Lecture „Neural Networks and Memristive Hardware Acclerators“
Prof. Dr. phil. nat. habil. Ronald Tetzlaff
Winter semester 2024/25

Lecture 5: PyTorch Tutorial

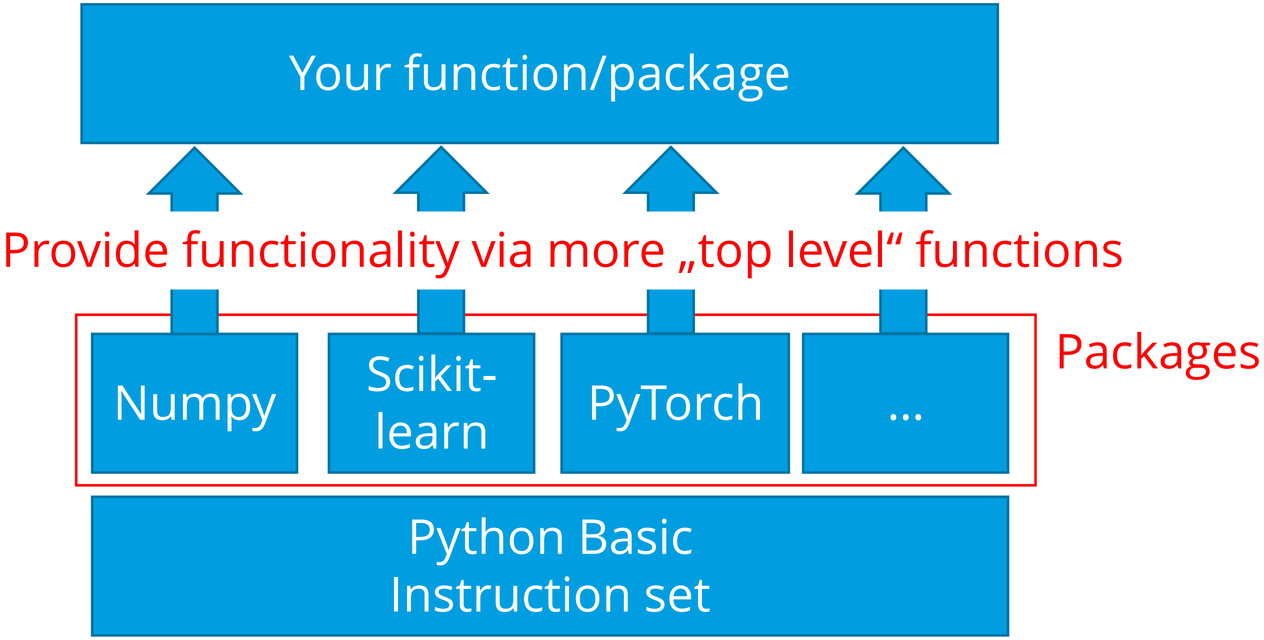
Dipl.-Ing. Steffen Seitz

PyTorch Tutorial : Study goals

- You understand what PyTorch can be used for.
- You know what a computation graph is.
- You are able to implement your own Neural Network architecture based on PyTorch.
- You know about training and evaluation mode in PyTorch.

PyTorch as a Module

Packages/Modules are nothing else than **classes** with **functions** written by other users free of charge! PyTorch is maintained by the „Meta“ AI Research Team.



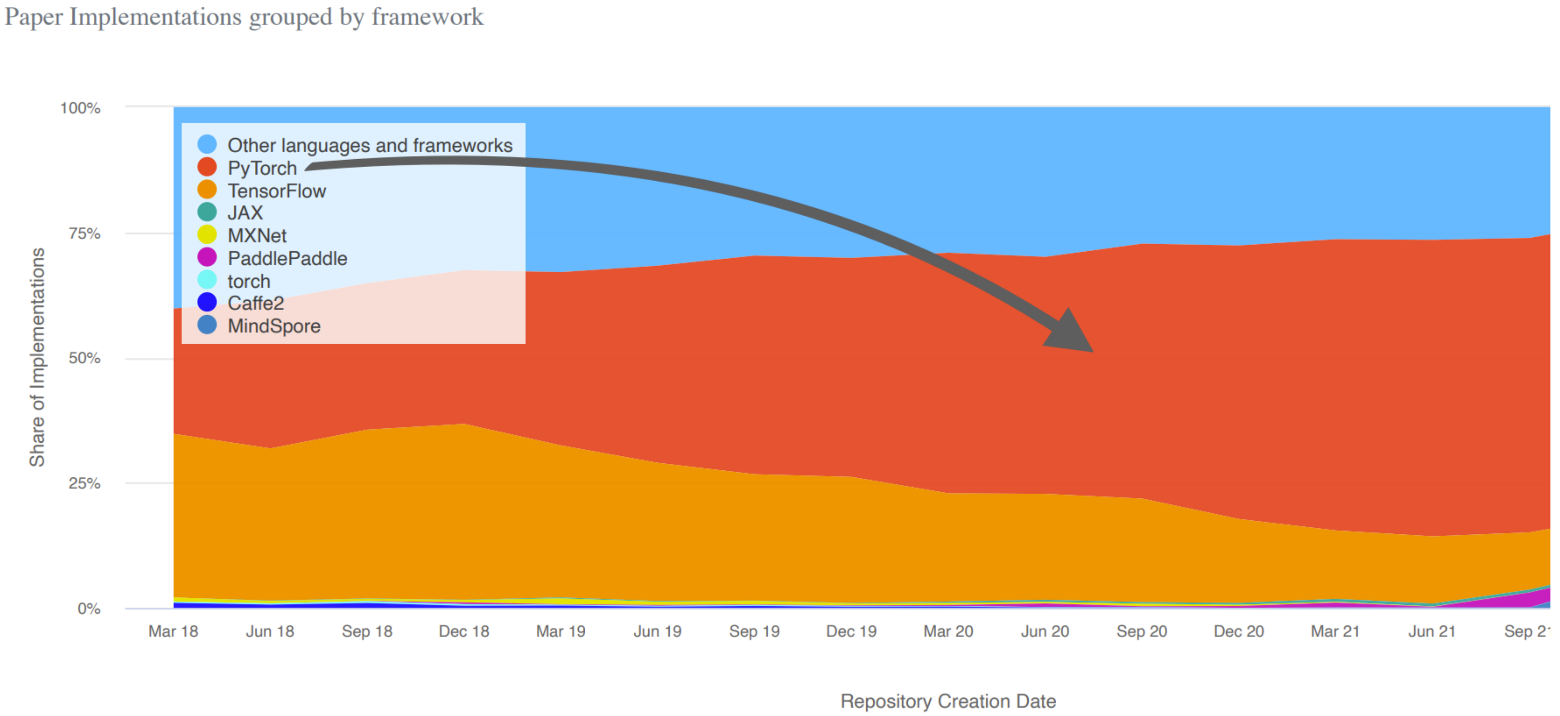
What is PyTorch?

PyTorch is a scripting language that is very „pythonic“ in terms of readability.

Tensor Library

Deep Learning Library

Automatic differentiation engine



PyTorch as a Tensor Computing Library

Scalar (rank-0 tensor)

```python
import torch

a = torch.tensor(1.)
a.shape
```

```
torch.Size([])
```

Vector (rank-1 tensor)

```python
a = torch.tensor([1., 2., 3.])
a.shape
```

```
torch.Size([3])
```

Matrix (rank-2 tensor)

```python
a = torch.tensor([[1., 2., 3.],
                  [2., 3., 4.]])
a.shape
```

```
torch.Size([2, 3])
```

torch.tensor ≈ numpy.array

Color Images are a Stack of Matrices

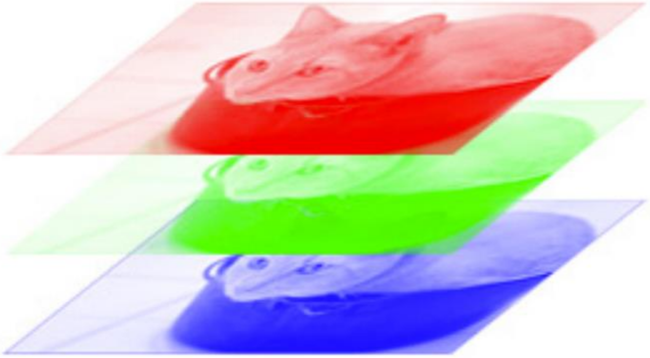


Image Source: https://code.tutsplus.com/tutorials/create-a-retro-crt-distortion-effect-using-rgb-shifting--active-3359

3D tensor
(rank-3 tensor)

```
a = torch.tensor([[[1., 2., 3.],
                   [2., 3., 4.]],
                  [[5., 6., 7.],
                   [8., 9., 10.]]])
a.shape
```

```
torch.Size([2, 2, 3])
```

Stack of Colour Images = 4D tensor (rank-4 tensor)

```
b = torch.stack((a, a))
b.shape
```

```
torch.Size([2, 2, 2, 3])
```

```
b
```

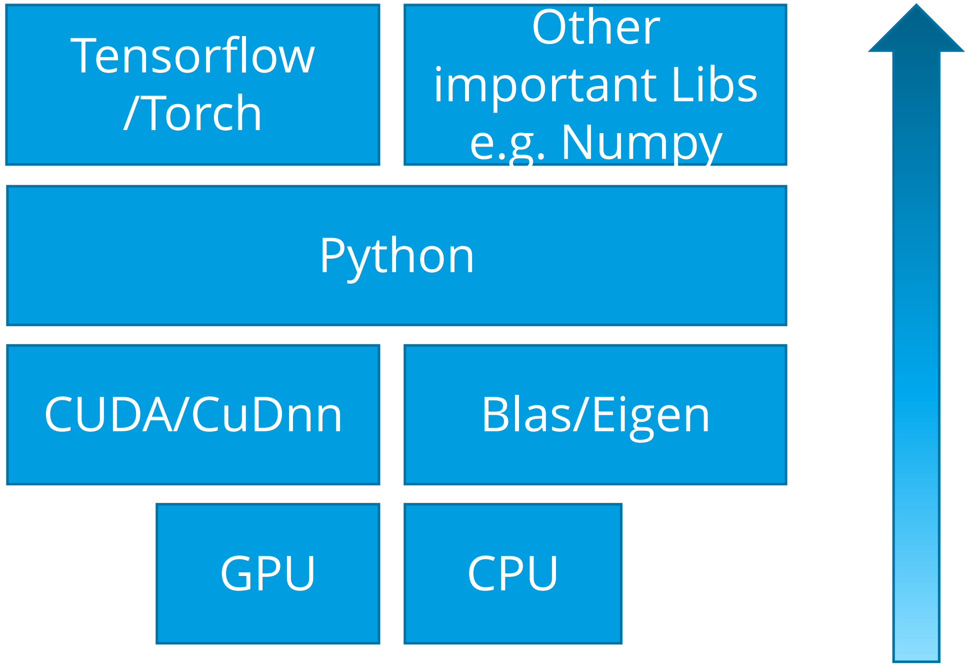
```
tensor([[[[ 1.,  2.,  3.],
          [ 2.,  3.,  4.]],

         [[ 5.,  6.,  7.],
          [ 8.,  9., 10.]]],


        [[[ 1.,  2.,  3.],
          [ 2.,  3.,  4.]],

         [[ 5.,  6.,  7.],
          [ 8.,  9., 10.]]]])
```

PyTorch and CUDA

PyTorch has CUDA Support!

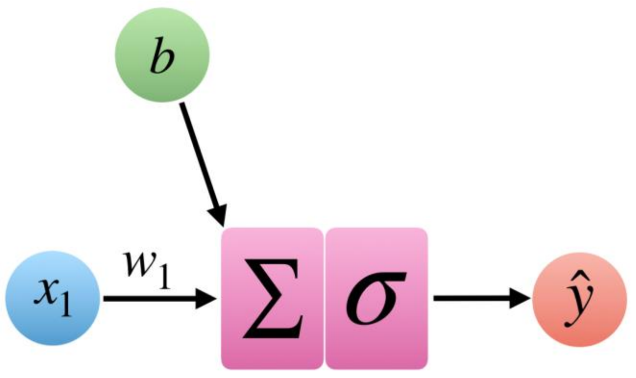
CUDA is a library by Nvidia used to speed up deep learning methods on GPUs.

It enables the transfer of data from the memory to the GPU accessible memory

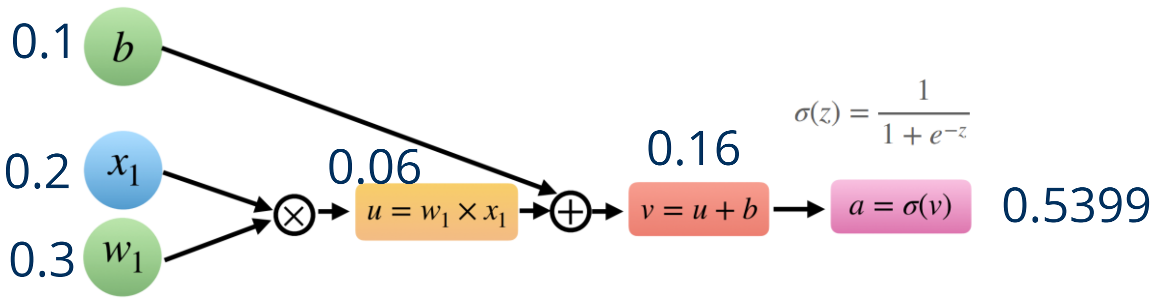


Computation Graph (Forward)

Perceptron (Forward Path)



Computational Graph (Forward Path)

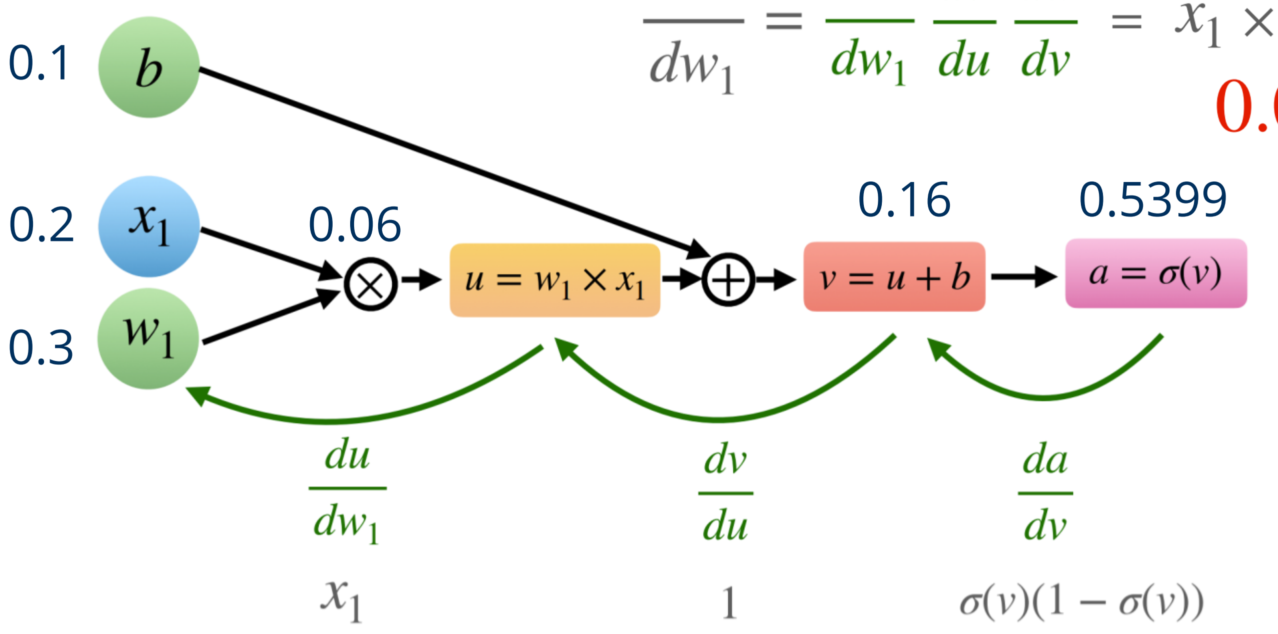


Forward pass in PyTorch

```
b = torch.tensor(0.1)
x1 = torch.tensor(0.2)
w1 = torch.tensor(0.3)

u = w1*x1
v = u + b
a = torch.sigmoid(v)
a
```

```
tensor(0.5399)
```

Computation Graph (Backward)



$$\frac{da}{dw_1} = \frac{du}{dw_1}\frac{dv}{du}\frac{da}{dv} = x_1 \times \sigma(v)(1-\sigma(v))$$

0.0497

Forward pass in PyTorch

```
b = torch.tensor(0.1)
x1 = torch.tensor(0.2)
w1 = torch.tensor(0.3)

u = w1*x1
v = u + b
a = torch.sigmoid(v)
a
```

```
tensor(0.5399)
```

Backward pass in PyTorch

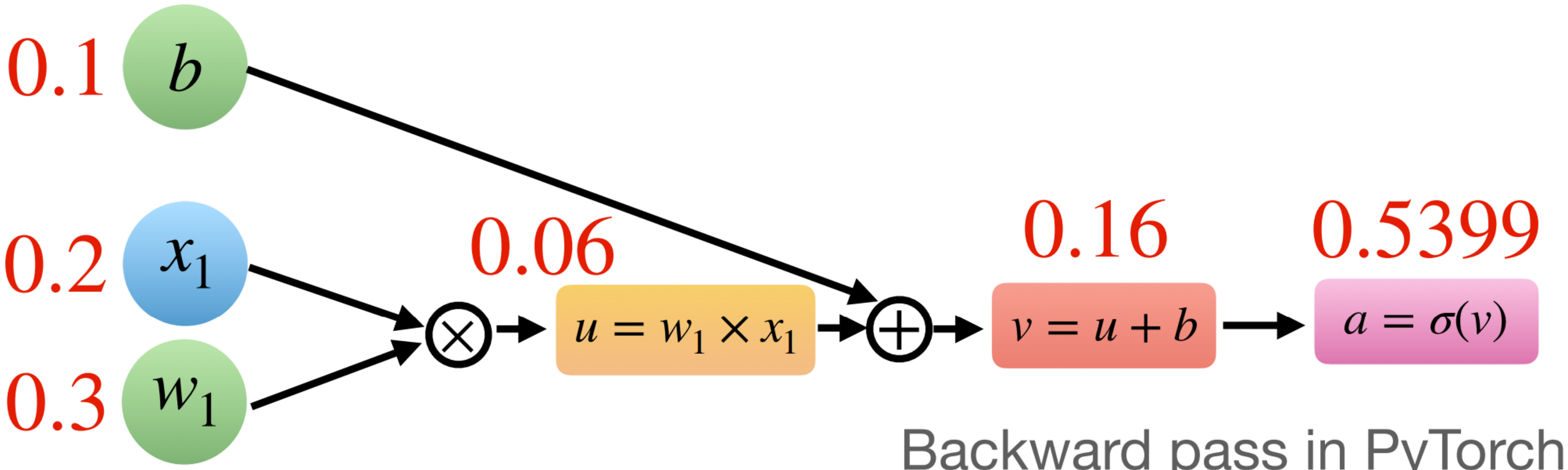
$$\frac{da}{dw_1} =$$

```
a * (1-a) * x1
```

```
tensor(0.0497)
```

It would be nice if we would be able to derive this automatically! → **PyTorch Autograd!**

Autograd




```
b = torch.tensor(0.1)
x1 = torch.tensor(0.2)
w1 = torch.tensor(0.3, requires_grad=True)

u = w1*x1
v = u + b
a = torch.sigmoid(v)
a
```

```
tensor(0.5399, grad_fn=<SigmoidBackward0>)
```

Backward pass in PyTorch automatically!

$$\frac{da}{dw_1} =$$

```
from torch.autograd import grad

grad(a, w1)
```

```
(tensor(0.0497),)
```

Exercise „Neural Network Implementation using Numpy”

Task: Build a Neural Network with a single hidden layer by only using Numpy!

Steps:

1. **Go** to the pre-prepared Colab: https://colab.research.google.com/drive/1d_00iXxgvC1kztQilPVkPsQAFe4k1Bzf
2. **Clone the code** to your personal space, so you can change it.
3. **Follow the instructions** in the notebook and fill out the gaps with your own solution!
4. **Change the dataset** creation function from „moons“ to „circles“ or „blob“

Deep Learning with PyTorch - Example

https://colab.research.google.com/drive/10696qeq3kyeibe2LD3UwkMXY3iZr-oN2?usp=sharing

Deep Learning Workflow in Python

Import all dependencies (e.g. Pytorch)

```
import pandas as pd
import torch, torchvision
from torch import nn, optim
import torch.nn.functional as F
from matplotlib import pyplot as plt
```

Set Hyperparameters / Define GPU

```
BATCH_SIZE = 64;
EPOCHS = 3;
LEARNING_RATE = 0.001;
RANDOM_SEED = 1
# GPU Device
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

Import Dataset (e.g. Pandas)

```
pd.read_csv('data.csv')
```

If your CSV is in your working directory

Dataset Preprocessing (Next Slide)

Define the Model

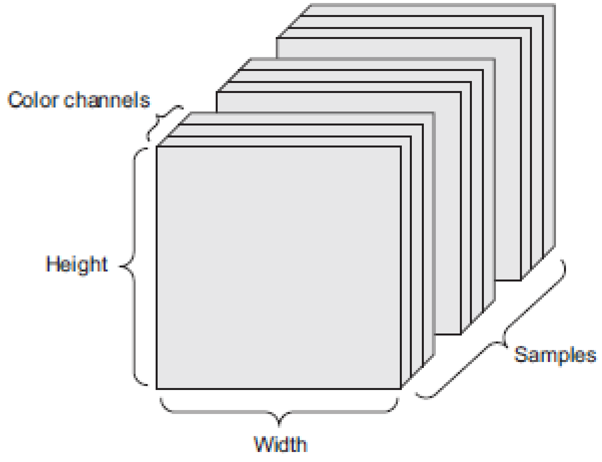
Define the Training loop

Train and Evaluate!

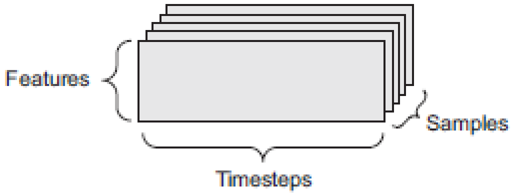
PyTorch

Data Preprocessing

Image Data



Time Series Data



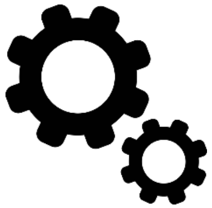
Normalization to counteract skewed data/classes:

Normalization

$$\mathbf{x}_{j,std} = \frac{\mathbf{x}_j - \mu_j}{\sigma_j}$$

Data split into 3 partitions

Training	Validation	Test
Train Network Weights	Optimize Network Parameters	Verify Results on unseen data



Define the Model

```
import pandas as pd
import torch, torchvision
from torch import nn, optim
import torch.nn.functional as F
from matplotlib import pyplot as plt
```

```
class FeedForwardNetwork(nn.Module):
    def __init__(self):
        super().__init__();
        self.fc1 = nn.Linear(784, 1024);     #fc1 = fully connected 1
        self.fc2 = nn.Linear(1024,512);
        self.fc3 = nn.Linear(512, 256);
        self.out = nn.Linear(256, 10);

    def forward(self, X):
        # Flatten X for it to be able to be passed through a linear layer
        # Shape of X will be [64, 1, 28, 28] before the view() call and [64, 784] after
        X = X.view(X.shape[0], -1);

        # Pass X through first, second, third and output linear layer
        X = F.relu(self.fc1(X));
        X = F.relu(self.fc2(X));
        X = F.relu(self.fc3(X));
        X = self.out(X);

        return X;
```

Layer with a state go here

Define how things are executed

Initialize the Model and Define the Training Loop

Check if a GPU is available

```
device
```

```
device(type='cuda', index=0)
```

We can ask for the device variable to check our device (GPU or CPU)

```
torch.manual_seed(RANDOM_SEED)
```

Additionally we save our random seed for reproducibility

```
model = FeedForwardNetwork();
'If you want to use GPU'
# model = model.to(device)
criterion = nn.CrossEntropyLoss();
optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE);
```

Now we create the instance from our model class and define the loss function and optimizer

```
model
```

```
FeedForwardNetwork(
  (fc1): Linear(in_features=784, out_features=1024, bias=True)
  (fc2): Linear(in_features=1024, out_features=512, bias=True)
  (fc3): Linear(in_features=512, out_features=256, bias=True)
  (out): Linear(in_features=256, out_features=10, bias=True)
)
```

Our model looks like expected

Initialize the Model and Define the Training Loop

```python
def validate(val_batch):
    model.eval();
    with torch.no_grad():
        X, y = val_batch;
        out = model(X);

        predictions = torch.argmax(out, dim=1);
        samples_correct = predictions[predictions == y].shape[0];

        val_loss = criterion(out, y); ' criterion = nn.CrossEntropyLoss();  '
        val_accuracy = samples_correct/X.shape[0] * 100;

    model.train();
    return val_loss, val_accuracy;
```

```python
class FeedForwardNetwork(nn.Module):
    def __init__(self):
        super().__init__();
        self.fc1 = nn.Linear(784, 1024);      #fc1 = fully connected 1
        self.fc2 = nn.Linear(1024,512);
        self.fc3 = nn.Linear(512, 256);
        self.out = nn.Linear(256, 10);

    def forward(self, X):
        # Flatten X for it to be able to be passed through a linear layer
        # Shape of X will be [64, 1, 28, 28] before the view() call and [64, 784] after
        X = X.view(X.shape[0], -1);

        # Pass X through first, second, third and output linear layer
        X = F.relu(self.fc1(X));
        X = F.relu(self.fc2(X));
        X = F.relu(self.fc3(X));
        X = self.out(X);

        return X;
```

Last layer seems to miss a softmax which is needed for classification

Softmax is built into the CrossEntropyLoss in PyTorch! Therefore is not needed here!

The Training Loop

If you omit this, gradients will be accumulated, which we usually don't want

Softmax is applied

The gradients are calculated

The optimizer updates the weights

Here we track the performance (Optional but recommended)

Feed Forward is performed

```python
train_losses = [];
train_accuracies = [];
val_losses = [];
val_accuracies = [];

val_iterator = iter(val_loader);

for epoch in range(EPOCHS):
    model.train()
    for i, (X, y) in enumerate(train_loader):
        # X is a tensor with shape [64, 1, 28, 28] (the batch of images)
        # 64: batchsize, 1: image channels (1 because grayscaled), 28: image height, 28: image width
        # y is a tensor with shape [64] (the batch of labels)
        optimizer.zero_grad();

        out = model(X);
        # out (tensor): [64, 10]

        predictions = torch.argmax(out, dim=1);
        batch_samples_correct = predictions[predictions == y].shape[0];
        batch_accuracy = batch_samples_correct / y.shape[0] * 100;
        batch_loss = criterion(out, y);

        batch_loss.backward();
        optimizer.step();

        try:
            val_loss, val_accuracy = validate(val_iterator.next());
        except StopIteration:
            val_iterator = iter(val_loader);
            val_loss, val_accuracy = validate(val_iterator.next());

        train_losses.append(batch_loss.item());
        train_accuracies.append(batch_accuracy);
        val_losses.append(val_loss.item());
        val_accuracies.append(val_accuracy);
```

Evaluation/Training Mode

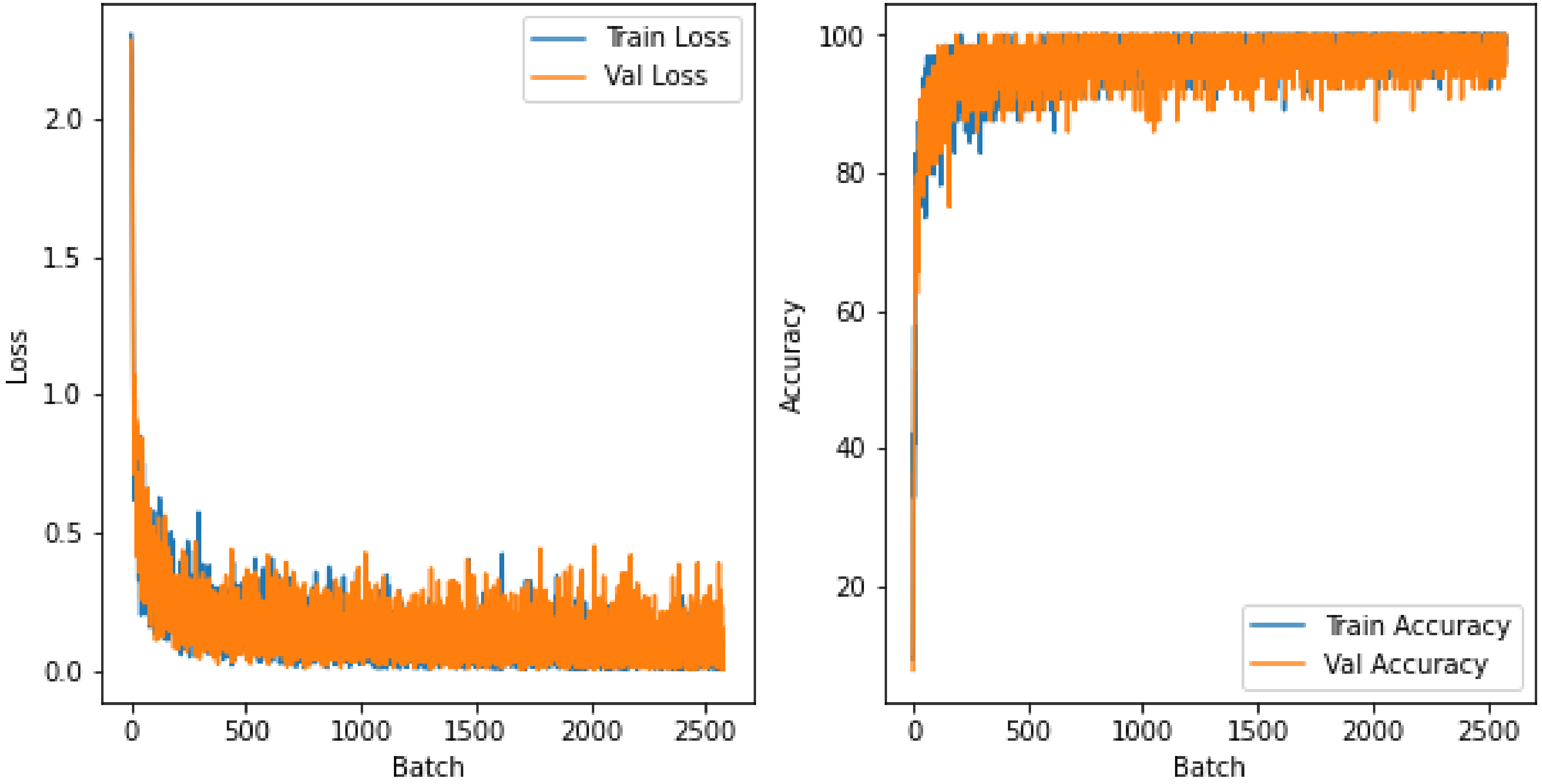
train() & .eval() modes matter for things like BatchNorm, Dropout etc.,

no.grad() prevents unnecessary graph construction

```
train_losses = [];
train_accuracies = [];
val_losses = [];
val_accuracies = [];

val_iterator = iter(val_loader);

for epoch in range(EPOCHS):
    model.train()
    for i, (X, y) in enumerate(train_loader):
        # X is a tensor with shape [64, 1, 28, 28] (the batch of images)
        # 64: batchsize, 1: image channels (1 because grayscaled), 28: image height, 28: image width
        # y is a tensor with shape [64] (the batch of labels)
        optimizer.zero_grad();

        out = model(X);
        # out (tensor): [64, 10]

            tions = torch.argmax(out, dim=1);
            samples_correct = predictions[predictions == y].shape[0];
            accuracy = batch_samples_correct / y.shape[0] * 100;
            loss = criterion(out, y);

            loss.backward();
            zer.step();


            l_loss, val_accuracy = validate(val_iterator.next());
             StopIteration:
            l_iterator = iter(val_loader);
            l_loss, val_accuracy = validate(val_iterator.next());

        _losses.append(batch_loss.item());
        train_accuracies.append(batch_accuracy);
        val_losses.append(val_loss.item());
        val_accuracies.append(val_accuracy);
```

```
def validate(val_batch):
    model.eval();
    with torch.no_grad():
        X, y = val_batch;
        out = model(X);

        predictions = torch.argmax(out, dim=1);
        samples_correct = predictions[predictions == y].shape[0];

        val_loss = criterion(out, y); ' criterion = nn.CrossEntropyLoss();
        val_accuracy = samples_correct/X.shape[0] * 100;

    model.train();
    return val_loss, val_accuracy;
```

Plot the Loss Functions and Accuracy over Time



Deep Learning with PyTorch – Live Demo

https://colab.research.google.com/drive/1lRwU6IEWwMW2oxNAWzE14elkSr8q3792

Literature

Code examples

- MNIST: https://github.com/rasbt/stat453-deep-learning-ss21/blob/main/L13/code/1-lenet5-mnist.ipynb
- CIFAR10: https://github.com/rasbt/stat453-deep-learning-ss21/blob/main/L13/code/3-cnn-cifar10.ipynb

Books

- Eli Stevens et al. „Deep Learning with PyTorch“, 2020, ISBN 9781617295263
  https://isip.piconepress.com/courses/temple/ece_4822/resources/books/Deep-Learning-with-PyTorch.pdf
- Sebastian Raschka “Machine Learning with PyTorch and Scikit-Learn” 2022, ISBN: 978-1801819312
  https://learning.oreilly.com/library/view/machine-learning-with/9781801819312/ - Login via SLUB account

Next lecture: 6. Sequential Neural Networks (Steffen Seitz)