



S&DS 265 / 565  
Introductory Machine Learning

# **Neural Networks (continued)**

November 15

Yale

# Reminders

- Quiz 5 this Thursday, November 17: Topic models, neural nets
- Assn 5 is out; start early! Due Dec. 1

# Last time

- Basic architecture of feedforward neural nets
- Biological analogy and inspiration
- Backpropagation — high level

# Today

- Examples: Regression, Tensorflow
- Backpropagation — more detail
- Examples: Classification

# Starting with regression

For linear regression, our loss function for an example  $(x, y)$  is

$$\begin{aligned}\mathcal{L} &= \frac{1}{2}(y - \beta^T x - \beta_0)^2 \\ &= \frac{1}{2}(y - f)^2\end{aligned}$$

where  $f(x) = \beta^T x + \beta_0$ .

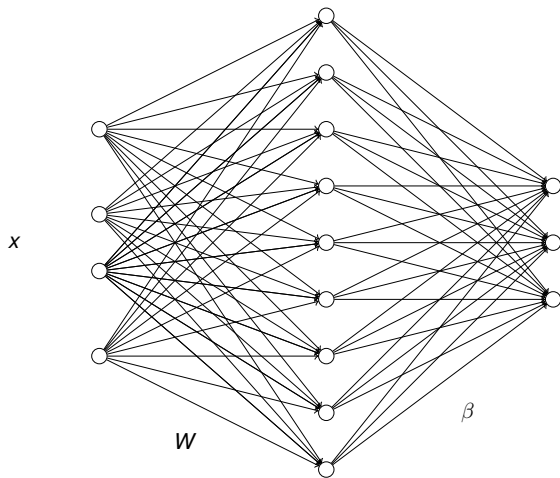
# Adding a layer

Loss is

$$\mathcal{L} = \frac{1}{2}(y - f(x))^2$$

where now  $f(x) = \beta^T h(x) + \beta_0$  where  $h(x) = Wx + b$ .

This can be viewed graphically.



# Equivalent to linear model

But this is just a linear model

$$f = \tilde{\beta}^T x + \tilde{\beta}_0$$

We get a reparameterization of a linear model; nothing new.

Need to add *nonlinearities*



# Nonlinearities

Add nonlinearity

$$h = \varphi(Wx + b)$$

*fixed* and *applied component-wise*

Typically the last layer is just linear (for both classification and regression):

$$f = \beta^T h + \beta_0$$

# Nonlinearities

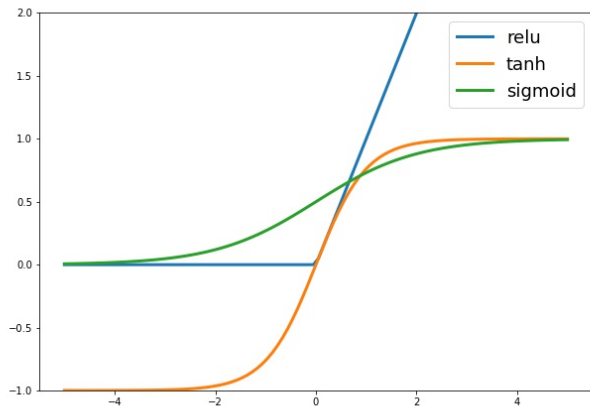
Commonly used nonlinearities:

$$\varphi(u) = \tanh(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$$

$$\varphi(u) = \text{sigmoid}(u) = \frac{1}{1 + e^{-u}}$$

$$\varphi(u) = \text{relu}(u) = \max(u, 0)$$

# Nonlinearities



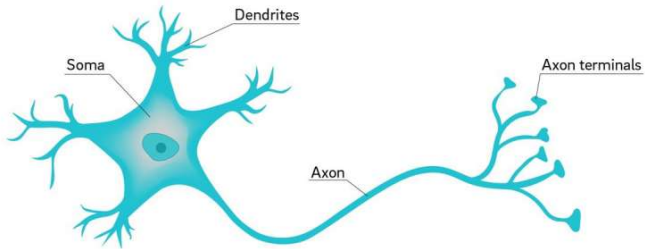
# Nonlinearities

So, a neural network is nothing more than a parametric regression model with a restricted type of nonlinearity

Why are they called neural networks?

# Biological Analogy

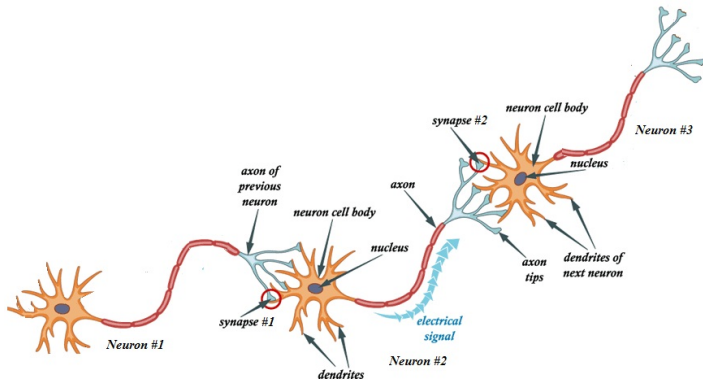
## Neuron

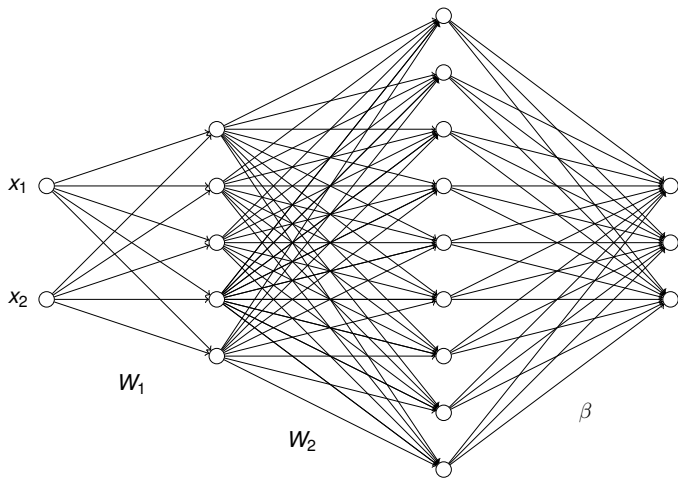


# Biological Analogy

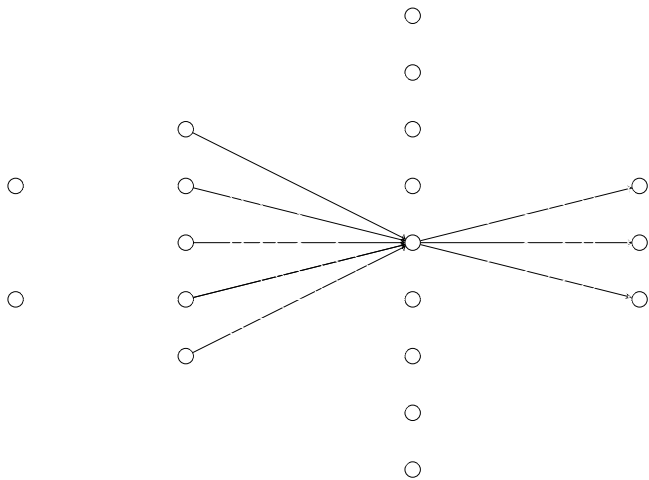
- The dendrites play the role of inputs, collecting signals from other neurons and transmitting them to the soma, which is the “central processing unit.”
- If the total input arriving at the soma reaches a threshold, an output is generated.
- The axon is the output device, which transmits the output signal to the dendrites of other neurons.

# Biological Analogy









# Sanity check

Let's work through a toy example to be sure we understand the “mechanics” of the neural net

2-dimensional inputs  $x = (x_1, x_2)^T$  and a binary classification

We have 2-layer neural net with three hidden neurons

$$h(x) = \varphi(Wx + b)$$

$$\beta^T h(x) = \log \left( \frac{p(Y = 1 | x)}{p(Y = 0 | x)} \right)$$

with  $\varphi(u) = \text{relu}(u) = \max(u, 0)$

# Sanity check

Let's work through a toy example to be sure we understand the “mechanics” of the neural net

2-dimensional inputs  $x = (x_1, x_2)^T$  and a binary classification

Suppose that the parameters (weights) of the network are

$$\begin{aligned}b &= (1, 0, -1)^T \\ W &= \begin{pmatrix} 1 & -1 \\ -2 & 1 \\ 0 & 1 \end{pmatrix} \\ \beta &= (1, 2, -1)^T\end{aligned}$$

What is the prediction of the network if  $x = (1, -1)^T$  or  $x = (-1, 1)^T$ ?

# Sanity check

Let's work through a toy example to be sure we understand the “mechanics” of the neural net

2-dimensional inputs  $x = (x_1, x_2)^T$  and a binary classification

Suppose that the parameters (weights) of the network are

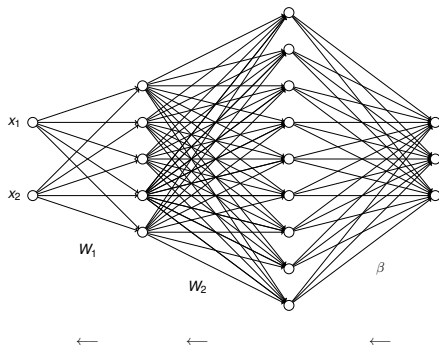
$$\begin{aligned}b &= (1, 0, -1)^T \\ W &= \begin{pmatrix} 1 & -1 \\ -2 & 1 \\ 0 & 1 \end{pmatrix} \\ \beta &= (1, 2, -1)^T\end{aligned}$$

How do the predictions change if  $\varphi = \tanh$ ?

# Training

- The parameters are trained by stochastic gradient descent.
- To calculate derivatives we just use the chain rule, working our way backwards from the last layer to the first.

# High level idea



Start at last layer, send error information back to previous layers

# Backprop calcs

We'll go through some of the backpropagation calculations. It's not essential that you can repeat all of these (but try!)

# Start simple

Loss is

$$\mathcal{L} = \frac{1}{2}(y - f)^2$$

The change in loss due to making a small change in output  $f$  is

$$\frac{\partial \mathcal{L}}{\partial f} = (f - y)$$

We now send this backward through the network



# Start simple

Loss is

$$\mathcal{L} = \frac{1}{2}(y - f)^2$$

Now suppose that  $f = ab$ :

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial a} &= \frac{\partial \mathcal{L}}{\partial f} \frac{\partial f}{\partial a} \\ &= \frac{\partial \mathcal{L}}{\partial f} \cdot b \\ &= (f - y) \cdot b\end{aligned}$$

# Start simple

Loss is

$$\mathcal{L} = \frac{1}{2}(y - f)^2$$

Now suppose that  $f = ab$ :

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial b} &= \frac{\partial f}{\partial b} \frac{\partial \mathcal{L}}{\partial f} \\ &= a \cdot \frac{\partial \mathcal{L}}{\partial f} \\ &= a \cdot (f - y)\end{aligned}$$

# Fancy verison

We need a matrix version of this. If  $A = BC$ , then

$$\frac{\partial \mathcal{L}}{\partial B} = \frac{\partial \mathcal{L}}{\partial A} C^T$$

$$\frac{\partial \mathcal{L}}{\partial C} = B^T \frac{\partial \mathcal{L}}{\partial A}$$

Check that the dimensions match up!

# Example

So if  $f = Wx + b$  then

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W} &= \frac{\partial \mathcal{L}}{\partial f} x^T \\ &= (f - y) x^T\end{aligned}$$

# Example

So if  $f = Wx + b$  then

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial b} &= \frac{\partial \mathcal{L}}{\partial f} \\ &= (f - y)\end{aligned}$$

# Two layers

Now add a layer:

$$f = W_2 h + b_2$$

$$h = W_1 x + b_1$$

Then we have

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W_2} &= \frac{\partial \mathcal{L}}{\partial f} h^T \\ &= (f - y) h^T\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial h} &= W_2^T \frac{\partial \mathcal{L}}{\partial f} \\ &= W_2^T (f - y)\end{aligned}$$

## Two layers

Now send this back (backpropagate) to the first layer:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W_1} &= \frac{\partial \mathcal{L}}{\partial h} x^T \\ &= W_2^T \frac{\partial \mathcal{L}}{\partial f} x^T \\ &= W_2^T (f - y) x^T\end{aligned}$$

# Adding a nonlinearity

Remember, this just gives a linear model! Need a nonlinearity:

$$h = \varphi(W_1 x + b_1)$$

$$f = W_1 h + b_2$$



# Adding a nonlinearity

If  $\varphi(u) = \text{relu}(u) = \max(u, 0)$  then this just becomes

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W_1} &= \mathbb{1}(h > 0) \frac{\partial \mathcal{L}}{\partial h} x^T \\ &= \mathbb{1}(h > 0) W_2^T \frac{\partial \mathcal{L}}{\partial f} x^T \\ &= \mathbb{1}(h > 0) W_2^T (f - y) x^T\end{aligned}$$

where

$$\mathbb{1}(u) = \begin{cases} 1 & u > 0 \\ 0 & \text{otherwise} \end{cases}$$

See notes on backpropagation for details

# Classification

For classification we use softmax to compute probabilities

$$(p_1, p_2, p_3) = \frac{1}{e^{f_1} + e^{f_2} + e^{f_3}} (e^{f_1}, e^{f_2}, e^{f_3})$$

The loss function is

$$\mathcal{L} = -\log P(y | x) = \log (e^{f_1} + e^{f_2} + e^{f_3}) - f_y$$

So, we have

$$\frac{\partial \mathcal{L}}{\partial f_k} = p_k - \mathbb{1}(y = k)$$

# Examples

Let's go to the notebooks!

# np-complete demo



## Minimal neural network implementation

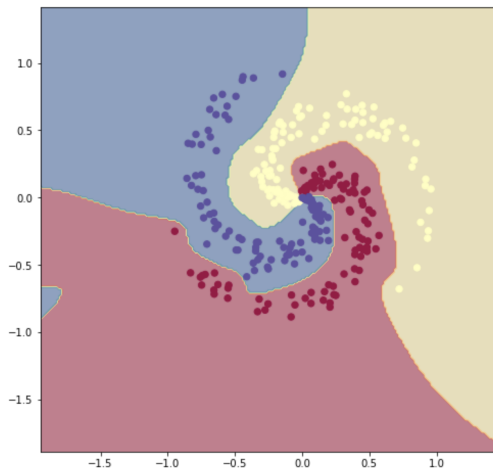
This is a "bare bones" implementation of a 2-layer neural network for classification, using rectified linear units as activation functions. The code is from Andrej Karpathy; please see [this page](#) for an annotated description of the code.

In assignment 7 you will extend this and experiment with some different settings of the parameters.

```
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'
plt.rcParams['axes.facecolor'] = 'lightgray'
```

# np-complete demo



# playground.tensorflow.org

Tinker With a **Neural Network** Right Here in Your Browser.  
Don't Worry, You Can't Break It. We Promise.



Epoch  
000,000

Learning rate  
0.03

Activation  
Tanh

Regularization  
None

Regularization rate  
0

Problem type  
Classification

## DATA

Which dataset do you want to use?



Ratio of training to test data: 50%

Noise: 0

Batch size: 10

## FEATURES

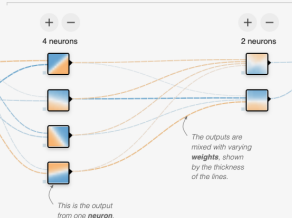
Which properties do you want to feed in?

$X_1$   
 $X_2$   
 $X_1^2$   
 $X_2^2$   
 $X_1 X_2$

## 2 HIDDEN LAYERS

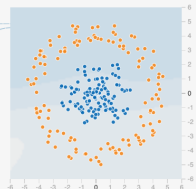
4 neurons

2 neurons



## OUTPUT

Test loss 0.509  
Training loss 0.512



# Summary

- In neural networks, “features” are linear operations followed by an activation function
- Based on a crude analogy with neurons in biological brains
- Neurons are deterministic functions of input; not latent variables
- A special type of (parametric) nonlinear regression model
- Trained using stochastic gradient descent
- Backpropagation is just the chain rule from calculus
- Applied iteratively from the last layer forwards