CS 179: LECTURE 13

INTRO TO MACHINE LEARNING

GOALS OF WEEKS 5-6

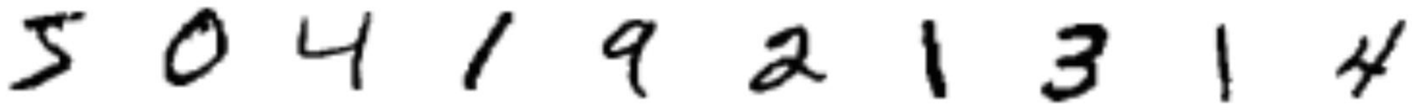
- What is machine learning (ML) and when is it useful?
- Intro to major techniques and applications
  - Give examples
- How can CUDA help?
- Departure from usual pattern: we will give the application first, and the CUDA later

HOW TO FOLLOW THIS LECTURE

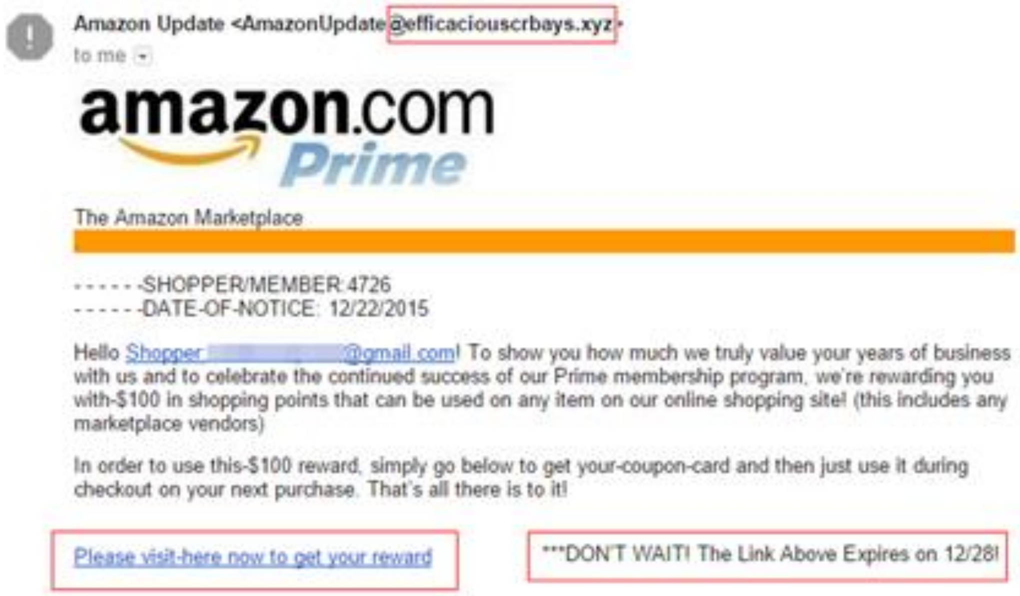
- This lecture and the next one will have a lot of math!
- Don't worry about keeping up with the derivations 100%
  - Important equations will be boxed
  - Key terms to understand: loss/objective function, linear regression, gradient descent, linear classifier
- The theory lectures will probably be boring for those of you who have done some machine learning (CS156/155) already

WHAT IS ML GOOD FOR?

- Handwriting recognition



- Spam detection



WHAT IS ML GOOD FOR?

- Teaching a robot how to do a backflip
  - https://youtu.be/fRj34o4hN4I
- Predicting the performance of a stock portfolio
- The list goes on!

WHAT IS ML?

- What do these problems have in common?
  - Some pattern we want to learn
  - No good closed-form model for it
  - LOTS of data
- What can we do?
  - Use **data** to learn a **statistical model** for the **pattern** we are interested in

DATA REPRESENTATION

- One data point is a vector x in $\mathbb{R}^d$
  - A 30×30 pixel image is a 900-dimensional vector (one component per pixel intensity)
  - If we are classifying an email as spam or not spam, set $d =$ number of words in dictionary
    - Count the number of times n_i that a word i appears in an email and set $x_i = n_i$
- The possibilities are endless ☺

WHAT ARE WE TRYING TO DO?

- Given an input $x \in \mathbb{R}^d$, produce an output y
- What is y?
  - Could be a real number, e.g. predicted return of a given stock portfolio
  - Could be 0 or 1, e.g. spam or not spam
  - Could be a vector in $\mathbb{R}^m$, e.g. telling a robot how to move each of its m joints
- Just like x, y can be almost anything ☺

EXAMPLE OF (x, y) PAIRS

- $\Bigg($  $, \begin{bmatrix}0\\0\\0\\0\\0\\1\\0\\0\\0\\0\end{bmatrix}\Bigg)$, $\Bigg($  $, \begin{bmatrix}1\\0\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}\Bigg)$, $\Bigg($  $, \begin{bmatrix}0\\1\\0\\0\\0\\0\\0\\0\\0\\0\end{bmatrix}\Bigg)$, $\Bigg($  $, \begin{bmatrix}0\\0\\0\\1\\0\\0\\0\\0\\0\\0\end{bmatrix}\Bigg)$, etc.

NOTATION

$$x' = \begin{pmatrix} 1 \\ x \end{pmatrix} \in \mathbb{R}^{d+1}$$

$$\mathbf{X} = \left(x^{(1)}, \dots, x^{(N)}\right) \in \mathbb{R}^{d \times N}$$

$$\mathbf{X}' = \left(x^{(1)'}, \dots, x^{(N)'}\right) \in \mathbb{R}^{(d+1) \times N}$$

$$\mathbf{Y} = \left(y^{(1)}, \dots, y^{(N)}\right)^T \in \mathbb{R}^{N \times m}$$

$$\mathbb{I}[p] = \begin{cases} 1 & p \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

STATISTICAL MODELS

- Given $(\mathbf{X}, \mathbf{Y})$ (N pairs of $\left(x^{(i)}, y^{(i)}\right)$ data), how do we accurately predict an output y given an input x?
- One solution: a model $f(x)$ parametrized by a vector (or matrix) w, denoted as $f(x; w)$
  - The task is finding a set of **optimal** parameters w

FITTING A MODEL

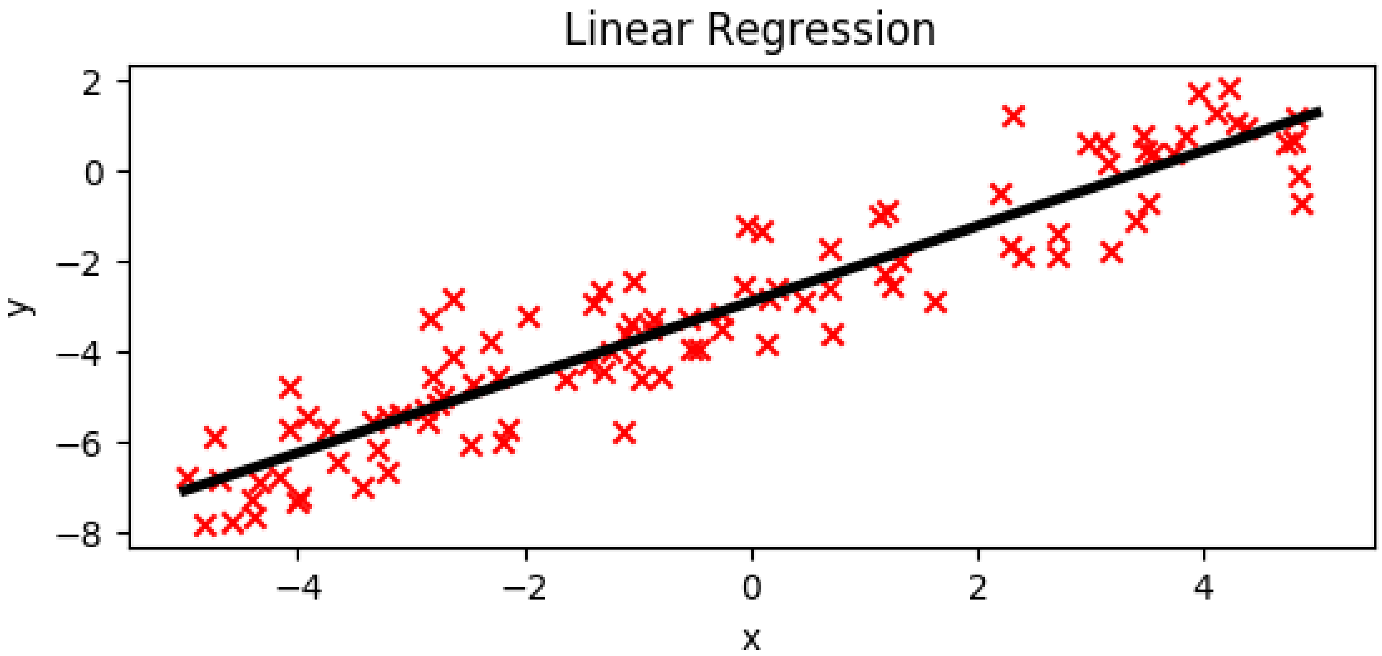
- So what does optimal mean?
  - Under some measure of closeness, we want $f(x; w)$ to be as close as possible to the true solution y for any input x
- This measure of closeness is called a **loss function** or **objective function** and is denoted $J(w; \mathbf{X}, \mathbf{Y})$ -- it depends on our data set $(\mathbf{X}, \mathbf{Y})$!
- To fit a model, we try to find parameters w^* that minimize $J(w; \mathbf{X}, \mathbf{Y})$, i.e. an **optimal** w

FITTING A MODEL

- What characterizes a good loss function?
  - Represents the magnitude of our model's error on the data we are given
  - Penalizes large errors more than small ones
  - Continuous and differentiable in w
  - Bonus points if it is also *convex* in w
- Continuity, differentiability, and convexity are to make minimization easier

LINEAR REGRESSION

- $f(x; w) = w_0 + \sum_{i=1}^{d} w_i x_i = w^T x'$
- Below: $d = 1$. $w^T x'$ is graphed.



LINEAR REGRESSION

- What should we use as a loss function?
  - Each $y^{(i)}$ is a real number
  - Mean-squared error is a good choice ☺
- $J(w; \mathbf{X}, \mathbf{Y}) = \frac{1}{N} \sum_{i=1}^{N} \left[f\left(x^{(i)}; w\right) - y^{(i)}\right]^2$

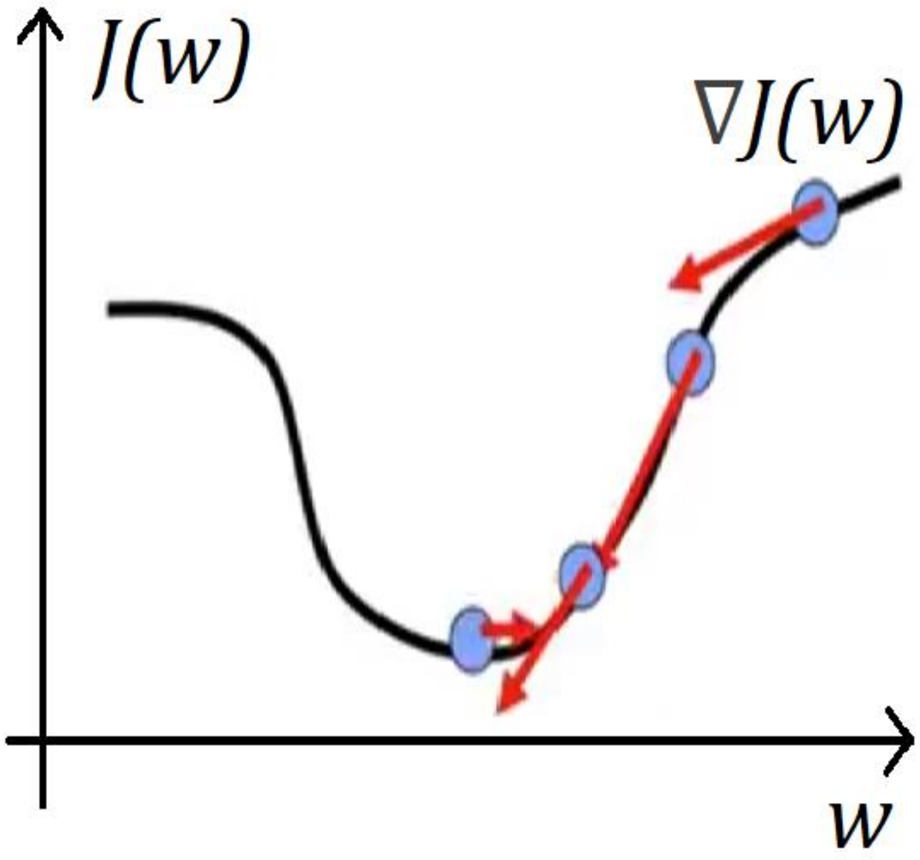
$$= \frac{1}{N} \sum_{i=1}^{N} \left[w^T x^{(i)\prime} - y^{(i)}\right]^2$$

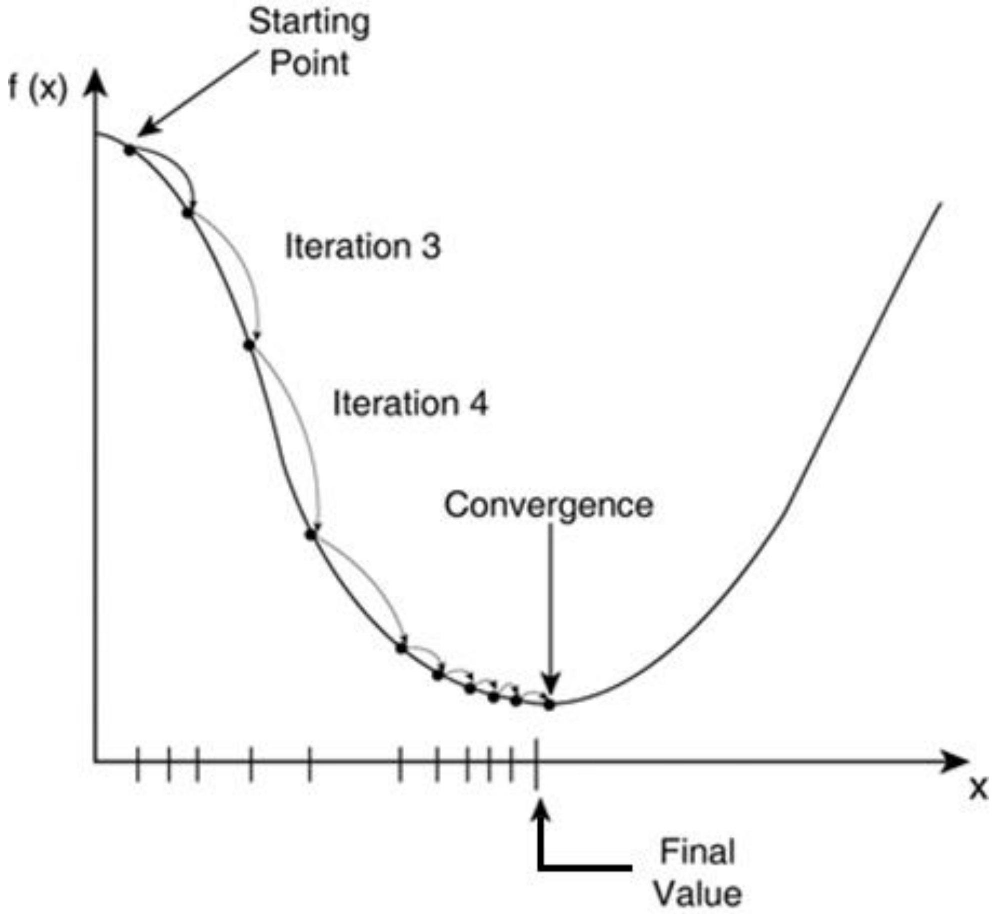
$$= \frac{1}{N} (w^T \mathbf{X}' - \mathbf{Y})^T (w^T \mathbf{X}' - \mathbf{Y})$$

GRADIENT DESCENT

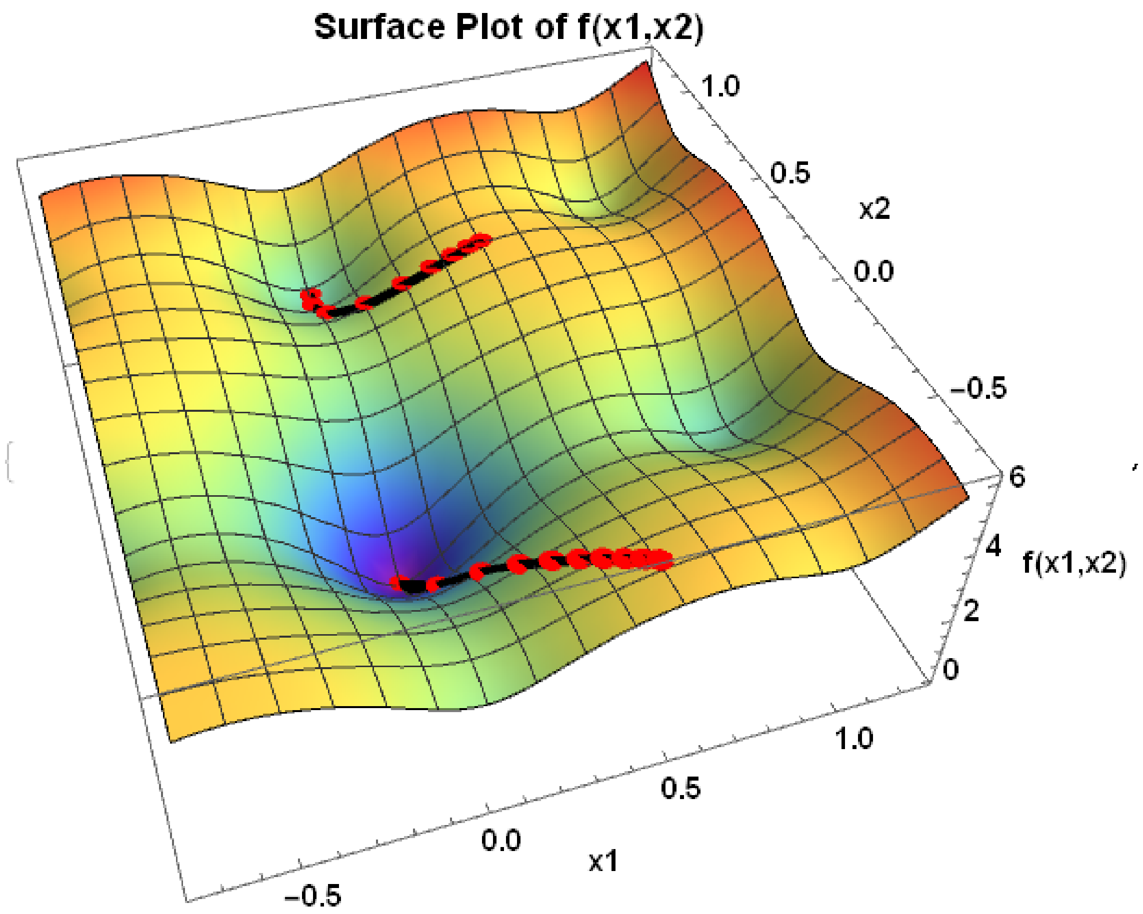
- How do we find $w^* = \underset{w \in \mathbb{R}^{d+1}}{\operatorname{argmin}} J(w; \mathbf{X}, \mathbf{Y})$?
- A function's gradient points in the direction of steepest ascent, and its negative in the direction of steepest descent
- Following the gradient downhill will cause us to converge to a local minimum!

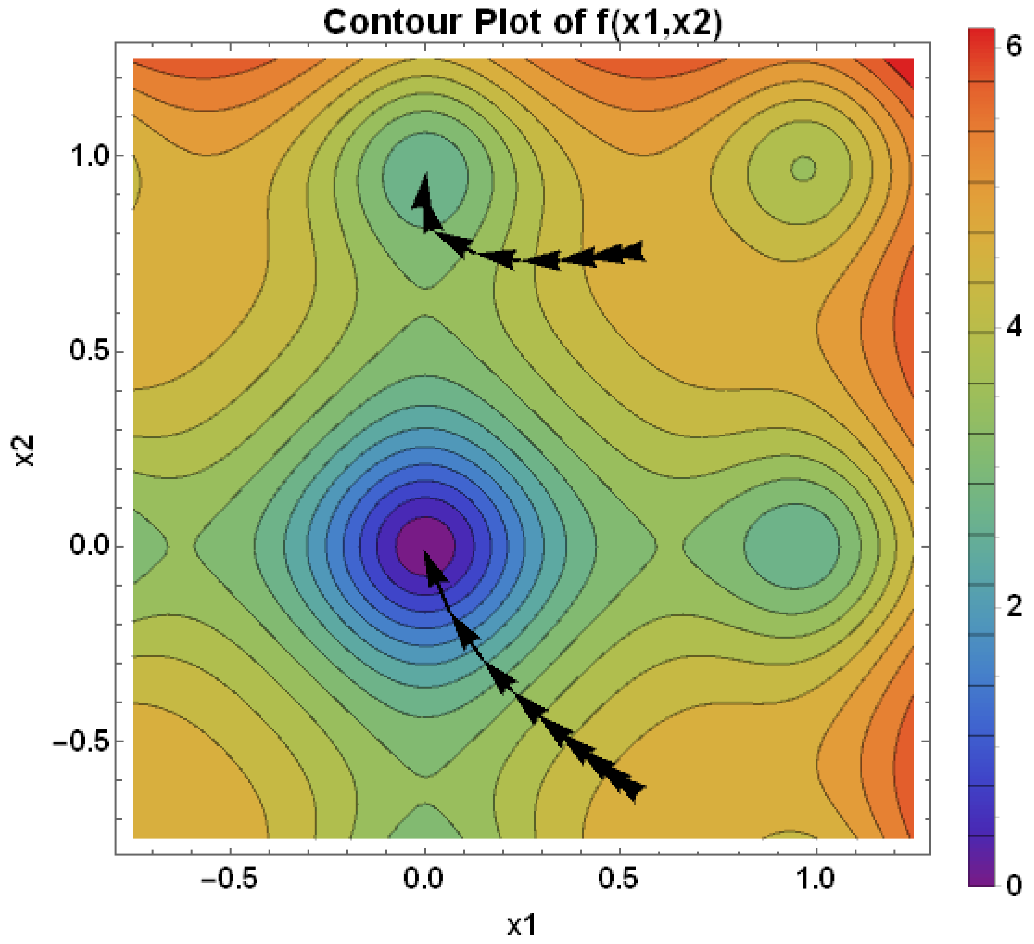
GRADIENT DESCENT





GRADIENT DESCENT





GRADIENT DESCENT

- Fix some constant learning rate η (0.03 is usually a good place to start)
- Initialize w randomly
  - Typically select each component of w independently from some standard distribution (uniform, normal, etc.)
- While w is still changing (hasn't converged)
  - Update $w \leftarrow w - \eta \nabla J(w; \mathbf{X}, \mathbf{Y})$

GRADIENT DESCENT

- For mean squared error loss in linear regression, $\nabla J(w; \mathbf{X}, \mathbf{Y}) = \frac{2}{N}\left(w^T \mathbf{X}' \mathbf{X}'^T - \mathbf{X}'\mathbf{Y}\right)$
- This is just linear algebra! GPUs are good at this kind of thing ☺
- Why do we care?
  - $f(x; w^*) = {w^*}^T x'$ is the model with the **lowest possible mean-squared error** on our training dataset $(\mathbf{X}, \mathbf{Y})$!

STOCHASTIC GRADIENT DESCENT

- The previous algorithm computes the gradient over the entire data set before stepping.
  - Called batch gradient descent
- What if we just picked a single data point $\left(x^{(i)}, y^{(i)}\right)$ at random, computed the gradient for that point, and updated the parameters?
  - Called stochastic gradient descent

STOCHASTIC GRADIENT DESCENT

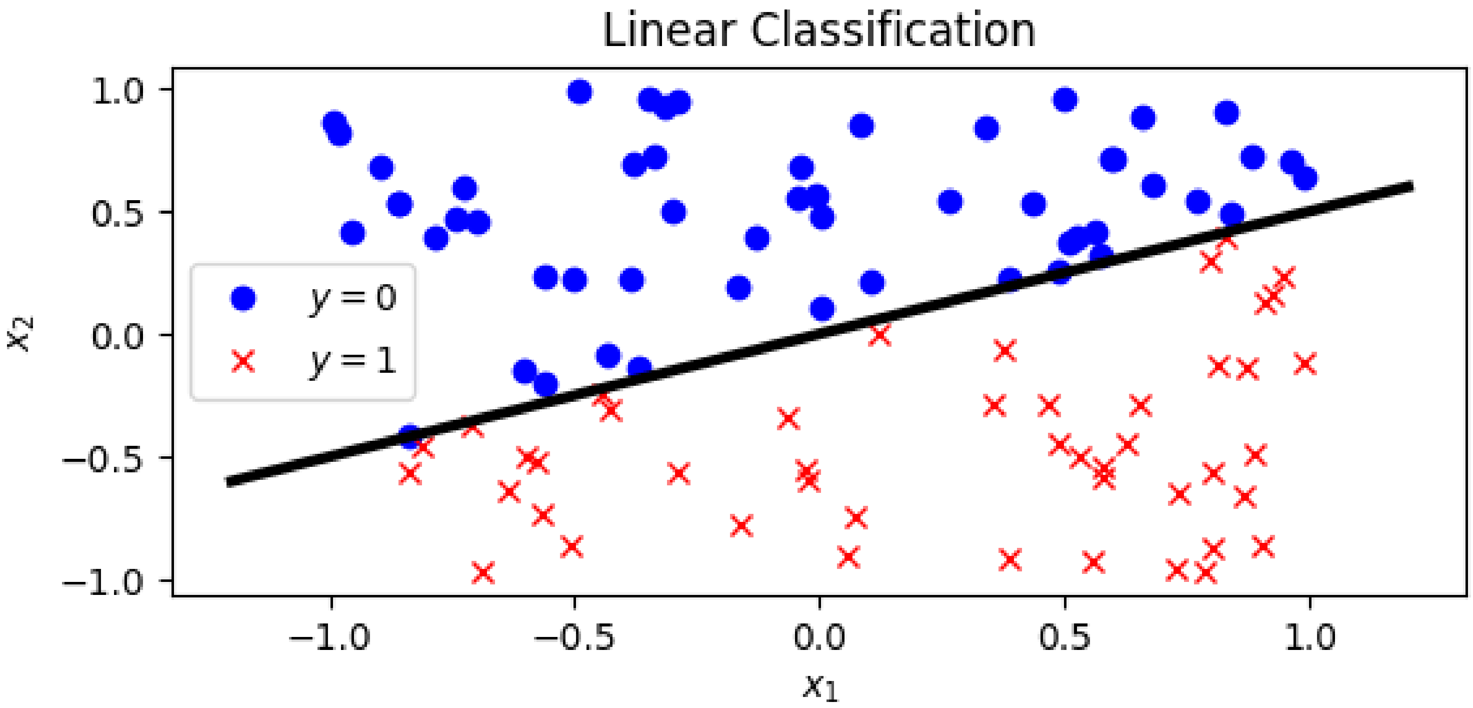
- Advantages of SGD
  - Easier to implement for large datasets
  - Works better for non-convex loss functions
  - Sometimes faster
- Often use SGD on a "mini-batch" of k examples rather than just one at a time
  - Allows higher throughput and more parallelization

BINARY LINEAR CLASSIFICATION

- $f(x; w) = \mathbb{I}[w^T x' > 0]$
- Divides $\mathbb{R}^d$ into two **half-spaces**
  - $w^T x' = 0$ is a hyperplane
    - A line in 2D, a plane in 3D, and so on
    - Known as the **decision boundary**
  - Everything on one side of the hyperplane is class 0 and everything on the other side is class 1

BINARY LINEAR CLASSIFICATION

- Below: $d = 2$. Black line is the decision boundary $w^T x' = 0$



MULTI-CLASS GENERALIZATION

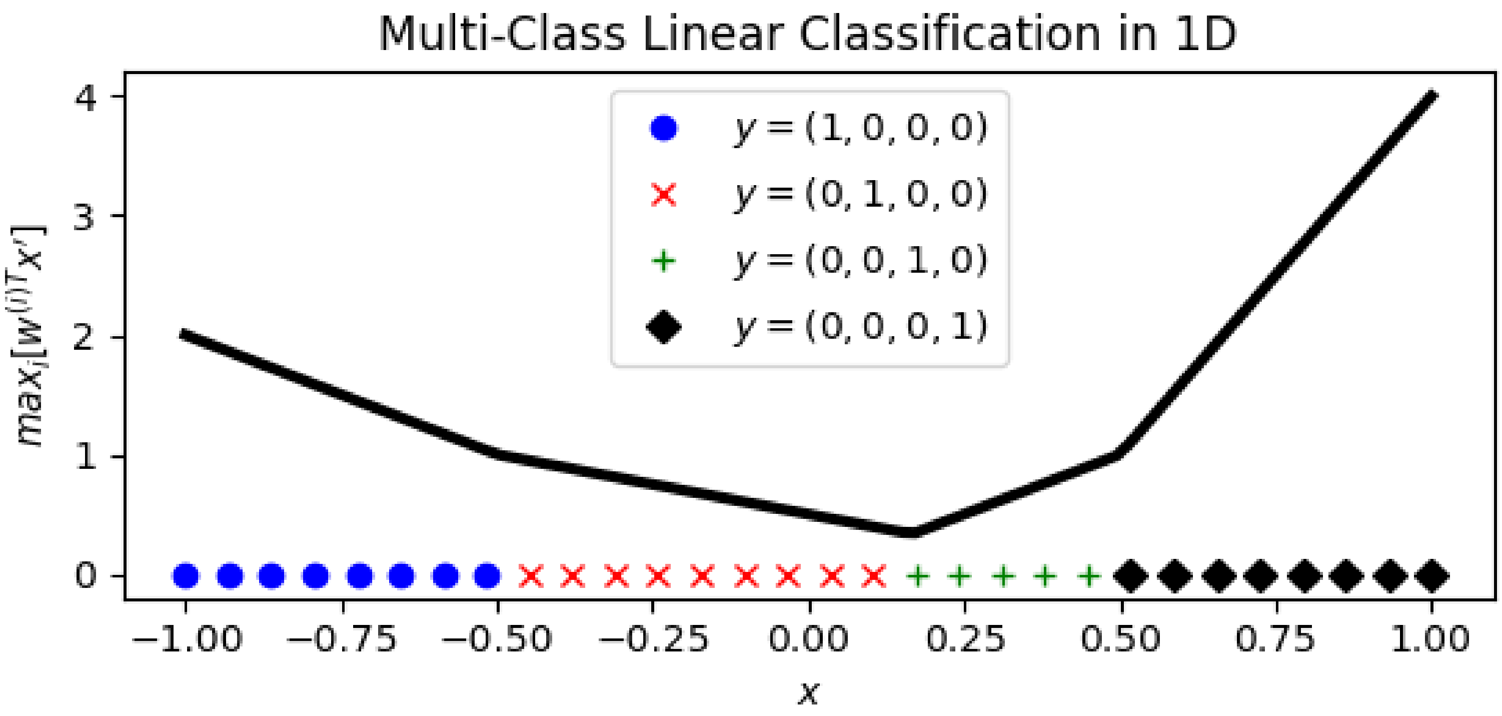
- We want to classify x into one of m classes
- For each input x, y is a vector in $\mathbb{R}^m$ with $y_k = 1$ if $\text{class}(x) = k$ and $y_j = 0$ otherwise (i.e. $y_k = \mathbb{I}[\text{class}(x) = k]$)
  - Known as a **one-hot vector**
- Our model $f(x; \mathbf{W})$ is parametrized by a $m \times (d + 1)$ matrix $\mathbf{W} = \left(w^{(1)}, \dots, w^{(m)}\right)$
- The model returns an m-dimensional vector (like y) with $f_k(x; \mathbf{W}) = \mathbb{I}\left[\arg\max_i {w^{(i)}}^T x' = k\right]$

MULTI-CLASS GENERALIZATION

- ${w^{(j)}}^T x' = {w^{(k)}}^T x'$ describes the intersection of 2 hyperplanes in $\mathbb{R}^{d+1}$ (where $x \in \mathbb{R}^d$)
  - Divides $\mathbb{R}^d$ into half-spaces; ${w^{(j)}}^T x' > {w^{(k)}}^T x'$ on one side, vice versa on the other side.
  - If ${w^{(j)}}^T x' = {w^{(k)}}^T x' = \max_i {w^{(i)}}^T x'$, this is a decision boundary!
- Illustrative figures follow

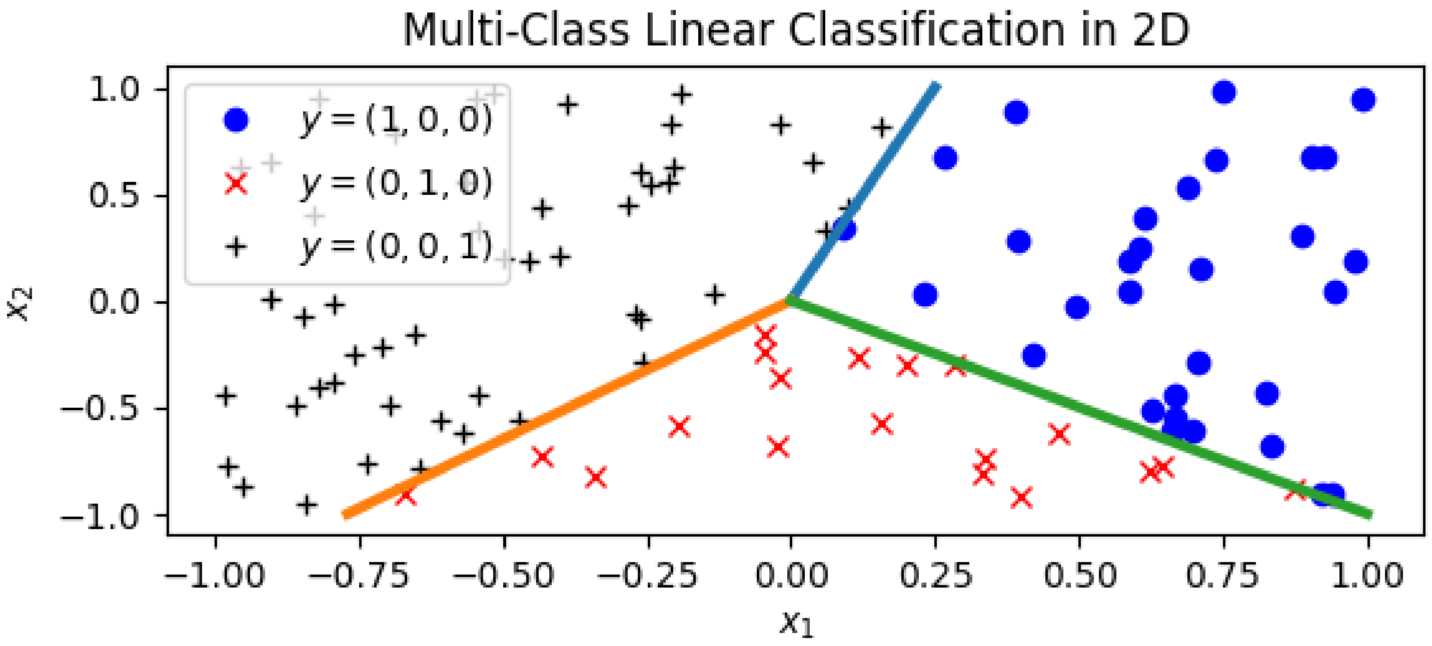
MULTI-CLASS GENERALIZATION

- Below: $d = 1, m = 4$. $\max_i w^{(i)^T} x'$ is graphed.



MULTI-CLASS GENERALIZATION

- Below: $d = 2, m = 3$. Lines are decision boundaries $w^{(j)^T}x = w^{(k)^T}x = \max_i w^{(i)^T}x$



MULTI-CLASS GENERALIZATION

- For $m = 2$ (binary classification), we get the scalar version by setting $w = w^{(1)} - w^{(0)}$
- $f_1(x; \mathbf{W}) = \mathbb{I}\left[\arg\max_i {w^{(i)}}^T x' = 1\right]$

$$= \mathbb{I}\left[{w^{(1)}}^T x' > {w^{(0)}}^T x'\right]$$

$$= \mathbb{I}\left[\left(w^{(1)} - w^{(0)}\right)^T x' > 0\right]$$

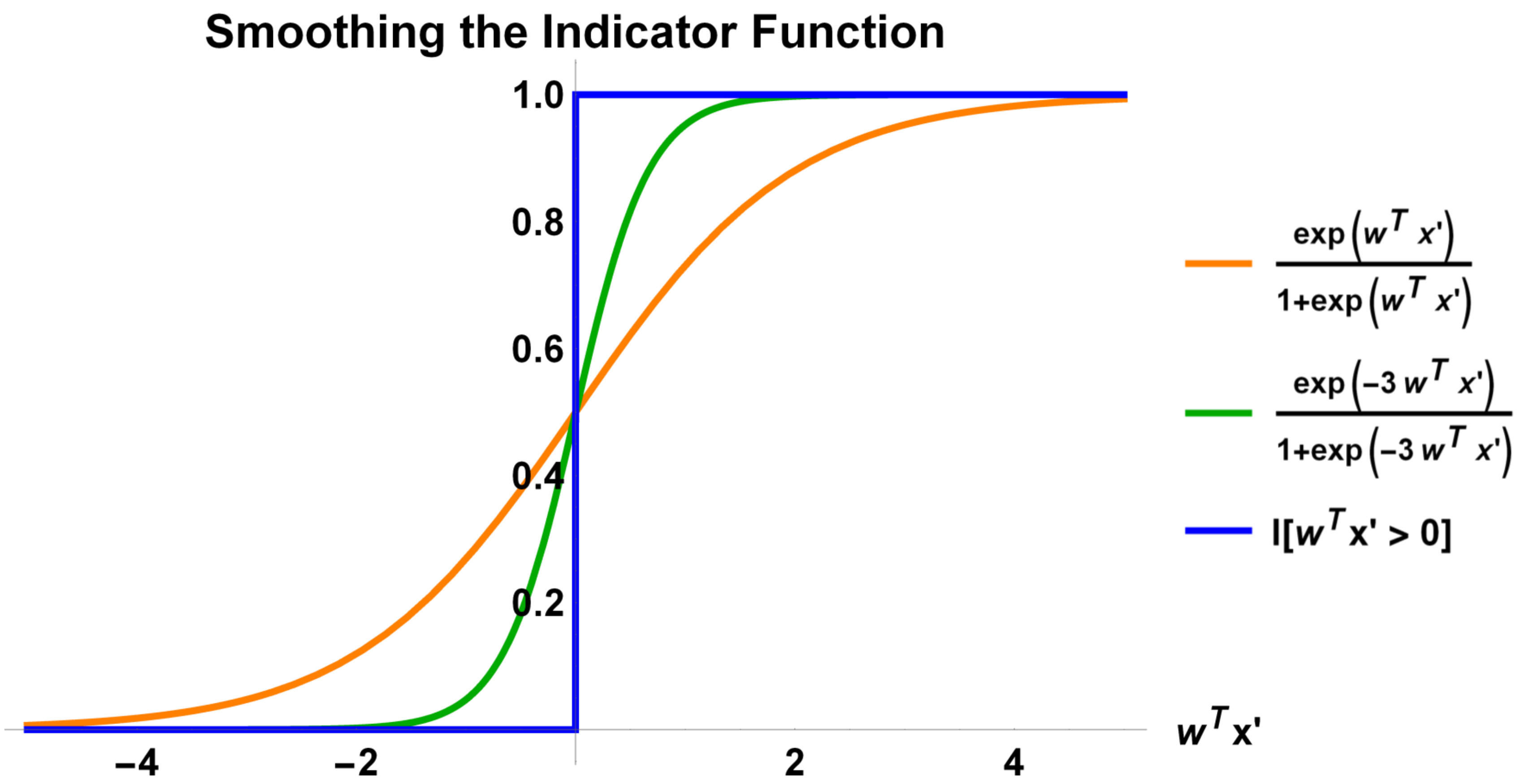
FITTING A LINEAR CLASSIFIER

- $f(x; w) = \mathbb{I}[w^T x' > 0]$
- How do we turn this into something continuous and differentiable?
- We really want to replace the indicator function $\mathbb{I}$ with a smooth function indicating the **probability** of whether y is 0 or 1, based on the value of $w^T x'$

PROBABILISTIC INTERPRETATION

- Interpreting $w^T x'$
  - $w^T x'$ large and positive
    - $\mathbb{P}[y = 0] \ll \mathbb{P}[y = 1]$
  - $w^T x'$ large and negative
    - $\mathbb{P}[y = 0] \gg \mathbb{P}[y = 1]$
  - $|w^T x'|$ small
    - $\mathbb{P}[y = 0] \approx \mathbb{P}[y = 1]$

PROBABILISTIC INTERPRETATION



PROBABILISTIC INTERPRETATION

- We therefore use the probability functions
  - $p_0(x; w) = \mathbb{P}[y = 0] = \frac{1}{1+\exp(w^T x')}$
  - $p_1(x; w) = \mathbb{P}[y = 1] = \frac{\exp(w^T x')}{1+\exp(w^T x')}$
- If $w = w^{(1)} - w^{(0)}$ as before, this is just

$$p_k(x; w) = \mathbb{P}[y = k] = \frac{\exp\left({w^{(k)}}^T x'\right)}{\exp\left({w^{(0)}}^T x'\right)+\exp\left({w^{(1)}}^T x'\right)}$$

PROBABILISTIC INTERPRETATION

- In the more general m-class case, we have

$$\boxed{p_k(x;\mathbf{W}) = \mathbb{P}[y_k = 1] = \frac{\exp\left({w^{(k)}}^T x'\right)}{\sum_{i=1}^{m} \exp\left({w^{(i)}}^T x'\right)}}$$

- This is called the **softmax activation** and will be used to define our loss function

THE CROSS-ENTROPY LOSS

- We want to heavily penalize cases where $y_k = 1$ with $p_k(x; \mathbf{W}) \ll 1$
- This leads us to define the **cross-entropy loss** as follows:

$$\boxed{J(\mathbf{W}; \mathbf{X}, \mathbf{Y}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{m} y_k^{(i)} \ln\left(p_k\left(x^{(i)}; \mathbf{W}\right)\right)}$$

MINIMIZING CROSS-ENTROPY

- As with mean-squared error, the cross-entropy loss is convex and differentiable ☺
- That means that we can use gradient descent to converge to a global minimum!
- This global minimum defines the **best possible** linear classifier with respect to the cross-entropy loss and the data set given

SUMMARY

- Basic process of constructing a machine learning model
- Choose an analytically well-behaved loss function that represents some notion of error for your task
- Use gradient descent to choose model parameters that minimize that loss function for your data set
- Examples: linear regression and mean squared error, linear classification and cross-entropy

NEXT TIME

- Gradient of the cross-entropy loss
- Neural networks
- Backpropagation algorithm for gradient descent