Logistic Regression
Multinomial Regression

CSE 253

Neural Networks

Lecture 2

Summary so far…

- Last time, we showed how the delta rule for linear regression can be derived by:
  1. Coming up with an appropriate loss function: Mean Squared Error
  2. Taking the derivative with respect to the weights
  3. Plugging the result into the formula for gradient descent

Clicker question

The general idea of gradient descent is:

A) Using calculus to figure out how to change the parameters to go *downhill* in some objective function

B) Using calculus to figure out how to change the parameters to go *uphill* in some objective function

C) To use the delta rule to change the weights.

D) To use calculus to take the derivative of the objective function with respect to the parameters, set it to 0, and solve.

Clicker question

The general idea of gradient descent is:

A) **Using calculus to figure out how to change the parameters to go *downhill* in some objective function**

B) Using calculus to figure out how to change the parameters to go *uphill* in some objective function

C) To use the delta rule to change the weights.

D) To use calculus to take the derivative of the objective function with respect to the parameters, set it to 0, and solve.

Clicker question

- The delta rule is:

A) $$\sum_{n=1}^{N}(t^n - y^n)^2 = \sum_{n=1}^{N}(\delta^n)^2$$

B) $w_i = w_i - \alpha \frac{\partial J}{\partial w_i}$, where J is some objective function

C) $$w_i = w_i - \alpha(t^n - y^n)x_i^n$$

D) $$w_i = w_i + \alpha(t^n - y^n)x_i^n$$

Clicker question

- The delta rule is:

A) $$\sum_{n=1}^{N}(t^n - y^n)^2 = \sum_{n=1}^{N}(\delta^n)^2$$

B) $$w_i = w_i - \alpha\frac{\partial J}{\partial w_i}, \text{ where } J \text{ is some objective function}$$

C) $$w_i = w_i - \alpha(t^n - y^n)x_i^n$$

D) $$\boxed{w_i = w_i + \alpha(t^n - y^n)x_i^n}$$

Clicker question

- The logistic function is:

A) $MSE = \frac{1}{N}\sum_{n=1}^{N}(t^n - y^n)^2$

B) $y(a) = g(a) = \frac{1}{1+e^{-a}}$

C) $w_i = w_i + \alpha(t^n - y^n)x_i^n$

D) $y_k = g(a_k) = \frac{e^{a_k}}{\sum_{j=1}^{c} e^{a_j}}$

Clicker question

- The logistic function is:

A) $MSE = \frac{1}{N}\sum_{n=1}^{N}(t^n - y^n)^2$

B) $\boxed{y(a) = g(a) = \frac{1}{1+e^{-a}}}$

C) $w_i = w_i + \alpha(t^n - y^n)x_i^n$

D) $y_k = g(a_k) = \frac{e^{a_k}}{\sum_{j=1}^{c} e^{a_j}}$

Summary so far…

- Last time, we derived the closed-form formula for linear regression by assuming the goal was to minimize SSE.
- We also showed how the delta rule for linear regression can be derived by:
  1. Using the same loss function
  2. Taking the derivative of it with respect to the weights
  3. Plugging the result into the formula for gradient descent
  4. Out pops the delta rule!

Summary so far…

- We also looked at perceptrons (a binary threshold unit)

$$y = \begin{cases} 1 & \text{if } y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

- Described what they can compute
- Showed how they could be trained by the same (delta) rule!

Summary so far...

- We then considered *linear discriminants*.
- Instead of:

$$y = \begin{cases} 1 & \text{if } y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

- We use: decide that **x** is in Category C_1 if $y(\mathbf{x}) \geq 0$ else **x** is in C_2
- This can then be generalized to multiple categories if we have a linear discriminant for each category, i.e., $y_k(\mathbf{x})$ for category k, and

$$\mathbf{x} \text{ is assigned to class } C_k \text{ if } k = \operatorname*{argmax}_j y_j(\mathbf{x})$$

Today

- First, some motivation for why we use the logistic activation function.
- What happens if we try to use Mean Squared Error for ***logistic regression***?
  1. We start with Mean Squared Error
  2. Take the derivative with respect to the weights
  3. Plug the result into the formula for gradient descent
  4. Out pops a ***bad*** learning rule!
- If time, softmax regression…

Today

- **First, some motivation for why we use the logistic activation function.**
- What happens if we try to use Mean Squared Error for ***logistic regression***?
  1. We start with Mean Squared Error
  2. Take the derivative with respect to the weights
  3. Plug the result into the formula for gradient descent
  4. Out pops a ***bad*** learning rule!
- If time, softmax regression…

Logistic Regression:

A generalization of the linear discriminant and the perceptron:

- A monotonic activation function $g()$:

  $$y(x) = g(w^T x + w_0)$$

  $$\text{where } g(x) = \frac{1}{1+e^{-x}}, \text{ the logistic function}$$

  Stop here to plot the logistic

- This is still considered a linear classifier, because since g is monotonic, the boundary will still be linear (even if it "ramps up")

- **To motivate this, imagine two gaussian-distributed categories with equal variance.**

Gaussian probability density functions:

$$p(x \mid C_1) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu_1}{\sigma}\right)^2}$$

$$p(x \mid C_2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu_2}{\sigma}\right)^2}$$

- By Bayes' rule:

$$p(C_1 \mid x) = \frac{p(x \mid C_1)P(C_1)}{p(x)} \text{ and } p(C_2 \mid x) = \frac{p(x \mid C_2)P(C_2)}{p(x)}$$

- And since these have to sum to 1 (if there are only two classes), the denominator is the sum of the numerators:

$$p(C_1 \mid x) = \frac{p(x \mid C_1)P(C_1)}{p(x \mid C_1)P(C_1) + p(x \mid C_2)P(C_2)}$$

A somewhat counterintuitive derivation:

$$p(C_1 \mid x) = \frac{p(x \mid C_1)P(C_1)}{p(x \mid C_1)P(C_1) + p(x \mid C_2)P(C_2)}$$

- Call these terms A and B, we have:

$$p(C_1 \mid x) = \frac{A}{A+B} = \frac{1}{1+\frac{B}{A}} = \frac{1}{1+e^{\ln(\frac{B}{A})}} = \frac{1}{1+e^{-\ln(\frac{A}{B})}} = \frac{1}{1+e^{-\ln\left(\frac{p(x|C_1)P(C_1)}{p(x|C_2)P(C_2)}\right)}} = \frac{1}{1+e^{-a}}$$

$$\text{where } a = \ln\left(\frac{p(x \mid C_1)P(C_1)}{p(x \mid C_2)P(C_2)}\right)$$

Stop here to plot the logistic

- In other words, the probability of class 1 follows a sigmoid as a function of the log ratio of the probability of class C_1 to the probability of class C_2 – the log odds, also called the *logit*

The logistic activation function:

$$p(C_1 \mid x) = \frac{1}{1+e^{-a}} = g(a)$$

$$\text{where } a = \ln\left(\frac{p(x \mid C_1)P(C_1)}{p(x \mid C_2)P(C_2)}\right)$$

Allows us to interpret the output as posterior probabilities – the probability of category C_1 given x.

Note: a can be written as (and there is a generalization to multi-dimensional gaussians
Where w and x are vectors)

$$a = wx + w_0, \text{ where}$$

$$w = \frac{\mu_1 - \mu_2}{\sigma}, \text{ and } w_0 = -\frac{1}{2}\frac{\mu_1^2}{\sigma} + \frac{1}{2}\frac{\mu_2^2}{\sigma} + \ln\frac{P(C_1)}{P(C_2)}$$

Learning

- That's nice, but:
  - We can't assume our data is Gaussian
  - We need to *learn* the weights.
  - There is no closed-form formula for this one!

- What to do?

- Gradient descent!

Gradient descent is like magic!

- It has been used to learn all sorts of things:
  - Machine translation
  - Playing Go
  - Driving cars
  - Programs (!)
  - Face recognition
  - Object recognition
  - Image captioning
  - Generating new images of faces

Today

- First, some motivation for why we use the logistic activation function.
- **What happens if we try to use Mean Squared Error for *logistic regression*?**
    1. We start with Mean Squared Error
    2. Take the derivative with respect to the weights
    3. Plug the result into the formula for gradient descent
    4. Out pops a ***bad*** learning rule!
- If time, softmax regression…

Notation reminder

- The output of the network is $y=g(a)$
- (or $y_k=g(a_k)$ for multiple outputs)
- $g(a)$ is called the *activation function*
- Here, $$g(a) = \frac{1}{1+e^{-a}}$$
- And a is the weighted sum of the inputs (the *net input*): $$a = \sum_{j=0}^{d} w_j x_j$$

Today

- What happens if we try to use Mean Squared Error for ***logistic regression***?
  1. We start with Mean Squared Error
  2. Take the derivative with respect to the weights
  3. Plug the result into the formula for gradient descent
  4. Out pops a ***bad*** learning rule!

Gradient Descent

- Recall the goal of gradient descent is to reduce the Objective function by adjusting the parameters to go downhill in the objective function (A.K.A., Loss, or Error)
- To do this, we move the parameters in the direction of the negative slope:

$$w_i = w_i - \alpha \frac{\partial MSE}{\partial w_i}$$

Gradient Descent

- So, we need to figure out this expression:

$$\frac{\partial MSE}{\partial w_i} = \frac{\partial \frac{1}{2N} \sum_{n=1}^{N} (t^n - y^n)^2}{\partial w_i} = \frac{1}{N} \sum_{n=1}^{N} \frac{\frac{1}{2} \partial (t^n - y^n)^2}{\partial w_i}$$

(the derivative of the sum is the sum of the derivatives)

Gradient Descent

- Let's figure out one element of the sum, so we can drop *n* (for now):

$$\frac{1}{N}\sum_{n=1}^{N}\frac{\frac{1}{2}\partial(t^n - y^n)^2}{\partial w_i} \longrightarrow \frac{\frac{1}{2}\partial(t - y)^2}{\partial w_i}$$

$$\frac{\frac{1}{2}\partial(t - y)^2}{\partial w_i} = \frac{2}{2}(t-y)\frac{\partial(t-y)}{\partial w_i} = (t-y)\left(\frac{\partial t}{\partial w_i} - \frac{\partial y}{\partial w_i}\right)$$

Gradient Descent

- Continuing along...

$$(t-y)\left(\frac{\partial t}{\partial w_i} - \frac{\partial y}{\partial w_i}\right) = (t-y)\left(-\frac{\partial y}{\partial w_i}\right) = \cancel{(t-y)\left(-\frac{\partial a}{\partial w_i}\right)}$$

(because the derivative of a constant is 0)

(because y=g(a)=a)

- WAIT! NO IT ISN'T! We are doing *logistic* regression now!

$$y(a) = g(a) = \frac{1}{1+e^{-a}}$$

Gradient Descent

- Now, we have to apply the *chain rule*

$$(t-y)\left(-\frac{\partial y}{\partial w_i}\right)=(t-y)\left(-\frac{\partial g(a)}{\partial a}\frac{\partial a}{\partial w_i}\right)=(t-y)\left(-g'(a)\frac{\partial a}{\partial w_i}\right)$$

Because y=$g(a)$

Gradient Descent

$$(t-y)\left(-g'(a)\frac{\partial a}{\partial w_i}\right)$$

$$=(t-y)\left(-g'(a)\frac{\partial \sum_{j=0}^{d} w_j x_j}{\partial w_i}\right) \qquad \text{because } a=\sum_{j=0}^{d} w_j x_j$$

$$=(t\text{ - }y)(\text{-}g'(a))x_i$$

Why?

Gradient Descent

- So gradient descent becomes:

$$w_i = w_i - \alpha \frac{\partial MSE}{\partial w_i}$$

$$w_i = w_i - \frac{\alpha}{N} \sum_{n=1}^{N} (t^n - y^n)(-g'(a^n))x_i^n$$

$$w_i = w_i + \frac{\alpha}{N} \sum_{n=1}^{N} (t^n - y^n)g'(a^n)x_i^n$$

Finally, cool fact:

$$\text{if } g(a) = \frac{1}{1+e^{-a}} \text{ then } g'(a) = g(a)(1-g(a))$$

- So gradient descent becomes:

$$= w_i + \alpha \frac{1}{N} \sum_{n=1}^{N} (t^n - y^n) g(a^n)(1-g(a^n)) x_i^n$$

What's the problem? This looks a lot like the delta rule!

Finally, cool fact:

$$\text{if } g(a) = \frac{1}{1+e^{-a}} \text{ then } g'(a) = g(a)(1-g(a))$$

- Thoughts?

$$w_i = w_i + \frac{\alpha}{N}\sum_{n=1}^{N}(t^n - y^n)g(a^n)(1-g(a^n))x_i^n$$

- There's nothing wrong with the math…
- What happens if y=g(a) is close to 0? 1?
- If we are very confident and *wrong*, it is very hard to change our answer!

So...

- Not knowing any better, for *years,* we used this learning rule with logistic outputs.
- This learning rule "works", but it is unnecessarily **slow**, because it can get stuck for a long time in the wrong answer.
- What happens with the learning rule you derived in the homework?
- There's no slope term!

What's going on?

- MSE is the *wrong* objective function for logistic regression!
- The *right* one is cross-entropy, which (as you have shown in your homework), leads to the delta rule.
- In the next lecture (or maybe this one), we'll explain why cross-entropy is The Right Thing To Do..

Summary so far…

- There is a generalization of the perceptron, called *logistic regression*
- It replaces the threshold function by a sigmoid:

$$y(a) = g(a) = \frac{1}{1 + e^{-a}}$$

…which allows us to go from a 0/1 classifier to a classifier that gives us a *probability* of category membership.

Summary so far…

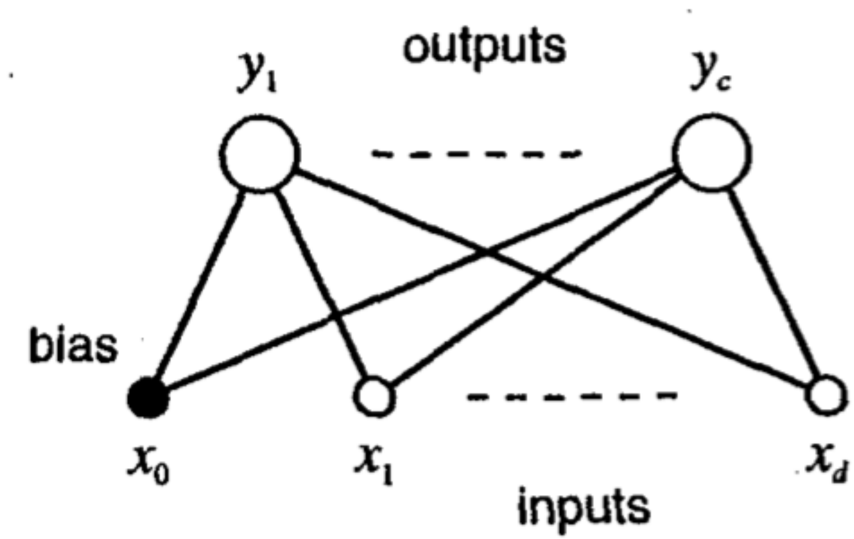
- For logistic regression, there is no closed form formula for the weights like linear regression.
- ➔ We have to use gradient descent to learn the weights.
- Minimizing the Mean Squared Error leads to a poor learning rule for logistic regression.
- Cross-entropy leads to a good learning rule for logistic regression.

Today

- First, some motivation for why we use the logistic activation function.
- What happens if we try to use Mean Squared Error for ***logistic regression***?
  1. We start with Mean Squared Error
  2. Take the derivative with respect to the weights
  3. Plug the result into the formula for gradient descent
  4. Out pops a ***bad*** learning rule!
- **If time, softmax regression…**

A generalization of logistic regression to multiple categories

The picture to have in your mind:



But now we will use a different activation function that turns the outputs into probabilities

Softmax: A generalization of the logistic function to multiple categories

$$a_k = \sum_{j=0}^{d} w_{jk} x_j = w_k^T x$$

Here, x_0=1, and w_{0k} is the *bias*.
Weights now need *two* indices

$$y_k = g(a_k) = \frac{e^{a_k}}{\sum_{j=1}^{c} e^{a_j}} > 0$$

This is the *softmax activation function*.
Note it is positive – no matter what a_k is.

$$\sum_{k=1}^{c} y_k = \sum_{k=1}^{c} \frac{e^{a_k}}{\sum_{j=1}^{c} e^{a_j}} = \frac{\sum_{k=1}^{c} e^{a_k}}{\sum_{j=1}^{c} e^{a_j}} = 1$$

And the outputs sum to 1

This is also known as the *softmax distribution*

A generalization of logistic regression to multiple categories

Recall we can think of the logistic as

$$P(C_1 \mid x) = y = g(a) = \frac{1}{1+e^{-a}}$$

Now we can think of the softmax as:

$$P(C_k \mid x) = y_k = g(a_k) = \frac{e^{a_k}}{\sum_{j=1}^{c} e^{a_j}}$$

A generalization of logistic regression to multiple categories

Now we can think of the softmax as:

$$P(C_k \mid x) = y_k = g(a_k) = \frac{e^{a_k}}{\sum_{j=1}^{c} e^{a_j}}$$

And as you know from your homework, this network can *also* be trained by the delta rule.

I won't derive it now because it will be part of your homework in the future...

Summary so far…

- We've seen four kinds of "neural networks":

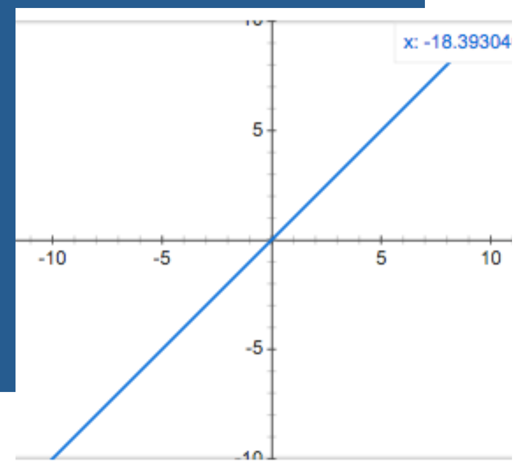
1. Linear networks (linear regression):

$$a_{out} = \sum_{j=0}^{d} w_j x_j = w^T x \text{ (here, } x_0 = 1,\ w_0 \text{ is the } \textit{bias})$$

The output, $y = g(a) = a$

(the activation function,

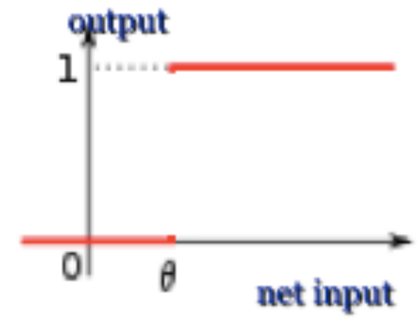
g is the identity fn)



Summary so far…

- We've seen four kinds of "neural networks":

2. Perceptrons:



$$a_{out} = \sum_{j=0}^{d} w_j x_j = w^T x \text{ (here, } x_0 = 1,\ w_0 \text{ is the } \textit{bias}\text{)}$$

$$\text{The output, } y = g(a) = \begin{cases} 1 & \text{if } a \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Where y=1 means x is in Category C_1, else C_2

Summary so far…

- We've seen four kinds of "neural networks":
- And perceptrons can be considered to be a ***linear discriminant***

$$a_{out} = \sum_{j=0}^{d} w_j x_j = w^T x \text{ (here, } x_0 = 1,\ w_0 \text{ is the } \textit{bias}\text{)}$$

The choice: if $a \geq 0$ then $\mathrm{x} \in \mathrm{C}_1$;

else $\mathrm{x} \in \mathrm{C}_2$

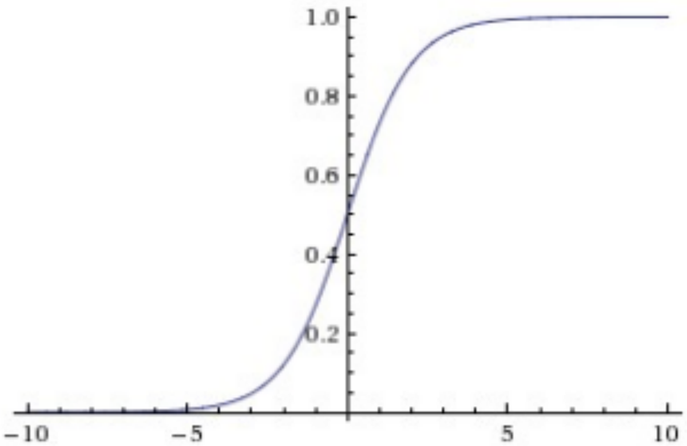
Summary so far…

- We've seen four kinds of "neural networks":

3. Logistic regression:

$$a = \sum_{j=0}^{d} w_j x_j = w^T x \ (\text{here}, x_0 = 1,\ w_0 \text{ is the } \textit{bias})$$

$$P(C_1 \mid x) = y = g(a) = \frac{1}{1 + e^{-a}}$$



Summary so far…

- We've seen four kinds of "neural networks":

4. Softmax regression

$$a_k = \sum_{j=0}^{d} w_{jk} x_j = w_k^T x \text{ (here, } x_0 = 1,\ w_0 \text{ is the } \textit{bias}\text{)}$$

$$P(C_k \mid x) = y_k = g(a_k) = \frac{e^{a_k}}{\sum_{j=1}^{c} e^{a_j}}$$

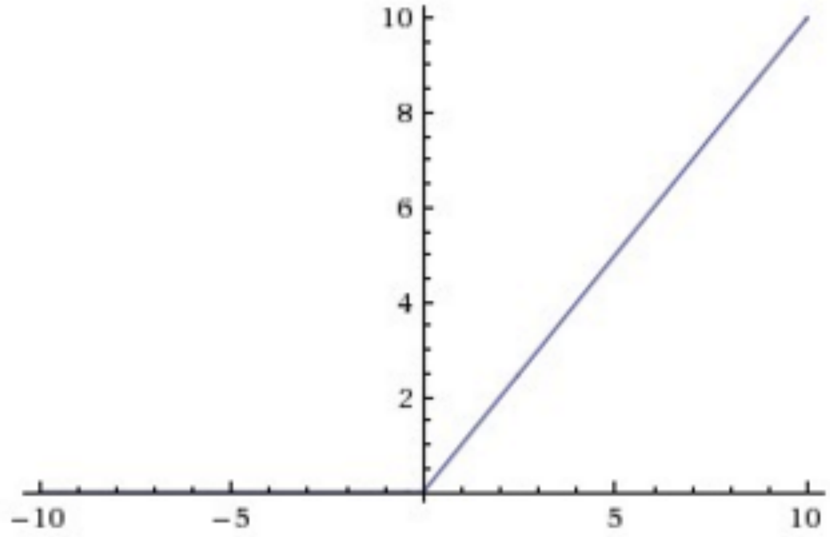
Summary so far…

And they ALL can be trained by the delta rule!

More activation functions

Rectified Linear Units (ReLU):

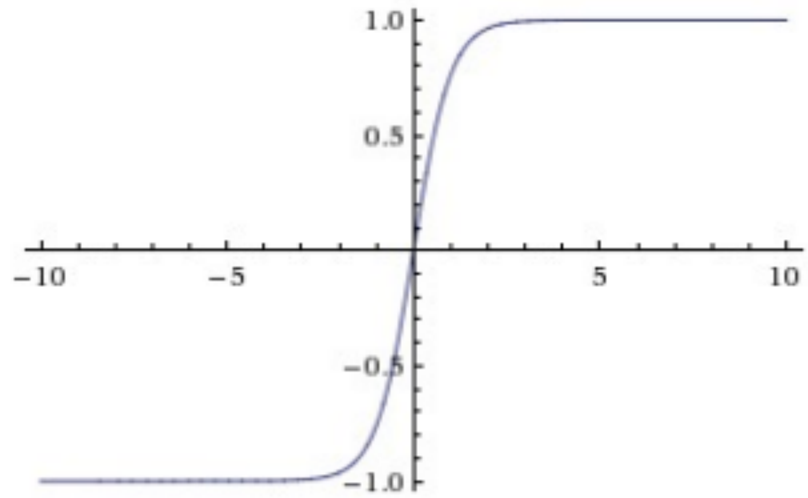
$$y = g(a) = \max(a, 0)$$



More activation functions

Tanh:

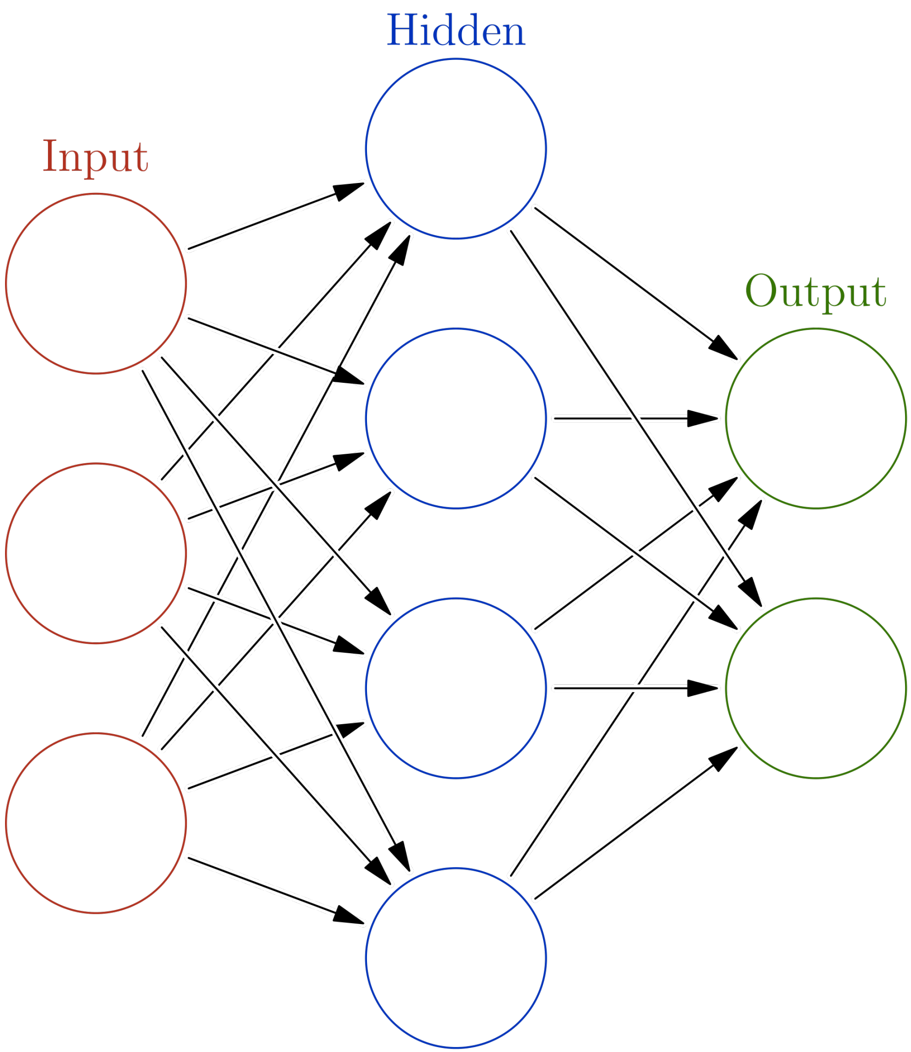
$$y = g(a) = \tanh(a)$$



Forward Propagation with a hidden layer

Forward propagation consists of applying an activation function at each layer of the network.

The hiddens could be ReLU – the outputs softmax.

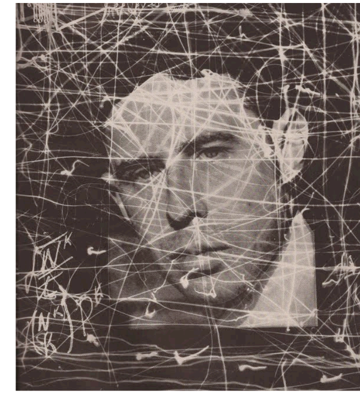


Summary

- There are theoretical ways to motivate perceptrons, linear regression, logistic regression, and softmax regression that lead to the activation functions we saw.
- We will talk more about this next time
- Some activation functions, like ReLU and tanh, are motivated by how well they work in a neural network – i.e., empirically.
- We will talk about that in the weeks to come.

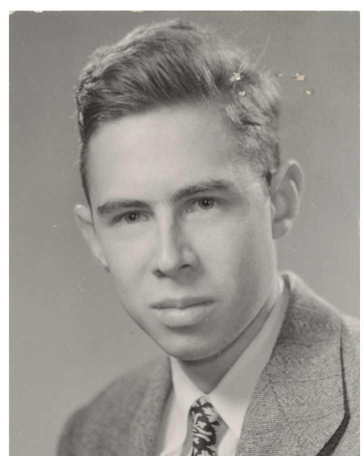
Summary

- Finally, I showed that this guy:

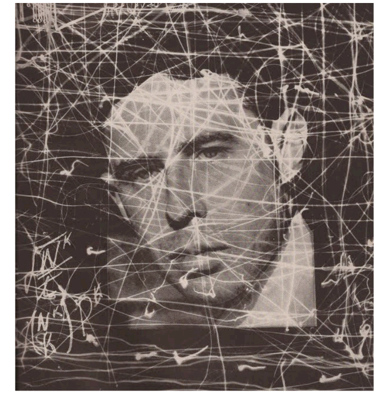


- Who is also this guy:

- And this guy:



Summary



- Finally, I showed that this guy:

- Is *not* Geoff Hinton,

- Or Yann LeCun!