

# Optimisation

ACTL3143 & ACTL5111 Deep Learning for Actuaries  
Patrick Laub





## Warning

This page is out of date for 2024, and will be updated shortly.

# Lecture Outline

- **Dense Layers in Matrices**
- Optimisation
- Loss and derivatives



# Logistic regression

Observations:  $\mathbf{x}_{i,\bullet} \in \mathbb{R}^2$ .

Target:  $y_i \in \{0, 1\}$ .

Predict:  $\hat{y}_i = \mathbb{P}(Y_i = 1)$ .

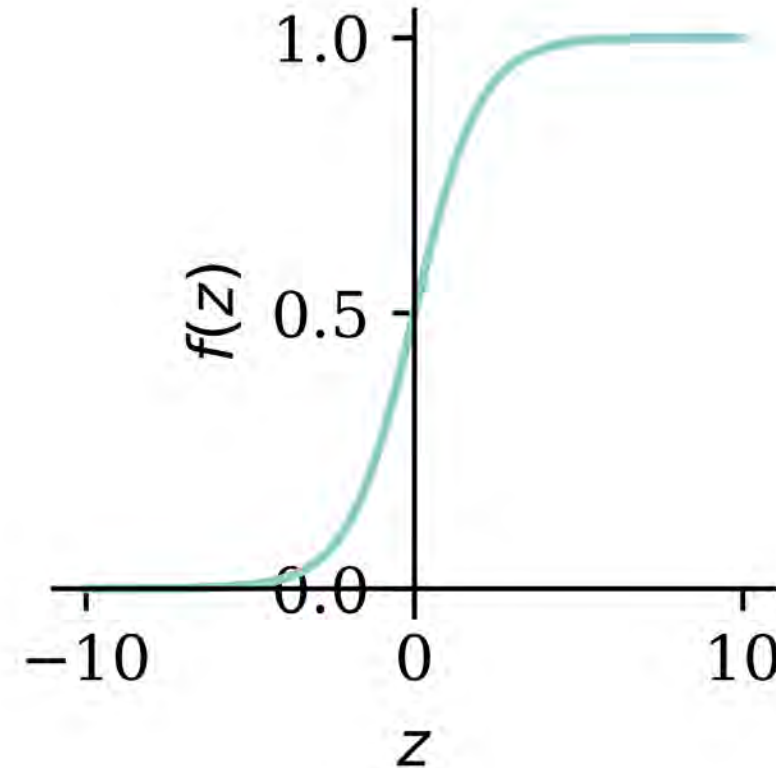
## The model

For  $\mathbf{x}_{i,\bullet} = (x_{i,1}, x_{i,2})$ :

$$z_i = x_{i,1}w_1 + x_{i,2}w_2 + b$$

$$\hat{y}_i = \sigma(z_i) = \frac{1}{1 + e^{-z_i}}.$$

```
1 import sympy
2 sympy.plot("1/(1 + exp(-z))");
```



# Multiple observations

```
1 data = pd.DataFrame({"x_1": [1, 3, 5], "x_2": [2, 4, 6], "y": [0, 1, 1]})
2 data
```

	x_1	x_2	y
0	1	2	0
1	3	4	1
2	5	6	1

Let  $w_1 = 1$ ,  $w_2 = 2$  and  $b = -10$ .

```
1 w_1 = 1; w_2 = 2; b = -10
2 data["x_1"] * w_1 + data["x_2"] * w_2 + b
```

```
0    -5
1     1
2     7
dtype: int64
```



# Matrix notation

Have  $\mathbf{X} \in \mathbb{R}^{3 \times 2}$ .

```
1 X_df = data[["x_1", "x_2"]]
2 X = X_df.to_numpy()
3 X
```

```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

Let  $\mathbf{w} = (w_1, w_2)^\top \in \mathbb{R}^{2 \times 1}$ .

```
1 w = np.array([[1], [2]])
2 w
```

```
array([[1],
       [2]])
```

$$\mathbf{z} = \mathbf{X}\mathbf{w} + b, \quad \mathbf{a} = \sigma(\mathbf{z})$$

```
1 z = X.dot(w) + b
2 z
```

```
array([[ -5],
       [  1],
       [  7]])
```

```
1 1 / (1 + np.exp(-z))
```

```
array([[0.01],
       [0.73],
       [1.   ]])
```



# Using a softmax output

Observations:  $\mathbf{x}_{i,\bullet} \in \mathbb{R}^2$ . Predict: Target:  $\mathbf{y}_{i,\bullet} \in \{(1, 0), (0, 1)\}$ .  
 $\hat{y}_{i,j} = \mathbb{P}(Y_i = j)$ .

**The model:** For  $\mathbf{x}_{i,\bullet} = (x_{i,1}, x_{i,2})$

$$z_{i,1} = x_{i,1}w_{1,1} + x_{i,2}w_{2,1} + b_1,$$

$$z_{i,2} = x_{i,1}w_{1,2} + x_{i,2}w_{2,2} + b_2.$$

$$\hat{y}_{i,1} = \text{Softmax}_1(\mathbf{z}_i) = \frac{e^{z_{i,1}}}{e^{z_{i,1}} + e^{z_{i,2}}},$$

$$\hat{y}_{i,2} = \text{Softmax}_2(\mathbf{z}_i) = \frac{e^{z_{i,2}}}{e^{z_{i,1}} + e^{z_{i,2}}}.$$



# Multiple observations

1 data

	x_1	x_2	y_1	y_2
0	1	2	1	0
1	3	4	0	1
2	5	6	0	1

Choose:

$$w_{1,1} = 1, w_{2,1} = 2,$$

$$w_{1,2} = 3, w_{2,2} = 4, \text{ and}$$

$$b_1 = -10, b_2 = -20.$$

```
1 w_11 = 1; w_21 = 2; b_1 = -10
2 w_12 = 3; w_22 = 4; b_2 = -20
3 data["x_1"] * w_11 + data["x_2"] * w_21 + b_1
```

```
0    -5
1     1
2     7
dtype: int64
```





# Matrix notation

Have  $\mathbf{X} \in \mathbb{R}^{3 \times 2}$ .

```
1 X
```

```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

$\mathbf{W} \in \mathbb{R}^{2 \times 2}$ ,  $\mathbf{b} \in \mathbb{R}^2$

```
1 W = np.array([[1, 3], [2, 4]])
2 b = np.array([-10, -20])
3 display(W); b
```

```
array([[1, 3],
       [2, 4]])
array([-10, -20])
```

$$\mathbf{Z} = \mathbf{XW} + \mathbf{b}, \quad \mathbf{A} = \text{Softmax}(\mathbf{Z}).$$

```
1 Z = X @ W + b
2 Z
```

```
array([[ -5,  -9],
       [  1,   5],
       [  7,  19]])
```

```
1 np.exp(Z) / np.sum(np.exp(Z),
2   axis=1, keepdims=True)
```

```
array([[9.82e-01, 1.80e-02],
       [1.80e-02, 9.82e-01],
       [6.14e-06, 1.00e+00]])
```



# Lecture Outline

- Dense Layers in Matrices
- **Optimisation**
- Loss and derivatives



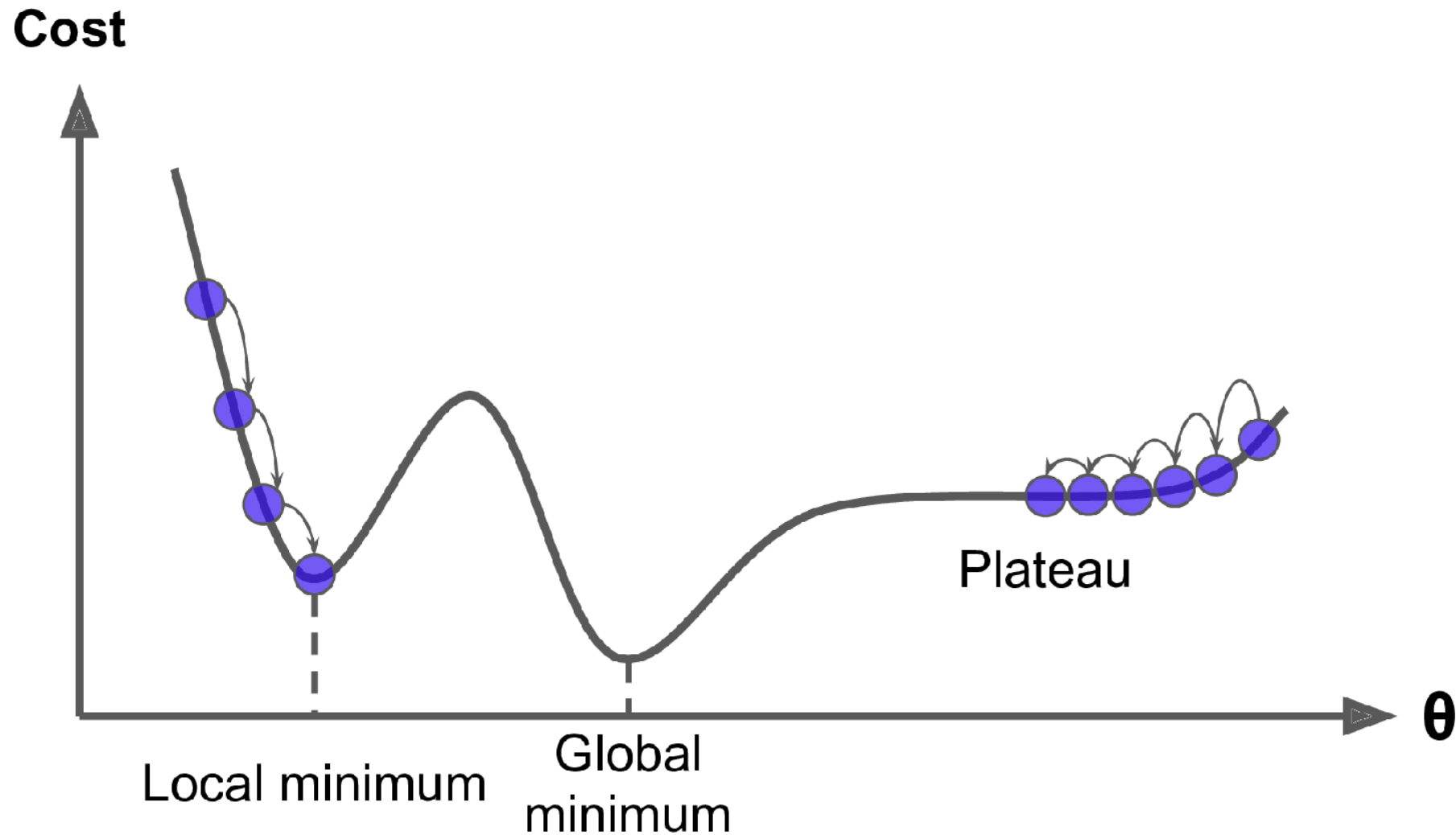
# Gradient-based learning

Make a guess:  50

Show derivatives: ☐ Reveal function: ☐



# Gradient descent pitfalls



Potential problems with gradient descent.

Source: Aurélien Géron (2019), *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd Edition, Figure 4-6.



# Go over all the training data

Called *batch gradient descent*.

```
1 for i in range(numEpochs):  
2     gradient = evaluate_gradient(loss_function, data, weights)  
3     weights = weights - learningRate * gradient
```



# Pick a random training example

Called *stochastic gradient descent*.

```
1 for i in range(numEpochs):  
2     rnd.shuffle(data)  
3     for example in data:  
4         gradient = evaluate_gradient(loss_function, example, weights)  
5         weights = weights - learningRate * gradient
```



# Take a group of training examples

Called *mini-batch gradient descent*.

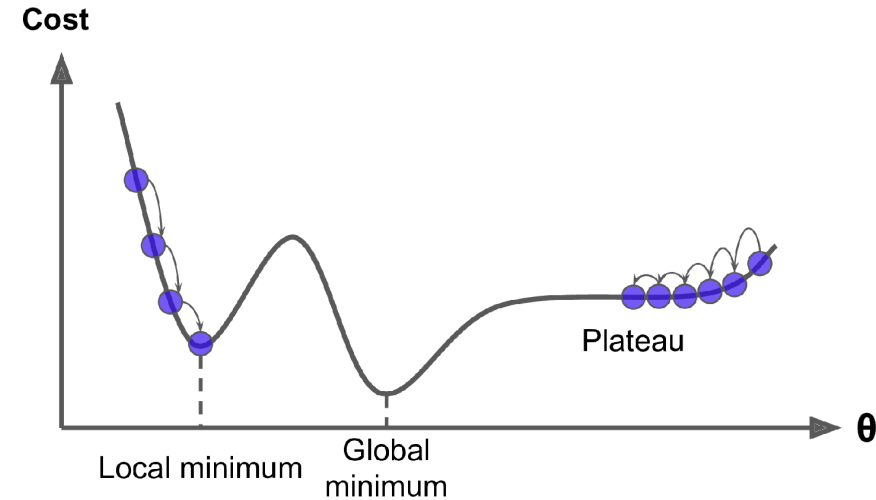
```
1 for i in range(numEpochs):  
2     rnd.shuffle(data)  
3     for b in range(numBatches):  
4         batch = data[b * batchSize : (b + 1) * batchSize]  
5         gradient = evaluate_gradient(loss_function, batch, weights)  
6         weights = weights - learningRate * gradient
```



# Mini-batch gradient descent

Why?

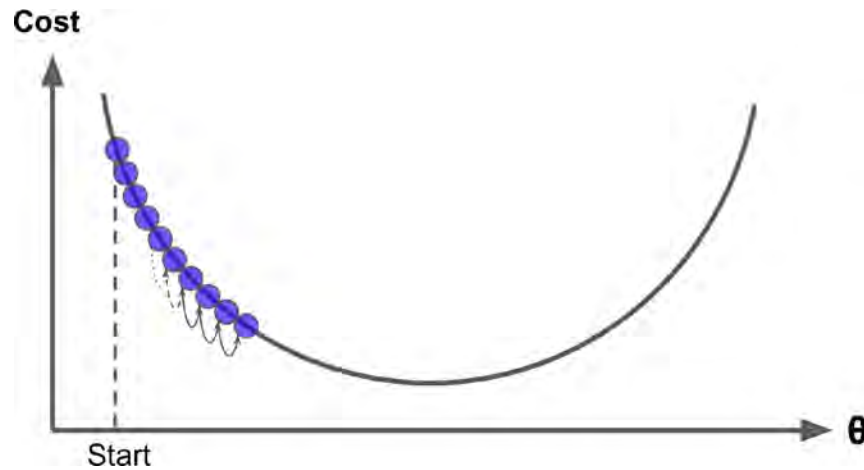
1. Because we have to (data is too big)
2. Because it is faster (lots of quick noisy steps > a few slow super accurate steps)
3. The noise helps us jump out of local minima



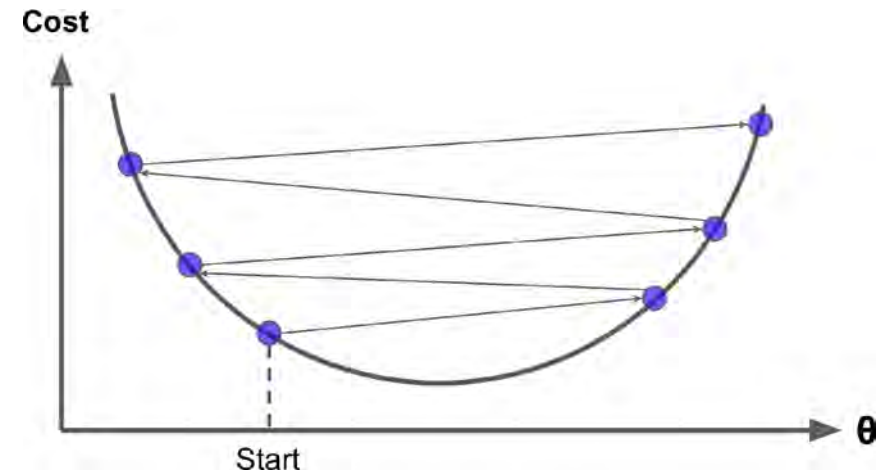
Example of jumping from local minima.



# Learning rates



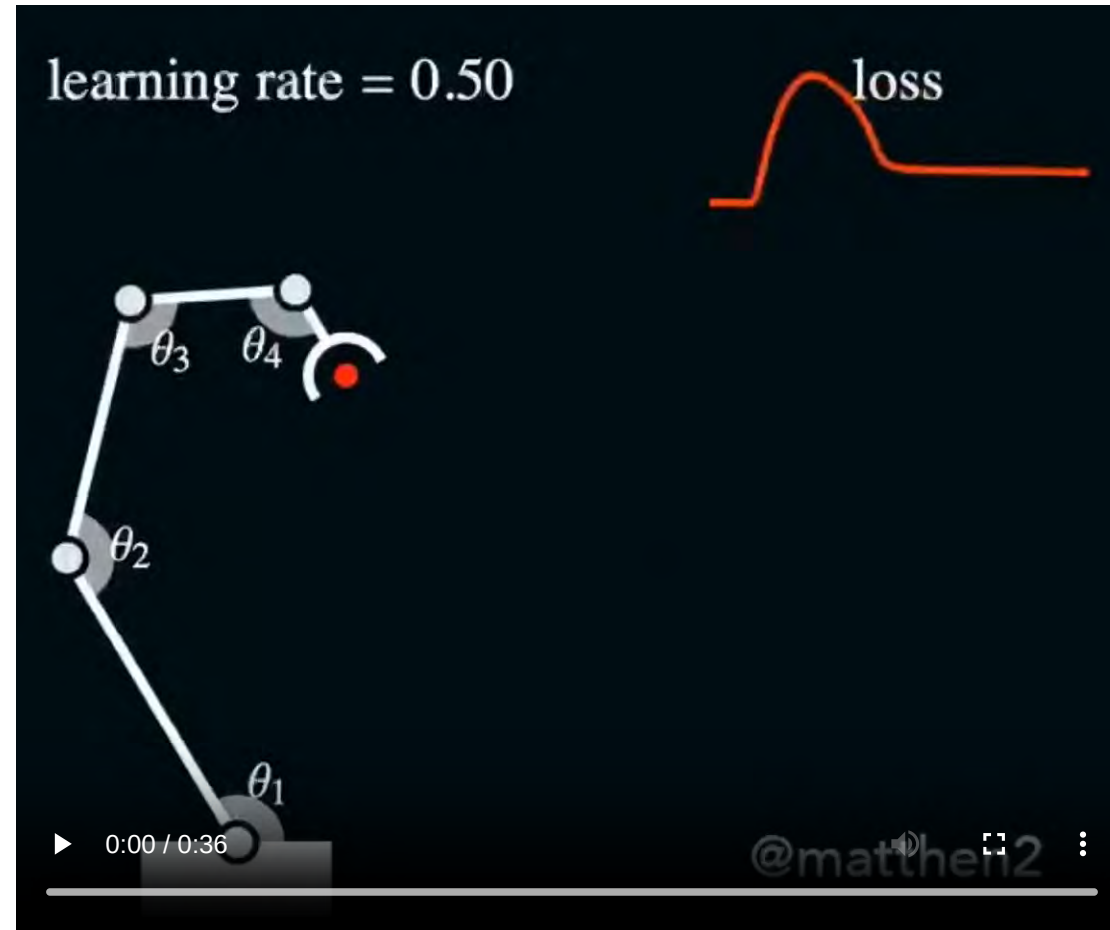
The learning rate is too small



The learning rate is too large



# Learning rates #2

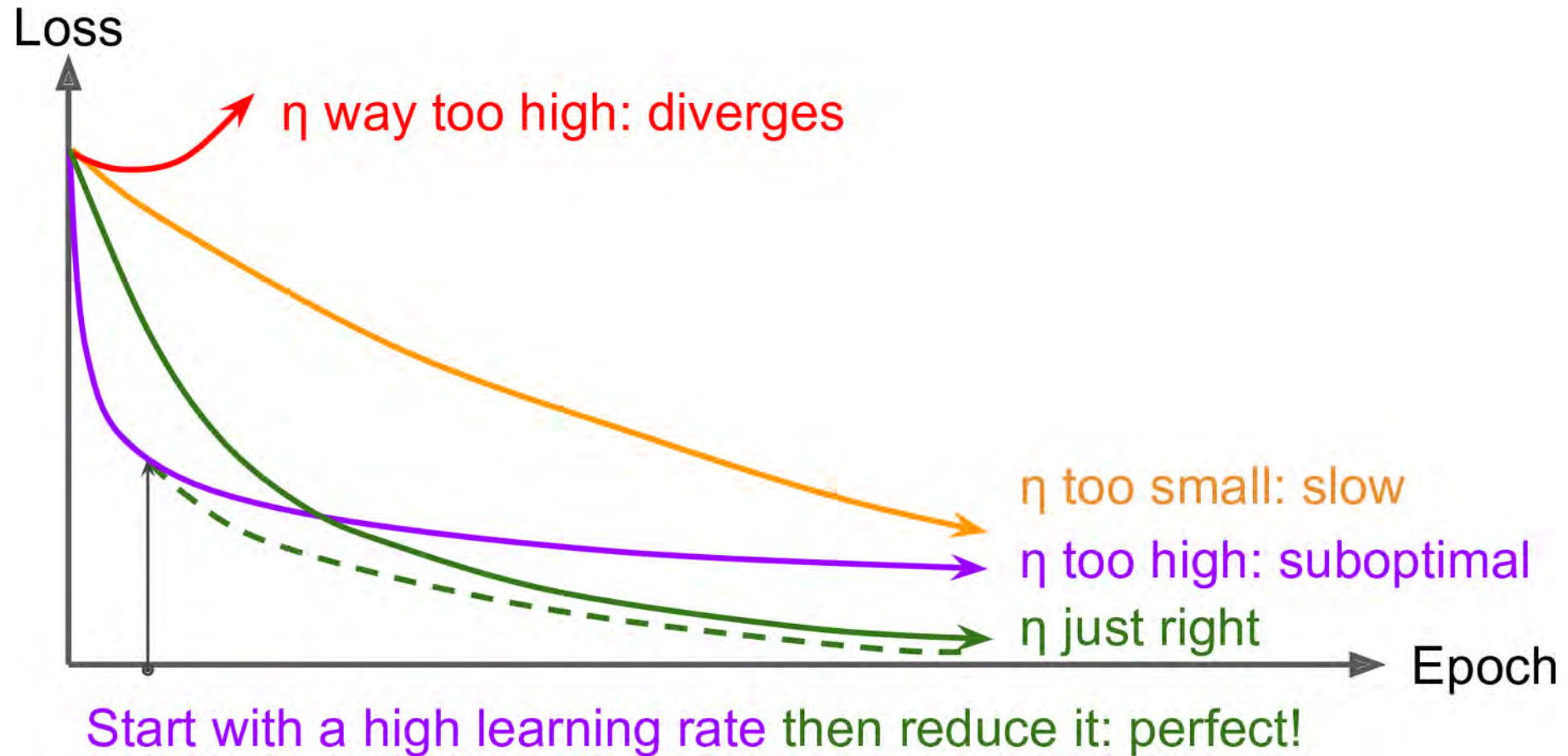


Changing the learning rates for a robot arm.

Source: Matt Henderson (2021), [Twitter post](#)



# Learning rate schedule



Learning curves for various learning rates  $\eta$

In training the learning rate may be tweaked manually.

# We need non-zero derivatives

This is why can't use accuracy as the loss function for classification.

Also why we can have the *dead ReLU* problem.

Neural Networks Part 5: ArgMax and SoftMax



# Lecture Outline

- Dense Layers in Matrices
- Optimisation
- **Loss and derivatives**



# Example: linear regression

$$\hat{y}(x) = wx + b$$

For some observation  $\{x_i, y_i\}$ , the (MSE) loss is

$$\text{Loss}_i = (\hat{y}(x_i) - y_i)^2$$

For a batch of the first  $n$  observations the loss is

$$\text{Loss}_{1:n} = \frac{1}{n} \sum_{i=1}^n (\hat{y}(x_i) - y_i)^2$$



# Derivatives

Since  $\hat{y}(x) = wx + b$ ,

$$\frac{\partial \hat{y}(x)}{\partial w} = x \text{ and } \frac{\partial \hat{y}(x)}{\partial b} = 1.$$

As  $\text{Loss}_i = (\hat{y}(x_i) - y_i)^2$ , we know

$$\frac{\partial \text{Loss}_i}{\partial \hat{y}(x_i)} = 2(\hat{y}(x_i) - y_i).$$



# Chain rule

$$\frac{\partial \text{Loss}_i}{\partial \hat{y}(x_i)} = 2(\hat{y}(x_i) - y_i), \quad \frac{\partial \hat{y}(x)}{\partial w} = x, \quad \text{and} \quad \frac{\partial \hat{y}(x)}{\partial b} = 1.$$

Putting this together, we have

$$\frac{\partial \text{Loss}_i}{\partial w} = \frac{\partial \text{Loss}_i}{\partial \hat{y}(x_i)} \times \frac{\partial \hat{y}(x_i)}{\partial w} = 2(\hat{y}(x_i) - y_i) x_i$$

and

$$\frac{\partial \text{Loss}_i}{\partial b} = \frac{\partial \text{Loss}_i}{\partial \hat{y}(x_i)} \times \frac{\partial \hat{y}(x_i)}{\partial b} = 2(\hat{y}(x_i) - y_i).$$





# Stochastic gradient descent (SGD)

Start with  $\boldsymbol{\theta}_0 = (w, b)^\top = (0, 0)^\top$ .

Randomly pick  $i = 5$ , say  $x_i = 5$  and  $y_i = 5$ .

$$\hat{y}(x_i) = 0 \times 5 + 0 = 0 \Rightarrow \text{Loss}_i = (0 - 5)^2 = 25.$$

The partial derivatives are

$$\begin{aligned} \frac{\partial \text{Loss}_i}{\partial w} &= 2(\hat{y}(x_i) - y_i) x_i = 2 \cdot (0 - 5) \cdot 5 = -50, \text{ and} \\ \frac{\partial \text{Loss}_i}{\partial b} &= 2(0 - 5) = -10. \end{aligned}$$

The gradient is  $\nabla \text{Loss}_i = (-50, -10)^\top$ .



# SGD, first iteration

Start with  $\boldsymbol{\theta}_0 = (w, b)^\top = (0, 0)^\top$ .

Randomly pick  $i = 5$ , say  $x_i = 5$  and  $y_i = 5$ .

The gradient is  $\nabla \text{Loss}_i = (-50, -10)^\top$ .

Use learning rate  $\eta = 0.01$  to update

$$\begin{aligned}\boldsymbol{\theta}_1 &= \boldsymbol{\theta}_0 - \eta \nabla \text{Loss}_i \\ &= \begin{pmatrix} 0 \\ 0 \end{pmatrix} - 0.01 \begin{pmatrix} -50 \\ -10 \end{pmatrix} \\ &= \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix}.\end{aligned}$$



# SGD, second iteration

Start with  $\theta_1 = (w, b)^\top = (0.5, 0.1)^\top$ .

Randomly pick  $i = 9$ , say  $x_i = 9$  and  $y_i = 17$ .

The gradient is  $\nabla \text{Loss}_i = (-223.2, -24.8)^\top$ .

Use learning rate  $\eta = 0.01$  to update

$$\begin{aligned}\theta_2 &= \theta_1 - \eta \nabla \text{Loss}_i \\ &= \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix} - 0.01 \begin{pmatrix} -223.2 \\ -24.8 \end{pmatrix} \\ &= \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix} + \begin{pmatrix} 2.232 \\ 0.248 \end{pmatrix} = \begin{pmatrix} 2.732 \\ 0.348 \end{pmatrix}.\end{aligned}$$



# Batch gradient descent (BGD)

For the first  $n$  observations  $\text{Loss}_{1:n} = \frac{1}{n} \sum_{i=1}^n \text{Loss}_i$  so

$$\begin{aligned} \frac{\partial \text{Loss}_{1:n}}{\partial w} &= \frac{1}{n} \sum_{i=1}^n \frac{\partial \text{Loss}_i}{\partial w} = \frac{1}{n} \sum_{i=1}^n \frac{\partial \text{Loss}_i}{\hat{y}(x_i)} \frac{\partial \hat{y}(x_i)}{\partial w} \\ &= \frac{1}{n} \sum_{i=1}^n 2(\hat{y}(x_i) - y_i) x_i. \end{aligned}$$

$$\begin{aligned} \frac{\partial \text{Loss}_{1:n}}{\partial b} &= \frac{1}{n} \sum_{i=1}^n \frac{\partial \text{Loss}_i}{\partial b} = \frac{1}{n} \sum_{i=1}^n \frac{\partial \text{Loss}_i}{\hat{y}(x_i)} \frac{\partial \hat{y}(x_i)}{\partial b} \\ &= \frac{1}{n} \sum_{i=1}^n 2(\hat{y}(x_i) - y_i). \end{aligned}$$



# BGD, first iteration ( $\theta_0 = \mathbf{0}$ )

	x	y	y_hat	loss	dL/dw	dL/db
0	1	0.99	0	0.98	-1.98	-1.98
1	2	3.00	0	9.02	-12.02	-6.01
2	3	5.01	0	25.15	-30.09	-10.03

So  $\nabla \text{Loss}_{1:3}$  is

```
1 nabla = np.array([df["dL/dw"].mean(), df["dL/db"].mean()])
2 nabla
```

```
array([-14.69, -6.  ])
```

so with  $\eta = 0.1$  then  $\theta_1$  becomes

```
1 theta_1 = theta_0 - 0.1 * nabla
2 theta_1
```

```
array([1.47, 0.6 ])
```



# BGD, second iteration

	x	y	y_hat	loss	dL/dw	dL/db
0	1	0.99	2.07	1.17	2.16	2.16
1	2	3.00	3.54	0.29	2.14	1.07
2	3	5.01	5.01	0.00	-0.04	-0.01

So  $\nabla \text{Loss}_{1:3}$  is

```
1 nabla = np.array([df["dL/dw"].mean(), df["dL/db"].mean()])
2 nabla
```

```
array([1.42, 1.07])
```

so with  $\eta = 0.1$  then  $\theta_2$  becomes

```
1 theta_2 = theta_1 - 0.1 * nabla
2 theta_2
```

```
array([1.33, 0.49])
```



# Glossary

- batches, batch size
- gradient-based learning, hill-climbing
- metrics
- stochastic (mini-batch) gradient descent

