

IMPORTANT ELEMENTS OF NEURAL NETWORK

- Loss Function



- Optimizer

- Activation Function



IMPORTANT ELEMENTS OF NEURAL NETWORK

- Loss Function



- Optimizer

- Activation Function



Pytorch.ipynb - Colab

Optimizer.pdf

File

C:/Users/mg/Downloads/Optimizer.pdf.pdf

Paused

Optimizer.pdf

1 / 2 | - 54% + |

Optimizers


An optimizer, in the context of machine learning and neural networks, refers to an algorithm used to minimize the loss function (error) during the training process. Its primary function is to adjust the model's parameters (weights and biases) iteratively to reduce the error and improve the model's accuracy and performance on the training data.

Key Functions of an Optimizer:


1. Loss Minimization:
 - The optimizer adjusts the weights and biases of the neural network based on the gradients of the loss function with respect to these parameters. Its goal is to find the optimal set of parameters that minimize the error between the predicted outputs and the actual target values.
2. Gradient Descent:
 - Most optimizers use some form of gradient descent, a fundamental optimization technique in machine learning. Gradient descent computes the gradient (derivative) of the loss function with respect to each parameter and updates the parameters in the direction that reduces the loss.
3. Learning Rate Control:
 - Optimizers often include parameters such as a learning rate, which controls the size of the steps taken during gradient descent. The learning rate determines how quickly or slowly the optimizer adjusts the weights. Choosing an appropriate learning rate is crucial for efficient training.

Types of Optimizers:

1. Gradient Descent Variants:
 - Stochastic Gradient Descent (SGD): Updates the parameters after computing the gradient based on a subset (batch) of training data.
 - Batch Gradient Descent: Computes the gradient using the entire training dataset but updates the parameters only after processing the entire batch.



WWW.AISCIENCES.IO



Udemy

Pytorch.ipynb - Colab

Optimizer.pdf

File

C:/Users/mg/Downloads/Optimizer.pdf.pdf

Search

Star

Share

Paused

Optimizer.pdf

1 / 2 | - 54% + |

Download

Print


An optimizer, in the context of machine learning and neural networks, refers to an algorithm used to minimize the loss function (error) during the training process. Its primary function is to adjust the model's parameters (weights and biases) iteratively to reduce the error and improve the model's accuracy and performance on the training data.

Key Functions of an Optimizer:


1. Loss Minimization:
 - The optimizer adjusts the weights and biases of the neural network based on the gradients of the loss function with respect to these parameters. Its goal is to find the optimal set of parameters that minimize the error between the predicted outputs and the actual target values.
2. Gradient Descent:
 - Most optimizers use some form of gradient descent, a fundamental optimization technique in machine learning. Gradient descent computes the gradient (derivative) of the loss function with respect to each parameter and updates the parameters in the direction that reduces the loss.
3. Learning Rate Control:
 - Optimizers often include parameters such as a learning rate, which controls the size of the steps taken during gradient descent. The learning rate determines how quickly or slowly the optimizer adjusts the weights. Choosing an appropriate learning rate is crucial for efficient training.

Types of Optimizers:

1. Gradient Descent Variants:
 - Stochastic Gradient Descent (SGD): Updates the parameters after computing the gradient based on a subset (batch) of training data.
 - Batch Gradient Descent: Computes the gradient using the entire training dataset but updates the parameters only after processing the entire batch.



WWW.AISCIENCES.IO



udemy

- Mini-batch Gradient Descent: A compromise between SGD and batch GD, where the gradient is computed and parameters updated for

Pytorch.ipynb - Colab

Optimizer.pdf

File

C:/Users/mg/Downloads/Optimizer.pdf.pdf

Paused

Optimizer.pdf

1 / 2 | - 68% + |

An optimizer, in the context of machine learning and neural networks, refers to an algorithm used to minimize the loss function (error) during the training process. Its primary function is to adjust the model's parameters (weights and biases) iteratively to reduce the error and improve the model's accuracy and performance on the training data.

Key Functions of an Optimizer:

- 1. Loss Minimization:**
 - The optimizer adjusts the weights and biases of the neural network based on the gradients of the loss function with respect to these parameters. Its goal is to find the optimal set of parameters that minimize the error between the predicted outputs and the actual target values.
- 2. Gradient Descent:**
 - Most optimizers use some form of gradient descent, a fundamental optimization technique in machine learning. Gradient descent computes the gradient (derivative) of the loss function with respect to each parameter and updates the parameters in the direction that reduces the loss.
- 3. Learning Rate Control:**
 - Optimizers often include parameters such as a learning rate, which controls the size of the steps taken during gradient descent. The learning rate determines how quickly or slowly the optimizer adjusts the weights. Choosing an appropriate learning rate is crucial for efficient training.

Types of Optimizers:

- 1. Gradient Descent Variants:**
 - Stochastic Gradient Descent (SGD):** Updates the parameters after computing the gradient based on a subset (batch) of training data.
 - Batch Gradient Descent:** Computes the gradient using the entire training dataset but updates the parameters only after processing the entire batch.

WWW.AISCIENCES.IO

Udemy

Pytorch.ipynb - Colab

Optimizer.pdf

File

C:/Users/mg/Downloads/Optimizer.pdf.pdf

Paused

Optimizer.pdf

1 / 2 | 68%


An optimizer, in the context of machine learning and neural networks, refers to an algorithm used to minimize the loss function (error) during the training process. Its primary function is to adjust the model's parameters (weights and biases) iteratively to reduce the error and improve the model's accuracy and performance on the training data.

Key Functions of an Optimizer:


1. Loss Minimization:
 - The optimizer adjusts the weights and biases of the neural network based on the gradients of the loss function with respect to these parameters. Its goal is to find the optimal set of parameters that minimize the error between the predicted outputs and the actual target values.
2. Gradient Descent:
 - Most optimizers use some form of gradient descent, a fundamental optimization technique in machine learning. Gradient descent computes the gradient (derivative) of the loss function with respect to each parameter and updates the parameters in the direction that reduces the loss.
3. Learning Rate Control:
 - Optimizers often include parameters such as a learning rate, which controls the size of the steps taken during gradient descent. The learning rate determines how quickly or slowly the optimizer adjusts the weights. Choosing an appropriate learning rate is crucial for efficient training.

Types of Optimizers:

1. Gradient Descent Variants:
 - Stochastic Gradient Descent (SGD): Updates the parameters after computing the gradient based on a subset (batch) of training data.
 - Batch Gradient Descent: Computes the gradient using the entire training dataset but updates the parameters only after processing the entire batch.



WWW.AISCIENCES.IO



Udemy

Pytorch.ipynb - Colab

Optimizer.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

RAM

Disk

Paused

+ Code

+ Text

All changes saved

0s

```
model = SimpleNN()
for name, params in model.named_parameters():
    print("Name:", name)
    print("Param", params)

# #Replacing MSE with built-in function
# # mse = torch.nn.MSELoss()
# mae = torch.nn.L1Loss()

# # Training step
# for i in range(1000):
#     preds = model(inputs)
#     loss = mae(preds, targets)
#     loss.backward()
#     if i%100 == 99:
#         print(loss.item())

#     with torch.no_grad():
#         w -= w.grad * 0.00001
#         b -= b.grad * 0.00001
#         w.grad.zero_()
#         b.grad.zero_()

# preds = model(inputs)
# loss = mae(preds, targets)
# print(loss)
```

Name: linear1.weight

Param: Parameter containing:

completed at 3:45 PM

Udemy

Pytorch.ipynb - Colab

Optimizer.pdf

File

C:/Users/mg/Downloads/Optimizer.pdf.pdf

68%

1 / 2

Paused

Optimizer.pdf

1

2

68%

Download


Print

based on the gradients of the loss function with respect to these parameters. Its goal is to find the optimal set of parameters that minimize the error between the predicted outputs and the actual target values.

2. Gradient Descent:
 - Most optimizers use some form of gradient descent, a fundamental optimization technique in machine learning. Gradient descent computes the gradient (derivative) of the loss function with respect to each parameter and updates the parameters in the direction that reduces the loss.
3. Learning Rate Control:
 - Optimizers often include parameters such as a learning rate, which controls the size of the steps taken during gradient descent. The learning rate determines how quickly or slowly the optimizer adjusts the weights. Choosing an appropriate learning rate is crucial for efficient training.

Types of Optimizers:

1. Gradient Descent Variants:
 - Stochastic Gradient Descent (SGD): Updates the parameters after computing the gradient based on a subset (batch) of training data.
 - Batch Gradient Descent: Computes the gradient using the entire training dataset but updates the parameters only after processing the entire batch.



WWW.AISCIONES.IO

- Mini-batch Gradient Descent: A compromise between SGD and batch GD, where the gradient is computed and parameters updated for smaller batches of data.

2. Adaptive Learning Rate Methods:

AISCIONES

Udemy

Pytorch.ipynb - Colab

Optimizer.pdf

File

C:/Users/mg/Downloads/Optimizer.pdf.pdf

Paused

Optimizer.pdf


2 / 2 | 68%

Types of Optimizers:

1. Gradient Descent Variants

o Stochastic Gradient Descent (SGD): Updates the parameters after computing the gradient based on a subset (batch) of training data.

o Batch Gradient Descent: Computes the gradient using the entire training dataset but updates the parameters only after processing the entire batch.



WWW.AISCIONES.IO

o Mini-batch Gradient Descent: A compromise between SGD and batch GD, where the gradient is computed and parameters updated for smaller batches of data.

2. Adaptive Learning Rate Methods:

o Adam (Adaptive Moment Estimation): Combines the advantages of adaptive learning rates and momentum methods to accelerate convergence.


o RMSprop (Root Mean Square Propagation): Optimizer that adjusts the learning rate for each parameter based on the average of recent magnitudes of the gradients.

o Adagrad (Adaptive Gradient Algorithm): Adjusts the learning rate dynamically for each parameter, based on the history of gradients for that parameter.

3. Momentum-based Methods:

o Momentum: Accumulates a weighted average of past gradients to determine the direction of updates, helping to accelerate convergence, especially in the presence of noise.

Choosing an Optimizer:



Udemy

Pytorch.ipynb - Colab

Optimizer.pdf

File

C:/Users/mg/Downloads/Optimizer.pdf.pdf

Paused

Optimizer.pdf


2 / 2 | 68%

Types of Optimizers:

1. Gradient Descent Variants:

o Stochastic Gradient Descent (SGD): Updates the parameters after computing the gradient based on a subset (batch) of training data.

o Batch Gradient Descent: Computes the gradient using the entire training dataset but updates the parameters only after processing the entire batch.



WWW.AISCIONES.IO

o Mini-batch Gradient Descent: A compromise between SGD and batch GD, where the gradient is computed and parameters updated for smaller batches of data.

2. Adaptive Learning Rate Methods:

o Adam (Adaptive Moment Estimation): Combines the advantages of adaptive learning rates and momentum methods to accelerate convergence.


o RMSprop (Root Mean Square Propagation): Optimizer that adjusts the learning rate for each parameter based on the average of recent magnitudes of the gradients.

o Adagrad (Adaptive Gradient Algorithm): Adjusts the learning rate dynamically for each parameter, based on the history of gradients for that parameter.

3. Momentum-based Methods:

o Momentum: Accumulates a weighted average of past gradients to determine the direction of updates, helping to accelerate convergence, especially in the presence of noise.

Choosing an Optimizer:



Udemy

Pytorch.ipynb - Colab

Optimizer.pdf

File

C:/Users/mg/Downloads/Optimizer.pdf.pdf

68%

2 / 2

Paused

Optimizer.pdf

2 / 2 | 68% +

Download Print

Adaptive Learning Rate Methods:

- o Adam (Adaptive Moment Estimation): Combines the advantages of adaptive learning rates and momentum methods to accelerate convergence.
- o RMSprop (Root Mean Square Propagation): Optimizer that adjusts the learning rate for each parameter based on the average of recent magnitudes of the gradients.
- o Adagrad (Adaptive Gradient Algorithm): Adjusts the learning rate dynamically for each parameter, based on the history of gradients for that parameter.

3. Momentum-based Methods:

- o Momentum: Accumulates a weighted average of past gradients to determine the direction of updates, helping to accelerate convergence, especially in the presence of noise.

Choosing an Optimizer:

- Task Specific: The choice of optimizer depends on the specific task, the dataset size, and the network architecture.
- Experimentation: It's common to experiment with different optimizers and their configurations to find the one that achieves the best results in terms of convergence speed and model accuracy.
- Tuning: Optimizers often have hyperparameters (e.g., learning rate, momentum), which may require tuning to achieve optimal performance.

Summary:

In summary, an optimizer is a crucial component in training neural networks and other machine learning models. It adjusts the model's parameters to minimize the error (loss function) by iteratively updating them based on gradients computed from training data. Optimizers vary in their approaches, from basic gradient descent variants to more sophisticated adaptive methods, each designed to improve training efficiency and convergence speed.

WWW AISCIENCES IO

AI SCIENCES

Udemy

Pytorch.ipynb - Colab

Optimizer.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

RAM

Disk

Paused

+ Code + Text

0s [3] [115., 140.], [76., 85.], [65., 75.]]

#Replacing Model with built-in function

class SimpleNN(nn.Module):

def __init__(self):

super().__init__()

self.linear1 = nn.Linear(3, 4)

self.act1 = nn.ReLU()

self.linear2 = nn.Linear(4, 2)

def forward(self, x):

x = self.linear1(x)

x = self.act1(x)

x = self.linear2(x)

return x

model = SimpleNN()

for name, params in model.named_parameters():

print("Name:", name)

print("Param", params)

#Replacing MSE with built-in function

mse = torch.nn.MSELoss()

mae = torch.nn.L1Loss()

Training step

0s completed at 3:45 PM

AISCIENCES

Udemy



Δ SCIENCES



Δ SCIENCES

1

...

```
mae = torch.nn.L1Loss()
```

```
def fit()
```

```
for i in range(1000):
```

```
preds = model(inputs)
```

```
loss = mae(preds, targets)
```

```
loss.backward()
```

```
if i%100 == 99:
```

```
print(loss.item())
```

```
with torch.no_grad():
```

```
w -= w.grad * 0.00001
```

```
b -= b.grad * 0.00001
```

```
w.grad.zero_()
```

```
b.grad.zero_()
```

```
preds = model(inputs)
```

```
loss = mae(preds, targets)
```

```
print(loss)
```

→

Name: linear1.weight

Param Parameter containing:

```
tensor([[ -0.1478, -0.5037, -0.5325],
```

 $[0.3457, 0.3467, -0.5086],$
$$[-0.1229, -0.5641, -0.5510],$$

1 0 0000 0 0000 0 000000

✓ 0s completed at 3:45 PM

Pytorch.ipynb - Colab

Optimizer.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

★

Paused

+ Code

+ Text

All changes saved

▶

```
#Replacing MSE with built-in function
# mse = torch.nn.MSELoss()
mae = torch.nn.L1Loss()

#Optim

# Training step
def fit(num_epochs, model, loss_fun, opt):
    for i in range(1000):
        preds = model(inputs)
        loss = mae(preds, targets)
        loss.backward()
        if i%100 == 99:
            print(loss.item())

        with torch.no_grad():
            w -= w.grad * 0.00001
            b -= b.grad * 0.00001
            w.grad.zero_()
            b.grad.zero_()

    preds = model(inputs)
    loss = mae(preds, targets)
    print(loss)
```

↗

Name: linear1.weight
Param Parameter containing:
tensor([[-0.1478, -0.5037, -0.5325],

✓ 0s

completed at 3:45 PM

AI SCIENCES

Udacity

Pytorch.ipynb - Colab

Optimizer.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

RAM

Disk

Paused

+ Code

+ Text

All changes saved

def forward(self, x):
 x = self.linear1(x)
 x = self.act1(x)
 x = self.linear2(x)
 return x

model = SimpleNN()
for name, params in model.named_parameters():
print("Name:", name)
print("Param", params)

#Replacing MSE with built-in function
mse = torch.nn.MSELoss()
mae = torch.nn.L1Loss()

#Optimizer
opt = torch.optim.Adam(model.parameters(), lr = 0.00001)

Training step
def fit(num_epochs, model, loss_fun, opt):
 for i in range(1000):
 preds = model(inputs)
 loss = mae(preds, targets)
 loss.backward()
 if i%100 == 99:
 print(loss.item())

 with torch.no_grad():

0s

completed at 3:45 PM

AISCIENCES

Udemy

Pytorch.ipynb - Colab

Optimizer.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

☆

Paused

RAM

Disk

↑

↓

↻

💬

⚙️

📄

🗑️

⋮

+ Code

+ Text

▶

```
# print("Param", params)

#Replacing MSE with built-in function
# mse = torch.nn.MSELoss()
mae = torch.nn.L1Loss()

#Optimizer
opt = torch.optim.Adam(model.parameters(), lr = 0.00001)

# Training step
def fit(num_epochs, model, loss_fun, opt):
    for i in range(1000):
        preds = model(inputs)
        loss = mae(preds, targets)
        loss.backward()
        if i%100 == 99:
            print(loss.item())

            with torch.no_grad():
                w -= w.grad * 0.00001
                b -= b.grad * 0.00001
                w.grad.zero_()
                b.grad.zero_()

        preds = model(inputs)
        loss = mae(preds, targets)
        print(loss)
```

🔍

{x}

🔑

📁

<>

☰

🖨️

🔗

Name: linear1.weight

✅ 0s

completed at 3:45 PM

Udemy

AI SCIENCES

Pytorch.ipynb - Colab

Optimizer.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

RAM

Disk

Paused

+ Code + Text

print("Param", params)

#Replacing MSE with built-in function

mse = torch.nn.MSELoss()

mae = torch.nn.L1Loss()

#Optimizer

opt = torch.optim.Adam(model.parameters(), lr = 0.00001)

Training step

def fit(num_epochs, model, loss_fun, opt):

for epoch in range(num_epochs):

preds = model(inputs)

loss = mae(preds, targets)

loss.backward()

if i%100 == 99:

print(loss.item())

with torch.no_grad():

w -= w.grad * 0.00001

b -= b.grad * 0.00001

w.grad.zero_()

b.grad.zero_()

preds = model(inputs)

loss = mae(preds, targets)

print(loss)

Name: linear1.weight

0s

completed at 3:45 PM

AI SCIENCES

Udemy

Pytorch.ipynb - Colab

Optimizer.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

☆
RAM
Disk

Paused

+ Code + Text All changes saved

print("Param", params)

#Replacing MSE with built-in function

mse = torch.nn.MSELoss()

mae = torch.nn.L1Loss()

#Optimizer

opt = torch.optim.Adam(model.parameters(), lr = 0.00001)

Training step

def fit(num_epochs, model, loss_fun, opt):

for epoch in range(num_epochs):

preds = model(inputs)

loss = loss_fun(preds, targets)

loss.backward()

if i%100 == 99:

print(loss.item())

with torch.no_grad():

w -= w.grad * 0.00001

b -= b.grad * 0.00001

w.grad.zero_()

b.grad.zero_()

preds = model(inputs)

loss = mae(preds, targets)

print(loss)

Name: linear1.weight

0s completed at 3:45 PM

ASCIENCES

Udemy

Pytorch.ipynb - Colab

Optimizer.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

RAM

Disk

Paused

↑

↓

↻

💬

⚙️

📄

🗑️

⋮

+ Code

+ Text

All changes saved

▶

```
# print("Param", params)

#Replacing MSE with built-in function
# mse = torch.nn.MSELoss()
mae = torch.nn.L1Loss()

#Optimizer
opt = torch.optim.Adam(model.parameters(), lr = 0.00001)

# Training step
def fit(num_epochs, model, loss_fn, opt):
    for epoch in range(num_epochs):
        preds = model(inputs)
        loss = loss_fn(preds, targets)

        loss.backward()
        if i%100 == 99:
            print(loss.item())

        opt.step()
        with torch.no_grad():
            w -= w.grad * 0.00001
            b -= b.grad * 0.00001
            w.grad.zero_()
            b.grad.zero_()

    preds = model(inputs)
    loss = mae(preds, targets)
```

ASCIENCES

0s completed at 3:45 PM

Udemy



```
loss = mae(preds, targets)
```

Pytorch.ipynb - Colab

Optimizer.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

+ Code + Text All changes saved

print("Param", params)

#Replacing MSE with built-in function

mse = torch.nn.MSELoss()

mae = torch.nn.L1Loss()

#Optimizer

opt = torch.optim.Adam(model.parameters(), lr = 0.00001)

Training step

def fit(num_epochs, model, loss_fn, opt):

for epoch in range(num_epochs):

preds = model(inputs)

loss = loss_fn(preds, targets)

loss.backward()

if i%100 == 99:

print(loss.item())

opt.step()

opt.zero_grad()

preds = model(inputs)

loss = mae(preds, targets)

print(loss)

Name: linear1.weight

Param Parameter containing:

tensor([[-0.1478, -0.5037, -0.5325],

[0.3457, 0.3457, 0.5086]

RAM

Disk

Paused

↑ ↓ ↻ ⌨ ⚙ 📄 🗑 ⋮

<>

0s completed at 3:45 PM

ASCIENCES

Udemy

Pytorch.ipynb - Colab

Optimizer.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

RAM

Disk

Paused

↑

↓

↻

💬

⚙️

📄

🗑️

⋮

+ Code

+ Text

▶

```
# print("Param", params)

#Replacing MSE with built-in function
# mse = torch.nn.MSELoss()
mae = torch.nn.L1Loss()

#Optimizer
opt = torch.optim.Adam(model.parameters(), lr = 0.00001)

# Training step
def fit(num_epochs, model, loss_fn, opt):
    for epoch in range(num_epochs):
        preds = model(inputs)
        loss = loss_fn(preds, targets)

        loss.backward()
        if i%100 == 99:
            print(loss.item())

        opt.step()
        opt.zero_grad()

    ⚡
    fit(1000, model, mae, opt)
    preds = model(inputs)
    loss = mae(preds, targets)
    print(loss)
```

Name: linear1.weight

0s

completed at 3:45 PM

Udemy

Pytorch.ipynb - Colab

Optimizer.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

RAM

Disk

Paused

↑

↓

↻

💬

⚙️

📄

🗑️

⋮

+ Code + Text

▶

```
# print("Param", params)

#Replacing MSE with built-in function
# mse = torch.nn.MSELoss()
mae = torch.nn.L1Loss()

#Optimizer
opt = torch.optim.Adam(model.parameters(), lr = 0.00001)

# Training step
def fit(num_epochs, model, loss_fn, opt):
    for epoch in range(num_epochs):
        preds = model(inputs)
        loss = loss_fn(preds, targets)

        loss.backward()
        if i%100 == 99:
            print(loss.item())

        opt.step()
        opt.zero_grad()

fit(1000, model, mae, opt)
preds = model(inputs)
print(pred)
```

...

AI SCIENCES

✓ 0s

completed at 3:45 PM

Udemy

Pytorch.ipynb - Colab

Optimizer.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

☆

Paused

+ Code

+ Text

RAM

Disk

↑

↓

↶

💬

⚙️

📄

🗑️

⋮

⋮

🔍

{x}

🔑

📁

```
mae = torch.nn.L1Loss()

#Optimizer
opt = torch.optim.Adam(model.parameters(), lr = 0.00001)

# Training step
def fit(num_epochs, model, loss_fn, opt):
    for epoch in range(num_epochs):
        preds = model(inputs)
        loss = loss_fn(preds, targets)

        loss.backward()
        if epoch%100 == 99:
            print(loss.item())

        opt.step()
        opt.zero_grad()

fit(1000, model, mae, opt)
preds = model(inputs)
print(pred)
```

↔

NameError

Traceback (most recent call last)

<ipython-input-5-89d353caa3c5> in <cell line: 42>()

40

41

----> 42 fit(1000, model, mae, opt)

43 preds = model(inputs)

44 print(pred)

2s

completed at 4:15 PM

AISCSCIENCES

Udemy

Pytorch.ipynb - Colab

Optimizer.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

+ Code + Text

1s

```
preds = model(inputs)
loss = loss_fn(preds, targets)

loss.backward()
if epoch%100 == 99:
    print(loss.item())

opt.step()
opt.zero_grad()

fit(1000, model, mae, opt)
preds = model(inputs)
print(pred)
```

↔

77.4625473022461
77.33414459228516
77.20309448242188
77.06471252441406
76.9249496459961
76.78411865234375
76.6419677734375
76.49844360351562
76.35286712646484
76.2035903930664

NameError Traceback (most recent call last)
<ipython-input-6-6f3b3379881e> in <cell line: 44>()
 42 fit(1000, model, mae, opt)
 43 preds = model(inputs)
--> 44 print(pred)

0s

completed at 4:15 PM

ASCIENCES

Udemy

Pytorch.ipynb - Colab

Optimizer.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

RAM

Disk

Paused

+ Code + Text

training step

def fit(num_epochs, model, loss_fn, opt):

for epoch in range(num_epochs):

preds = model(inputs)

loss = loss_fn(preds, targets)

loss.backward()

if epoch%100 == 99:

print(loss.item())

opt.step()

opt.zero_grad()

fit(1000, model, mae, opt)

preds = model(inputs)

print(preds)

[0] preds

Tensor

77.46254730

77.33414459

77.20309448

77.06471252

76.92494964

76.78411865

76.64196777

76.49844360

76.35286712

76.20359039

NameError

Traceback (most recent call last)

<ipython-input-6-6f3b3379881e> in <cell line: 44>()

42 fit(1000, model, mae, opt)

0s completed at 4:15 PM

DISSCIENCES

Udemy

Pytorch.ipynb - Colab

Optimizer.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

RAM

Disk

Paused

+ Code + Text Saving...

1s

```
# training step
def fit(num_epochs, model, loss_fn, opt):
    for epoch in range(num_epochs):
        preds = model(inputs)
        loss = loss_fn(preds, targets)

        loss.backward()
        if epoch%100 == 99:
            print(loss.item())

        opt.step()
        opt.zero_grad()

fit(1000, model, mae, opt)
preds = model(inputs)
print(preds)
```

88.49784088134766

88.44050593144531

88.37797546386719

88.31217956542969

88.24705505371094

88.18204498291016

88.1168441772461

88.05128479003906

87.9852523803711

87.91487884521484

tensor([[0.9533, 0.3031],

[2.1716, 0.5500],

[0.7623, 6.7681],

[-0.0032, 0.0011])

0s completed at 4:16 PM

ASCIENCES

Udemy

Pytorch.ipynb - Colab

Optimizer.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

RAM

Disk

Paused

+ Code + Text All changes saved

1s

```
model = SimpleNN()
# for name, params in model.named_parameters():
#     print("Name:", name)
#     print("Param", params)

#Replacing MSE with built-in function
# mse = torch.nn.MSELoss()
mae = torch.nn.L1Loss()

#Optimizer
opt = torch.optim.Adam(model.parameters(), lr = 0.00001)

# Training step
def fit(num_epochs, model, loss_fn, opt):
    for epoch in range(num_epochs):
        preds = model(inputs)
        loss = loss_fn(preds, targets)

        loss.backward()
        if epoch%100 == 99:
            print(loss.item())

    opt.step()
    opt.zero_grad()

fit(1000, model, mae, opt)
preds = model(inputs)
print(preds)
```

ASCIENCES

0s completed at 4:16 PM

Pytorch.ipynb - Colab

Optimizer.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

RAM

Disk

Paused

+ Code + Text All changes saved

0s

```
targets = np.array([[56, 70],
                    [81, 101],
                    [119, 133],
                    [22, 37],
                    [103, 119],
                    [98, 110],
                    [88, 95],
                    [115, 140],
                    [76, 85],
                    [65, 75]], dtype='float32')

inputs = torch.from_numpy(inputs)
targets = torch.from_numpy(targets)
print(inputs)
print(targets)
```

tensor([[73., 67., 43.],
 [91., 88., 64.],
 [87., 134., 58.],
 [102., 43., 37.],
 [69., 96., 70.],
 [85., 100., 60.],
 [95., 80., 55.],
 [105., 120., 75.],
 [78., 90., 50.],
 [82., 70., 45.]])

tensor([[56., 70.],
 [81., 101.],
 [119., 133.],
 [22., 37.],
 [103., 119.],
 [98., 110.],
 [88., 95.]])

0s

completed at 4:16 PM

AI SCIENCES

Udemy

Pytorch.ipynb - Colab

Optimizer.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

+ Code

+ Text

All changes saved

0s

▶

[103, 119],

[98, 110],

[88, 95],

[115, 140],

[76, 85],

[65, 75]], dtype='float32')

inputs = torch.from_numpy(inputs)

targets = torch.from_numpy(targets)

print(inputs)

print(targets)

↩

tensor([[73., 67., 43.],

[91., 88., 64.],

[87., 134., 58.],

[102., 43., 37.],

[69., 96., 70.],

[85., 100., 60.],

[95., 80., 55.],

[105., 120., 75.],

[78., 90., 50.],

[82., 70., 45.]])

tensor([[56., 70.],

[81., 101.],

[119., 133.],

[22., 37.],

[103., 119.],

[98., 110.],

[88., 95.],

[115., 140.],

[76., 85.],

[65., 75.]])

RAM

Disk

Paused

ASCIENCES

0s

completed at 4:16 PM

UdenX

Pytorch.ipynb - Colab

Optimizer.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

☆

Paused

✓ RAM

Disk

↑

↓

↻

💬

⚙️

📄

🗑️

⋮

+ Code

+ Text

All changes saved

✓ 1s

▶

```
# print("Param", params)

#Replacing MSE with built-in function
# mse = torch.nn.MSELoss()
mae = torch.nn.L1Loss()

#Optimizer
opt = torch.optim.Adam(model.parameters(), lr = 0.00001)

# Training step
def fit(num_epochs, model, loss_fn, opt):
    for epoch in range(num_epochs):
        preds = model(inputs)
        loss = loss_fn(preds, targets)

        loss.backward()
        if epoch%100 == 99:
            print(loss.item())

        opt.step()
        opt.zero_grad()

fit(1000, model, mae, opt)
preds = model(inputs)
print(preds)
```

88.49784088134766

88.44050598144531

88.49784088134766

88.44050598144531

✓ 0s

completed at 4:16 PM

AI SCIENCES

Udemy

Pytorch.ipynb - Colab

Optimizer.pdf

colab.research.google.com/drive/1imQs4frEn2sYP0VW7SjxgJwC5ZgNZdpv#scrollTo=BZYFUI5XqzSc

☆

Paused

✓ RAM

Disk

+ Code

+ Text

All changes saved

✓ 1s

▶

print("Param", params)

#Replacing MSE with built-in function

mse = torch.nn.MSELoss()

mae = torch.nn.L1Loss()

#Optimizer

opt = torch.optim.Adam(model.parameters(), lr = 0.00001)

Training step

def fit(num_epochs, model, loss_fn, opt):

for epoch in range(num_epochs):

preds = model(inputs)

loss = loss_fn(preds, targets)

loss.backward()

if epoch%100 == 99:

print(loss.item())

opt.step()

opt.zero_grad()

fit(1000, model, mae, opt)

preds = model(inputs)

print(preds)

88.49784088134766

88.44050598144531

ASCIENCES

0s completed at 4:16 PM

Udemy