



S&DS 365 / 565  
Intermediate Machine Learning

# Neural Networks for Classification

February 9 and 14

Yale

# Reminders

- Assignment 1 will be posted later today
- Quiz 1 will be Wednesday, Feb 16; material up to Feb 14
- Check Canvas/EdD for office hours

# Today: RKHS and overview of neural nets

- Discussion of RKHS concepts
- Basic architecture of feedforward neural nets
- Backpropagation
- Examples from TensorFlow
- We'll assume some familiarity with these ideas

# Recall: Logistic Regression

Form of conditional probability model:

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

Equivalently:

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

# Recall: 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  $k = \underline{0}, \dots, K - 1$ . 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.

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

# Adding a layer

Loss is

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

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

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

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

applied component-wise.

For regression, the last layer is just linear:

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

# Nonlinearities

Commonly used nonlinearities:

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

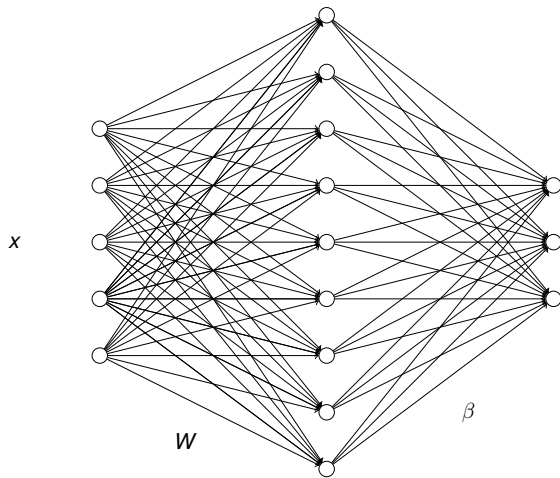
$$\phi(u) = \text{sigmoid}(u) = \frac{e^u}{1 + e^u}$$

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

# Nonlinearities

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

# Two-layer dense network



# 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.

# Training

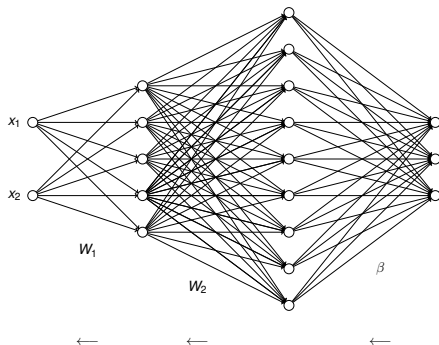
- For the last layer,  $\mathcal{L} = \frac{1}{2}(y - f)^2$  and

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

- Next, we compute

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

# High level idea



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



# 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

# 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

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



# Interactive examples

`https://playground.tensorflow.org/`

# Summary

- Neural nets are trained using stochastic gradient descent
- Implemented using backpropagation
- Can be automated to train complex networks (with no math!)