Guest Lecture: Michael Gelbart (PHD student)

(Visiting Mark's group this year)
mgelbert@seas.harvard.edu

- intro to neural networks from scratch
- next class: deep learning

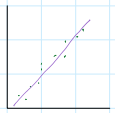
What is a NN or ANN?

$f(x) = 2x$ "function"

$f(x) = ax^b$ "class of functions"

polynomials undetermined number of parameters

- NN is a family similar to this



$\Rightarrow$ fitting (NN does the same)

Terminology

σ : nonlinear element-wise function
"activation function"

$\underline{W}$: "weights" - free parameters (matrix)

b : "biases" - free parameters

x : "units", or "neurons", or "hidden units"
↳ hidden if we treat it like a black box

train/learn/optimize : set $\underline{W}$ and b (to model data well)

supervised learning: learn $y = f(x)$ from labeled data

regression: "learn $\mathbb{R}^7$ to $\mathbb{R}^4$" for example

classification: "What word is it"

size of output layer: size of classes

Neural Networks

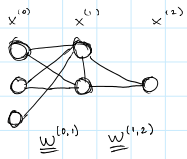
doesn't have to be the same size across layers

$$x^{(j+1)} = \sigma\left(\underline{W}^{(j,j+1)} x^{j} + b^{(j)}\right)$$

*note: we can hide $\underline{b}$ inside $\underline{w}$

"layers" how many times until we hit x^j

Recursive:
affine transformation of x by W and b, then σ, and continue until x^n

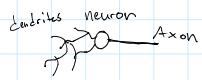


"visualizing matrix multiplication"

how to pick W? next class

On linearity:
"6 parameters"
$f(x) = ab(x + cd)(e + f)$
well not really;
its just some
$f(x) = a'x + b'$

features "learned" by the NN: look at W, which look like filters similar to the brain
"biologically inspired"



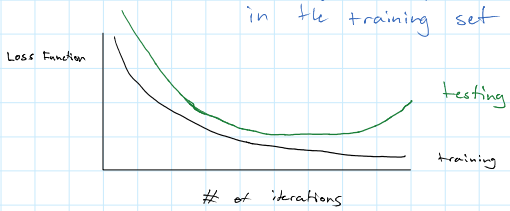
thresholding operation, "firing neurons"

"synaptic weights"

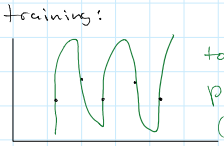
Training: a lot of weights
ie 6 layers, 1000's of weights
1000 x 1000, million parameters: high dimensional
not convex: hard to optimize
(could be in a local optimum)

Overfitting:

"getting good at peculiarities" in the training set

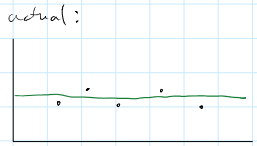


training:



too high polynomial (minimize error)

actual:



Backpropagation

fancy name for taking the derivative

Is an algorithm to compute the gradient of the Loss with respect to the weights, efficiently (making extensive use of chain rule)

we want to minimize the loss

$$z^{(0)} \to a^{(1)} \to z^{(1)} \to a^{(2)} \to z^{(2)} \to L$$

$W^{(0,1)}$ $W^{(1,2)}$

a dependancy graph

$$z_j^n = \sigma(a_j^n)$$
$$a_j^n = \sum W_{ji}^{(n-1,n)} z_i^{(n-1)}$$

$$L = \frac{1}{2}\sum_j (z_j - \hat{z}_j)^2$$ L2 loss

Loss wrt $W_{1\to 2}$

$$\frac{\partial L}{\partial W_{ji}^{(1,2)}} = \frac{\partial L}{\partial a_j^{(2)}} \frac{\partial a_j^{(2)}}{\partial W_{ji}^{(1,2)}}$$

each element of $\underline{a}$ only depends on a particular column of $\underline{W}$

$$\frac{\partial L}{\partial a_j^{(2)}} = \frac{\partial L}{\partial z_j^{(2)}} \frac{\partial z_j^{(2)}}{\partial a_j^{(2)}}$$

$z_i^{(1)}$

derivative of L2

$z_j^{(2)} - \hat{z}_j$ $\sigma(a_j^{(2)})$

we need the intermediate states

$$\therefore \frac{\partial L}{\partial W_{ji}^{(1,2)}} = (z_j^{(2)} - \hat{z}_j)\, \sigma'(a_j^{(2)})\, z_i^{(1)}$$

Loss wrt $W_{0\to 1}$

$$\frac{\partial L}{\partial W_{ji}^{(0,1)}} = \frac{\partial L}{\partial a_j^{(1)}} \frac{\partial a_j^{(1)}}{\partial W_{ji}^{(0,1)}} = z_i^{(0)} \frac{\partial L}{\partial z_j^{(1)}} \frac{\partial z_j^{(1)}}{\partial a_j^{(1)}}$$

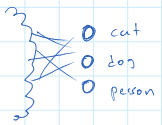
$$= z_i^{(0)} \sigma'(a_j^{(1)}) \frac{\partial L}{\partial z_j^{(1)}}$$

$$= z_i^{(0)} \sigma'(a_j^{(1)}) \sum_k \frac{\partial a_k^{(2)}}{\partial z_j^{(1)}} \frac{\partial L}{\partial a_k}$$

all the $\vec{a}$ depends on element of z.

$$= z_i^{(0)} \sigma'(a_j^{(1)}) \sum_k W_{kj}^{(1,2)} \underbrace{(z_k^{(2)} - \hat{z}_k)\, \sigma'(a_k^{(2)})}_{\text{reuse, and recursive}}$$

Generally trained with gradient descent, and some "learning rate"



$\Rightarrow$ normalize to probability