

SECTION 4

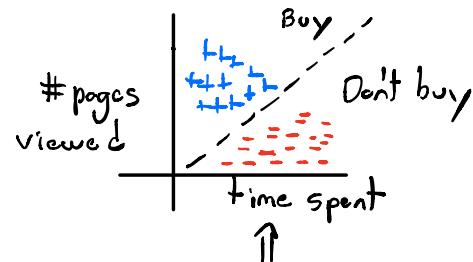
Classifying more than 2 things at the time

Prediction: Section Introduction (Video 13)
and outline

Recall \rightarrow Earlier studies \rightarrow LoR / Binary classification



- We collect data & try to predict 1 of 2 possible labels.
- \hookrightarrow e.g. $Buy = \sigma \{ w_1 \cdot (\text{time spent on site}) + w_2 \cdot (\# \text{ pages viewed}) \}$
- If only two inputs \rightarrow we can plot the data on a 2-D scatter plot.



If we can find a line that fits the data \rightarrow Linearly separable

- If we have a linearly separable problem \rightarrow LoR is fine \rightarrow Linear classifier.
 - \hookrightarrow Note we are not limited to two inputs.
 - \rightarrow Visualizing higher dimensions is not possible.
 - \hookrightarrow Many realistic datasets \rightarrow 100's or 1000's of inputs
- Main point \rightarrow Linearly separable data is separated by straight (not curved) "lines".

- (NN) \rightarrow More advanced
 - \rightarrow We can have non linearly separable data
 - \rightarrow LoR is not appropriate, NN's are!

Linear function: $w^T x$

NN are nonlinear! $\leftarrow \begin{cases} \text{Anything that cannot} \\ \text{be simplified into } w^T x \text{ is} \rightarrow \text{e.g.} \\ x^2, x^3 \end{cases}$ nonlinear.

But! \rightarrow They are nonlinear in a very specific way. $\} \Rightarrow$ They combine multiple LoR units (Neurons)

- First part of the course \rightarrow Forming a non-linear classifier \rightarrow NN!

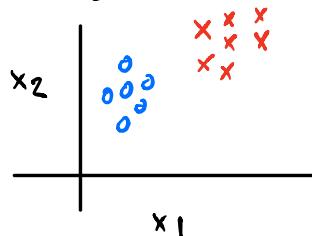
- This course → Also focuses on
 - multi-class classification:

e.g. Instead of Buy (1) or Don't buy (0)
 ↳ we have: begin checkout vs. add to cart
 vs. abandon cart --- etc.

e.g. classifying a car brand based on a picture of a car → > 2 brands of cars in the world

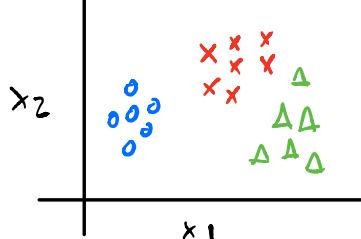
Need a classifier that can classify more than 2 things: **SOFTMAX**

Binary classification



vs.

Multi-class classification



⇒ Rest of section

- Put concepts into code:
- We will always follow the pattern → Theory = Code
- write softmax code + NN
- Apply to real-world problem

(E-commerce
project)

- We will focus
on prediction with a
NN

Given an input vector, how do I
interpret the numbers that are output?

- These numbers represent the
probabilities that the inputs belong to
each class. \hookrightarrow **Softmax**

Note \hookrightarrow outputs won't make sense
at first

LoR & NN \rightarrow Both have weights
However \rightarrow This section \rightarrow No "training"
 \downarrow by GD / ...
PREDICTION \Rightarrow weights \leftrightarrow
ONLY! \Rightarrow **won't make sense!**

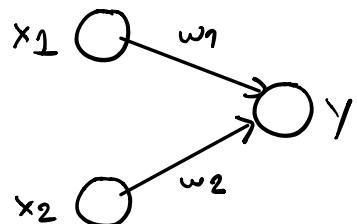
Finally: \rightarrow Typical ML process

Create Model \rightarrow Predict \rightarrow Train \rightarrow Predict
 \hookrightarrow Brain \downarrow Not accurate
Not smart!

\curvearrowleft \downarrow Accurate predictions **Smart!**

- From Logistic Regression to Neural Networks (Video 14)

Logistic Regression \rightarrow Let's gently transition from
The neuron LoR to NN:



$$a = \omega^T x + b = x_1 w_1 + x_2 w_2 + b$$

$$p(y=1|x) = \sigma(a) = \frac{1}{1+e^{-a}}$$

$$\text{Prediction} = \text{round}(p(y=1|x))$$

$$= 1 \text{ if } p(y=1|x) > 0.5$$

$$\text{else } 0$$

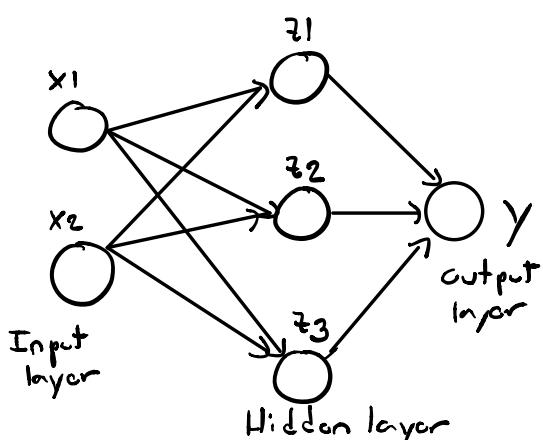
To create a NN \rightarrow

Consider the feed-forward (FF/FP)

(Forward propagation) method:

Basic idea \rightarrow Add more LoR units (Neurons)

as follows:



$$z_j = \sigma(\sum_i w_{ij} x_i + b_j)$$

$$p(y=1|x) = \sigma(\sum_j v_j z_j + c)$$

where: $i = 1, 2$

$j = 1, 2, 3$

* In recent years \rightarrow Researchers have found success with deep NN's, that is, NN's w/ many layers

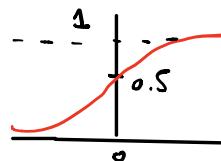
- Note that W is $[l] \times [l+1]$
now a matrix; $W \in \mathbb{R}$

- Also, note that each unit in the hidden layer has its own bias b_j .

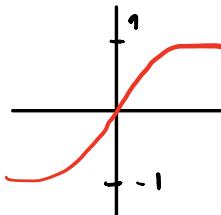
Nonlinearities \rightarrow Make NN's powerful as we have non-linear classifiers.

Examples of nonlinearities

- Sigmoid $\rightarrow \sigma(x) = \frac{1}{1 + e^{-x}}$



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



Exercise: Show the relationship between tanh & sigmoid:

$$\begin{aligned} \tanh(x) &= \frac{e^x}{e^x + e^{-x}} - \frac{e^{-x}}{e^x + e^{-x}} \\ &= \frac{e^x}{e^x(1 + e^{-2x})} - \left(\frac{e^{-x} + e^x}{e^x + e^{-x}} - \frac{e^x}{e^x + e^{-x}} \right) \end{aligned}$$

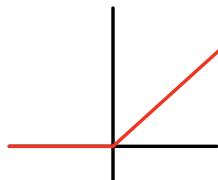
$$= \underbrace{\frac{1}{1+e^{-2x}}}_{\sigma(2x)} - \left(1 + \underbrace{\frac{1}{1+e^{-2x}}}_{\sigma(2x)} \right)$$

$$\sigma(2x) - 1 + \sigma(2x)$$

Then:

$$\tanh(x) = 2\sigma(2x) - 1$$

A vertically / horizontally scaled version of $\sigma(x)$



- $\text{relu}(x) = \max(0, x)$
 - $\underbrace{\text{Rectified linear}}_{\text{unit}} \Rightarrow \text{Looks simpler}$
 - $\text{but has been shown to work extremely well for many applications, including CV.}$

↳ Example of FP

$$x = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \rightarrow \text{First, obtain } a$$

$$w = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

$$a = w^T x + b$$

$$a = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} + 0$$

Broadcasted

$$a = \begin{bmatrix} 1 \\ 1 \end{bmatrix}_{m \times 1}$$

$$\text{For which } \sigma(a) = \begin{bmatrix} 0.731 \\ 0.731 \end{bmatrix}$$

Now, we obtain $y: \rightarrow v = \begin{bmatrix} 1 \\ z \end{bmatrix}$

$$\begin{aligned} y = P(y=1|x) &= \sigma(v^T \underbrace{\sigma(a)}_z + c) \\ &= \sigma([1 \ 1] \begin{bmatrix} 0.731 \\ 0.731 \end{bmatrix} + 0) \\ &= \sigma(1.462) = 0.912 \end{aligned}$$

Later, how do we choose W, V & b , $\leftarrow \begin{cases} \text{Since } > 0.5 \rightarrow \\ \text{Predict 1!} \end{cases}$
So that our predictions are accurate?

• Vector notation

Numpy operations are more efficient than Python for loops.

$$\begin{aligned} z_j &= \sigma(\sum_i w_{ij} x_i + b_i) \rightarrow z_j = \sigma(w_j^T x + b_j) \\ \rightarrow z &= \sigma(W^T x + b) \end{aligned}$$

$$P(y|x) = \sigma(\sum_j v_j z_j + c) \rightarrow P(y|x) = \sigma(v^T z + c)$$

$\hookrightarrow X$ is a D -dimensional vector ($D=2$ in the previous image)

z is an M -dimensional vector ($M=3$ in the previous image)

• Matrix Notation

→ we usually want to consider more than one sample at a time.

→ It's even more efficient to process multiple samples at the same time, e.g.

- X is an $N \times D$ matrix ($N =$ number of samples)
- Z is an $N \times M$ matrix
- $p(y|X)$ [Sometimes called "y"] is an $N \times 1$ matrix (For Binary classification)
 $\hookrightarrow N \times k$ matrix (For k -classes)

$$W \in \mathbb{R}^{D \times M}, b \in \mathbb{R}^{M \times 1}, v \in \mathbb{R}^{N \times 1}, c \in \mathbb{R}$$

$$z = \sigma(xw + b)$$

$$y = \sigma(zv + c)$$

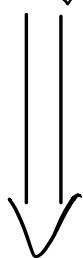
• Interpreting the weights of a NN (Video 15)

- Interpreting the LoR weights → Easy to interpret
- Interpreting a NN → Harder to interpret

But! They are interpretable $\hookleftarrow \begin{cases} \text{Have faced criticism} \\ \text{before.} \end{cases}$

\downarrow
 However, the geometry must be understood first.

A single Neuron → Lets explore the interpretation of a single neuron (Recall that a NN is just a layer of Neurons)



- We want to predict whether a person is at risk for disease X.

- Example:

↳ our subject is not obese, smokes and exercises daily.

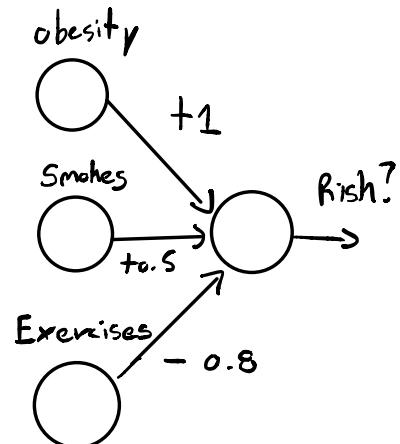
- Input vector $x = [0 \ 1 \ 1]$

- Prediction:

$$\hookrightarrow \sigma(0.5 - 0.8) = \sigma(-0.3) = 0.426 = 42.6\%$$

→ we predict "No"

- It is clear how these weights affect the output → we care about two things:
 - sign
 - magnitude
- Smokes → bad (+ve)
- Exercises → good (-ve)



What about a NN?

- Cannot do this \rightarrow Past the first layer, the weights lose their physical meaning.

- Example: Neuron 1 has:

$$+ 0.5 \text{ Smoking}$$

$$- 0.9 \text{ exercise}$$

Neuron 2 has:

$$- 0.1 \text{ Smoking}$$

$$+ 0.3 \text{ exercise}$$

Can we "combine" the

smoking weights? \rightarrow e.g. $0.5 - 0.1 = 0.4$

No!

- Sigmoid / tanh / \sim flattens



cut extreme values

and make them nonlinear.

In addition, we

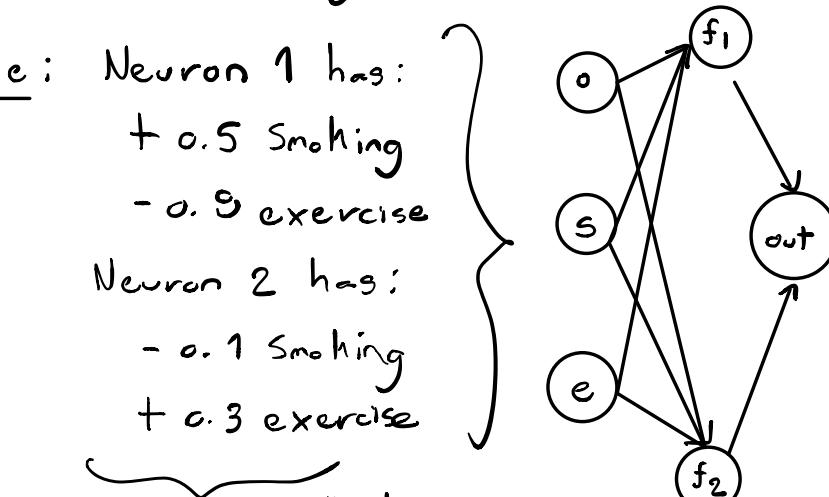
cannot remove the \Rightarrow we end up with a
nonlinearities linear model \rightarrow A single
neuron.

Important: What makes a NN so

powerful? \rightarrow Beyond the

first layer the weights no

longer represent \rightarrow obesity, smoking, exercise



- Instead, they represent something else entirely.
 ↳ Something that cannot be expressed as obesity, smoking and exercise.

↓
 The fact that we cannot express the weights in terms of the raw inputs is precisely what makes NN so powerful.

Geometry

↳ • when people talk about "interpretability", they are usually looking for simple interpretations

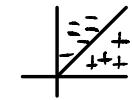
e.g. → Linear neuron
 → Decision trees

Important: ↴
 ↓

what do their decision boundaries look like? (geometric meaning)

These are clearly not very expressive.

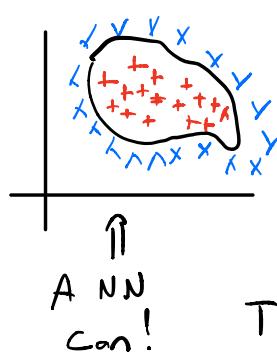
↳ Linear model → • Data b/w each class is separated by a line or plane



Tree → • Separate by axis-aligned lines



- What if we want to find a DB like the following? →



Lor
→ we would have
to manually engineer
features
These may not be
interpretable.

- Thus, trying to interpret a model in terms of its raw features is the wrong question to ask.



If you do so, you are limiting the model's geometry to be a straight line.



Better: what does the DB look like?



Later → Working with images → Provides an interesting way of interpreting the weights of a NN:

- Much larger scale → High dimensionality of the images
- NN

↳ Learn a hierarchical representation of the data.

- Example → Classifying faces



X Depth + Y

In other words, the more deeper the NN, the more complex the ideas it can represent.

↳ Research ++ → Images

Central message

Easily interpretable.

↳ When you want to interpret the weights of a NN, sometimes it doesn't make sense to look at individual weights.

||
 ↳ Individual weights → Represent how important a feature is.

When we look at images, we're looking at weights in aggregate.

However, features may be highly nonlinear combinations of the raw inputs. → No interpretation

We see which images activate which neurons.

- Softmax (Video 16)

↳ what happens if your output has more than 2 categories?

→ previously we discussed Binary classification

↳ Examples

↳ (Humidity, ground is wet, Month, location) → (Rain, No rain)

↳ (Exercise frequency, age, BMI, nutrient intake) →

→ The binary classification problems (Disease, No disease)
are Yes/No questions.

→ In many cases, there are many classes to choose from:

k classes:

• Facebook k Images:
(Advertising Applications)

valuables data to determine advertising type.
↳ {

- Faces
- Cars
- Wedding dresses
- Environment

MNIST: • Digits between

0 - 9

- Mathematical extension of Binary Classification

- Recall that for Binary classification:

↳

- only need one output node → can represent both classes!
- $P(Y=1|X)$
- why? → we can represent the other category as follows:

$$P(Y=0|X) = 1 - P(Y=1|X)$$

- we can do this in another way → Binary output with two nodes

$$\begin{array}{c} w_1 \rightarrow \text{circle} \\ \alpha_1 = w_1^T X \end{array}$$

$$\begin{array}{c} w_2 \rightarrow \text{circle} \\ \alpha_2 = w_2^T X \end{array}$$

↳ we normalize the sum of the outputs such that they sum to one:

→ we do this as follows:

$$\exp(\alpha_1) > 0$$

$$\exp(\alpha_2) > 0$$

→ Then, we calculate the output of class 1:

$$P(Y=1|X) = \frac{e^{\alpha_1}}{e^{\alpha_1} + e^{\alpha_2}}$$

→ Similarly for α_2 :

$$P(Y=2|x) = \frac{e^{\alpha_2}}{[e^{\alpha_1} + e^{\alpha_2}]}$$

Notice that:

$$P(Y=1|x) + P(Y=2|x) = 1$$

and:

$$w \in \mathbb{R}^{D \times 2}$$

Now, we can generalize for K classes:

Softmax for K classes

$$P(Y=k|X) = \frac{\exp(\alpha_k)}{Z}$$

$$Z = \exp(\alpha_1) + \exp(\alpha_2) + \dots + \exp(\alpha_K)$$

$$W = [w_1, w_2, \dots, w_K] \quad (\text{A } D \times K$$

matrix)

Vectorized LoR w/ softmax

$$A_{N \times K} = X_{N \times D} W_{D \times K}$$

$$\Rightarrow Y_{N \times K} = \underbrace{\text{softmax}}_{\text{Explanation on video 18}}(A_{N \times K})$$

Sigmoid vs. Softmax (video 17)

- When are the sigmoid and softmax equivalent?
- Consider the following:

$$P(Y=1|X) = \frac{e^{(\omega_1^T X)}}{(e^{\omega_1^T X} + e^{\omega_0^T X})}$$

Divide Top & Bottom of $P(Y=1|X)$

$$\begin{aligned} \rightarrow \frac{e^{\omega_1^T X} / e^{\omega_1^T X}}{\left(\frac{e^{\omega_1^T X}}{e^{\omega_1^T X}} + \frac{e^{\omega_0^T X}}{e^{\omega_1^T X}} \right)} &= \frac{1}{1 + e^{(\omega_0^T X - \omega_1^T X)}} \\ &= \frac{1}{1 + e^{-(\omega_1 - \omega_0)^T X}} \end{aligned}$$

Exactly the same as sigmoid
 given $w = w_1 - w_0 \rightarrow$ using two
 weights is redundant when there
 are only two classes.

Sigmoid vs. Softmax \rightarrow From a software design perspective, it is always safer to use softmax



\rightarrow It can handle both $K=2$ & $K > 2$ cases

Feed forward in Slow-mo (part 1) (video 18)

- Previous lectures \rightarrow Working under the assumption that you're familiar with:

Fast-paced! \leftarrow {
Builds on what we already know:

- Making predictions
- Interpreting output probabilities
- Multiplying vectors and matrices,

- This lecture \rightarrow Numerical examples to solidify ideas.

• Feed Forward Example

- Training Data $\rightarrow X, Y$

$$X = \begin{bmatrix} 0 & 3.5 \\ 1 & 2 \\ 1 & 0.5 \end{bmatrix}_{3 \times 2} \quad Y = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}_{3 \times 1}$$

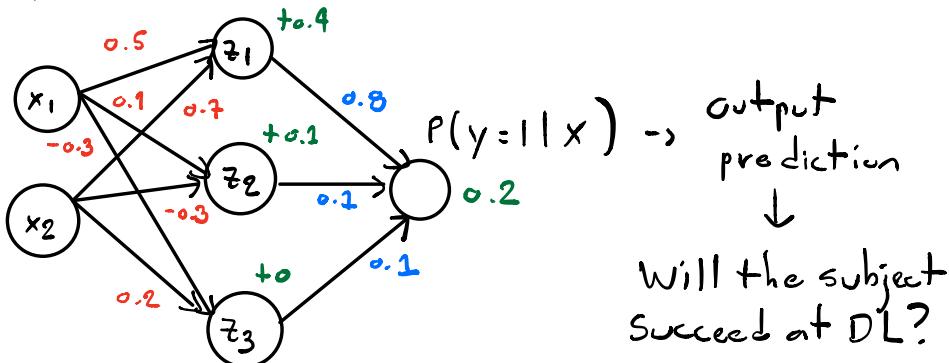
- Then, we have: $N = 3 \rightarrow 3$ samples

$D = 2 \rightarrow 2$ input features

- Our features are: x_1 : has_technical_degree
(Boolean)
- x_2 : hours_spent_studying_per_day
(Float)
- our target labels represent: $y_j = \text{Succeed-at-deep-learning}$
- Note: First, we will consider BC & later MCC.
- Targets are not used during prediction, only training (Next section)
 - Prediction? \rightarrow we take the input & predict an output using the NN.

NN structure

L₁ • Input is of size $D=2$



why does $M=3$? \rightarrow Assume, for now, this is fine.

Not a straightforward answer! \rightarrow Next section

Now, our weight matrix W is as follows:

$$W = \begin{bmatrix} 0.5 & 0.1 & -0.3 \\ 0.7 & -0.3 & 0.2 \end{bmatrix}$$

Note the weights are arbitrary!
Need training!

Note that $W \in \mathbb{R}^{D \times M}$ or $W \in \mathbb{R}^{[L-1] \times [L]}$

- In general, W_{ij} connects to node i in the previous layer, to node j in the next layer.

- Our bias vector is as follows:

$$b = \begin{bmatrix} 0.4 \\ 0.1 \\ 0 \end{bmatrix} \text{ where } b \in \mathbb{R}^{M \times 1}$$

- The weights of the hidden to output layer are as follows:

$$V = \begin{bmatrix} 0.8 \\ 0.1 \\ -0.1 \end{bmatrix} \text{ where } V \in \mathbb{R}^{M \times 1}$$

- The bias term is as follows:

$$c = 0.2 \quad \text{where } c \in \mathbb{R}$$

→ Making predictions

Take the first sample

$$X = \begin{bmatrix} 0 & 3.5 \\ 1 & 2 \\ 1 & 0.5 \end{bmatrix}_{3 \times 2} \Rightarrow x = \begin{bmatrix} 0 \\ 3.5 \end{bmatrix}$$

Initially, we compute z_1 as follows

$$z_1 = \tanh\left(\underbrace{\sum_{i=1}^2 w_i x_i}_{+ b_1}\right)$$

why \tanh ? → This is an advanced concept!

We will discuss it later in the course.

$$\begin{aligned} \text{Then } z_1 &= \tanh(0.5 \times 0 + 0.7 \times 3.5 + 0.4) \\ &= \tanh(2.85) = 0.993 \end{aligned}$$

Similarly for $z_2 = -0.740$

$$z_3 = 0.604$$

For which:

$$z = \begin{bmatrix} 0.993 \\ -0.740 \\ 0.604 \end{bmatrix}$$

$$\text{Now, we obtain } P(y=1 | x) = \sigma\left(\sum_{j=1}^3 v_j z_j + c\right)$$

$$\begin{aligned} &= \sigma(0.5 \times 0.993 \\ &\quad + 0.1 \times (-0.74) \\ &\quad - 0.1 \times (0.604) \\ &\quad + 0.2) \\ &= \sigma(0.86) = \underbrace{0.70} \end{aligned}$$

Subject #1 has
70% of succeeding at DL!

- That was a lot of work for 1 example!
- Consider the following, more efficient, vectorized process to obtain the same result:

$$z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \tanh(W^T x + b)$$

where $W \in \mathbb{R}^{D \times M} \rightarrow W^T \in \mathbb{R}^{M \times D}$
 $x \in \mathbb{R}^{D \times 1}$
 $b \in \mathbb{R}^{M \times 1}$

Note: $\underbrace{W^T x}_{M \times 1} + \underbrace{b}_{M \times 1}$

For which

$$z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \tanh \left(\begin{bmatrix} 0.5 & 0.7 \\ 0.1 & -0.3 \\ -0.3 & 0.2 \end{bmatrix} \begin{bmatrix} 0 \\ 3.5 \end{bmatrix} + \begin{bmatrix} 0.4 \\ 0.1 \\ 0 \end{bmatrix} \right)$$

$$z = \begin{bmatrix} 0.993 \\ -0.740 \\ 0.604 \end{bmatrix}$$

Similarly for the output layer:

$$P(y=1 | x) = \sigma(V^T z + c)$$

where: $V \in \mathbb{R}^{M \times 1} \rightarrow V^T \in \mathbb{R}^{1 \times M}$
 $z \in \mathbb{R}^{M \times 1}$
 $c \in \mathbb{R}$

Note: $V^T z \in \mathbb{R}$ & $c \in \mathbb{R}$

$$\Rightarrow P(y=1 | x) = \sigma([0.8 \ 0.1 \ -0.1] \begin{bmatrix} 0.993 \\ -0.740 \\ 0.604 \end{bmatrix} + 0.2)$$

$$P(y=1 | x) = \sigma(0.86) = \underbrace{0.7}$$

Same answer!

- Now, the question is, can we do the above vectorized operations for multiple samples at the same time?

↳ Yes, we do so as follows:

$$P(y=1 | x) = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \sigma(\tanh(xw + b)V + c)$$

- Note → Both the sigmoid and the tanh functions are applied in an element-wise basis.

→ x & w switched? → Samples are stored

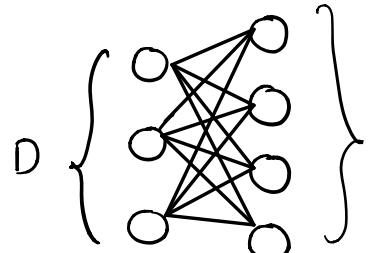
Careful with $\in \left\{ \begin{array}{l} \text{horizontally in a data} \\ \text{matrix} & \text{vertically,} \\ \text{the inner} & \text{as individual samples!} \end{array} \right\}$

dimensions!



Golden rule
of multiplication

- Consider a NN with D inputs &
 M hidden units



shape of X : $N \times D$

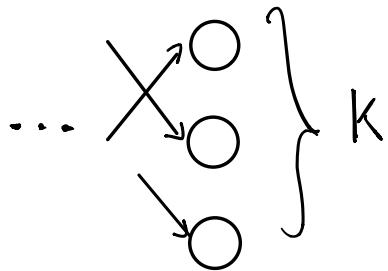
shape of W : $D \times M$

\Rightarrow shape of Z : $N \times M$

given: $Z = \underbrace{X}_{N \times D} \underbrace{W}_{D \times M}$

what about softmax?

- Helpful to think what the targets will look like
- Softmax \rightarrow Means we will have multiple output nodes. \rightarrow output is a vector.
- k classes \rightarrow output is a length k vector.



- Now, the output for 1 sample is of size K
e.g.

$$y = \begin{bmatrix} : \\ : \end{bmatrix}^K$$
- For which, if we consider N samples simultaneously,

$$Y = \left[\begin{array}{c} -y_1^T- \\ \vdots \\ -y_N^T- \end{array} \right] \underbrace{\quad}_{K} \Bigg\}^N \quad Y \in \mathbb{R}^{N \times k}$$

↑

- Each row sums to 1
 \hookrightarrow Probability that the sample belongs to the k^{th} class.

- Then, for multiple samples, output should be a 2-D matrix of size $N \times K$.

↳ Given this, we must express our targets (also) as a 2D matrix \rightarrow we do so with an indicator matrix.

 Labels \rightarrow High level: Real-life meaning
 e.g. Input: An image of a car
 output: Is the car a Jag or a BMW?

we have two labels (0/1)

Binary! \Downarrow \rightarrow Convenient given that:
 $0 \leq \sigma(z) \leq 1$

what happens \Leftrightarrow { e.g. output is 0.75 \rightarrow Predict 1
 when we have more output is 0.49 \rightarrow Predict 0
 than two labels? $K > 2$



Continue counting \rightarrow e.g. If we have three labels,
 use 0, 1, 2

↳ Category 0 - BMW
 Category 1 - Jaguar
 Category 2 - Volkswagen

- However! \rightarrow No need to round as in BC!
 \hookrightarrow softmax will output a probability distribution.
- Note \rightarrow we label the targets from $0 \dots k-1$

\swarrow
why \rightarrow We will be using these targets to index arrays:

Consider the matrix A : $A \in \mathbb{R}^{N \times K}$

Then $A[0] \rightarrow$ Accesses the
; first element

$A[k-1] \rightarrow$ Accesses the last element!

One-Hot Encoding

- Consider a NN with 6 outputs;
- our target vector y is as follows

$$Y = \begin{bmatrix} 0 \\ 5 \\ 1 \\ 3 \\ 0 \\ 2 \end{bmatrix} \xrightarrow{\text{one-hot encoding}} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad \left\{ \begin{array}{l} N=6 \\ \underbrace{\hspace{10em}}_{k=6} \end{array} \right.$$

6 distinct Categories &
6 Samples

- We encode our categories with the labels $0 \dots 5$.
- Sci-Kit Learn allows you to use strings as targets
Internally \rightarrow still uses OHE

◦ Label representations

- Keep in mind that the encoding we use for our categories has no geometrical meaning:

Labels were encoded randomly! } "Jag" is not "closer" to 0
 "BMW" nor it is "further" 1 away from "VW" 5

◦ OHE

↳ we create our indicator matrix as follows:

$$\text{Indicator}(n, k) = 1 \text{ if } Y(n, k) == k \\ \text{else } 0.$$

- Consider the following pseudocode:

```

def convert_numerical_targets_to_indicator_matrix(Yin)
  N = length(Yin)
  K = max(Yin) + 1
  Yout = zeros(N, K)
  for n in range(N):
    Yout[n, Yin[n]] = 1 # Yin indexes Yout!
  return Yout
  # values must be from
  # 0 ... K-1
  
```

- Why does this make sense?

Target indicator matrices \rightarrow can also be thought of as probabilities.

- Targets are certain! (1 or 0)
- Softmax outputs are predictions (uncertain)

\hookrightarrow e.g. Target: $[0.1 \ 0.1 \ 0.8]$
Prediction: $[0 \ 0 \ 1]$

\uparrow
*(Softmax) was
our prediction
Correct (target)*

Feed forward in slow-mo (part 2) (video 19)

- The NN outputs (predictions) will not be exactly 0 or 1 (Because they are predictions, not certainties)
- Targets are exactly 0 or 1 (They are certain)

\hookrightarrow Goal \rightarrow After training, the max probability corresponds to the known target.

\downarrow
Again: $[0.1, 0.1, 0.5, 0.2, 0.1]$

$[0, 0, 1, 0, 0]$
 \hookrightarrow Target

Visual Example of Softmax output

- $(N \times k) = (2 \times 3)$ softmax output

	Category 0	Category 1	Category 2
Sample 1 (Input = X_1 , Target = Y_1)	$p(Y_1=0 x_1)$	$p(Y_1=1 x_1)$	$p(Y_1=2 x_1)$
Sample 2 (Input = X_2 , Target = Y_2)	$p(Y_2=0 x_2)$	$p(Y_2=1 x_2)$	$p(Y_2=2 x_2)$
Sample 3 (Input = X_3 , Target = Y_3)	$p(Y_3=0 x_3)$	$p(Y_3=1 x_3)$	$p(Y_3=2 x_3)$

- Simple Example Comparing softmax output vs. Target Indicator

<u>Predictions</u>			<u>Targets</u>		
0.2	0.7	0.1	0	1	0
0.4	0.3	0.3	0	0	1
0	0.4	0.6	0	0	1
0.3	0.5	0.2	1	0	0

█ → Correct prediction
 █ → wrong prediction

 Classification
 Accuracy = $2/4 = 50\%$

- How to compare them in code

`prediction_labels = np.argmax(softmax_outputs, axis=1)`

`target_labels = np.argmax(target_indicator, axis=1)`

check if correct

`accuracy = sum(prediction_labels == target_labels) / N`

- Note \rightarrow `np.argmax` is the inverse of converting to indicator.

↳ Verify:

`Y == np.argmax(`
 `convert_numerical_targets_to_indicator_matrix(Yin),`
 `axis=1)`

All of the vector values of the above boolean value should be True.

- Back to our data set:

- Recall that in our example $|A| = 2$:

- Then, we can obtain the following indicator matrix:

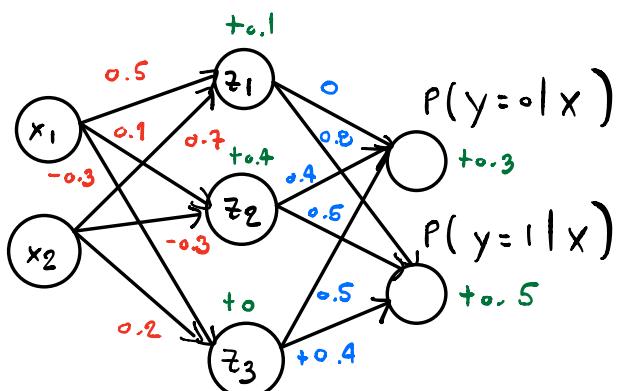
$$Y = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}_{3 \times 1} \Rightarrow \text{Indicator} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}_{3 \times 2}$$

- After training, we would like our output probabilities to be close to the targets, e.g.:

$$\text{Predictions} = \begin{bmatrix} 0.01 & 0.99 \\ 0.02 & 0.98 \\ 0.99 & 0.01 \end{bmatrix}_{3 \times 2}$$

- Strange numbering? \rightarrow Rule of thumb: If math related use a 1-based count. On the other hand, if it is related to programming / Implementation then Language dependent! \leftarrow use a zero-based count.
- e.g. Indicator matrix is indexed from 0 ... n-1

Our NN is modified as follows:



The V matrix of weights is modified as follows:

$$V = \begin{bmatrix} 0 & 0.8 \\ 0.4 & 0.5 \\ 0.5 & 0.4 \end{bmatrix}_{3 \times 2} \quad C = \begin{bmatrix} 0.3 \\ 0.6 \end{bmatrix}_{2 \times 1}$$

- Let's calculate the softmax output for the data:

Recall that $\vec{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \tanh(\vec{w}^T \vec{x} + b)$

$$\vec{z} = \begin{bmatrix} 0.993 \\ -0.740 \\ 0.604 \end{bmatrix}_{3 \times 1}$$

Now:

$$= \text{softmax}(\vec{v}^T \vec{z} + c)$$

$$\begin{bmatrix} P(y=0|x) \\ P(y=1|x) \end{bmatrix} = \text{softmax} \left(\begin{bmatrix} 0 & 0.4 & 0.5 \\ 0.5 & 0.5 & 0.4 \end{bmatrix} \begin{bmatrix} 0.993 \\ -0.740 \\ 0.604 \end{bmatrix} + \begin{bmatrix} 0.3 \\ 0.6 \end{bmatrix} \right)$$

$$\text{softmax} \left(\begin{bmatrix} 0.306 \\ 1.166 \end{bmatrix} \right)$$

$$= \begin{bmatrix} \frac{e^{0.306}}{e^{0.306} + e^{1.166}} \\ \frac{e^{1.166}}{e^{0.306} + e^{1.166}} \end{bmatrix} = \begin{bmatrix} 0.30 \\ 0.70 \end{bmatrix}$$

Same answer as

before!

- weights were chosen ↵|| as taught in lecture/video 17

- We obtain similar results due to this!

- Finally -> we may perform the vectorized operations above for the N samples!

$$P(r|x) = \text{softmax}(\tanh(xw + b)v + c)$$

↳ Notice that $b \in \mathbb{R}^{M \times 1}$
and $xw \in \mathbb{R}^{N \times M}$

↓
How is the matrix addition $xw + b$
valid?

↓
→ Bias b must be applied to all input
samples:

Define $b \Rightarrow b = \begin{bmatrix} b^T \\ \vdots \\ b^T \end{bmatrix} \}^N$
as follows

for vectorized operations that
consider all samples.

↓
↳ Similarly for all bias vectors
in each hidden layer!

Numpy does it automatically:

$$X \cdot \text{dot}(w) + b$$

$\underbrace{\phantom{X \cdot \text{dot}(w) + b}_{\text{Broadcasting!}}}$

Prediction Quizzes (Video 25)

Question 1: Supposed we removed all the nonlinearities from the NN. what is the result?

$$\hat{y} = \sigma(\tanh(xw + b)v + c)$$

Solution → Removing all nonlinearities results in the following (consider one sample):

$$\begin{aligned}\hat{y} &= v^T(w^T x + b) + c \\ \Rightarrow \hat{y} &= \underbrace{v^T x}_{} + c\end{aligned}$$

A NN becomes
No more complex
than LoR!
 $\Leftarrow \left\{ \begin{array}{l} \text{our prediction becomes a} \\ \text{Linear Regressor!} \rightarrow \text{option 2} \end{array} \right.$

Question 2: • Input $x = [1 \ 2]$ $N=1$
 • Input-to-hidden: $D=2$

$$w_1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}_{2 \times 2} \quad M=2$$

• Hidden to output:

$$w_2 = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}_{2 \times 2} \quad K=2$$

- Hidden layer non-linearity:

$\tanh()$

- $b = 0, c = 0$
- Softmax output
- What is the output?

$$Y = \text{Softmax} \left(\tanh \left([1 \ 2] \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \right) \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \right)$$

$$Y = \text{Softmax} \left(\tanh \left(\begin{bmatrix} 3 \\ 1 \end{bmatrix}^T \right) \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \right)$$

$$Y = \text{Softmax} \left(\begin{bmatrix} 0.995 & 0.7165 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \right)$$

$$Y = \text{Softmax} \left(\begin{bmatrix} 0.7165 \\ 1.7566 \end{bmatrix}^T \right) = \begin{bmatrix} 0.2611 & 0.7389 \end{bmatrix}$$

*option #4
As a 2D-array!*

$$\frac{e^{0.7165}}{e^{0.7165} + e^{1.7566}} = \frac{2.047}{7.939} = 0.2611$$

Prediction Summary (video 26)

↳ what did we do in this section?

↳ • Recall: This course was built upon concepts based on Binary classification, using LoR. ✓

- Binary Classification $\leftarrow \begin{cases} & \bullet \text{ NN} \rightarrow \text{LoR stacked in layers} \quad \text{✓} \\ & \bullet \text{ We learned that we can use any number of outputs by changing from sigmoid to softmax.} \\ & \bullet \text{ Alternatives to sigmoid in hidden layers} \rightarrow \text{tanh and ReLU AF.} \quad \text{✓} \\ & \bullet \text{ Softmax, Full NN (predictions) in code.} \quad \text{✓} \\ & \underbrace{\qquad\qquad\qquad}_{\text{Visualized data (3 blobs)}} \quad \text{✓} \\ & \bullet \text{ Prediction on e-commerce dataset.} \quad \text{✓} \end{cases}$
- Section Main goal → How do we go from input to output?

$$P(y=k|x) = \text{softmax}(V^T f(W^T x))$$

- what do the outputs mean?
- Weights were just random
⇒ Predictions are also random.

↳ Hence, Inaccurate.

- Next section -> change the weights
to make predictions accurate.