Gradient descent

Autumn 2020

```
################################
##                            ##
##  Deep Learning in Python   ##
##                            ##
################################
```

`§1 Introduction to Deep Learning in Python`

`§1.2 Optimizing a neural network with backward propagation`

`§1.2.2 Gradient descent`

1. How does gradient descent work?

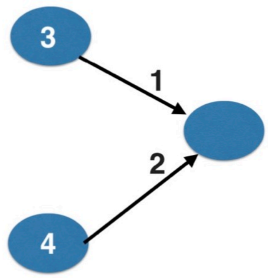
- If the slope is positive:
  - going opposite the slope means moving to lower numbers
  - subtract the slope from the current value
  - too big a step might lead astray
- The solution to avoid leading astray is using the learning rate:
  - update each weight by subtracting *learning rate* × *slope*

2. What are the steps of the slope calculation?

- To calculate the slope for a weight, need to multiply:
  - the slope of the loss function with respect to value at the node which fed into
    - ex. *the slope of the mean-squared loss function*, in this case,

      $2 \cdot (Predicted\ Value - Actual\ Value) = 2 \cdot Error$
  - the value of the node that feeds into the weight
  - the slope of the activation function with respect to the value fed into

3. Code the slope calculation and weights updating:



[1]:
```
import numpy as np

weights = np.array([1, 2])
input_data = np.array([3, 4])
target = 6
learning_rate = 0.01

preds = (weights * input_data).sum()
error = preds - target
print(error)
```

```
5
```

[2]:
```
gradient = 2 * input_data * error
gradient
```

[2]:
```
array([30, 40])
```

[3]:
```
weights_updated = weights - learning_rate * gradient

preds_updated = (weights_updated * input_data).sum()
error_updated = preds_updated - target
print(error_updated)
```

```
2.5
```

4. Practice exercises for gradient descent:

▶ Code pre-loading:

[4]:
```
import numpy as np

input_data = np.array([1, 2, 3])

weights = np.array([0, 2, 1])

target = 0
```

▶ The slope calculation practice:

[5]:

```
# Calculate the predictions: preds
preds = (weights * input_data).sum()

# Calculate the error: error
error = preds - target

# Calculate the slope: slope
slope = 2 * error * input_data

# Print the slope
print(slope)
```

```
[14 28 42]
```

▶ **The model weights improving practice:**

[6]:

```
# Set the learning rate: learning_rate
learning_rate = 0.01

# Calculate the predictions: preds
preds = (weights * input_data).sum()

# Calculate the error: error
error = preds - target

# Calculate the slope: slope
slope = 2 * input_data * error

# Update the weights: weights_updated
weights_updated = weights - learning_rate * slope

# Get updated predictions: preds_updated
preds_updated = (weights_updated * input_data).sum()

# Calculate updated error: error_updated
error_updated = preds_updated - target

# Print the original error
print(error)

# Print the updated error
print(error_updated)
```

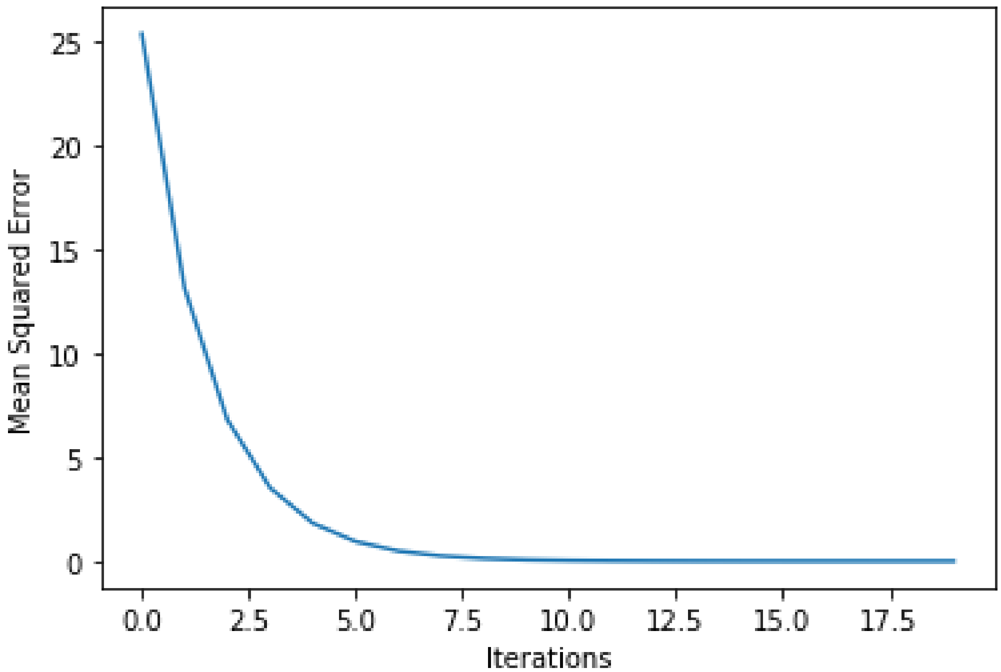
```
7
5.04
```

▶ **Code pre-loading:**

[7]:
```python
import matplotlib.pyplot as plt


def get_error(input_data, target, weights):
    preds = (weights * input_data).sum()
    error = preds - target
    return (error)


def get_slope(input_data, target, weights):
    error = get_error(input_data, target, weights)
    slope = 2 * input_data * error
    return (slope)


def get_mse(input_data, target, weights):
    errors = get_error(input_data, target, weights)
    mse = np.mean(errors**2)
    return (mse)
```

▶ **Weights multiple updates practice:**

[8]:
```python
n_updates = 20
mse_hist = []

# Iterate over the number of updates
for i in range(n_updates):
    # Calculate the slope: slope
    slope = get_slope(input_data, target, weights)

    # Update the weights: weights
    weights = weights - 0.01 * slope

    # Calculate mse with new weights: mse
    mse = get_mse(input_data, target, weights)

    # Append the mse to mse_hist
    mse_hist.append(mse)

# Plot the mse history
plt.plot(mse_hist)
plt.xlabel('Iterations')
plt.ylabel('Mean Squared Error')
plt.show()
```


Mean Squared Error
Iterations
25
20
15
10
5
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5