

# Gradient Descent

*Wed Sept 20th, 2017*

James McInenrey

Adapted from slides by Francisco J. R. Ruiz

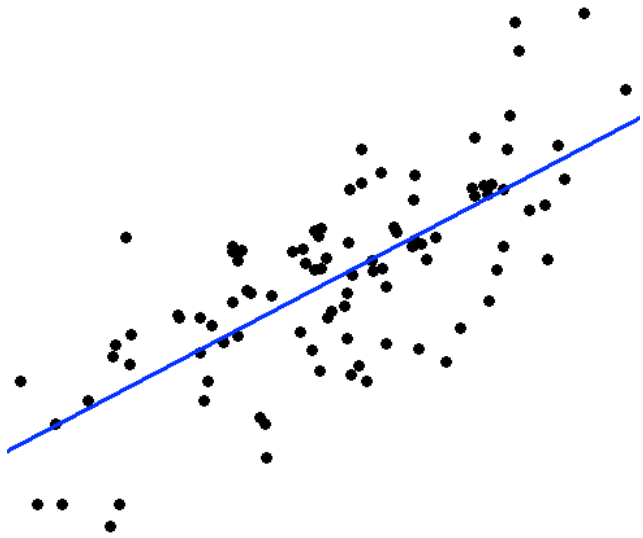
# Housekeeping

A few clarifications of and adjustments to the course schedule:

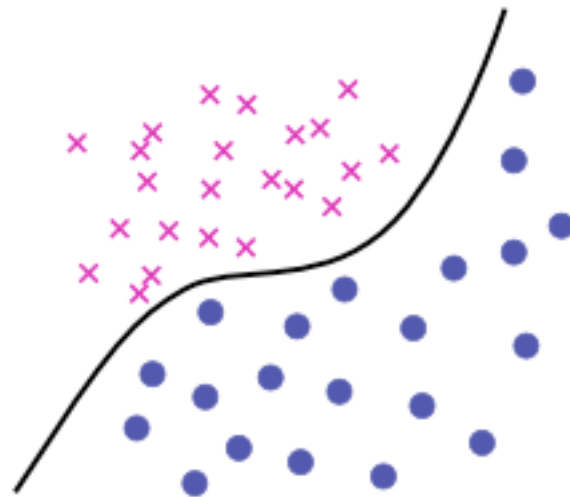
- No more breaks at the midpoint of the lecture.
- The lecture will run to 3:55pm. We encourage as many questions as you can muster at the end. If there are no more questions then I will expand on the day's topic until 3:55pm.
- We start at 2:40pm. Schedule glitches are resolved and will not happen again.
- Sometimes I ask TAs go through homework at the start of the lecture. They know what you got wrong most often and it helps to get to know each other.

# Introduction

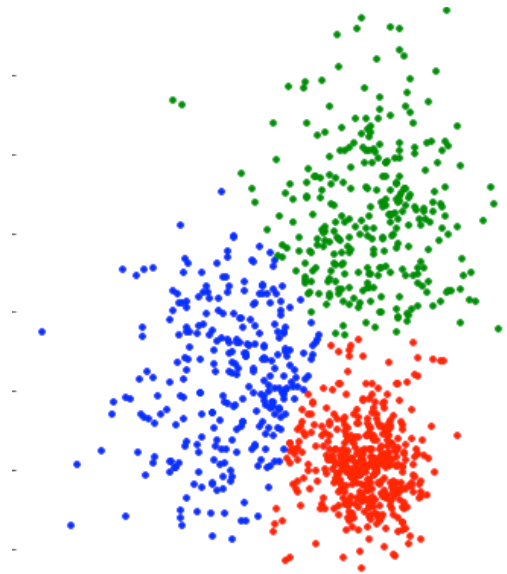
- In the coming weeks, we will explore parametric methods for the following tasks.



Regression



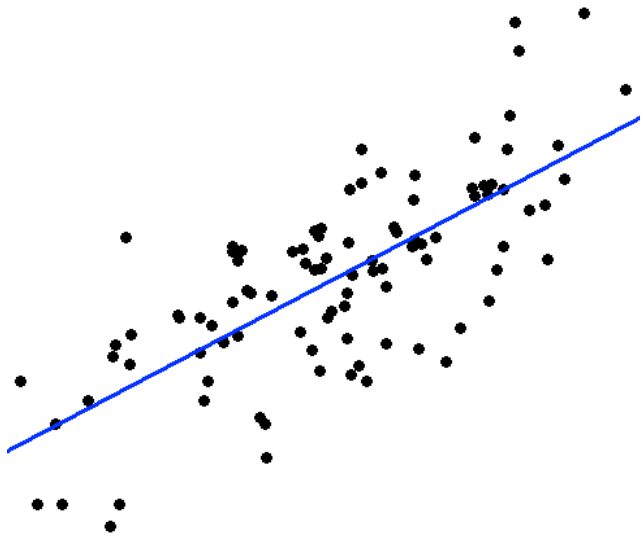
Classification



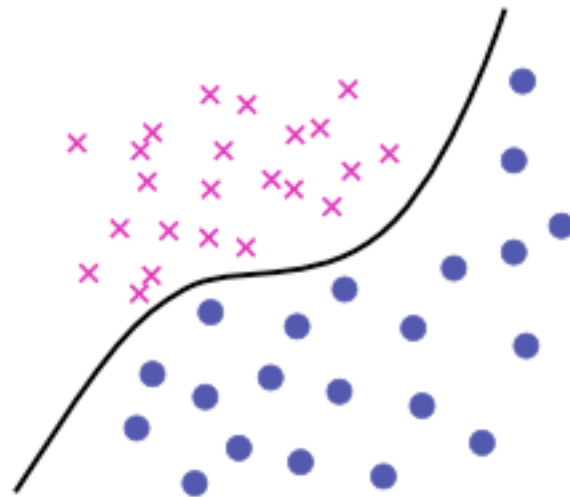
Clustering

# Introduction

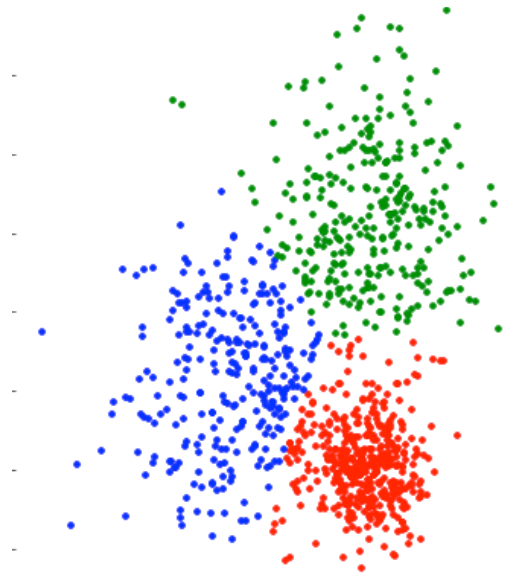
- In the coming weeks, we will explore parametric methods for the following tasks.
- All can be addressed using *maximum likelihood estimation*, which is, at heart, an optimization problem.



Regression



Classification



Clustering

# Optimization

- *Optimization* appears in many machine learning algorithms
  - Supervised and unsupervised learning
  - Basic and advanced methods

# Optimization

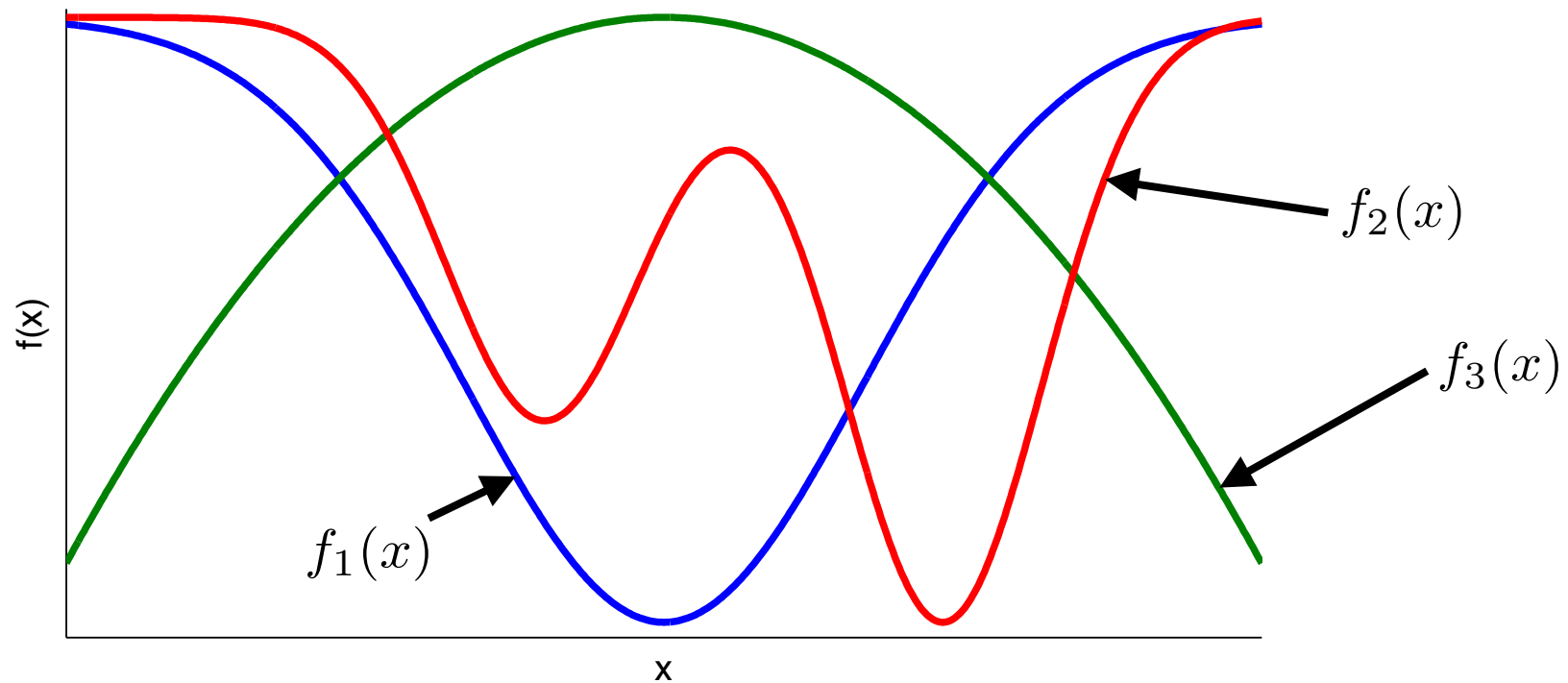
- ***Optimization:*** Minimize an objective function

$$\mathbf{x}^* = \min_{\mathbf{x}} f(\mathbf{x})$$

- **Example:** Naive Bayes classification
  - The function is the sum of squared errors of each class data point and feature dimension. Can anybody explain why?
  - Find the means and variances that minimize the function.

# Optimization

- The function can have one, multiple or no **local optima**



# Optimization

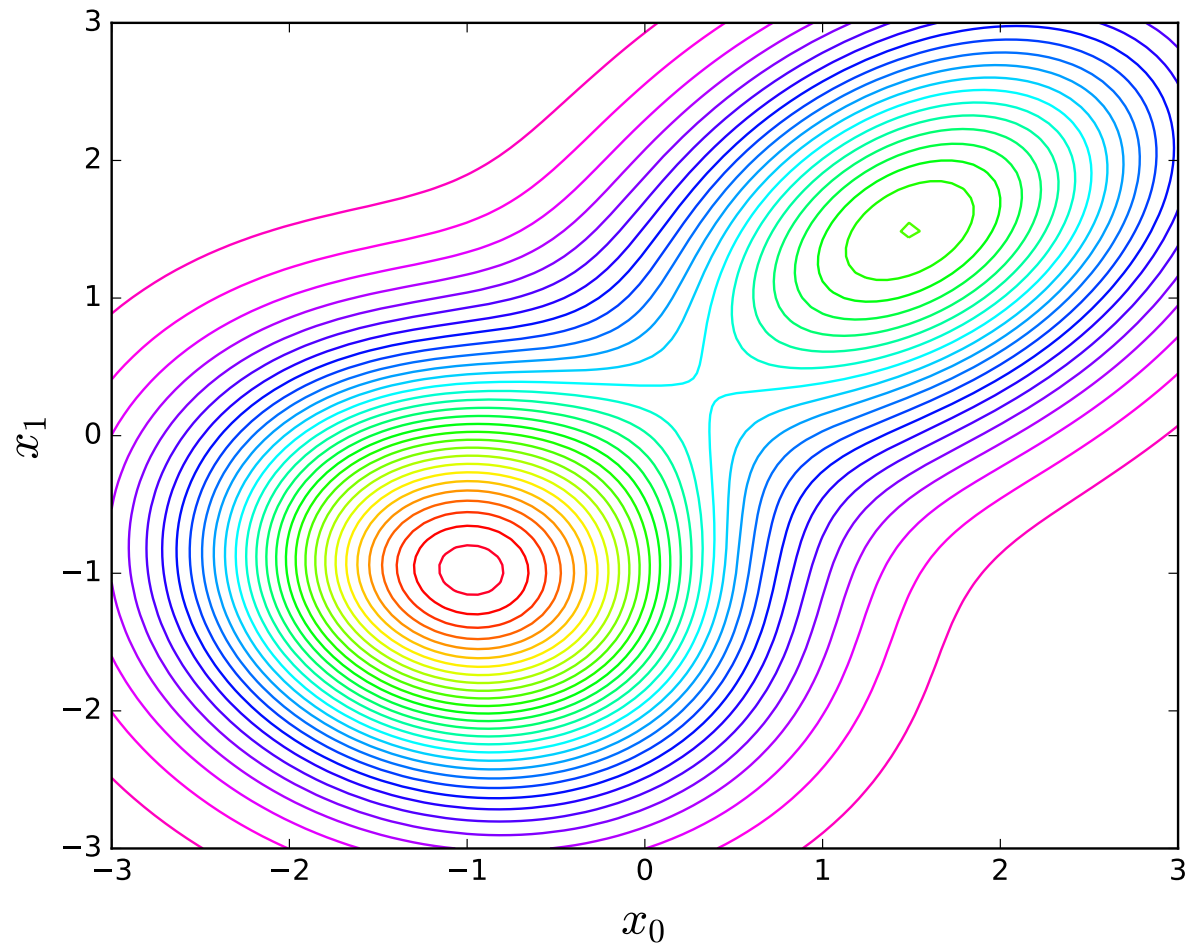
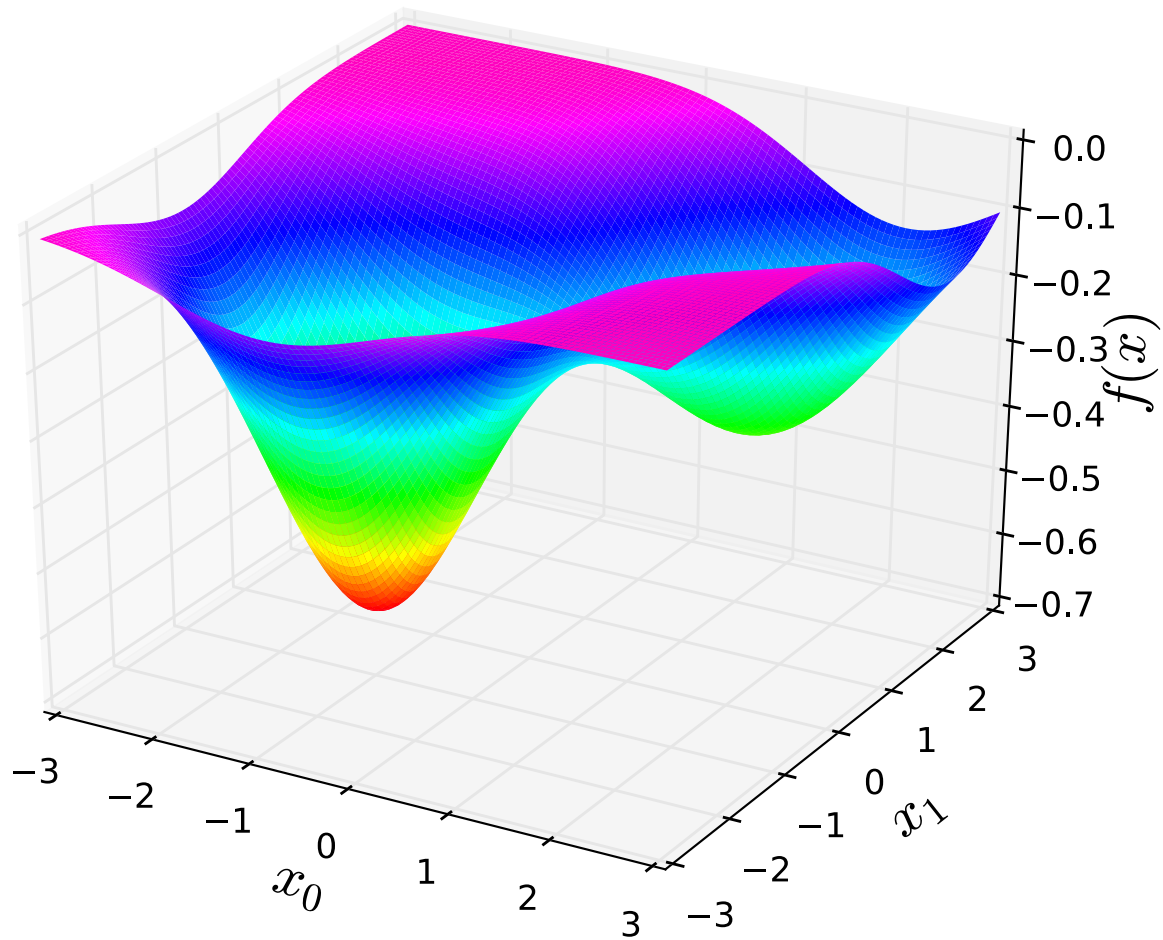
- In some cases, we can find the optima analytically
  - Examples: naive Bayes classification, linear regression
- In most practical cases, we need an iterative algorithm
  - Examples: logistic regression, neural networks



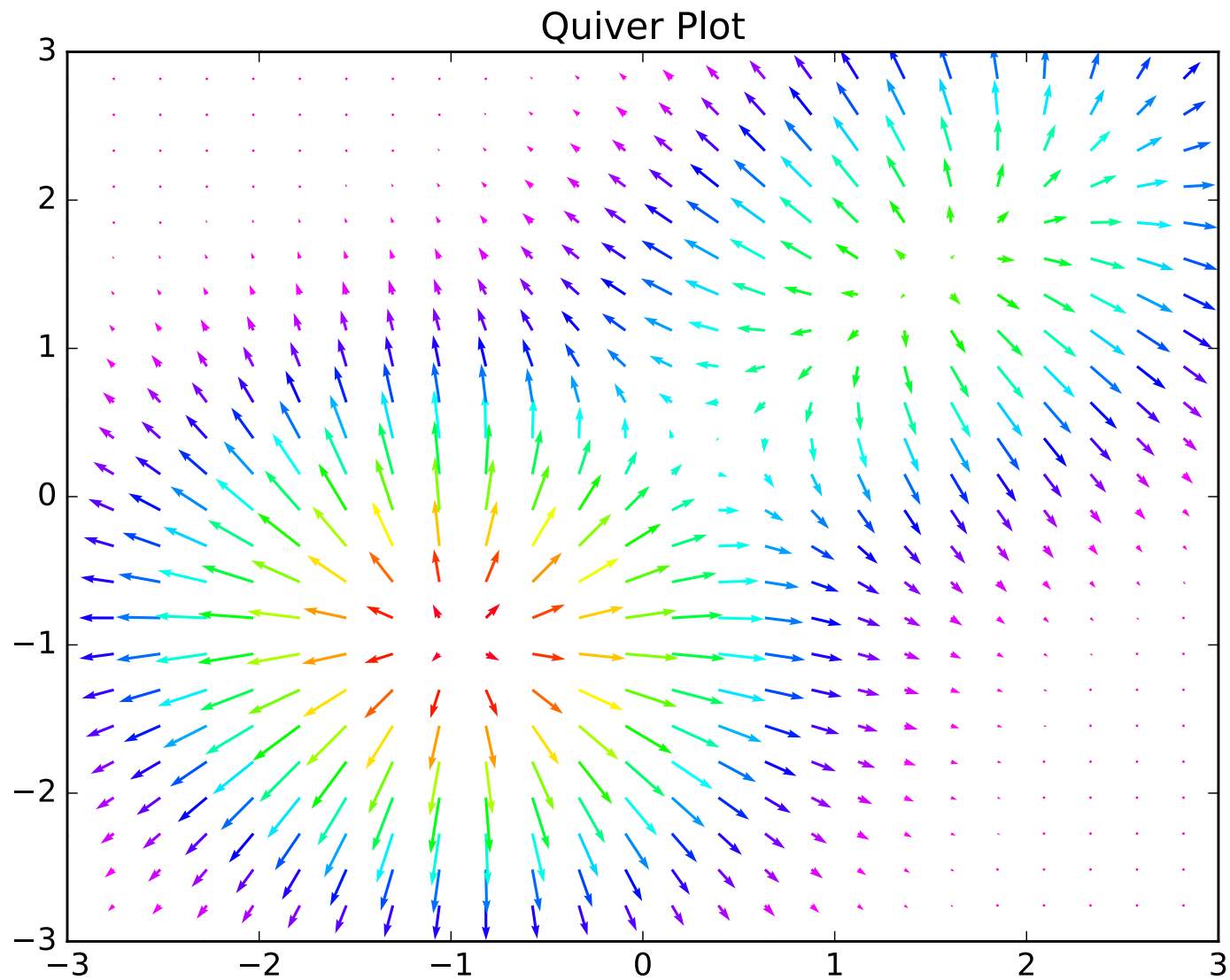
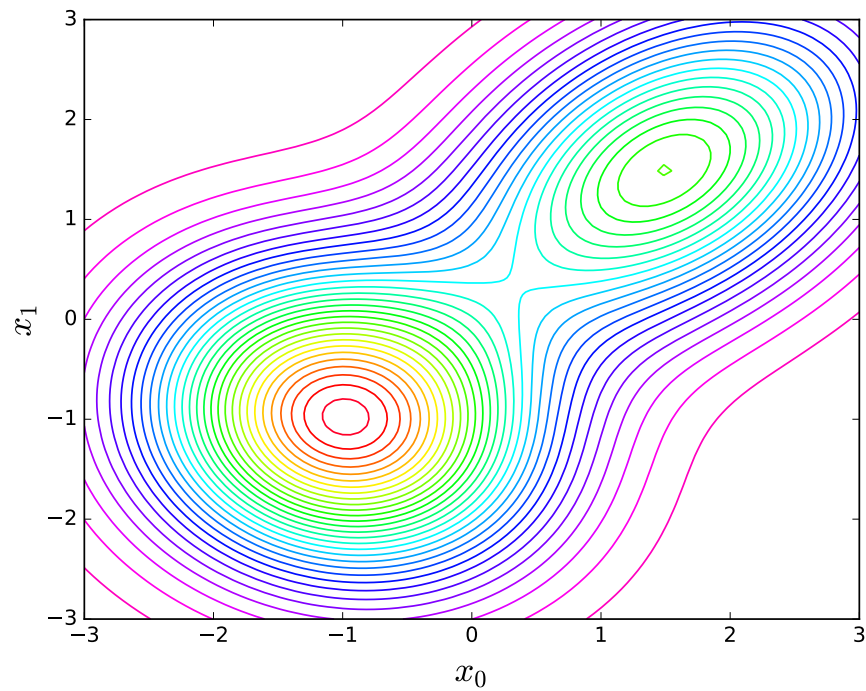
# Gradient Descent

- Gradient descent is an algorithm for optimization
  - Simple to implement
  - Intuitive interpretation
  - Works also in high dimensions

# Gradient

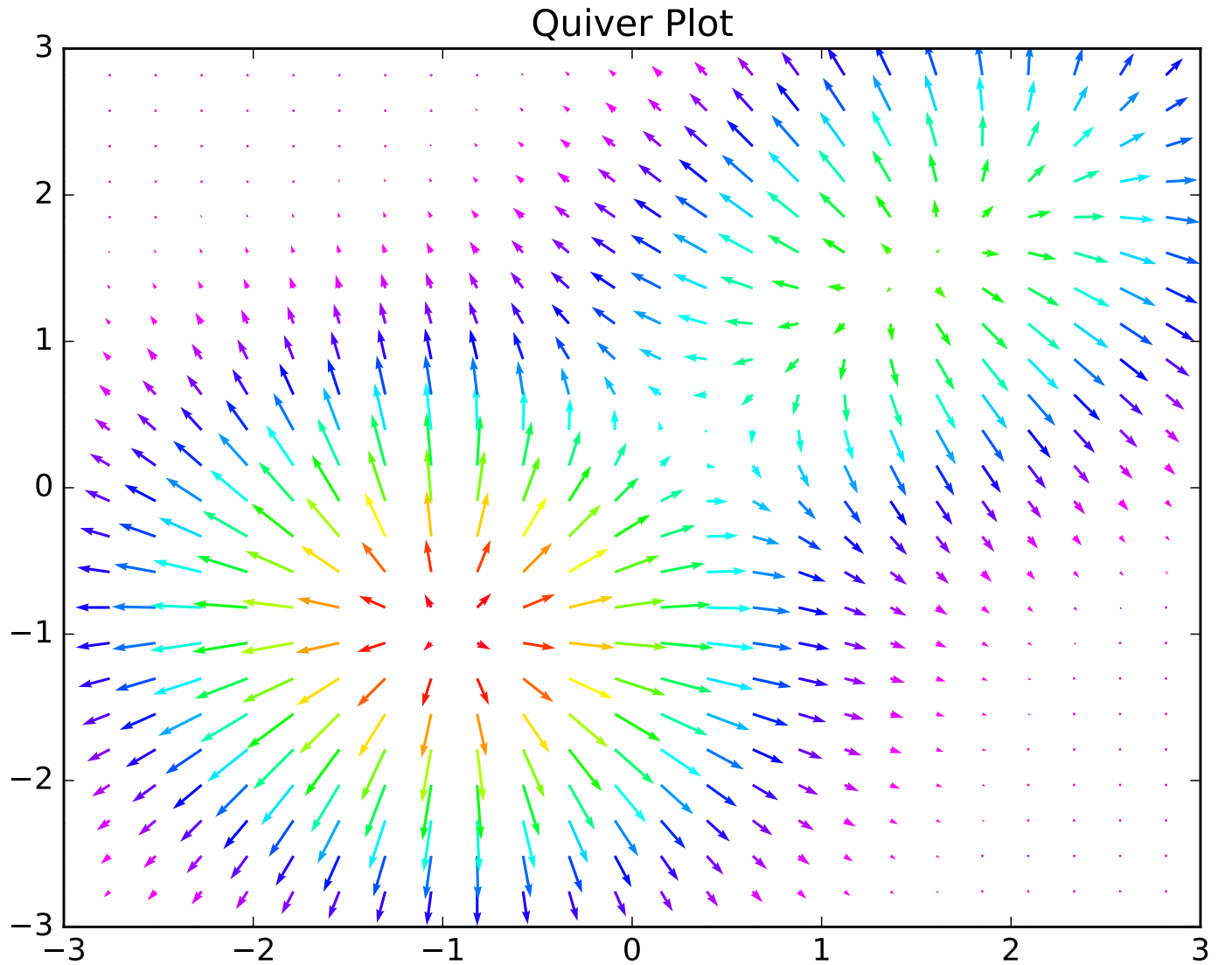


# Gradient



# Gradient

The gradient gives  
the direction of  
steepest ascent

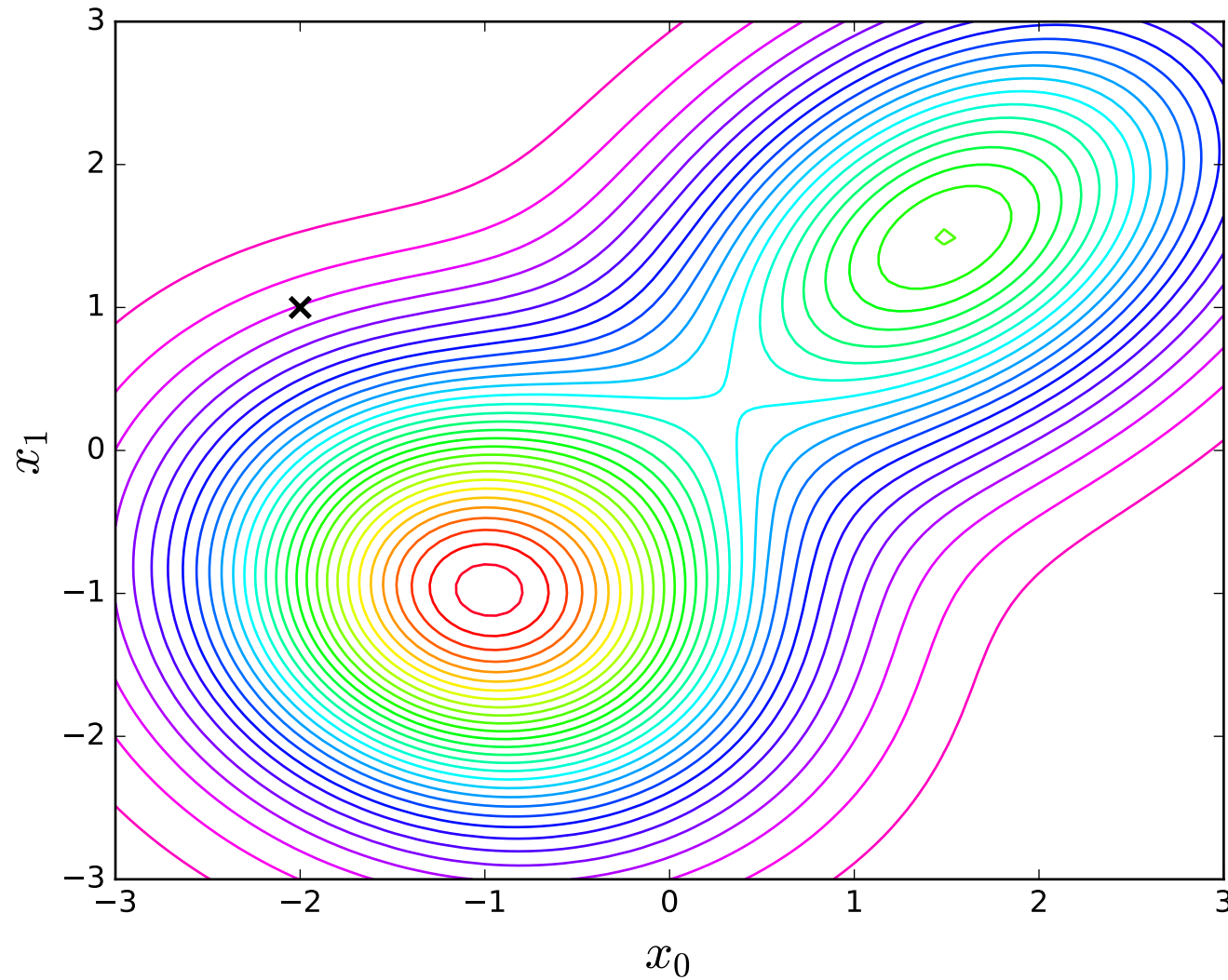


# Gradient Descent: Algorithm

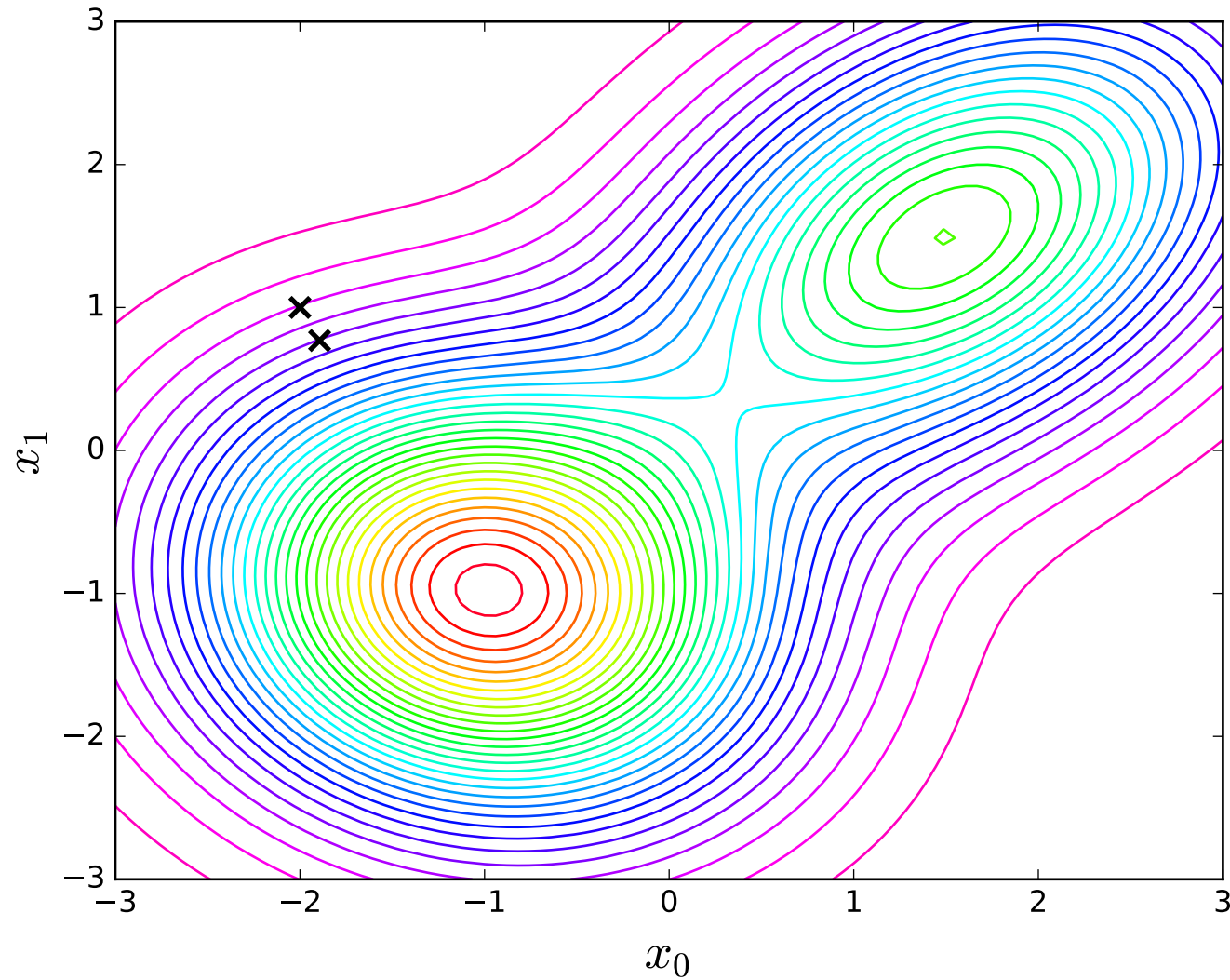
- **Algorithm:**

1. Set  $\mathbf{x}$  to initial guess
  2. Refine the current value of  $\mathbf{x}$
  3. If not converged, go back to step 2
- In step 2, follow the negative gradient

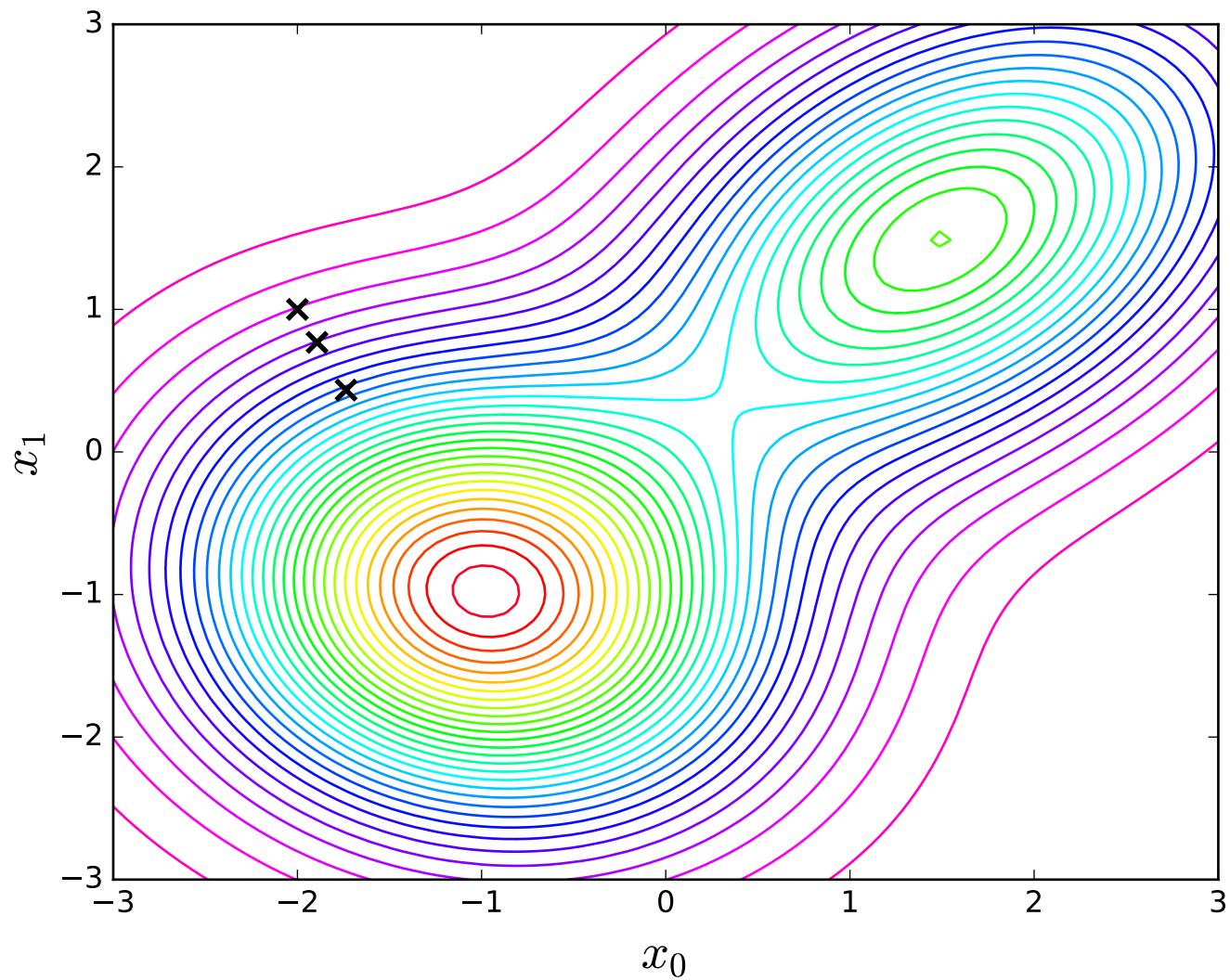
# Gradient Descent: Algorithm



# Gradient Descent: Algorithm

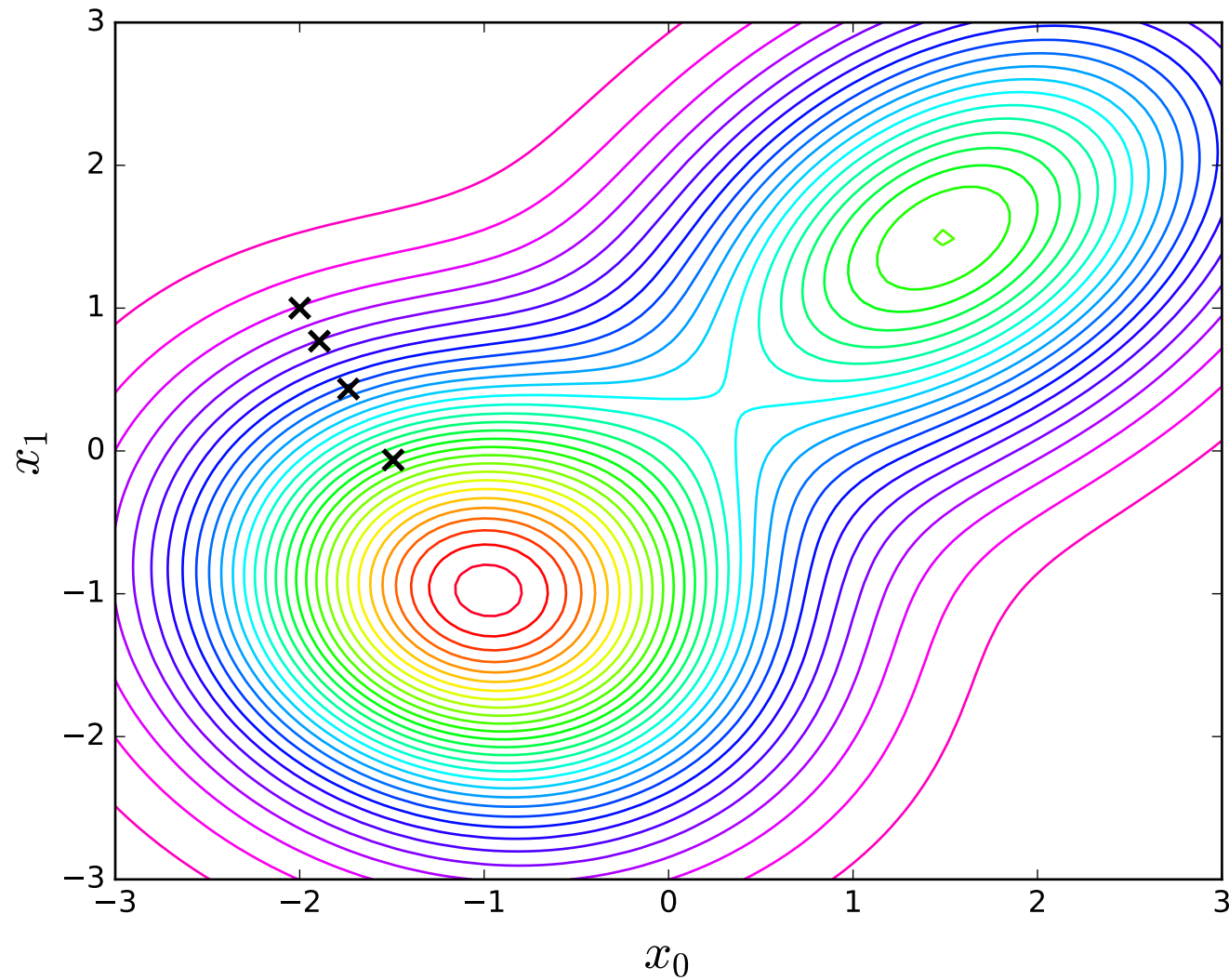


# Gradient Descent: Algorithm

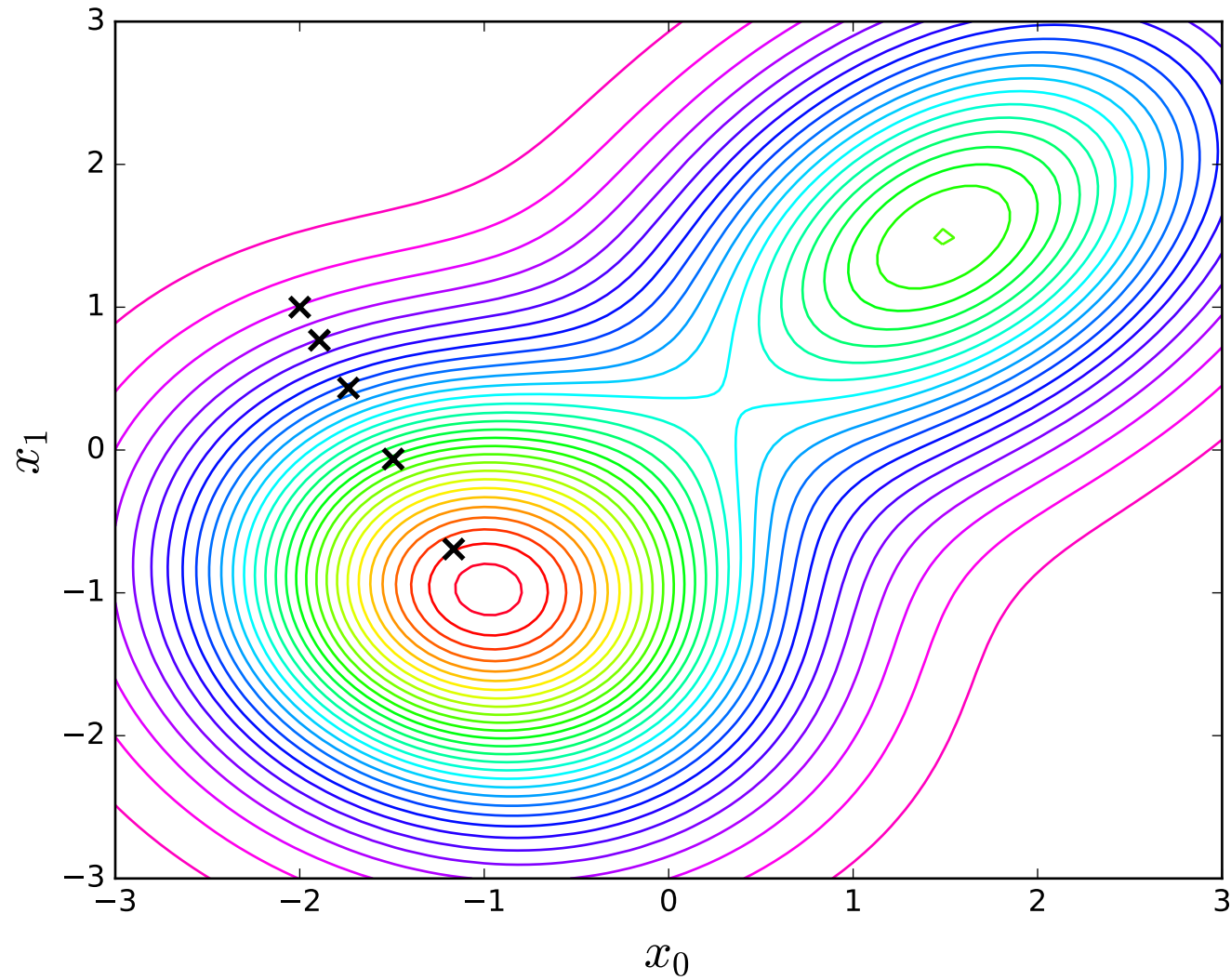




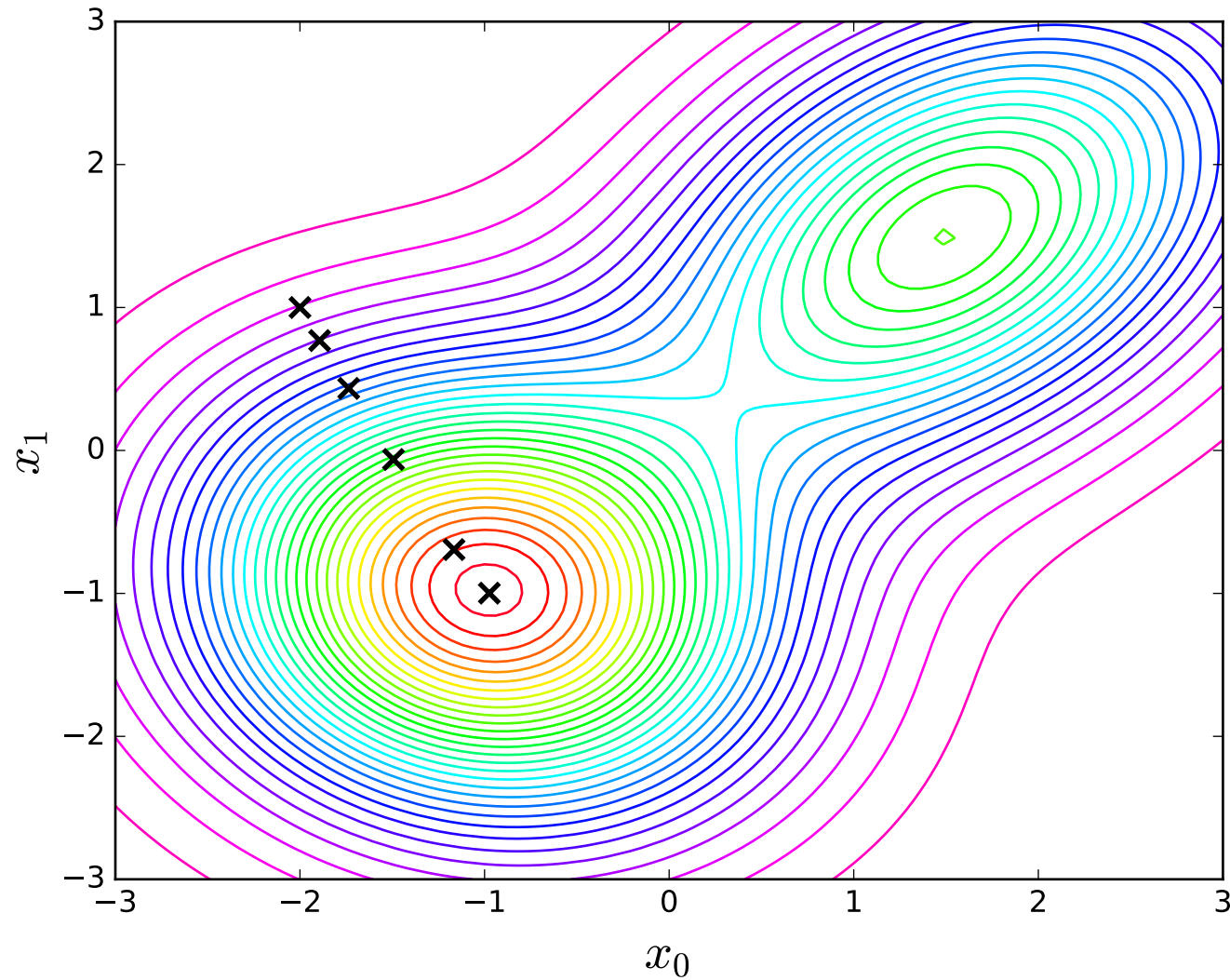
# Gradient Descent: Algorithm



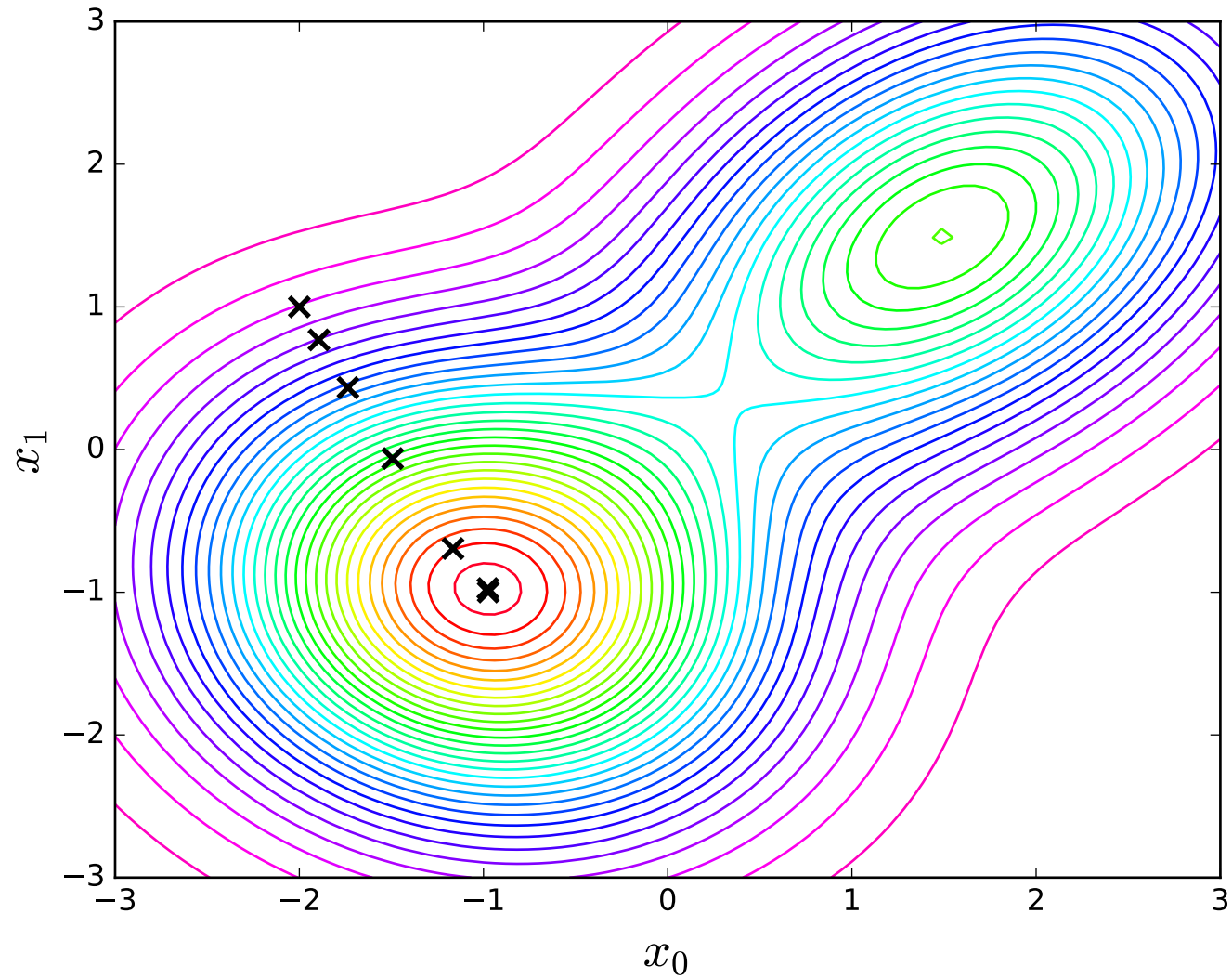
# Gradient Descent: Algorithm



# Gradient Descent: Algorithm



# Gradient Descent: Algorithm



# Gradient Descent: Algorithm

- Each update is:

$$\mathbf{x}^{\text{new}} = \mathbf{x}^{\text{old}} + \rho \cdot \nabla_{\mathbf{x}} f(\mathbf{x}^{\text{old}}) \quad (\text{Maximization})$$

$$\mathbf{x}^{\text{new}} = \mathbf{x}^{\text{old}} - \rho \cdot \nabla_{\mathbf{x}} f(\mathbf{x}^{\text{old}}) \quad (\text{Minimization})$$

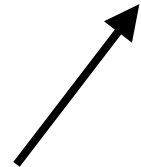
# Gradient Descent: Algorithm

- Each update is:


$$\mathbf{x}^{\text{new}} = \mathbf{x}^{\text{old}} + \rho \cdot \nabla_{\mathbf{x}} f(\mathbf{x}^{\text{old}}) \quad (\text{Maximization})$$

$$\mathbf{x}^{\text{new}} = \mathbf{x}^{\text{old}} - \rho \cdot \nabla_{\mathbf{x}} f(\mathbf{x}^{\text{old}}) \quad (\text{Minimization})$$

Current value of  $\mathbf{x}$



Step size



Gradient



# Convergence

- We stop the algorithm:
  - When  $\mathbf{x}$  does not change much
  - When the gradient is small
  - After a fixed number of iterations

# Limitations

- Not guaranteed to find the global optimum
- Choosing the step size is a nuisance
- Only takes into account gradient information
- Each iteration requires an entire pass through the data set



# Limitations

- Not guaranteed to find the global optimum  
Do multiple random restarts and pick best run [higher dimensions make this less effective]
- Choosing the step size is a nuisance
- Only takes into account gradient information
- Each iteration requires an entire pass through the data set

# Limitations

- Not guaranteed to find the global optimum
  - Do multiple random restarts and pick best run [higher dimensions make this less effective]
- Choosing the step size is a nuisance
  - RMSprop, AdaGrad, Adam [have their own hyperparameters]
- Only takes into account gradient information
- Each iteration requires an entire pass through the data set

# Limitations

- Not guaranteed to find the global optimum
  - Do multiple random restarts and pick best run [higher dimensions make this less effective]
- Choosing the step size is a nuisance
  - RMSprop, AdaGrad, Adam [have their own hyperparameters]
- Only takes into account gradient information
  - Newton's method [requires second derivative]
- Each iteration requires an entire pass through the data set

# Limitations

- Not guaranteed to find the global optimum
  - Do multiple random restarts and pick best run [higher dimensions make this less effective]
- Choosing the step size is a nuisance
  - RMSprop, AdaGrad, Adam [have their own hyperparameters]
- Only takes into account gradient information
  - Newton's method [requires second derivative]
- Each iteration requires an entire pass through the data set
  - Stochastic gradient descent...

# Gradient descent

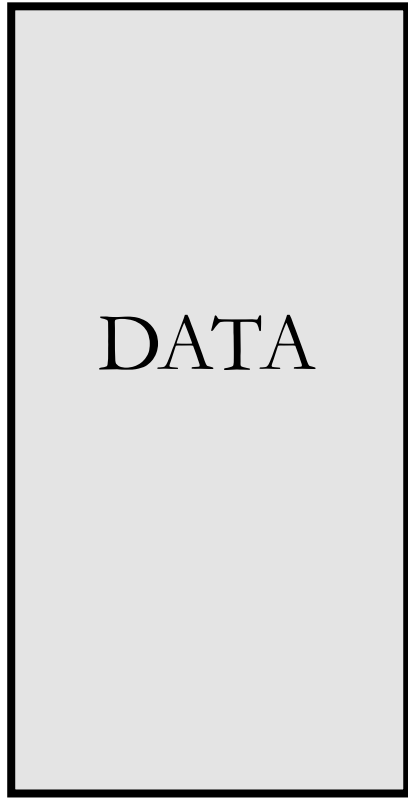
- We minimize the loss with respect to parameters
- Gradient descent

$$\nabla_x \mathcal{L} = \sum_i^N \nabla_x f_i(x)$$

- Too expensive:  $N$  can be large
- Early iterations: does a full pass over the data based on “bad” initial parameters

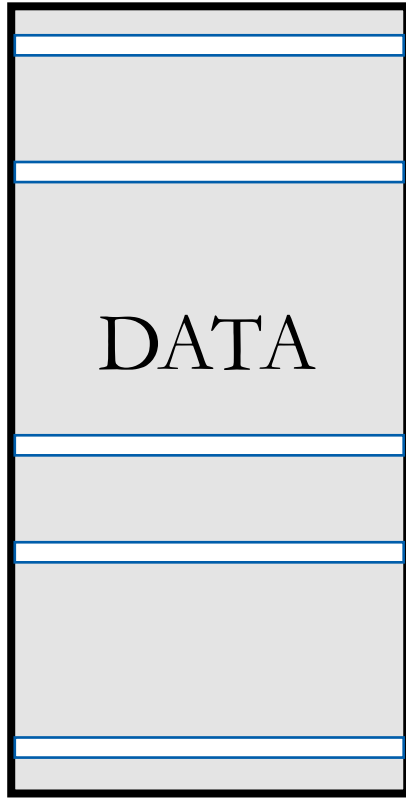
# Inference: SGD

- Stochastic gradient ascent



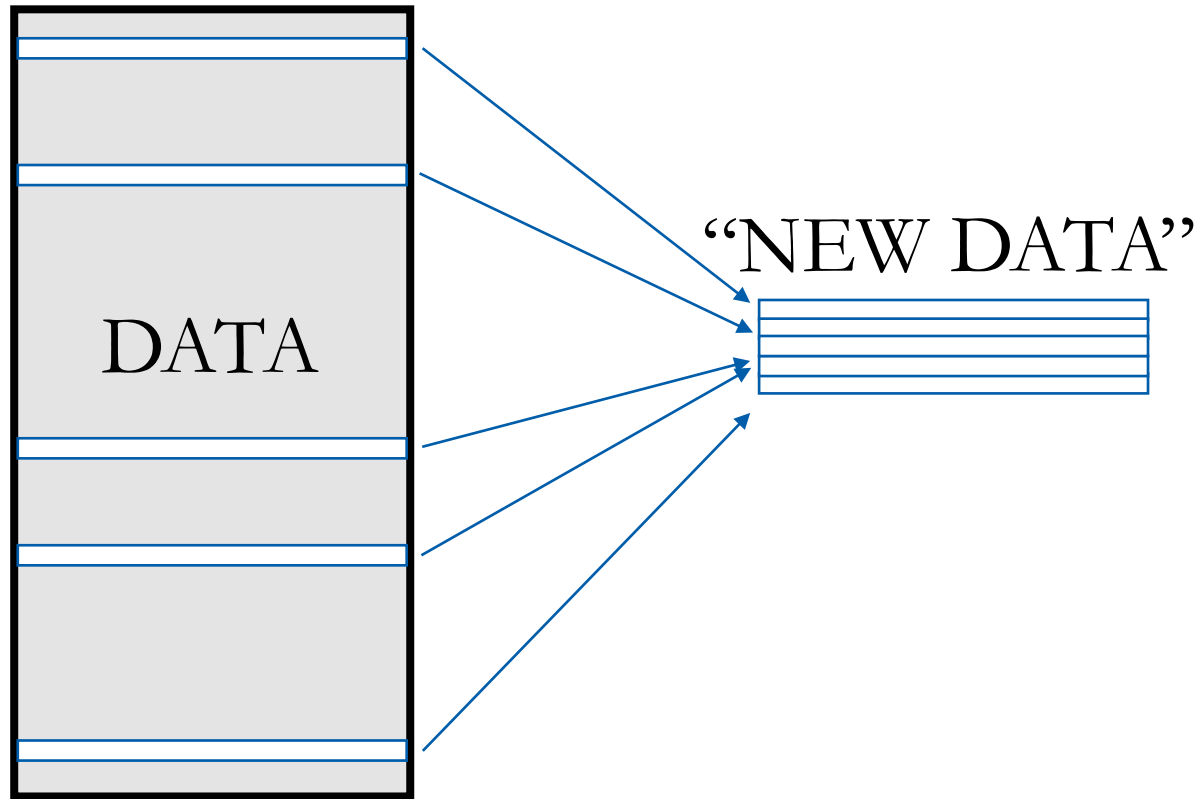
# Inference: SGD

- Stochastic gradient ascent



# Inference: SGD

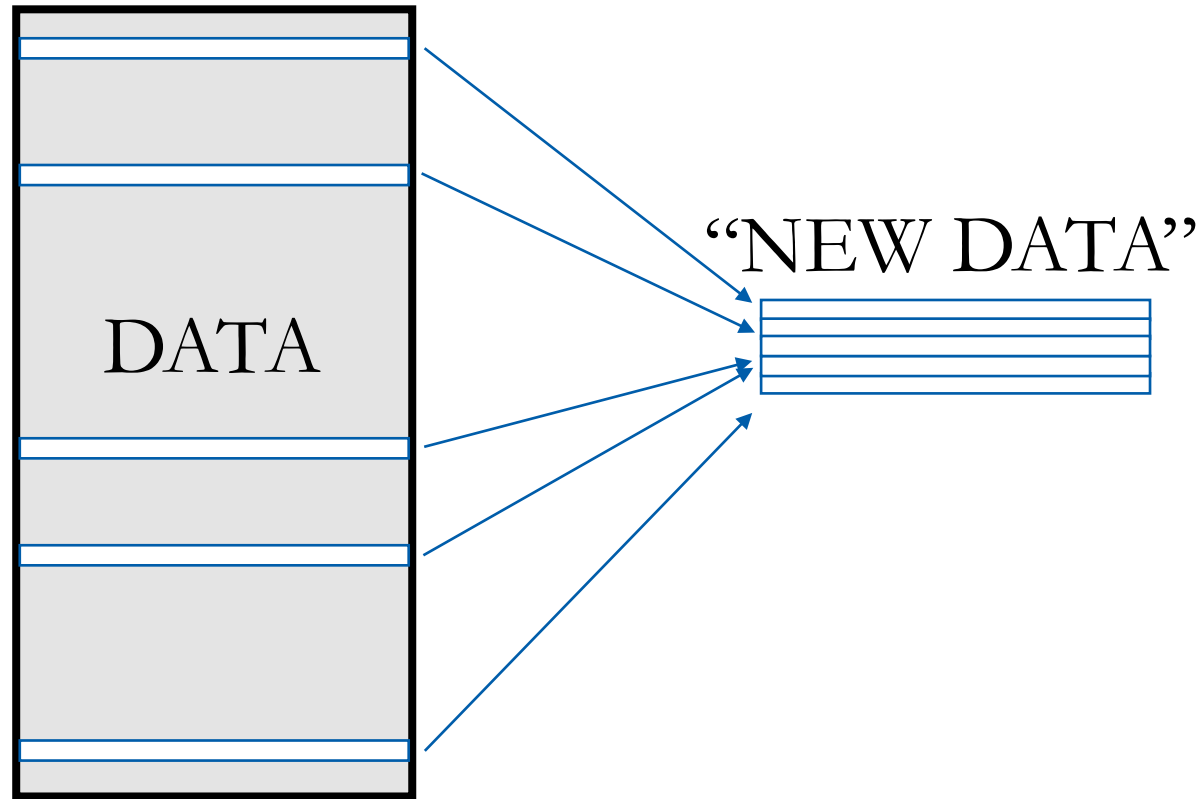
- Stochastic gradient ascent





# Inference: SGD

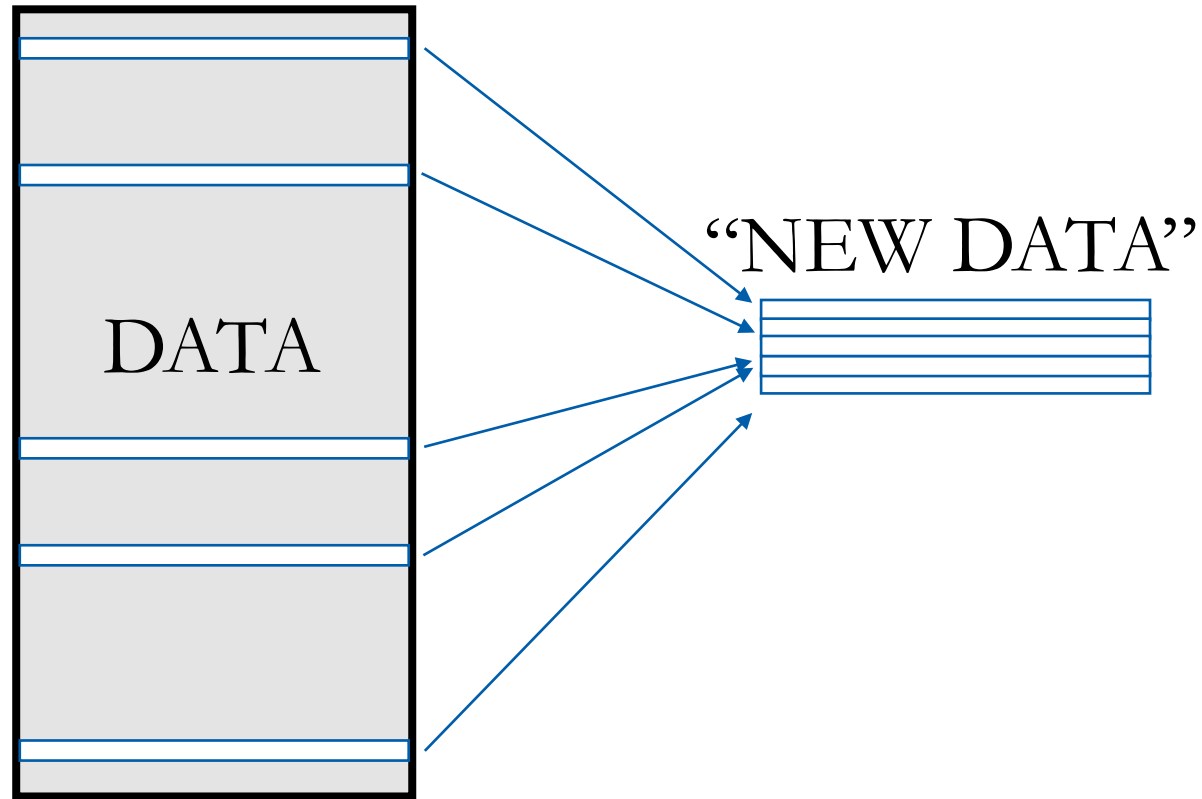
- Stochastic gradient ascent



- Subsample a **minibatch** of data
- Pretend that your dataset consists of this minibatch

# Inference: SGD

- Stochastic gradient ascent



- At each iteration of gradient descent, subsample a new minibatch
- Eventually, we end up using the entire dataset

# Inference: SGD

- Why does this work? (Analogy). Helps avoid local optima.

$$\mathbb{E}[\nabla_x f(x)] \approx \frac{1}{B} \sum_{i=1}^B \nabla f_i(x)$$

- Need learning rate to have the following properties:

$$\sum_t \rho_t \rightarrow \infty, \quad \sum_t \rho_t^2 < \infty$$

- [Robbins & Monro, 1951]  $\rho_t = (\rho_0 + t)^{-\kappa}$

# Issues in SGD

- Tradeoffs in learning rate:
  - too low, convergence may take a very long time
  - too high, may not converge at all
- Stopping criteria: use validation data
- Convexity vs. non-convexity

# Automatic Differentiation

- Modern ML systems are often implemented in libraries that allow you to automatically find gradients of “arbitrary” models.
- Advantages:
  - modular approach to learning rates, model components, error functions
  - reduce bugs, no manual derivation
  - software encapsulation of models (i.e. easier model comparison)
- Limitations:
  - handling “arbitrary” models is the goal not the current state (e.g., how are time series handled?)

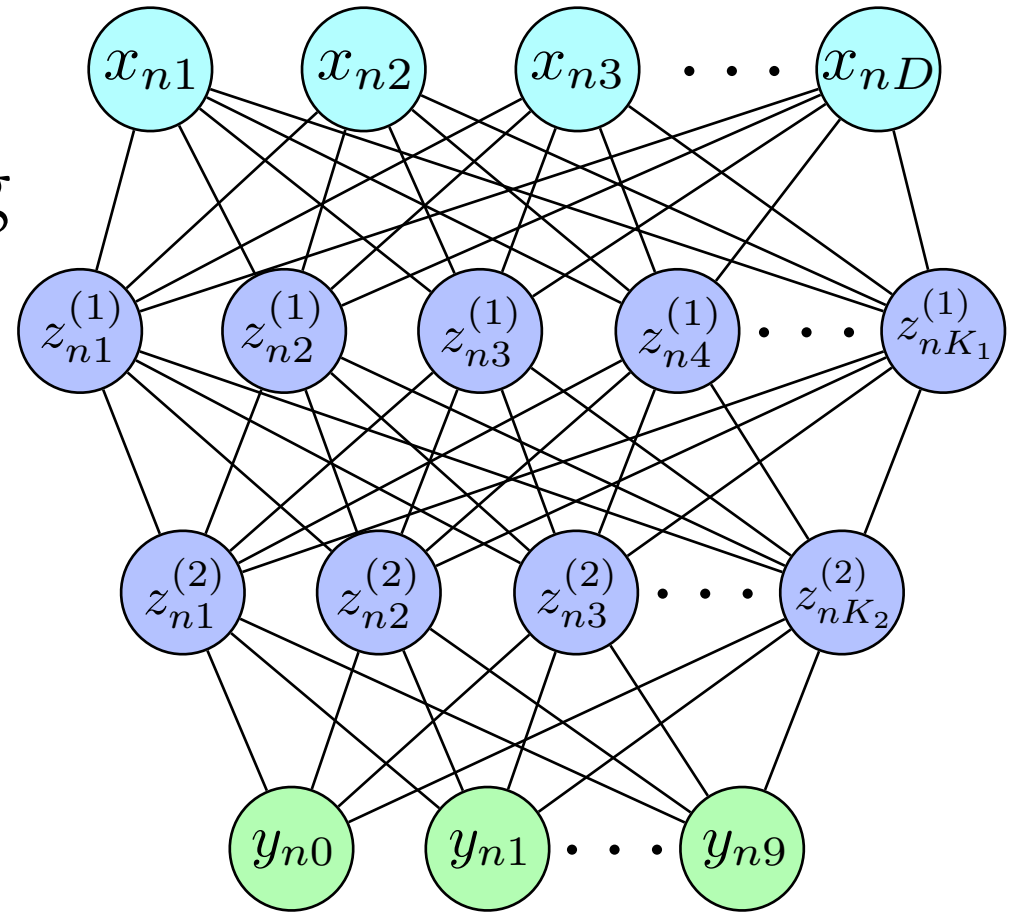


# Computational Graph

- The computational graph defines:
  - A. Data
  - B. Variables
  - C. Computations (network architecture, loss)
  - D. Optimizer
  - E. Other tasks (e.g., predictions on test)

# Computational Graph

- The computational graph
  - does not compute anything
  - does not hold any values
  - just specifies the model and variables



# Sessions

- A session allows you to
  - execute graph (or part of the graph)
  - allocate memory to hold variables
  - do computations



# Computational Graph & Sessions

```
import tensorflow as tf
```

# Computational Graph & Sessions

```
import tensorflow as tf

# Declare a graph
graph_name = tf.Graph()
with graph_name.as_default():
    # Declare inputs, variables, computations, ...
    aux_variable = tf.Variable( tf.constant(42.0) )
```

# Computational Graph & Sessions

```
import tensorflow as tf

# Declare a graph
graph_name = tf.Graph()
with graph_name.as_default():
    # Declare inputs, variables, computations, ...
    aux_variable = tf.Variable( tf.constant(42.0) )

with tf.Session(graph=graph_name) as session_name:
    # Initialize the variable
    tf.initialize_all_variables().run()
    # Print its value
    print(aux_variable.eval())
```

# Placeholders

- A placeholder
  - declares a variable with no specified value
  - “promises” to specify the value later

# Placeholders

```
import tensorflow as tf
```

# Placeholders

```
import tensorflow as tf

# Declare a graph
graph_name = tf.Graph()
with graph_name.as_default():
    # Declare a placeholder
    aux_var1 = tf.placeholder(tf.float32)
    # Define another variable
    aux_var2 = aux_var1 + 1.0
```

# Placeholders

```
import tensorflow as tf

# Declare a graph
graph_name = tf.Graph()
with graph_name.as_default():
    # Declare a placeholder
    aux_var1 = tf.placeholder(tf.float32)
    # Define another variable
    aux_var2 = aux_var1 + 1.0

with tf.Session(graph=graph_name) as session_name:
    # Create a feeder
    feed_dict = {aux_var1: 3.0}
    # Print the second variable
    print( aux_var2.eval(feed_dict=feed_dict) )
```

# Summary

- Gradient descent does maximum likelihood estimation
- Step sizes
- Stochastic gradient descent