Cagri Ozcinar, PhD

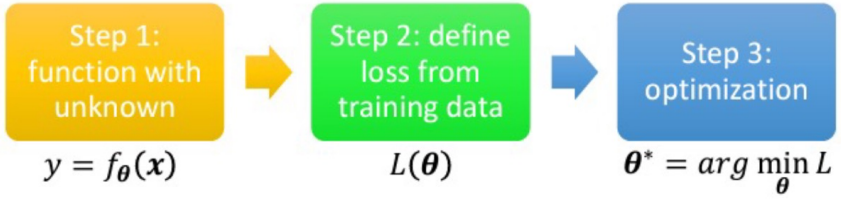
LTTI.00.032 – Machine Learning in Synthetic Biology

Recap

- The essence of deep learning is to find best weights for the network that minimizes the error (loss)

Cagri Ozcinar, PhD

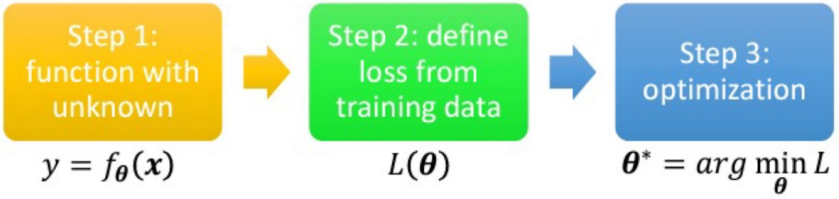
Recap



- The essence of deep learning is to find best weights for the network that minimizes the error (loss)

Cagri Ozcinar, PhD

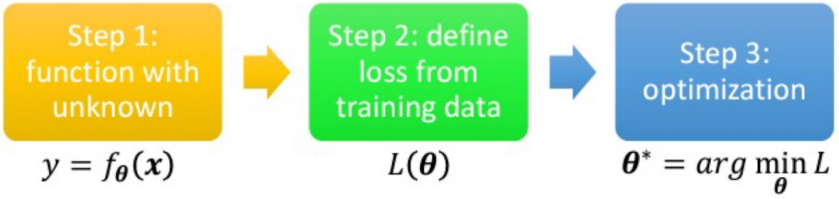
Recap



- The essence of deep learning is to find best weights for the network that minimizes the error (loss)
- This is done via an iterative process where weights are updated in each iteration in a direction that minimizes the loss

Cagri Ozcinar, PhD

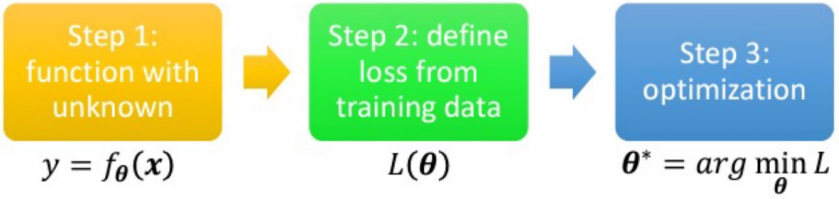
Recap



- The essence of deep learning is to find best weights for the network that minimizes the error (loss)
- This is done via an iterative process where weights are updated in each iteration in a direction that minimizes the loss
- To find this direction we need the slope of the loss function for a given weight. This is achieved by computing the derivatives (gradient)

Cagri Ozcinar, PhD

Recap



- The essence of deep learning is to find best weights for the network that minimizes the error (loss)
- This is done via an iterative process where weights are updated in each iteration in a direction that minimizes the loss
- To find this direction we need the slope of the loss function for a given weight. This is achieved by computing the derivatives (gradient)
- It is computationally expensive to compute derivatives for millions of weights

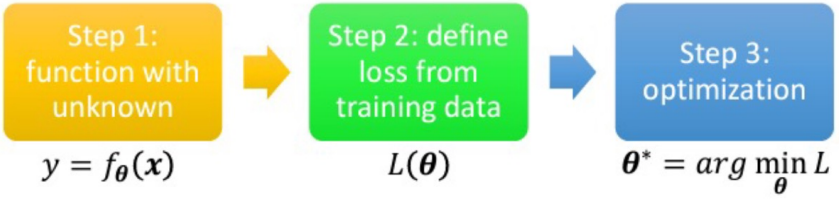
Recap

- Backpropagation makes this computation possible by using the chain rule in calculus

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. nature, 323(6088), 533-536.

Cagri Ozcinar, PhD

Recap



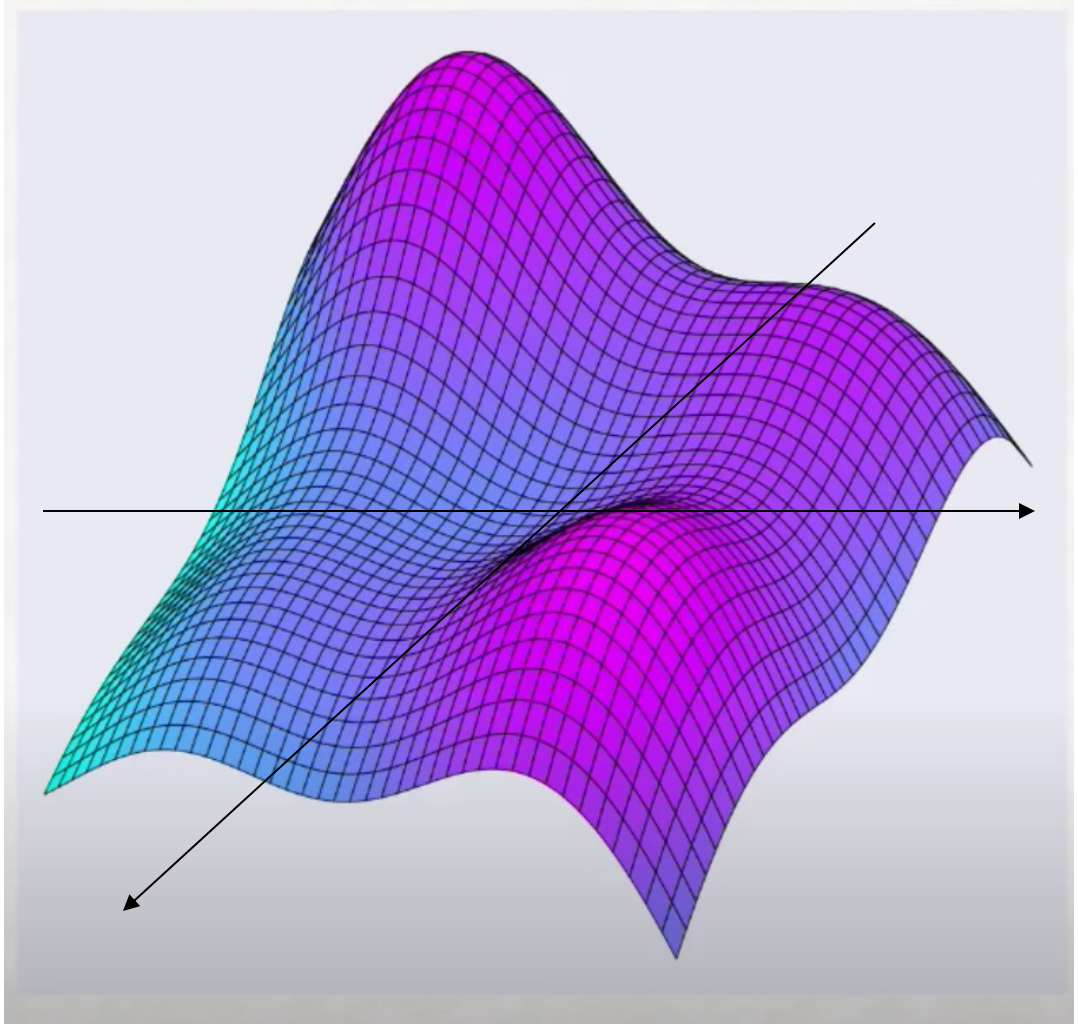
- Backpropagation makes this computation possible by using the chain rule in calculus

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. nature, 323(6088), 533-536.

- Gradient decent is a general term for calculating the gradient and updating the weights

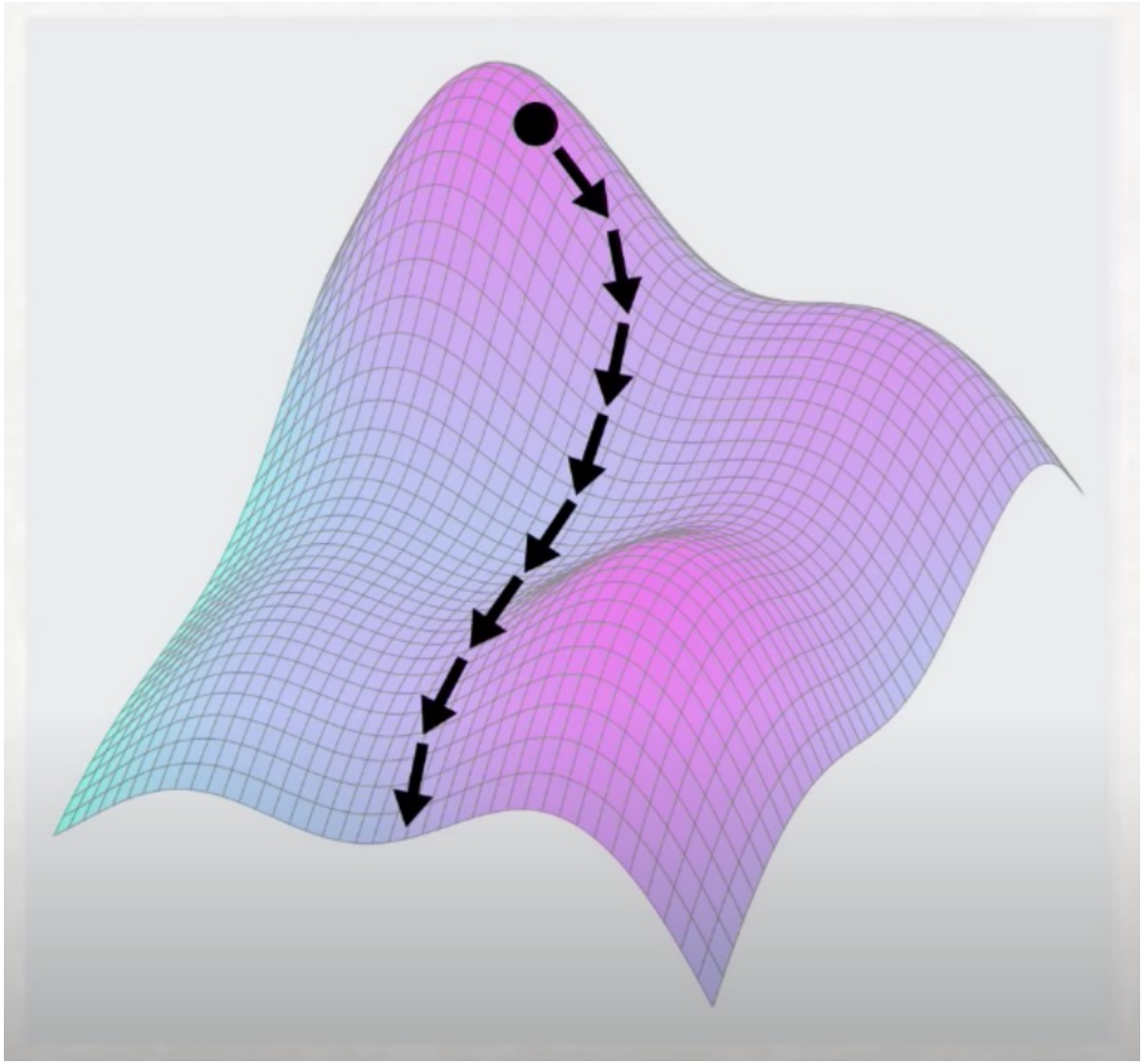
Cagri Ozcinar, PhD

Gradient Decent



Cagri Ozcinar, PhD

Gradient Decent

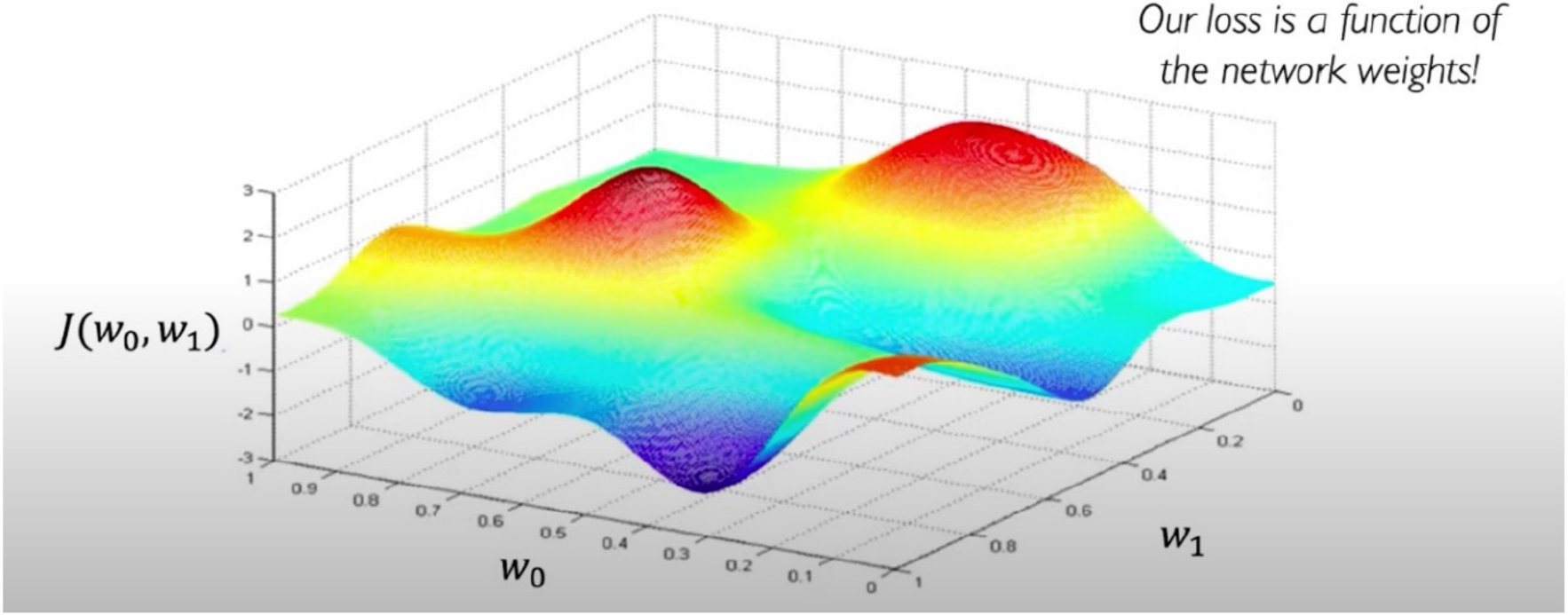


Cagri Ozcinar, PhD

Loss optimization

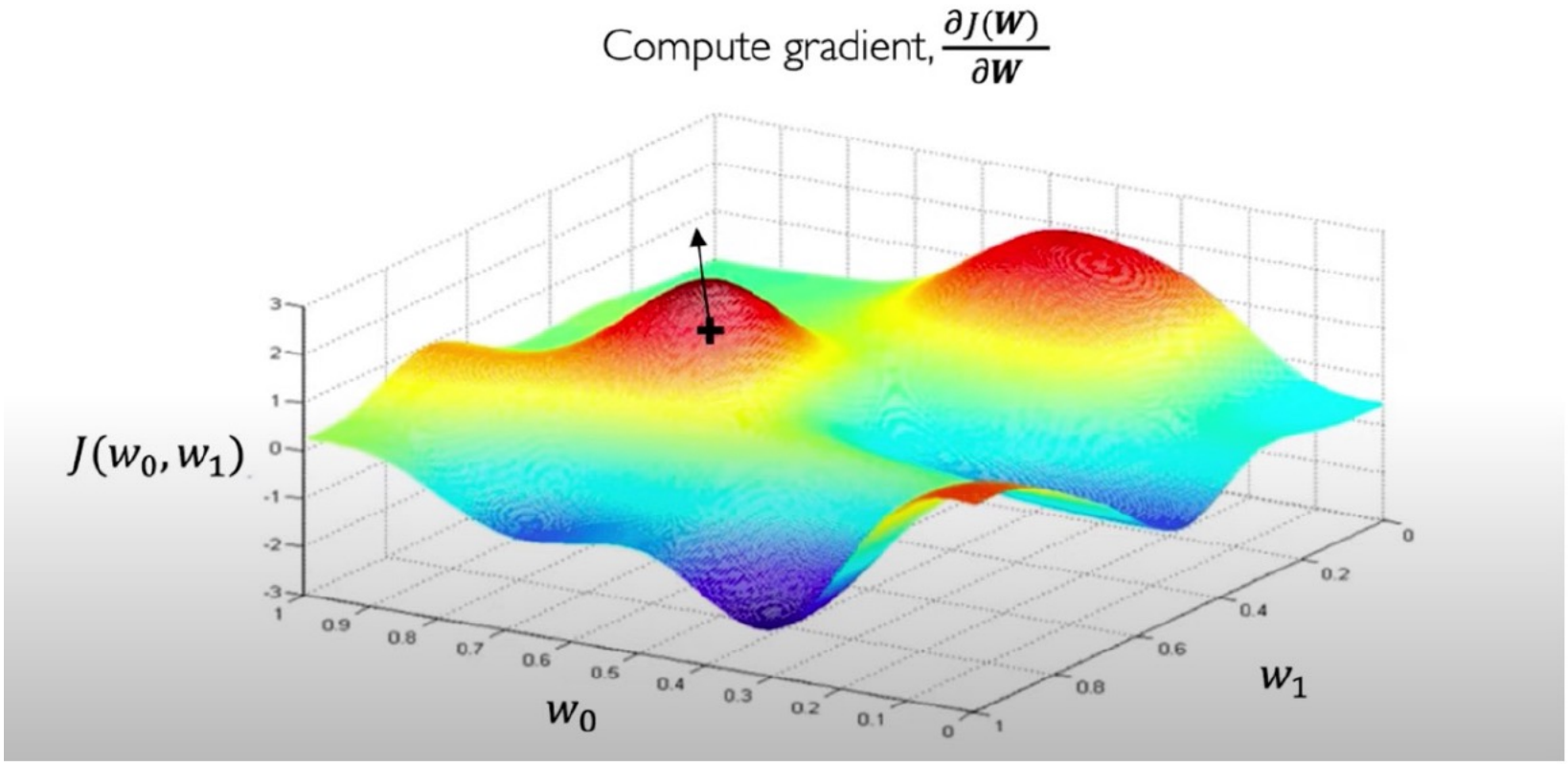
$$W^* = \underset{W}{\operatorname{argmin}} J(W)$$

Remember:
Our loss is a function of the network weights!



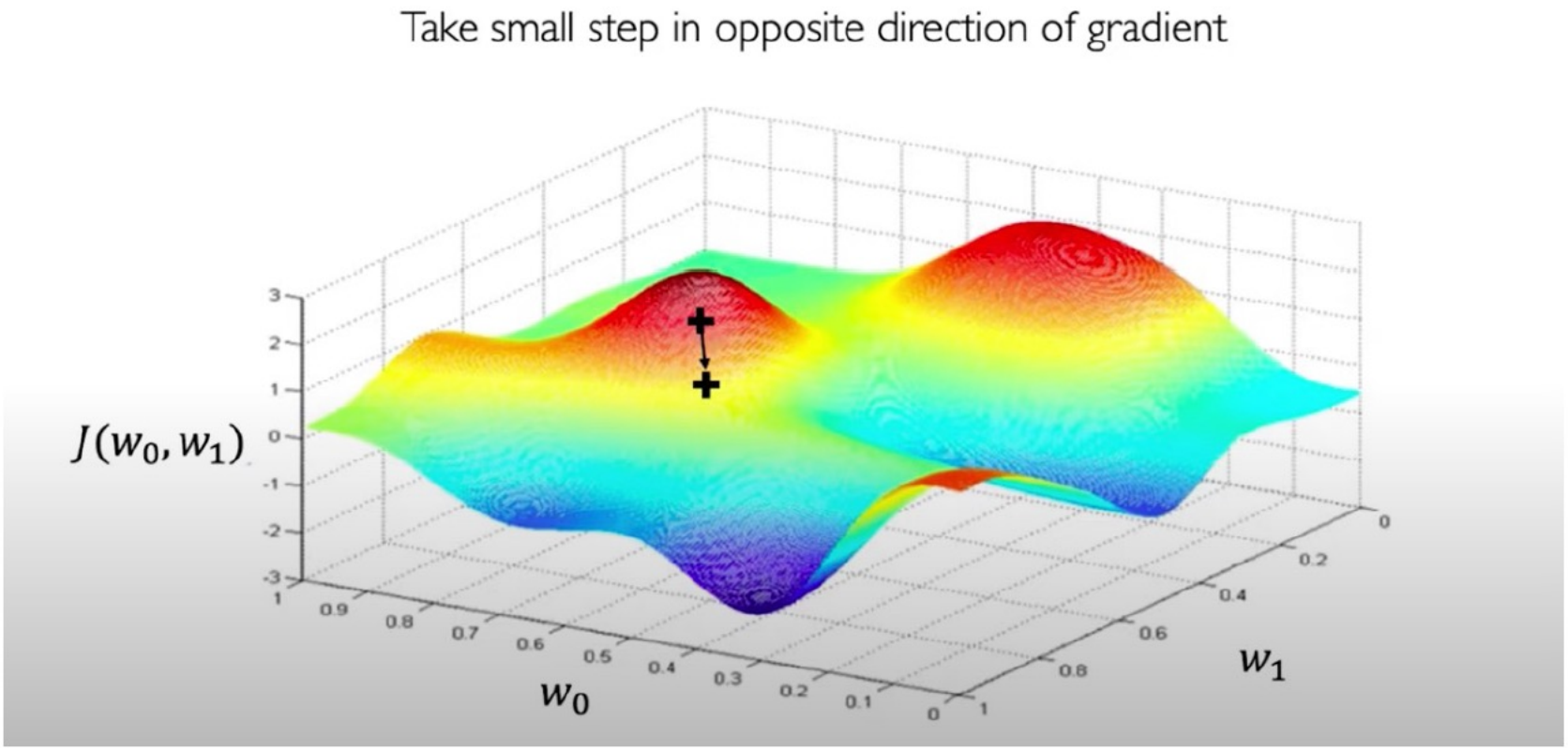
Cagri Ozcinar, PhD

Loss optimization



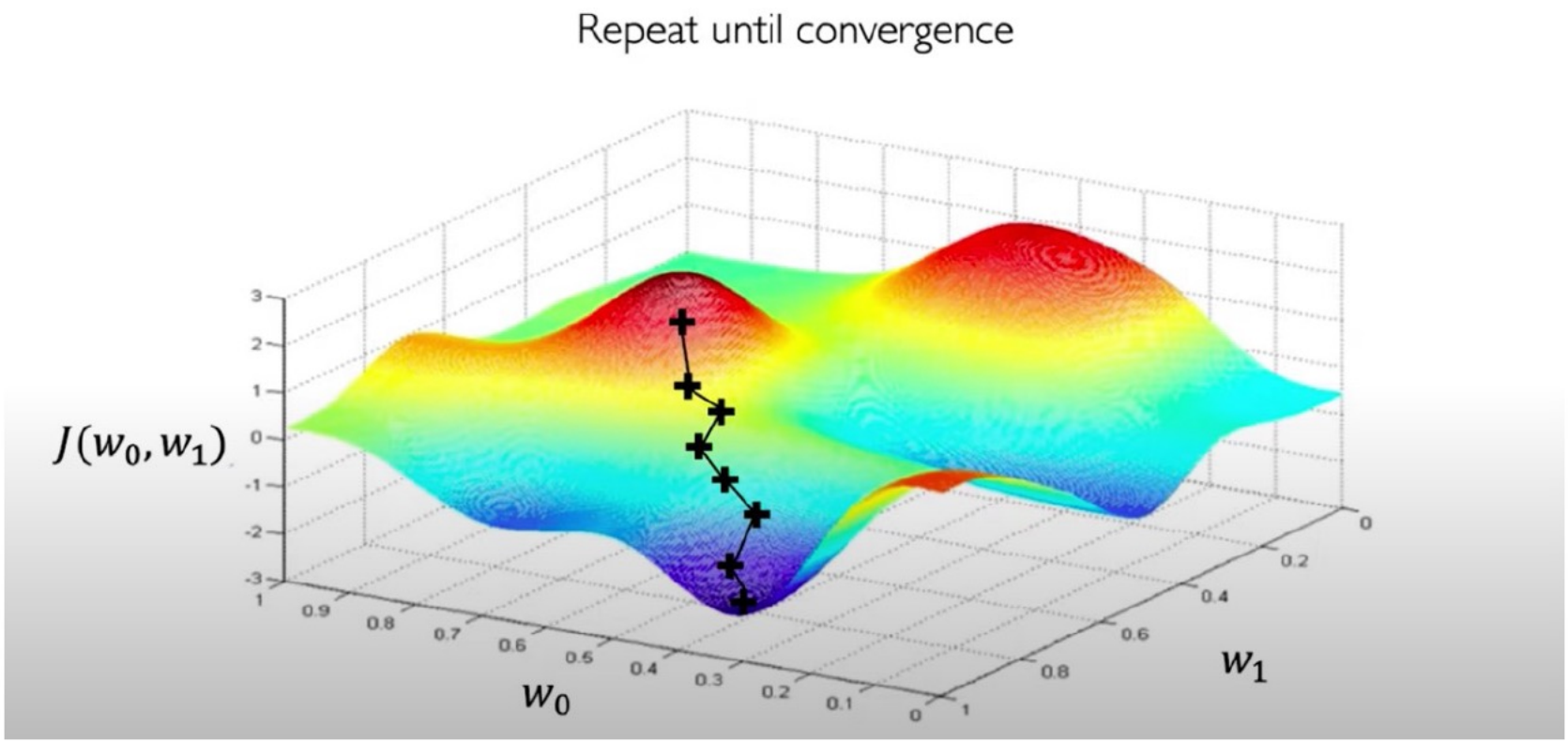
Cagri Ozcinar, PhD

Loss optimization



Cagri Ozcinar, PhD

Gradient decent



Cagri Ozcinar, PhD

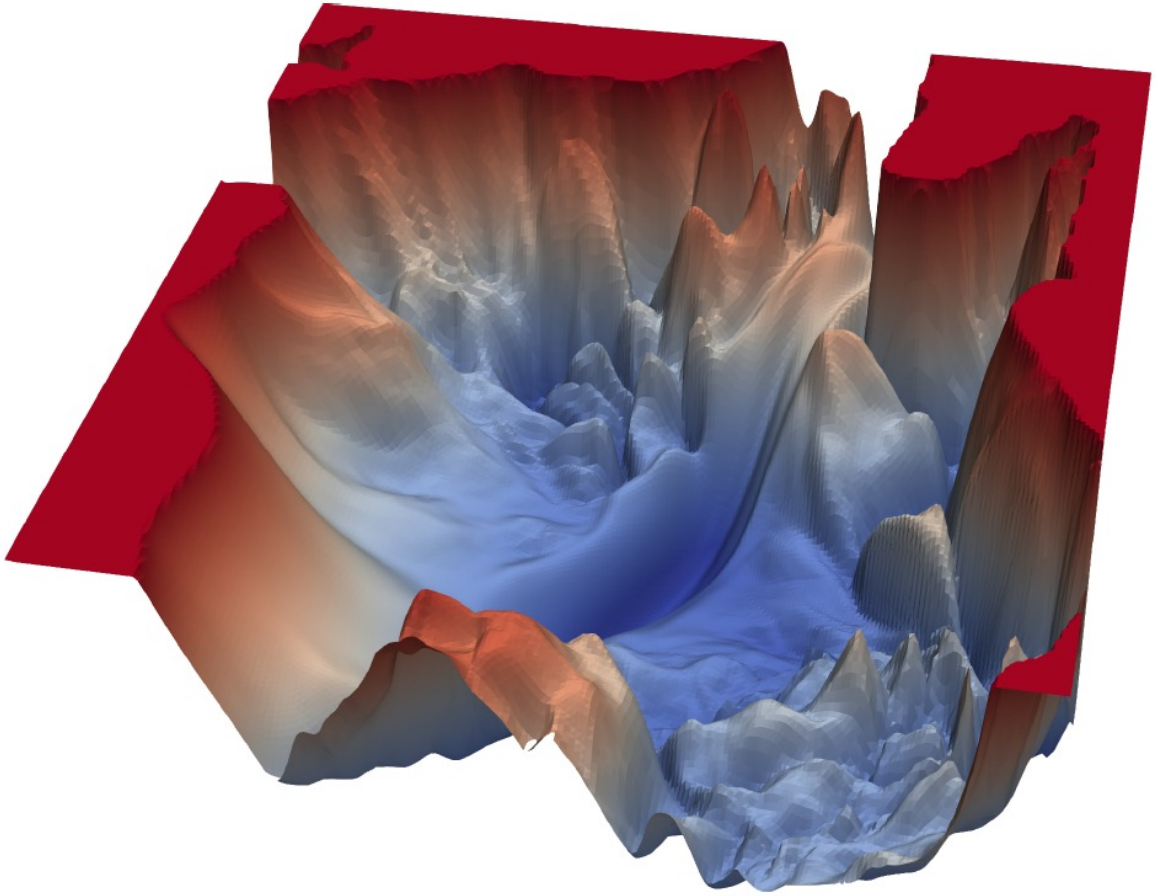
Gradient decent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Cagri Ozcinar, PhD

Gradient decent



Cagri Ozcinar, PhD

Loss function optimization

Remember:

Optimization through gradient descent

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

Cagri Ozcinar, PhD

Loss function optimization

Remember:

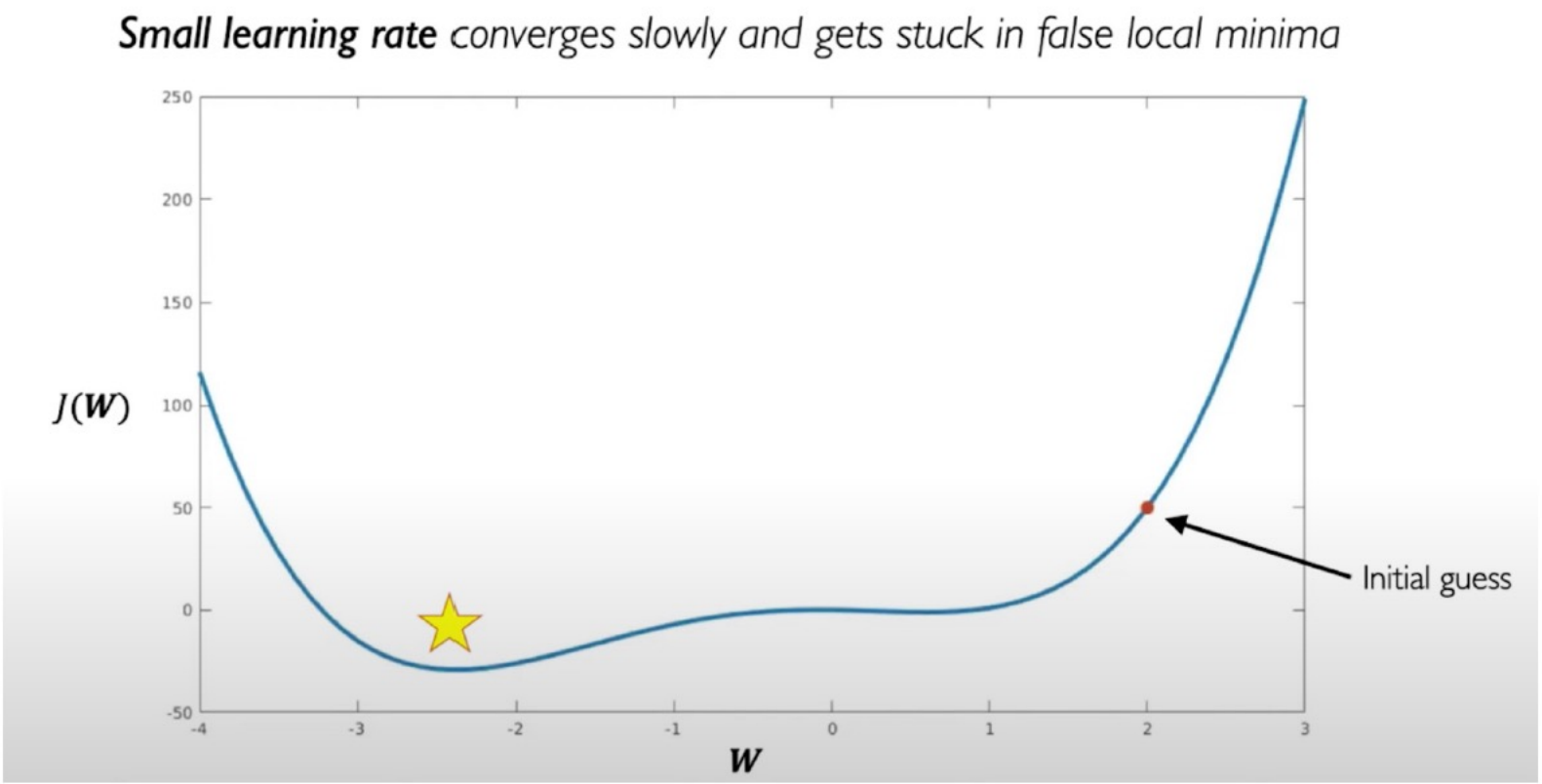
Optimization through gradient descent

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

How can we set the learning rate?

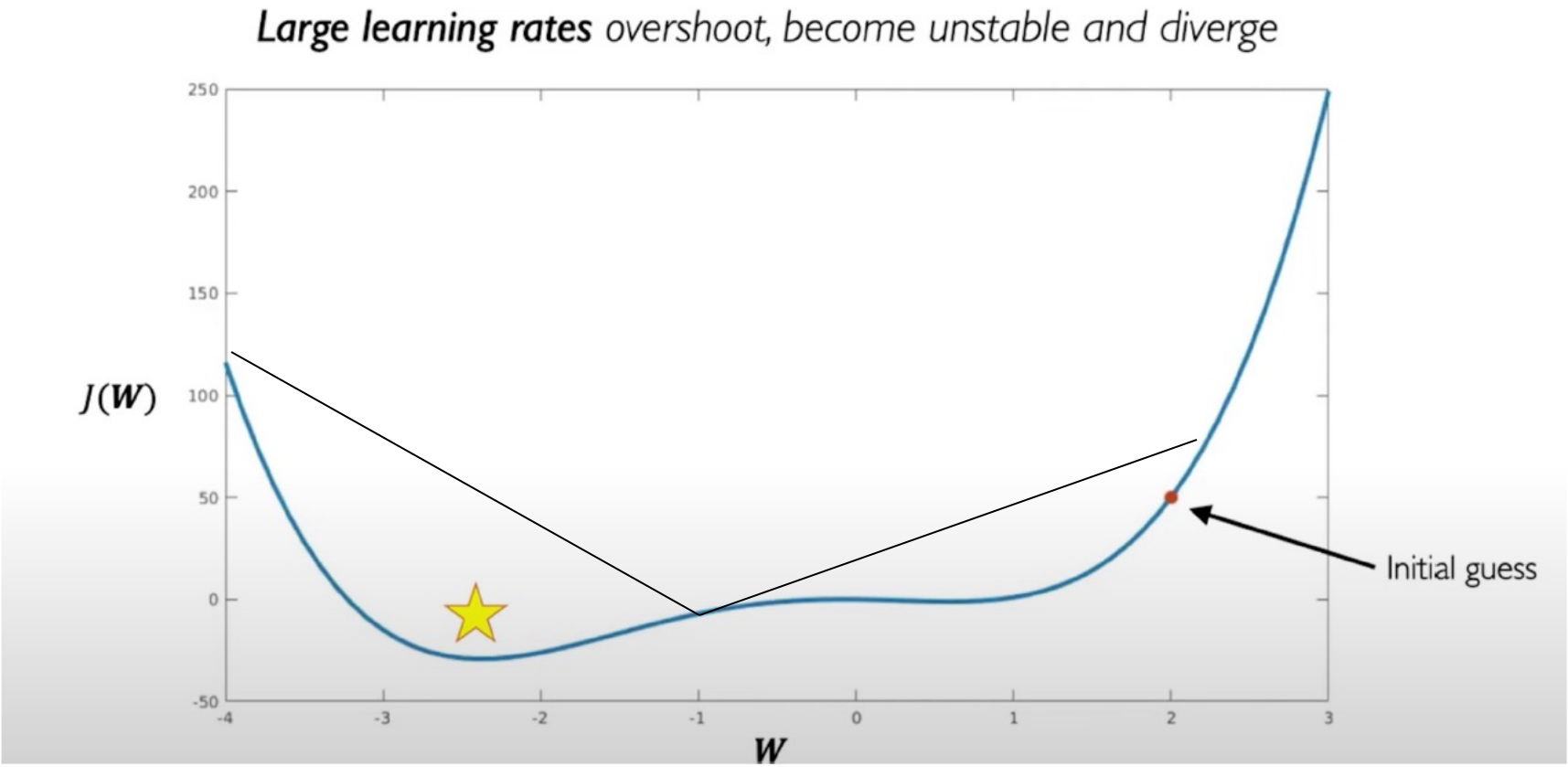
Cagri Ozcinar, PhD

Setting the learning rate



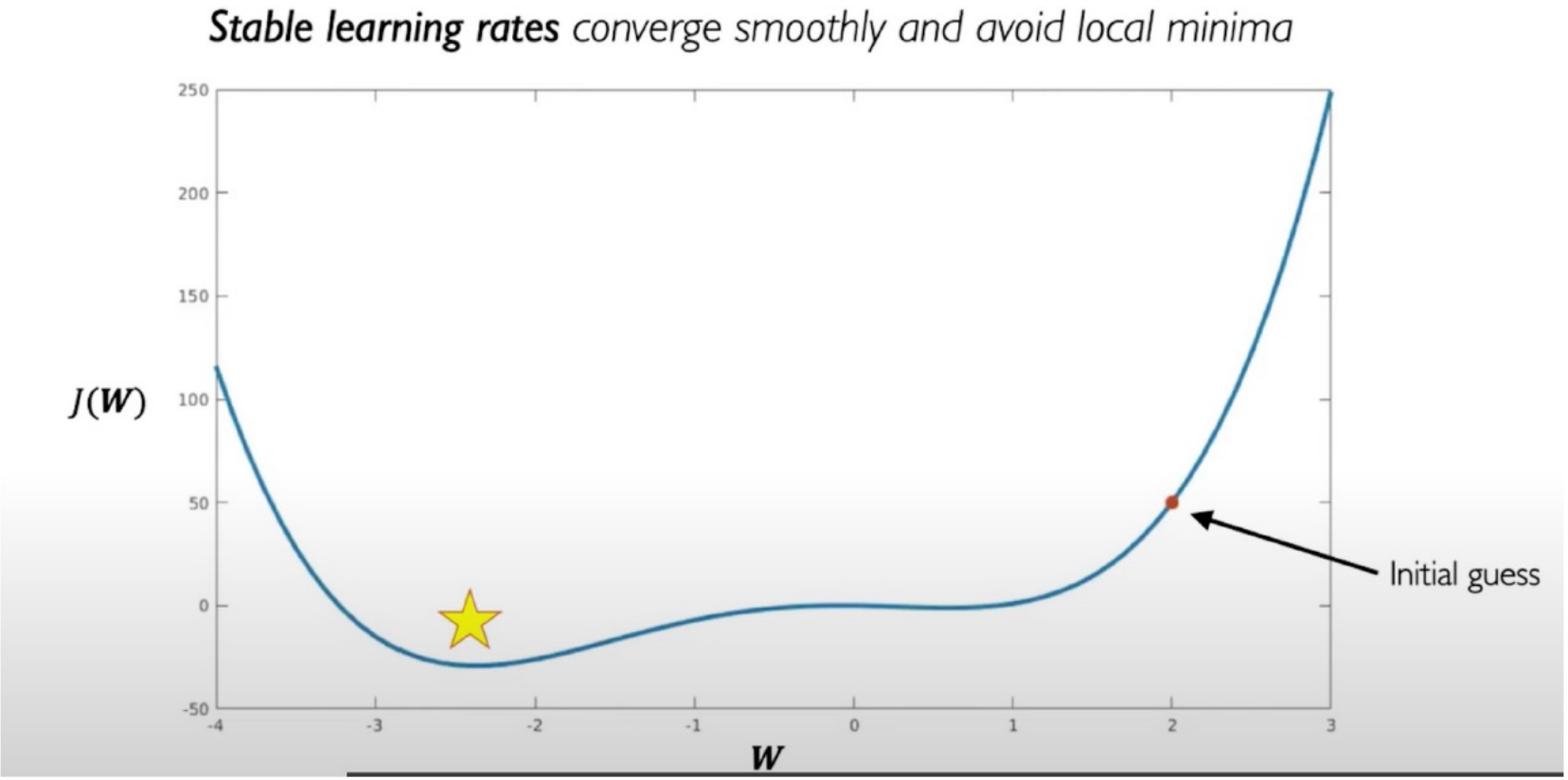
Cagri Ozcinar, PhD

Setting the learning rate



Cagri Ozcinar, PhD

Setting the learning rate



Cagri Ozcinar, PhD

How to deal with this

Idea 1.

Try lots of different learning rates and see what works “just right”

Cagri Ozcinar, PhD

How to deal with this

Idea 1.

Try lots of different learning rates and see what works “just right”

Idea 2.

Design an adaptive learning rate that adapts to the landscape

Cagri Ozcinar, PhD

Gradient decent algorithms

SGD

Adam

Adadelta

Adagrad

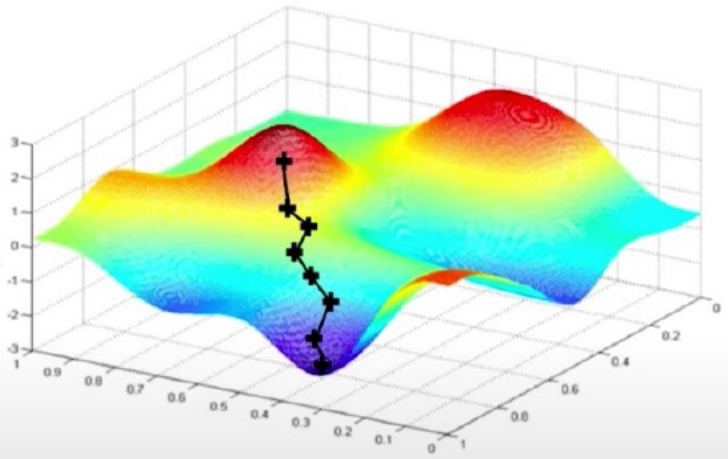
RMSProp

Cagri Ozcinar, PhD

Gradient decent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

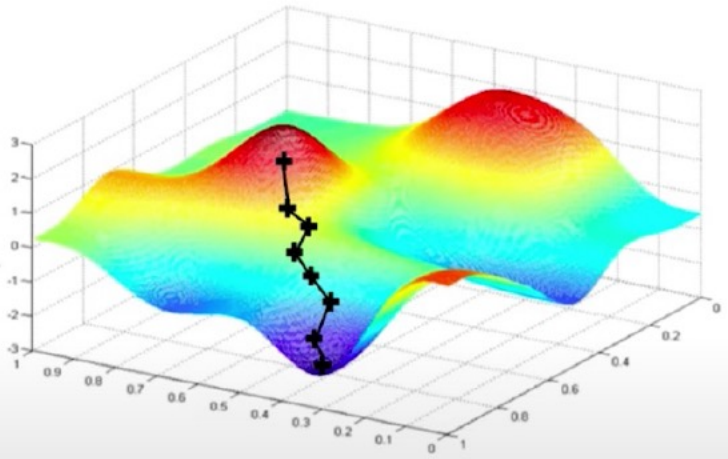


Cagri Ozcinar, PhD

Gradient decent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

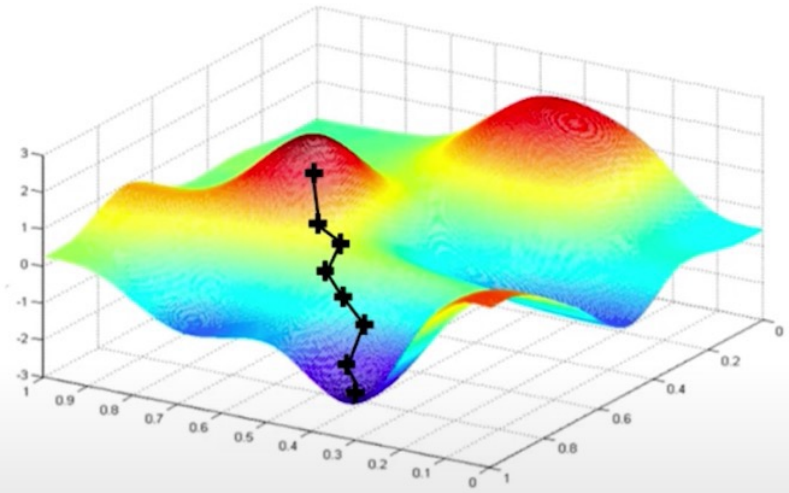


Cagri Ozcinar, PhD

Stochastic Gradient Decent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point i
4. Compute gradient, $\frac{\partial J_i(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights

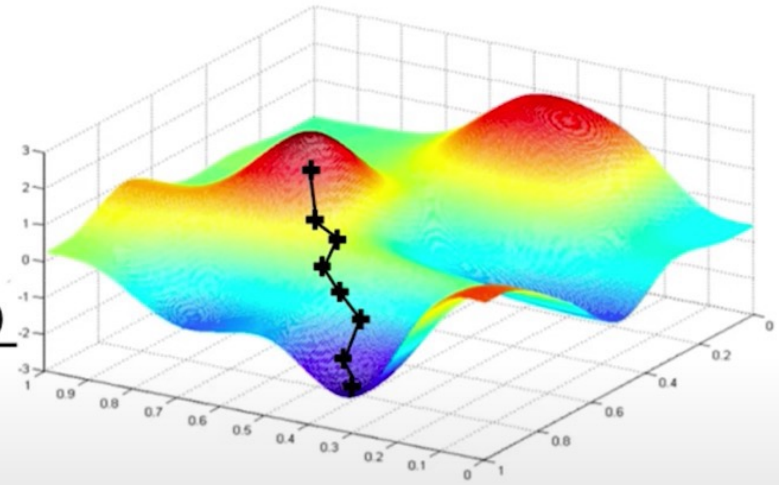


Cagri Ozcinar, PhD

Stochastic Gradient Decent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B} \sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights



Cagri Ozcinar, PhD

Mini-batches while training

- More accurate estimation of gradient
- Smoother convergence
- Allows for larger learning rates
- Leads to fast training

Cagri Ozcinar, PhD

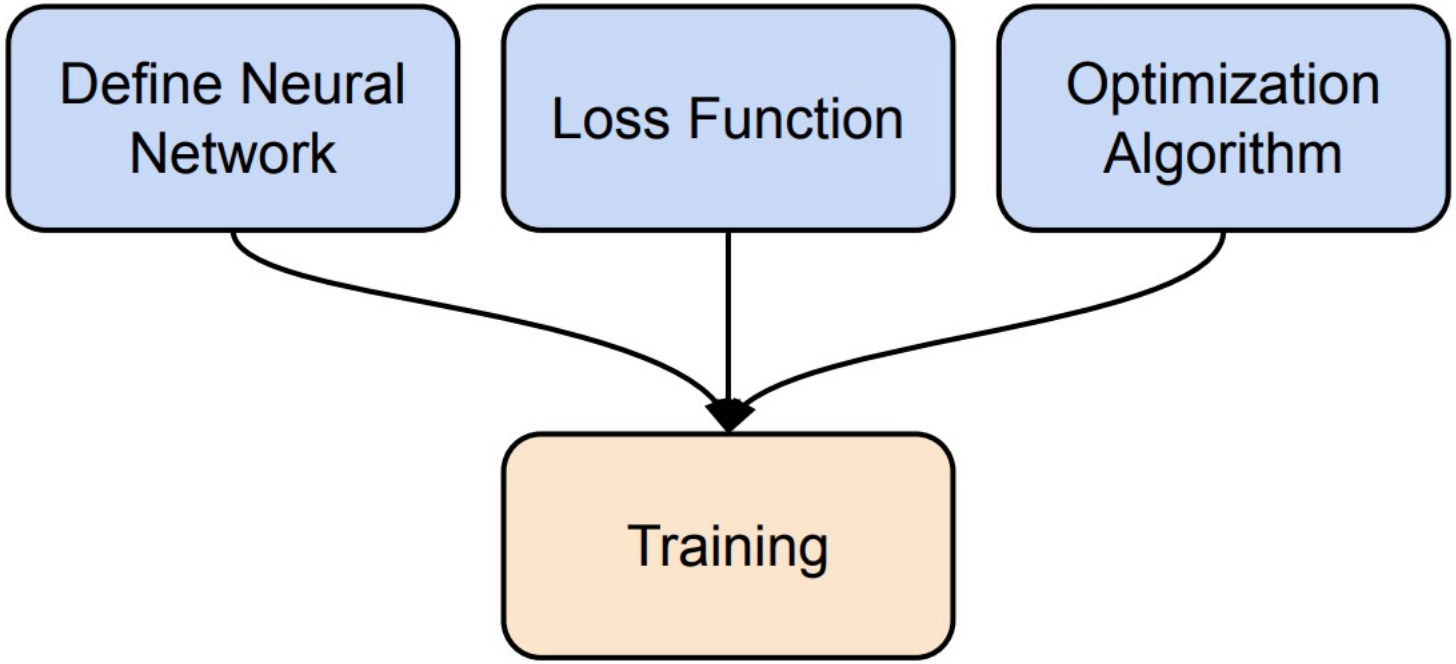
Practical

What is PyTorch?

- An **machine learning framework** in Python
- Two main features:
  - N-dimensional **Tensor** computation on **GPUs**
  - Automatic differentiation for training deep neural networks

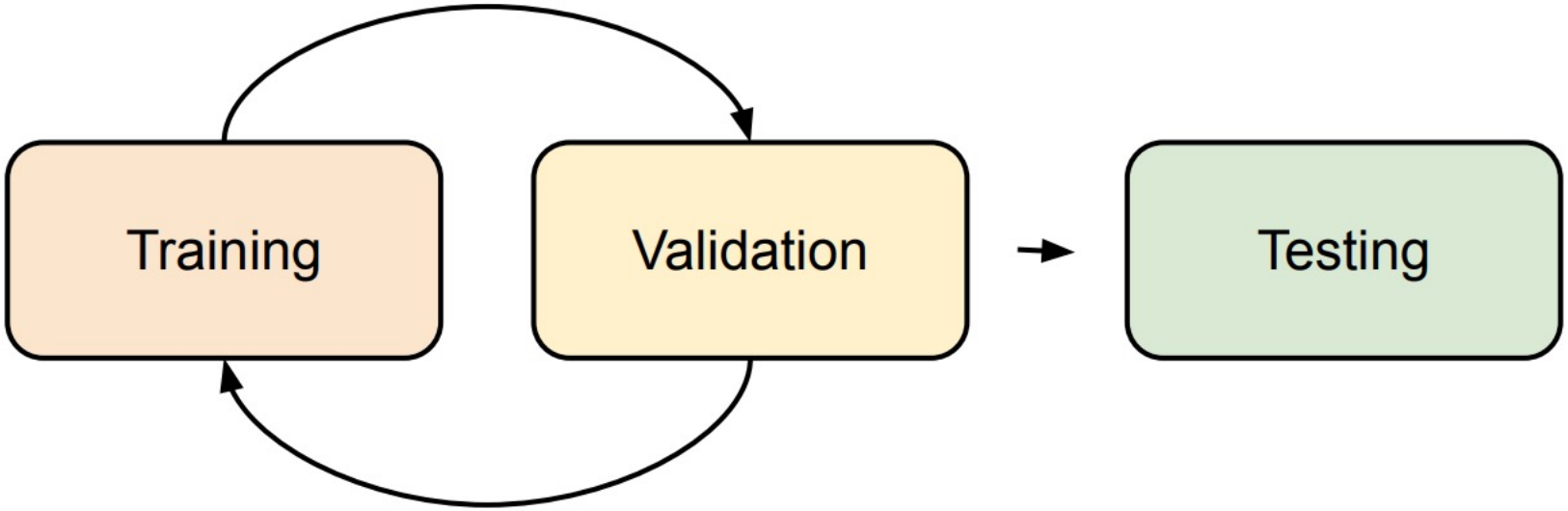
Cagri Ozcinar, PhD

Training Neural Networks



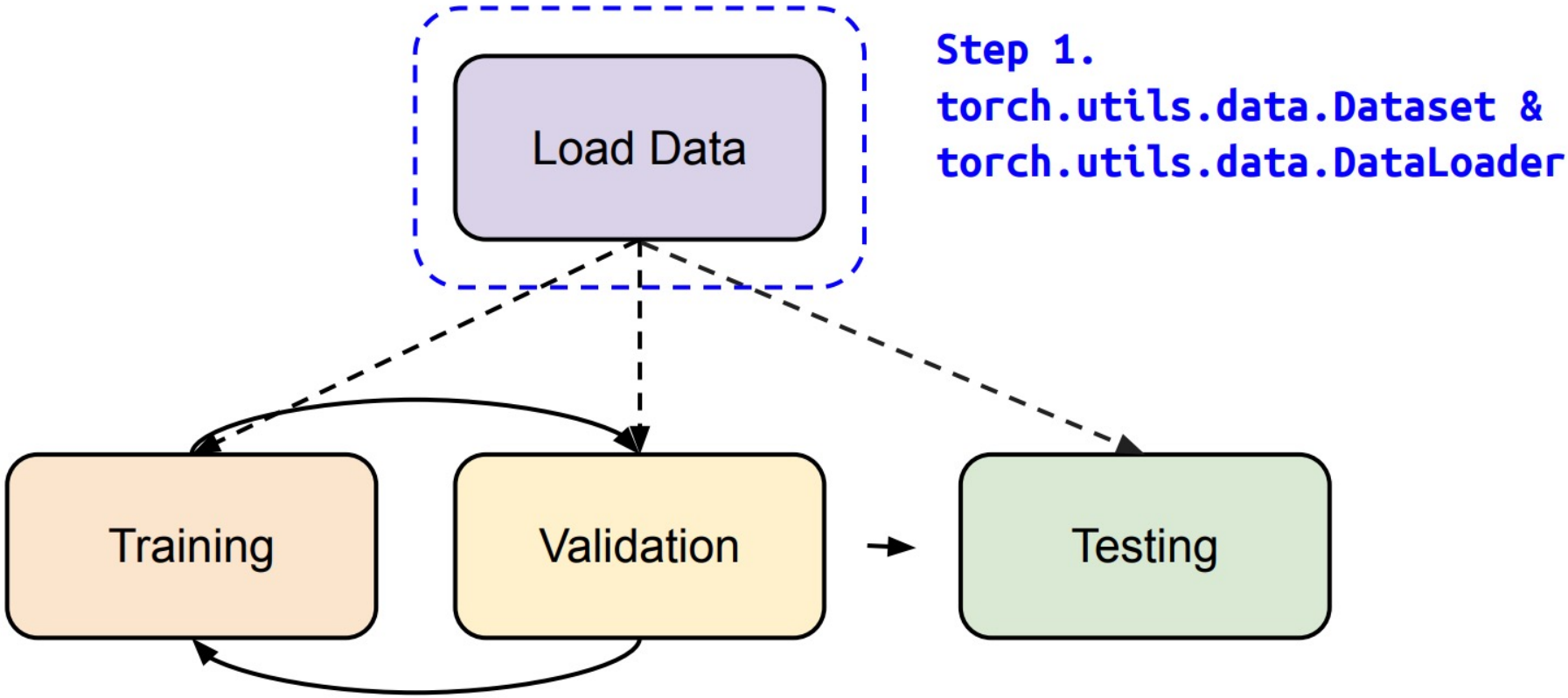
Cagri Ozcinar, PhD

Training & Testing Neural Networks



Cagri Ozcinar, PhD

Training & Testing Neural Networks in PyTorch



Cagri Ozcinar, PhD

Dataset & Dataloader

- `Dataset:` stores data samples and expected values
- `Dataloader:` groups data in batches, enables multiprocessing

- `dataset = MyDataset(file)`
- `dataloader = DataLoader(dataset, batch_size, shuffle=True)`

Training: True
Testing: False

Cagri Ozcinar, PhD

Dataset & Dataloader

- `Dataset:` stores data samples and expected values
- `Dataloader:` groups data in batches, enables multiprocessing

- `dataset = MyDataset(file)`
- `dataloader = DataLoader(dataset, batch_size, shuffle=True)`

Training: True
Testing: False

Cagri Ozcinar, PhD

Dataset & Dataloader

```
from torch.utils.data import Dataset, DataLoader

class MyDataset(Dataset):
    def __init__(self, file):
        self.data = ...

    def __getitem__(self, index):
        return self.data[index]

    def __len__(self):
        return len(self.data)
```

Read data & preprocess

Returns one sample at a time

Returns the size of the dataset

Cagri Ozcinar, PhD

Dataset & Dataloader

```
from torch.utils.data import Dataset, DataLoader

class MyDataset(Dataset):
    def __init__(self, file):
        self.data = ...

    def __getitem__(self, index):
        return self.data[index]

    def __len__(self):
        return len(self.data)
```

Read data & preprocess

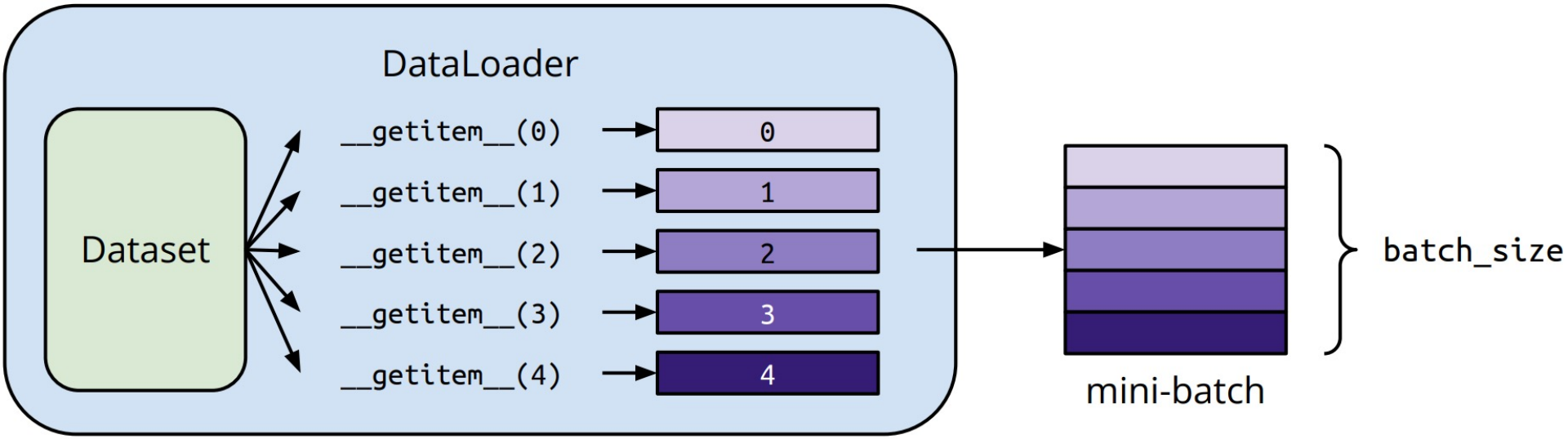
Returns one sample at a time

Returns the size of the dataset

Cagri Ozcinar, PhD

Dataset & Dataloader

```
dataset = MyDataset(file)

dataloader = DataLoader(dataset, batch_size=5, shuffle=False)
```

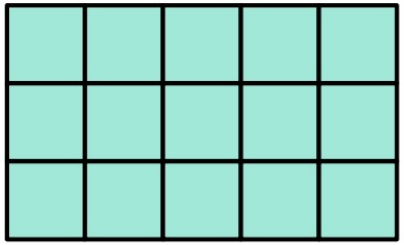


Cagri Ozcinar, PhD

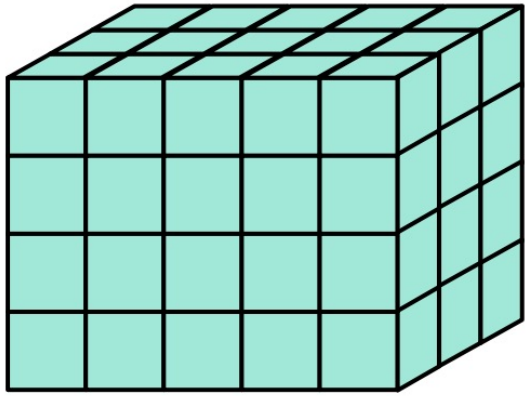
Tensors

- High-dimensional matrices (arrays)

1-D tensor
e.g. audio



2-D tensor
e.g. black&white images

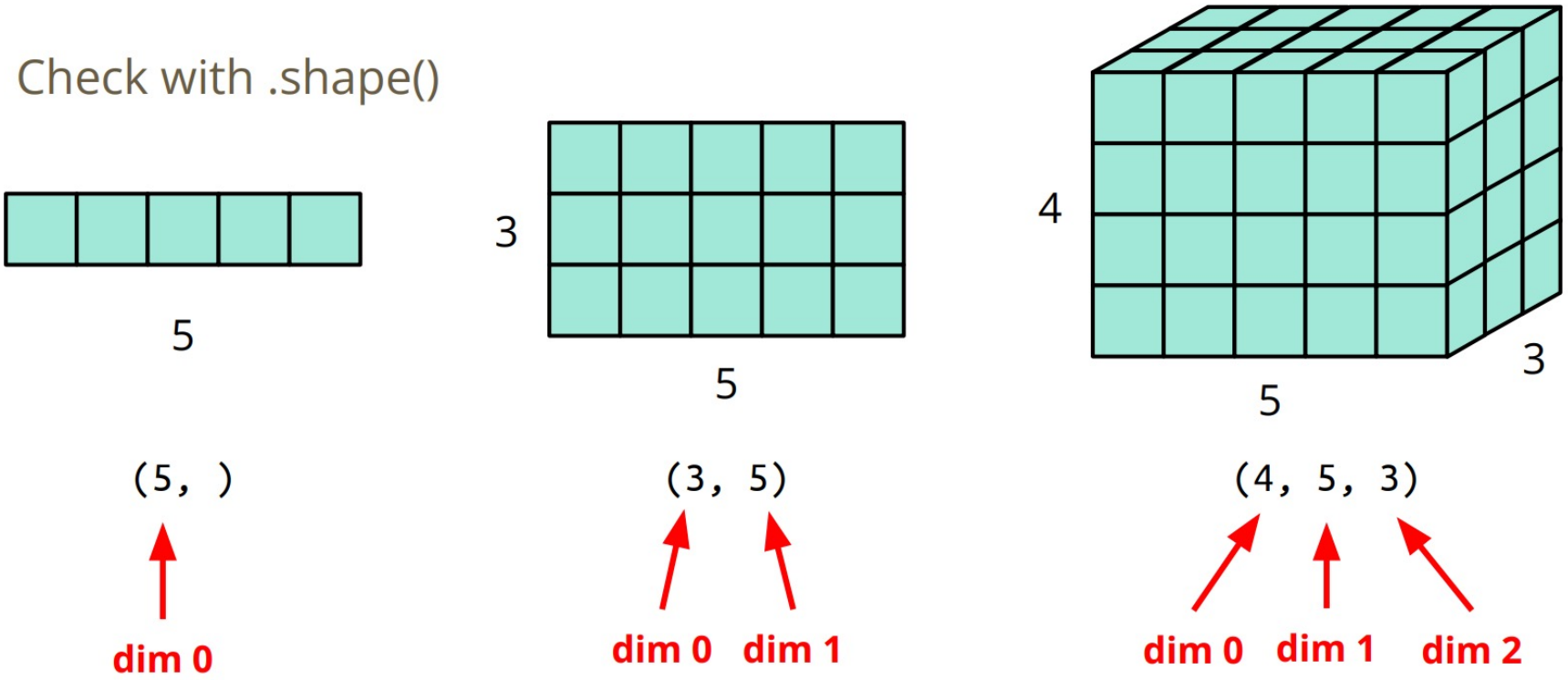


3-D tensor
e.g. RGB images

Cagri Ozcinar, PhD

Tensors – Shape of Tensors

- Check with .shape()



Note: `dim` in PyTorch == `axis` in NumPy

Cagri Ozcinar, PhD

Tensors – Creating Tensors

- Directly from data (list or numpy.ndarray)

```
x = torch.tensor([[1, -1], [-1, 1]])
```

```
tensor([[1., -1.],
        [-1., 1.]])
```

```
x = torch.from_numpy(np.array([[1, -1], [-1, 1]]))
```

- Tensor of constant zeros & ones

```
x = torch.zeros([2, 2])
```

```
tensor([[0., 0.],
        [0., 0.]])
```

```
x = torch.ones([1, 2, 5])
```

shape

```
tensor([[[1., 1., 1., 1., 1.],
         [1., 1., 1., 1., 1.]]])
```

Cagri Ozcinar, PhD

Tensors – Common Operations

Common arithmetic functions are supported, such as:

- Addition

```
z = x + y
```

- Subtraction

```
z = x - y
```

- Power

```
y = x.pow(2)
```

- Summation

```
y = x.sum()
```

- Mean

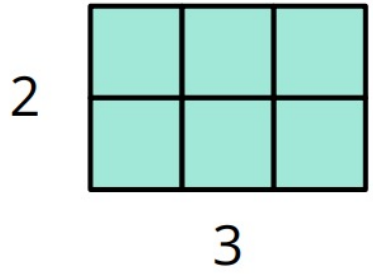
```
y = x.mean()
```

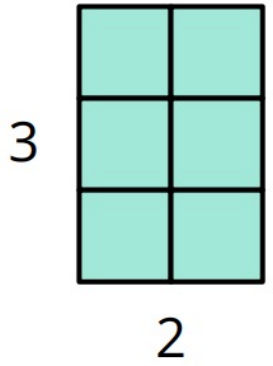
Cagri Ozcinar, PhD

Tensors – Common Operations

- **Transpose**: transpose two specified dimensions

```
>>> x = torch.zeros([2, 3])
>>> x.shape
torch.Size([2, 3])
>>> x = x.transpose(0, 1)
>>> x.shape
torch.Size([3, 2])
```





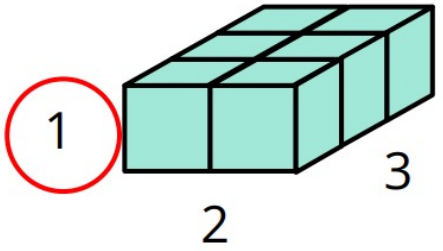
Cagri Ozcinar, PhD

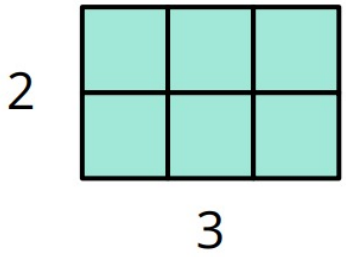
Tensors – Common Operations

- **Squeeze**: remove the specified dimension with length = 1

```
>>> x = torch.zeros([1, 2, 3])
>>> x.shape
torch.Size([1, 2, 3])
>>> x = x.squeeze(0)
>>> x.shape
torch.Size([2, 3])
```

(dim = 0)





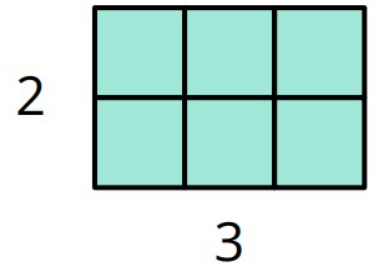
Cagri Ozcinar, PhD

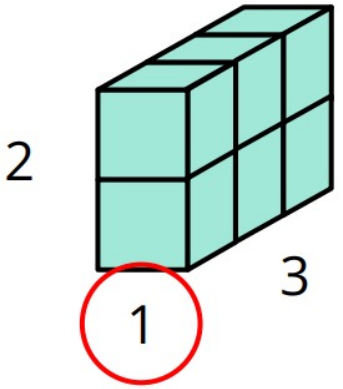
Tensors – Common Operations

- **Unsqueeze**: expand a new dimension

```
>>> x = torch.zeros([2, 3])
>>> x.shape
torch.Size([2, 3])
>>> x = x.unsqueeze(1)
>>> x.shape
torch.Size([2, 1, 3])
```

(dim = 1)



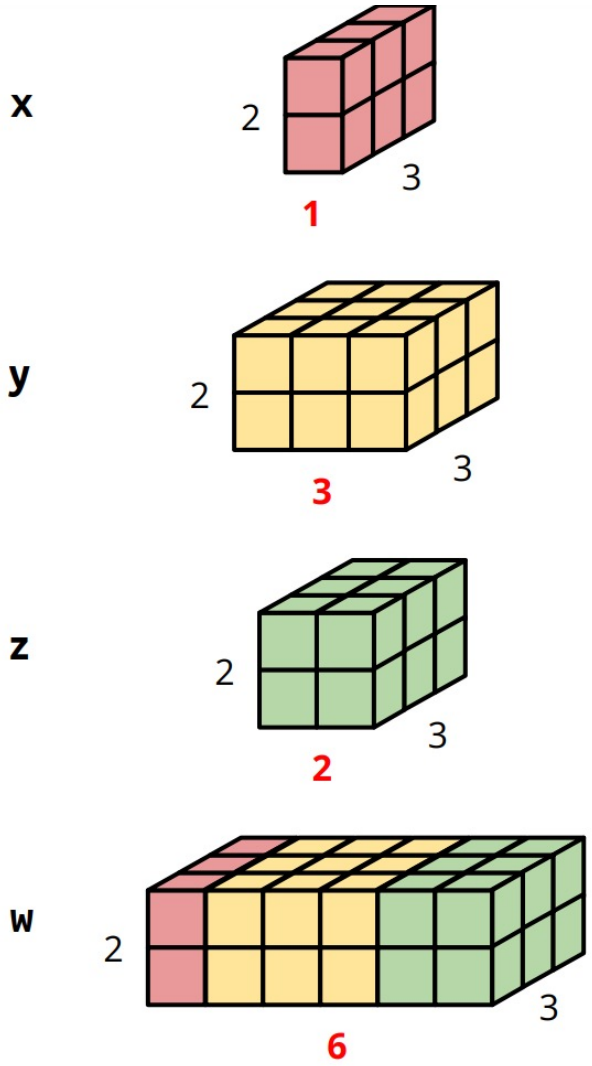


Cagri Ozcinar, PhD

Tensors – Common Operations

- **Cat**: concatenate multiple tensors

```
>>> x = torch.zeros([2, 1, 3])
>>> y = torch.zeros([2, 3, 3])
>>> z = torch.zeros([2, 2, 3])
>>> w = torch.cat([x, y, z], dim=1)
>>> w.shape
torch.Size([2, 6, 3])
```



more operators: https://pytorch.org/docs/stable/tensors.html

Cagri Ozcinar, PhD

Tensors – Data Type

- Using different data types for model and data will cause errors.

Data type	dtype	tensor
32-bit floating point	`torch.float`	`torch.FloatTensor`
64-bit integer (signed)	`torch.long`	`torch.LongTensor`

Cagri Ozcinar, PhD

Tensors – PyTorch vs Numpy

- Similar attributes

PyTorch	NumPy
`x.shape`	`x.shape`
`x.dtype`	`x.dtype`

Cagri Ozcinar, PhD

Tensors – PyTorch vs Numpy

- Many functions have the same names as well

PyTorch	NumPy
`x.reshape / x.view`	`x.reshape`
`x.squeeze()`	`x.squeeze()`
`x.unsqueeze(1)`	`np.expand_dims(x, 1)`

Tensors – Device

- Tensors & modules will be computed with **CPU** by default

  Use **.to()** to move tensors to appropriate devices.

- CPU

```
x = x.to('cpu')
```

- GPU

```
x = x.to('cuda')
```

Cagri Ozcinar, PhD

Tensors – Device (GPU)

- Check if your computer has NVIDIA GPU

```
torch.cuda.is_available()
```

- Multiple GPUs: specify `'cuda:0'`, `'cuda:1'`, `'cuda:2'`, ...

Cagri Ozcinar, PhD

Tensors – Gradient Calculation

1
```
>>> x = torch.tensor([[1., 0.], [-1., 1.]], requires_grad=True)
```
2
```
>>> z = x.pow(2).sum()
```
3
```
>>> z.backward()
```
4
```
>>> x.grad
tensor([[ 2.,  0.],
        [-2.,  2.]])
```

1
$$x = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$$

2
$$z = \sum_i \sum_j x_{i,j}^2$$

3
$$\frac{\partial z}{\partial x_{i,j}} = 2x_{i,j}$$

4
$$\frac{\partial z}{\partial x} = \begin{bmatrix} 2 & 0 \\ -2 & 2 \end{bmatrix}$$

Cagri Ozcinar, PhD

Tensors – Gradient Calculation

1
```
>>> x = torch.tensor([[1., 0.], [-1., 1.]], requires_grad=True)
```
2
```
>>> z = x.pow(2).sum()
```
3
```
>>> z.backward()
```
4
```
>>> x.grad
tensor([[ 2.,  0.],
        [-2.,  2.]])
```

1
$$x = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}$$

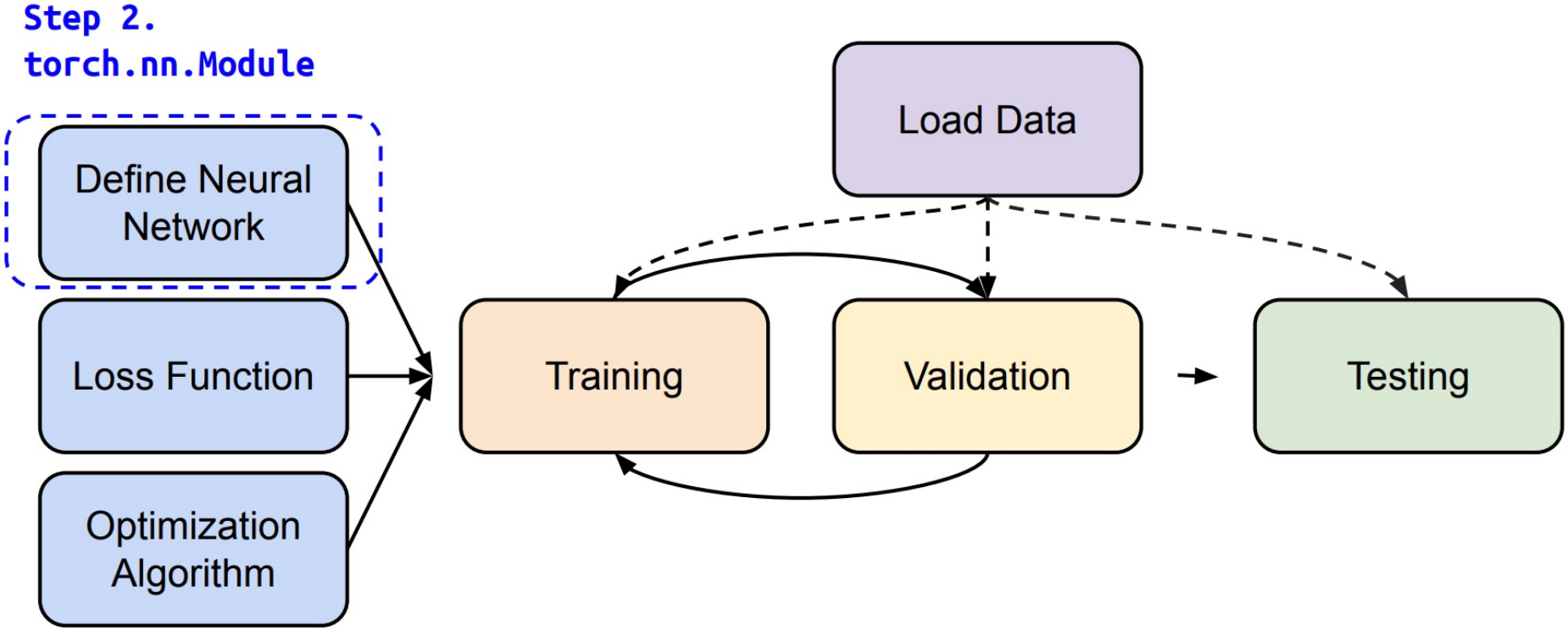
2
$$z = \sum_i \sum_j x_{i,j}^2$$

3
$$\frac{\partial z}{\partial x_{i,j}} = 2x_{i,j}$$

4
$$\frac{\partial z}{\partial x} = \begin{bmatrix} 2 & 0 \\ -2 & 2 \end{bmatrix}$$

Cagri Ozcinar, PhD

Training & Testing NN in PyTorch

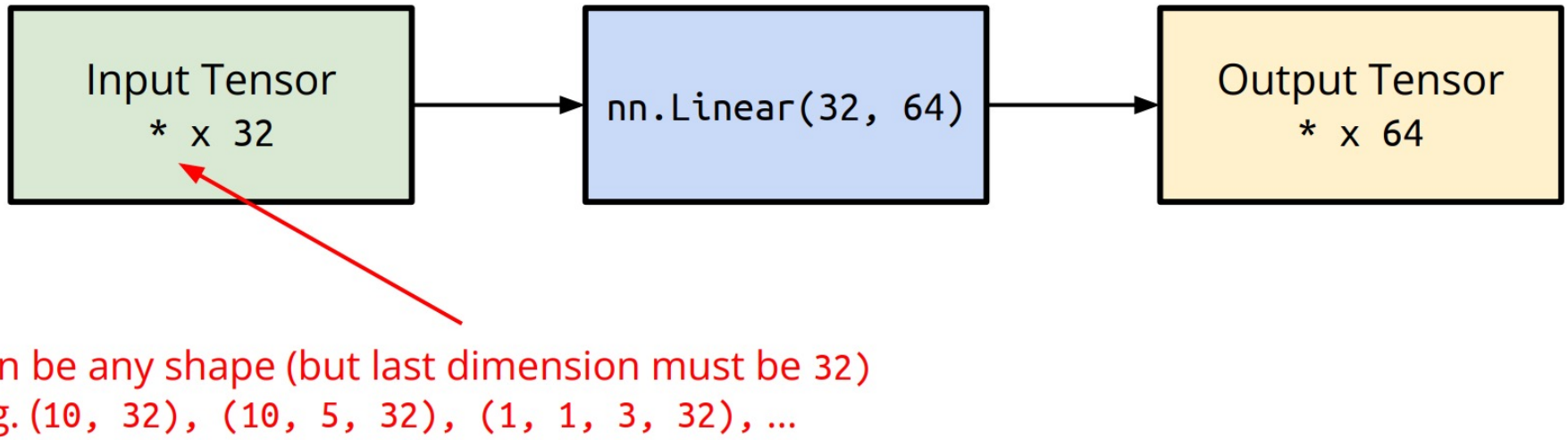


Cagri Ozcinar, PhD

Network Layers

- Linear Layer (**Fully-connected** Layer)

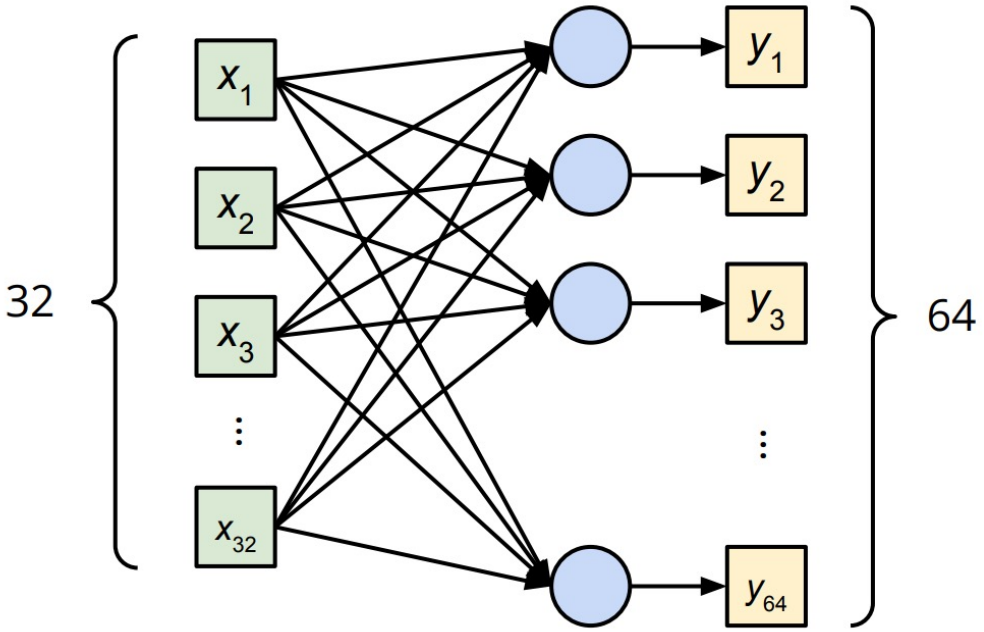
```
nn.Linear(in_features, out_features)
```

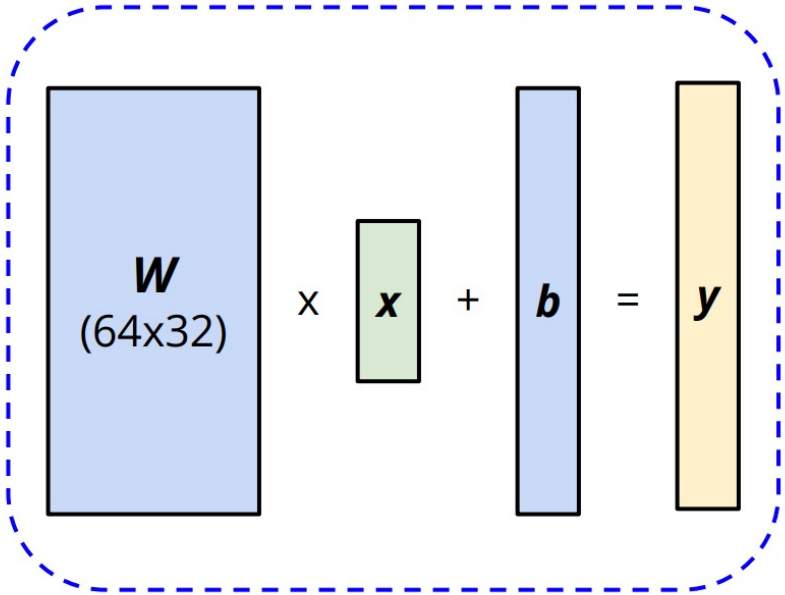


Cagri Ozcinar, PhD

Network Layers

- Linear Layer (**Fully-connected** Layer)



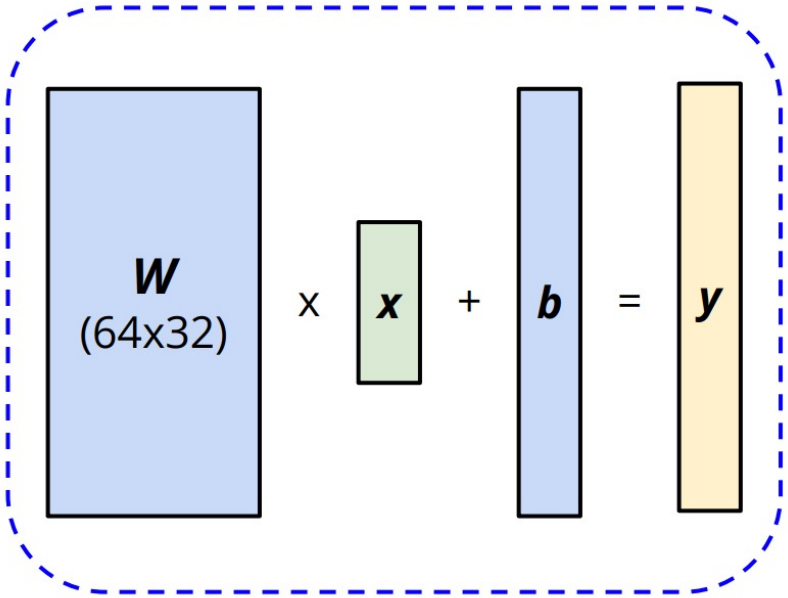


Cagri Ozcinar, PhD

Network Layers

- Linear Layer (**Fully-connected** Layer)

```
>>> layer = torch.nn.Linear(32, 64)
>>> layer.weight.shape
torch.Size([64, 32])
>>> layer.bias.shape
torch.Size([64])
```

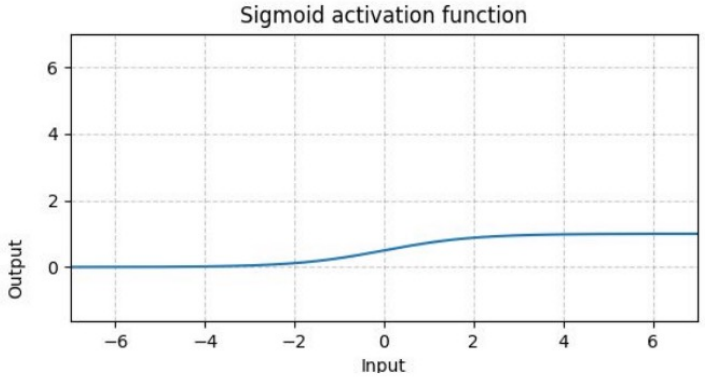


Cagri Ozcinar, PhD

Non-Linear Activation Functions

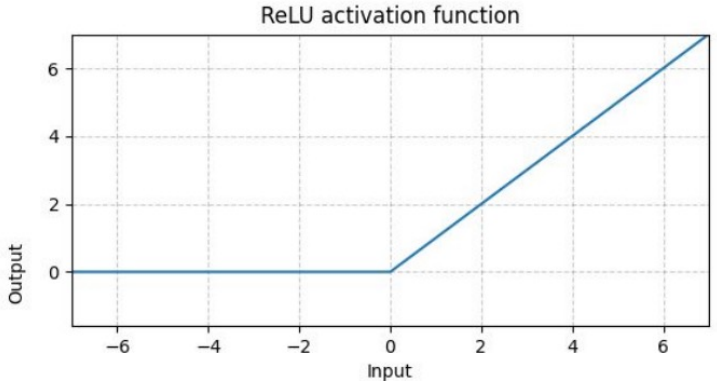
- Sigmoid Activation

```
nn.Sigmoid()
```



- ReLU Activation

```
nn.ReLU()
```



Cagri Ozcinar, PhD

Build your own NN

```
import torch.nn as nn

class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(10, 32),
            nn.Sigmoid(),
            nn.Linear(32, 1)
        )

    def forward(self, x):
        return self.net(x)
```

Initialize your model & define layers

Compute output of your NN

Cagri Ozcinar, PhD

Build your own NN

```
import torch.nn as nn

class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(10, 32),
            nn.Sigmoid(),
            nn.Linear(32, 1)
        )

    def forward(self, x):
        return self.net(x)
```

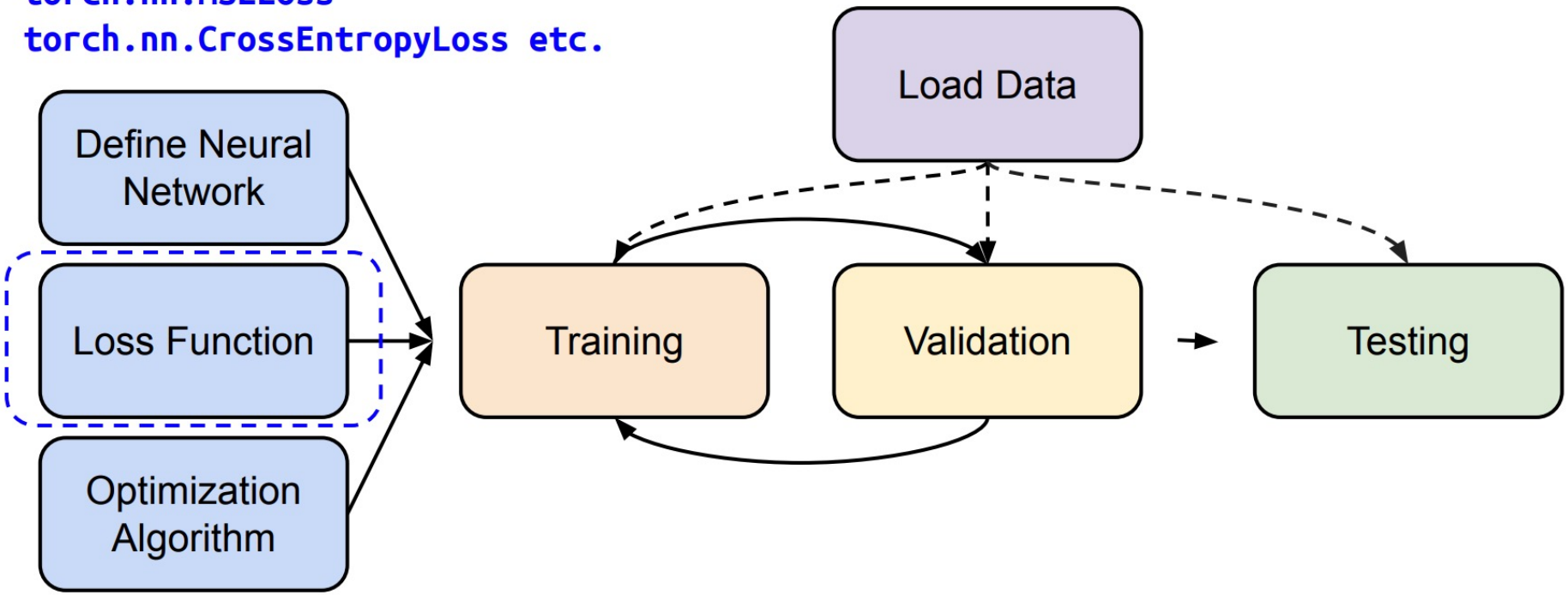
=

```
import torch.nn as nn

class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        self.layer1 = nn.Linear(10, 32)
        self.layer2 = nn.Sigmoid(),
        self.layer3 = nn.Linear(32,1)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        return out
```

Cagri Ozcinar, PhD

Training & Testing NN in PyTorch

```
Step 3.
torch.nn.MSELoss
torch.nn.CrossEntropyLoss etc.
```



Cagri Ozcinar, PhD

Loss Functions

- Mean Squared Error (for regression tasks)

```
criterion = nn.MSELoss()
```

- Cross Entropy (for classification tasks)

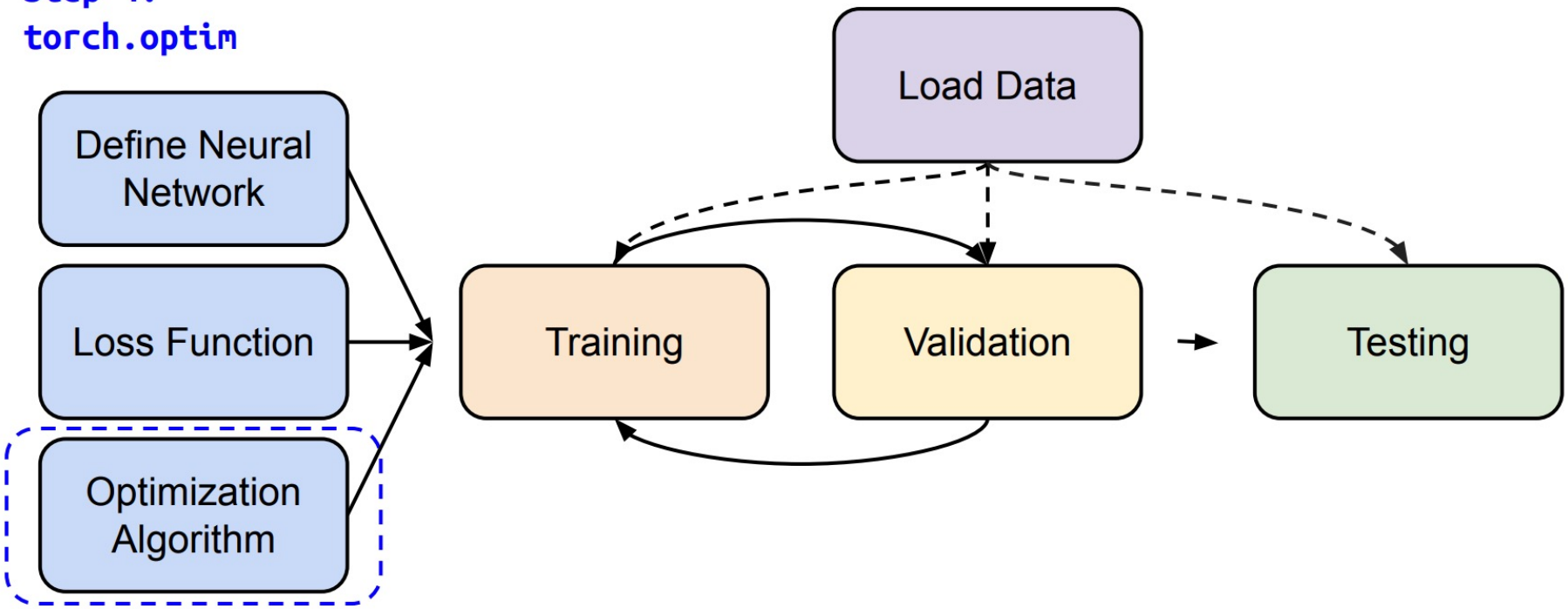
```
criterion = nn.CrossEntropyLoss()
```

- `loss = criterion(model_output, expected_value)`

Cagri Ozcinar, PhD

Training & Testing NN in PyTorch

Step 4.
torch.optim



Cagri Ozcinar, PhD

Optimization

- Gradient-based **optimization algorithms** that adjust network parameters to reduce error.

- E.g. Stochastic Gradient Descent (SGD)

```
torch.optim.SGD(model.parameters(), lr, momentum = 0)
```

Cagri Ozcinar, PhD

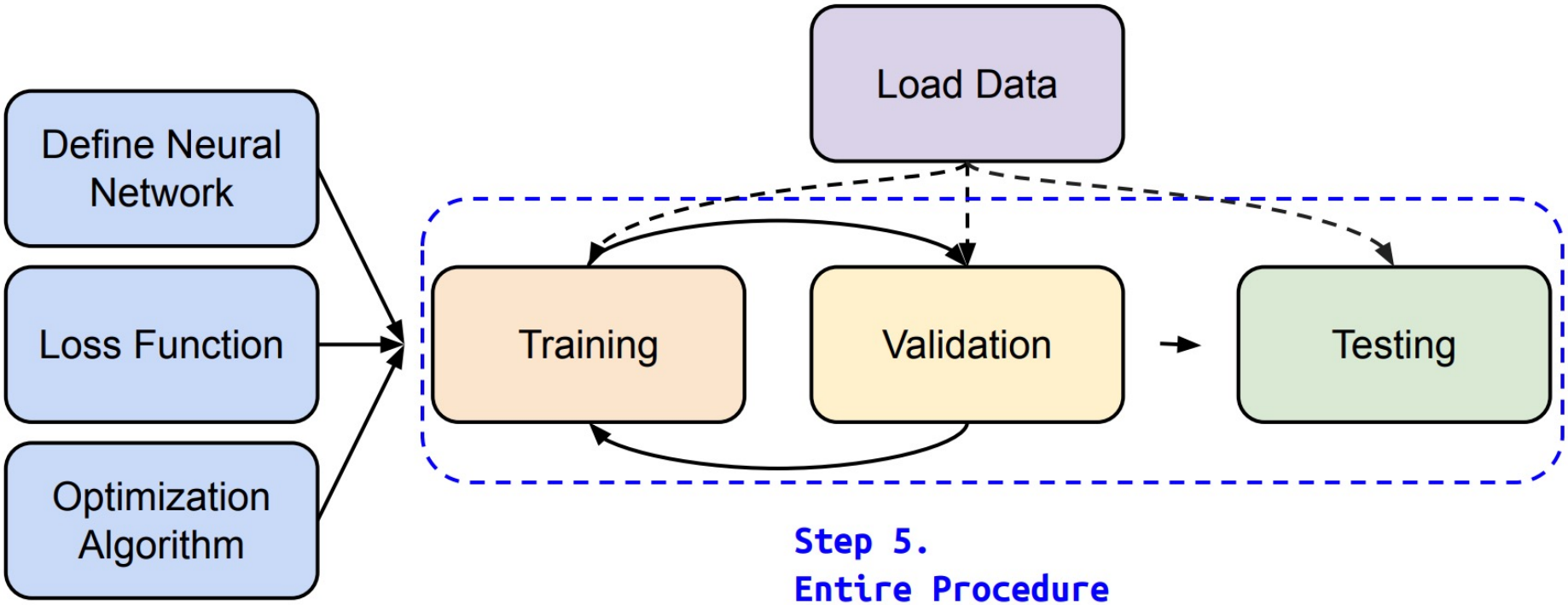
Optimization

```
optimizer = torch.optim.SGD(model.parameters(), lr, momentum = 0)
```

- For every batch of data:
  1. Call `optimizer.zero_grad()` to reset gradients of model parameters.
  2. Call `loss.backward()` to backpropagate gradients of prediction loss.
  3. Call `optimizer.step()` to adjust model parameters.

Cagri Ozcinar, PhD

Training & Testing NN in PyTorch



Cagri Ozcinar, PhD

NN Training Setup

```
dataset = MyDataset(file)                                read data via MyDataset

tr_set = DataLoader(dataset, 16, shuffle=True)           put dataset into Dataloader

model = MyModel().to(device)                             construct model and move to device (cpu/cuda)

criterion = nn.MSELoss()                                 set loss function

optimizer = torch.optim.SGD(model.parameters(), 0.1)     set optimizer
```

Cagri Ozcinar, PhD

NN Training Loop

```
for epoch in range(n_epochs):                    iterate n_epochs
    model.train()                                set model to train mode
    for x, y in tr_set:                          iterate through the dataloader
        optimizer.zero_grad()                    set gradient to zero
        x, y = x.to(device), y.to(device)        move data to device (cpu/cuda)
        pred = model(x)                          forward pass (compute output)
        loss = criterion(pred, y)                compute loss
        loss.backward()                          compute gradient (backpropagation)
        optimizer.step()                         update model with optimizer
```

Cagri Ozcinar, PhD

NN Validation Loop

```
model.eval()                                        set model to evaluation mode

total_loss = 0

for x, y in dv_set:                                 iterate through the dataloader

    x, y = x.to(device), y.to(device)               move data to device (cpu/cuda)

    with torch.no_grad():                           disable gradient calculation

        pred = model(x)                             forward pass (compute output)

        loss = criterion(pred, y)                   compute loss

    total_loss += loss.cpu().item() * len(x)        accumulate loss

    avg_loss = total_loss / len(dv_set.dataset)     compute averaged loss
```

Cagri Ozcinar, PhD

NN Testing Loop

```
model.eval()                                    set model to evaluation mode

preds = []

for x in tt_set:                                iterate through the dataloader

    x = x.to(device)                            move data to device (cpu/cuda)

    with torch.no_grad():                       disable gradient calculation

        pred = model(x)                         forward pass (compute output)

        preds.append(pred.cpu())                collect prediction
```

Cagri Ozcinar, PhD

- `model.eval()`

  Changes behaviour of some model layers, such as dropout and batch normalization.

- `with torch.no_grad()`

  Prevents calculations from being added into gradient computation graph. Usually used to prevent accidental training on validation/testing data.

Cagri Ozcinar, PhD

Save/Load Trained Models

- Save

```
torch.save(model.state_dict(), path)
```

- Load

```
ckpt = torch.load(path)
model.load_state_dict(ckpt)
```

Cagri Ozcinar, PhD

More About PyTorch

- torchaudio
  - speech/audio processing
- torchtext
  - natural language processing
- torchvision
  - computer vision
- skorch
  - scikit-learn + pyTorch

Cagri Ozcinar, PhD

More About PyTorch

- torchaudio
  - speech/audio processing
- torchtext
  - natural language processing
- torchvision
  - computer vision
- skorch
  - scikit-learn + pyTorch