Lecture 7 – Neural Networks I

Today

Predicting with neural networks

- Artificial neurons
- Activation functions
- Feed-forward neural networks
  - ·One layer ·Multiple layers ·Output

See slides from masterclass 2 for general introduction to neural networks and deep learning

Artificial neurons

At its core, an artificial neuron is essentially just another way of expressing a perceptron.

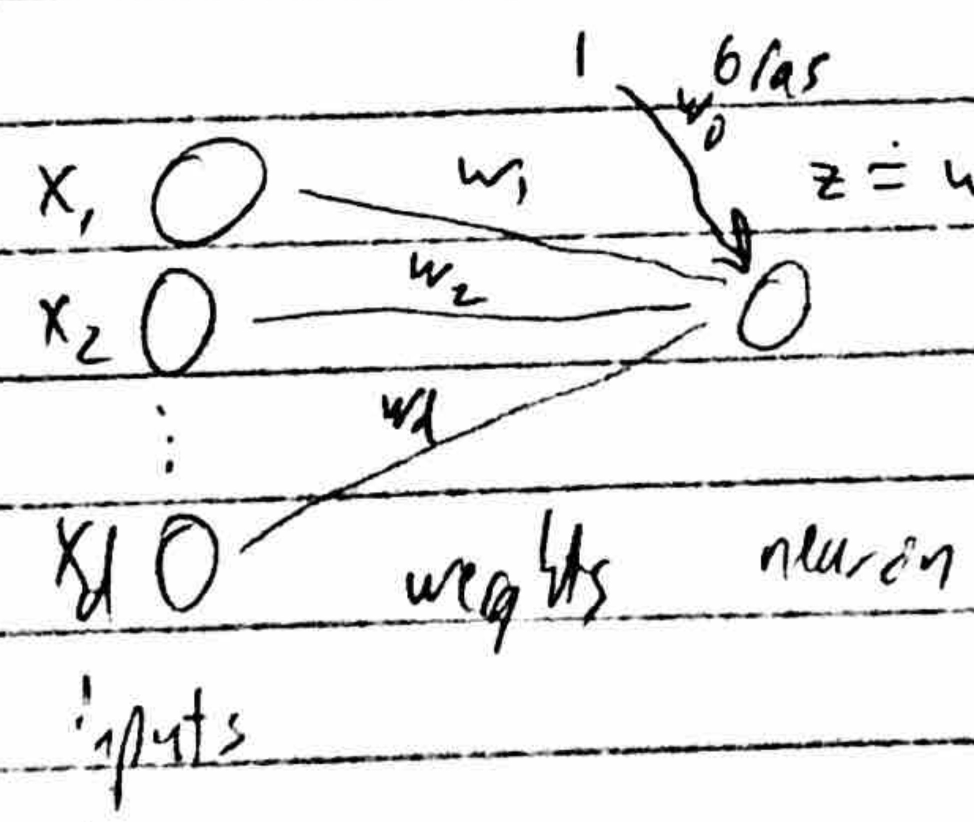
Remember: To make a prediction, perceptron computes $\theta \cdot x + \theta_0$.

$$\theta \cdot x + \theta_0 = \begin{pmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_d \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} + \theta_0 = \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_d x_d + \theta_0$$

An artificial neuron is virtually identical, but we change notation and terminology and we draw a diagram.

$\theta \to w$ "theta" → "weights"

$\theta_0 \to w_0$ "offset" → "bias"



$z = w_1 x_1 + w_2 x_2 + \dots + w_d x_d + w_0$ ← input to neuron

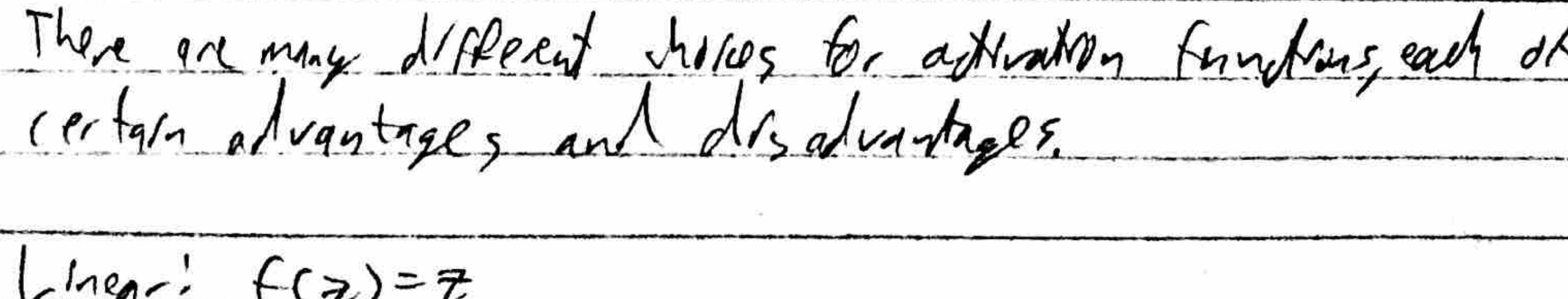
z, which is equivalent to $\theta \cdot x + \theta_0$, is the input to the neuron

If the neuron outputs z, then the output is linear and we've just built a linear classifier (we can add sign function at the end to make it classify.)

(?) How can we make this neuron a more powerful classifier?
As with perceptron, we can add non-linearity.
Unlike non-linear data transformations and kernels, the non-linearity is going to be applied after computing $\theta \cdot x + \theta_0$ ($z = w_1 x_1 + w_2 x_2 + \dots + w_d x_d + w_0$).

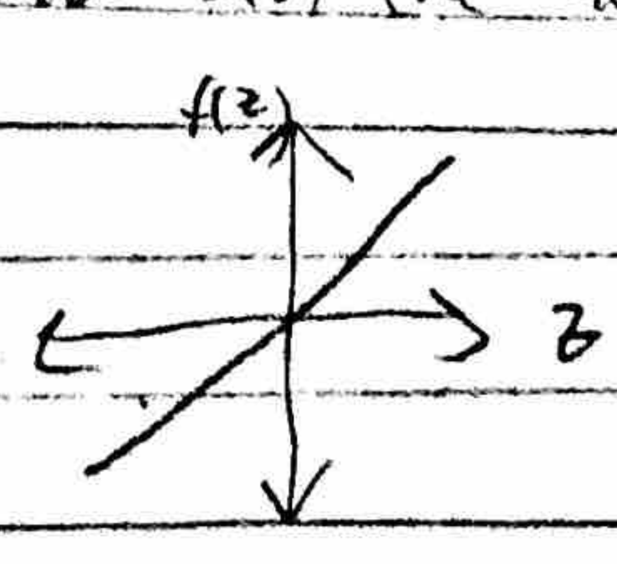
<u>Activation functions</u>
Activation functions are applied to the input z to a neuron.
The output of the neuron is the result of applying the activation function.
Notation: $f(z)$

Note: As with perceptron, I'm often going to leave out the bias (offset) for simplicity, but when actually building these models it should be included.



There are many different choices for activation functions, each of which has certain advantages and disadvantages.

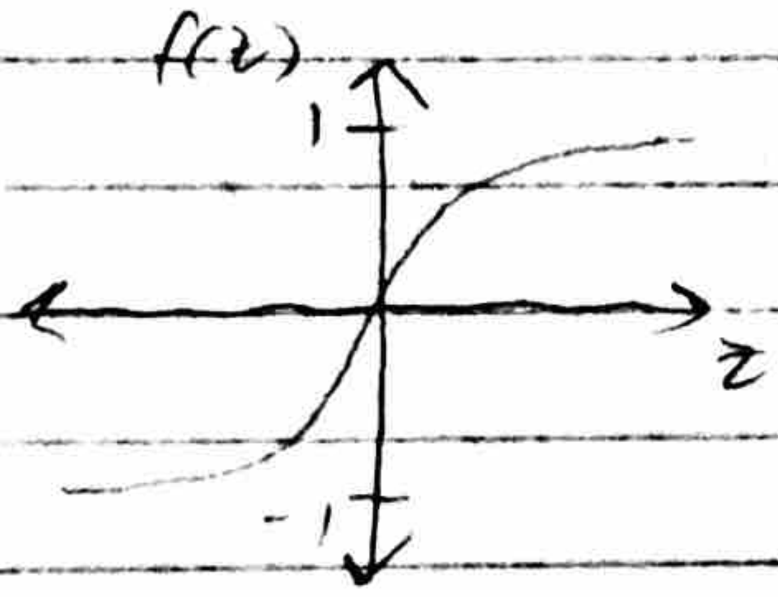
<u>Linear</u>: $f(z) = z$
Bad activation function because then the neuron can only classify linearly separable datasets



The following are good activation functions and are commonly used.
They are good because:
- non-linear
- relatively simple
- easy to compute the derivative/gradient
  ^ (useful for learning a neural network, which we'll talk about next time)

(?) Anyone know how to draw?

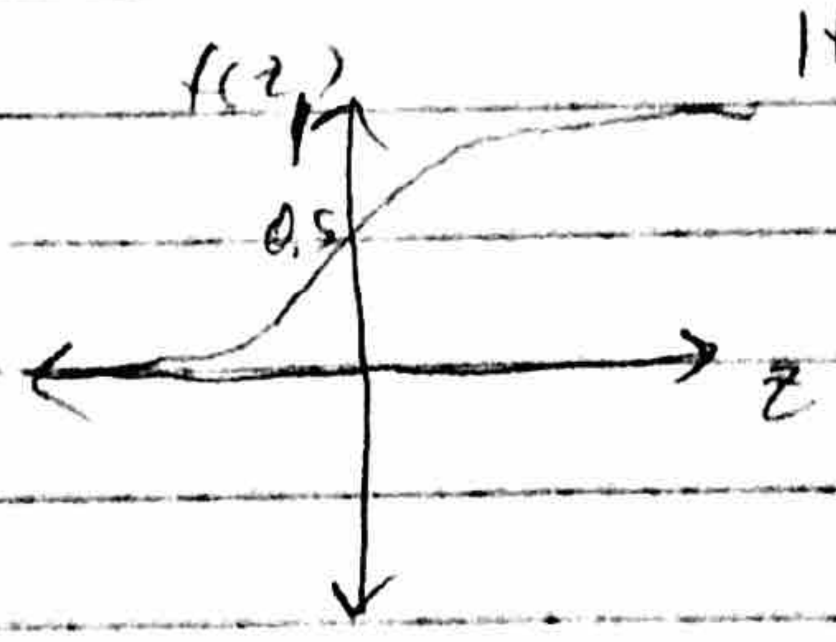
Hyperbolic tangent: $f(z) = \tanh(z)$



Notes:
- Approaches -1 on the left and +1 on the right
- Smooth approximation to the sign function used in perceptron

(?) Anyone know what σ is or how to draw?

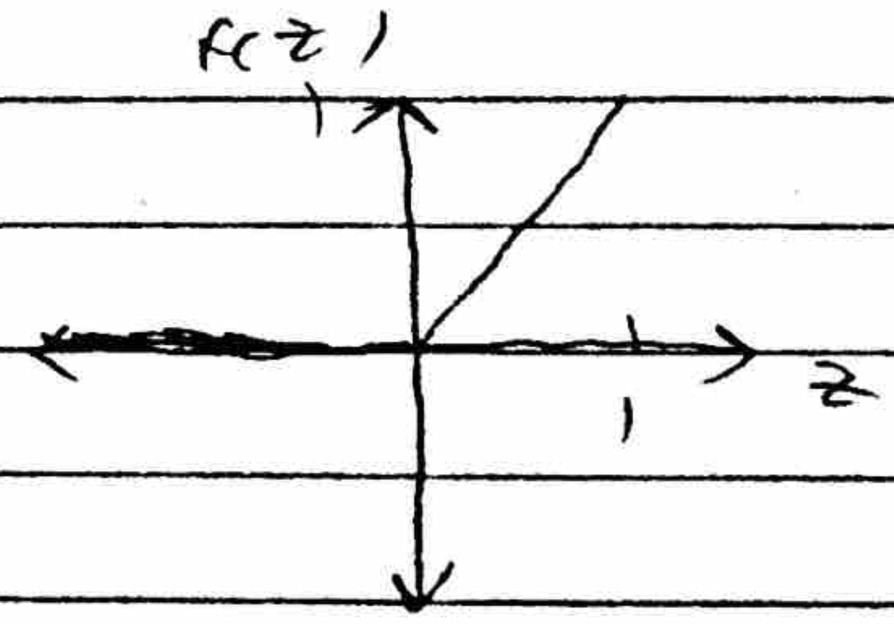
Sigmoid: $f(z) = \sigma(z) = \dfrac{1}{1+e^{-z}}$ (logistic function)



Notes:
- Approaches 0 on the left and +1 on the right
- Output can be interpreted as a probability

7-4

(?) Ask about max? ReLU: $f(z) = \max(0, z)$ (rectified linear unit)



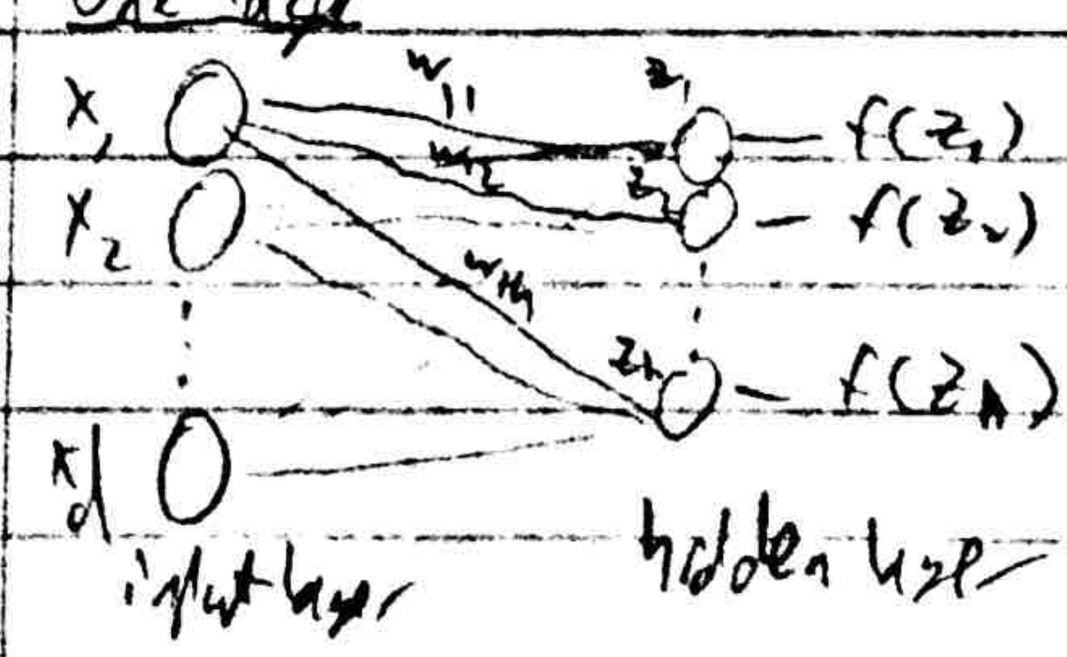
Notes:

- 0 for $z \le 0$, linear for $z \ge 0$
- Used most commonly because of simplicity and good derivative (no saturation)

Feed-forward neural networks

One layer

(?) Ask about matrix form



A layer can be conveniently written in vector and matrix form.

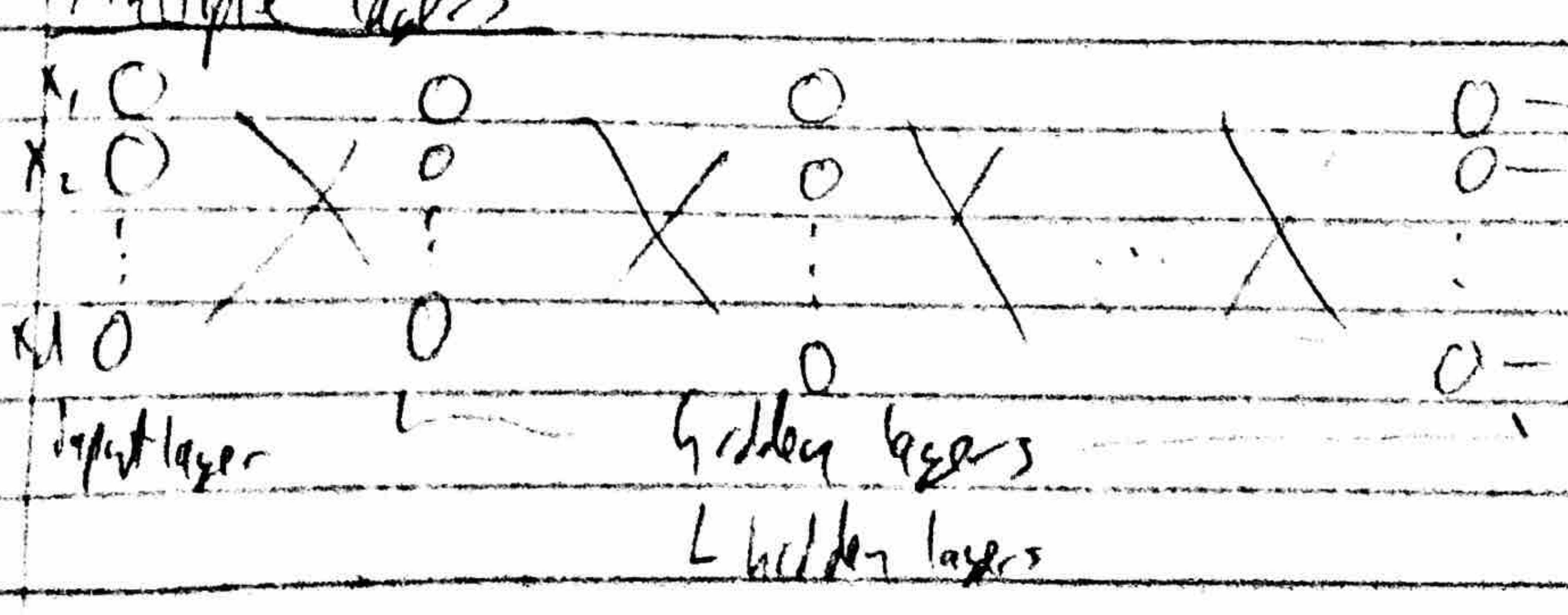
$$x^T = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} \qquad W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1h} \\ w_{21} & w_{22} & \cdots & w_{2h} \\ \vdots & & & \vdots \\ w_{d1} & w_{d2} & \cdots & w_{dh} \end{bmatrix}$$

$$z^T = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_h \end{pmatrix} \qquad A = \begin{pmatrix} f(z_1) \\ f(z_2) \\ \vdots \\ f(z_h) \end{pmatrix} \qquad \boxed{\begin{array}{l} z = XW \\ A = f(z) = f(XW) \end{array}}$$

Notes:

- Weights are labelled w_{ij} where i = index of previous node and j = index of next node
- h is the # of neurons in what is called the hidden layer
- h and d do not have to be equal (and rarely are)
- typically all neurons in the hidden layer use the same activation function

Multiple layers



Notes:
- Output of one hidden layer is used as the input of the next
- The # of neurons in each hidden layer do not need to match
- The same activation function is typically used across all hidden layers

(?) Ask for visual flow

In matrix form:

$$\hat{y}_L = f(z_L) = f(A_{L-1}W_L) = f(f(z_{L-1})W_L) = f(f(A_{L-2}W_{L-1})W_L)$$
$$= f(f(f(\dots f(f(XW_1)W_2)\dots)W_{L-1})W_L)$$

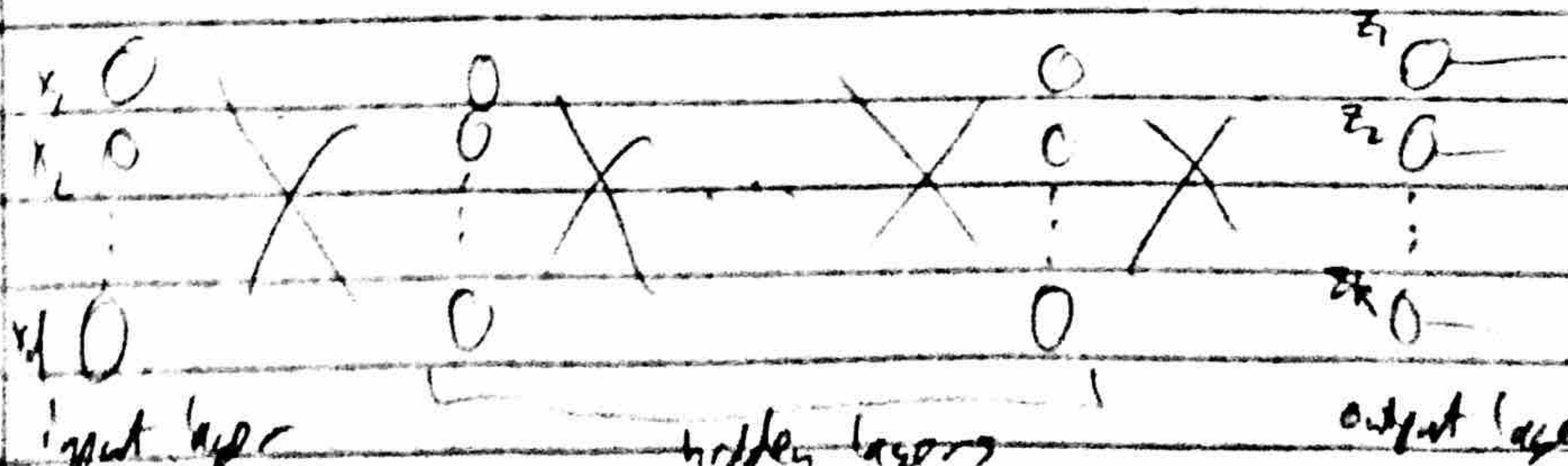
Side note:
If f was linear, then the above would be $XW_1W_2\dots W_{L-1}W_L = XW$ where $W = W_1W_2\dots W_{L-1}W_L$ which is just equivalent to a single layer neural network.
Therefore we need the non-linear functions in order for multiple layers to add power to the neural network.

Output layer
We need to take the output of the last layer of the neural network and
(?) somehow extract a prediction from it. How?

Idea: Add a special output layer.

For now we'll just talk about building an output layer for a classification problem.

First idea: output layer should have one neuron for each class.
Let's say we have k classes.



Second idea: Turn the outputs into a probability distribution.

z_1 O — p_1

z_2 O — p_2

z_k O — p_k

$p_1, p_2, \ldots, p_k$ represent the probability of that class

$0 \le p_i \le 1$

$p_1 + p_2 + \ldots + p_k = 1$

Then our prediction is simply the class with the greatest probability.

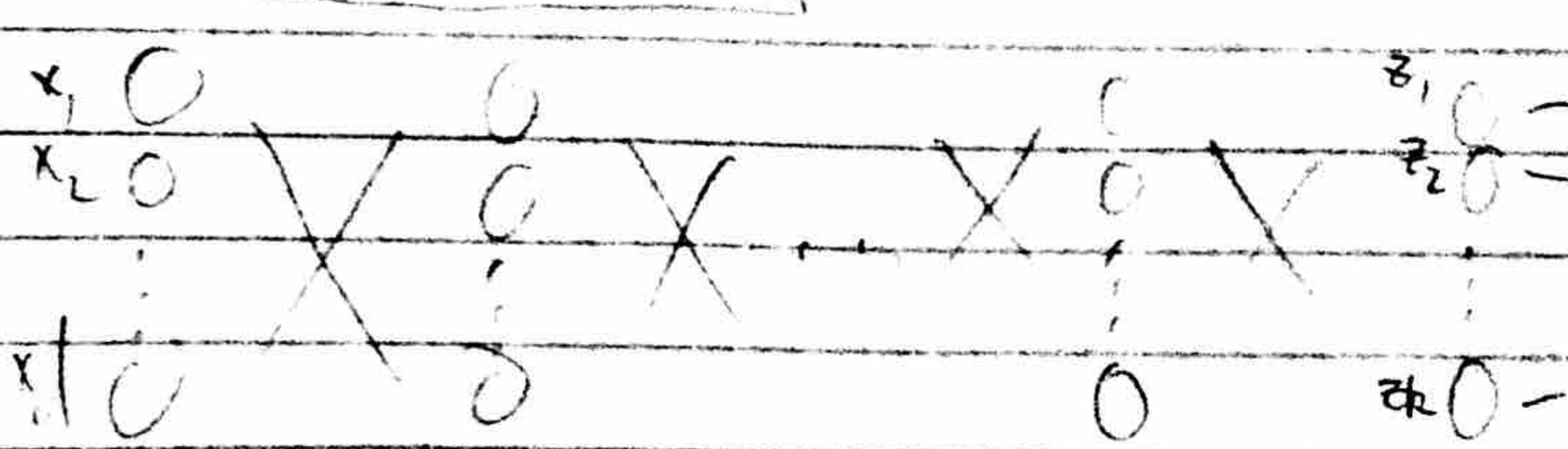
? What activation function could turn $z_1, z_2, \ldots, z_k$ into a probability distribution?

Desired attributes:

- Positive $\rightarrow e^{z_i}$
- Normalize to sum to one $\rightarrow \dfrac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$

Softmax: $\boxed{p_i = \dfrac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}}$

$\boxed{\text{Prediction} = \operatorname{argmax}_i p_i}$



$p_1 = \dfrac{e^{z_1}}{\sum_{j=1}^{k} e^{z_j}}$

$p_2 = \dfrac{e^{z_2}}{\sum_{j=1}^{k} e^{z_j}}$

$p_k = \dfrac{e^{z_k}}{\sum_{j=1}^{k} e^{z_j}}$

prediction $= \operatorname{argmax}_i p_i$

Regression: Single linear neuron   Probability: Single sigmoid neuron