CS5242 : Neural Networks and Deep Learning

Lecture 3: Vanilla Neural Networks
Inference and Learning

Semester 1 2021/22

Xavier Bresson

https://twitter.com/xbresson

Department of Computer Science
National University of Singapore (NUS)

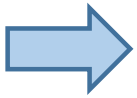
Outline

- Image classification task
- Neural networks for image classification
- Forward pass for inference
- Understanding linear layers
- Understanding the backward pass
- Matrix formulation
- Mini-batch learning

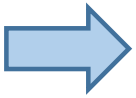
Outline

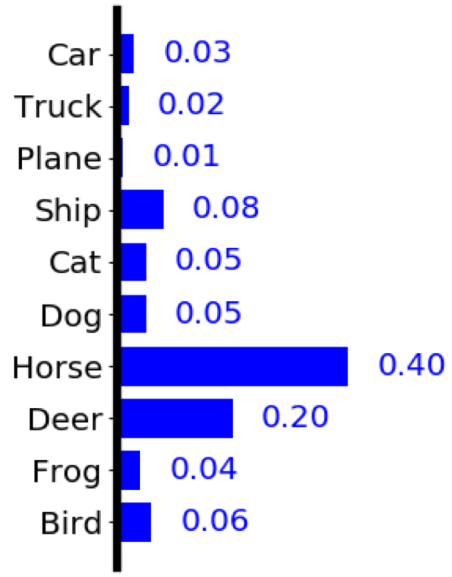
- Image classification task
- Neural networks for image classification
- Forward pass for inference
- Understanding linear layers
- Understanding the backward pass
- Matrix formulation
- Mini-batch learning

Classification task



Algorithm





Input

$3 \times 32 \times 32$ tensor

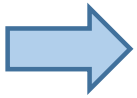
Output

Probabilities over classes

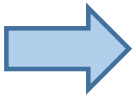
Outline

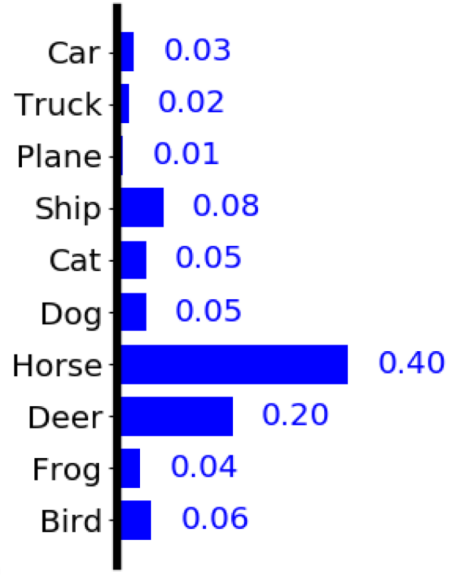
- Image classification task
- **Neural networks for image classification**
- Forward pass for inference
- Understanding linear layers
- Understanding the backward pass
- Matrix formulation
- Mini-batch learning

Neural network as classification algorithm



Neural Net





Input

$3 \times 32 \times 32$ tensor

The classification task is solved by the forward pass, which consists in passing the input image through a series of linear and non-linear operations to obtain the solution.

Output

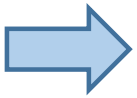
Probabilities over classes

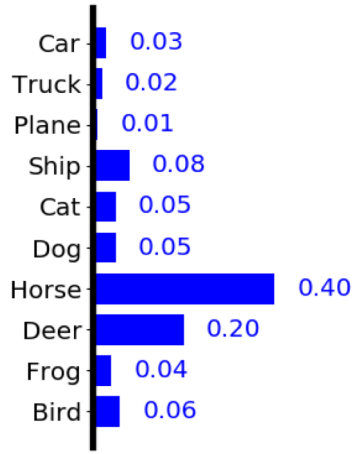
Outline

- Image classification task
- Neural networks for image classification
- **Forward pass for inference**
- Understanding linear layers
- Understanding the backward pass
- Matrix formulation
- Mini-batch learning

Forward pass

- Inference/prediction:
  - How to take 3072 numbers and convert them into 10 positive numbers that sum to one (probability function)?





Input

$3 \times 32 \times 32 = 3072$ numbers

Output

10 positive numbers that sum to one

Simplest possible transformations

- Inference is done by a linear layer (LL) + a (non-linear) softmax layer.
  - This is the simplest way to take 3072 numbers and to convert them into 10 positive numbers that sum to one (probability function).

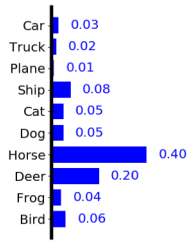
$$\begin{bmatrix} .14 \\ .72 \\ .54 \\ .82 \\ \vdots \\ .75 \end{bmatrix}$$

Linear Transformation

$$\begin{bmatrix} -1.6 \\ -0.9 \\ -1.1 \\ -2.5 \\ 0.1 \\ 0.5 \\ 1.7 \\ 1.2 \\ -0.3 \\ 0.2 \end{bmatrix}$$

Softmax Function

$$\begin{bmatrix} .01 \\ .03 \\ .02 \\ .01 \\ .08 \\ .11 \\ .38 \\ .23 \\ .05 \\ .08 \end{bmatrix}$$



3072 numbers

10 numbers

10 positive numbers that sum to one

Linear layer

- Linear transformation:

$$Wx + b$$

  - Operation that changes the data dimension from 3072 to 10.
  - Linear layers have both a weight matrix and a bias vector:

$$\begin{bmatrix} .14 & -.20 & -.39 & -.45 & \cdots & -.50 \\ -.22 & -.52 & .44 & -.69 & \cdots & -.75 \\ -.06 & .32 & -.29 & .94 & \cdots & .97 \\ .68 & .32 & .33 & -.84 & \cdots & .34 \\ .55 & -.30 & .90 & .05 & \cdots & .79 \\ -.15 & .02 & .45 & .84 & \cdots & -.08 \\ .24 & -.65 & -.38 & .38 & \cdots & .63 \\ .17 & .27 & -.67 & -.99 & \cdots & -.22 \\ -.74 & .43 & .08 & .47 & \cdots & .95 \\ .15 & -.78 & .59 & -.60 & \cdots & .38 \end{bmatrix} \begin{bmatrix} .12 \\ .32 \\ .74 \\ .65 \\ \vdots \\ .42 \\ .18 \end{bmatrix} + \begin{bmatrix} 2.5 \\ -1.2 \\ -3.4 \\ 1.7 \\ 5.6 \\ -3.1 \\ 2.4 \\ -5.0 \\ 4.7 \\ 1.2 \end{bmatrix} = \begin{bmatrix} -1.6 \\ -0.9 \\ -1.1 \\ -2.5 \\ 0.1 \\ 0.5 \\ 1.7 \\ 1.2 \\ -0.3 \\ 0.2 \end{bmatrix}$$

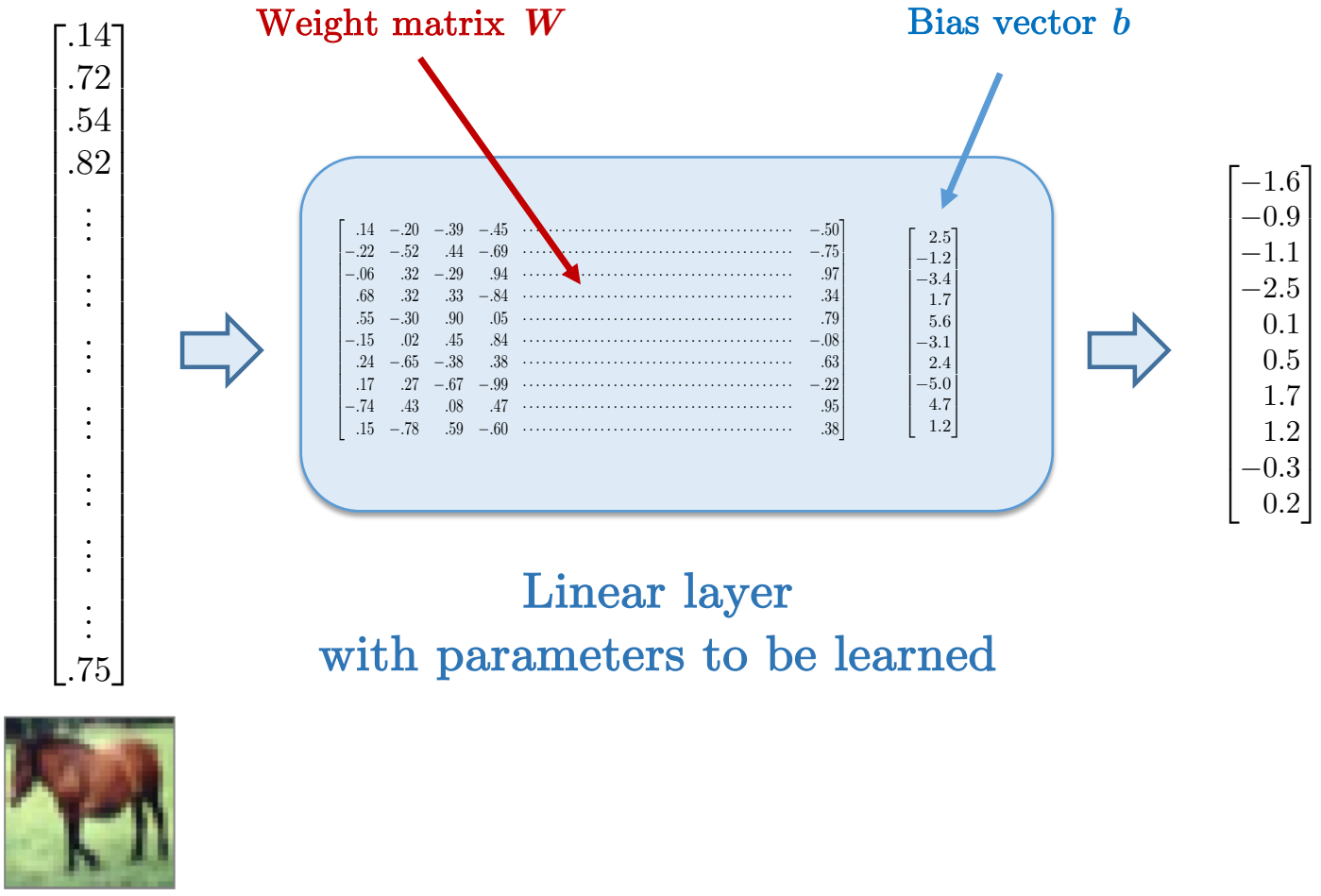
Weight matrix W

Input data x
3072 numbers

Bias vector b

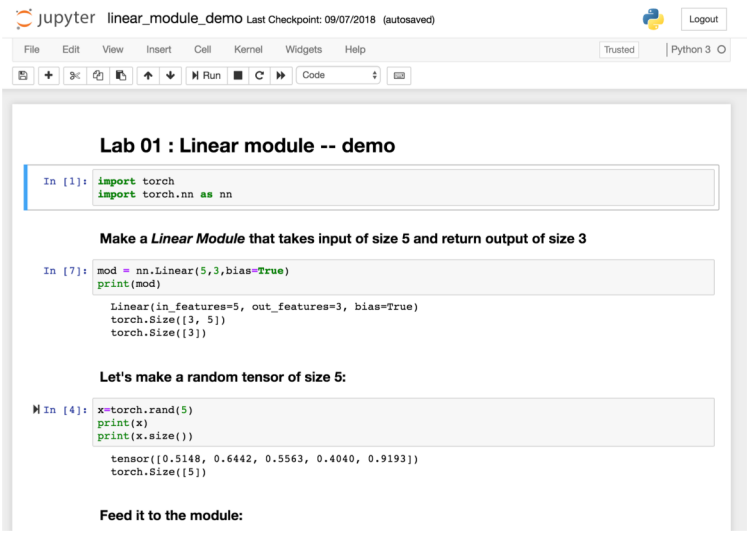
10 values

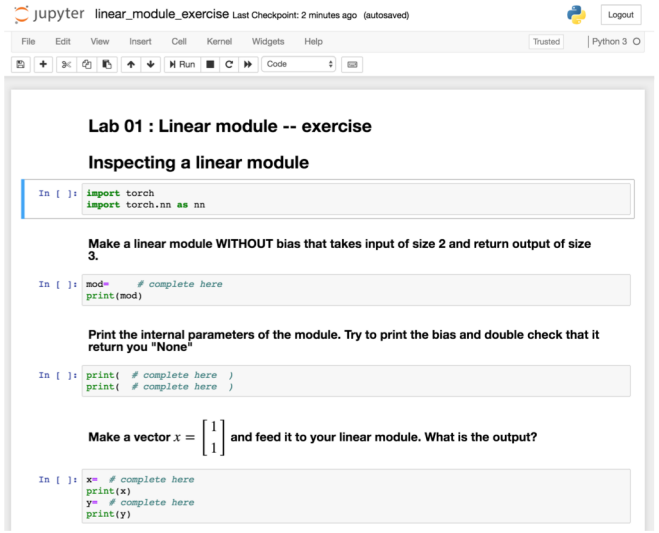
Linear layer in PyTorch



Lab 01

- Linear layer module

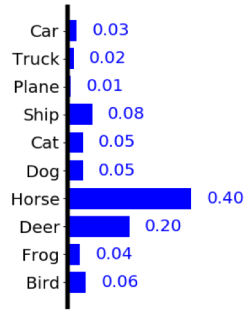




Softmax layer

- Softmax layer converts the 10 scores into a probability distribution over the 10 classes:

$$\begin{bmatrix} -1.6 \\ -0.9 \\ -1.1 \\ -2.5 \\ 0.1 \\ 0.5 \\ 1.7 \\ 1.2 \\ -0.3 \\ 0.2 \end{bmatrix} \xrightarrow{\exp()} \begin{bmatrix} 0.2 \\ 0.4 \\ 0.3 \\ 0.1 \\ 1.1 \\ 1.6 \\ 5.5 \\ 3.3 \\ 0.7 \\ 1.2 \end{bmatrix} \xrightarrow{\text{divide by sum()}} \begin{bmatrix} .01 \\ .03 \\ .02 \\ .01 \\ .08 \\ .11 \\ .38 \\ .23 \\ .05 \\ .08 \end{bmatrix}$$



sum = 14.53

sum = 1

Scores

Positive numbers

Probabilities over classes

Softmax layer

- These two operations (exponentiate then unit normalization) can be done into one single step by the Softmax function:

$$\text{Softmax}\left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}\right) = \begin{bmatrix} \frac{e^{x_1}}{e^{x_1}+e^{x_2}+e^{x_3}+e^{x_4}} \\ \frac{e^{x_2}}{e^{x_1}+e^{x_2}+e^{x_3}+e^{x_4}} \\ \frac{e^{x_3}}{e^{x_1}+e^{x_2}+e^{x_3}+e^{x_4}} \\ \frac{e^{x_4}}{e^{x_1}+e^{x_2}+e^{x_3}+e^{x_4}} \end{bmatrix}$$

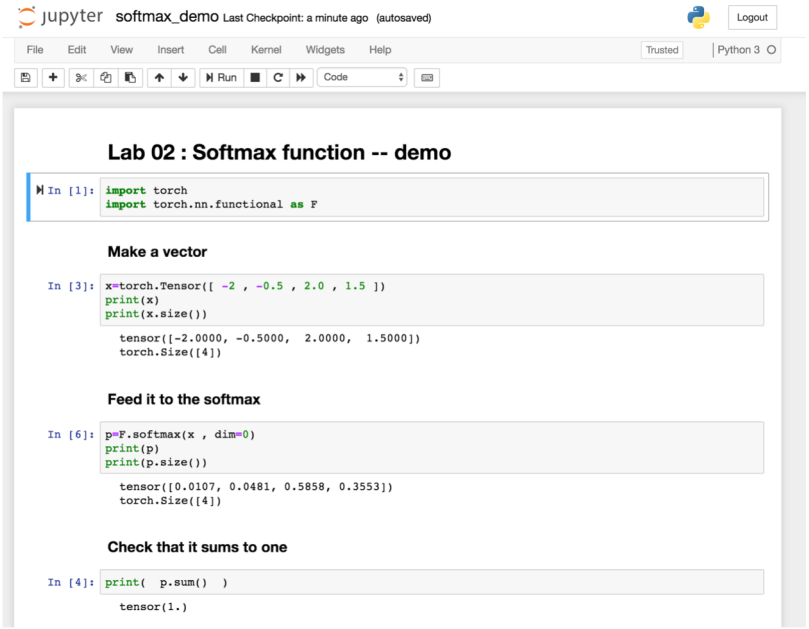
- It is called "Softmax" because it detects the Max :

Examples:

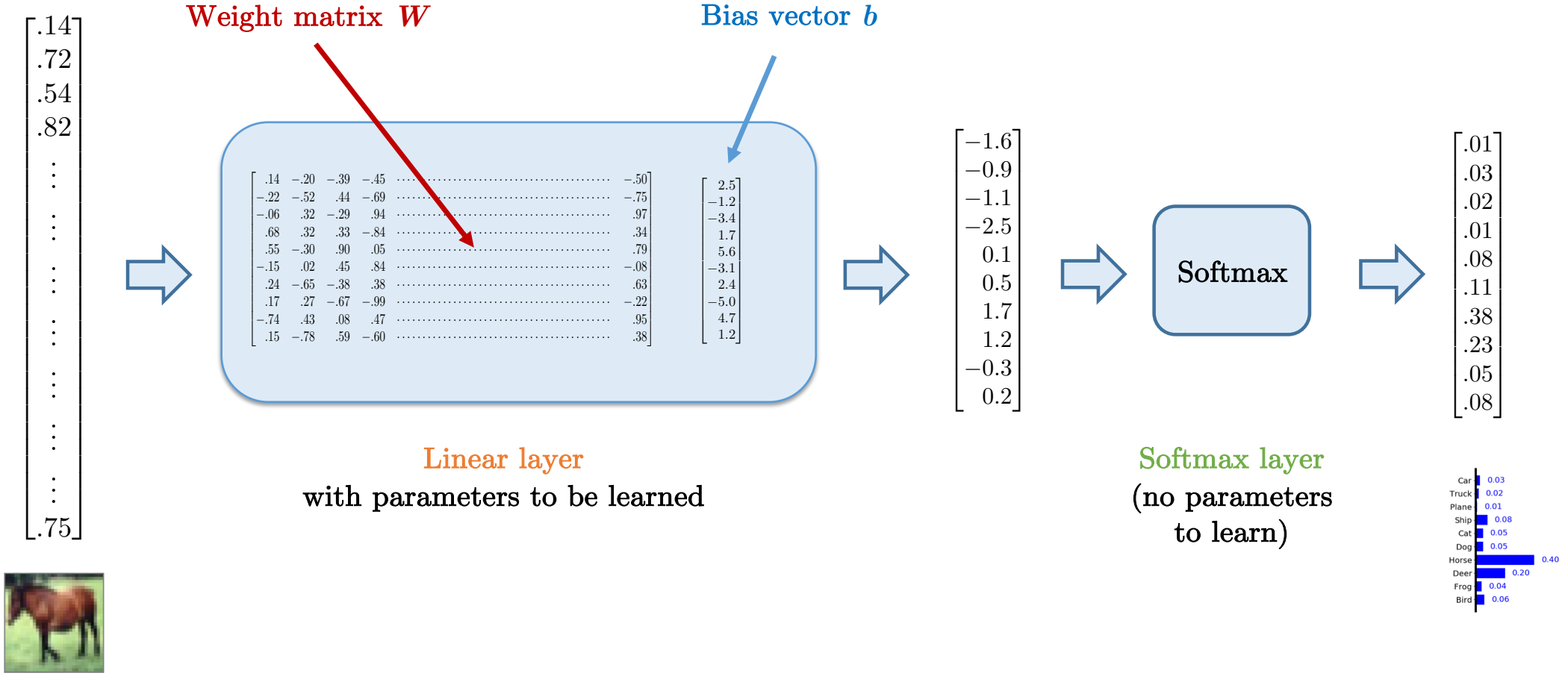
$$\text{Softmax}\left(\begin{bmatrix} -2.0 \\ -0.5 \\ 2.0 \\ 1.5 \end{bmatrix}\right) = \begin{bmatrix} .01 \\ .04 \\ .59 \\ .36 \end{bmatrix} \qquad \text{Softmax}\left(\begin{bmatrix} -2.0 \\ -0.5 \\ 7.0 \\ 1.5 \end{bmatrix}\right) = \begin{bmatrix} .00 \\ .00 \\ .99 \\ .01 \end{bmatrix}$$

Lab 02

- Softmax layer in PyTorch



One-layer neural network



Linear layer

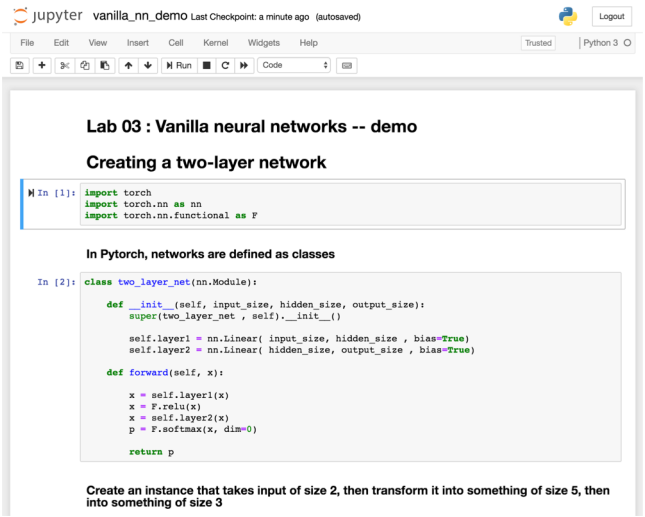
with parameters to be learned

Softmax layer

(no parameters to learn)

Lab 03

- Design a vanilla neural network in PyTorch :



jupyter vanilla_nn_demo Last Checkpoint: a minute ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

Lab 03 : Vanilla neural networks -- demo

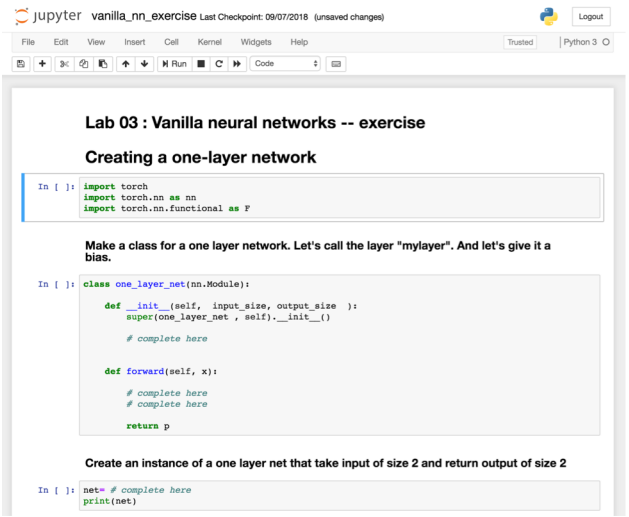
Creating a two-layer network

In [1]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

In Pytorch, networks are defined as classes

In [2]:
```
class two_layer_net(nn.Module):

    def __init__(self, input_size, hidden_size, output_size):
        super(two_layer_net , self).__init__()

        self.layer1 = nn.Linear( input_size, hidden_size , bias=True)
        self.layer2 = nn.Linear( hidden_size, output_size , bias=True)

    def forward(self, x):

        x = self.layer1(x)
        x = F.relu(x)
        x = self.layer2(x)
        p = F.softmax(x, dim=0)

        return p
```

Create an instance that takes input of size 2, then transform it into something of size 5, then into something of size 3



jupyter vanilla_nn_exercise Last Checkpoint: 09/07/2018 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

Lab 03 : Vanilla neural networks -- exercise

Creating a one-layer network

In [ ]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
```

Make a class for a one layer network. Let's call the layer "mylayer". And let's give it a bias.

In [ ]:
```
class one_layer_net(nn.Module):

    def __init__(self,  input_size, output_size  ):
        super(one_layer_net , self).__init__()

        # complete here


    def forward(self, x):

        # complete here
        # complete here

        return p
```

Create an instance of a one layer net that take input of size 2 and return output of size 2

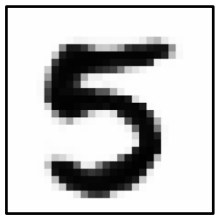
In [ ]:
```
net= # complete here
print(net)
```

Outline

- Image classification task
- Neural networks for image classification
- Forward pass for inference
- **Understanding linear layers**
- Understanding the backward pass
- Matrix formulation
- Mini-batch learning

MNIST dataset

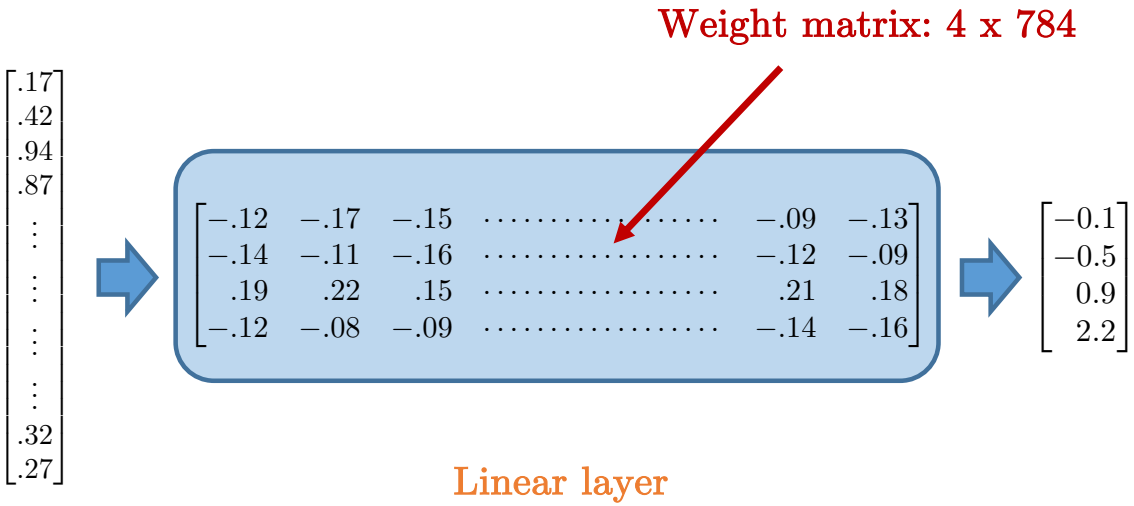
- Number of classes: 10
- Training set: 50,000 labelled images
- Test set: 10,000 labelled images



28 x 28 grayscale image

MNIST dataset

- For simplicity, we will pretend :
  - We have only 4 classes: Zero, One, Two, Three.
  - We have no bias b.



Linear layer
with parameters W to be learned

What linear layers do?

- A linear layer is a template matching algorithm !
- Each row of the weight matrix W is a template (encoding some data property):

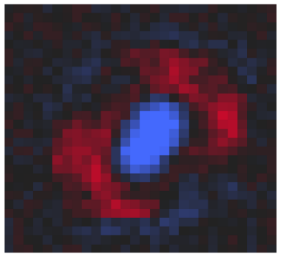
$$\begin{bmatrix} -.12 & -.17 & -.15 & \cdots & -.09 & -.13 \\ -.14 & -.11 & -.16 & \cdots & -.12 & -.09 \\ .19 & .22 & .15 & \cdots & .21 & .18 \\ -.12 & -.08 & -.09 & \cdots & -.14 & -.16 \end{bmatrix}$$

4 x 784 weight matrix W

Row zero reshaped as (28 x 28)

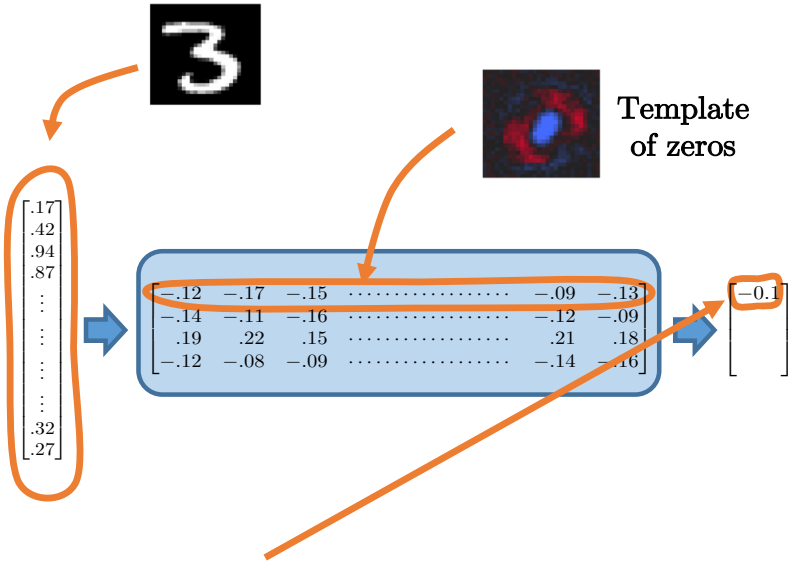
$$\begin{bmatrix} .14 & -.20 & -.39 & -.45 \\ -.94 & -.05 & .57 & .42 \\ .45 & -.46 & .97 & .08 \\ -.72 & .77 & -.81 & -.56 \\ -.34 & -.25 & -.39 & -.50 \end{bmatrix}$$

||

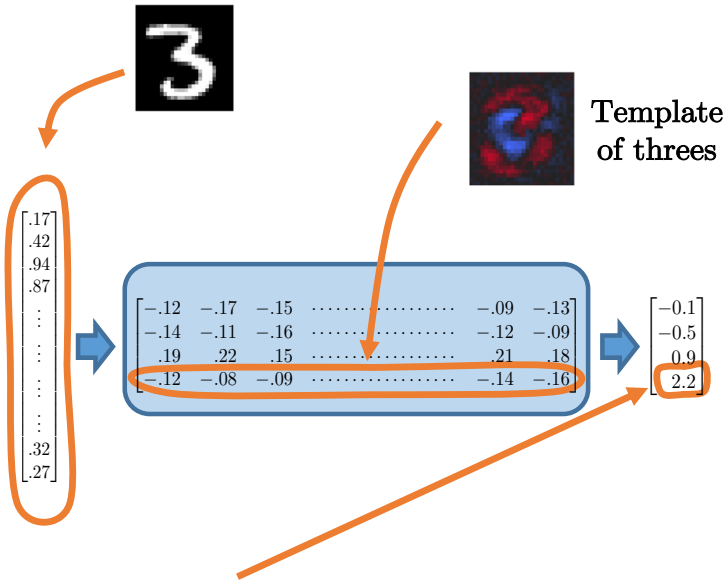


Template matching

- Template matching performs vector-vector multiplication between data and class template.
  - It produces a matching score between data and class template.



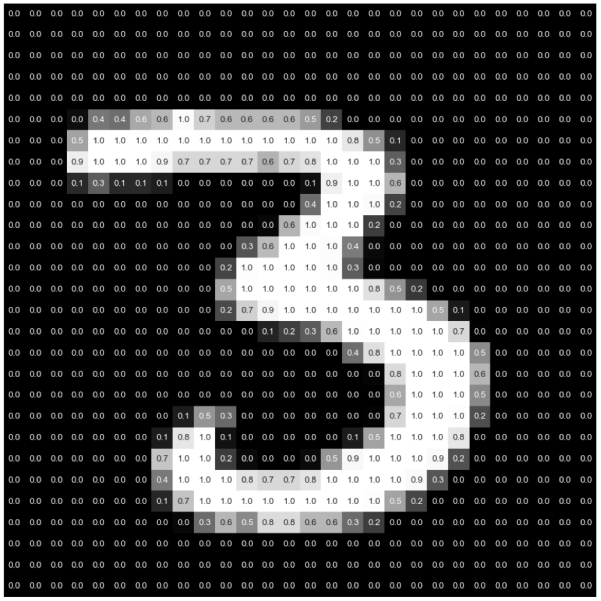
Matching score between the input data and the template of zeros

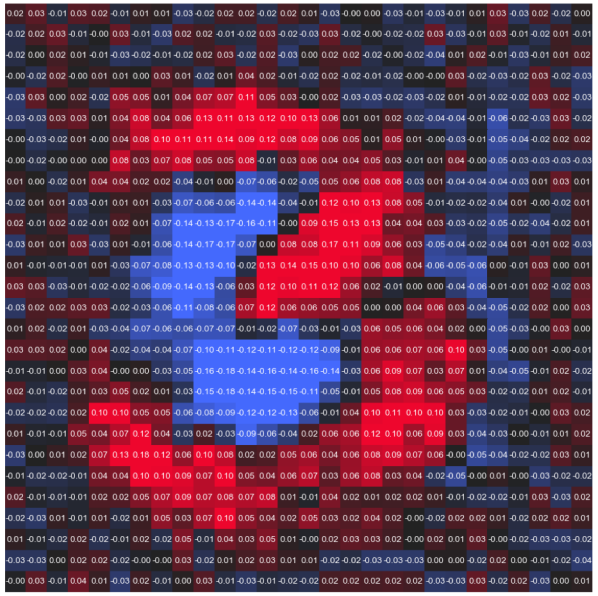


Matching score between the input data and the template of threes

Template matching

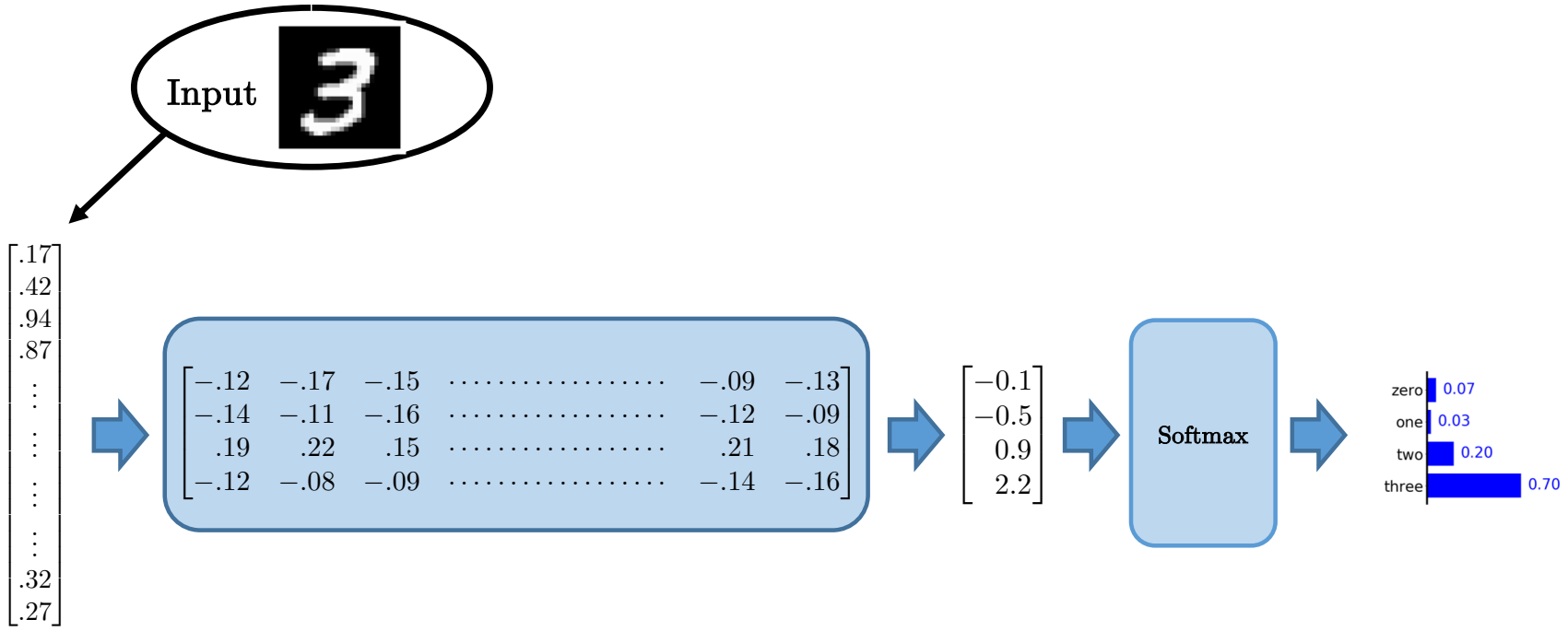
- Matching score = inner product of input image and class template





From score to probability

- Scores are converted into probabilities with the softmax layer :



Template matching

- Template matching property of DL architectures is a first principle.
- However, deep learning is more than template matching !
- Most research focus on finding the right architecture to solve complex tasks (that cannot be solved by simple template matching only).

Outline

- Image classification task
- Neural networks for image classification
- Forward pass for inference
- Understanding linear layers
- **Understanding the backward pass**
- Matrix formulation
- Mini-batch learning

How to learn the network parameters?

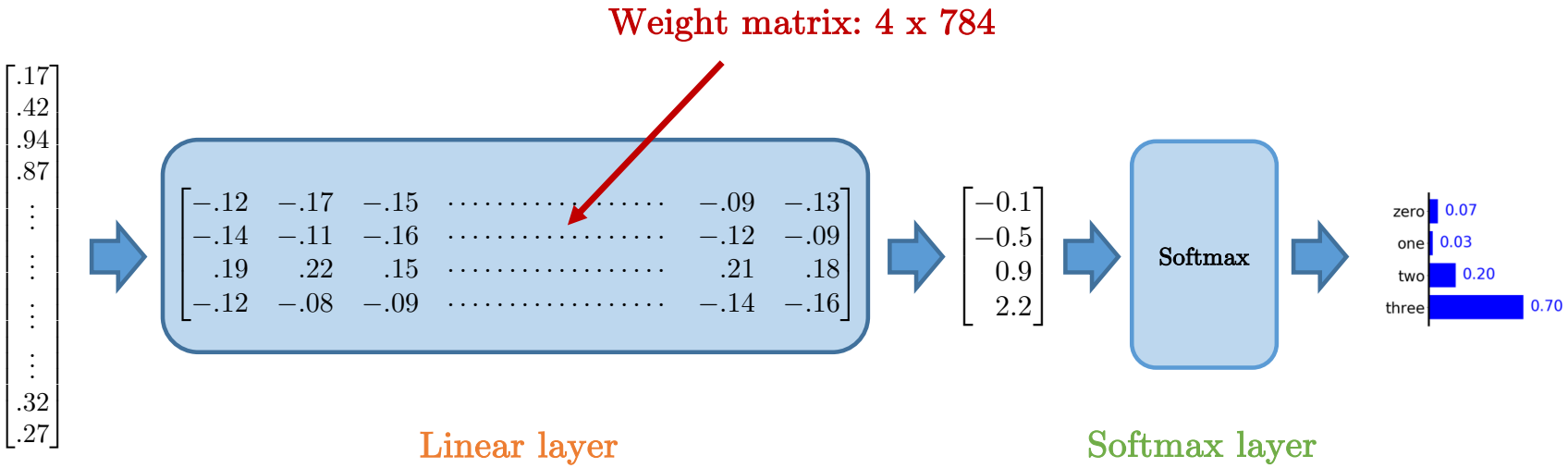
- So far, we have done inference : We have used the weight parameters that best classify images.
- We need to learn these parameters (here the weight and bias of the linear layer).
- Network parameters are learned by backpropagation:
  - A backward pass that backpropagates the error of prediction to adjust the weights on the network.
  - The update rule of the weights is :

$$\text{template} = \text{template} + \text{lr} \times (\text{error}) \times (\text{input picture})$$

$$W = W + \text{lr}\,(\text{error vector} \otimes \text{input picture})$$

Template learning

- For simplicity, we will still pretend :
  - We have only 4 classes: Zero, One, Two, Three.
  - We have no bias b.

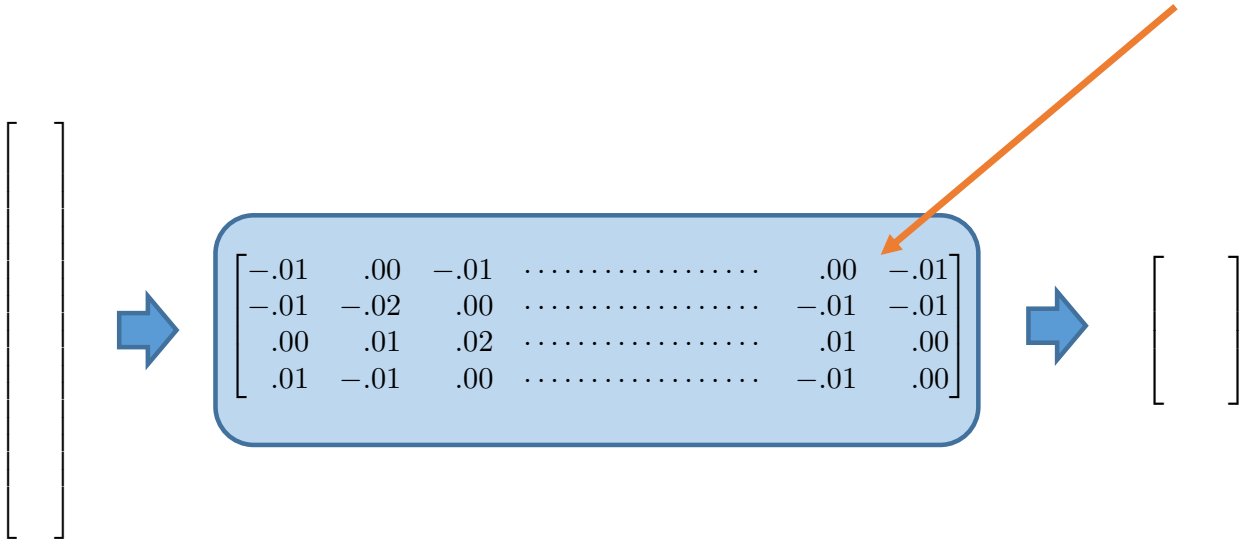


Linear layer
with parameters W to be learned

Softmax layer
(no parameters to learn)

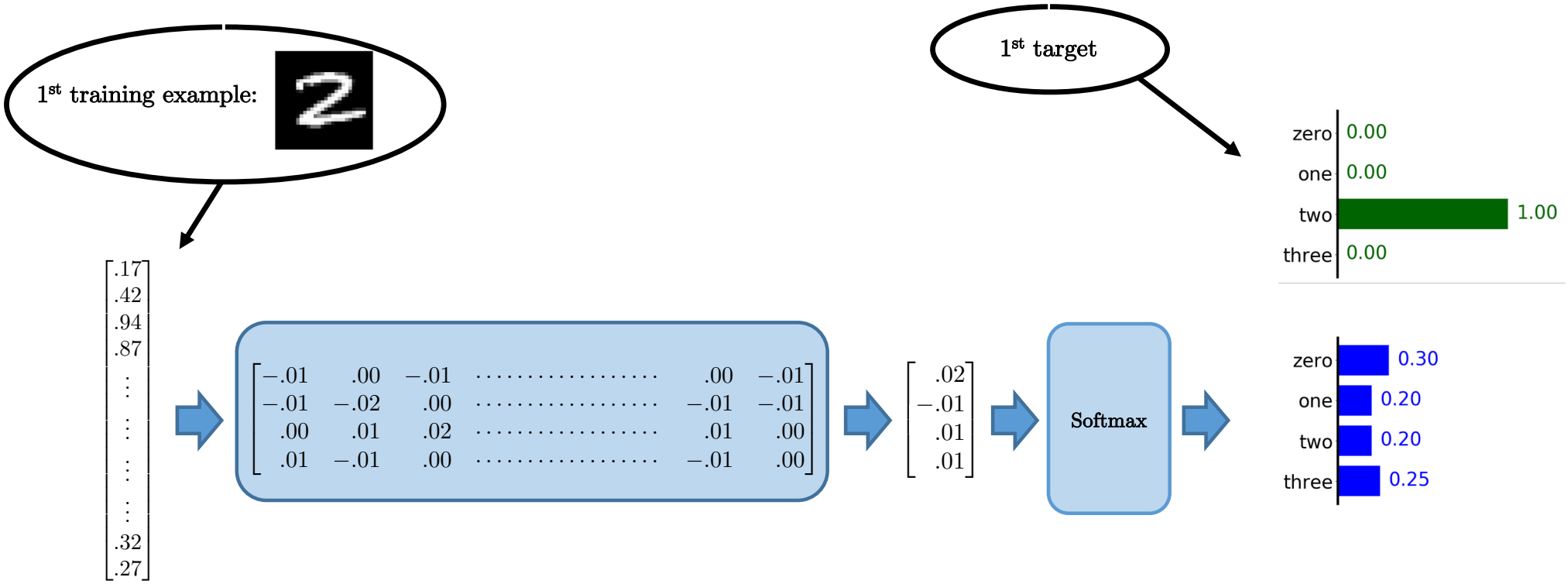
Template learning

- How does the network learn good templates?
- We will use an iterative step that will update progressively the templates to minimize the error of prediction.
- Initialization:
  - At the beginning of the training the weight matrix is initialized with small random numbers.



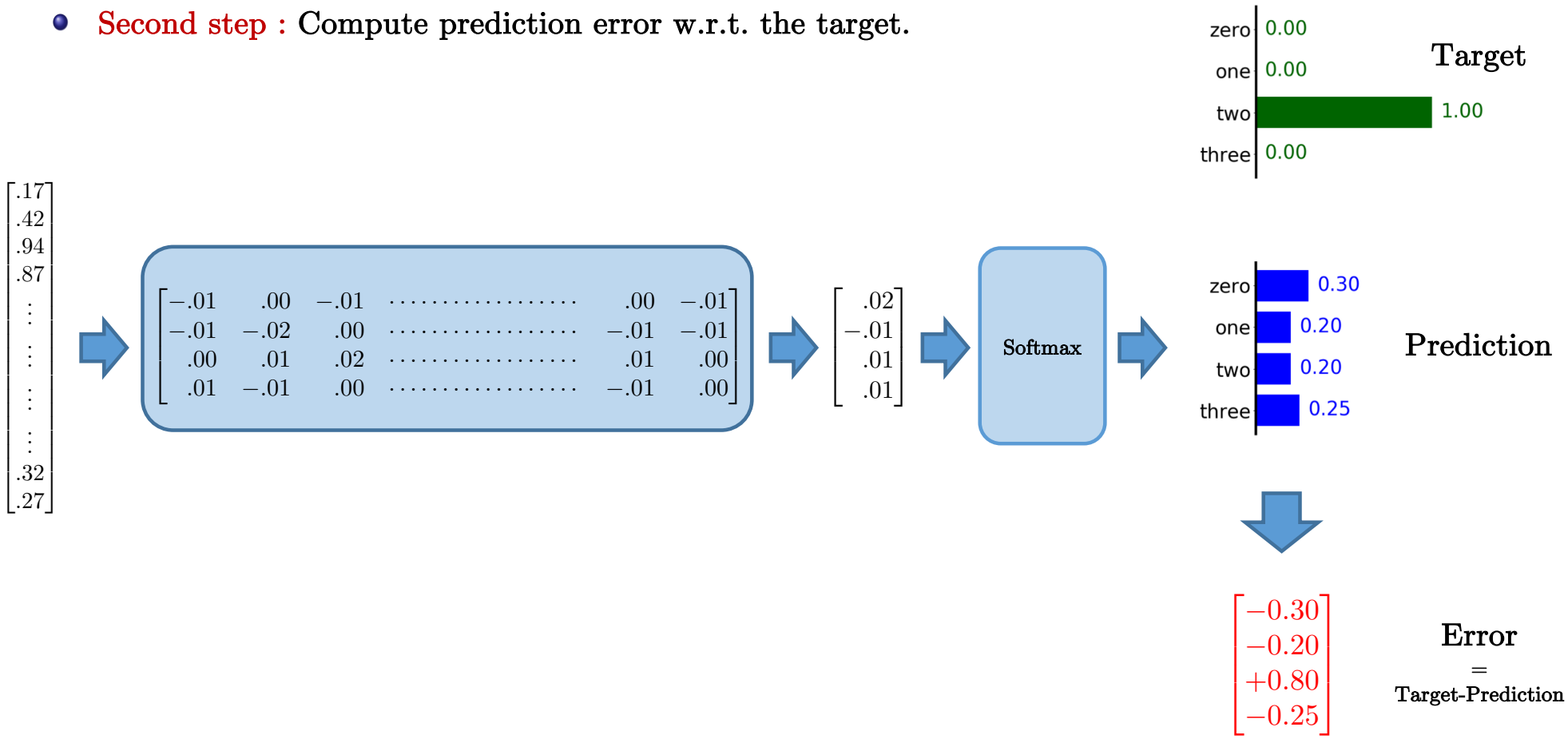
Forward pass

- **First step :** Compute the class probabilities for the 1st training data.



Prediction error

- Second step : Compute prediction error w.r.t. the target.



Template update

- Third step : Update templates

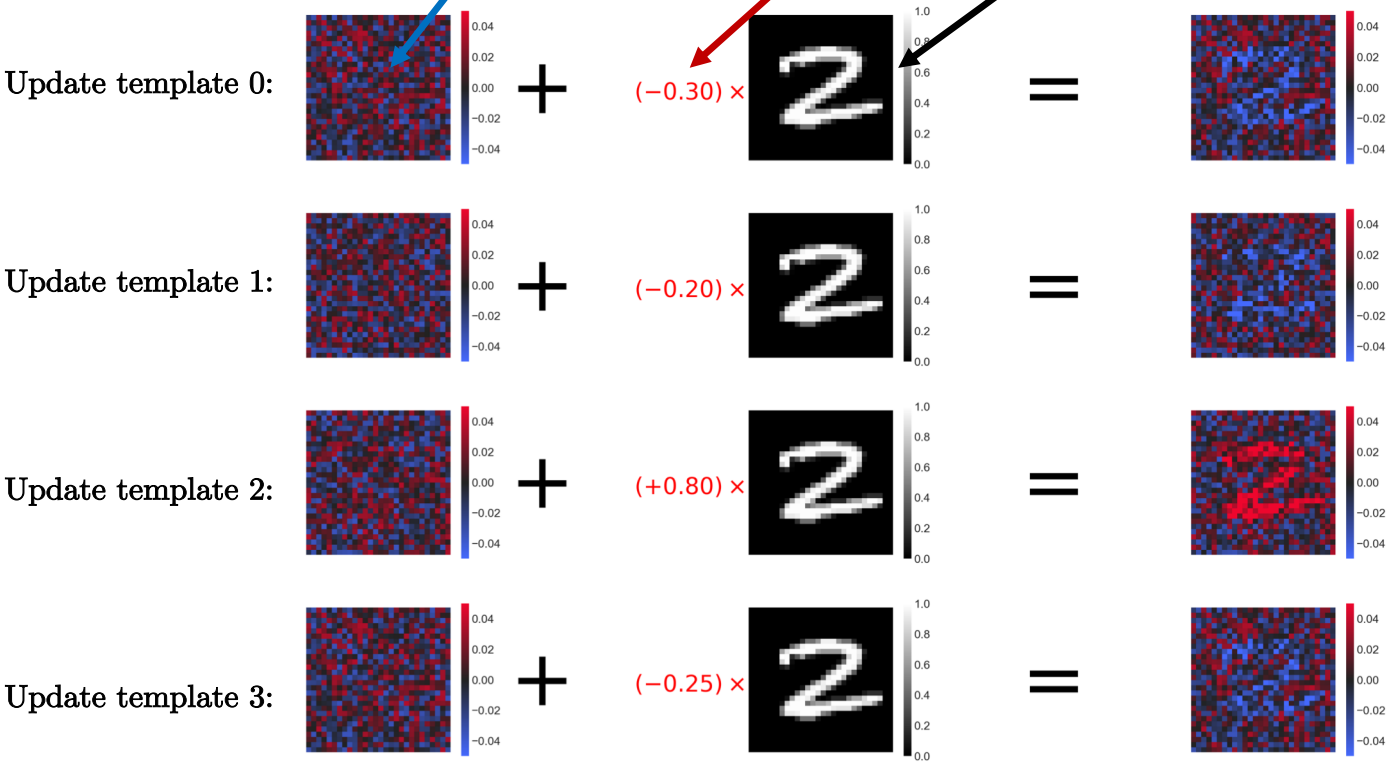
$$\text{template} = \text{template} + \text{lr} \times (\text{error}) \times (\text{input picture})$$

  - Each time the network is presented with a picture, let say: 
  - It adds this picture to the correct class template (here class 2).
  - And removes it from the other class templates (here classes 0, 1, 3).

Template update

- Applying the template update:

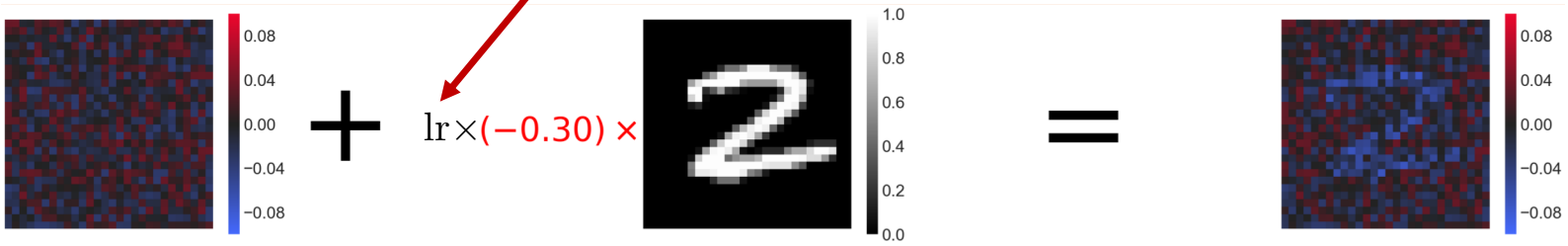
$$\text{template} = \text{template} + \text{lr} \times (\text{error}) \times (\text{input picture})$$



Learning rate

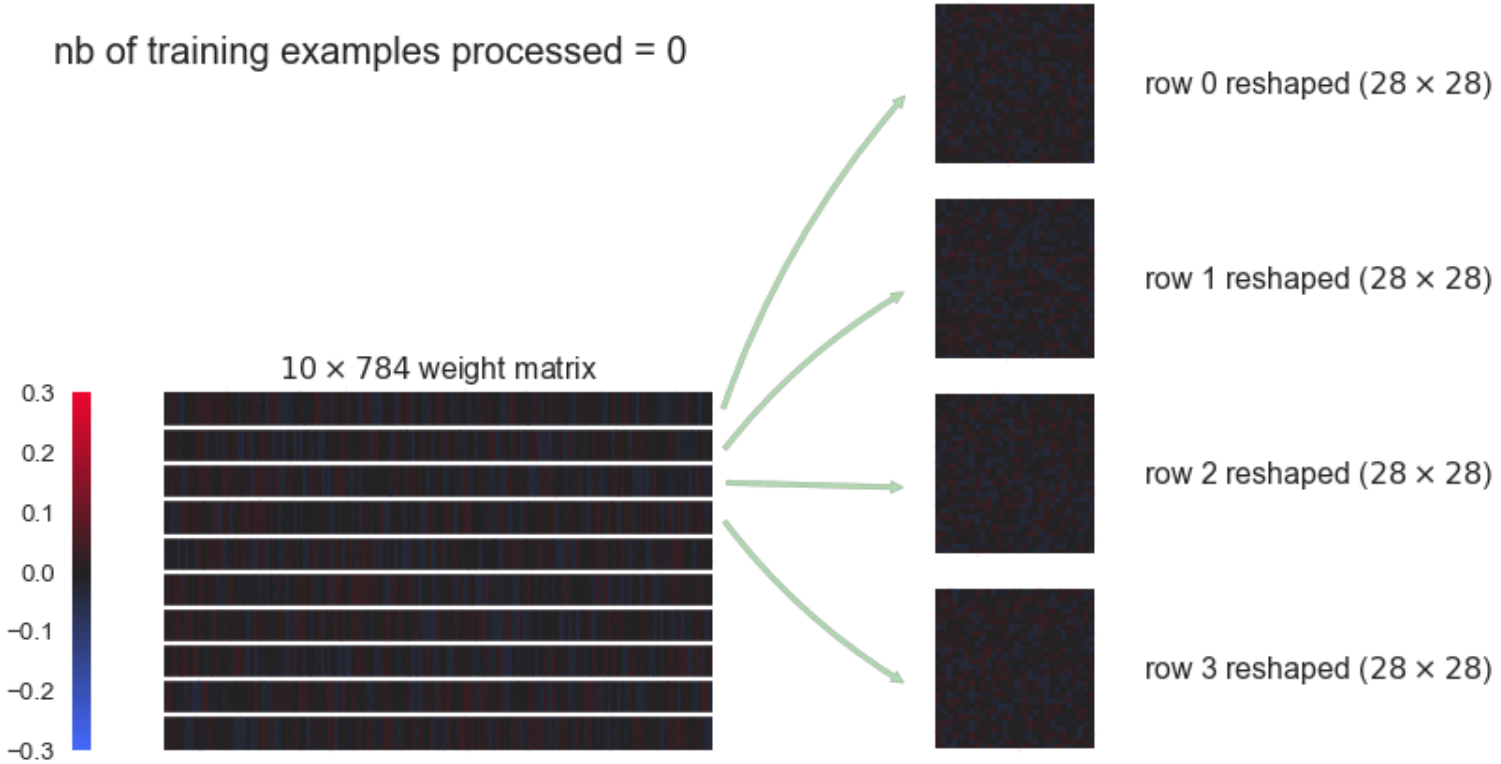
- The learning rate (lr) is a very important hyper-parameter.
- The learning rate controls the learning speed:
  - Too small : Learning is too long.
  - Too large: Learning is too fast and it does not work.
- Typical choice to update the templates is

$$\text{lr} \approx 0.01$$



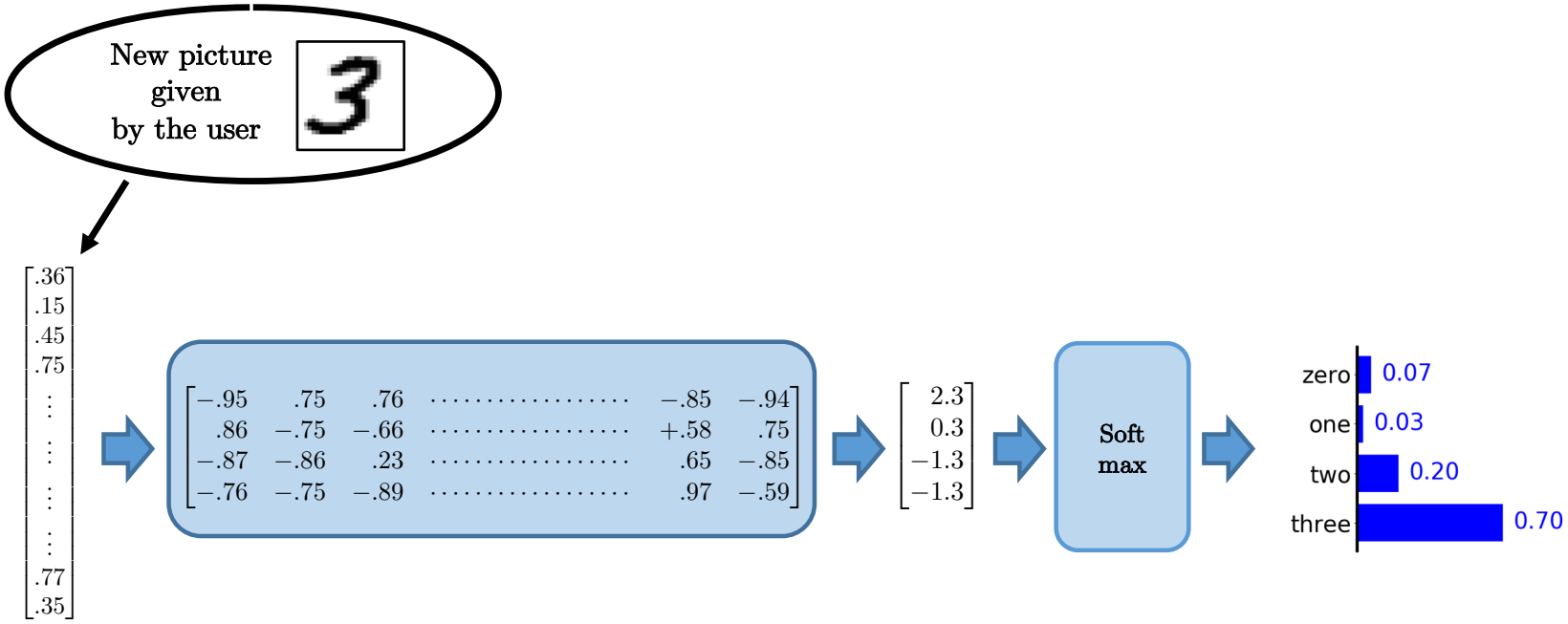
Iterate

- Iterate this process with all training data and multiple times:
  - After many iterations, the templates look good :



Network ready for inference

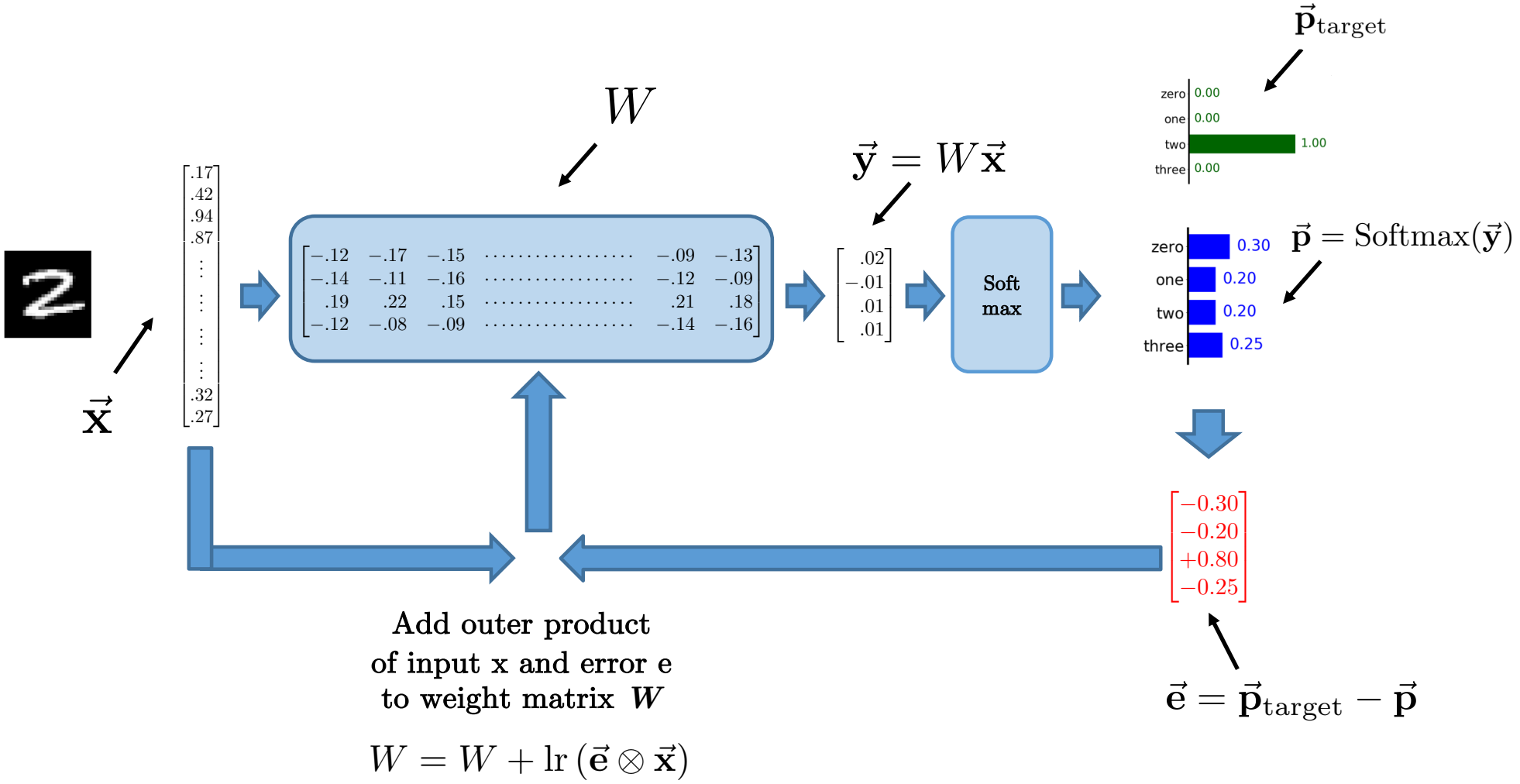
- After many iterations, the templates do not update anymore, the training is finished.
- The network is ready for inference for new data.



Outline

- Image classification task
- Neural networks for image classification
- Forward pass for inference
- Understanding linear layers
- Understanding the backward pass
- **Matrix formulation**
- Mini-batch learning

Matrix notation



Matrix formulation of outer product

$$\vec{\mathbf{e}} \otimes (\text{input picture}) = \begin{bmatrix} -0.30 \\ -0.20 \\ +0.80 \\ -0.25 \end{bmatrix} \begin{bmatrix} .17 & .42 & .94 & \cdots & \cdots & .32 & .27 \end{bmatrix}$$

$$= \begin{bmatrix} -0.30 \times \text{input picture} \\ -0.20 \times \text{input picture} \\ +0.80 \times \text{input picture} \\ -0.25 \times \text{input picture} \end{bmatrix}$$

Forward and backward pass

- Forward pass:

$$\vec{\mathbf{y}} = W\vec{\mathbf{x}} \qquad \text{Compute the scores}$$

$$\vec{\mathbf{p}} = \text{Softmax}(\vec{\mathbf{y}}) \qquad \text{Compute the probabilities}$$

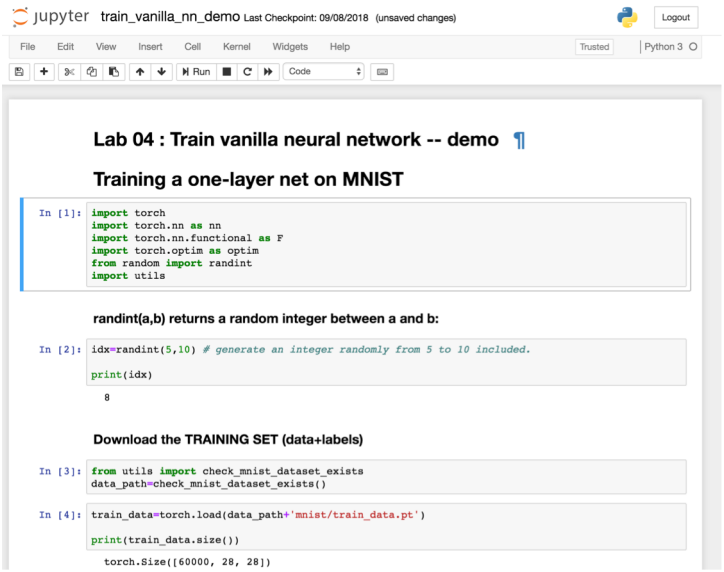
- Backward pass:

$$\vec{\mathbf{e}} = \vec{\mathbf{p}}_{\text{target}} - \vec{\mathbf{p}} \qquad \text{Compute the error}$$

$$W = W + \text{lr}\,(\vec{\mathbf{e}} \otimes \vec{\mathbf{x}}) \qquad \text{Update the templates}$$

Lab 04

- Train a one layer net


jupyter train_vanilla_nn_demo Last Checkpoint: 09/08/2018 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

Lab 04 : Train vanilla neural network -- demo ¶

Training a one-layer net on MNIST

In [1]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from random import randint
import utils
```

randint(a,b) returns a random integer between a and b:

In [2]:
```
idx=randint(5,10) # generate an integer randomly from 5 to 10 included.

print(idx)
```

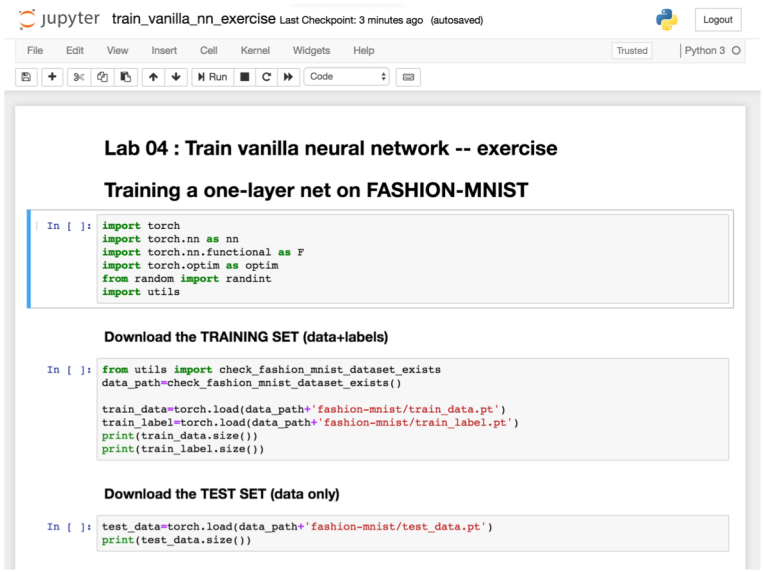
```
8
```

Download the TRAINING SET (data+labels)

In [3]:
```
from utils import check_mnist_dataset_exists
data_path=check_mnist_dataset_exists()
```

In [4]:
```
train_data=torch.load(data_path+'mnist/train_data.pt')

print(train_data.size())
```

```
torch.Size([60000, 28, 28])
```


jupyter train_vanilla_nn_exercise Last Checkpoint: 3 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

Lab 04 : Train vanilla neural network -- exercise

Training a one-layer net on FASHION-MNIST

In [ ]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from random import randint
import utils
```

Download the TRAINING SET (data+labels)

In [ ]:
```
from utils import check_fashion_mnist_dataset_exists
data_path=check_fashion_mnist_dataset_exists()

train_data=torch.load(data_path+'fashion-mnist/train_data.pt')
train_label=torch.load(data_path+'fashion-mnist/train_label.pt')
print(train_data.size())
print(train_label.size())
```

Download the TEST SET (data only)

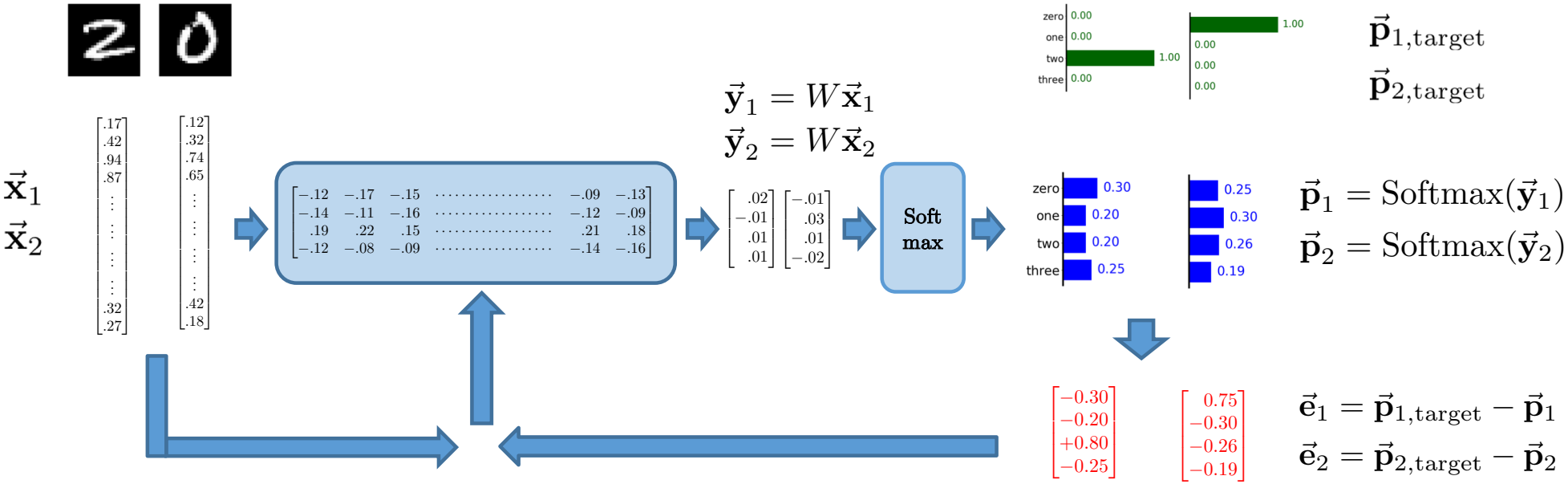
In [ ]:
```
test_data=torch.load(data_path+'fashion-mnist/test_data.pt')
print(test_data.size())
```

Outline

- Image classification task
- Neural networks for image classification
- Forward pass for inference
- Understanding linear layers
- Understanding the backward pass
- Matrix formulation
- **Mini-batch learning**

Mini-batch

- Rather than processing one image data at a time, let us process multiple images at a time :



$$W = W + \text{lr}\,(\vec{\mathbf{e}}_1 \otimes \vec{\mathbf{x}}_1) + \text{lr}\,(\vec{\mathbf{e}}_2 \otimes \vec{\mathbf{x}}_2)$$

Mini-batch matrix formulation

$$\vec{x}_1 \quad \vec{x}_2 \quad \Downarrow \quad X$$

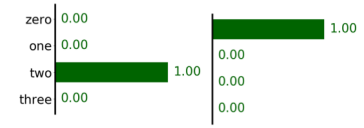
$$\begin{bmatrix} .17 \\ .42 \\ .94 \\ .87 \\ \vdots \\ .32 \\ .27 \end{bmatrix} \quad \begin{bmatrix} .12 \\ .32 \\ .74 \\ .65 \\ \vdots \\ .42 \\ .18 \end{bmatrix}$$

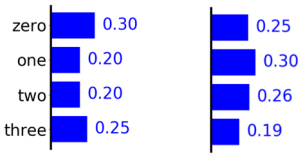
$$\begin{bmatrix} -.12 & -.17 & -.15 & \cdots & -.09 & -.13 \\ -.14 & -.11 & -.16 & \cdots & -.12 & -.09 \\ .19 & .22 & .15 & \cdots & .21 & .18 \\ -.12 & -.08 & -.09 & \cdots & -.14 & -.16 \end{bmatrix}$$

$$\vec{y}_1 = W\vec{x}_1$$
$$\vec{y}_2 = W\vec{x}_2$$
$$\Downarrow$$
$$Y = WX$$

$$\begin{bmatrix} .02 \\ -.01 \\ .01 \\ .01 \end{bmatrix} \quad \begin{bmatrix} -.01 \\ .03 \\ .01 \\ -.02 \end{bmatrix}$$

Soft max





$$\begin{bmatrix} -0.30 \\ -0.20 \\ +0.80 \\ -0.25 \end{bmatrix} \quad \begin{bmatrix} 0.75 \\ -0.30 \\ -0.26 \\ -0.19 \end{bmatrix}$$

$$\vec{p}_{1,\text{target}}$$
$$\vec{p}_{2,\text{target}}$$
$$\Downarrow$$
$$P_{\text{target}}$$

$$\vec{p}_1 = \text{Softmax}(\vec{y}_1)$$
$$\vec{p}_2 = \text{Softmax}(\vec{y}_2)$$
$$\Downarrow$$
$$\text{P} = \text{Softmax}(\text{Y})$$

$$\vec{e}_1 = \vec{p}_{1,\text{target}} - \vec{p}_1$$
$$\vec{e}_2 = \vec{p}_{2,\text{target}} - \vec{p}_2$$
$$\Downarrow$$
$$E = P_{\text{target}} - P$$

$$W = W + \text{lr}\,(\vec{e}_1 \otimes \vec{x}_1 + \vec{e}_2 \otimes \vec{x}_2)$$
$$\Downarrow$$
$$W = W + \text{lr}\,(EX^T)$$

Sum of outer product can be written as a mat-mat multiplication.

Matrix formulation of outer product

$$\vec{\mathbf{e}}_1 \otimes \vec{\mathbf{x}}_1 + \vec{\mathbf{e}}_2 \otimes \vec{\mathbf{x}}_2 = \begin{bmatrix} -0.30 \\ -0.20 \\ +0.80 \\ -0.25 \end{bmatrix} \begin{bmatrix} .17 & .42 & .94 & .87 & \cdots & \cdots & \cdots & \cdots & .32 & .27 \end{bmatrix} + \begin{bmatrix} +0.75 \\ -0.30 \\ -0.26 \\ -0.19 \end{bmatrix} \begin{bmatrix} .12 & .32 & .74 & .65 & \cdots & \cdots & \cdots & \cdots & .42 & .18 \end{bmatrix}$$

$$\Downarrow$$

$$EX^T = \begin{bmatrix} -0.30 & +0.75 \\ -0.20 & -0.30 \\ +0.80 & -0.26 \\ -0.25 & -0.19 \end{bmatrix} \begin{bmatrix} .17 & .42 & .94 & .87 & \cdots & \cdots & \cdots & \cdots & .32 & .27 \\ .12 & .32 & .74 & .65 & \cdots & \cdots & \cdots & \cdots & .42 & .18 \end{bmatrix}$$

Forward and backward pass for mini-batch

- **Forward pass:**

$$Y = WX \qquad \text{Compute the scores}$$

$$\mathrm{P} = \mathrm{Softmax}(\mathrm{Y}) \qquad \text{Compute the probabilities}$$

- **Backward pass:**

$$E = P_{\text{target}} - P \qquad \text{Compute the error}$$

$$W = W + \mathrm{lr}\left(EX^T\right) \qquad \text{Update the templates}$$

Serial vs Batch

- What is the difference between serial and batch learning?
  - Serial learning :

is first processed by

$$\begin{bmatrix} -.01 & .00 & -.01 & \cdots & .00 & -.01 \\ -.01 & -.02 & .00 & \cdots & -.01 & -.01 \\ .00 & .01 & .02 & \cdots & .01 & .00 \\ .01 & -.01 & .00 & \cdots & -.01 & .00 \end{bmatrix}$$

The errors is computed and then the weights are updated.

Then

is processed by

$$\begin{bmatrix} -.06 & .03 & .03 & \cdots & .00 & -.01 \\ .08 & .03 & -.04 & \cdots & -.01 & -.01 \\ -.04 & -.09 & .05 & \cdots & .01 & .15 \\ .12 & -.05 & .03 & \cdots & -.01 & .18 \end{bmatrix}$$

The errors is computed and then the weight are updated.

  - Serial mode : The weights have been updated twice.
    - When the second image is processed, it benefits of better freshly updated weights. As a consequence the network trains fast.

Serial vs Batch

- Batch learning :

are both processed by

$$\begin{bmatrix} -.01 & .00 & -.01 & \cdots & .00 & -.01 \\ -.01 & -.02 & .00 & \cdots & -.01 & -.01 \\ .00 & .01 & .02 & \cdots & .01 & .00 \\ .01 & -.01 & .00 & \cdots & -.01 & .00 \end{bmatrix}$$

The errors are computed and then the weight are updated.

Batch mode : The weights have been updated only once.

Serial vs Batch

- No batch (single data): So processing one image after the other, and updated the weights each time, seems clearly better.

Batch size = 1

- Mini-batch: On the other hand, processing images by batches of e.g. 200, and updating the weights only every 200 images, is also an attractive option for GPUs.

Batch size = 200

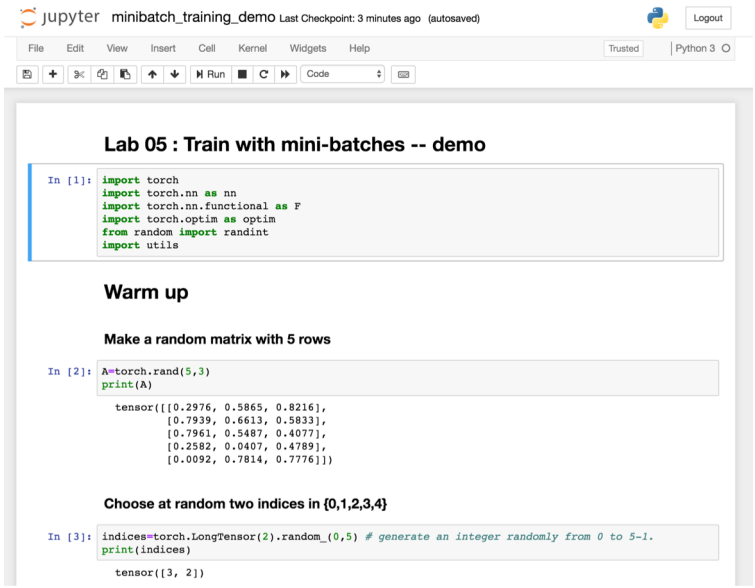
- Full batch: What about doing batch of 60,000 (all training set) and processing everything in parallel? Not good:
  - Slow update: updating the weights only every 60,000 images.
  - Overfitting issue

Batch size = 60,000

Best choice!

Lab 05

- Train a net with minibatch


jupyter minibatch_training_demo Last Checkpoint: 3 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

Lab 05 : Train with mini-batches -- demo

```
In [1]: import torch
        import torch.nn as nn
        import torch.nn.functional as F
        import torch.optim as optim
        from random import randint
        import utils
```

Warm up

Make a random matrix with 5 rows

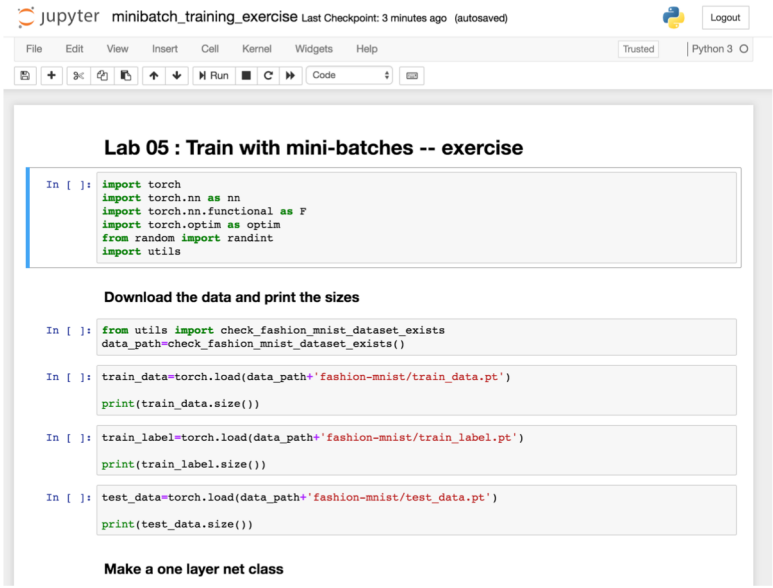
```
In [2]: A=torch.rand(5,3)
        print(A)
```

```
tensor([[0.2976, 0.5865, 0.8216],
        [0.7939, 0.6613, 0.5833],
        [0.7961, 0.5487, 0.4077],
        [0.2582, 0.0407, 0.4789],
        [0.0092, 0.7814, 0.7776]])
```

Choose at random two indices in {0,1,2,3,4}

```
In [3]: indices=torch.LongTensor(2).random_(0,5) # generate an integer randomly from 0 to 5-1.
        print(indices)
```

```
tensor([3, 2])
```


jupyter minibatch_training_exercise Last Checkpoint: 3 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

Lab 05 : Train with mini-batches -- exercise

```
In [ ]: import torch
        import torch.nn as nn
        import torch.nn.functional as F
        import torch.optim as optim
        from random import randint
        import utils
```

Download the data and print the sizes

```
In [ ]: from utils import check_fashion_mnist_dataset_exists
        data_path=check_fashion_mnist_dataset_exists()
```

```
In [ ]: train_data=torch.load(data_path+'fashion-mnist/train_data.pt')

        print(train_data.size())
```

```
In [ ]: train_label=torch.load(data_path+'fashion-mnist/train_label.pt')

        print(train_label.size())
```

```
In [ ]: test_data=torch.load(data_path+'fashion-mnist/test_data.pt')

        print(test_data.size())
```

Make a one layer net class

Questions?