SECTION IV

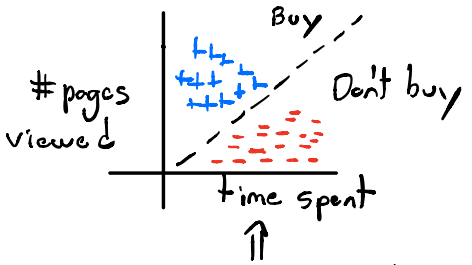
Classifying more than 2 things at the time

Prediction: Section Introduction and outline (Video 13)

Recall -> Earlier studies -> LoR / Binary classification

↓

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



⇑

If we can find a line that fits the data -> Linearly separable

- If we have a linearly separable problem -> LoR is fine -> Linear classifier.
  - ↳ Note we are not limited to two inputs.
  - -> Visualizing higher dimensions is not possible.
    - ↳ Many realistic datasets -> 100's or 1000's of inputs

↓

- Main point -> Linearly separable data is separated by straight (not curved) "lines".

- (NN) -> More advanced ↓
  - -> we can have nonlinearly separable data
  - -> LoR is not appropriate, NN's are!

Linear function: $w^T x$

NN are nonlinear! <= { Anything that cannot be simplified into $w^T x$ is nonlinear. -> e.g. x^2, x^3

BUT! -> They are nonlinear in a very specific way. } => They combine multiple LoR units (Neurons)

- First part of the course -> Forming a non-linear classifier -> 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.

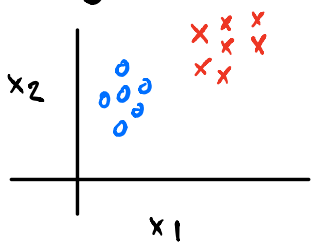
e.g. classifying a car brand based on a picture of a car → > 2 brands of cars in the world

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

↓

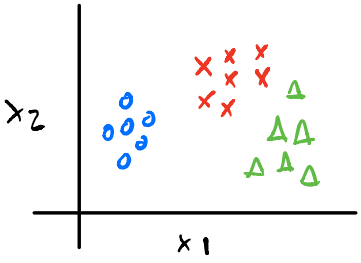
Can be applied to LoR as well, however, the sigmoid suffices for the LoR / Binary classification.

Binary classification



vs.

Multi-class classification

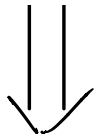


=>

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. ⇝ Softmax

Note ⇝ outputs won't make sense at first

LoR & NN → Both have weights
However → This section → No "training" by GD/....
⇓
PREDICTION ONLY! ⇒ weights won't make sense!

Finally: → Typical ML process

Create Model → Predict → Train → Predict

Brain

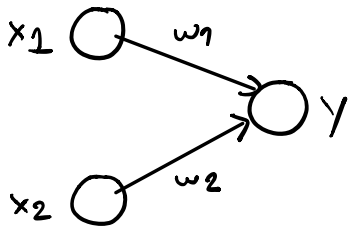
Predict → Not accurate (Not smart!)

Train ← Predict → Accurate predictions (Smart!)

- From Logistic Regression to Neural Networks (video 14)

Logistic Regression -> Let's gently transition from LoR to NN:

The neuron



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

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

Prediction = round($p(y=1|x)$)

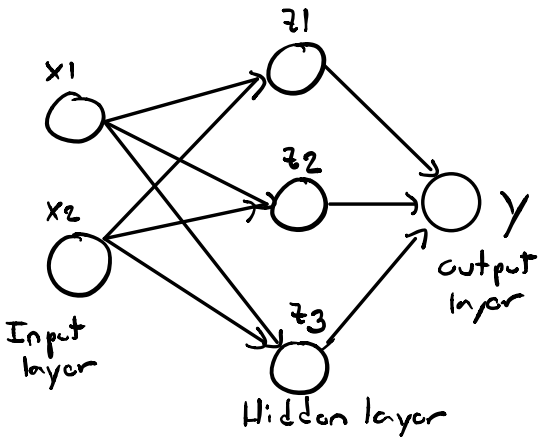
= 1 if $p(y=1|x) > 0.5$

else 0

To create a NN ->

Consider the feed-forward (FF/FP) (Forward propagation) method:

↳ Basic idea -> Add more LoR units (Neurons) as follows:



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

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

where: $i = 1, 2$

$j = 1, 2, 3$

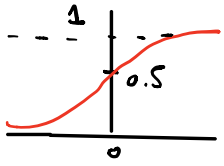
* In recent years -> Researchers have found success with deep NN's, that is, NN's w/ many layers

- 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 it's own bias b_j.

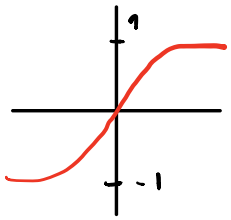
Nonlinearities -> Make NN's powerful as we have non-linear classifiers.

Examples of nonlinearities

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



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



Exercise: Show the relationship between tanh & sigmoid:

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

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

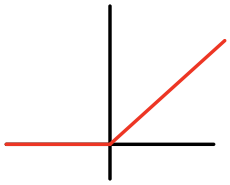
$$= \underbrace{\frac{1}{1+e^{-2x}}}_{\sigma(2x)} - 1 + \underbrace{\frac{1}{1+e^{-2x}}}_{\sigma(2x)}$$

Then:

$$\boxed{\tanh(x) = 2\sigma(2x) - 1}$$

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

- $\text{relu}(x) = \max(0, x)$



Rectified linear unit

Looks simpler but has been shown to work extremely well for many applications, including CV.

↳ Example of FP

$X = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ → First, obtain a

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

$b = c = 0$

$$a = W^T X + b$$

Broadcasted ↑

$$a = \underbrace{\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}}_{M \times D} \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_{D \times 1} + 0$$

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

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

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

$$y = P(y=1|x) = \sigma(v^T \underbrace{\sigma(a)}_{z} + c)$$

$$= \sigma\left(\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 0.731 \\ 0.731 \end{bmatrix} + 0\right)$$

$$= \sigma(1.462) = \underbrace{0.812}$$

Since $> 0.5 \rightarrow$ Predict 1!

Later, how do we choose W, V & b, so that our predictions are accurate?

- Vector notation

Numpy operations are more efficient than Python for loops.

$$z_j = \sigma\left(\sum_i W_{ij} x_i + b_i\right) \rightarrow z_j = \sigma(W_j^T x + b_j)$$

$$\rightarrow z = \sigma(W^T x + b)$$

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

↳ 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)
    L, $N \times K$ matrix (For K-classes)

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

  $Z = \sigma(XW + b)$

  $Y = \sigma(Zv + c)$

- **Interpreting the weights of a NN** (video 15)

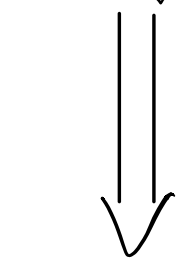
  - Interpreting the LoR weights → Easy to interpret
  - Interpreting a NN → Harder to interpret

  BUT! They are interpretable ⇐ { Have faced criticism before.

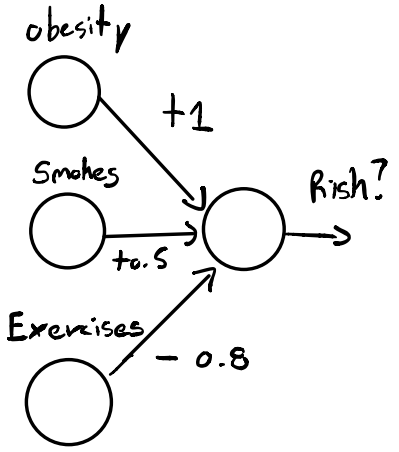
  ⇓

  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.



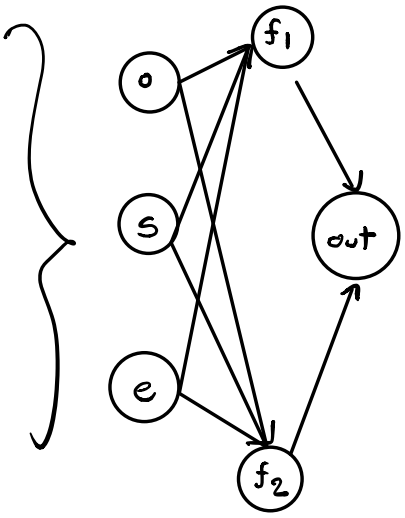
  - Input vector $x = [0\ 1\ 1]$
  - Prediction:
    - $\sigma(0.5 - 0.8) = \sigma(-0.3) = 0.426 = 42.6\%$
    - → we predict "No"

- 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 -> Past the first layer, the weights lose their physical meaning.

- Example: Neuron 1 has:
  + 0.5 Smoking
  - 0.9 exercise
  Neuron 2 has:
  - 0.1 Smoking
  + 0.3 exercise



Can we "combine" the Smoking weights? -> e.g. $+0.5 - 0.1 = 0.4$ No!

- Sigmoid / tanh / -> Flattens out extreme values and make them nonlinear.

⇓

In addition, we Cannot remove the nonlinearities => We end up with a linear model -> A single neuron.

Important: What makes a NN so powerful? -> Beyond the first layer the weights no longer represent -> 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

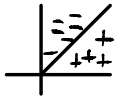
↳ • when people talk about "interpretability", they are usually looking for simple interpretations

e.g. → Linear neuron
→ Decision trees

Important: ↵

⇓

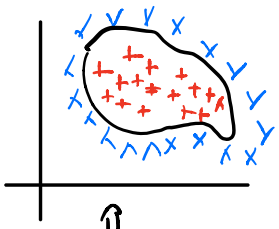
what do their decision boundaries look like? (Geometric meaning)

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

Tree → 
• Separate by axis-aligned lines

These are clearly not very expressive.

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



A NN can!

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 models geometry to be a straight line.

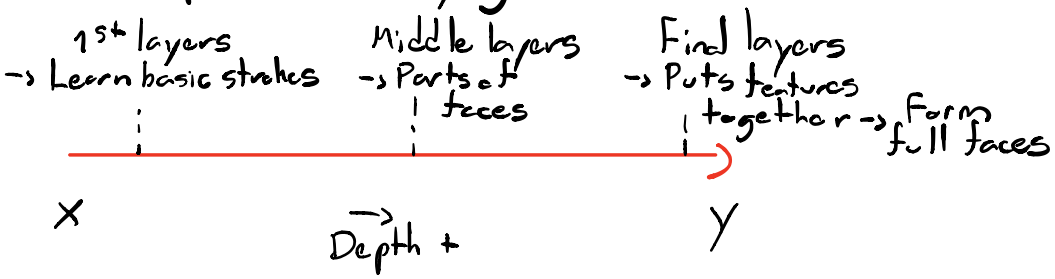
⇓

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



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

↳ Research ++ → Images

Easily interpretable.

Central message

↳ 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.

↓

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

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

↓

we see which images activate which neurons.

- Softmax (video 16)

  L, what happens if your output has more than 2 categories?

  → previously we discussed Binary classification

  L, Examples

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

  L, (Exercise frequency, age, BMI, nutrient intake) → (Disease, No disease)

  → The binary classification problems are Yes/No questions.

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

  K classes:

  - Facebook Images: (Advertising Applications) → valuable data to determine advertising type.
    - Faces
    - Cars
    - Wedding dresses
    - Environment

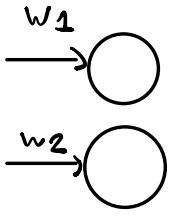
  MNIST: • Digits between 0-9

- Mathematical extension of Binary Classification

- Recall that for Binary classification:
  - only need one output node -> can represent both classes!
  - $P(Y=1 \mid X)$
  - why? -> we can represent the other category as follows:

$$P(Y=0 \mid X) = 1 - P(Y=1 \mid X)$$

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



  - we normalize the sum of the outputs such that they sum to one:
  - -> we do this as follows:

$$exp(a_1) > 0$$

$$exp(a_2) > 0$$

  - -> Then, we calculate the output of class 1:

$$P(Y=1 \mid X) \quad \frac{e^{a_1}}{[e^{a_1} + e^{a_2}]}$$

-> Similarly for a_2:

$$P(Y=2|x) = \frac{e^{a_2}}{[e^{a_1}+e^{a_2}]}$$

Notice that:

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

and:

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

Now:

Softmax for K classes

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

$$Z = \exp(a_1) + \exp(a_2) + \cdots + \exp(a_k)$$

$W = [w_1, w_2, \cdots, w_k]$ (A $D \times k$ matrix)

Vectorized LoR w/ softmax

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

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

Sigmoid vs. Softmax (video 17)

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

$$P(Y=1|X) = \frac{e^{(w_1^T x)}}{\left(e^{w_1^T x} + e^{w_0^T x}\right)}$$

Divide Top & Bottom at $P(Y=1|X)$ by $e^{w_1^T x}$

$$\rightarrow \quad \frac{e^{w_1^T x} / e^{w_1^T x}}{\left(\frac{e^{w_1^T x}}{e^{w_1^T x}} + \frac{e^{w_0^T x}}{e^{w_1^T x}}\right)} = \frac{1}{1 + e^{(w_0^T x - w_1^T x)}}$$

$$= \frac{1}{1 + e^{-(w_1^T x - w_0^T x)}}$$

$$= \frac{1}{1 + e^{-(w_1 - w_0)^T x}}$$

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

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

↓

→ It can handle both $K = 2$ & $K > 2$ cases

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

- Previous lectures → working under the assumption that you're familiar with:
  - Making predictions
  - Interpreting output probabilities
  - Multiplying vectors and matrices,

  Fast-paced! ⇐ Builds on what we already know;
- This lecture → Numerical examples to solidify ideas.
- Feed Forward Example
  - Training Data → X, Y

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

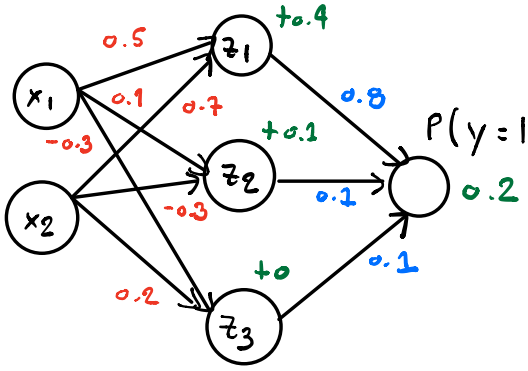
  - Then, we have: $N = 3$ → 3 samples

    $D = 2$ → 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 = succeed_at_deep_learning
- Note: First, we will consider BC & later MCC.
  - Targets are not used during prediction, only training (Next section)

  Prediction? -> we take the input & predict an output using the NN.

- NN structure
  - Input is of size D=2



$P(y=1|x)$ -> output prediction ↓ Will the subject succeed at DL?

why does M=3? -> Assume, for now, this is fine.

Not a straightforward answer! -> 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 $W = \mathbb{R}^{[l-1] \times [l]}$

- 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:

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

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

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

- 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

$$z_1 = \underbrace{\tanh}\left(\sum_{i=1}^{2} W_{i1} x_i + b_1 \right)$$

why tanh? → This is an advanced concept!
We will discuss it later in the course.

Then → $z_1 = \tanh(0.5 \times 0 + 0.7 \times 3.5 + 0.4)$
$= \tanh(2.85) = 0.993$

Similarly for $z_2 = -0.740$
$z_3 = 0.604$

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

Now, we obtain $P(y=1 \mid x) = \sigma\left(\sum_{j=1}^{3} V_j z_j + C \right)$

$= \sigma(0.8 \times 0.993$
$+ 0.1 \times (-0.74)$
$- 0.1 \times (0.604)$
$+ 0.2)$
$= \sigma(0.86) = \underbrace{0.70}$

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:

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

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:

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

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

$$\Rightarrow P(y=1 \mid x) = \sigma\left(\begin{bmatrix} 0.8 & 0.1 & -0.1 \end{bmatrix} \begin{bmatrix} 0.993 \\ -0.740 \\ 0.604 \end{bmatrix} + 0.2 \right)$$

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

Same answer!

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

↳ Yes, we do so as follows:

$$P(Y=1|X) = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \sigma(\tanh(XW+b)V+c)$$

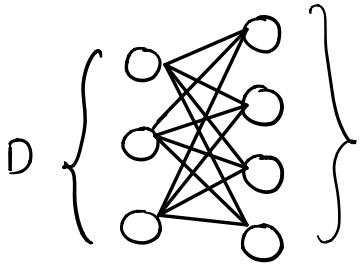
- Note → Both the sigmoid and the tanh functions are applied in an element-wise basis.

→ X & W switched? → Samples are stored horizontally in a data matrix & vertically as individual samples!

Careful with the inner dimensions! ⇓

Golden rule of multiplication ↑

- Consider a NN with D inputs & M hidden units



shape of X: N x D

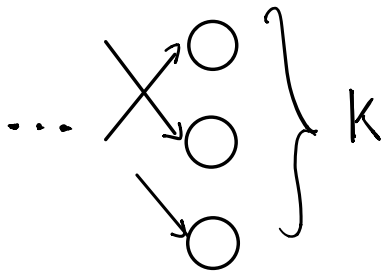
shape of W: D x M

⇒ shape of Z: N x 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 -> Means we will have multiple output nodes. ↪ output is a vector.
- K classes -> output is a length K vector.



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

  e.g.

$$y = \begin{bmatrix} \vdots \end{bmatrix} \Big\} K$$

- For which, if we consider N samples simultaneously,

$$Y = \underbrace{\begin{bmatrix} - y_1^T - \\ \vdots \\ - y_N^T - \end{bmatrix}}_{K} \Big\} N \qquad Y \in \mathbb{R}^{N \times K}$$

  - Each row sums to 1
    - ↳ 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 → we do so with an indicator matrix.

Labels → We use one-hot encoding!

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! → Convenient given that: $0 \leq \sigma(z) \leq 1$

⇓

e.g. output is 0.75 → Predict 1

output is 0.49 → Predict 0

what happens when we have more than two labels? $K > 2$

⇓

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

↳ Category 0 - BMW

Category 1 - Jaguar

Category 2 - Volkswagen

- However! -> No need to round as in BC!
  - ↳ Softmax will output a probability distribution.

- Note -> we label the targets from $0 \dots K-1$

  why -> We will be using these targets to index arrays:

  Consider the matrix A; $A \in \mathbb{R}^K$

  Then A[0] -> Accesses the first element
  ⋮
  A[K-1] -> Accesses the last element!

- One-Hot Encoding

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

$$y = \begin{bmatrix} 0 \\ 5 \\ 1 \\ 3 \\ 0 \\ 2 \end{bmatrix} \Rightarrow \text{one-hot encoding} \Rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad N = 6$$

6 distinct categories & 6 samples

$K = 6$

- We encode our categories with the labels $0 \dots 5$.
- Sci-kit Learn allows you to use strings as targets

  Internally -> 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" (0) is not "closer" to "BMW" (1) nor it is "further" away from "VW" (5)

- OHE
  - ↳ we create our indicator matrix as follows:

$$\text{Indicator}(n,k) = 1 \text{ if } Y(n,k) == K$$

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 -> can also be thought of as probabilities.

⇓

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

↳ e.g. Target: [0.1 0.1 0.8]
Prediction: [0 0 1]

↑ our prediction (Softmax) was 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)

↳ Goal -> After training, the max probability corresponds to the known target.

⇓

Again: [0.1, 0.1, 0.5, 0.2, 0.1] ← Prediction

[0, 0, 1, 0, 0] ← Target

Visual Example of Softmax output

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

	Category 0	Category 1	Category 2
Sample 1 ($Input = X_1$, $Target = Y_1$)	$P(Y_1 = 0 \mid X_1)$	$P(Y_1 = 1 \mid X_1)$	$P(Y_1 = 2 \mid X_1)$
Sample 2 ($Input = X_2$, $Target = Y_2$)	$P(Y_1 = 0 \mid X_2)$	$P(Y_1 = 1 \mid X_2)$	$P(Y_1 = 2 \mid X_2)$
Sample 3 ($Input = X_3$, $Target = Y_3$)	$P(Y_1 = 0 \mid X_3)$	$P(Y_1 = 1 \mid X_3)$	$P(Y_1 = 2 \mid X_3)$

- Simple Example Comparing Softmax output vs. Target Indicator

Predictions

0.2	0.7	0.1
0.4	0.3	0.3
0	0.4	0.6
0.3	0.5	0.2

Targets

0	1	0
0	0	1
0	0	1
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 → 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 $K = 2$:
  - 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.:

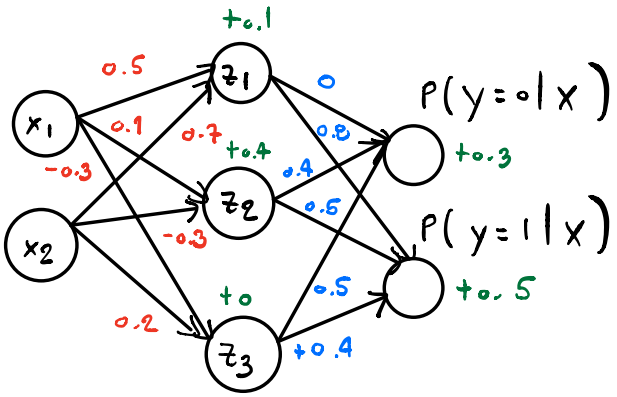
$$\text{Predictions} = \begin{bmatrix} 0.01 & 0.99 \\ 0.02 & 0.98 \\ 0.99 & 0.01 \end{bmatrix}_{3\times 2}$$

- strange numbering? -> Rule of thumb: If math related use a 1-based count. on the other hand, if it is related to programming / Implementation then use a zero-based count.

Language dependent! ←

e.g. Indicator matrix is indexed from $0 \dots K-1$

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

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

Now:

$$= \text{Softmax}(V^T z + c)$$

$$\begin{bmatrix} P(y=0|x) \\ P(y=1|x) \end{bmatrix} = \text{Softmax}\left(\begin{bmatrix} 0 & 0.4 & 0.5 \\ 0.8 & 0.5 & 0.4 \end{bmatrix} \begin{bmatrix} 0.993 \\ -0.740 \\ 0.604 \end{bmatrix} + \begin{bmatrix} 0.3 \\ 0.6 \end{bmatrix} \right)$$

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

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

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!

$$P(y|x) = \text{Softmax}\left(\tanh(XW+b)V + c\right)$$

↳ 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 as follows $\Rightarrow b = \begin{bmatrix} — & b^T & — \\ & \vdots & \\ — & b^T & — \end{bmatrix}$ (N rows, M columns)

for vectorized operations that consider all samples.

↳ Similarly for all bias vectors in each hidden layer!

⇓

Numpy does it automatically:

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

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):

$$\hat{y} = V^T(W^T x + b) + c$$

$$\Rightarrow \hat{y} = \underbrace{V^T x' + c}$$

our prediction becomes a Linear Regressor! → option 2

Question 2:

- Input $X = \begin{bmatrix} 1 & 2 \end{bmatrix}$ $N = 1$, $D = 2$
- Input-to-hidden:

$$W_1 = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}_{2 \times 2} \quad M = 2$$

- 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?

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

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

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

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

option #4

As a 2D-array!

$$\frac{e^{0.7165}}{e^{0.7165} + e^{1.7566}} = \frac{2.047}{7.839} = 0.2611$$

Prediction Summary (video 26)

↳ what did we do in this section?

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

- NN → LoR stacked in layers ☑
- We learned that we can use any number of outputs by changing from sigmoid to softmax.
- Alternatives to sigmoid in hidden layers → tanh and ReLU AF. ☑
- Softmax, Full NN (predictions) in code. ☑
  - Visualized data (3 blobs) ☑
- Prediction on e-commerce dataset. ☑

Binary Classification ⇐ { the bullets above

⇓

Multi-class Classification

- Section Main goal → How do we go from input to output?

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

- what do the outputs mean?
- Weights were just random

⇒ Predictions are also random,

↳ Hence, Inaccurate.

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