Lecture 1: Regression

Stefan Bucher

Stefan Bucher

UNIVERSITY OF CAMBRIDGE

Open in Colab

Prince (2023, chaps. 2, 6, 8).[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

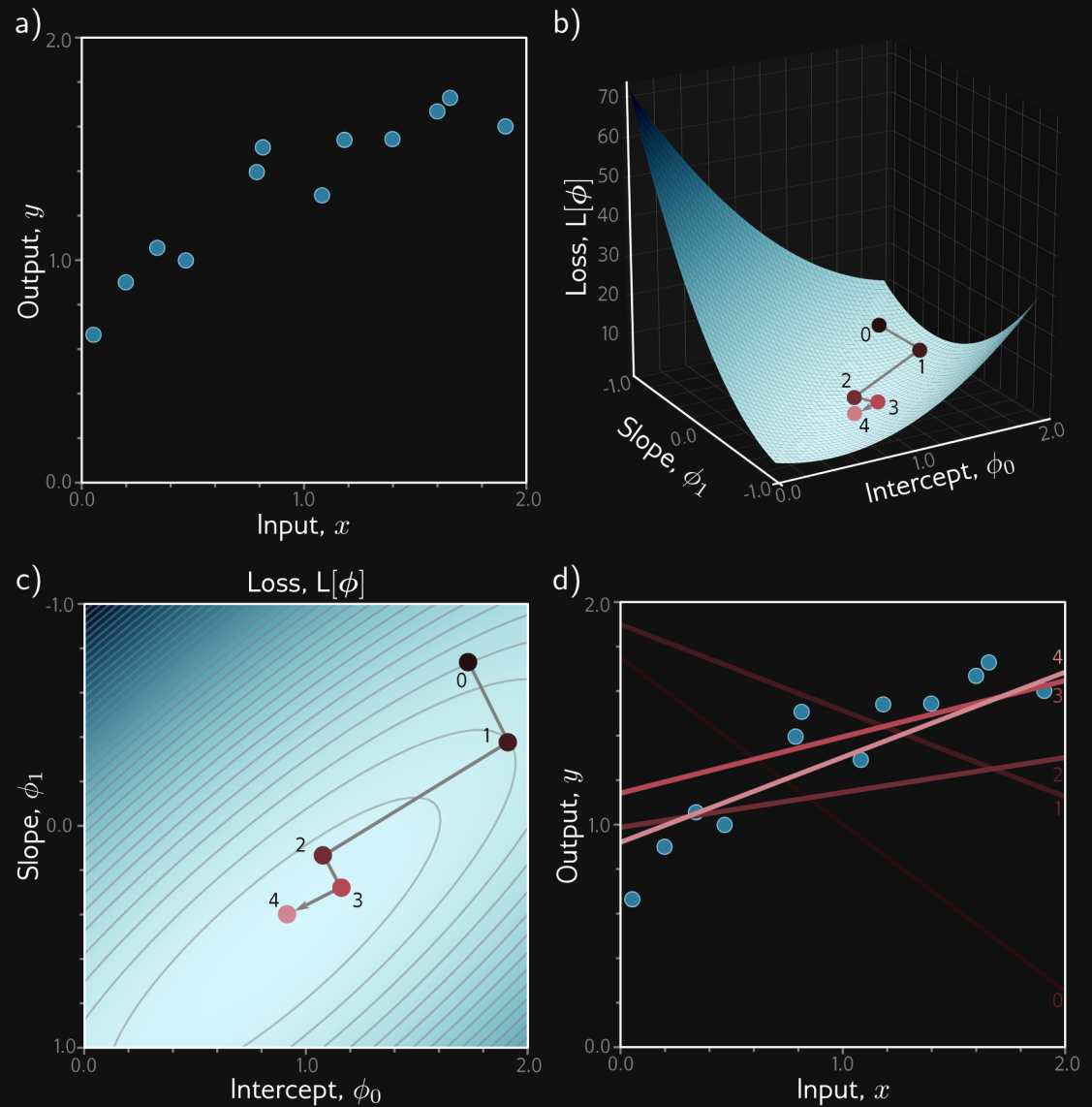
$$\hat{\phi} = \arg\min_{\phi} L(\phi)$$

Prince (2023, chap. 6)

Gradient Descent[1]

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

where α fixed learning rate or line search.

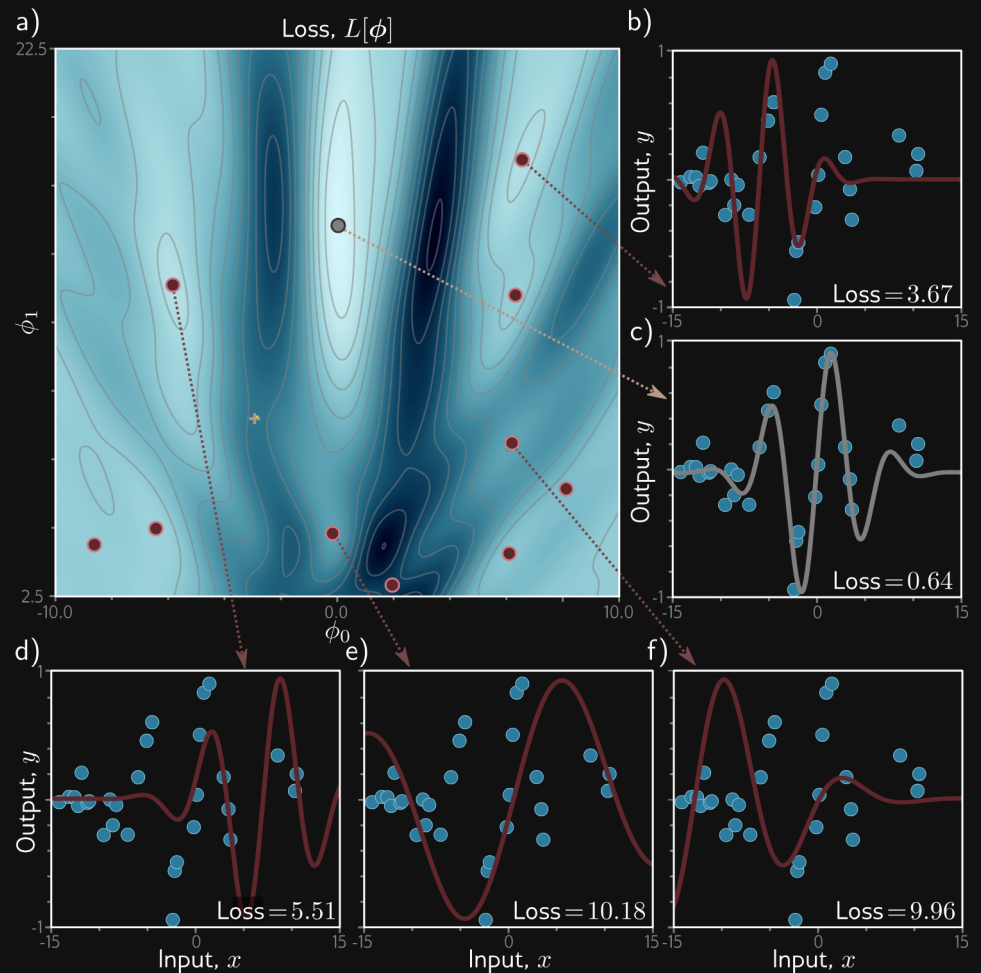


1. Cauchy (1847)

UNIVERSITY OF CAMBRIDGE

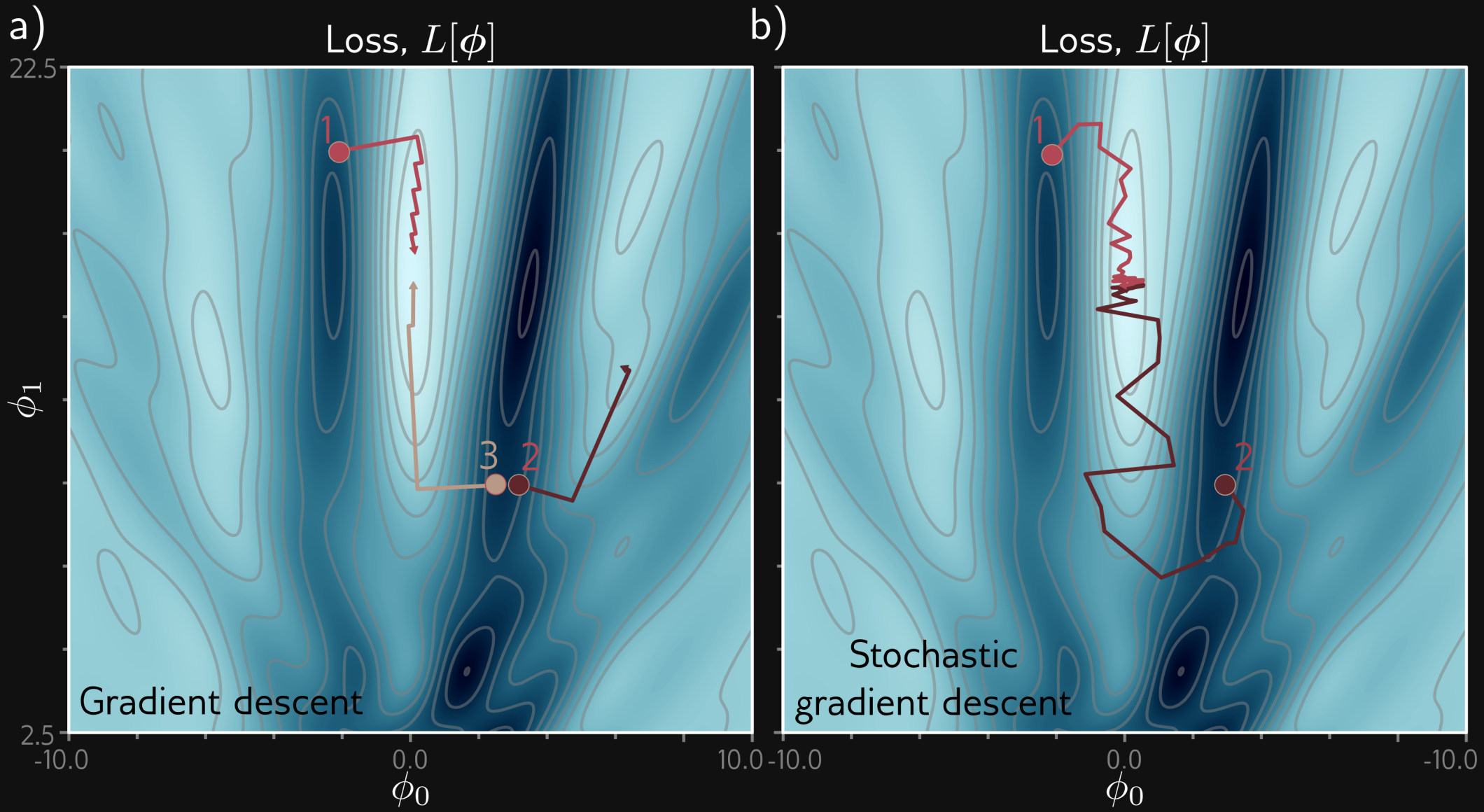
Nonlinear Model

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



UNIVERSITY OF CAMBRIDGE

Stochastic Gradient Descent

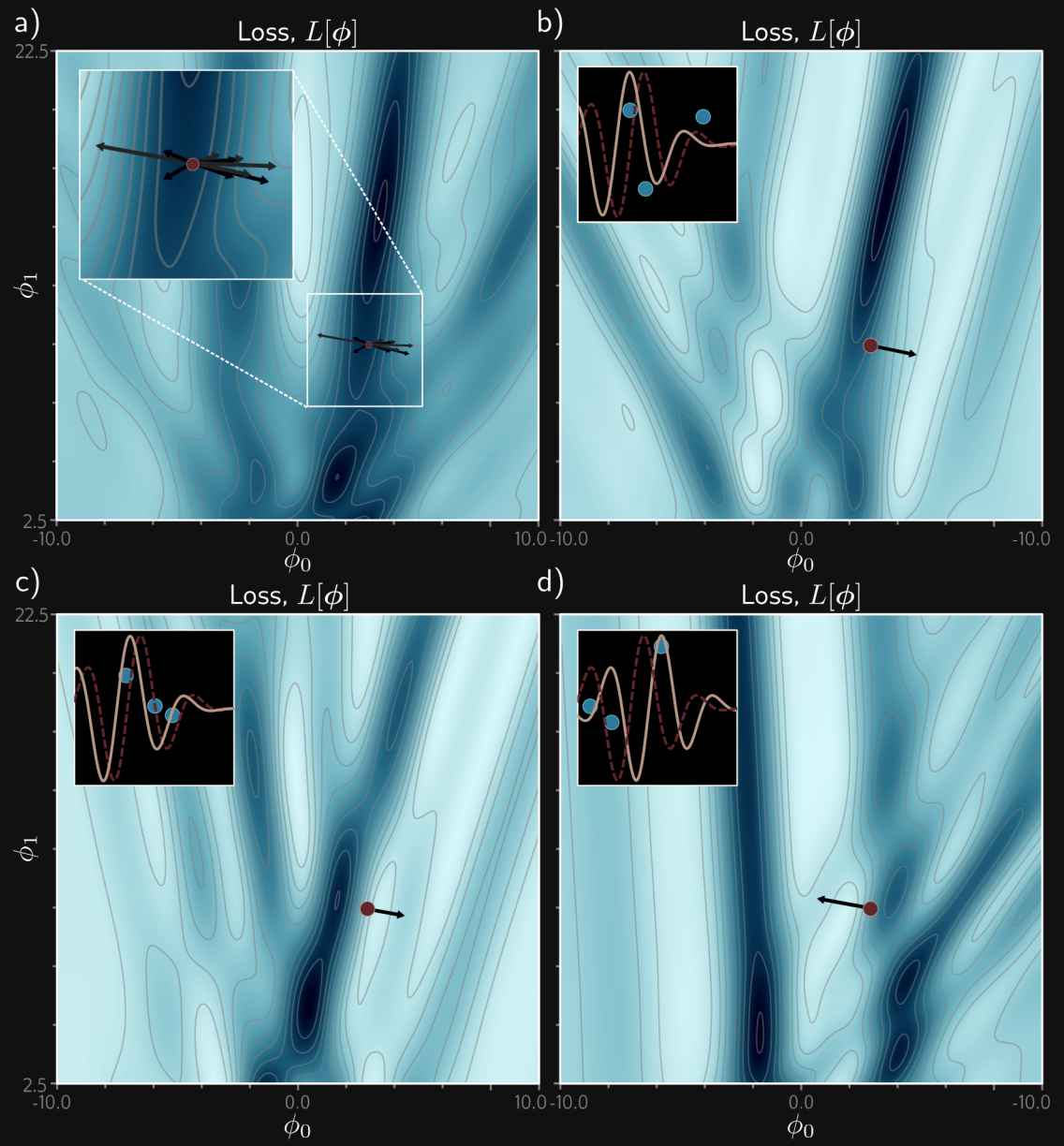


Stochastic Gradient Descent [1]

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.



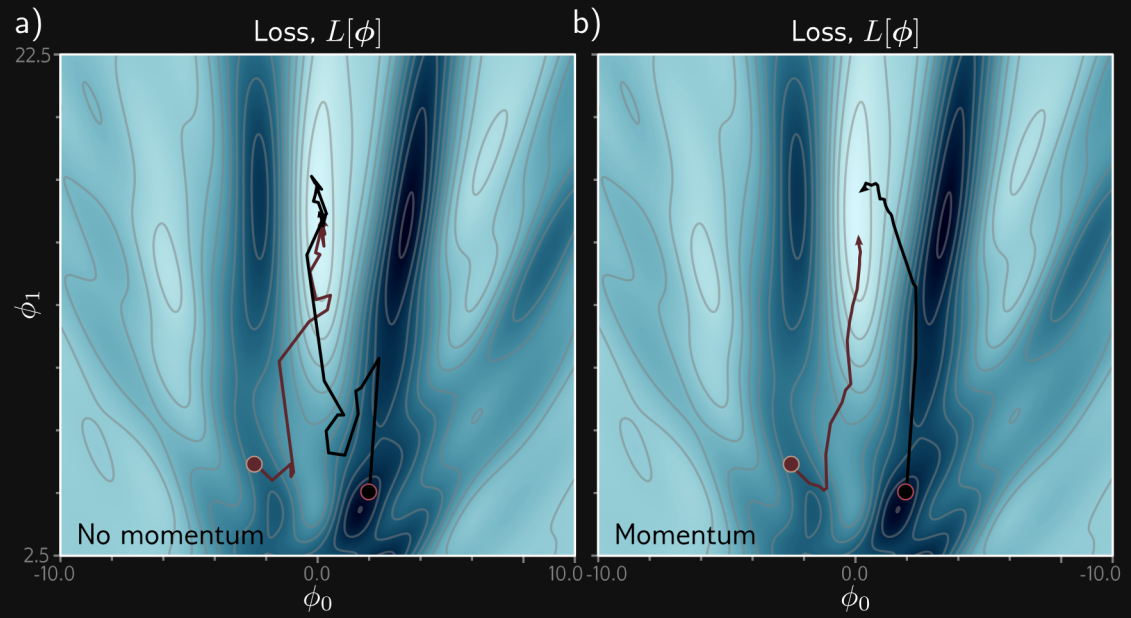
1. Robbins & Monro (1951)

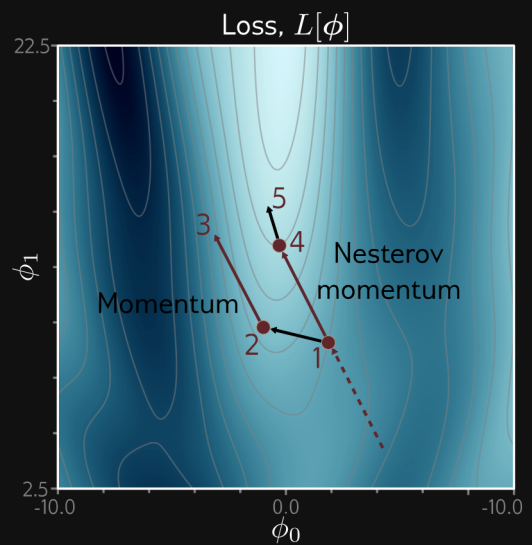
Nesterov Accelerated Momentum [1]

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

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

adds momentum (m_t) *before* evaluating gradient.





1. Nesterov (1983)

UNIVERSITY OF CAMBRIDGE

Adaptive Moment Estimation (Adam) [1]

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

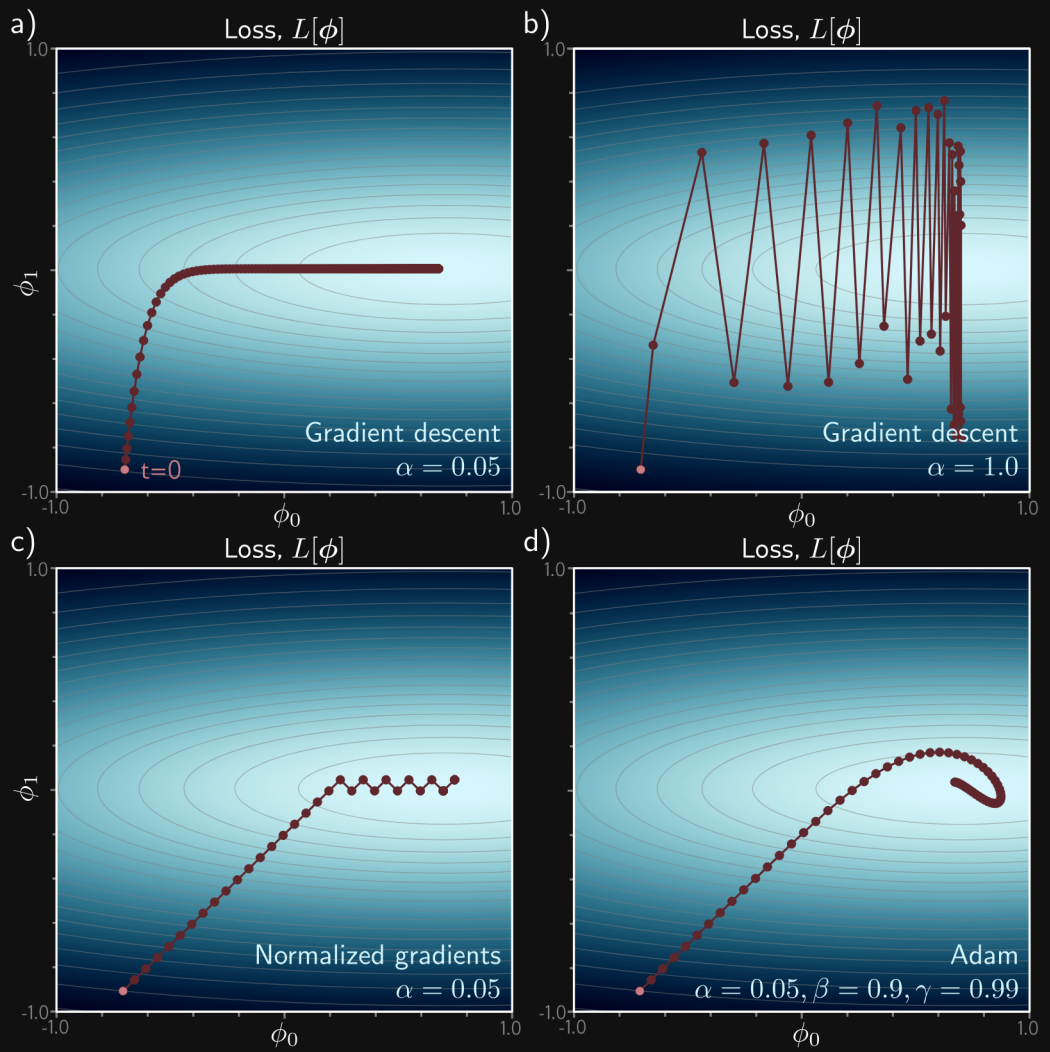
$$m_{t+1} \leftarrow \beta m_t + (1-\beta) \sum_{i \in B_t} \nabla_\phi l_i(\phi_t; x_i, y_i)$$

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

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

1. Kingma & Ba (2015)

Adaptive Moment Estimation (Adam)



PyTorch

PyTorch

Setup

```
 1 import torch
 2 from torch import nn
 3 import torch.optim as optim
 4 from torchvision import datasets
 5 from torch.utils.data import DataLoader
 6 from torchvision.transforms import ToTensor, Compose, Grayscale
 7 import random
 8
 9 if torch.backends.mps.is_available():
10   DEVICE = torch.device("mps") # Apple Silicon Metal Performance Shaders
11 elif torch.cuda.is_available():
12   DEVICE = torch.device("cuda") # GPU
13 else:
14   DEVICE = torch.device("cpu") # CPU
```

Key components:

Tensors

Tensors “live” on a device

```
1 X_train = torch.tensor(X.values, dtype=torch.float32).to(DEVICE)
2 y_train = torch.tensor(y.values, dtype=torch.float32).to("cpu")
3 print(X_train.device)
4 X_train =X_train.to("cpu")
5 print(X_train.device)
```

```
mps:0
cpu
```

The Previous Regression

```
1 solution = torch.linalg.lstsq(X_train, y_train)
2 coefficients = solution.solution[:-1].flatten()  # All coefficients except
3 intercept = solution.solution[-1].item()  # Intercept
4 print(coefficients[2], intercept)
```

```
tensor(519.8459) 152.13348388671875
```

The Previous Regression

```python
class LinearRegressionModel(nn.Module):
    def __init__(self, input_dim):
        super(LinearRegressionModel, self).__init__()
        self.linear = nn.Linear(input_dim, 1)

    def forward(self, x):
        return self.linear(x)

model = LinearRegressionModel(input_dim=X_train.shape[1])
criterion = nn.MSELoss()
# optimizer = optim.SGD(model.parameters(), lr=0.01)
optimizer = optim.Adam(model.parameters(), lr=0.01)

# Training loop
epochs = 15000
for epoch in range(epochs):
    # Forward pass
    outputs = model(X_train)
```

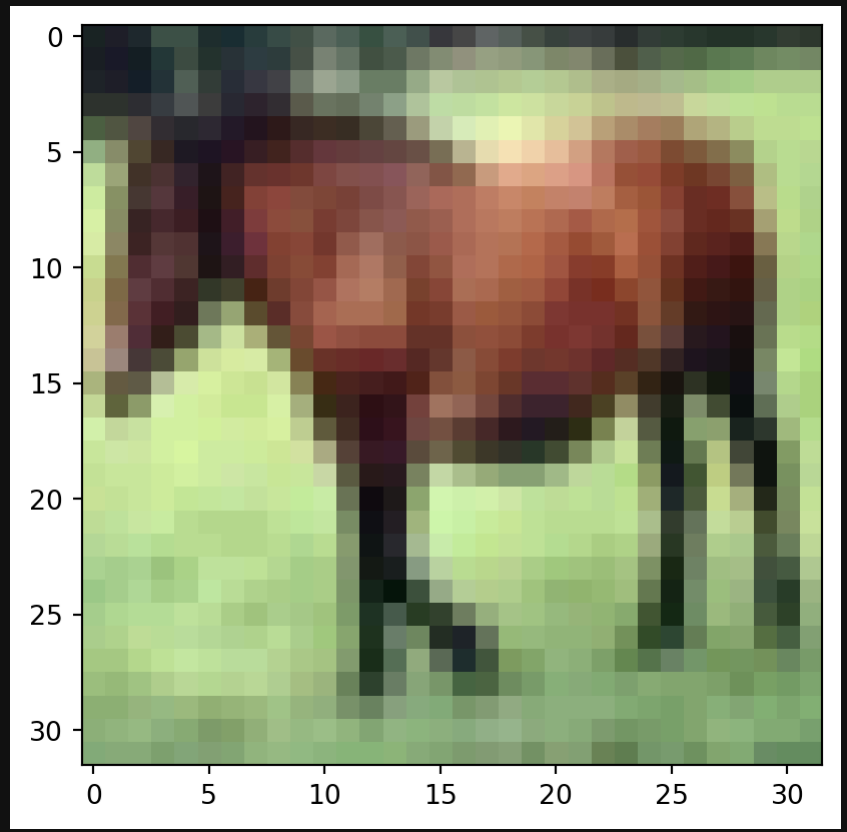
```
Epoch [1000/15000], Loss: 23613.3828
Epoch [2000/15000], Loss: 19056.0820
Epoch [3000/15000], Loss: 15332.7529
Epoch [4000/15000], Loss: 12342.8027
```

```
Epoch [5000/15000], Loss: 10015.9141
Epoch [6000/15000], Loss: 8294.0117
Epoch [7000/15000], Loss: 7116.4990
Epoch [8000/15000], Loss: 6407.3130
Epoch [9000/15000], Loss: 6063.0112
Epoch [10000/15000], Loss: 5949.3794
Epoch [11000/15000], Loss: 5930.7837
Epoch [12000/15000], Loss: 5929.8906
Epoch [13000/15000], Loss: 5929.8848
Epoch [14000/15000], Loss: 5929.8848
```

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

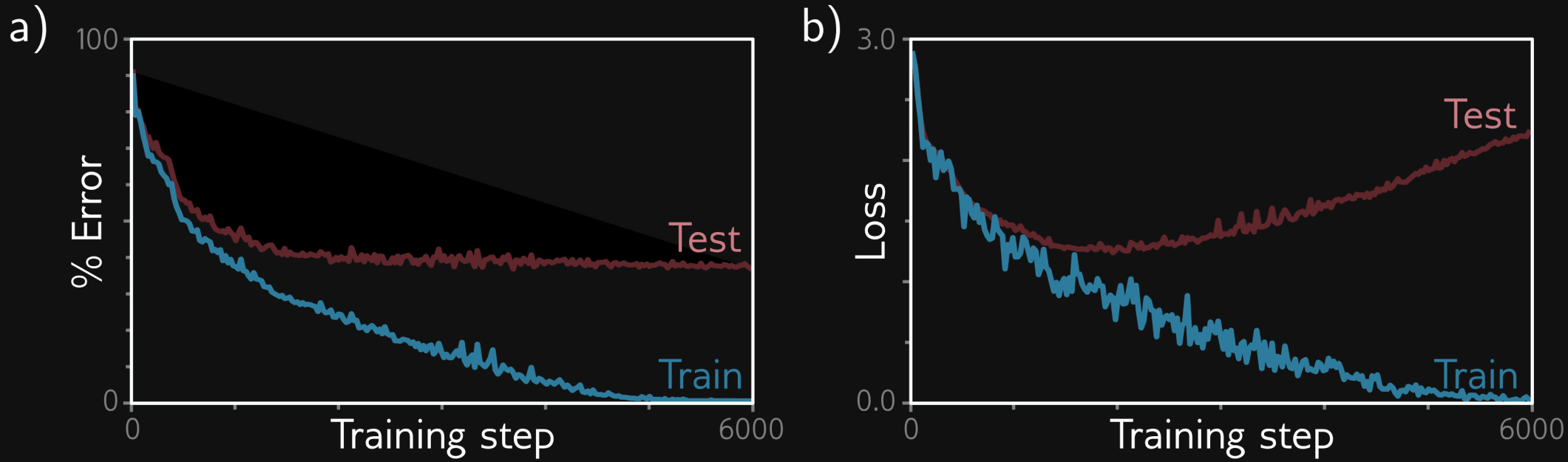
Data Loader

```
# 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()
```

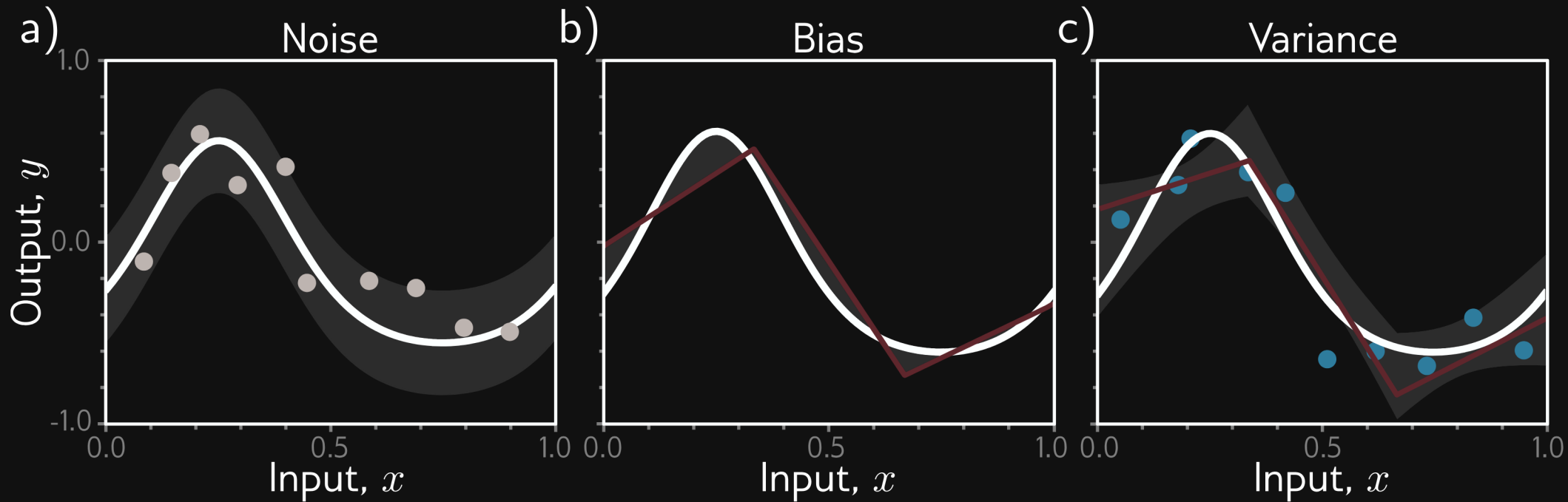
Measuring Performance

Prince (2023, chap. 8)

Training and Test Error



Error Sources



UNIVERSITY OF CAMBRIDGE

Error Decomposition

In a linear regression,

$$L[x] = (f[x, \phi] - \mathbb{E}[y|x] + \mathbb{E}[y|x] - y[x])^2$$
$$= (f[x, \phi] - \mathbb{E}[y|x])^2 + 2(f[x, \phi] - \mathbb{E}[y|x])(\mathbb{E}[$$

$y[x]$ and hence L are stochastic, so taking expectations w.r.t. test data y

$$\mathbb{E}_y[L[x]] = (f[x, \phi] - \mathbb{E}[y|x])^2 + 2(f[x, \phi] - \mathbb{E}[y|x]$$
$$= (f[x, \phi] - \mathbb{E}[y|x])^2 + 0 + \sigma^2$$

Taking expectations with respect to training data D

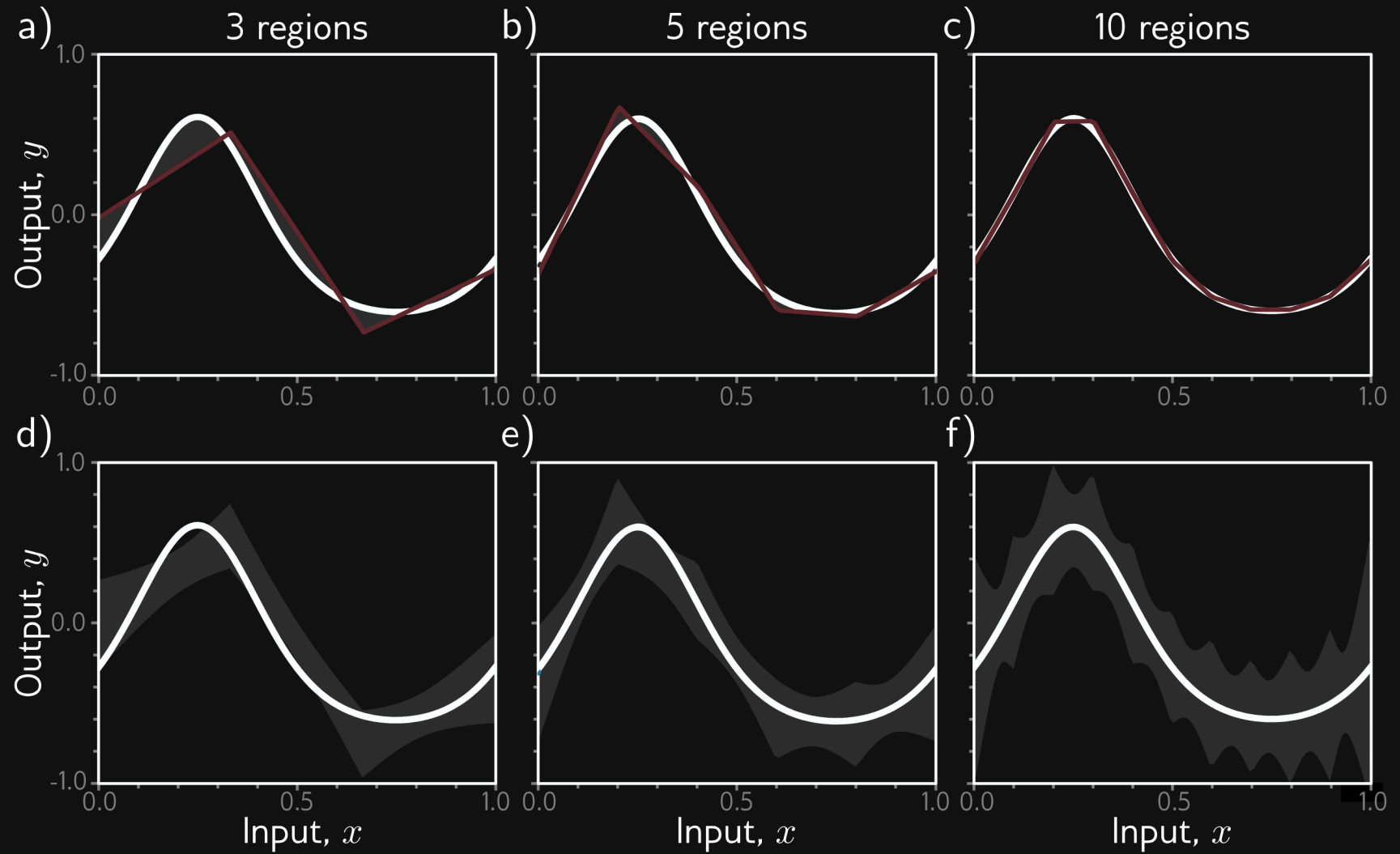
$$\mathbb{E}_D\left[\mathbb{E}_y\left[L[x]\right]\right] = \mathbb{E}_D\left[(f[x,\phi[D]] - \mathbb{E}_D\left[f[x,\phi[D]]\right]\right]$$
$$= \text{Variance} + \text{Bias}^2 + \text{Noise}$$

where

- variance due to sample noise in training data
- bias due to model misspecification
- noise due to inherent uncertainty in $y|x$

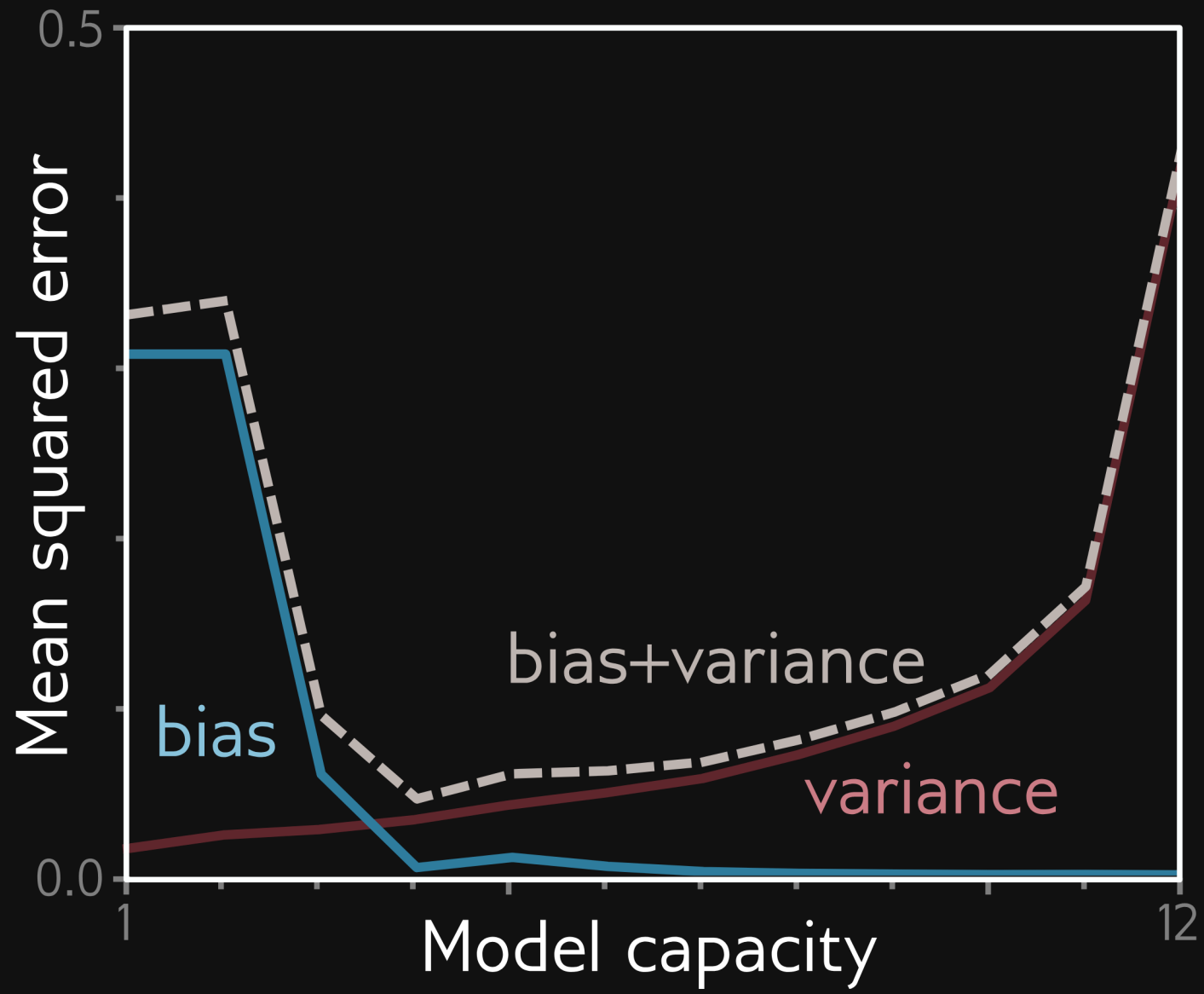
More Training Data: Less Variance

Increasing Model Capacity: Less Bias but More Variance



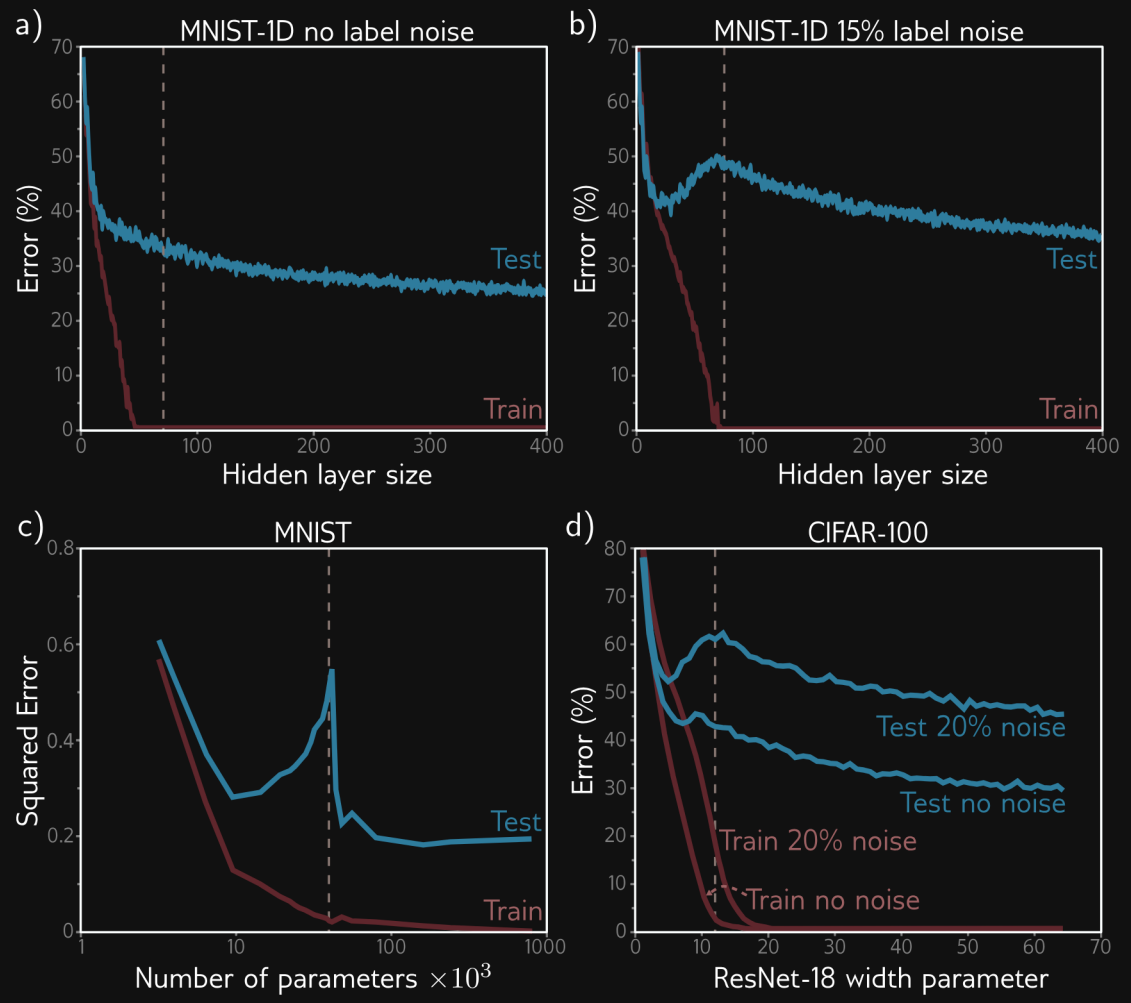
UNIVERSITY OF CAMBRIDGE

Bias-Variance Tradeoff



UNIVERSITY OF CAMBRIDGE

Double Descent (Overparametrization)



UNIVERSITY OF CAMBRIDGE

Overparametrization permits (and initialization and/or fitting may encourage) smoother interpolation between training data (implicit regularization), an inductive bias resulting in better generalization.

Cross-Validation

- Learn model parameters on training set
- Choose hyperparameters using validation set
- Evaluate final model on test set

References

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