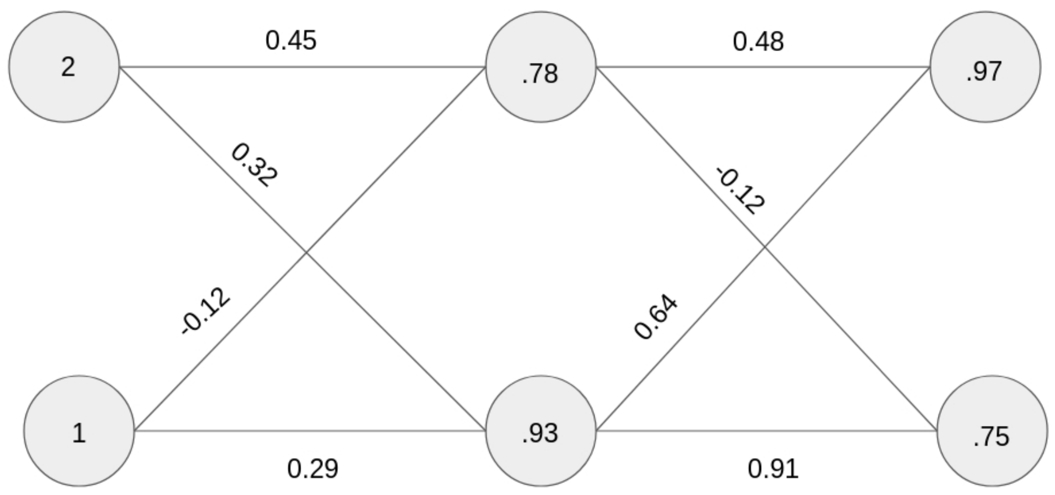
# Activation functions

INTRODUCTION TO DEEP LEARNING WITH PYTORCH

**Ismail Elezi**
Ph.D. Student of Deep Learning

# Motivation



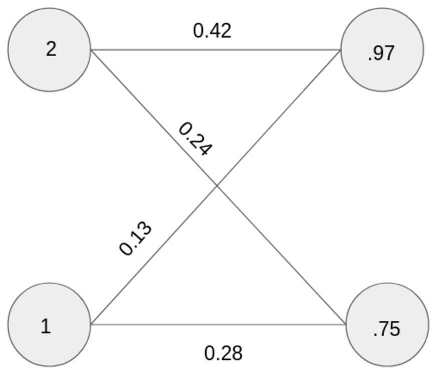

```
input_layer = torch.tensor([2., 1.])
weight_1 = torch.tensor([[0.45, 0.32], [-0.12, 0.29]])
hidden_layer = torch.matmul(input_layer, weight_1)
weight_2 = torch.tensor([[0.48, -0.12], [0.64, 0.91]])
output_layer = torch.matmul(hidden_layer, weight_2)
print(output_layer)
```

```
tensor([ 0.9696,  0.7527])
```

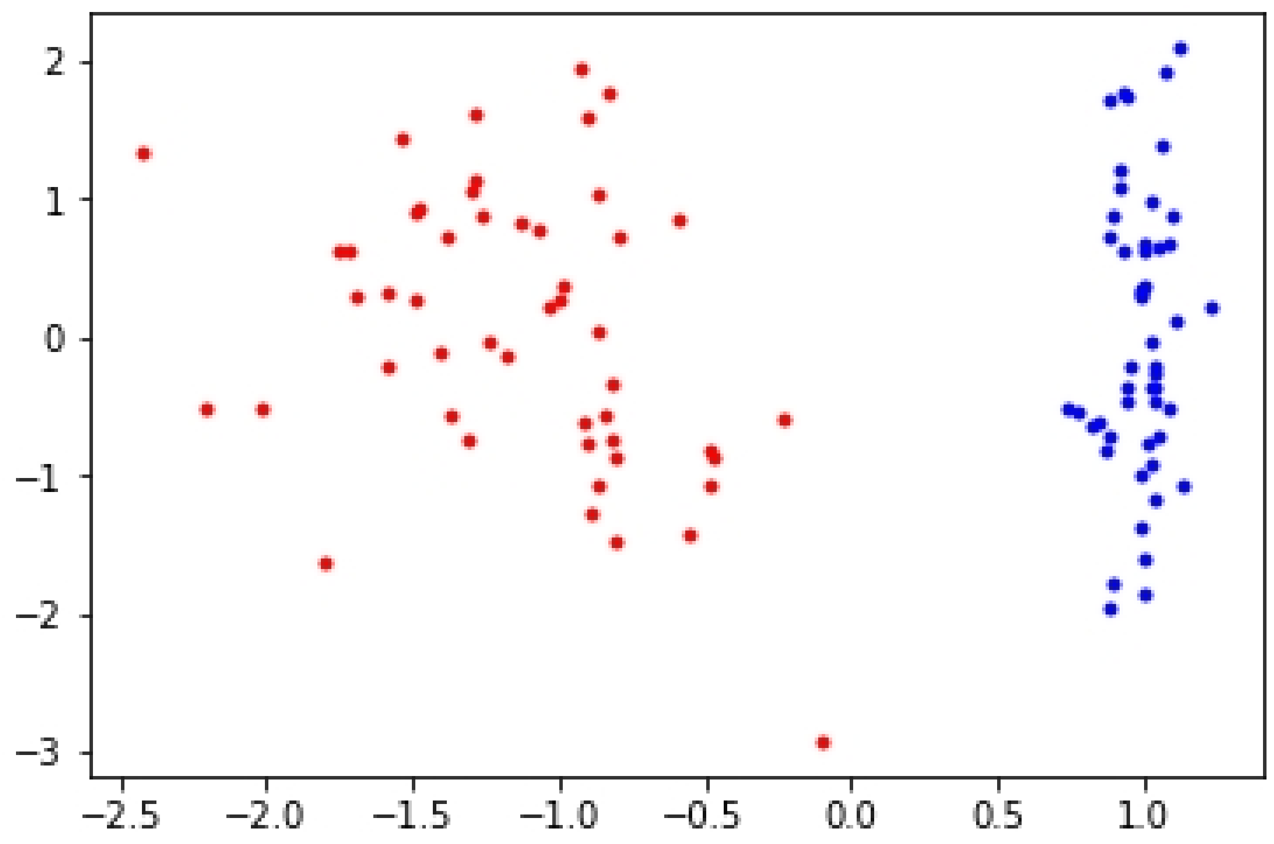
# Matrix multiplication is a linear transformation

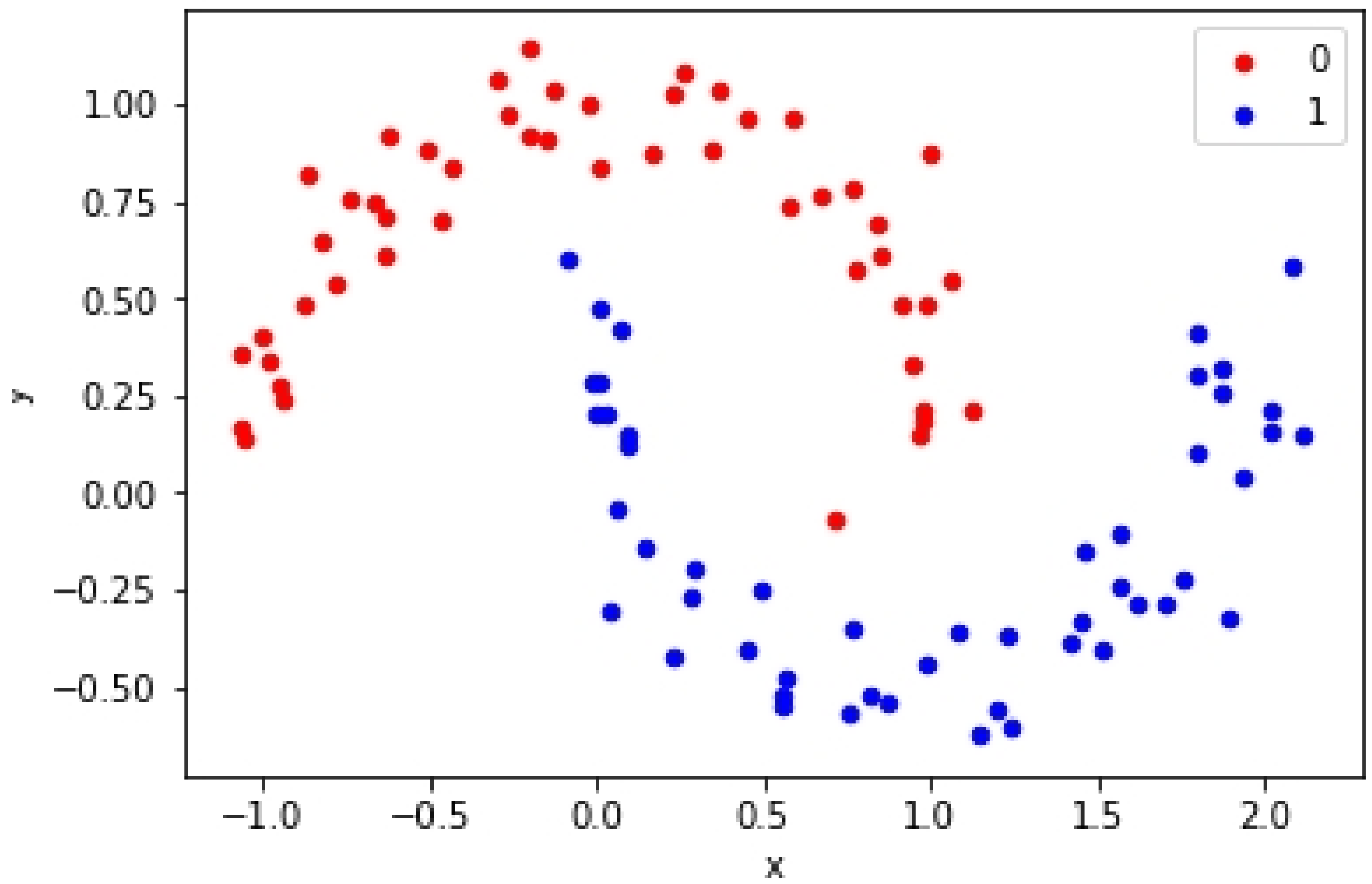
```
input_layer = torch.tensor([2., 1.])
weight_1 = torch.tensor([[0.45, 0.32], [-0.12, 0.29]])
weight_2 = torch.tensor([[0.48, -0.12], [0.64, 0.91]])
weight = torch.matmul(weight_1, weight_2)
output_layer = torch.matmul(input_layer, weight)
print(output_layer)
print(weight)
```

```
tensor([ 0.9696,  0.7527])
tensor([[ 0.4208,  0.2372], [ 0.1280,  0.2783]])
```



# Non linearly separable datasets

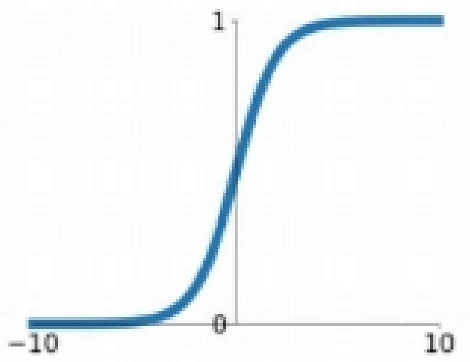


# Activation functions

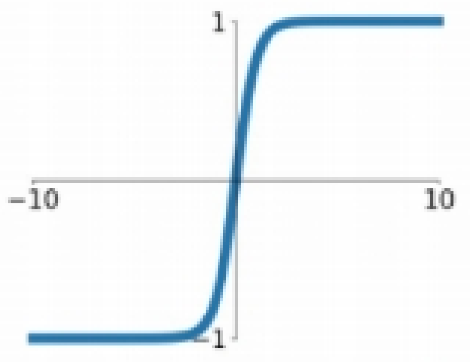
**Sigmoid**

$\sigma(x) = \frac{1}{1+e^{-x}}$



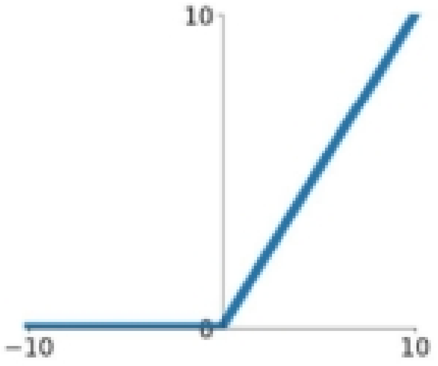

**tanh**

$\tanh(x)$



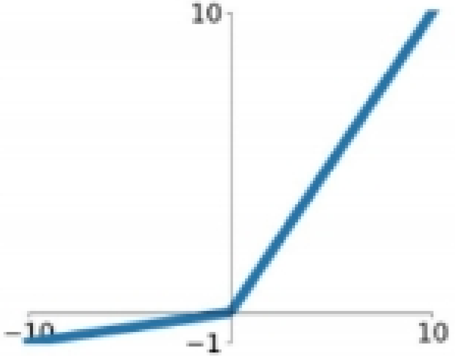

**ReLU**

$\max(0, x)$




**Leaky ReLU**

$\max(0.1x, x)$

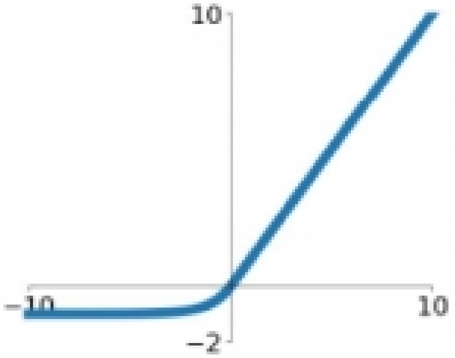



**Maxout**

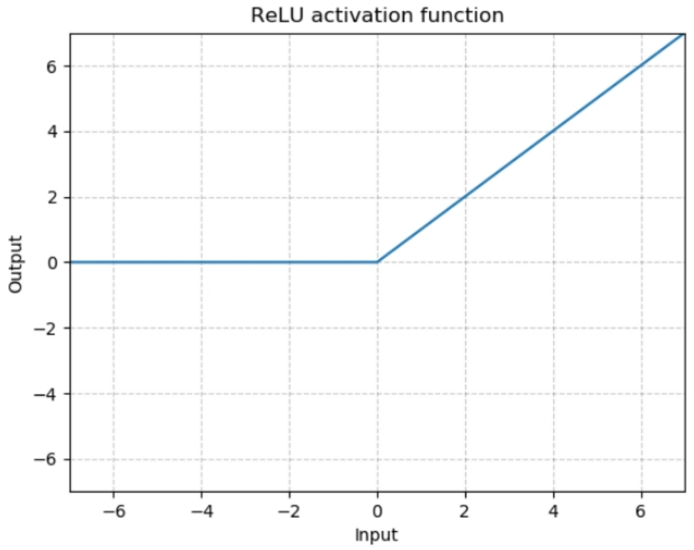
$\max(w_1^T x + b_1, w_2^T x + b_2)$

**ELU**

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



# ReLU activation function




`ReLU(x) = max(0, x)`

```python
import torch.nn as nn
relu = nn.ReLU()

tensor_1 = torch.tensor([2., -4.])
print(relu(tensor_1))

tensor_2 = torch.tensor([[2., -4.], [1.2, 0.]])
print(relu(tensor_2))
```

```
tensor([ 2.,  0.])


tensor([[ 2.0000,  0.0000],
        [ 1.2000,  0.0000]])
```

# Let us implement some activation functions

INTRODUCTION TO DEEP LEARNING WITH PYTORCH

# Loss functions

INTRODUCTION TO DEEP LEARNING WITH PYTORCH

**Ismail Elezi**
Ph.D. Student of Deep Learning

# Loss Functions

- Initialize neural networks with random weights.
- Do a forward pass.
- Calculate loss function (1 number).
- Calculate the gradients.
- Change the weights based on gradients.

- For regression: least squared loss.
- For classification: softmax cross-entropy loss.
- For more complicated problems (like object detection), more complicated losses.

# Softmax Cross-Entropy Loss

| | |
|---|---|
| cat | **3.2** |
| car | 5.1 |
| frog | -1.7 |

$$\boxed{P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}}$$ Softmax Function

# Softmax Cross-Entropy Loss

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax Function

Probabilities must be >= 0

| | | | |
|---|---|---|---|
| cat | **3.2** | | **24.5** |
| car | 5.1 | $\xrightarrow{\text{exp}}$ | 164.0 |
| frog | -1.7 | | 0.18 |

unnormalized probabilities

# Softmax Cross-Entropy Loss

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax Function

| | | exp | Probabilities must be >= 0 | normalize | Probabilities must sum to 1 |
|---|---|---|---|---|---|
| cat | **3.2** | | **24.5** | | **0.13** |
| car | 5.1 | | 164.0 | | 0.87 |
| frog | -1.7 | | 0.18 | | 0.00 |
| | | | unnormalized probabilities | | probabilities |

# Softmax Cross-Entropy Loss

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax Function

| | | Probabilities must be >= 0 | Probabilities must sum to 1 |
|---|---|---|---|
| cat | **3.2** | **24.5** | **0.13** |
| car | 5.1 | 164.0 | 0.87 |
| frog | -1.7 | 0.18 | 0.00 |
| | | unnormalized probabilities | probabilities |

$\xrightarrow{\text{exp}}$ $\xrightarrow{\text{normalize}}$

L = - ln(0.13) = 2.0404

# CE loss in PyTorch

```
logits = torch.tensor([[3.2, 5.1, -1.7]])
ground_truth = torch.tensor([0])
criterion = nn.CrossEntropyLoss()

loss = criterion(logits, ground_truth)
print(loss)
```

```
tensor(2.0404)
```

# CE loss in PyTorch

```
logits = torch.tensor([[10.2, 5.1, -1.7]])
loss = criterion(logits, ground_truth)
print(loss)
```

```
tensor(0.0061)
```

```
logits = torch.tensor([[-10, 5.1, -1.7]])
loss = criterion(logits, ground_truth)
print(loss)
```

```
tensor(15.1011)
```

# Let's practice!

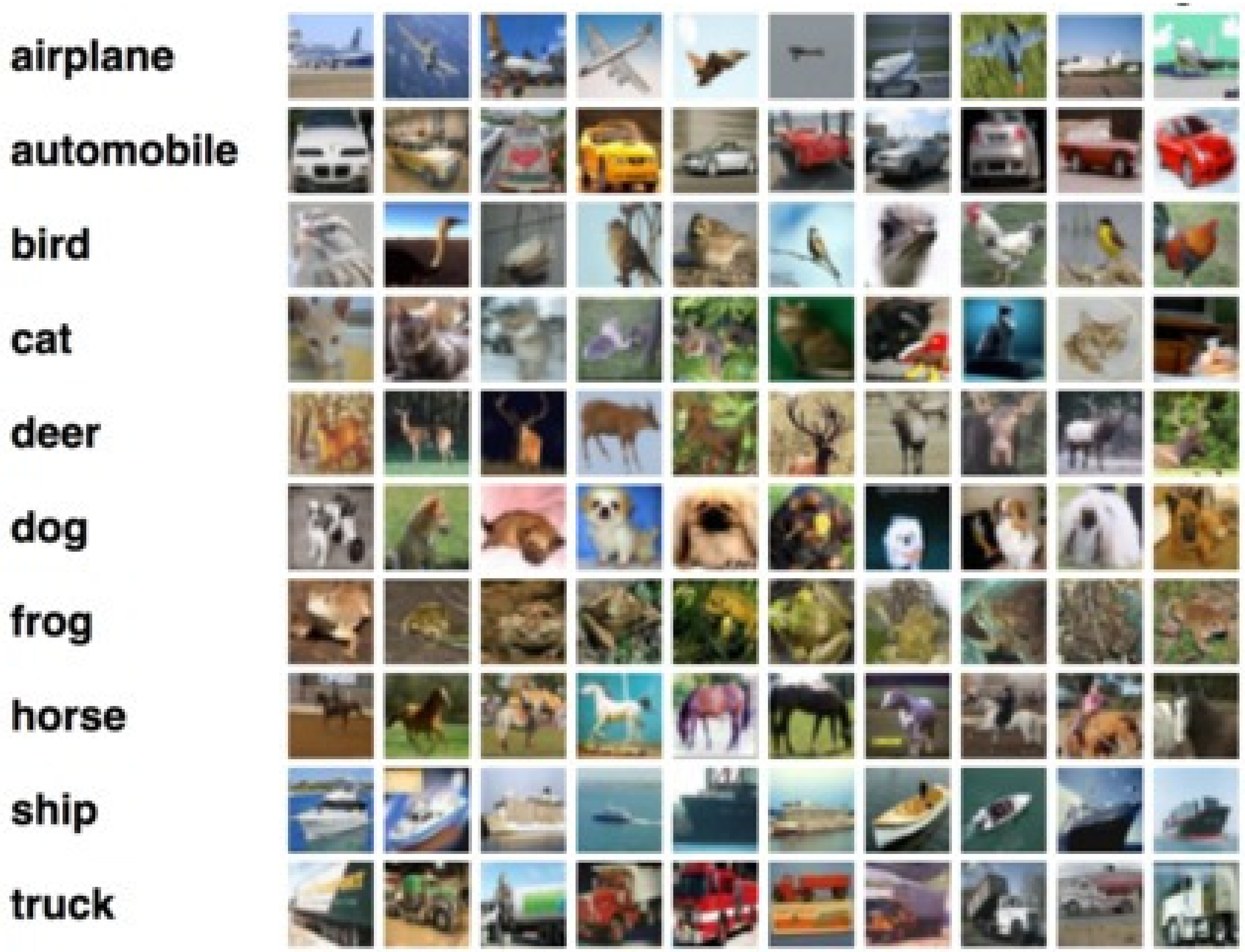
INTRODUCTION TO DEEP LEARNING WITH PYTORCH

# Preparing a dataset in PyTorch

INTRODUCTION TO DEEP LEARNING WITH PYTORCH

**Ismail Elezi**
Ph.D. Student of Deep Learning

# MNIST and CIFAR-10



# Datasets and Dataloaders

```
import torch
import torchvision
import torch.utils.data
import torchvision.transforms as transforms
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.4914, 0.48216, 0.44653),
                          (0.24703, 0.24349, 0.26159))])
```

# Datasets and Dataloaders

```
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=32,
                                          shuffle=True, num_workers=4)

testloader = torch.utils.data.DataLoader(testset, batch_size=32,
                                         shuffle=False, num_workers=4)
```

# Inspecting the dataloader

```
print(testloader.dataset.test_data.shape, trainloader.dataset.train_data.shape)
```

```
(10000, 32, 32, 3), (50000, 32, 32, 3)
```

```
print(testloader.batch_size)
```

```
32
```

```
print(trainloader.sampler)
```

```
<torch.utils.data.sampler.RandomSampler object at 0x7f0612fb85c0>
```

# Let's practice!

INTRODUCTION TO DEEP LEARNING WITH PYTORCH

# Training neural networks

INTRODUCTION TO DEEP LEARNING WITH PYTORCH

**Ismail Elezi**
Ph.D. Student of Deep Learning

datacamp

# Recipe for training neural networks

- Prepare the dataloaders.
- Build a neural network.

Loop over:

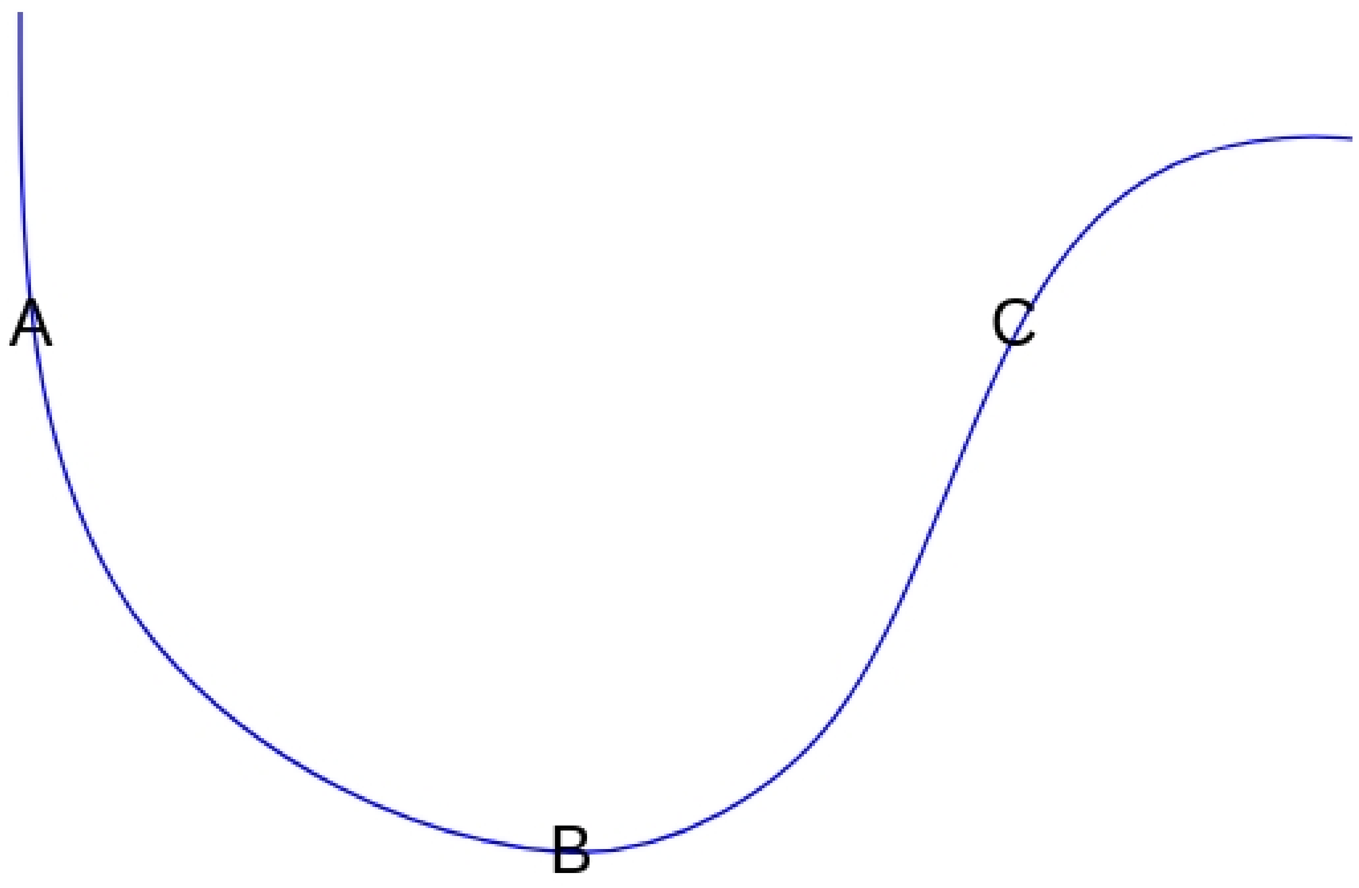
- Do a forward pass.
- Calculate loss function (1 number).
- Calculate the gradients.
- Change the weights based on gradients.

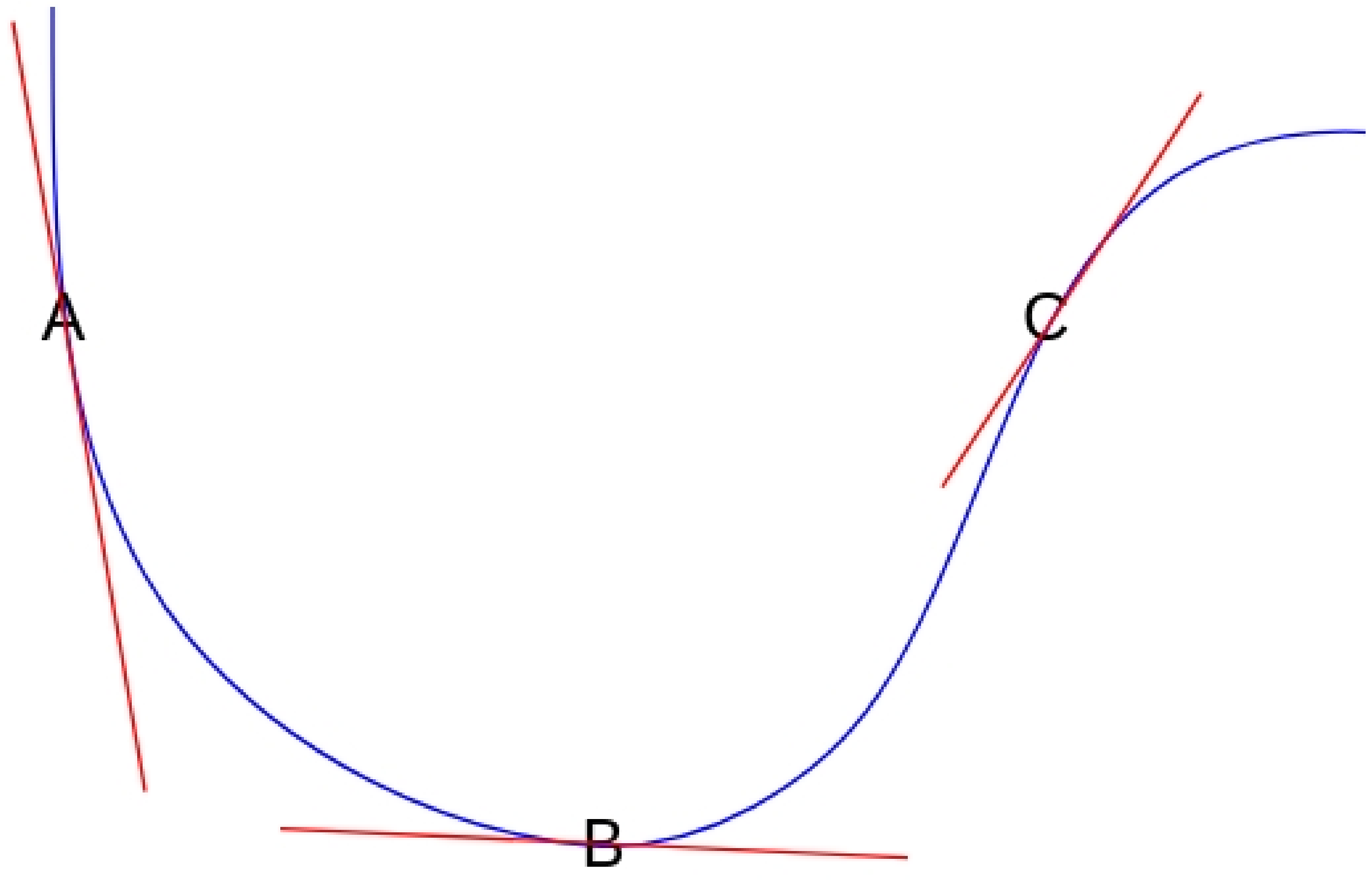
- Lesson 2.3.
- Lesson 1.4 and Lesson 2.1.

Loop over:

- Lesson 1.2.
- Lesson 2.2.
- Lesson 1.3.
- weight -= weight_gradient * learning_rate.

# Gradient descent



# Recap - Dataloaders

```
import torch
import torchvision
import torch.utils.data
import torchvision.transforms as transforms

transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.4914, 0.48216, 0.44653), (0.24703, 0.24349, 0.26159))])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)

testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)

trainloader = torch.utils.data.DataLoader(trainset, batch_size=32, shuffle=True, num_workers=4)

testloader = torch.utils.data.DataLoader(testset, batch_size=100, shuffle=False, num_workers=4)
```

# Neural Networks - Recap

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(32 * 32 * 3, 500)
        self.fc2 = nn.Linear(500, 10)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        return self.fc2(x)
```

# Training the Neural Network

```
net = Net()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=3e-4)

for epoch in range(10):  # loop over the dataset multiple times
    for i, data in enumerate(trainloader, 0):
        # Get the inputs
        inputs, labels = data
        inputs = inputs.view(-1, 32 * 32 * 3)

        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

# Using the net to get predictions

```
correct, total = 0, 0
predictions = []
net.eval()
for i, data in enumerate(testloader, 0):
    inputs, labels = data
    inputs = inputs.view(-1, 32*32*3)
    outputs = net(inputs)
    _, predicted = torch.max(outputs.data, 1)
    predictions.append(outputs)
    total += labels.size(0)
    correct += (predicted == labels).sum().item()

print('The testing set accuracy of the network is: %d %%' % (100 * correct / total))
```

```
The testing set accuracy of the network is: 53 %
```

# Let's practice!

INTRODUCTION TO DEEP LEARNING WITH PYTORCH