Tuesday, Feb 24

- Missed quiz policy
  1. Reach out before class
  2. Take quiz at first OH after

Goal: Check understanding

Incentivize class attendance

- Late assignment policy
  - ↳ 24 hours late window (life happens)
  - ↳ hard deadline after that

Plan

- Logistic Regression
- Pytorch demo

Optimization

exact or gradient descent

Both need the gradient, so...

Lemma: $\nabla_w \mathcal{L}(w) = X^T(\sigma(Xw) - y)$

$z = \langle w, x \rangle$, $\hat{y} = \sigma(z)$

Fact 1: $\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1 - \sigma(z)) = \hat{y}(1-\hat{y})$

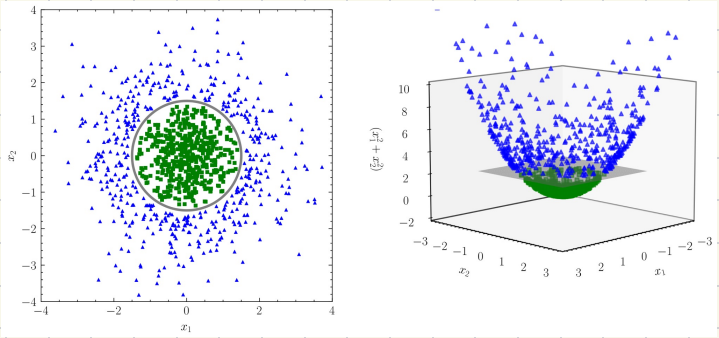
$$\ell = -y \log \hat{y} - (1-y)\log(1-\hat{y})$$

Fact 2: $\frac{\partial \ell}{\partial \hat{y}} = -\frac{y}{\hat{y}} + \frac{(1-y)}{(1-\hat{y})} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})}$

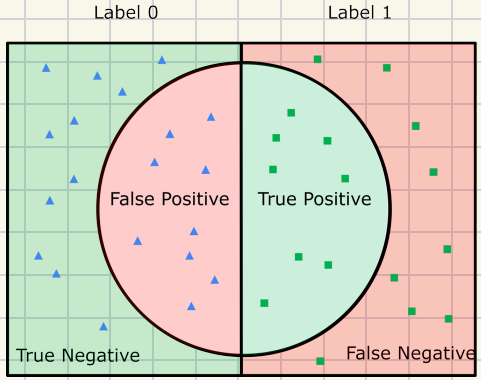
$$\nabla_w \mathcal{L}(w) = \sum_{i=1}^{n} \frac{\partial \ell^{(i)}}{\partial \hat{y}^{(i)}} \frac{\partial \hat{y}^{(i)}}{\partial z^{(i)}} \nabla_w z^{(i)}$$

$$= \dots$$

Non-linear Transformations



Measuring Error

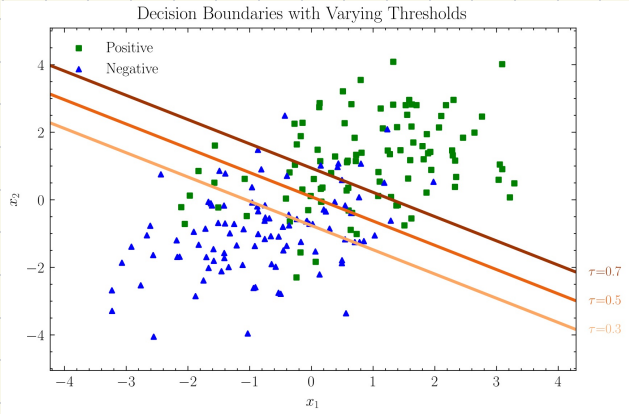


TPR = $\frac{\text{True Positive}}{\text{False Negative}}$

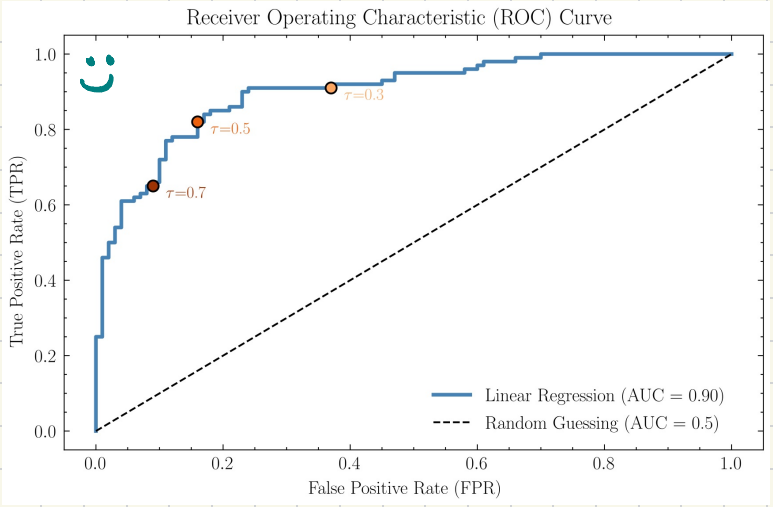
FPR = $\frac{\text{False Positive}}{\text{True Negative}}$

Precision = $\frac{\text{True Positive}}{\text{False Positive}}$

Varying Threshold



Receiver Operating Characteristic Curve



Output 1 if $\sigma(\langle w, x \rangle) > 1/2$ (τ)

Output 0 else

Decreasing τ:

↑ TPR

↑ FPR

Thursday, Feb 26

- Exam 3/5

Previously:

1. Choose features (and transformations)
2. Optimize model

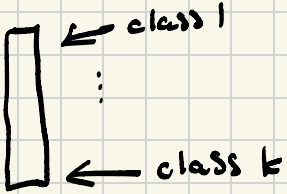
Neural networks allow us to do both!

Multiple Classes

$f: \mathbb{R}^d \to \mathbb{R}^K \qquad W \in \mathbb{R}^{K \times d}$

Wx

$K \times d \quad d \times 1$



$$f(x) = \text{softmax}(Wx)$$

$$\mathcal{L}(W) = -\sum_{i=1}^{n} \sum_{j=1}^{K} \mathbb{1}[y^{(i)} = j] \log(f_j(x^{(i)}))$$

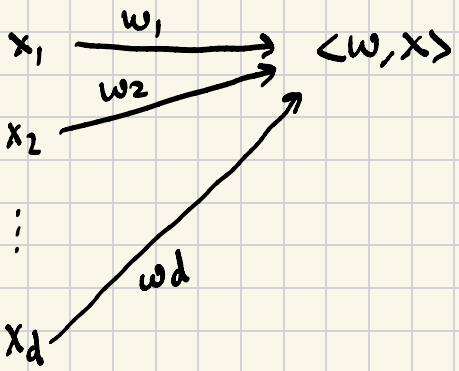
Goal: Convert to probability distribution

1. Between 0 and 1
2. Sum to 1

$z \in \mathbb{R}^K$

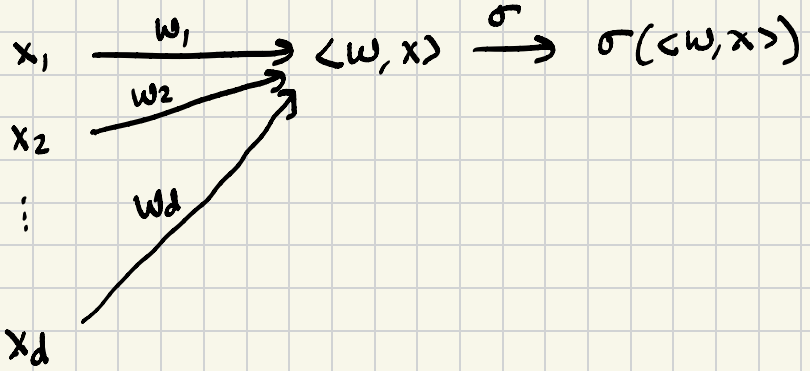
$$\text{softmax}(z) = \frac{1}{\sum_{j=1}^{K} e^{z_j}} \begin{bmatrix} e^{z_1} \\ e^{z_2} \\ \vdots \\ e^{z_K} \end{bmatrix}$$

Linear Regression $f: \mathbb{R}^d \to \mathbb{R}$



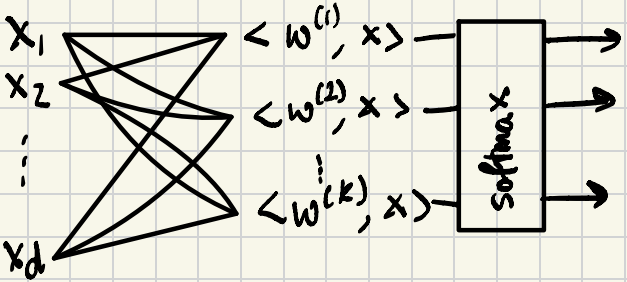
$$\mathcal{L}(w) = \frac{1}{n} \sum_{i=1}^{n} \left(f(x^{(i)}) - y^{(i)} \right)^2$$

Binary Logistic Regression $f: \mathbb{R}^d \to (0,1)$



$$\mathcal{L}(w) = -\sum_{i=1}^{n} y^{(i)} \log \left(f(x^{(i)}) \right) + (1 - y^{(i)}) \log \left(1 - f(x^{(i)}) \right)$$

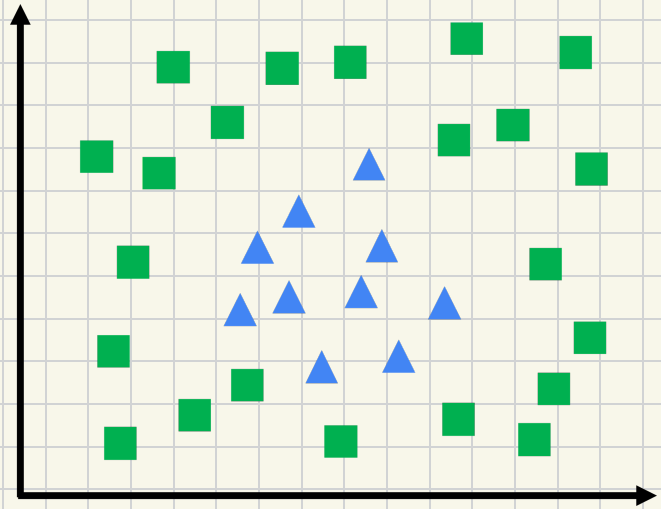
Multi-class Logistic Regression $f: \mathbb{R}^d \to (0,1)^k$



$\text{softmax}(Wx)$

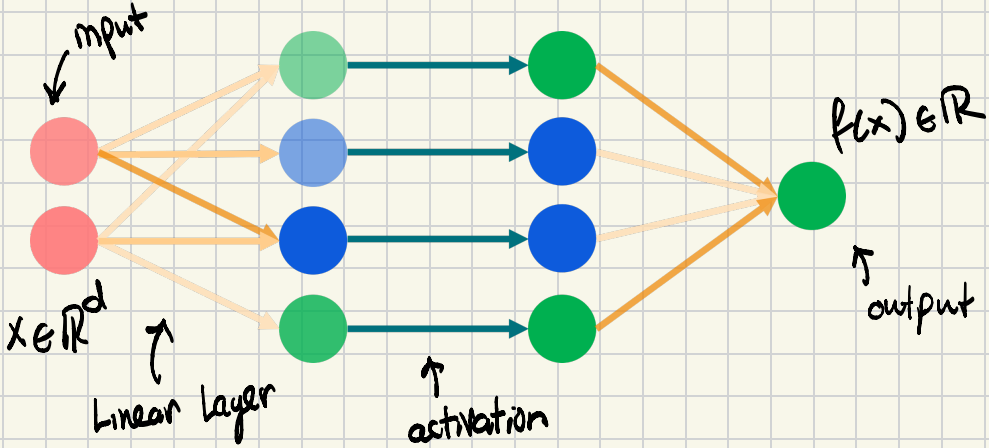
$$\mathcal{L}(w) = -\sum_{i=1}^{n} \sum_{j=1}^{n} \mathbb{1}[y^{(i)} = j] \log f_j(x^{(i)})$$

Motivating Example



Could (manually) add degree-2 features or...

Neural Networks



Linear Algebraic view):

Options Include

- Activation
  - ↳ ReLU
  - ↳ Sigmoid
  - ↳ Tanh
  - ↳ Softmax
- Number of layers
- Number of neurons
- Loss function
  - ↳
  - ↳
  - ↳

Backpropagation

$f: \mathbb{R} \to \mathbb{R}$

$f(x)$

$$\frac{\partial f}{\partial x} = \lim_{t \to 0} \frac{f(x+t) - f(x)}{t}$$

$g: \mathbb{R} \to \mathbb{R}$

$$\frac{\partial f(g(x))}{\partial x} = \lim_{t \to 0} \frac{f(g(x+t)) - f(g(x))}{t}$$

$$= \lim_{t \to 0} \frac{f(g(x+t)) - f(g(x))}{g(x+t) - g(x)} \cdot \frac{g(x+t) - g(x)}{g(x)}$$

$$= \frac{\partial f}{\partial g} \frac{\partial g}{\partial x}$$

$f: \mathbb{R}^m \to \mathbb{R}$

$f(x_1, \dots, x_m)$

$$\frac{\partial f}{\partial x_i} = \lim_{t \to 0} \frac{f(x_1, \dots, x_i + t, \dots, x_m) - f(x_1, \dots, x_i, \dots, x_m)}{t}$$

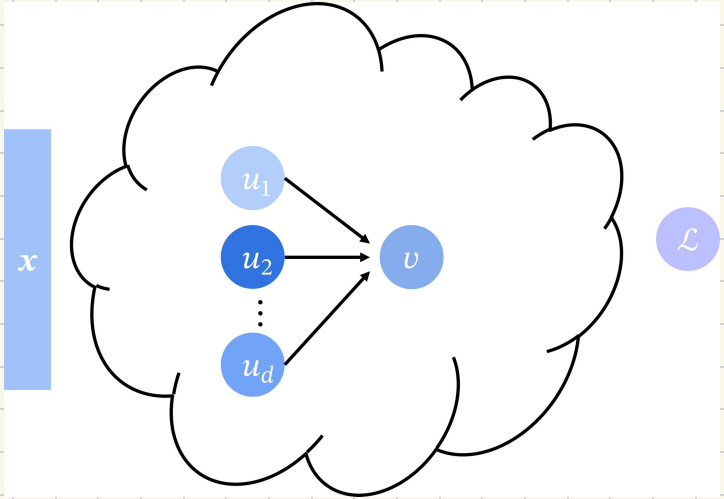
$g_i: \mathbb{R} \to \mathbb{R}$

$$\frac{\partial f(g_1(x), \dots, g_m(x))}{\partial x} =$$

$$\frac{\partial f}{\partial g_1} \frac{\partial g_1}{\partial x} + \dots + \frac{\partial f}{\partial g_m} \frac{\partial g_m}{\partial x}$$

Forward

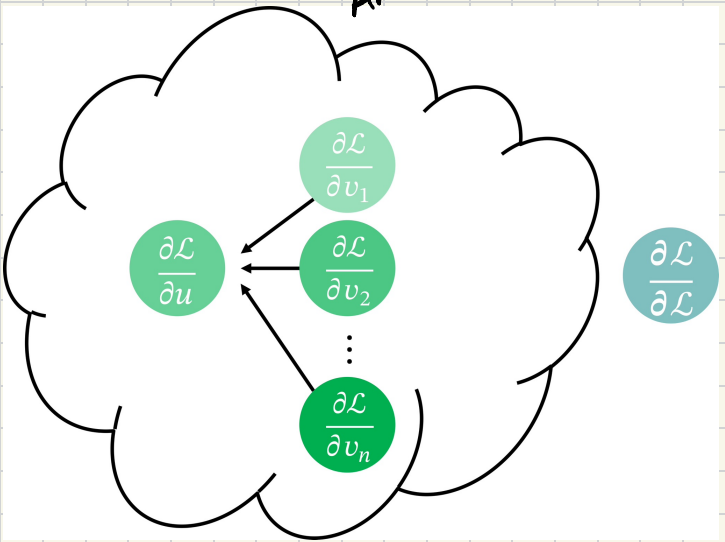
Already know $u_1, \dots, u_d$



$v(u_1, \dots, u_d)$

Backward

Already know $\frac{\partial \mathcal{L}}{\partial v_1}, \dots, \frac{\partial \mathcal{L}}{\partial v_n}$



$$\frac{\partial \mathcal{L}}{\partial u} = \frac{\partial \mathcal{L}}{\partial v_1}\frac{\partial v_1}{\partial u} + \dots + \frac{\partial \mathcal{L}}{\partial v_n}\frac{\partial v_n}{\partial u}$$

Linear Algebraic View of Backpropagation

Neural nets known since 1950's

Backprop known since 1960's

Why now?

GPUs used starting 2010's

↑ really good at matrix multiplication,

so training and inference cast as matrix multiplication

Forward

$$v = Wu \qquad v_i = \sum_{j=1}^{d} [W]_{i,j} u_j \qquad \frac{\partial v_i}{\partial u_j} = [W]_{i,j}$$

Backward

$$\nabla_u \mathcal{L} = W^T \nabla_v \mathcal{L} \qquad \frac{\partial \mathcal{L}}{\partial u_j} = \sum_{i=1}^{n} \frac{\partial \mathcal{L}}{\partial v_i} \frac{\partial v_i}{\partial u_j} = \sum_{i=1}^{n} [W^T]_{j,i} \frac{\partial \mathcal{L}}{\partial v_i}$$

Note: Instead of the gradients with respect to the neurons, we are actually interested in the gradients of the loss function with respect to the parameters, because we are updating the *parameters* during training. But, we need the neuron gradients to compute the parameter gradients: Once we have the neuron gradients, we compute the parameter gradients with another application of the chain rule.

The power of the linear algebraic view is that we can use hardware specialized for matrix multiplication to efficiently compute the gradients!

Neural networks are a powerful tool for learning complex functions, but notoriously difficult to understand because of their many parameters and nonlinearities. The big surprise of deep learning is that, even though they are not convex functions, gradient descent works remarkably well to train them. We don't have good intuition for what happens in high-dimensional space (the loss function depends on the number of parameters, which can be very large), but it *seems* like there's almost always a parameter update that at least slightly improves the loss function so we rarely get stuck in a bad local minima.