Tuesday, March 10

Midterm 1

Minimum:	46
Median:	74
Mean:	74
Maximum:	99
standard deviation:	16

Grades in canvas!

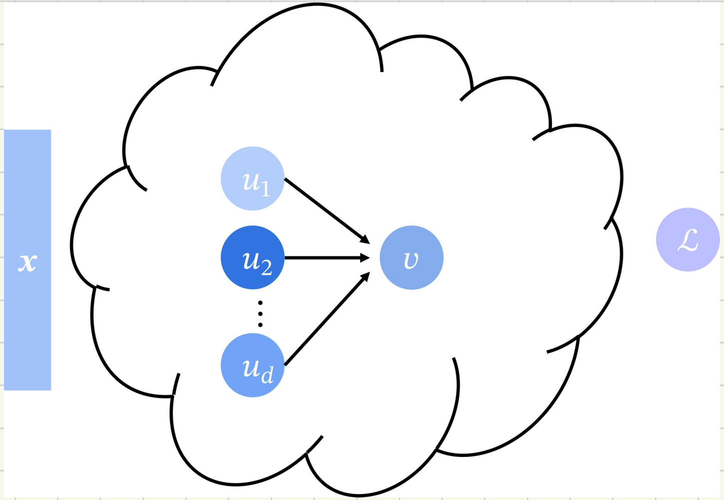
Long tail, will curve eg. 50% to D+/C-

Plan

- Backpropagation
- Convolutional Neural Networks

Forward

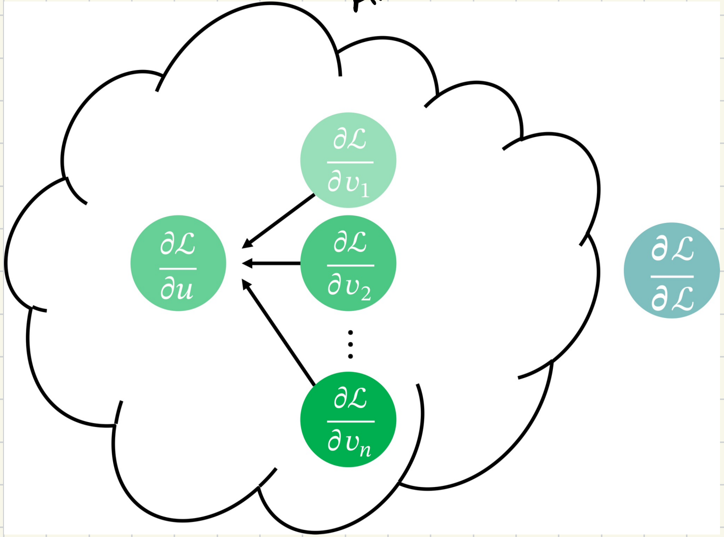
Already know $u_1, \ldots, u_d$



$v(u_1, \ldots, u_d)$

Backward

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



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

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

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

$v = Wu$

$$v_i = \sum_{j=1}^{d} [W]_{i,j} u_j$$

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

Backward

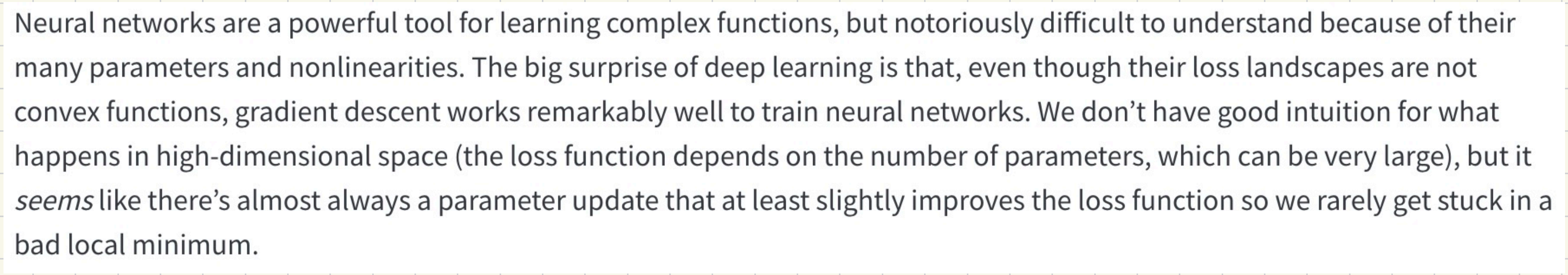
$\nabla_u \mathcal{L} = W^T \nabla_v \mathcal{L}$

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

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.

Motivating Convolutional Layers

Goal: Classification on an image

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}$$

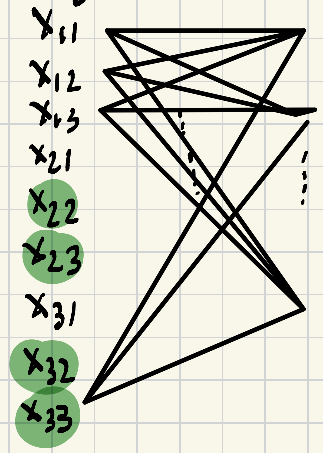
3x3

Each value is color of a pixel

We would like:

1. Fewer parameters in layer
2. Preserve context
3. Shift invariance

Using linear layer...



Q: If 400x400 pixels, Linear layer where input has same size as output, how many parameters?

Convolutional Neural Network

☑ locality preserving

☑ shift invariant

"kernel"/weights

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} \star \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} = \begin{bmatrix} w_{11}x_{11} + w_{12}\cdot x_{12} + x_{21}w_{21} & w_{11}x_{12} + w_{12}x_{13} + w_{21}x_{22} + w_{22}x_{23} \\ \dots & \dots \end{bmatrix}$$

Q: $$\begin{bmatrix} 1 & 1 & 1 \\ -1 & -1 & -1 \\ 1 & 1 & 1 \end{bmatrix} \star \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$$

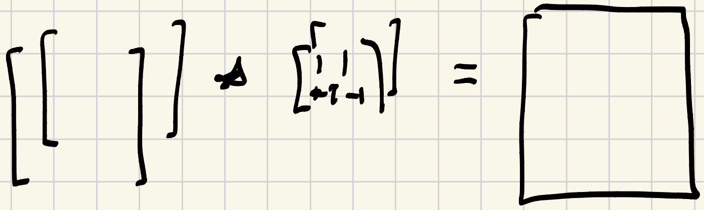
$$\star \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}$$

Q: If 400 x 400 pixels, convolutional layer with d x d kernel, how many parameters?

CNNs continued:

ezyang.github.io/convolution-visualizer/

- Padding: add variables around edges
- Strides: distance between kernel applications
- Pooling: aggregate values
- Channels and 3d convolution:



- Transposed Convolution: <u>Increase</u> dimension

$$\overset{\text{Input}}{\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}} \star^T \overset{\text{kernel}}{\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}} = \begin{bmatrix} 0 & 0 & \\ 0 & 0 & \\ & & \end{bmatrix} + \begin{bmatrix} & 1 & 1 \\ & -1 & -1 \\ & & \end{bmatrix} + \begin{bmatrix} & & \\ 2 & 2 & \\ -2 & -2 & \end{bmatrix} + \begin{bmatrix} & & \\ & 3 & 3 \\ & -3 & -3 \end{bmatrix}$$

Q: How do we train?

Thursday, March 12

- Happy spring break!
- No OH Monday 3/23

Plan

Self-attention

We process

↳ "rich" features with linear layers

↳ "local" features (image/audio/video) with convolutional layers

Q: How do we process sequential data such as text?

↑ size of text changes eg sentence, paragraph, page

Goals:

↳ Decompose text into tokens

↳ Represent tokens as vectors

↳ Process "similar" tokens together

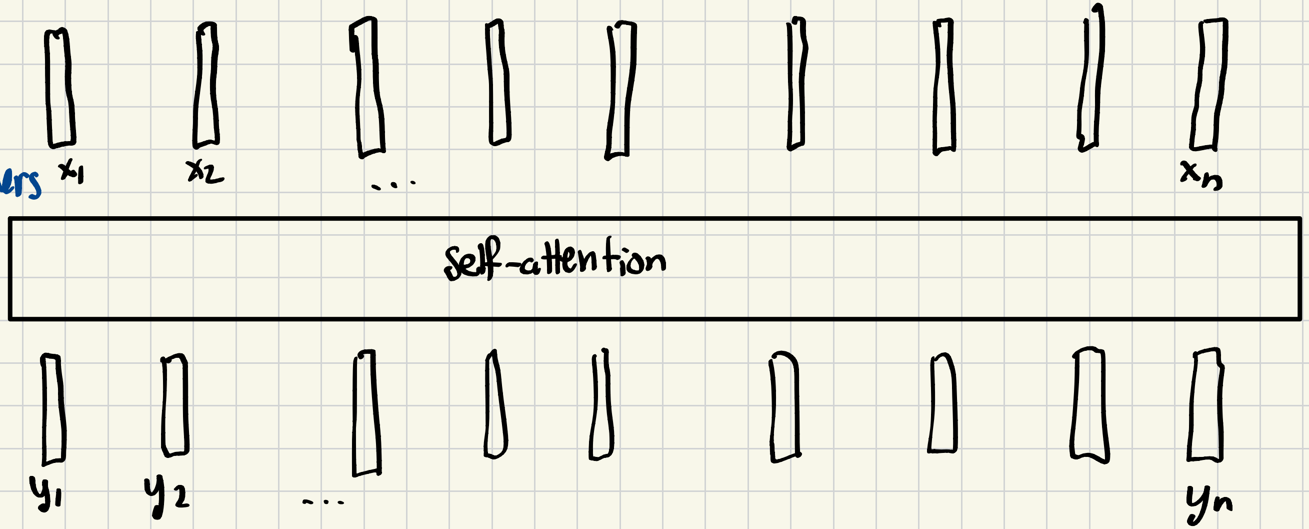
↳ Encode order of tokens

Input: The weather today is warm and sunny.

Tokens: "the" "weather" "today" "is" "warm" "and" "sun" "ny" "."

Vectors:

using autoencoders



Notice: same number of inputs and outputs

Intuition

Write y_i as linear combination of x_j weighted by similarity.

$$y_i = \sum_{j=1}^{n} \underset{\substack{\uparrow \\ \text{similarity} \\ \text{between} \\ x_i \text{ and } x_j}}{\langle x_i, x_j \rangle} \underset{\substack{\uparrow \\ \text{input vectors}}}{x_j} = \sum_{j=1}^{n} \langle q_i, k_j \rangle v_j$$

Notice each x_k appears in three different places:

- ↪ Key
- ↪ query
- ↪ value

Where are the parameters??

$$q_i = W_q x_i \qquad X \in \mathbb{R}^{n \times d}$$

$$k_j = W_k x_j$$

$$v_j = V_j x_j$$

$$Q = X\underset{d\times d}{W_q}, \quad K = X\underset{d\times d}{W_k}, \quad V = X\underset{d\times d}{W_v}$$

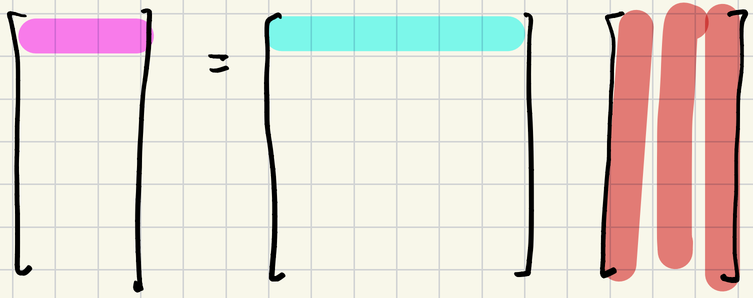
$$\underset{n\times n}{S} = QK^T = XW_qW_k^TX^T$$

$$[S]_{i,j} = \langle q_i, k_j \rangle$$

$$\underset{\text{"scores"}}{S} = \begin{bmatrix} \langle q_1, k_1\rangle & \langle q_1, k_2\rangle \dots \\ \vdots & \\ & \end{bmatrix} 1/\sqrt{d}$$

$$\underset{\text{"attention"}}{A} = \begin{bmatrix} \text{softmax}(\langle q_1, k_1\rangle & \langle q_1, k_2\rangle \dots) \\ \text{softmax}(&) \\ \vdots & \end{bmatrix}$$

$$Y = \underset{n\times n}{A}\,\underset{n\times d}{V}$$



$$Y_i = \sum_{j=1}^{n} \text{sim}(i,j)\, V_j$$

Suppose we have three tokens (eg "I love climbing") represented in $\mathbb{R}^2$

$X = \begin{bmatrix} 1 & 1 \\ 0 & -1 \\ 1 & 0 \end{bmatrix}$ $W_q = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$ $W_k = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $W_v = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

$K =$ $V =$ $Q =$

$A =$
↖ no need to simplify

Attention Mask

Goal: We want tokens to only "look back"

$$A = \begin{array}{c|c} & j \\ \hline & \begin{bmatrix} 1 & & & & & \\ .3 & .7 & & & & \\ .1 & .4 & .5 & & & \\ 0 & 0 & .1 & .9 & & \\ 0 & .2 & 0 & .6 & .2 & \\ .9 & 0 & 0 & 0 & 0 & .1 \end{bmatrix} \\ i & \end{array}$$

$[A]_{i,j}$ is how much token i attends to j

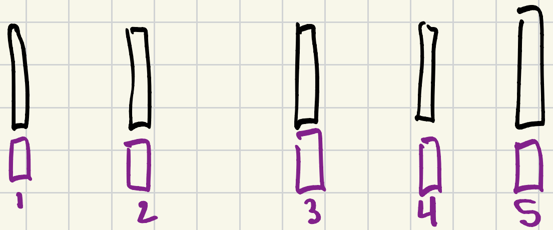
Q: Side Benefit with auto regression?

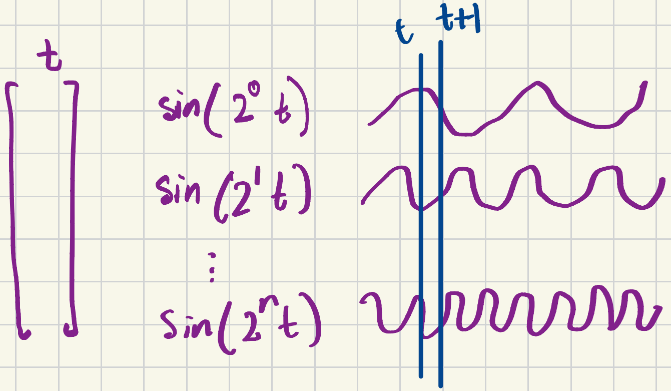
Positional Encoding

"Jack gave water to Jill"

vs.

"Jill gave water to Jack"




t
$\sin(2^0 t)$
$\sin(2^1 t)$
$\vdots$
$\sin(2^n t)$
t
t+1