Tuesday, March 3

- Exam on Thursday
- Self-grade due Friday

Plan

- Review neural nets
- Questions
- Backpropagation

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

$$\begin{tikzcd}
x_1 \arrow[r, "w_1"] & \langle w, x \rangle \\
x_2 \arrow[ru, "w_2"] & \\
\vdots & \\
x_d \arrow[ruuu, "w_d"'] &
\end{tikzcd}$$

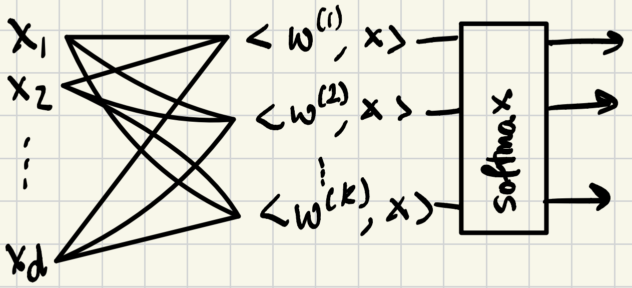
$$\mathcal{L}(w) = \frac{1}{n} \sum_{i=1}^{n} \left(f(x^{(i)}) - y^{(i)} \right)^2$$

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

$$\begin{tikzcd}
x_1 \arrow[r, "w_1"] & \langle w, x \rangle \arrow[r, "\sigma"] & \sigma(\langle w, x \rangle) \\
x_2 \arrow[ru, "w_2"] & & \\
\vdots & & \\
x_d \arrow[ruuu, "w_d"] & &
\end{tikzcd}$$

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

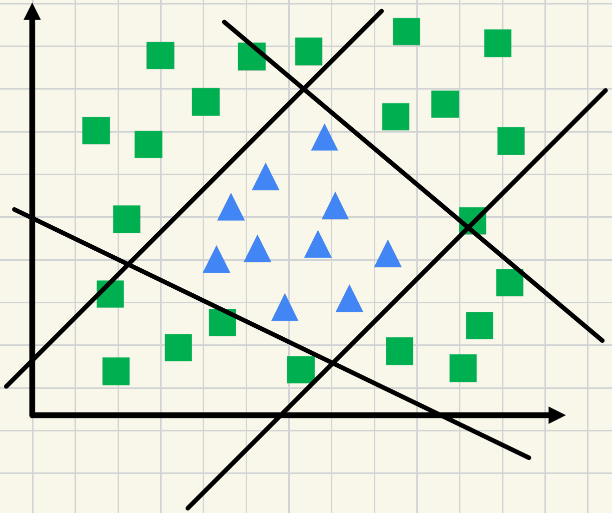
Multi-class Logistic Regression $f: \mathbb{R}^d \to (0,1)^k$



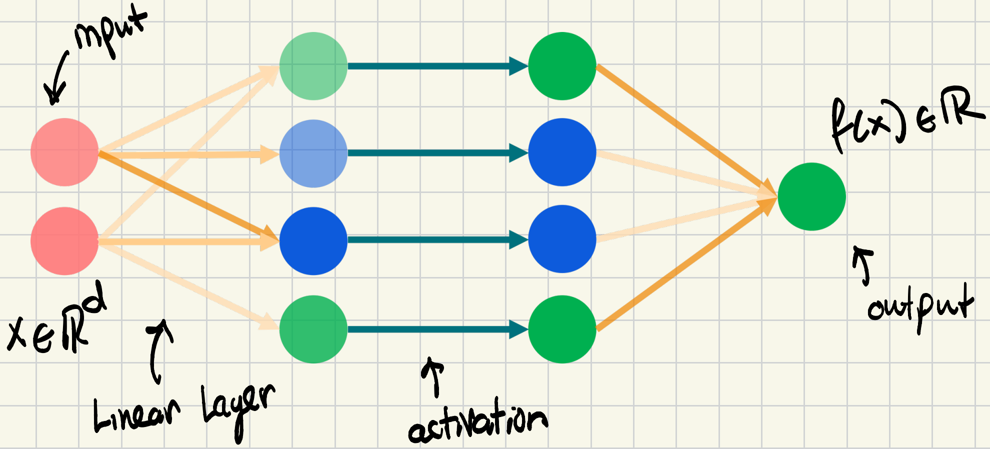
softmax(Wx)

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

Motivating Example



Neural Networks



Chain Rule

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

$f(x)$

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

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

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

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

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

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

$f(x_1, \dots, x_m)$

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

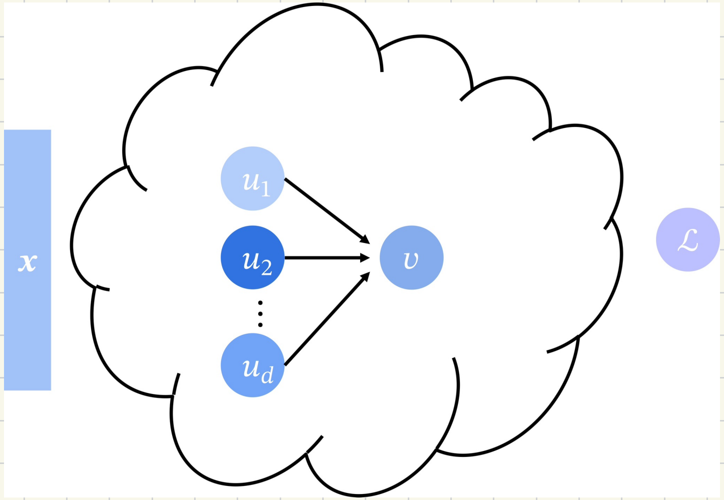
$g_i : \mathbb{R} \to \mathbb{R}$

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

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

Forward

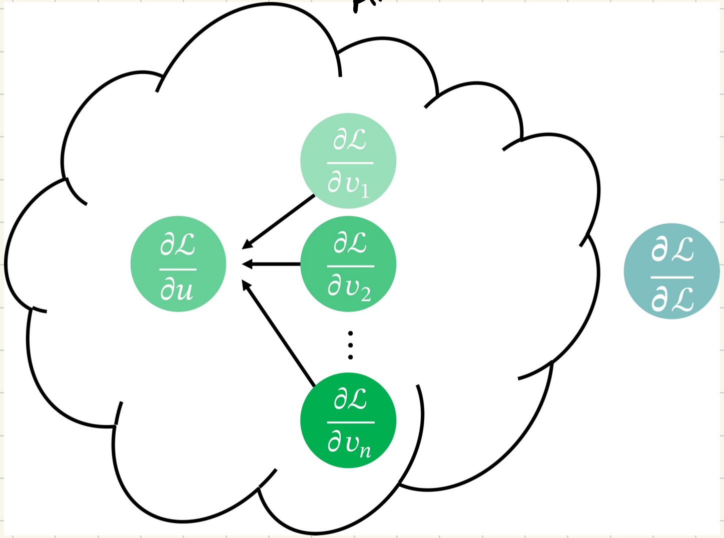
Already know $u_1, \dots, u_d$



$v(u_1, \dots, u_d)$

Backward

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


$\frac{\partial \mathcal{L}}{\partial v_1}$
$\frac{\partial \mathcal{L}}{\partial u}$
$\frac{\partial \mathcal{L}}{\partial v_2}$
$\frac{\partial \mathcal{L}}{\partial v_n}$
$\frac{\partial \mathcal{L}}{\partial \mathcal{L}}$

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

Linear Algebraic View of Backpropagation

Neural nets known since 1950's

Backprop known since 1980'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: Actually want gradient of parameters

$$\frac{\partial \mathcal{L}}{\partial [W]_{i,j}} = \frac{\partial \mathcal{L}}{\partial v_i} \frac{\partial v_i}{\partial [W]_{i,j}} = \frac{\partial \mathcal{L}}{\partial v_i} u_j$$

$$\nabla_W \mathcal{L} = (\nabla_v \mathcal{L}) \, u^T$$

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 their loss landscapes are not convex functions, gradient descent works remarkably well to train neural networks. 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 minimum.