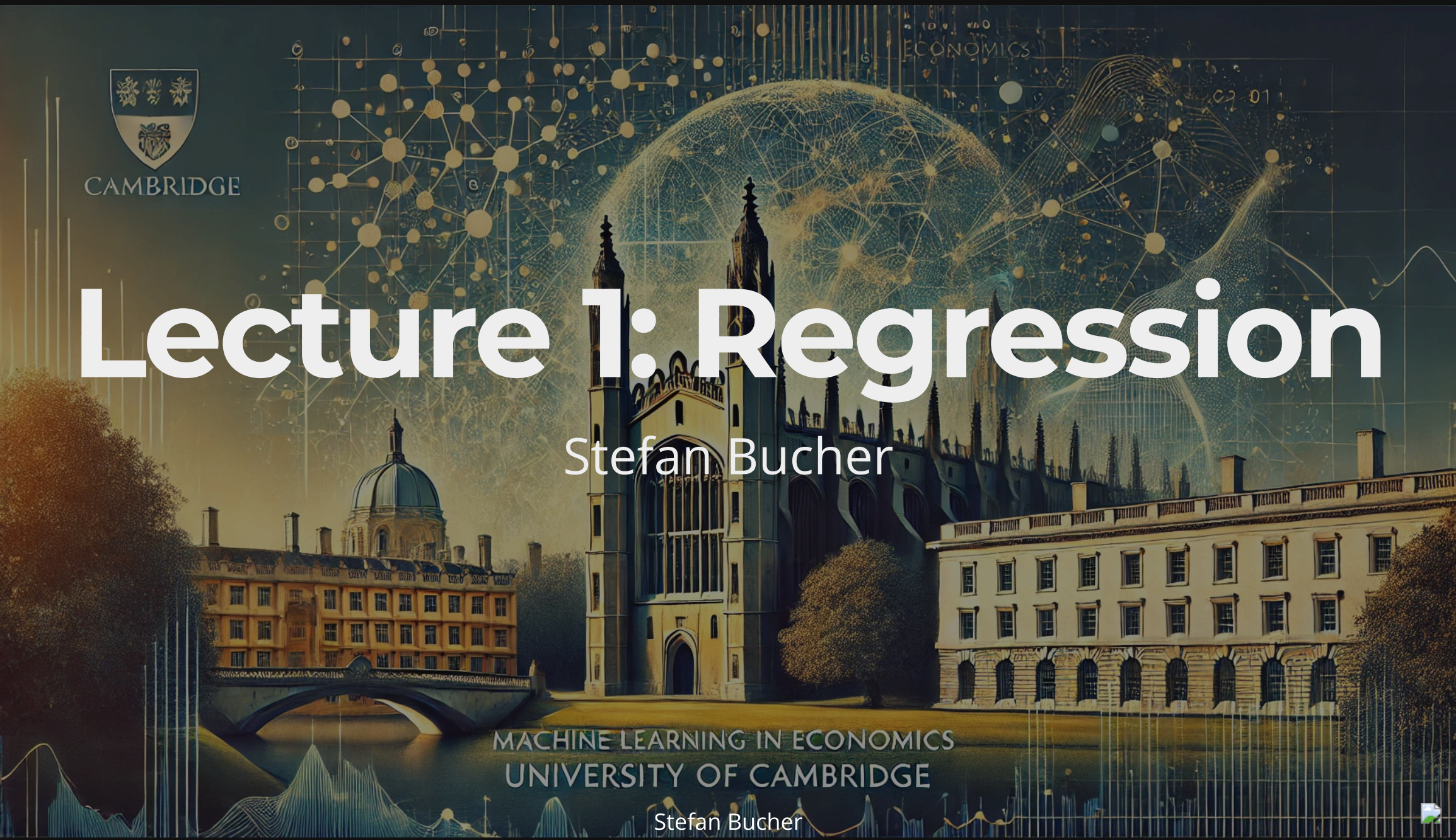


Lecture 1: Regression

Stefan Bucher

Open in Colab

Prince (2023, chaps. 2, 6, 8, 9).[1]

1. Figures taken or adapted from Prince (2023). All rights belong to the original author and publisher. These materials are intended solely for educational purposes.

Linear Regression

Prince (2023, chap. 2)

Linear Model

$$\mathbf{y} = \mathbf{X}\beta + \varepsilon$$

Minimizing the mean-squared error (MSE)

$$\min_{\beta} \frac{1}{n}(y - X\beta)^T(y - X\beta)$$

Minimizing MSE Loss (2.1-2.3)

Analytical Solution

$$\min_{\beta} \frac{1}{n}(y - X\beta)^T(y - X\beta)$$

Setting the gradient with respect to β to zero

$$0 = \nabla_{\beta}[y^T y - 2\beta^T X^T y + \beta^T X^T X\beta] = -2X^T y + 2X$$

yields the BLUE estimator (Gauss-Markov)

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

scikit-learn

scikit-learn

```
1 from sklearn.linear_model import LinearRegression
2 model = LinearRegression()
3 model.fit(X_train, y_train)
4 print(model.coef_[2], model.intercept_)
```

```
546.1898661888232 152.05949984245817
```

```
Text(0, 0.5, 'Diabetes Risk')
```


Model Fitting & Optimization

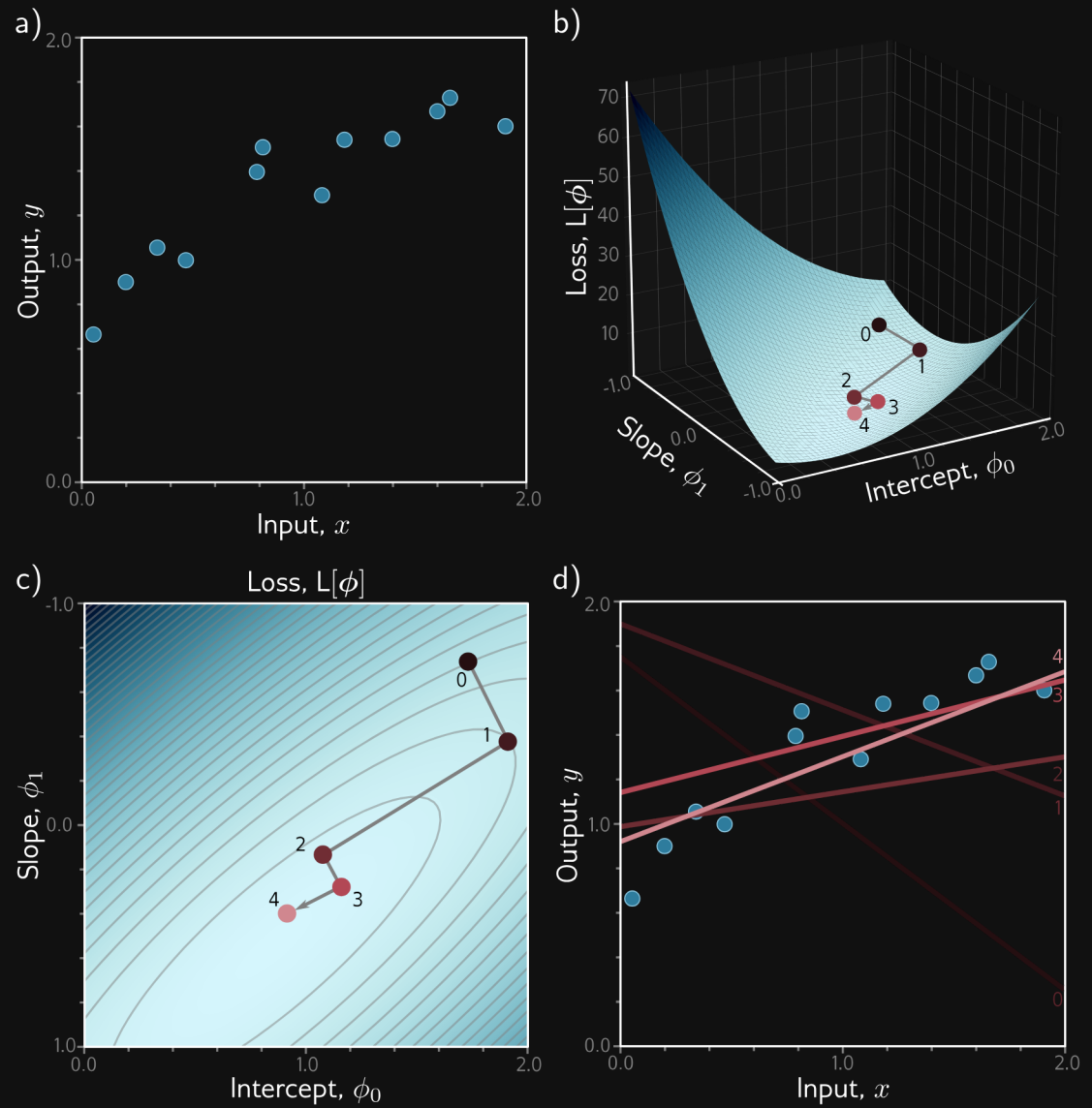
$$\hat{\phi} = \arg\min_{\phi} L(\phi)$$

Prince (2023, chap. 6)

Gradient Descent[1]

$$\phi \leftarrow \phi - \alpha \nabla_{\phi} L(\phi)$$

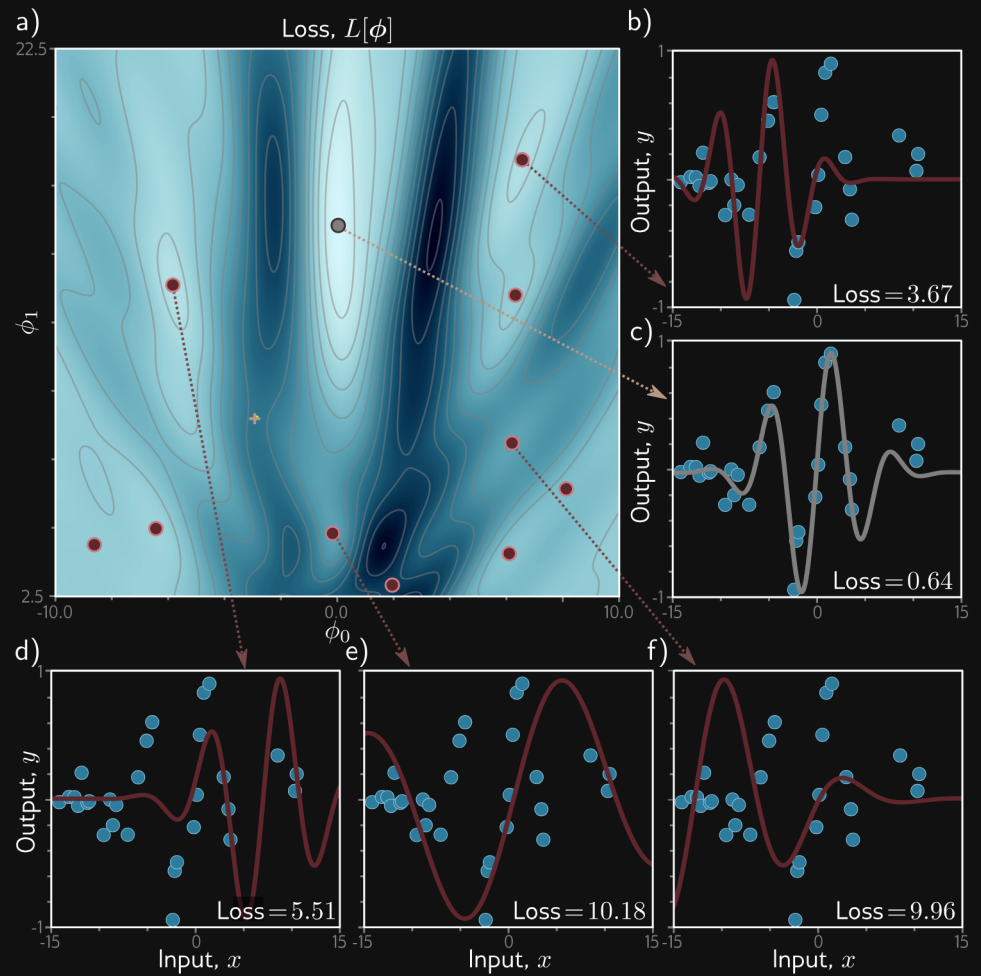
where α fixed learning rate or line search.



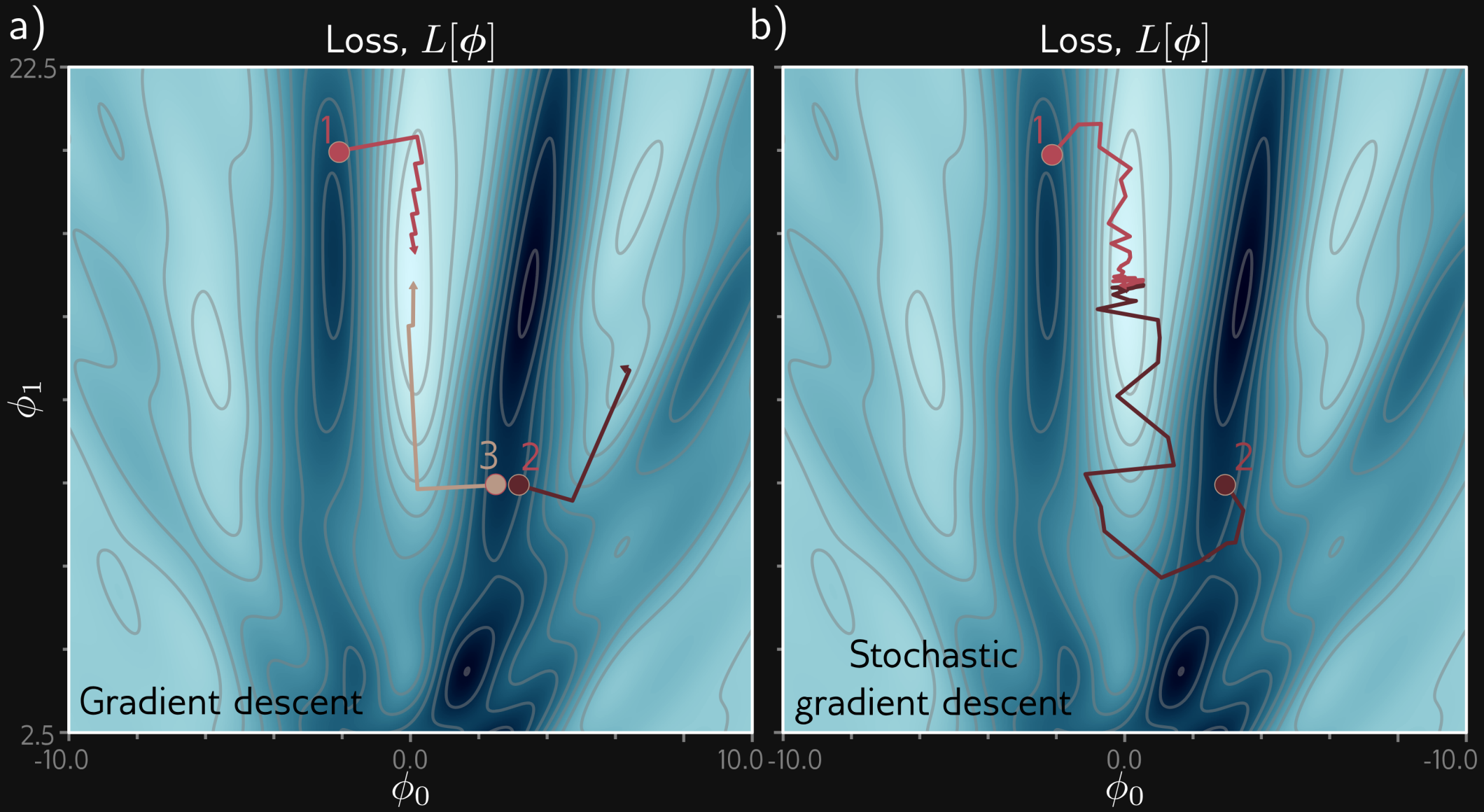
1. Cauchy (1847)

Nonlinear Model

Loss functions generally non-convex, possibly multiple local minima.



Stochastic Gradient Descent

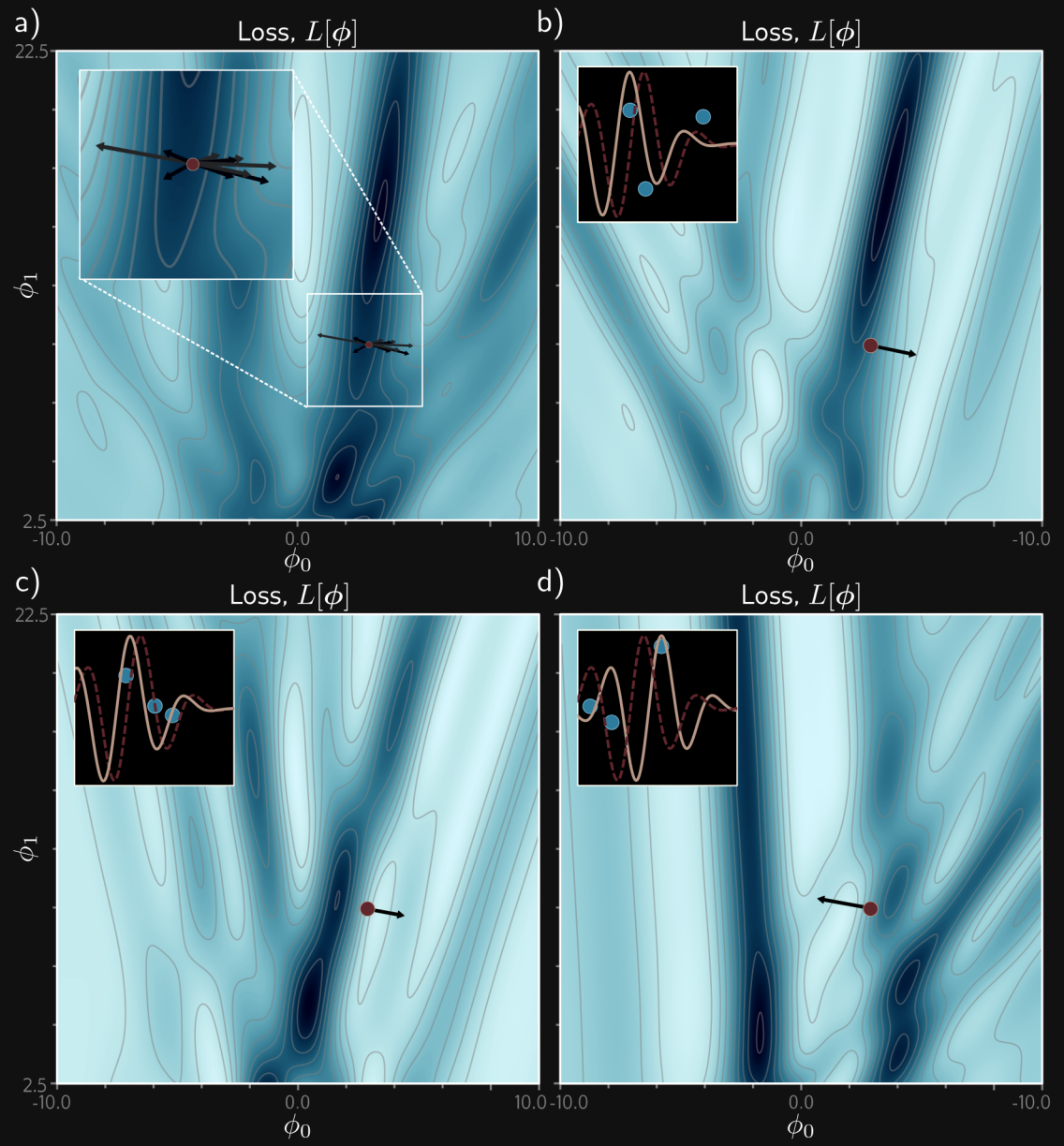


Stochastic Gradient Descent [1]

Compute gradient on a mini-batch B_t of data points

$$\phi_{t+1} \leftarrow \phi_t - \alpha \sum_{i \in B_t} \nabla_\phi l_i(\phi_t; x_i, y_i)$$

- Sampled without replacement within each epoch (sweep of training data).
- Learning rate often decreasing over time.



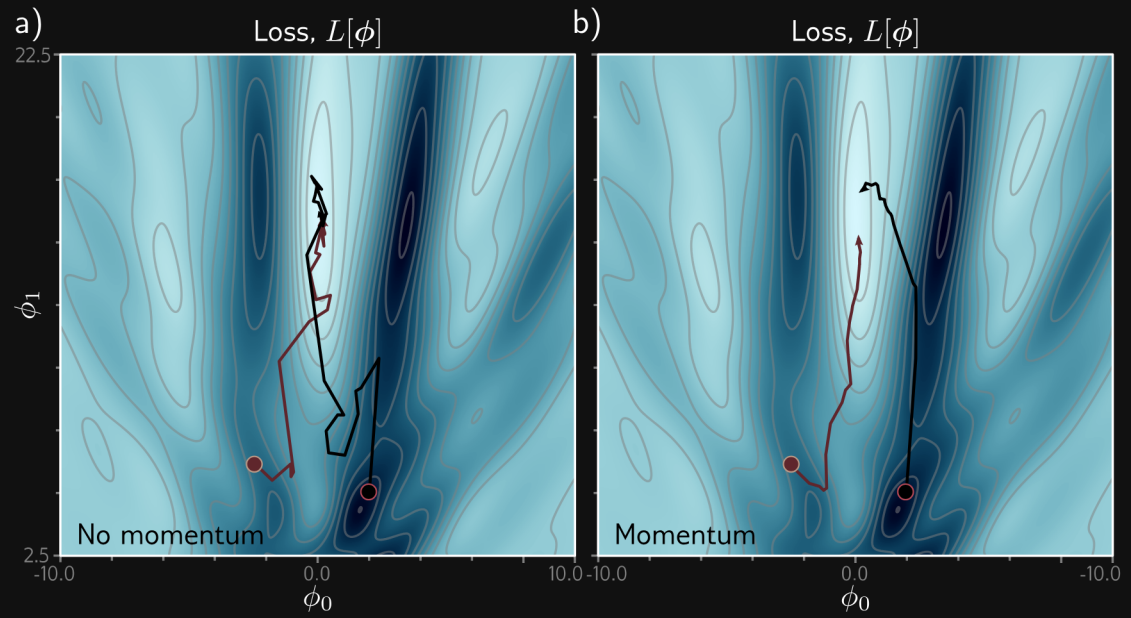
1. Robbins & Monro (1951)

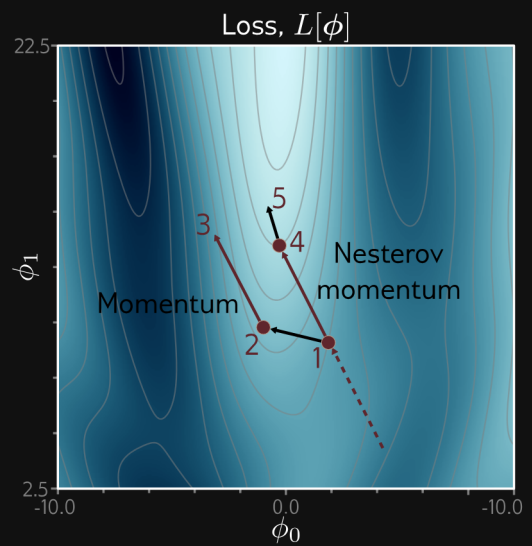
Nesterov Accelerated Momentum [1]

$$m_{t+1} \leftarrow \beta m_t + (1-\beta) \sum_{i \in B_t} \nabla_\phi l_i(\phi_t - \alpha\beta m_t; x_i, y_i)$$

$$\phi_{t+1} \leftarrow \phi_t - \alpha m_{t+1}$$

adds momentum (m_t) *before* evaluating gradient.





1. Nesterov (1983)

Stefan Bucher

Adaptive Moment Estimation (Adam) [1]

Normalizes the gradient while relying on momentum to avoid convergence issues.

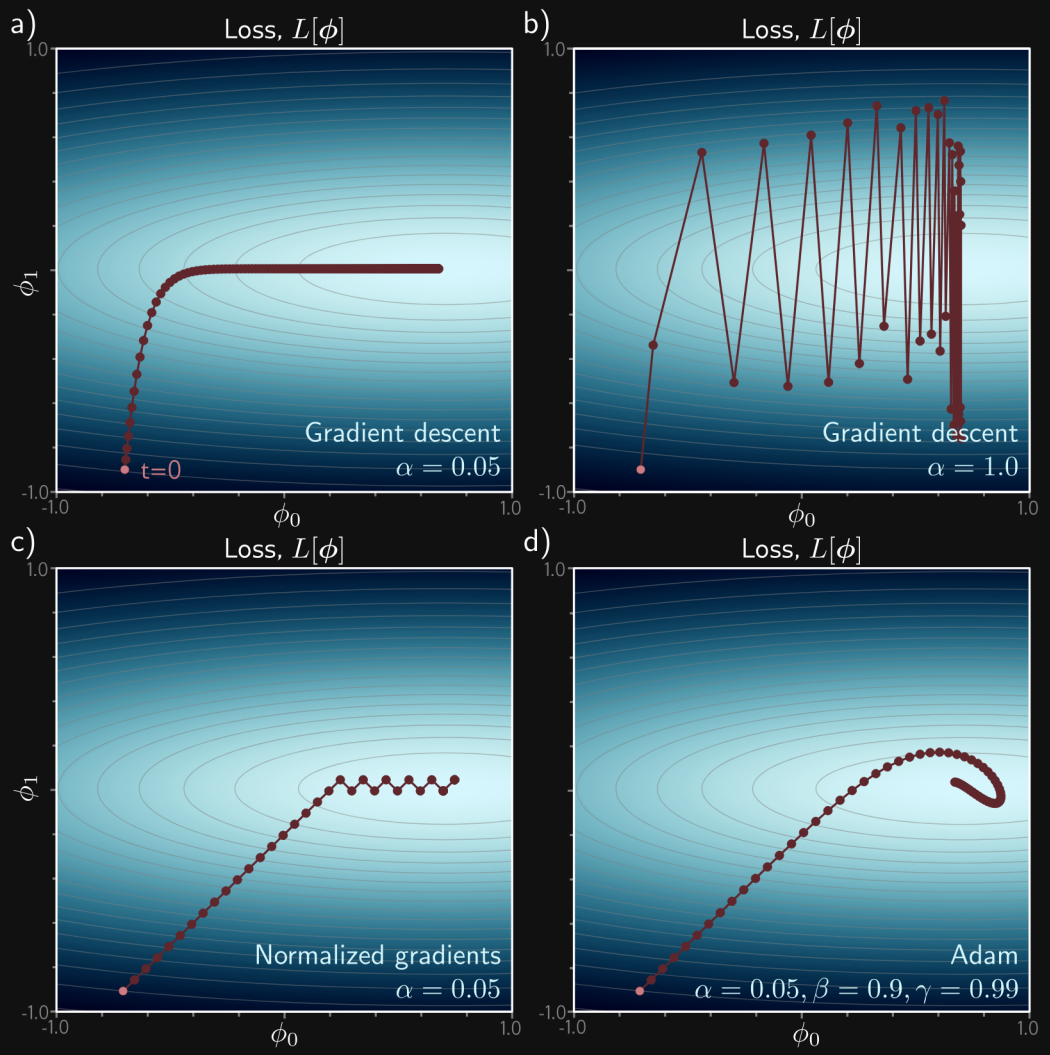
$$m_{t+1} \leftarrow \beta m_t + (1-\beta) \sum_{i \in B_t} \nabla_\phi l_i(\phi_t; x_i, y_i)$$

$$v_{t+1} \leftarrow \gamma v_t + (1-\gamma) \left(\sum_{i \in B_t} \nabla_\phi l_i(\phi_t; x_i, y_i) \right)^2$$

$$\phi_{t+1} \leftarrow \phi_t - \alpha \frac{\frac{m_{t+1}}{1-\beta^{t+1}}}{\sqrt{\frac{v_{t+1}}{1-\gamma^{t+1}}} + \epsilon}$$

1. Kingma & Ba (2015)

Adaptive Moment Estimation (Adam)



PyTorch

PyTorch

Setup

```python
import torch
from torch import nn
from torchvision import datasets
from torch.utils.data import DataLoader
from torchvision.transforms import ToTensor, Compose, Grayscale
import random

if torch.backends.mps.is_available():
  DEVICE = torch.device("mps") # Apple Silicon Metal Performance Shaders
elif torch.cuda.is_available():
  DEVICE = torch.device("cuda") # GPU
else:
  DEVICE = torch.device("cpu") # CPU
```

Key components:

- tensors are like numpy arrays but GPU-accelerated
- autograd

Tensors

Tensors “live” on a device

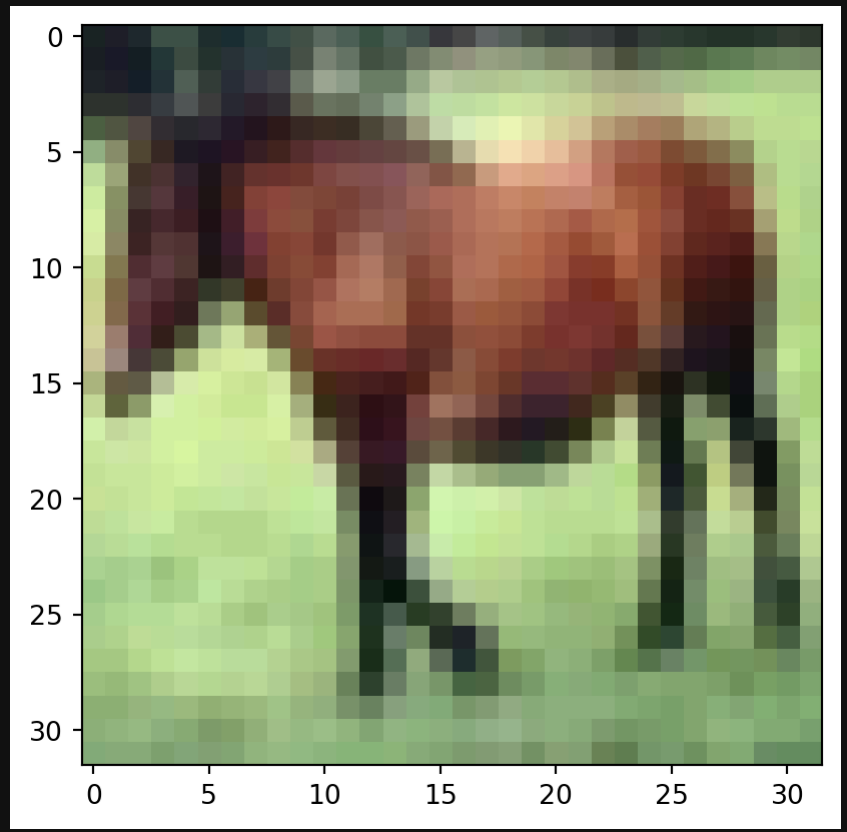
```
1 x = torch.randn(2, 2, device="cpu")
2 print(x.device)
3 x = x.to(DEVICE)
4 print(x.device)
```

```
cpu
mps:0
```


Datasets

```python
training_data = datasets.CIFAR10(
    root="data",  # path where the images will be stored
    train=True,
    download=True,
    transform=ToTensor()
    )
test_data = datasets.CIFAR10(
    root="data",
    train=False,
    download=True,
    transform=ToTensor()
    )
image, label = training_data[7]
print(f"Label: {training_data.classes[label]}")
print(f"Image size: {image.shape}")
plt.imshow(image.permute(1, 2, 0)) # imshow expects channel order Height x
plt.show()
```

```
Label: horse
Image size: torch.Size([3, 32, 32])
```


0
5
10
15
20
25
30
0
5
10
15
20
25
30

Data Loader

```
# Data Loader for Batching
def seed_worker(worker_id):
  worker_seed = torch.initial_seed() % 2**32
  numpy.random.seed(worker_seed)
  random.seed(worker_seed)
g_seed = torch.Generator()
g_seed.manual_seed(torch.initial_seed() % 2**32)

train_dataloader = DataLoader(training_data, batch_size=64, shuffle=True, r
test_dataloader = DataLoader(test_data,  batch_size=64, shuffle=True, num_w

# Load the next batch
batch_images, batch_labels = next(iter(train_dataloader))
print('Batch size:', batch_images.shape)
plt.imshow(batch_images[0].permute(1, 2, 0))
plt.show()
```


References

Prince, Simon J. D. 2023. *Understanding Deep Learning*. Cambridge, Massachusetts: The MIT Press.