INTRODUCTION TO DEEP LEARNING WS 18/19

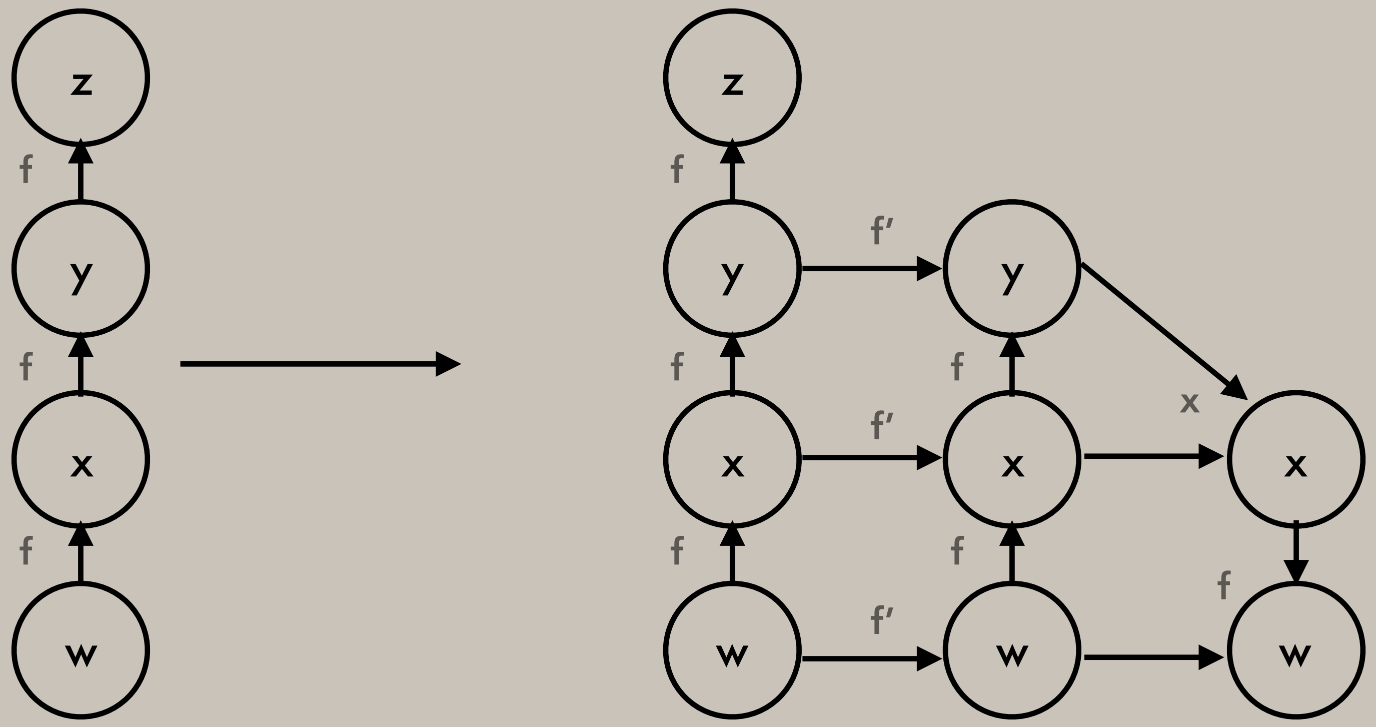
WEEK 1 RECAP

Kristin VOGEL
Tunç YILMAZ

BACK-PROPAGATION

Explain the back-propagation algorithm and its importance for neural network training!

- **Back-propagation** (backprop) refers to the method for computing the gradient of MLP for the given input data. In a next step, another algorithm such as stochastic gradient descent is used to perform learning using this gradient.
- In order to compute the gradient information from the cost flows backward through the network
- Backprop makes use of the chain rule and performs a Jacobian-gradient product for each operation in the graph. To speed up, backprop calculates each derivative just once and saves it, to make it available for repeated use. This proceeding comes at the costs of memory space.
- The following figure illustrates backprop in a computational graph:



GRADIENT DESCENT

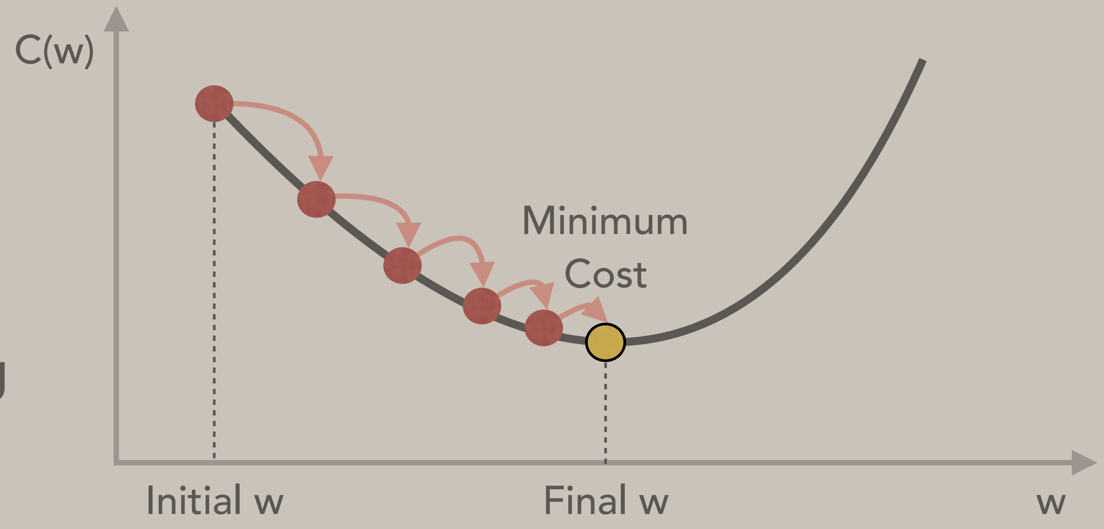
<u>How does gradient descent work? Bonus: What are the differences between stochastic, batch and mini-batch gradient descent?</u>

- Suppose C(**θ**) is the cost function of a neural network where **θ** defines the parameter space of the network consisting of all weights and biases. **∇**C(**θ**) stands for the derivative vector consisting of all the partial derivatives of the cost function with respect to every element of **θ**. In such a case, the parameter space **θ** is updated by **gradient descent** as:

$$\boldsymbol{\theta}' = \boldsymbol{\theta} - \alpha \nabla C(\boldsymbol{\theta})$$

- We can imagine it's working principle by considering a very simple case where the cost function consists of only a single weight, w.

- In gradient descent, the cost function consists of all training examples, which is computationally expensive.

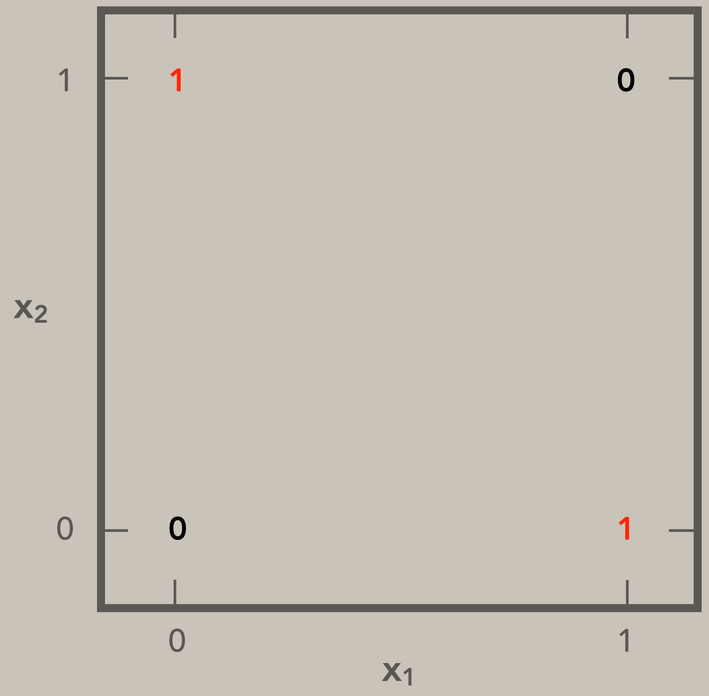


- **Stochastic gradient descent** aims to tackle this problem by randomly selecting a **mini-batch** of training examples, where mini-batch is a subset of the whole **batch** used in gradient descent. In this way, it takes more steps to reach the minimum of the cost function, yet the computational work is much less compared to gradient descent.

NON-LINEARITY

Why do we need non-linearities?

- Remember the XOR function represented below. The function takes on two binary values (x1 and x2) , and returns 1 when any one of these values is 0, and returns 0 otherwise:



- In the original **x** space, say, when $x_1 = 0$, y must increase as x_2 increases. Also when $x_1 = 1$, y must decrease as x_2 increases. A linear model must assign a fixed coefficient w_2 to x_2 and at the same time cannot use x_1 to change w_2. Therefore it cannot simply solve this problem. As it can be exemplified by the XOR function, some problems therefore require **non-linear** models for a thorough solution.

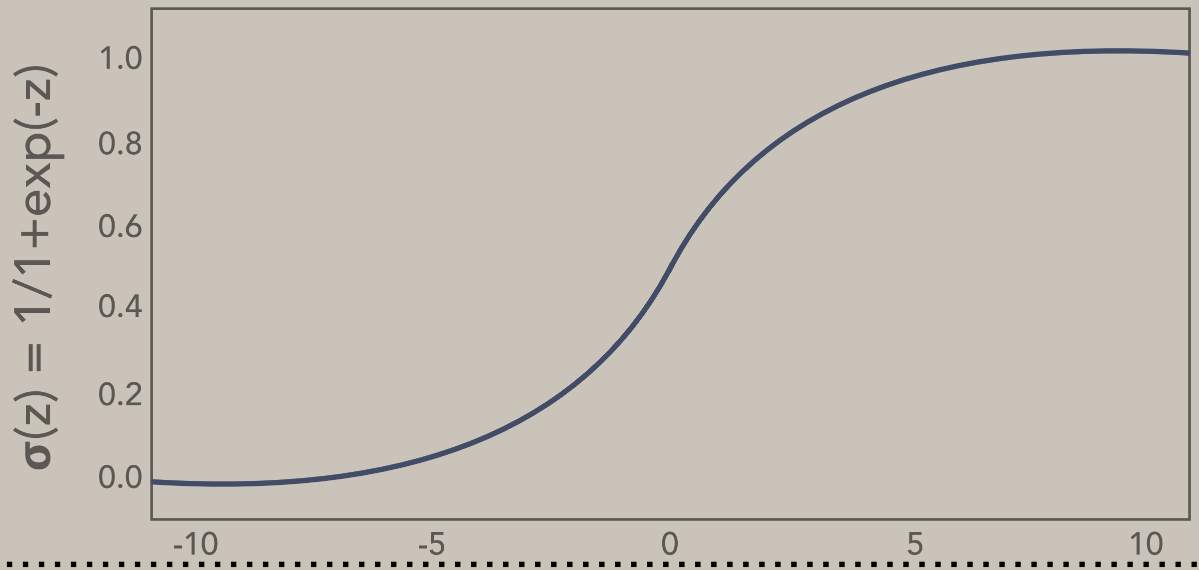
ACTIVATION FUNCTIONS

What activation functions do you know? Sketch them and discuss their pros and cons!

SIGMOID

Can be used for probabilities = range is (0,1)

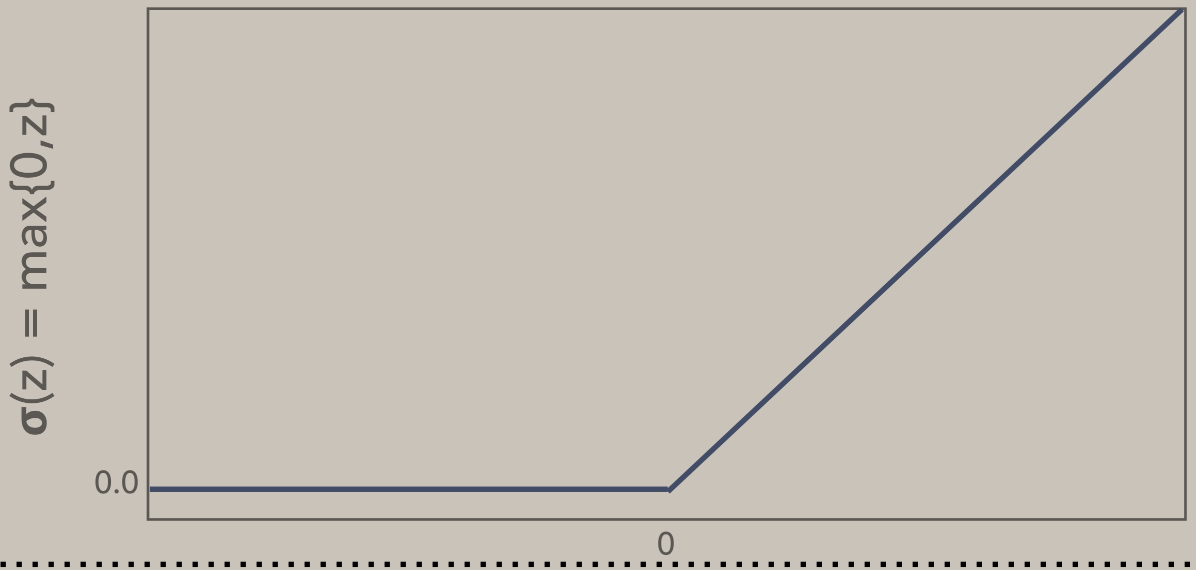
Insensitive to small changes in its input at extremes



ReLU

Although being nonlinear, preserves some linear qualities

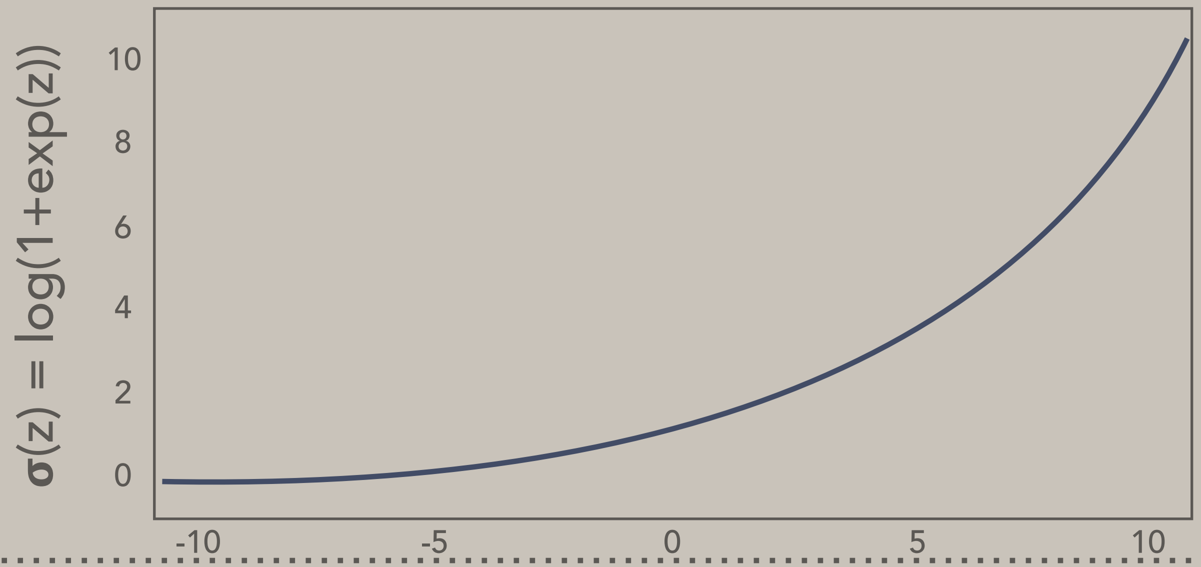
Units must be activated initially by assigning small weights



SOFTPLUS

Smoother than ReLU near 0, its range is (0,∞)

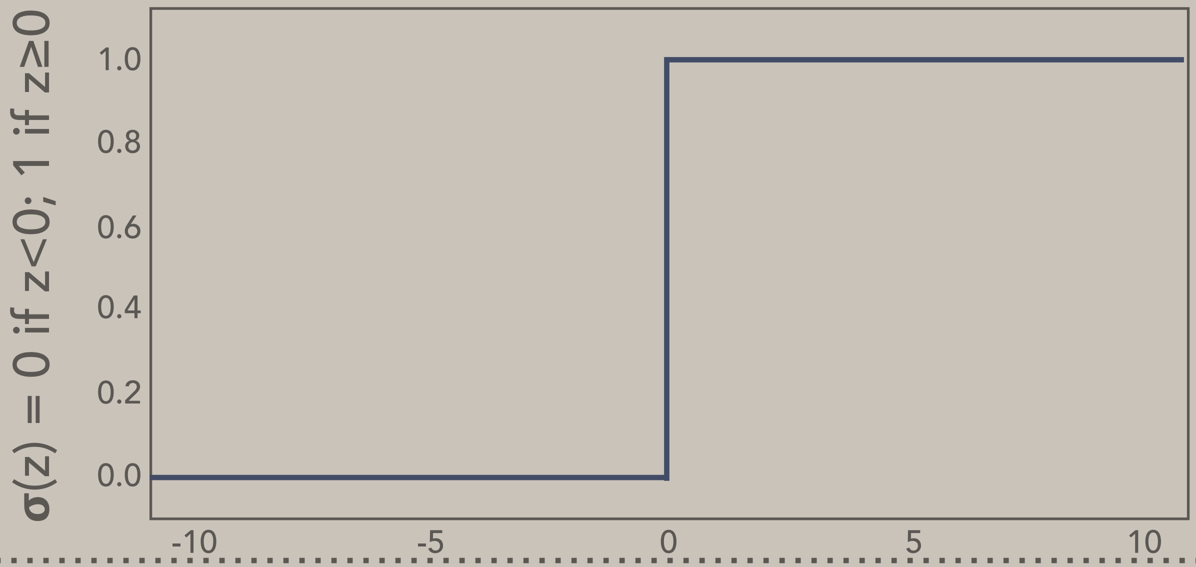
Derivative is difficult since it contains log



PERCEPTRON

Very simple

Incompatible with back propagation



CHOICE OF OUTPUT UNITS

<u>How do the choice of output units and the loss function relate?</u>

- In general, we seek a loss function with a range of output values that comply with the actual output values of the dataset.
- For example:
  - if we have a classification problem, actual outputs will be distributed with probabilities ranging between 0 and 1. In such a case, sigmoid or perceptron might be relevant.
  - if we have a regression problem, an activation function like softplus that can yield a range of values greater than 1 might be used.

Output Type	Output Distribution	Output Layer	Cost Function
Binary	Bernouilli	Sigmoid	Binary Cross-Entropy
Discrete	Multinouilli	Softmax	Discrete Cross-Entropy
Continuous	Gaussian	Linear	Gaussian Cross-Entropy (MSE)
Continuous	Mix of Gaussian	Mixture Density	Cross-Entropy
Continuous	Arbitrary	See Part III	Various

NEGATIVE LOG-LIKELIHOOD

Why is the negative log-likelihood a popular choice as loss function?

- In neural networks, the gradient of the loss function is expected to be large and predictable enough in order to facilitate the job of the learning algorithm.
- Activation functions that saturate at certain points do not meet this expectation.
- An example of such a function might be one that involves an exponential component which results in saturation for significantly negative values.
- Negative log-likelihood helps avoid this problem by undoing the exponential component of output units.

MAXOUT

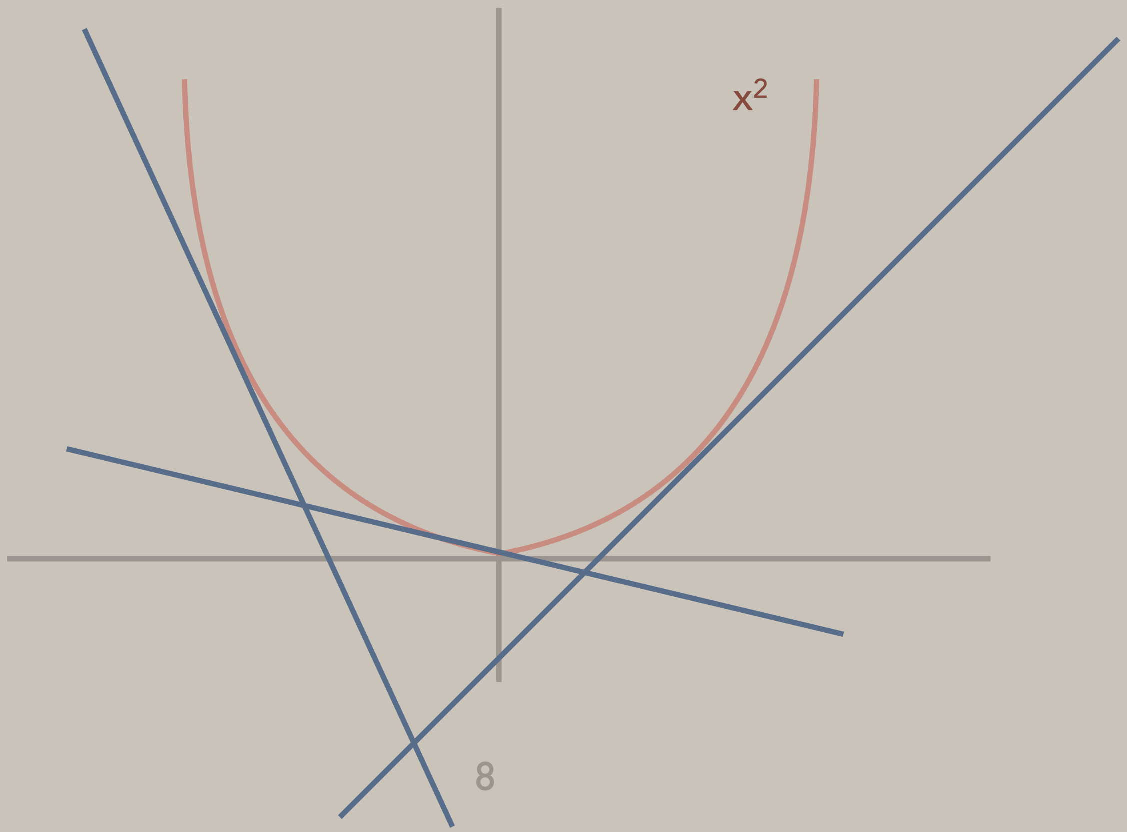
What is Maxout? Sketch a layer with Maxout activation!

- Maxout is a piecewise linear approximation to any convex function.
- It a generalisation of rectified linear units that divide z into k values instead of applying an element-wise function g(z). Maxout units gives the maximum element of one of these groups as follows:

$g(z)_i = \max_{j \in G^{(i)}} z_j$, where $G^{(i)}$ is the set of indices into the inputs for group i

- Each mahout unit is parametrised by k weight vectors instead of 1, so they require more regularisation in comparison to ReLU. A simple example can be given as:

Here, maxout approximates the x^2 function by 3 linear functions. The maximum of these functions will be given by the maxout unit



LINEAR HIDDEN LAYER

Why use a linear hidden layer?

- A simple kind of output unit that uses an affine transformation with no nonlinearity such as $\mathbf{y} = \mathbf{W}^T\mathbf{h} + \mathbf{b}$ is called a linear unit.
- Linear hidden layers consisting of such units are preferred when the output of the network tried to produce the mean of a conditional Gaussian distribution. In such a case, maximizing the log-likelihood is equivalent to minimising the MSE.
- Since linear units do not saturate, they cause no difficulty for gradient descent based learning algorithms.

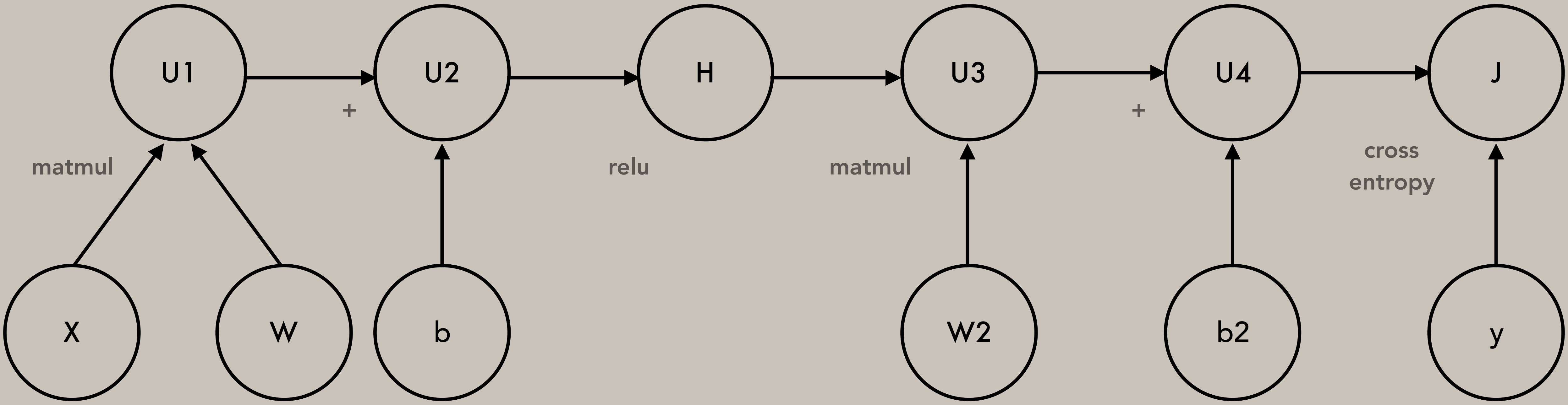
UNIVERSAL APPROXIMATION THEOREM

What does the universal approximation theorem (practically) mean?

- The principle of **universal approximation theorem** is that any continuous function (linear or non-linear) on a closed and bounded subset of $\mathbb{R}^n$ can be approximated by a feedforward neural network with a linear output layer and at least one hidden layer with a squashing activation function (such as a sigmoid function that squeezes in values to a range of (0,1))
- It practically means that regardless of the properties of the function that we're trying to learn, a large feedforward network suffices to represent or approximate this function.
- However we should keep in mind that it is an approximation rather than a guaranteed learning with generalization.

COMPUTATIONAL GRAPHS

Sketch and explain the computation graph for the cross-entropy loss of an MLP with 1 hidden layer!


U1
U2
H
U3
U4
J
matmul
+
relu
matmul
+
cross entropy
X
W
b
W2
b2
y