00, 147.79it/s]

cost=2.655260, accuracy=0.501600


training,   e=35: 100%|| 84/84 [00:00<00:00, 102.87it/s]
validation, e=35: 100%|| 17/17 [00:00<00:00, 185.91it/s]

cost=2.646725, accuracy=0.497400


training,   e=36: 100%|| 84/84 [00:00<00:00, 104.94it/s]
validation, e=36: 100%|| 17/17 [00:00<00:00, 186.12it/s]

cost=2.630066, accuracy=0.499200


training,   e=37: 100%|| 84/84 [00:00<00:00, 103.39it/s]
validation, e=37: 100%|| 17/17 [00:00<00:00, 171.69it/s]

cost=2.635276, accuracy=0.510300


training,   e=38: 100%|| 84/84 [00:00<00:00, 105.63it/s]
validation, e=38: 100%|| 17/17 [00:00<00:00, 185.57it/s]

cost=2.623842, accuracy=0.537000


training,   e=39: 100%|| 84/84 [00:00<00:00, 106.26it/s]
validation, e=39: 100%|| 17/17 [00:00<00:00, 185.60it/s]

cost=2.581603, accuracy=0.537500


training,   e=40: 100%|| 84/84 [00:00<00:00, 104.66it/s]
validation, e=40: 100%|| 17/17 [00:00<00:00, 185.04it/s]

cost=2.520833, accuracy=0.545400
```

```
training,   e=41: 100%|| 84/84 [00:00<00:00, 103.92it/s]
validation, e=41: 100%|| 17/17 [00:00<00:00, 185.01it/s]

cost=2.467117, accuracy=0.550700



training,   e=42: 100%|| 84/84 [00:00<00:00, 108.89it/s]
validation, e=42: 100%|| 17/17 [00:00<00:00, 185.26it/s]

cost=2.397191, accuracy=0.553800



training,   e=43: 100%|| 84/84 [00:00<00:00, 109.47it/s]
validation, e=43: 100%|| 17/17 [00:00<00:00, 185.62it/s]

cost=2.353979, accuracy=0.556800



training,   e=44: 100%|| 84/84 [00:00<00:00, 97.42it/s]
validation, e=44: 100%|| 17/17 [00:00<00:00, 140.45it/s]

cost=2.332697, accuracy=0.558900



training,   e=45: 100%|| 84/84 [00:00<00:00, 98.00it/s]
validation, e=45: 100%|| 17/17 [00:00<00:00, 185.64it/s]

cost=2.311087, accuracy=0.559800



training,   e=46: 100%|| 84/84 [00:00<00:00, 110.28it/s]
validation, e=46: 100%|| 17/17 [00:00<00:00, 185.35it/s]

cost=2.300523, accuracy=0.559800



training,   e=47: 100%|| 84/84 [00:00<00:00, 109.90it/s]
validation, e=47: 100%|| 17/17 [00:00<00:00, 185.31it/s]

cost=2.287708, accuracy=0.561300



training,   e=48: 100%|| 84/84 [00:00<00:00, 103.65it/s]
validation, e=48: 100%|| 17/17 [00:00<00:00, 183.99it/s]
```

```
cost=2.282187, accuracy=0.559300


training,   e=49: 100%|| 84/84 [00:00<00:00, 102.65it/s]
validation, e=49: 100%|| 17/17 [00:00<00:00, 140.79it/s]

cost=2.276578, accuracy=0.565600


training,   e=50: 100%|| 84/84 [00:00<00:00, 88.83it/s]
validation, e=50: 100%|| 17/17 [00:00<00:00, 186.06it/s]

cost=2.284204, accuracy=0.563600


training,   e=51: 100%|| 84/84 [00:00<00:00, 108.71it/s]
validation, e=51: 100%|| 17/17 [00:00<00:00, 185.19it/s]

cost=2.280297, accuracy=0.559800


training,   e=52: 100%|| 84/84 [00:00<00:00, 109.63it/s]
validation, e=52: 100%|| 17/17 [00:00<00:00, 185.28it/s]

cost=2.264273, accuracy=0.560100


training,   e=53: 100%|| 84/84 [00:00<00:00, 109.92it/s]
validation, e=53: 100%|| 17/17 [00:00<00:00, 185.38it/s]

cost=2.253747, accuracy=0.562300


training,   e=54: 100%|| 84/84 [00:00<00:00, 96.16it/s]
validation, e=54: 100%|| 17/17 [00:00<00:00, 140.83it/s]

cost=2.235027, accuracy=0.565500


training,   e=55: 100%|| 84/84 [00:00<00:00, 100.62it/s]
validation, e=55: 100%|| 17/17 [00:00<00:00, 185.84it/s]

cost=2.240275, accuracy=0.562500
```

```
training,   e=56: 100%|| 84/84 [00:00<00:00, 109.86it/s]
validation, e=56: 100%|| 17/17 [00:00<00:00, 185.31it/s]

cost=2.242104, accuracy=0.563700



training,   e=57: 100%|| 84/84 [00:00<00:00, 109.61it/s]
validation, e=57: 100%|| 17/17 [00:00<00:00, 185.91it/s]

cost=2.230347, accuracy=0.565200



training,   e=58: 100%|| 84/84 [00:00<00:00, 109.60it/s]
validation, e=58: 100%|| 17/17 [00:00<00:00, 185.05it/s]

cost=2.238238, accuracy=0.564800



training,   e=59: 100%|| 84/84 [00:00<00:00, 110.51it/s]
validation, e=59: 100%|| 17/17 [00:00<00:00, 185.30it/s]

cost=2.227793, accuracy=0.566000



training,   e=60: 100%|| 84/84 [00:00<00:00, 110.40it/s]
validation, e=60: 100%|| 17/17 [00:00<00:00, 185.50it/s]

cost=2.220233, accuracy=0.567100



training,   e=61: 100%|| 84/84 [00:01<00:00, 68.95it/s]
validation, e=61: 100%|| 17/17 [00:00<00:00, 132.96it/s]

cost=2.215265, accuracy=0.567500



training,   e=62: 100%|| 84/84 [00:00<00:00, 96.49it/s]
validation, e=62: 100%|| 17/17 [00:00<00:00, 152.10it/s]

cost=2.218362, accuracy=0.567000



training,   e=63: 100%|| 84/84 [00:00<00:00, 109.85it/s]
validation, e=63: 100%|| 17/17 [00:00<00:00, 185.02it/s]
```

```
cost=2.209902, accuracy=0.568500


training,   e=64: 100%|| 84/84 [00:00<00:00, 109.06it/s]
validation, e=64: 100%|| 17/17 [00:00<00:00, 184.59it/s]

cost=2.204271, accuracy=0.565200


training,   e=65: 100%|| 84/84 [00:00<00:00, 110.31it/s]
validation, e=65: 100%|| 17/17 [00:00<00:00, 184.42it/s]

cost=2.209475, accuracy=0.570000


training,   e=66: 100%|| 84/84 [00:00<00:00, 109.82it/s]
validation, e=66: 100%|| 17/17 [00:00<00:00, 184.34it/s]

cost=2.203692, accuracy=0.569300


training,   e=67: 100%|| 84/84 [00:00<00:00, 109.60it/s]
validation, e=67: 100%|| 17/17 [00:00<00:00, 184.10it/s]

cost=2.203329, accuracy=0.561700


training,   e=68: 100%|| 84/84 [00:00<00:00, 109.12it/s]
validation, e=68: 100%|| 17/17 [00:00<00:00, 183.43it/s]

cost=2.199000, accuracy=0.572300


training,   e=69: 100%|| 84/84 [00:00<00:00, 110.46it/s]
validation, e=69: 100%|| 17/17 [00:00<00:00, 184.49it/s]

cost=2.188360, accuracy=0.573500


training,   e=70: 100%|| 84/84 [00:00<00:00, 110.33it/s]
validation, e=70: 100%|| 17/17 [00:00<00:00, 184.57it/s]

cost=2.185778, accuracy=0.574800
```

```
training,   e=71: 100%|| 84/84 [00:00<00:00, 103.58it/s]
validation, e=71: 100%|| 17/17 [00:00<00:00, 183.74it/s]

cost=2.187600, accuracy=0.567200



training,   e=72: 100%|| 84/84 [00:00<00:00, 92.39it/s]
validation, e=72: 100%|| 17/17 [00:00<00:00, 141.34it/s]

cost=2.195178, accuracy=0.568600



training,   e=73: 100%|| 84/84 [00:00<00:00, 110.59it/s]
validation, e=73: 100%|| 17/17 [00:00<00:00, 184.50it/s]

cost=2.176681, accuracy=0.570600



training,   e=74: 100%|| 84/84 [00:00<00:00, 110.14it/s]
validation, e=74: 100%|| 17/17 [00:00<00:00, 184.73it/s]

cost=2.174270, accuracy=0.571300



training,   e=75: 100%|| 84/84 [00:00<00:00, 105.69it/s]
validation, e=75: 100%|| 17/17 [00:00<00:00, 183.77it/s]

cost=2.175344, accuracy=0.571000



training,   e=76: 100%|| 84/84 [00:00<00:00, 106.54it/s]
validation, e=76: 100%|| 17/17 [00:00<00:00, 184.54it/s]

cost=2.177708, accuracy=0.567100



training,   e=77: 100%|| 84/84 [00:00<00:00, 110.35it/s]
validation, e=77: 100%|| 17/17 [00:00<00:00, 184.66it/s]

cost=2.171814, accuracy=0.573500



training,   e=78: 100%|| 84/84 [00:00<00:00, 102.89it/s]
validation, e=78: 100%|| 17/17 [00:00<00:00, 183.57it/s]
```

```
cost=2.172597, accuracy=0.573700


training,   e=79: 100%|| 84/84 [00:00<00:00, 104.52it/s]
validation, e=79: 100%|| 17/17 [00:00<00:00, 184.40it/s]

cost=2.162491, accuracy=0.573200


training,   e=80: 100%|| 84/84 [00:00<00:00, 110.31it/s]
validation, e=80: 100%|| 17/17 [00:00<00:00, 184.57it/s]

cost=2.164888, accuracy=0.572300


training,   e=81: 100%|| 84/84 [00:00<00:00, 110.80it/s]
validation, e=81: 100%|| 17/17 [00:00<00:00, 184.71it/s]

cost=2.167944, accuracy=0.569700


training,   e=82: 100%|| 84/84 [00:00<00:00, 110.42it/s]
validation, e=82: 100%|| 17/17 [00:00<00:00, 184.88it/s]

cost=2.168157, accuracy=0.576600


training,   e=83: 100%|| 84/84 [00:00<00:00, 110.62it/s]
validation, e=83: 100%|| 17/17 [00:00<00:00, 184.48it/s]

cost=2.161013, accuracy=0.573400


training,   e=84: 100%|| 84/84 [00:00<00:00, 110.86it/s]
validation, e=84: 100%|| 17/17 [00:00<00:00, 184.94it/s]

cost=2.155828, accuracy=0.574200


training,   e=85: 100%|| 84/84 [00:00<00:00, 111.65it/s]
validation, e=85: 100%|| 17/17 [00:00<00:00, 184.75it/s]

cost=2.159542, accuracy=0.574700
```

```
training,   e=86: 100%|| 84/84 [00:00<00:00, 111.03it/s]
validation, e=86: 100%|| 17/17 [00:00<00:00, 185.27it/s]

cost=2.160014, accuracy=0.572000



training,   e=87: 100%|| 84/84 [00:00<00:00, 110.15it/s]
validation, e=87: 100%|| 17/17 [00:00<00:00, 184.36it/s]

cost=2.157137, accuracy=0.574100



training,   e=88: 100%|| 84/84 [00:00<00:00, 109.97it/s]
validation, e=88: 100%|| 17/17 [00:00<00:00, 184.61it/s]

cost=2.159303, accuracy=0.576700



training,   e=89: 100%|| 84/84 [00:00<00:00, 110.28it/s]
validation, e=89: 100%|| 17/17 [00:00<00:00, 184.68it/s]

cost=2.151820, accuracy=0.573100



training,   e=90: 100%|| 84/84 [00:00<00:00, 110.99it/s]
validation, e=90: 100%|| 17/17 [00:00<00:00, 183.54it/s]

cost=2.152159, accuracy=0.572700



training,   e=91: 100%|| 84/84 [00:00<00:00, 110.73it/s]
validation, e=91: 100%|| 17/17 [00:00<00:00, 184.42it/s]

cost=2.153817, accuracy=0.573500



training,   e=92: 100%|| 84/84 [00:00<00:00, 110.49it/s]
validation, e=92: 100%|| 17/17 [00:00<00:00, 184.44it/s]

cost=2.152125, accuracy=0.573900



training,   e=93: 100%|| 84/84 [00:00<00:00, 110.21it/s]
validation, e=93: 100%|| 17/17 [00:00<00:00, 184.56it/s]
```

```
cost=2.152750, accuracy=0.573000
```

```
training,   e=94: 100%|| 84/84 [00:00<00:00, 110.07it/s]
validation, e=94: 100%|| 17/17 [00:00<00:00, 184.83it/s]
```

```
cost=2.150829, accuracy=0.573100
```

```
training,   e=95: 100%|| 84/84 [00:00<00:00, 110.35it/s]
validation, e=95: 100%|| 17/17 [00:00<00:00, 184.69it/s]
```

```
cost=2.148718, accuracy=0.575400
```

```
training,   e=96: 100%|| 84/84 [00:00<00:00, 108.63it/s]
validation, e=96: 100%|| 17/17 [00:00<00:00, 180.89it/s]
```

```
cost=2.143742, accuracy=0.574200
```

```
training,   e=97: 100%|| 84/84 [00:00<00:00, 108.98it/s]
validation, e=97: 100%|| 17/17 [00:00<00:00, 180.97it/s]
```

```
cost=2.142858, accuracy=0.572100
```

```
training,   e=98: 100%|| 84/84 [00:00<00:00, 108.55it/s]
validation, e=98: 100%|| 17/17 [00:00<00:00, 184.65it/s]
```

```
cost=2.138705, accuracy=0.571700
```

```
training,   e=99: 100%|| 84/84 [00:00<00:00, 110.00it/s]
validation, e=99: 100%|| 17/17 [00:00<00:00, 184.65it/s]
```

```
cost=2.145199, accuracy=0.574300
```

1.1.5 Feedback

Aaaaaand we're done

If you have any suggestions on how we could improve this session, please let us know in the following cell. What did you particularly like or dislike? Did you miss any contents?

1.1.6 Additional Tasks

Task 6a: Implement the `forward` function for the `SoftMax` layer.
Task 6b: Implement the `estimate` function for the `CrossEntropy` cost.
Task 6c: Implement the `gradient` function for the `CrossEntropy` cost.
Task 6d: Implement the `backward` function for the `SoftMax` layer.
Task 6e: Test your implementation by setting up a network using the soft max layer and cross entropy cost in combination.

In [7]:
```
class SoftMax(Layer):
    def __init__(self):
        self.x_out = None

    def forward(self, x_in):
        # ----- add code for task 6a between comments -----
        # -------------------------------------------------
        return self.x_out

    def backward(self, d_out):
        # ----- add code for task 6d between comments -----
        # -------------------------------------------------
        return d_in


class CrossEntropy(Cost):
    def __init__(self):
        self.x_in = None
        self.target = None
        self.eps = 1e-12

    def estimate(self, target, x_in):
        # ----- add code for task 6b between comments -----
        # -------------------------------------------------
        return cost

    def gradient(self, d_out):
        # ----- add code for task 6c between comments -----
        # -------------------------------------------------
        return gradient
```

In [8]:
```
# define hyperparameters
input_size = 28**2
label_size = 10
batch_size = 600
learning_rate = 0.00001
number_epochs = 100

# ----- add code for task 6e between comments -----
# -------------------------------------------------
```