

S&DS 265 / 565  
**Introductory Machine Learning**

# **Neural Networks**

October 31



# Reminders

- Quiz 4 today
- Assn 4 due next week

# For today

- Finishing up topic models – Gibbs sampling
- Neural networks

# Outline: Neural networks

- Connecting to embeddings and logistic regression
- Alternative view of linear models
- Adding nonlinearities — activation functions
- Biological analogy and inspiration
- Backpropagation — high level
- Examples: Regression

# Logistic Regression

Recall:

$$\log \left( \frac{P(y = 1 | x)}{P(y = 0 | x)} \right) = \beta^T x + \beta_0$$

Equivalently:

$$P(Y = 1 | x) \propto e^{\beta^T x + \beta_0}$$

# Logistic Regression

In the multi-class case we have

$$P(Y = k | x) \propto e^{\beta_k^T x + \beta_{k0}}, \quad k = 1, \dots, K - 1$$

We can write this in ML terminology as

$$\text{Softmax} \left( \left\{ \beta_k^T x + \beta_{k0} \right\} \right)$$

Note: Can also use  $\beta_k$  for  $k = 0, \dots, K - 1$ . This will be “overparameterized”

# Logistic Regression

What if  $x$  is an image, represented as pixels? It might be hard to get an accurate classifier.

Want to learn *feature representation*  $\phi(x)$ .

The model becomes

$$P(Y = k | x) \propto e^{\beta_k^T \phi(x) + \beta_{k0}}, \quad k = 0, 1, \dots, K - 1$$

The parameters of  $\phi$  and the parameters  $\beta$  need to be learned/trained.

# Word embeddings

Applying this to language modeling, we could have a bigram model

The model becomes

$$P(v | w) \propto e^{\beta_v^T \phi(w)}$$

where  $\phi(w)$  is a learned feature vector or “embedding vector” for each word.

The parameters  $\beta_v$  are also embeddings. By making them the same as  $\phi$  we get the word2vec model.

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

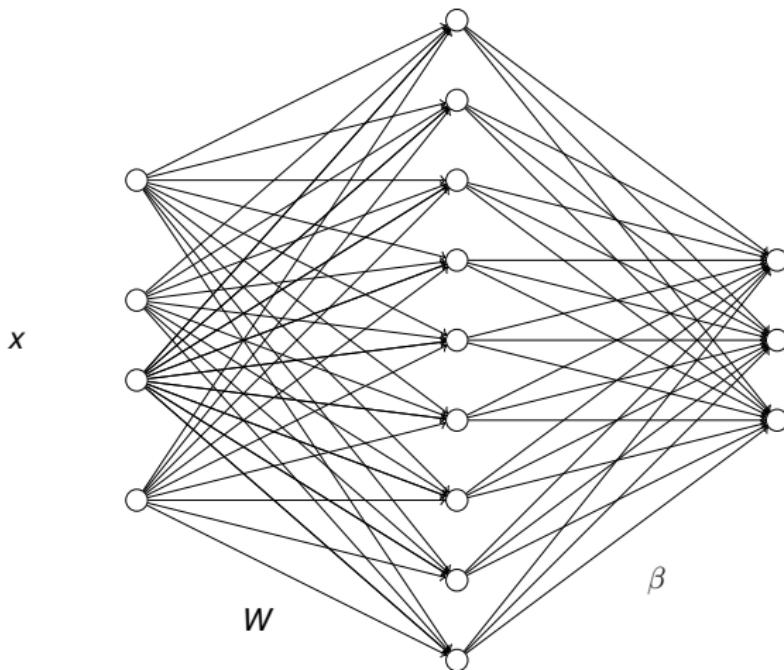
# Adding a layer

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

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

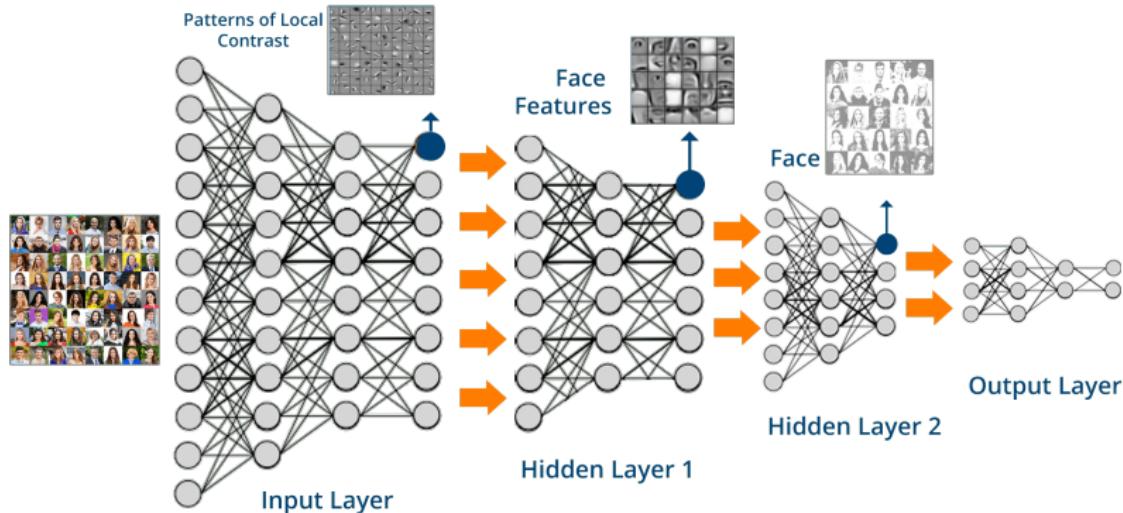
We get a reparameterization of a linear model; nothing new.

Need to add *nonlinearities*

# Neural networks

- Linear models are usually too simple to give good prediction rules
- Nonlinear models allow us to fit the data better—but run the risk of *overfitting*
- Neural networks are kind of in-between linear regression and  $k$ -nearest neighbors
- *Neural networks are layered linear models with carefully restricted nonlinearities*

# Deep neural networks



- Heuristics motivated from simplified view of the brain
- A particular form of nonlinear classification/regression

# Everyday example



Add a Caption

Look Up Australian Shepherd >

Sunday • Oct 20, 2024 • 9:24 AM

IMG\_2224

Adjust



Results



Results

Siri Knowledge

**Berneese mountain dog**  
Dog breed  
[Wikipedia](#)

**Australian Shepherd**  
Dog breed  
[Wikipedia](#)

# Everyday example



The image shows a brown and white dog lying on its back on a green lawn. A circular white overlay with a small silhouette of a dog inside is positioned over the dog's head. Below the image, the word "Results" is centered.

Siri Knowledge

**Pembroke Welsh Corgi**  
Dog breed  
[Wikipedia](#)

**Australian Shepherd**  
Dog breed  
[Wikipedia](#)

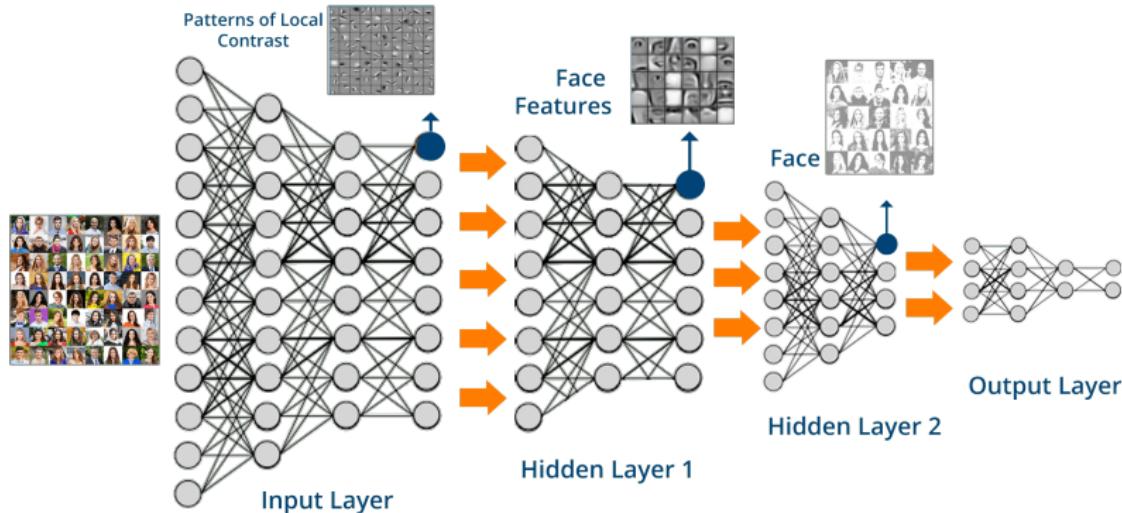
The neural network extracts “features” of the image that can be used to find similar images

# Everyday example

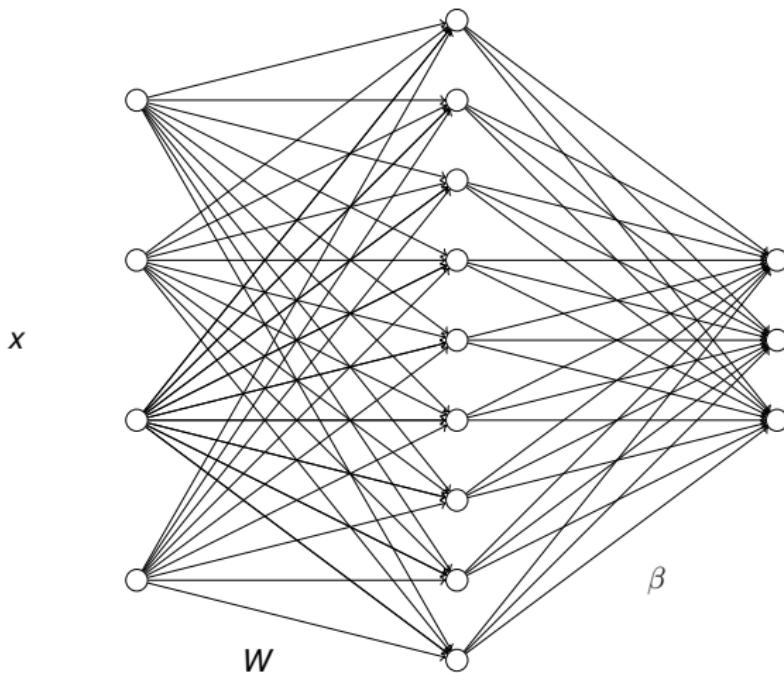


The neural network extracts “features” of the image that can be used to find similar images

# Deep neural networks



- Heuristics motivated from simplified view of the brain
- A particular form of nonlinear classification/regression



# Nonlinearities

Add *fixed nonlinearity*, for example

$$h = \max(Wx + b, 0)$$

*applied component-wise* to each “neuron”

- Typically the last layer is just linear
- So, a neural net is a linear model, but with complex *predictor variables* or “features” that are automatically learned from data

# Nonlinearities

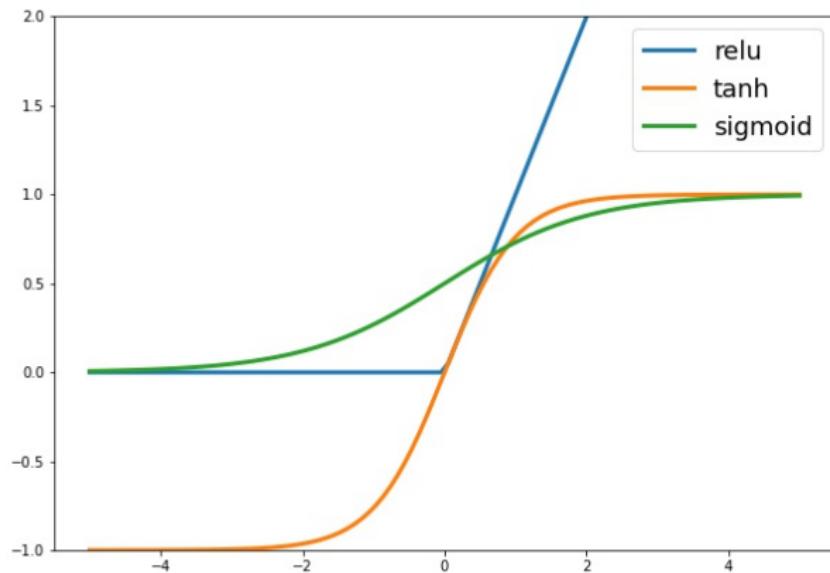
Commonly used nonlinearities:

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

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

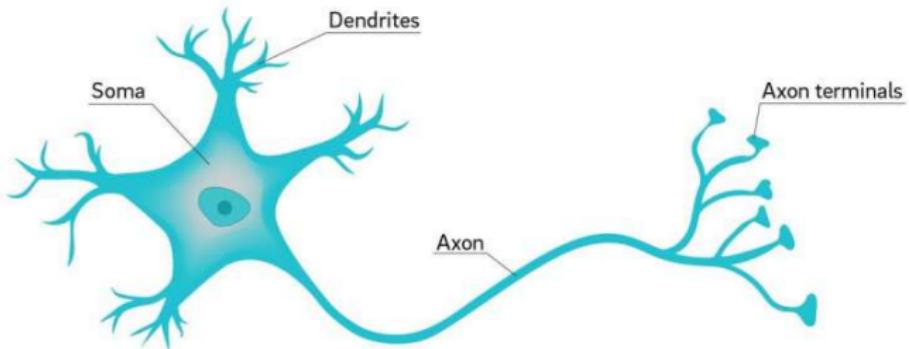
$$\phi(u) = \text{relu}(u) = \max(u, 0)$$

# Nonlinearities



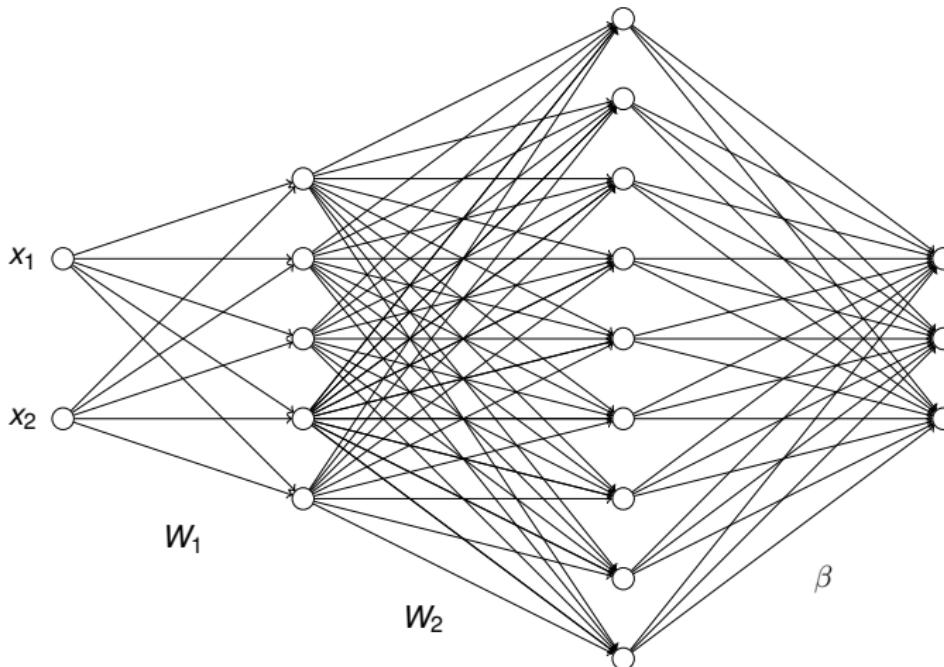
# Biological analogy

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 — this is the nonlinearity!
- The axon is the output device, which transmits the output signal to the dendrites of other neurons.



inputs

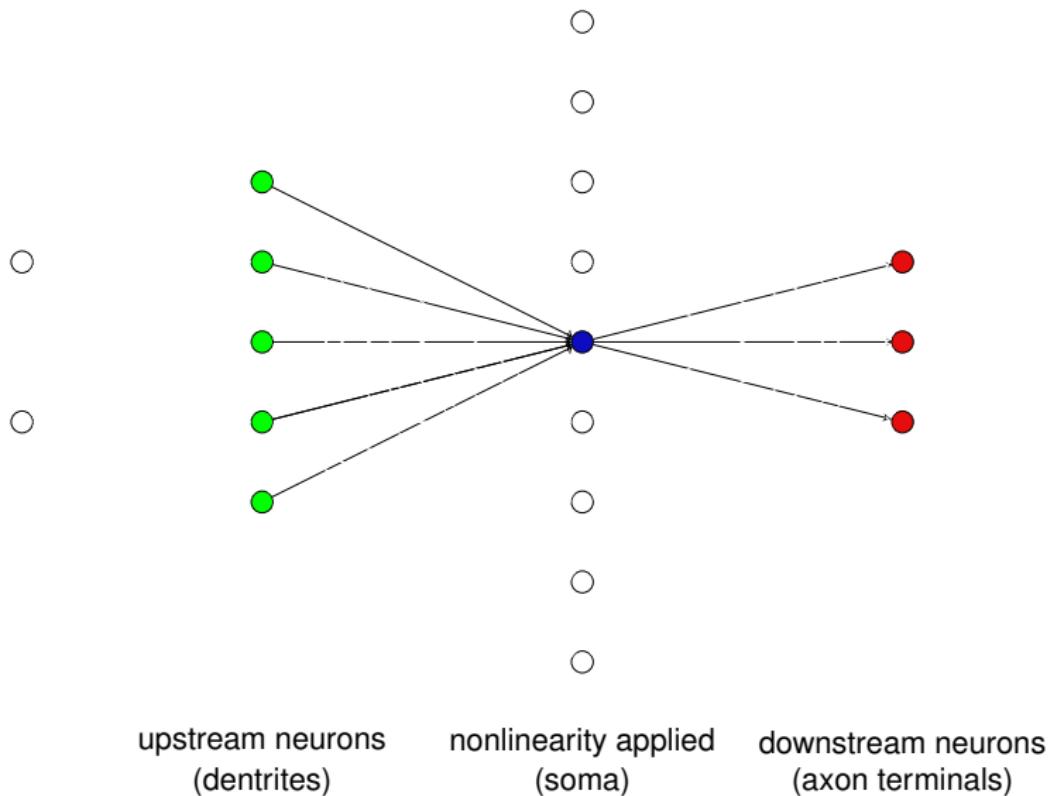
4 neurons

layer 1

9 neurons

layer 2

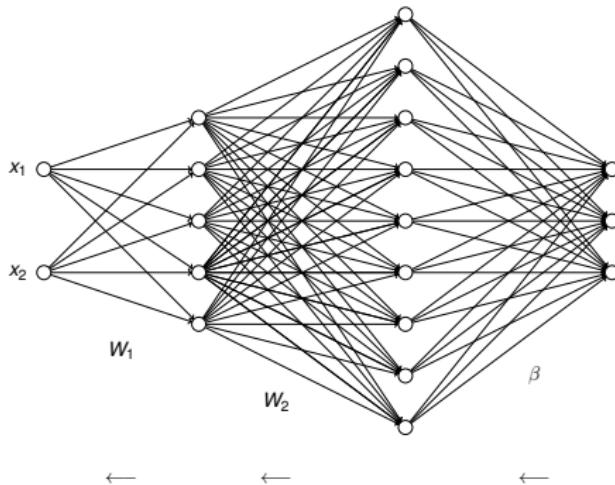
3 outputs



# Training

- The parameters are trained by “stochastic gradient descent”
  - ▶ Stream through the data, using small batches
  - ▶ Add a little bit to each network weight — proportional to how sensitive the predictions are to that weight (derivative)
  - ▶ Derivatives calculated automatically — no math!
  - ▶ Stop if overfitting is detected!

# High level idea: Backpropagation



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

**These types of neural networks are called  
“multilayer perceptrons” (MLPs)**

They can be surprisingly robust to overfitting!

# Demo



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? Which properties do you want to feed in? Ratio of training to test data: 50% Noise: 0 Batch size: 10

FEATURES

2 HIDDEN LAYERS

4 neurons   2 neurons

$x_1$     $x_2$     $x_1^2$     $x_2^2$     $x_1 x_2$

The outputs are mixed with varying weights shown by the thickness of the lines.

This is the output from one neuron. Hover to see it larger.

OUTPUT Test loss 0.524 Training loss 0.517

A screenshot of the TensorFlow playground interface. At the top, there are controls for epoch (000,000), learning rate (0.03), activation function (Tanh), regularization (None), regularization rate (0), and problem type (Classification). Below these are sections for 'DATA' (dataset selection, ratio of training/test data at 50%, noise level at 0, batch size at 10), 'FEATURES' (input variables  $x_1$ ,  $x_2$ ,  $x_1^2$ ,  $x_2^2$ ,  $x_1 x_2$ ), and 'HIDDEN LAYERS' (two layers with 4 and 2 neurons respectively). A note explains that outputs are mixed with varying weights. A callout points to a single neuron's output. To the right is a scatter plot showing two classes of data points (blue and orange) separated by a curved decision boundary. Loss values are displayed as 0.524 (Test loss) and 0.517 (Training loss).

<https://playground.tensorflow.org/>

# Summary

- For complex data we may want to learn features of the inputs
- This representation can be part of a logistic regression model
- Features that are linear transforms followed by a nonlinearity form the building blocks of (artificial) neural networks
- Based on a crude analogy with neurons in biological brains
- Trained using stochastic gradient descent
- Can be thought of as a particular type of nonlinear regression model