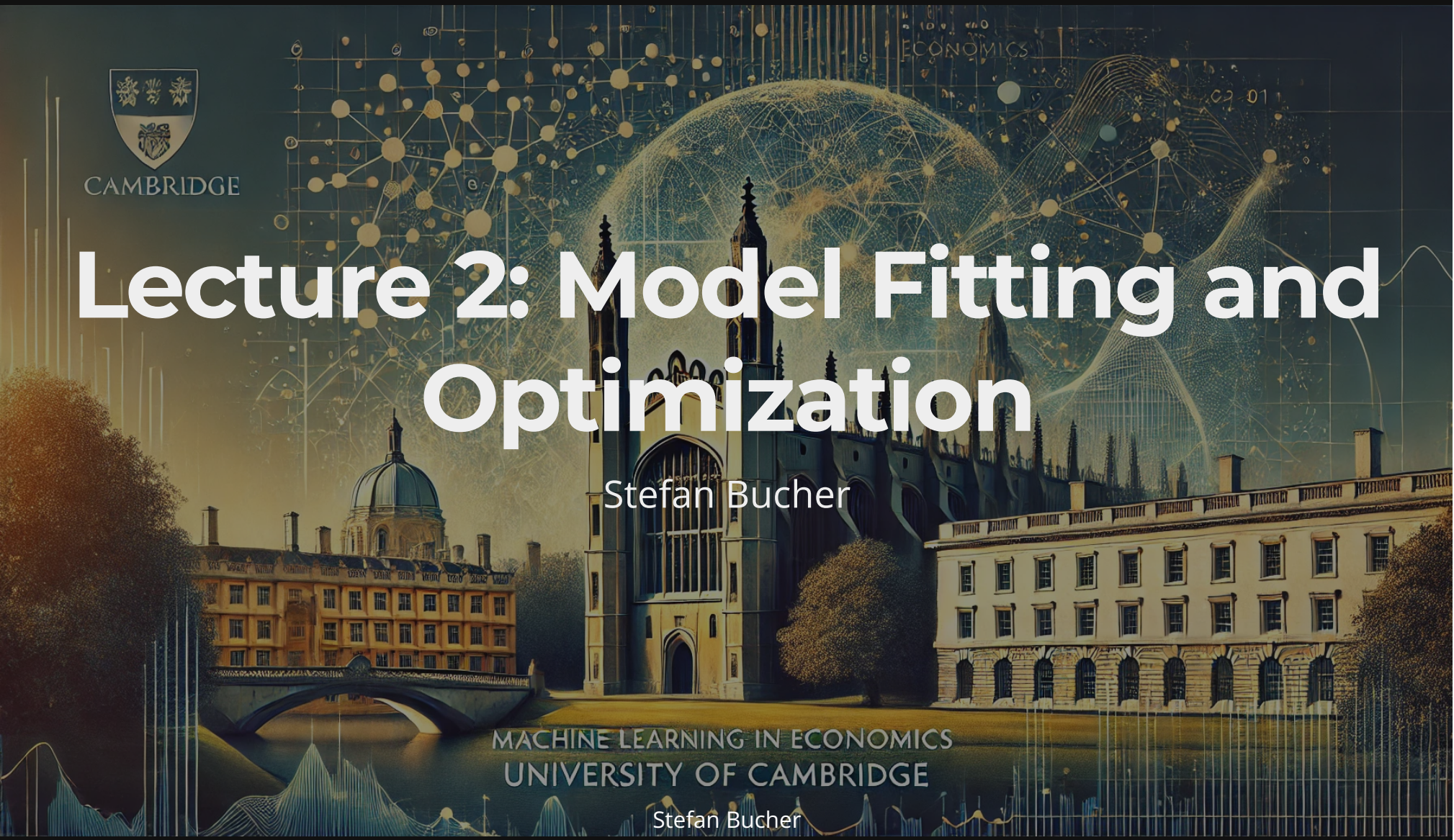


Lecture 2: Model Fitting and Optimization

Stefan Bucher

Open in Colab

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

What is a Model?

> “A model is a probability distribution over a sequence of [observable] random variables.” — Thomas Sargent

Review: Linear Regression

What did we do?

- We chose a model (distribution) $\mathbf{Y}|\mathbf{X} \sim \mathcal{N}(\mathbf{X}\beta, \sigma^2\mathbf{I}_n)$ with parameters β
- We chose an objective function (loss function) $L(\beta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - X_i\beta)^2$
- We (analytically) found the optimal parameter values $\hat{\beta}$ that minimize the loss function

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

Model Fitting & Optimization

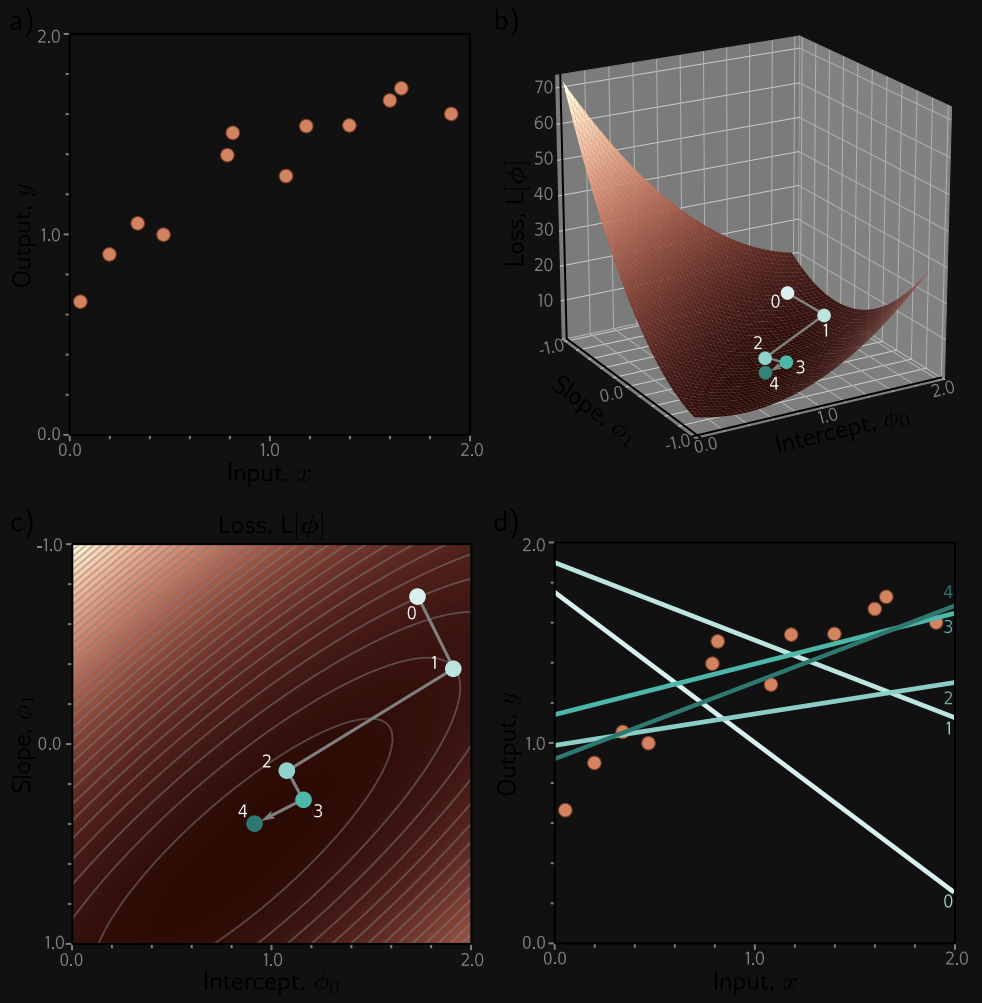
$$\hat{\phi} = \arg\min_{\phi} L(\phi)$$

Prince (2023, chap. 6)

Gradient Descent[1]

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

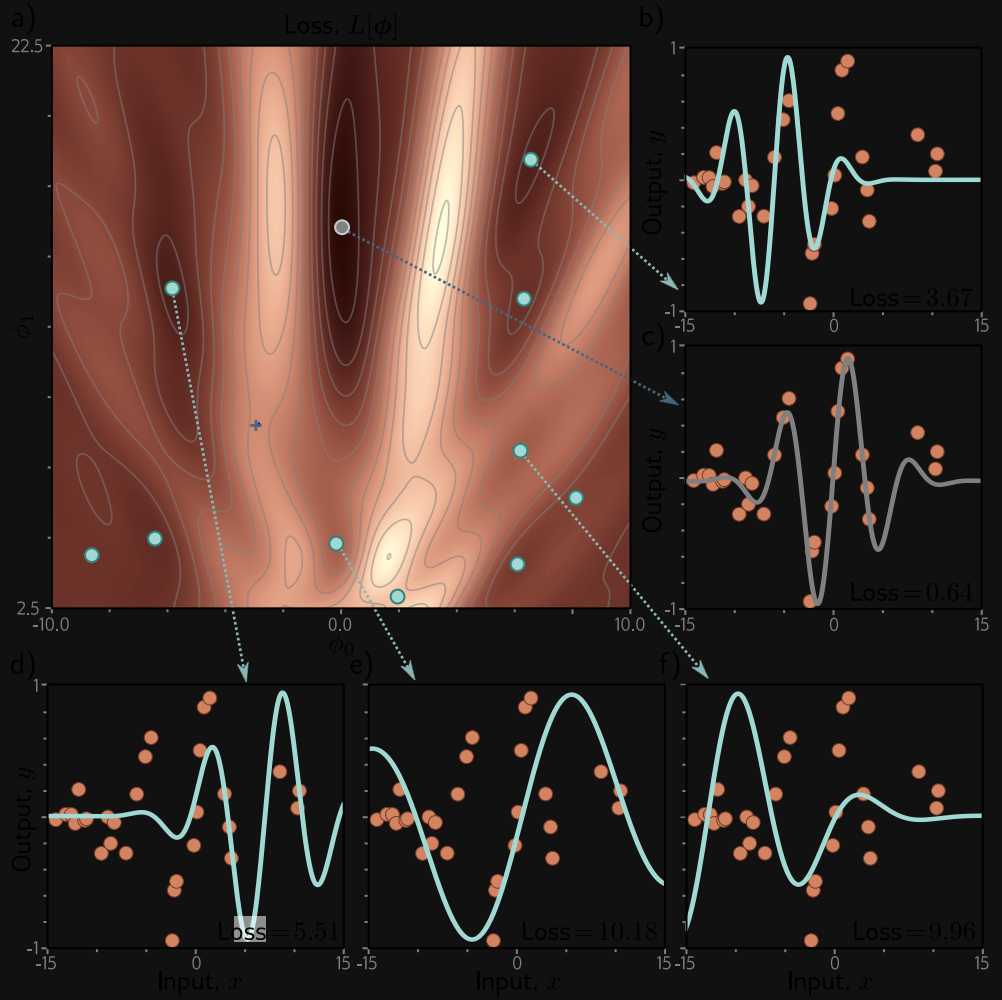
where α fixed learning rate or line search.


a)
Output, y
Input, x
b)
Loss, L[ϕ]
Slope, ϕ_1
Intercept, ϕ_0
c)
Loss, L[ϕ]
Slope, ϕ_1
Intercept, ϕ_0
d)
Output, y
Input, x

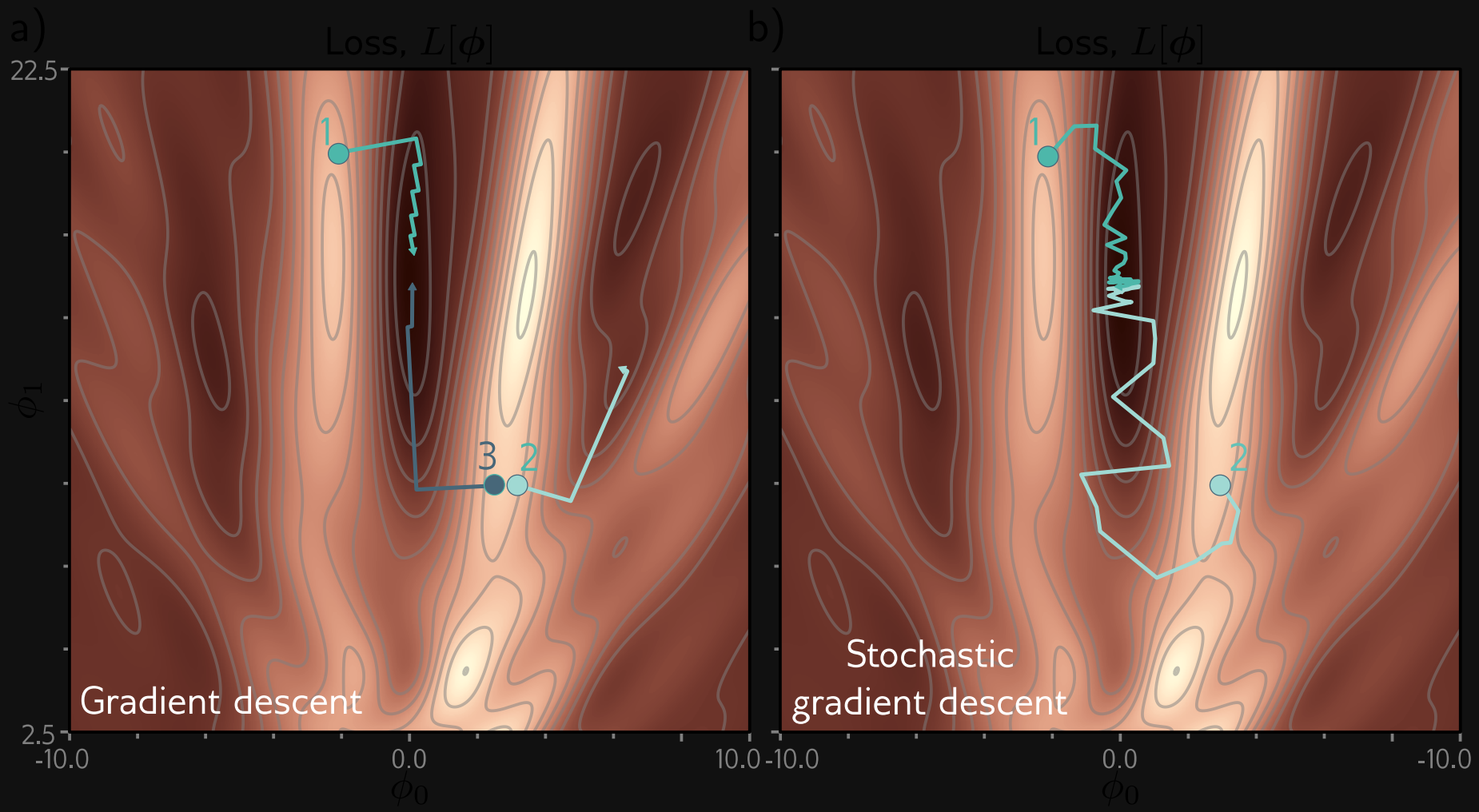
. Cauchy (1847)

Nonlinear Model

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



Stochastic Gradient Descent

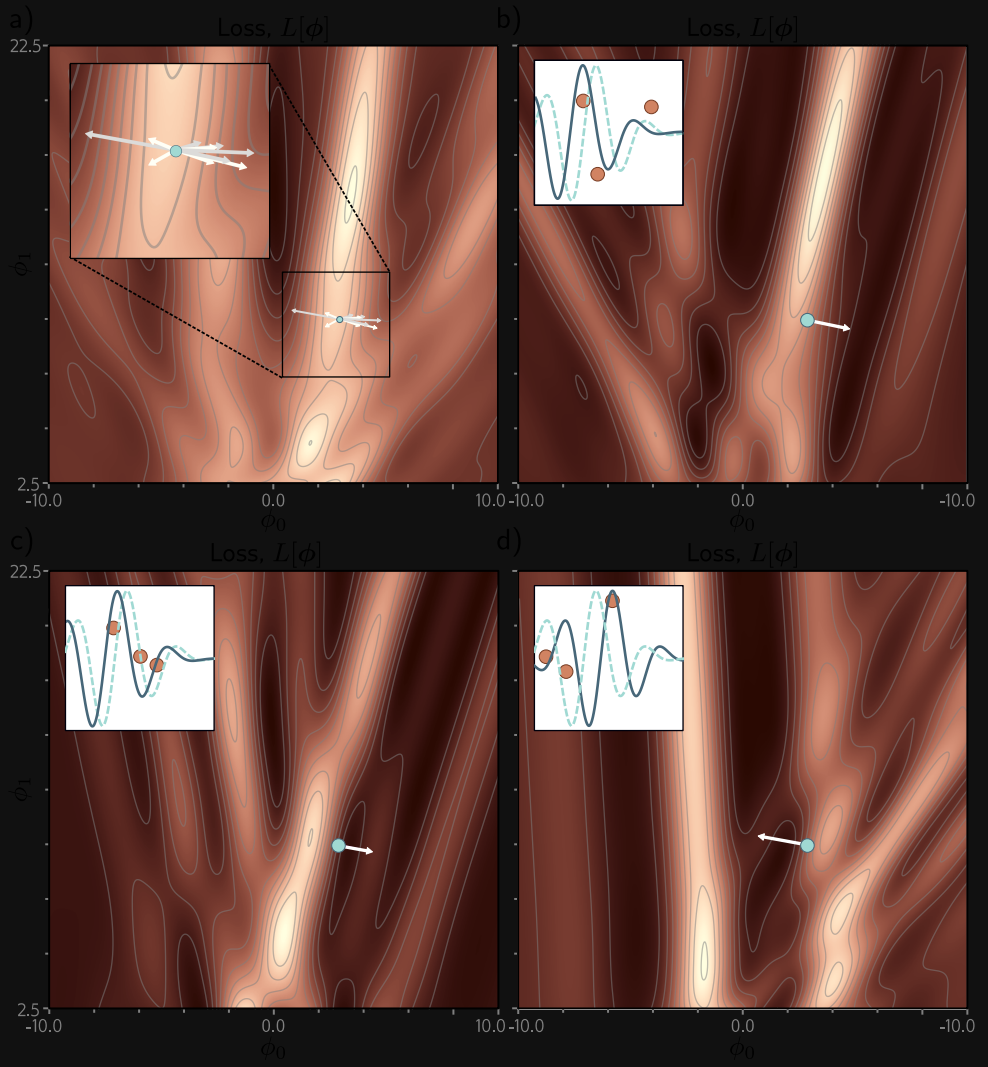


Stochastic Gradient Descent [1]

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.



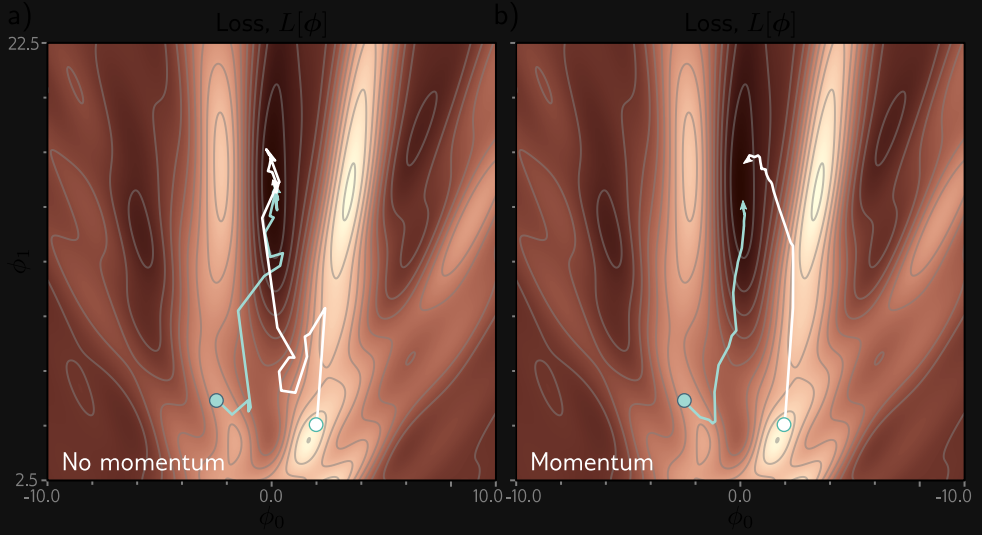
. Robbins & Monro (1951)

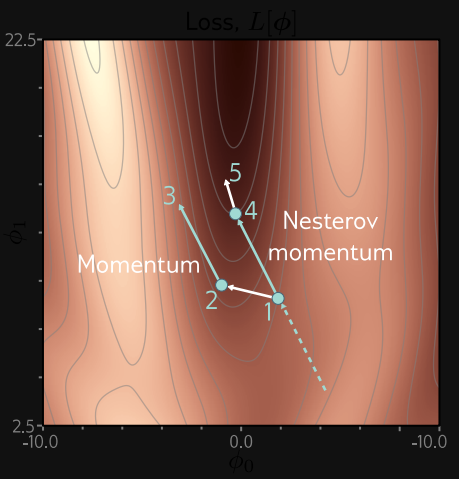
Nesterov Accelerated Momentum [1]

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

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

adds momentum (m_t) *before* evaluating gradient.





. Nesterov (1983)

Adaptive Moment Estimation (Adam) [1]

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

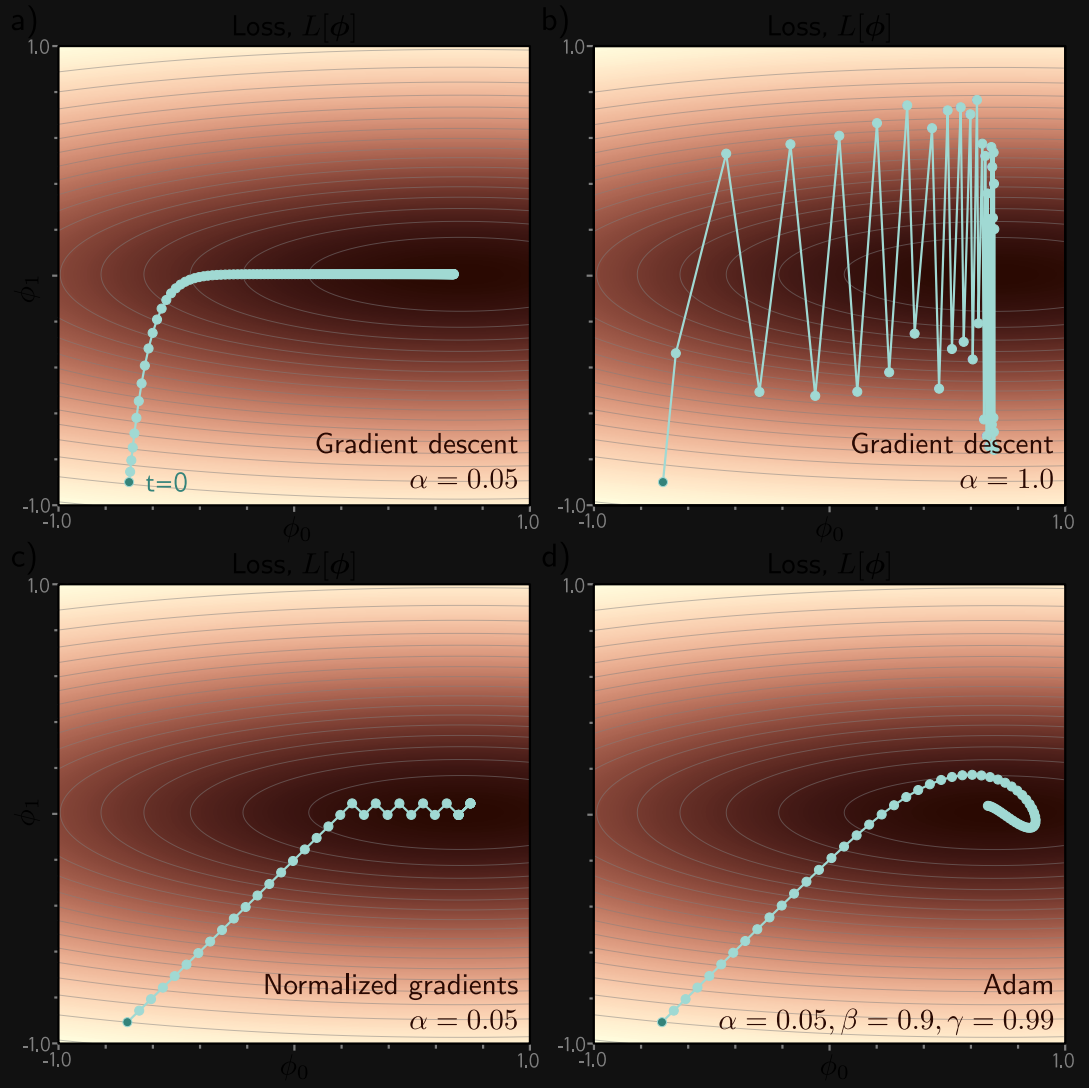
$$m_{t+1} \leftarrow \beta m_t + (1-\beta) \sum_{i \in B_t} \nabla_\phi l_i(\phi_t; x_i, y_i)$$

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

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

. 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 from sklearn import datasets as sk_datasets
 8 import matplotlib.pyplot as plt
 9 import random
10
11 if torch.backends.mps.is_available():
12   DEVICE = torch.device("mps") # Apple Silicon Metal Performance Shaders
13 elif torch.cuda.is_available():
14   DEVICE = torch.device("cuda") # GPU
15 else:
16   DEVICE = torch.device("cpu") # CPU
17
18 X, y = sk_datasets.load_diabetes(return_X_y=True, as_frame=True)
19 X = X.iloc[-200:]
20 y = y.iloc[-200:]
```

Key components:

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

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 the intercept
3 intercept = solution.solution[-1].item()  # Intercept
4 print(coefficients[2], intercept)
```

```
tensor(753.9764) -112.08257293701172
```

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)
    loss = criterion(outputs, y_train)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

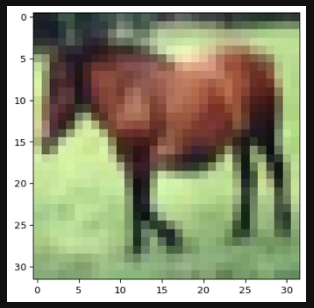
```
Epoch [1000/15000], Loss: 27419.0312
Epoch [2000/15000], Loss: 24670.3789
Epoch [3000/15000], Loss: 22169.6973
Epoch [4000/15000], Loss: 19888.8906
Epoch [5000/15000], Loss: 17810.4355
Epoch [6000/15000], Loss: 15922.4639
```

```
Epoch [7000/15000], Loss: 14218.2754
Epoch [8000/15000], Loss: 12694.2559
Epoch [9000/15000], Loss: 11347.4893
Epoch [10000/15000], Loss: 10174.7949
Epoch [11000/15000], Loss: 9172.2383
Epoch [12000/15000], Loss: 8334.8389
Epoch [13000/15000], Loss: 7656.2744
Epoch [14000/15000], Loss: 7128.9736
Epoch [15000/15000], Loss: 6743.2417
linear.weight: [[ 53.528606  21.761     31.14344   -8.373393  32.205452 -49.149002
   58.254898  85.72925  -20.953453 -15.810432]]
linear.bias: [133.13715]
```

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 Width x Color(RGB)
plt.show()
```

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



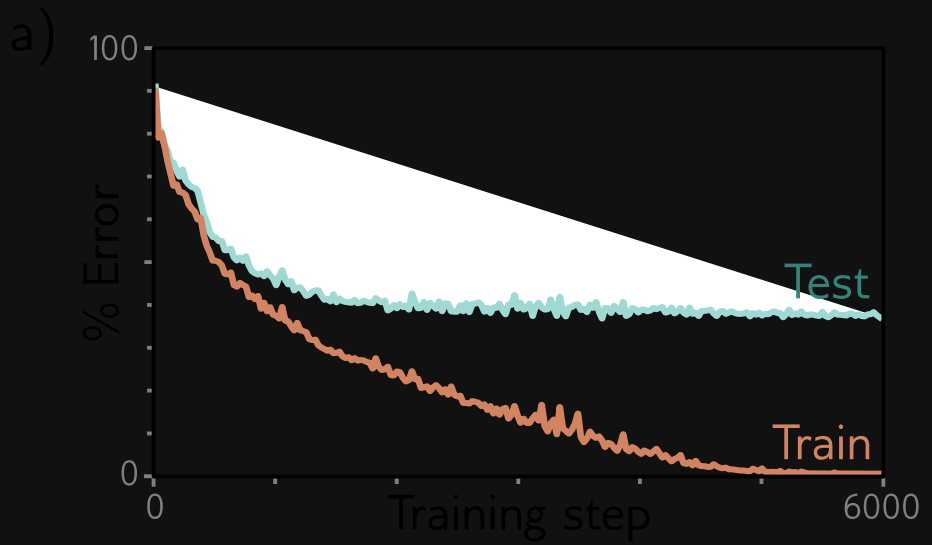
Data Loader

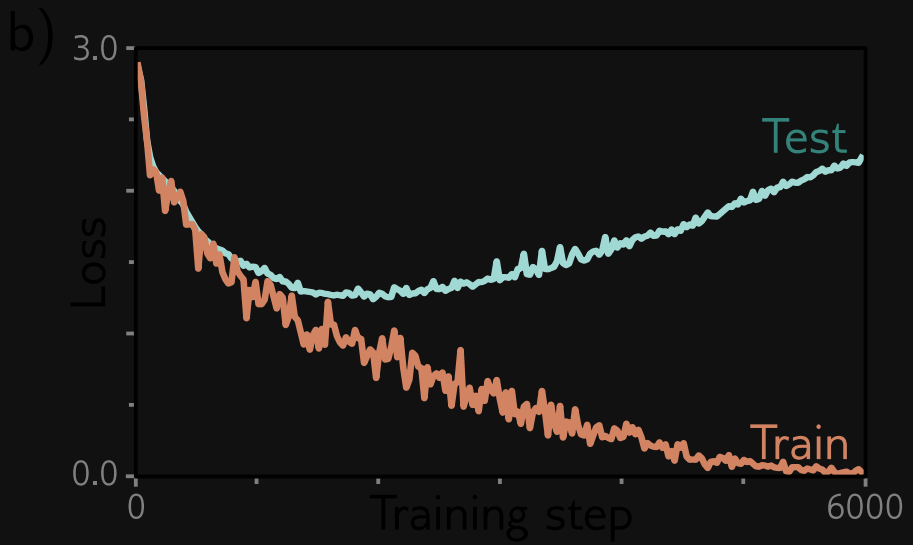
```
# Data Loader for Batching
def seed_worker(worker_id):
  worker_seed = torch.initial_seed() % 2**32
  numpy.random.seed(worker_seed)
  random.seed(worker_seed)
g_seed = torch.Generator()
g_seed.manual_seed(torch.initial_seed() % 2**32)

train_dataloader = DataLoader(training_data, batch_size=64, shuffle=True, num_workers=2, worker_init_
test_dataloader = DataLoader(test_data,  batch_size=64, shuffle=True, num_workers=2, worker_init_fn=s

# 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

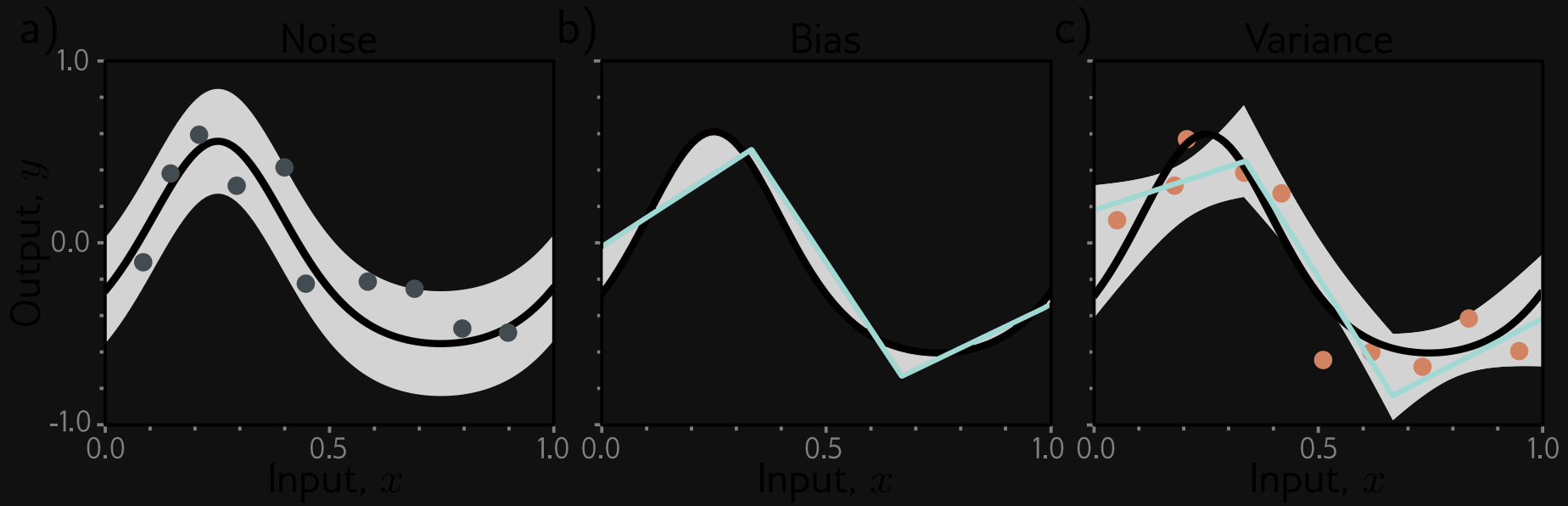
Prince (2023, chap. 8)

Training and Test Error





Error Sources



Error Decomposition

In a linear regression,

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

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

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

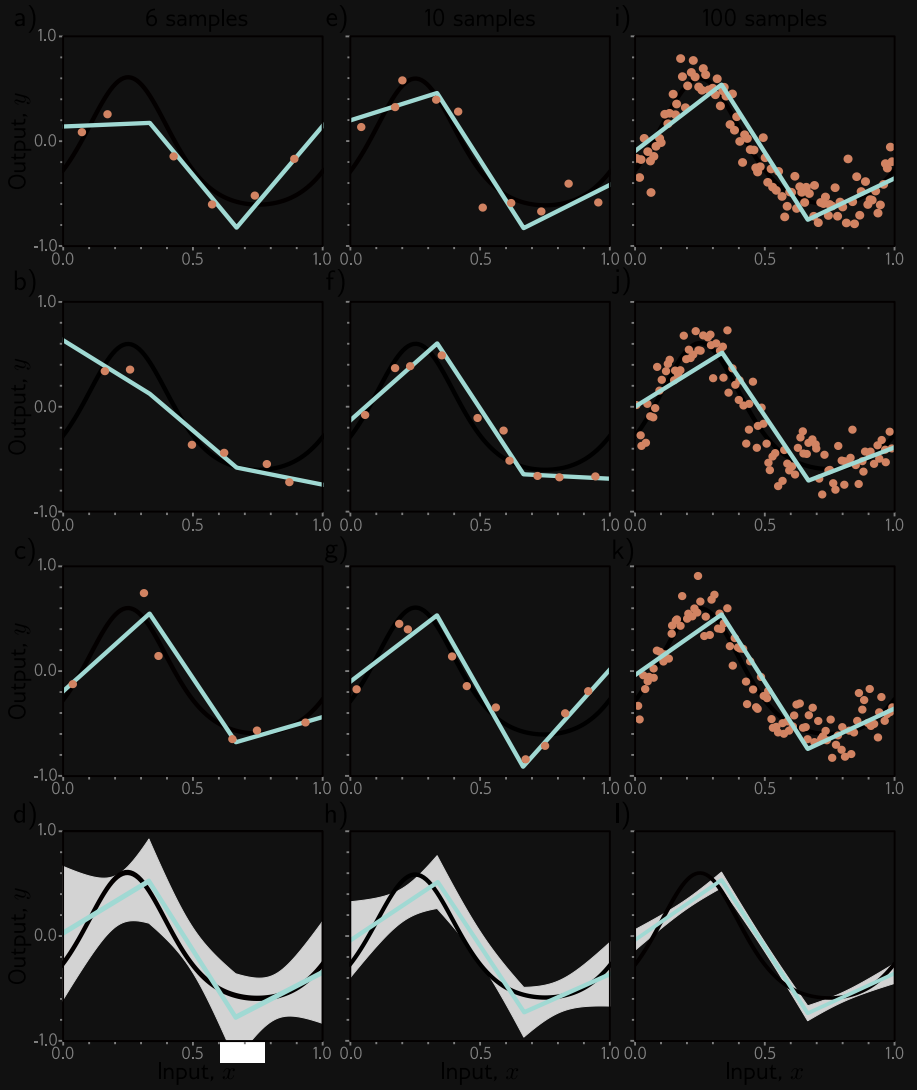
Taking expectations with respect to training data D

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

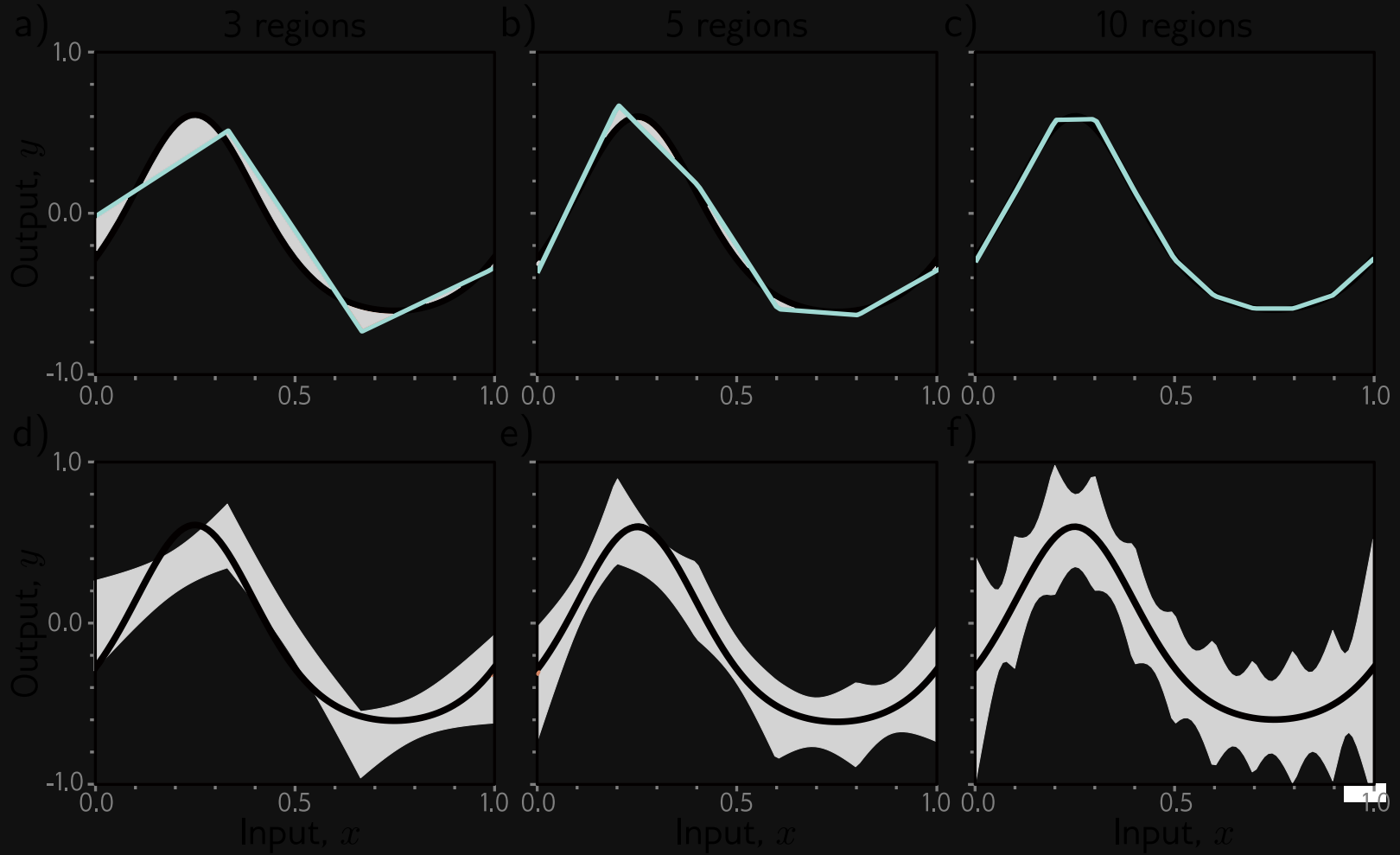
where

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

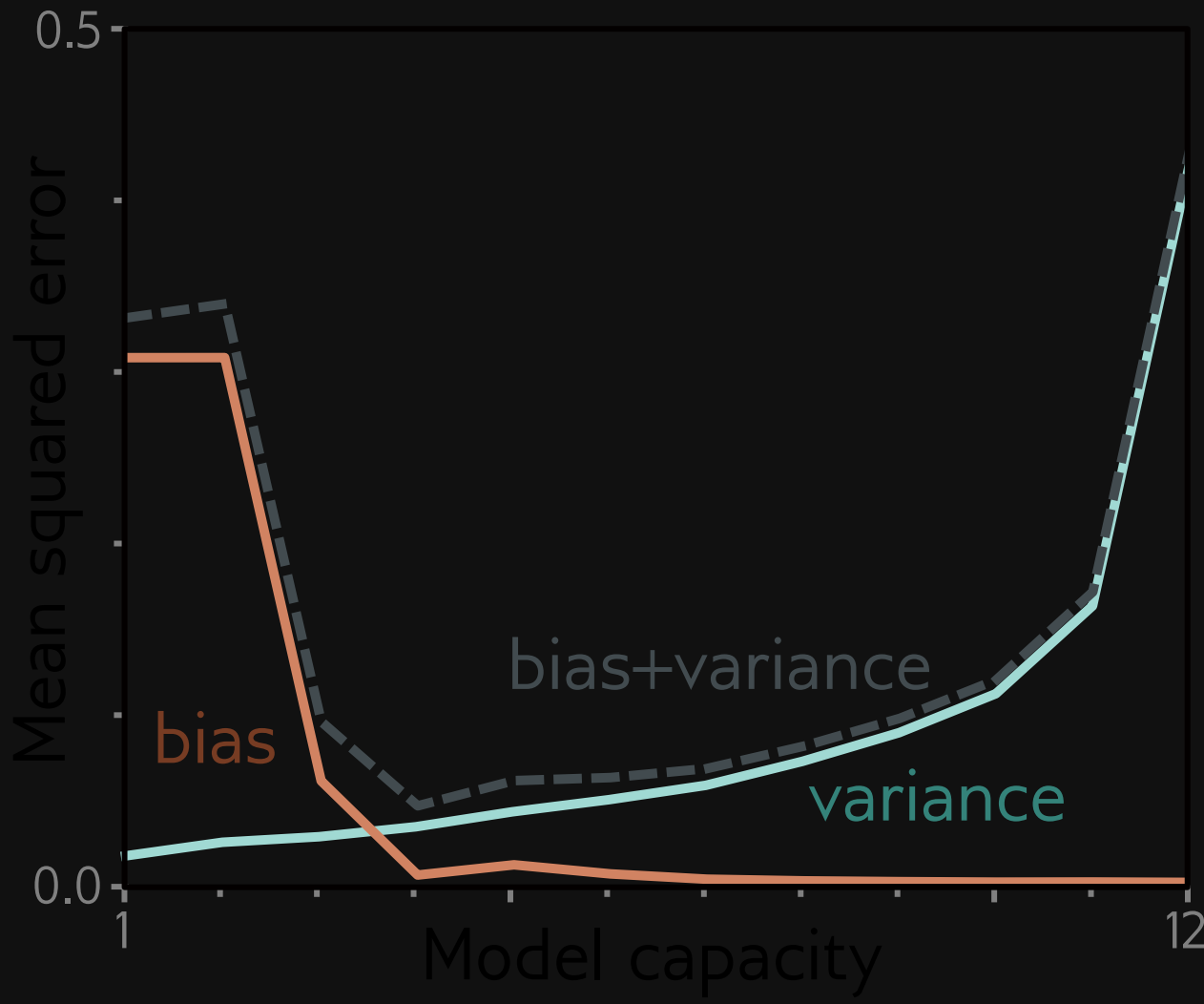
More Training Data: Less Variance



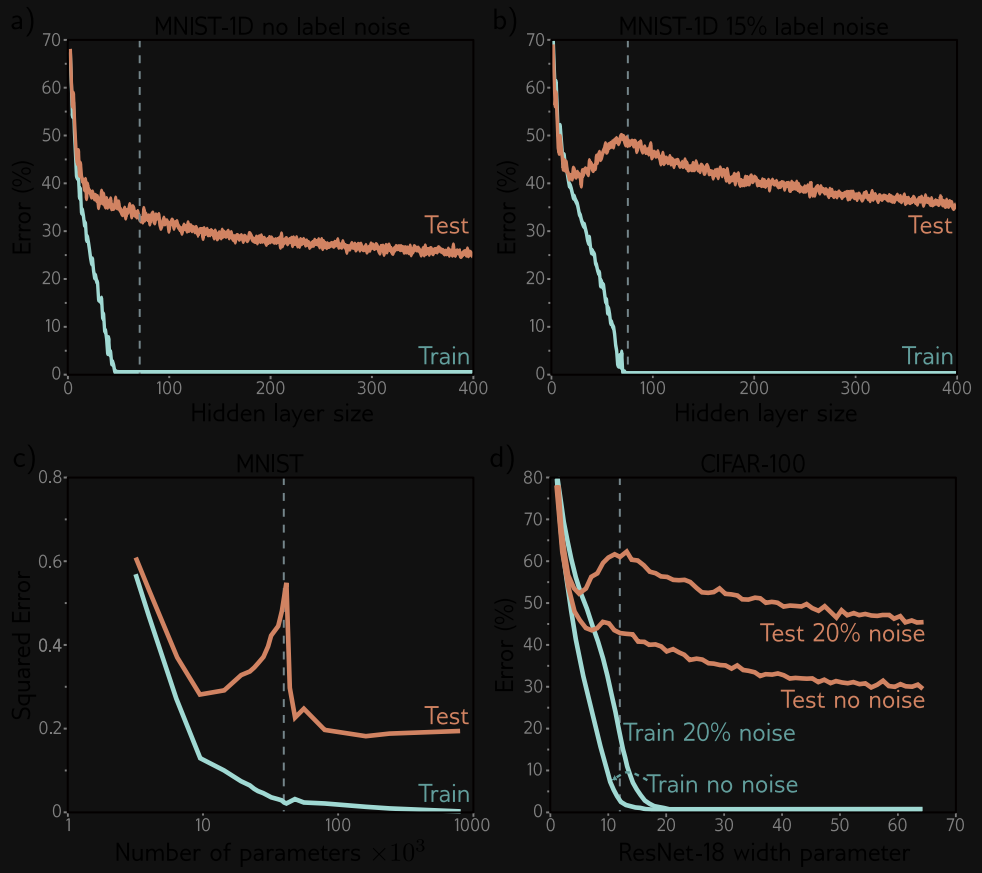
Increasing Model Capacity: Less Bias but More Variance



Bias-Variance Tradeoff



Double Descent (Overparametrization)



Overparametrization permits (and initialization and/or fitting may encourage) smoother interpolation between training data (implicit

Cross-Validation

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

Classification: Discrete Choice

Classification

- Linear regression: continuous outcome (e.g. house price, earnings, etc.)
- Classification: discrete outcome (e.g. product choice, default vs no default, employment status)

Discrete Choice

- Outcome: $y_i \in \{1, \ldots, J\}$ (choice among J alternatives), covariates $\mathbf{X}_i$
- Goal: model $P(y_i = j|\mathbf{X}_i)$

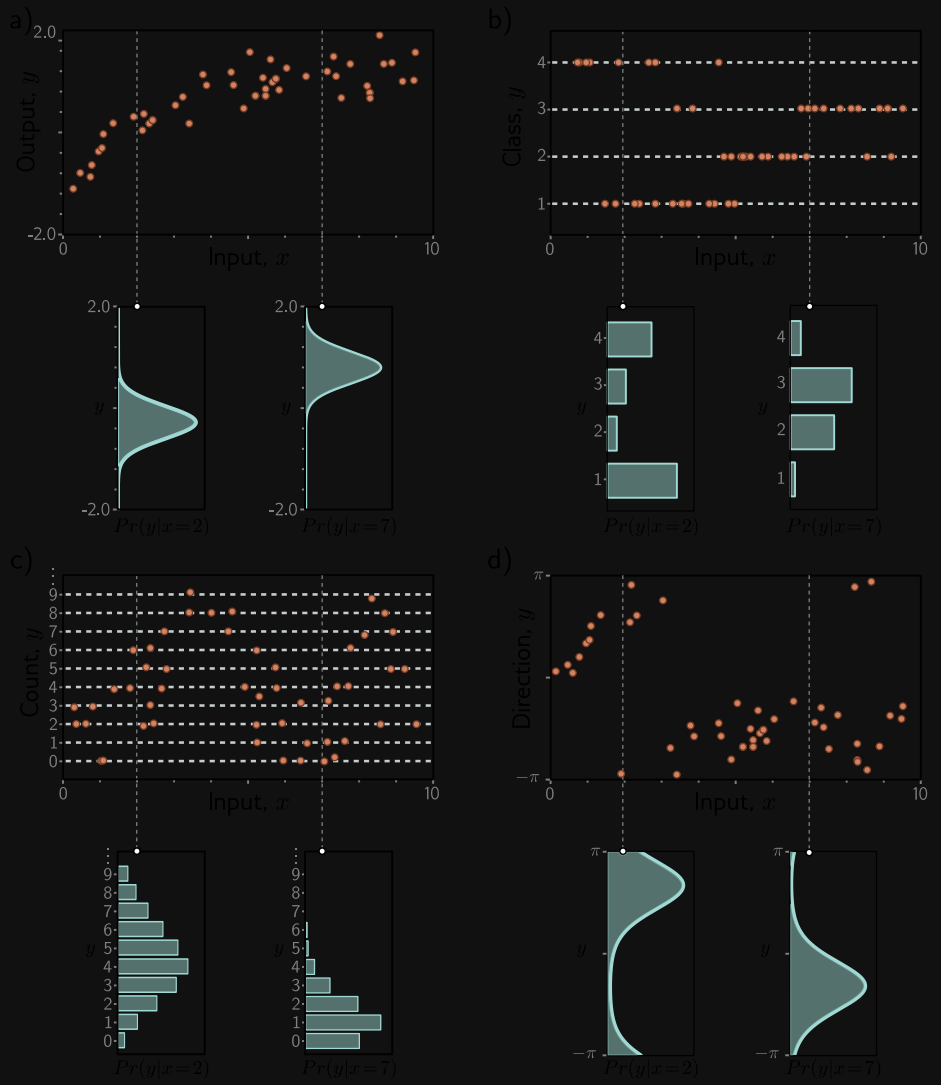
Random Utility Model and Multinomial Logit

- **Random utility**[1]: agent chooses alternative with highest utility
- $U_{ij} = \mathbf{X}_i \beta_j + \varepsilon_{ij}$; choose j iff $U_{ij} > U_{ia}$ for all $a \neq j$
- If ε_{ij} i.i.d. **Gumbel** (extreme value type I), choice probabilities have a closed form (**multinomial logit**, or softmax)

$$P(y_i = j|\mathbf{X}_i) = \frac{\exp(\mathbf{X}_i \beta_j)}{\sum_{a=1}^{J} \exp(\mathbf{X}_i \beta_a)}$$

. McFadden (1974); Nobel Prize (2000)

Regression vs. Classification

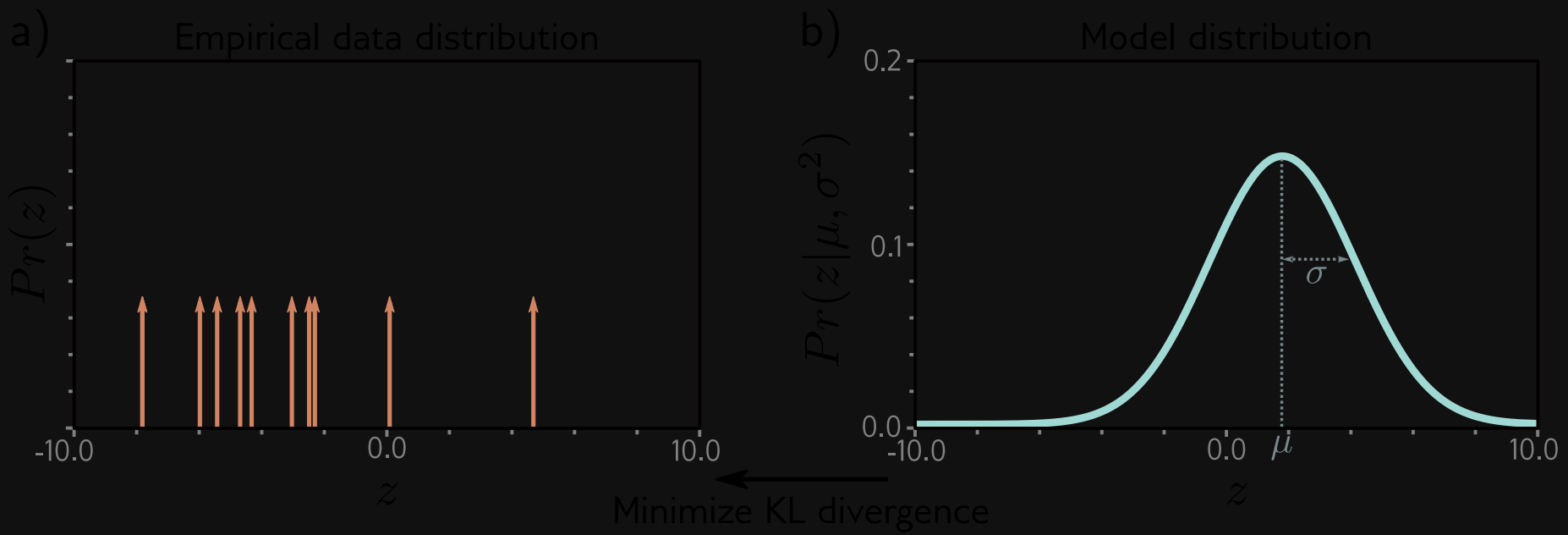


Loss Functions

Prince (2023, chap. 5)

What is a Good Model?

Model (distribution) Q that is *close* to the true distribution P (of which we observe an empirical sample).



Cross-Entropy Loss (= Max LLH)

Minimizing KL div. from model $Q_\theta(y|x)$ to empirical $P(y|x)$

$$
\begin{aligned}
\hat{\theta} &= \arg\min_\theta D_{KL}[P||Q] \\
&= \arg\min_\theta \int_{-\infty}^{\infty} p(y|x) \log \frac{p(y|x)}{q_\theta(y|x)} dy \\
&= \arg\min_\theta - \int_{-\infty}^{\infty} p(y|x) \log q_\theta(y|x) dy \\
&= \arg\min_\theta - \int_{-\infty}^{\infty} \left(\frac{1}{I} \sum_{i=1}^{I} \delta[y - y_i] \right) \log q_\theta(y|x) dy \\
&= \arg\min_\theta - \sum_{i=1}^{I} \log q_\theta(y_i|x_i)
\end{aligned}
$$

is equivalent to minimizing the negative log-likelihood

$$\hat{\phi} = \arg\min_{\phi} - \sum_{i=1}^{I} \log q(y_i | \theta_i = f[x_i, \phi])$$

Distribution parametrized by ML model.

Maximizing Log-Likelihood

$$\hat{\phi} = \arg\max_{\phi} \prod_{i=1}^{I} q(y_i|\theta_i = f[x_i, \phi])$$

$$= \arg\max_{\phi} \log \prod_{i=1}^{I} q(y_i|\theta_i = f[x_i, \phi])$$

$$= \arg\min_{\phi} - \sum_{i=1}^{I} \log q(y_i|\theta_i = f[x_i, \phi])$$

Lin. Reg.: Least Squares Loss

Assuming $y_i|x_i \sim N(f[x_i, \phi], \sigma^2)$

$$\hat{\phi} = \underset{\phi}{\arg\min} - \sum_{i=1}^{I} \log q(y_i|\theta_i = f[x_i, \phi])$$

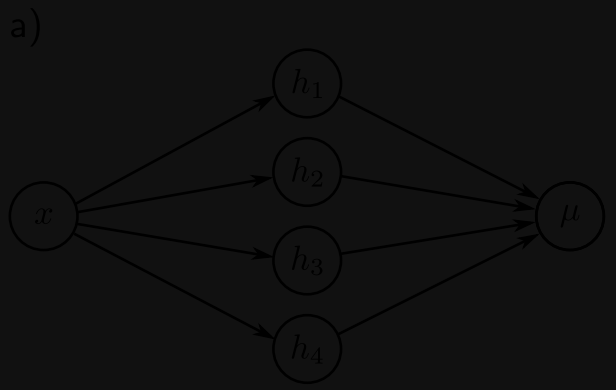
$$= \underset{\phi}{\arg\min} - \sum_{i=1}^{I} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y_i - f[x_i, \phi])^2}{2\sigma^2}\right)\right)$$

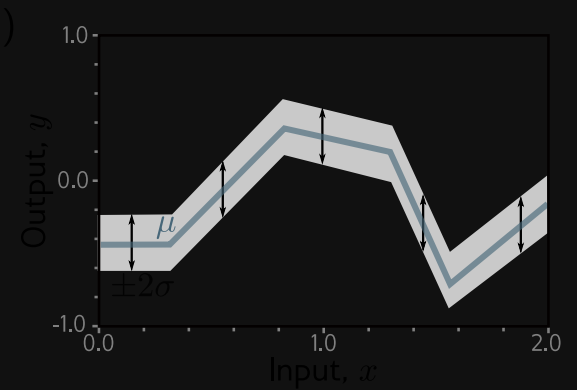
$$= \underset{\phi}{\arg\min} \sum_{i=1}^{I} -\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \frac{(y_i - f[x_i, \phi])^2}{2\sigma^2}$$

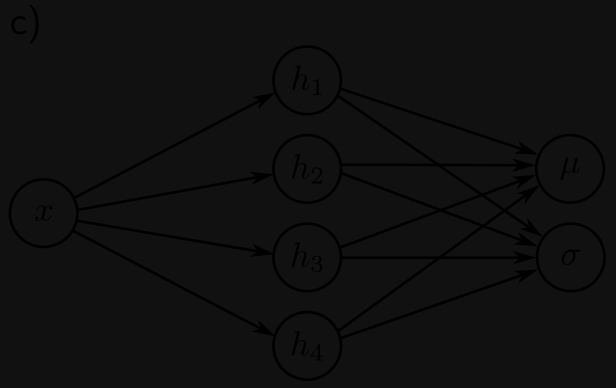
$$= \underset{\phi}{\arg\min} \sum_{i=1}^{I} (y_i - f[x_i, \phi])^2$$

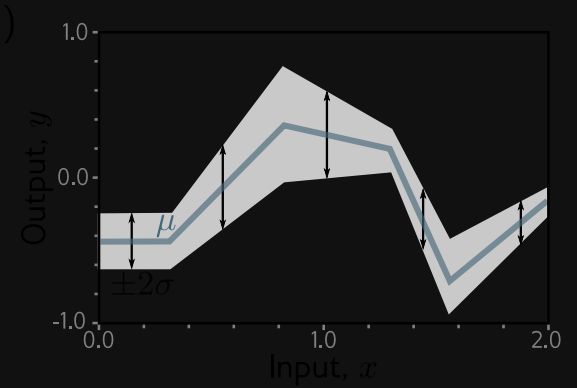
Heteroscedastic Regression

$$\arg\min_{\phi} -\sum_{i=1}^{I} \log\left(\frac{1}{\sqrt{2\pi f_2[x_i,\phi]^2}}\exp\left(-\frac{(y_i - f_1[x_i,\phi])^2}{2f_2[x_i,\phi]^2}\right)\right)$$









Binary Classification: Binary Cross-Entropy Loss

Assuming $y_i|x_i \sim Ber(\lambda_i = logit(f[x_i, \phi]))$

$$\hat{\phi} = \arg\min_{\phi} -\sum_{i=1}^{I} \log\Big((1 - logit(f[x_i, \phi]))^{1-y_i} logit(f[x_i, \phi])^{y_i}\Big)$$

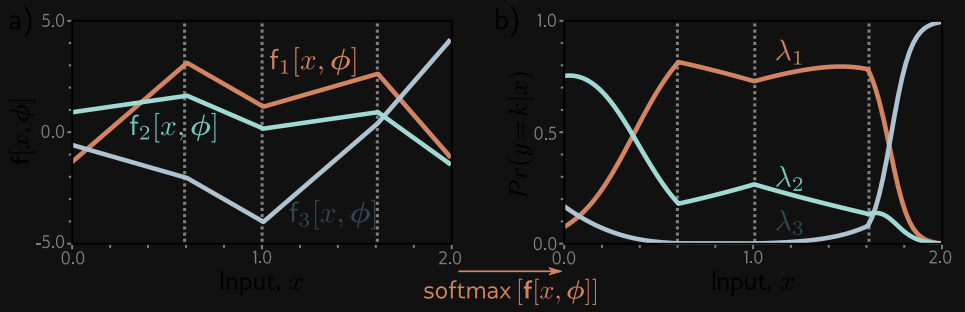
$$= \arg\min_{\phi} \sum_{i=1}^{I} -(1 - y_i)\log(1 - logit(f[x_i, \phi])) - y_i \log(logit(f[x_i,$$

Multiclass Classification: Multiclass Cross-Entropy Loss

Assuming $q(y_i = k|x_i) = softmax_k(f[x_i, \phi])$

$$\hat{\phi} = \underset{\phi}{\arg\min} - \sum_{i=1}^{I} \log\left(softmax_{y_i}(f[x_i, \phi])\right)$$

$$= \underset{\phi}{\arg\min} - \sum_{i=1}^{I} \left(f_{y_i}[x_i, \phi] - \log\left(\sum_{k'=1}^{K} \exp(f_{k'}[x_i, \phi]) \right) \right)$$



Multivariate Output

Assuming independent dimensions $q(y|f[x,\phi]) = \prod_d q(y_d|f_d[x,\phi])$

$$\hat{\phi} = \underset{\phi}{\arg\min} - \sum_{i=1}^{I} \log\left(\prod_d q(y_{i,d}|f_d[x_i,\phi])\right)$$

$$= \underset{\phi}{\arg\min} - \sum_{i=1}^{I} \sum_d \log\left(q(y_{i,d}|f_d[x_i,\phi])\right)$$

References

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