Introduction

Overview

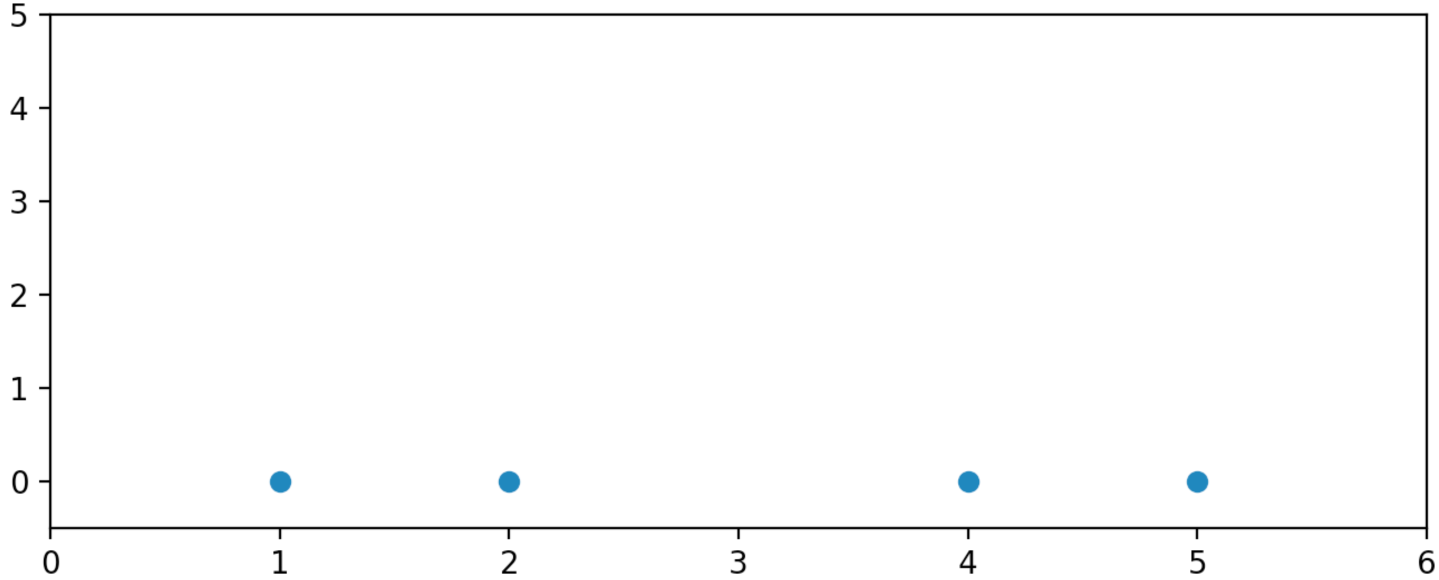
Ideas introduced

- Loss function
- Optimizing a loss function
- Gradient descent
- Numeric precision
- Formation of a loss function
- Formulating a loss function that is convex
- Formulate a search function
- Calculus as a tool for derivatives
- Connection to deep learning

DataScience@SMU

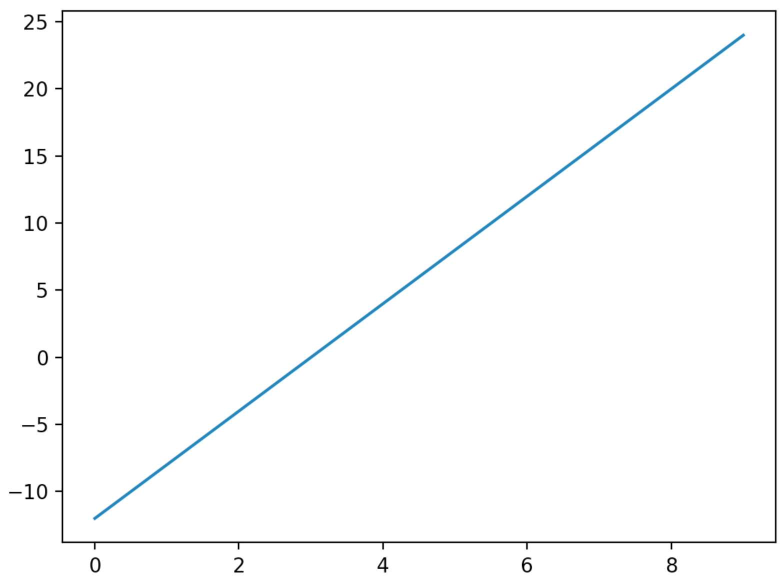
Formulating a Loss Function

Loss Functions




```
import matplotlib.pyplot as plt
import numpy as np
values = [1,2,4,5]

plt.figure(figsize=(8,3))
plt.scatter(values,np.zeros(4))
plt.ylim(-.5, 5)
plt.xlim(0, 6)
plt.show()
```

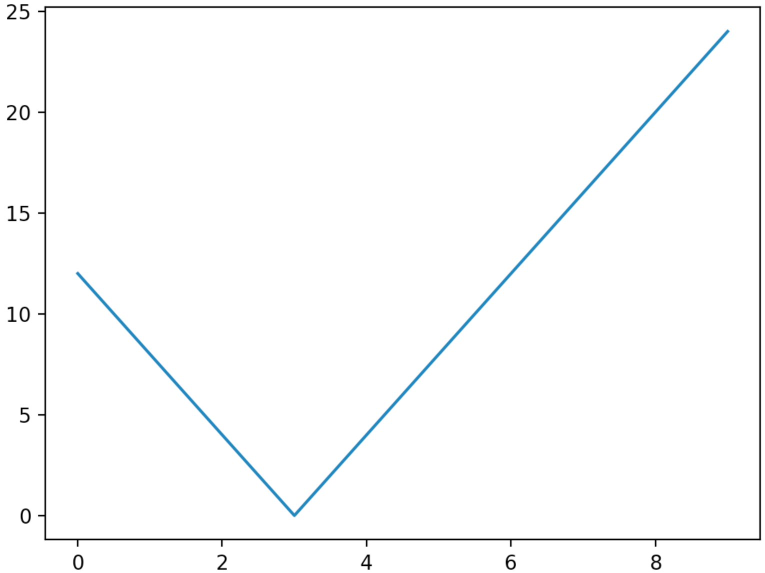
Simple Problem: Find the Best Predictor of Data (We Know It's 3)



- Loss function:
  - Represents how wrong a prediction is
- Issue:
  - 0 "wrongness" is best want positive loss values

```
import matplotlib.pyplot as plt
import numpy as np
values = [1,2,4,5]

def calc_left_right(data, middle):
  left = []
  right = []
  for x in data:
    if x < middle:
      left.append(x)
    else:
      right.append(x)
  error = (sum([middle-x for x in left]) +
          sum([middle-x for x in right]))
  return error

plt.plot([calc_left_right(values, x)
          for x in range(10)])
plt.show()
```

A Properly Formatted Loss Function



Minimum is optimal

```
import matplotlib.pyplot as plt
import numpy as np
values = [1,2,4,5]

# Utilize a minimization technique
def calc_left_right(data, middle):
  left = []
  right = []
  for x in data:
    if x < middle:
      left.append(x)
    else:
      right.append(x)
  return abs(sum([middle-x for x in left]) +
             sum([middle-x for x in right]))

plt.plot([calc_left_right(values, x)
          for x in range(10)])
plt.show()
```

DataScience@SMU

Search Function

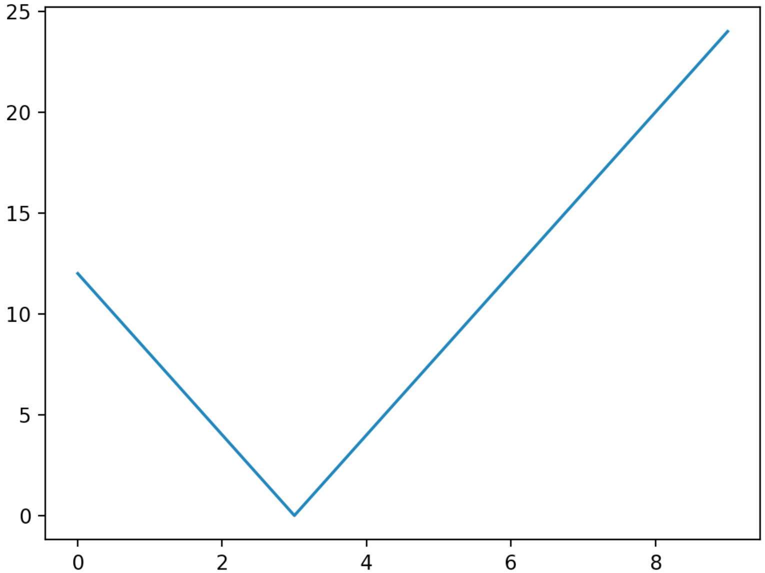
Optimization

Searching for the optimal loss value

- **Components to solve this problem**
  - Model space: way to predict outputs
  - Loss function (“objective”): measurement of how wrong model is on data
  - Searcher: how to find best model
  - Goal: find model with smallest loss

- Previous model of averages
  - Model space: average
  - Loss: how incorrect average
  - Searcher: by inspection (graphed the function, found min)

A Loss Function (Revisited)



Minimum is optimal

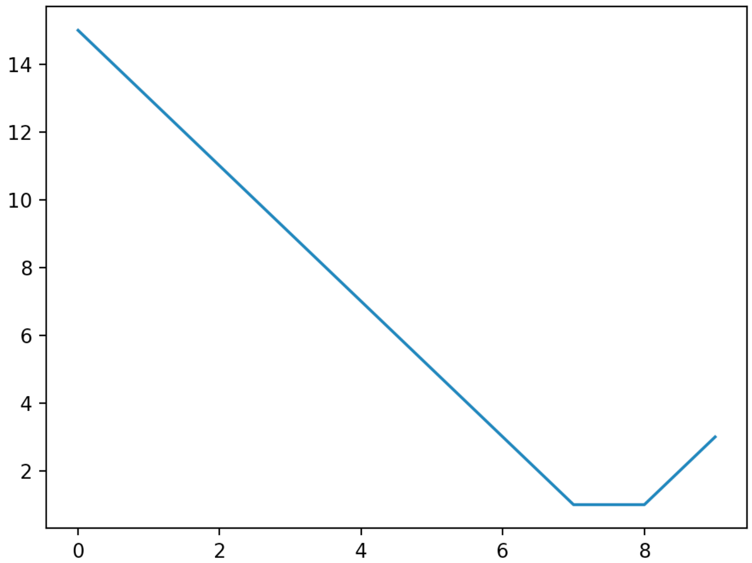
```
import matplotlib.pyplot as plt
import numpy as np
values = [1,2,4,5]

# Utilize a minimization technique
def calc_left_right(data, middle):
  left = []
  right = []
  for x in data:
    if x < middle:
      left.append(x)
    else:
      right.append(x)
  return abs(sum([middle-x for x in left]) +
             sum([middle-x for x in right]))

plt.plot([calc_left_right(values, x)
          for x in range(10)])
plt.show()
```

Approaches

- Try a bunch of values
  - “Grid search”
  - Iterative methods
- Scientific computing issues
  - Tolerance
  - Step size

Scientific Computing Issues

Tolerance + step size




```
def find_middle(data):
  first_guess = data[0]
  while(calc_left_right(data, first_guess) > 0):
    if (calc_left_right(data, first_guess-1) >
        calc_left_right(data, first_guess)):
        first_guess = first_guess+1
    else:
      first_guess = first_guess-1
  return first_guess

find_middle([5,10])

plt.plot([calc_left_right([5,10], x)
          for x in range(10)])
plt.show()
```

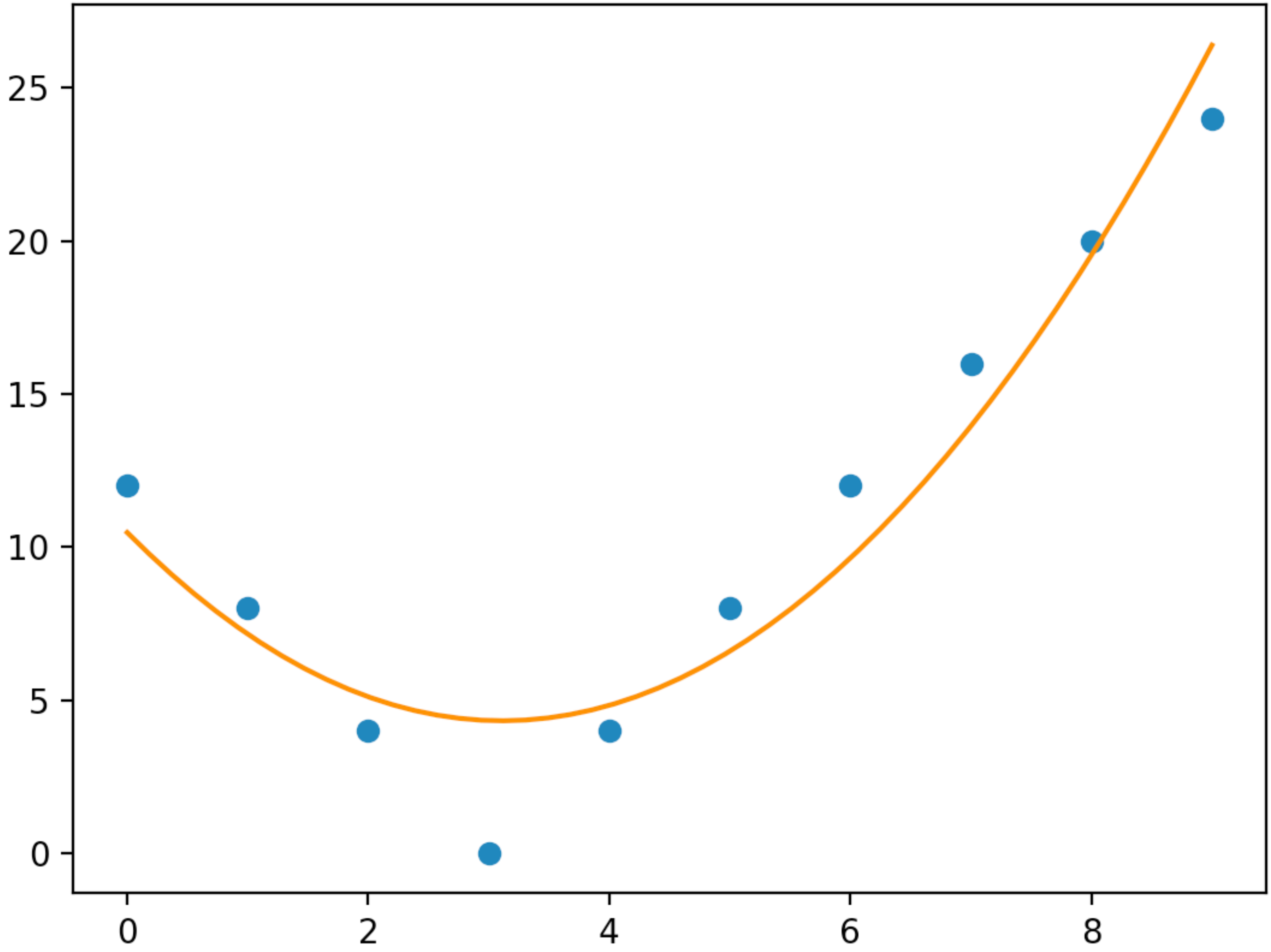
```
def find_middle(data, tol=.0001):
  first_guess = data[0]
  while(calc_left_right(data, first_guess) > tol):
    print (first_guess)
    if (calc_left_right(data, first_guess-tol) >
        calc_left_right(data, first_guess)):
        first_guess = first_guess + tol
    else:
      first_guess = first_guess - tol
  return first_guess

# Notice how many estimates it takes!
find_middle([5,10],tol=1)
find_middle([5,10],tol=.5)
find_middle([5,10],tol=.001)
```

```
7.359999999999995
7.3699999999999495
7.379999999999949
7.389999999999949
7.399999999999949
7.409999999999949
7.419999999999948
7.429999999999948
7.439999999999948
7.449999999999948
7.459999999999948
7.469999999999947
7.479999999999947
7.489999999999947
7.499999999999947
>>>
```

Gradient

```
points = np.array([(x,calc_left_right(values, x))
                   for x in range(10)])
x = points[:,0]
y = points[:,1]
z = np.polyfit(x, y, 2)
f = np.poly1d(z)
x_new = np.linspace(x[0], x[-1], 50)
y_new = f(x_new)

plt.plot(x,y,'o', x_new, y_new)
ax = plt.gca()
ax.set_axis_bgcolor((0.898, 0.898, 0.898))
fig = plt.gcf()
```

Calculus? Pictures first

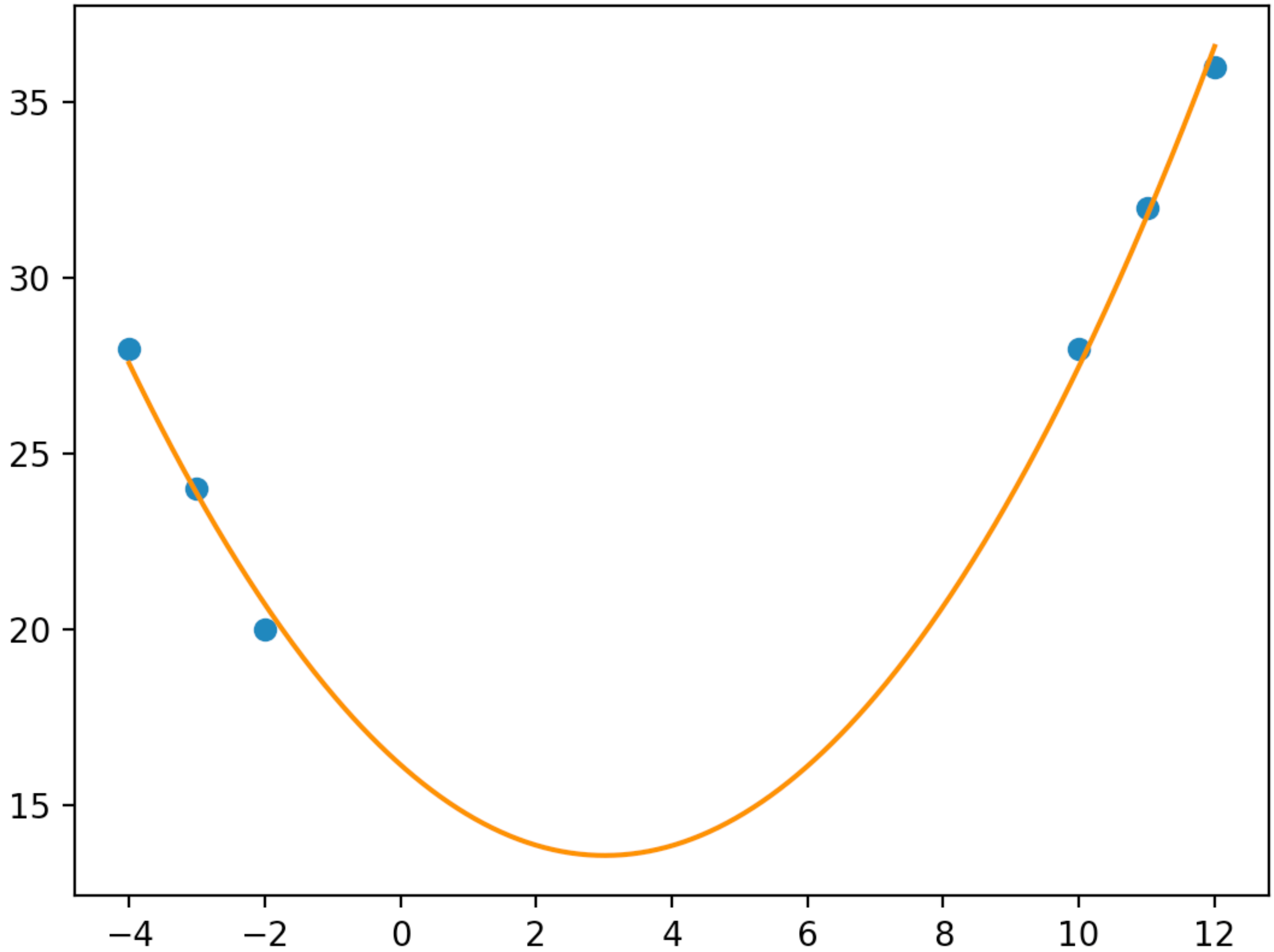


Optimizing Your Search Function

Gradient = derivative = slope

- **Idea:** Calculate gradient, move in that direction (walk through picture)
- **Issues:**
  - Step size
  - Tolerance, convergence
  - Nonconvex functions

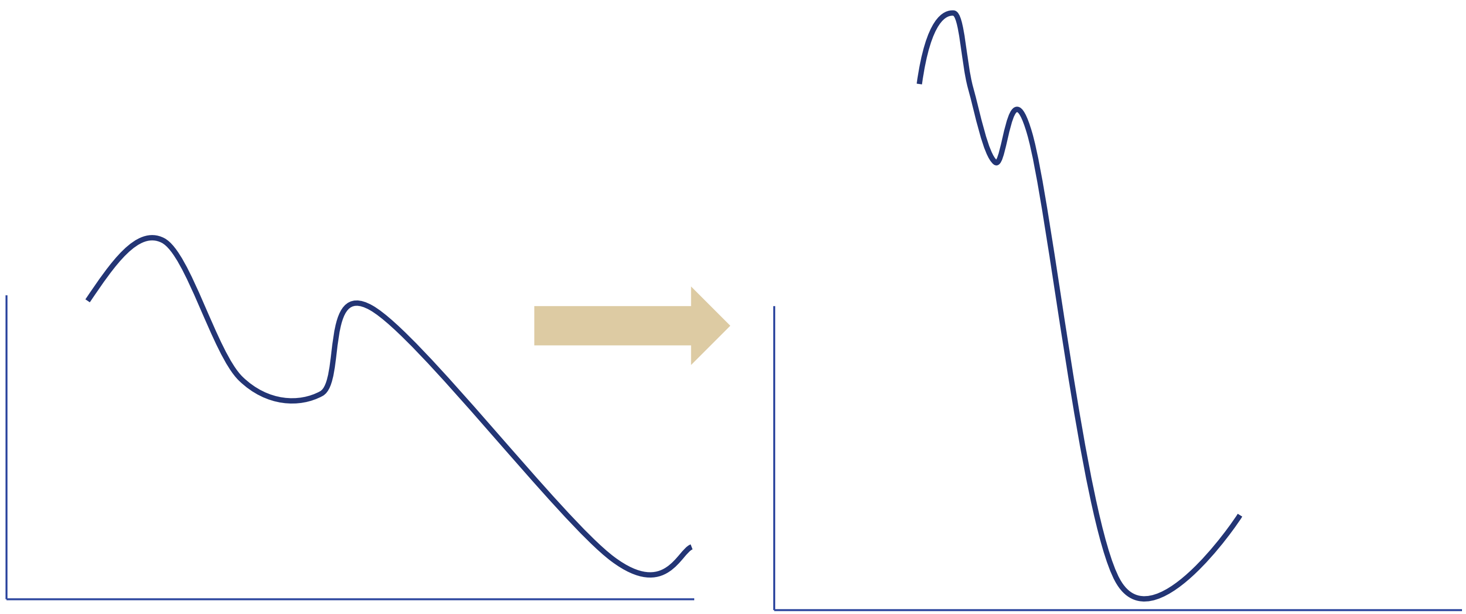
```
points = np.array([(x,calc_left_right(values, x))
                   for x in [-4,-3,-2, 10,11,12]])
# get x and y vectors
x = points[:,0]
y = points[:,1]
# calculate polynomial
z = np.polyfit(x, y, 2)
f = np.poly1d(z)
# calculate new x's and y's
x_new = np.linspace(x[0], x[-1], 500)
y_new = f(x_new)
plt.plot(x,y,'o', x_new, y_new)
ax = plt.gca()
ax.set_axis_bgcolor((0.898, 0.898, 0.898))
fig = plt.gcf()
```


35
30
25
20
15
−4
−2
0
2
4
6
8
10
12

DataScience@SMU

Connection to Deep Learning

Connection to Deep Learning



Transform the loss function to allow for faster convergence. Notice how relationships between points are preserved but steepness is increased.

Gradient Descent in DL

Optimization hard: many parameters, too many models

Approximate gradient descent

Large computation, but parallelizable!

DataScience@SMU