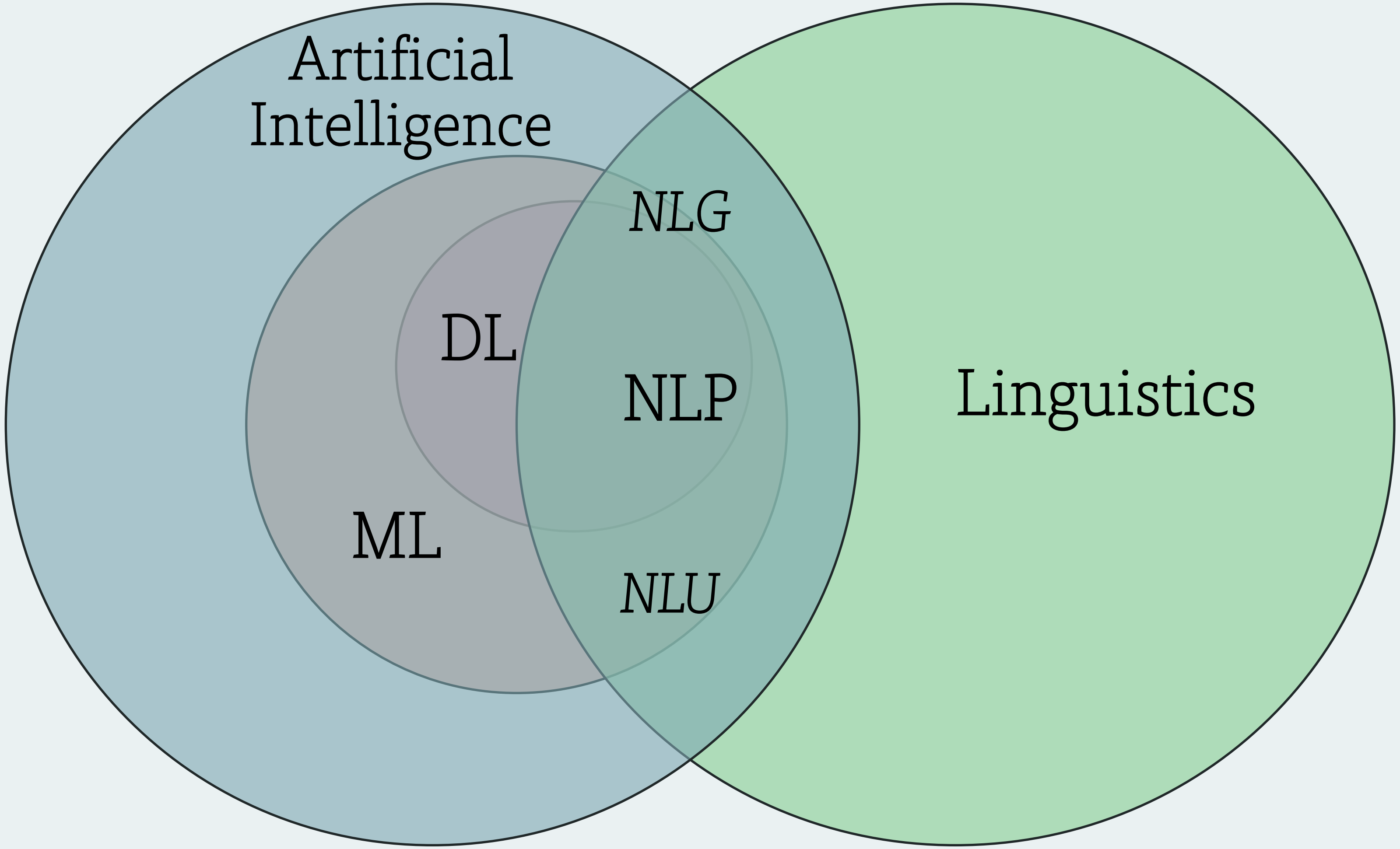
NLP for Social Sciences

5. Neural Networks and Language Modelling

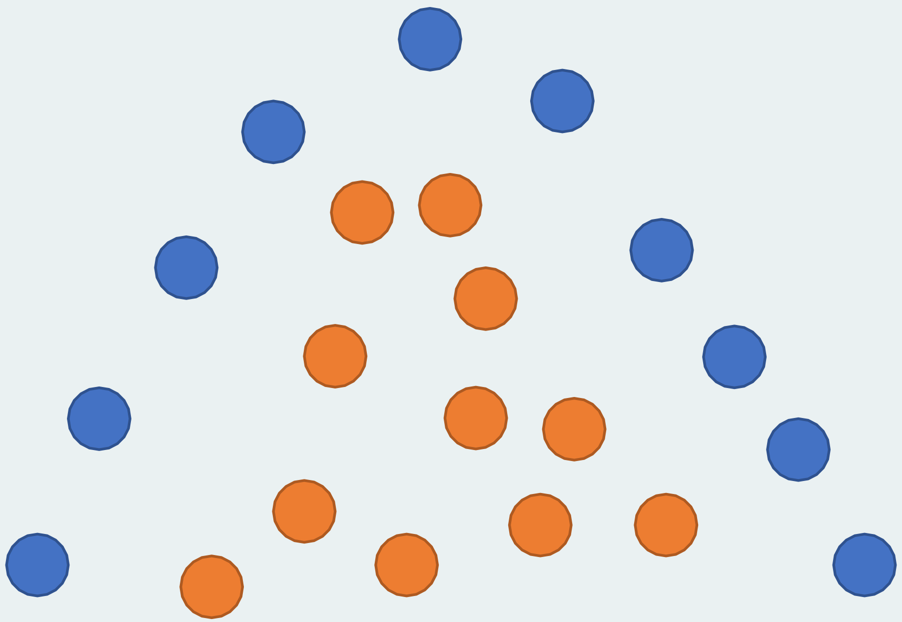
Irina Proskurina, Université de Lyon, de Lyon 2, Laboratoire ERIC

1. Neural Networks

Natural Language Processing

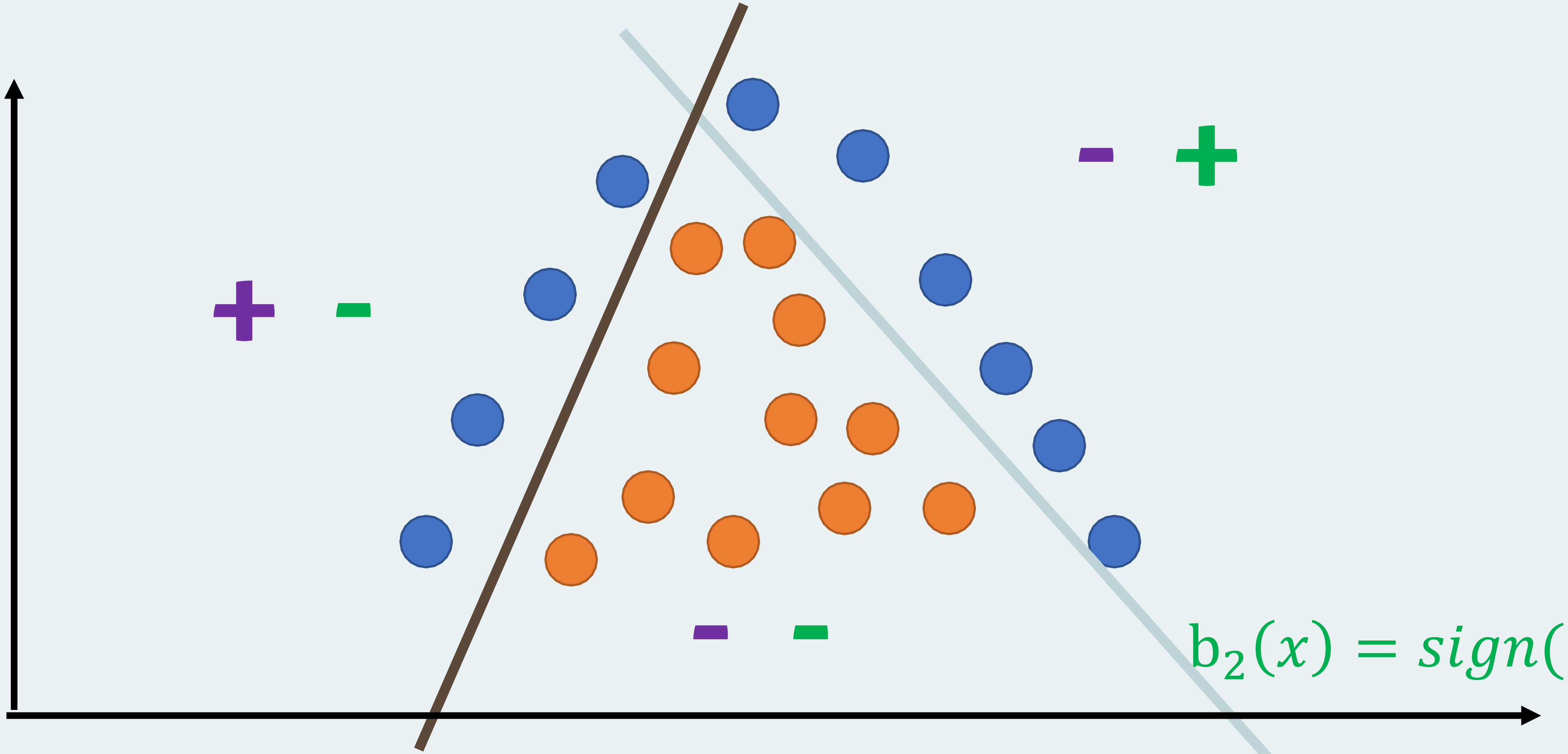


Neural Network Structure



Nonlinear patterns

Neural Network Structure

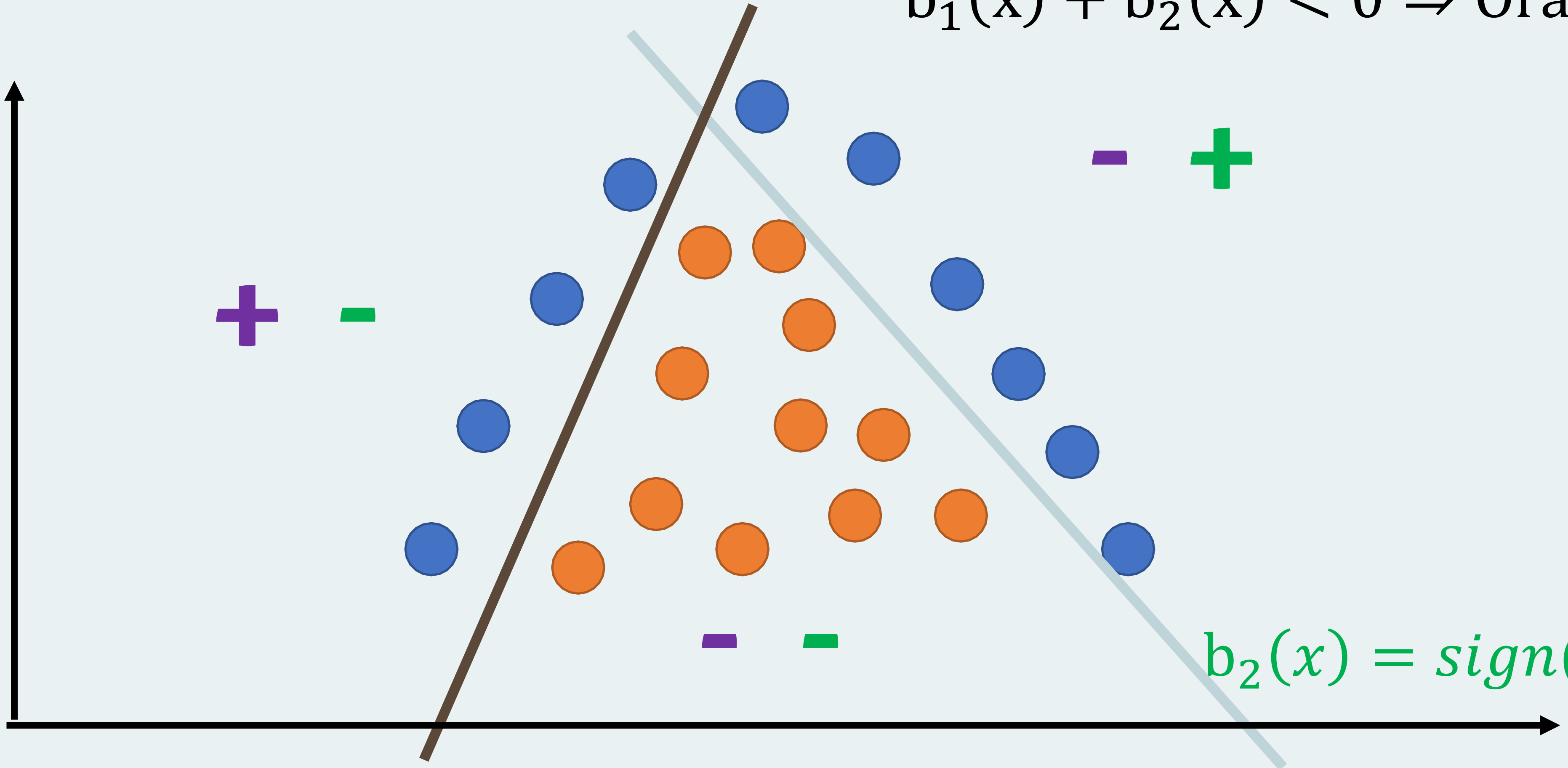


$b_2(x) = sign(2x_1 + x_2 -$

$b_1(x) = sign(-2x_1 + x_2 + 5)$

Neural Network Structure

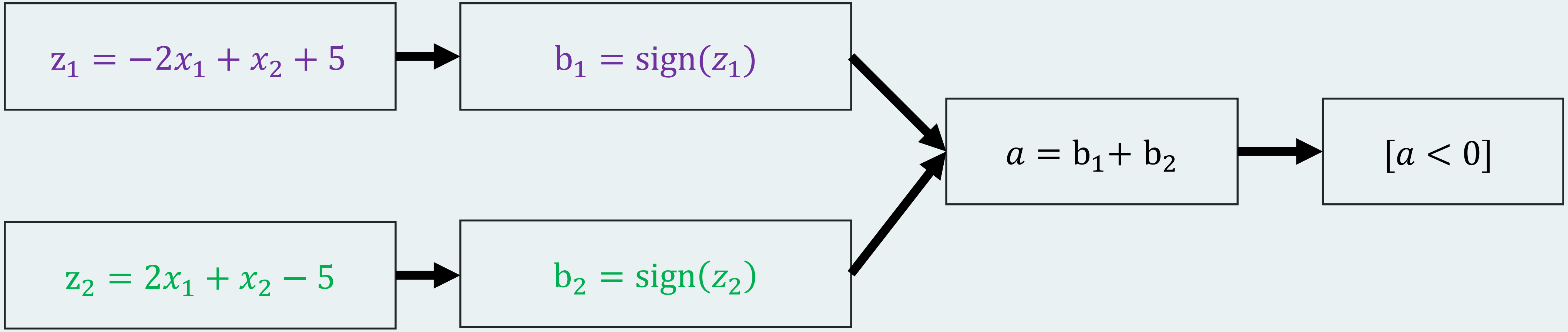
$$\mathrm{b}_1(\mathrm{x}) + \mathrm{b}_2(\mathrm{x}) < 0 \Rightarrow \text{Orange class}$$



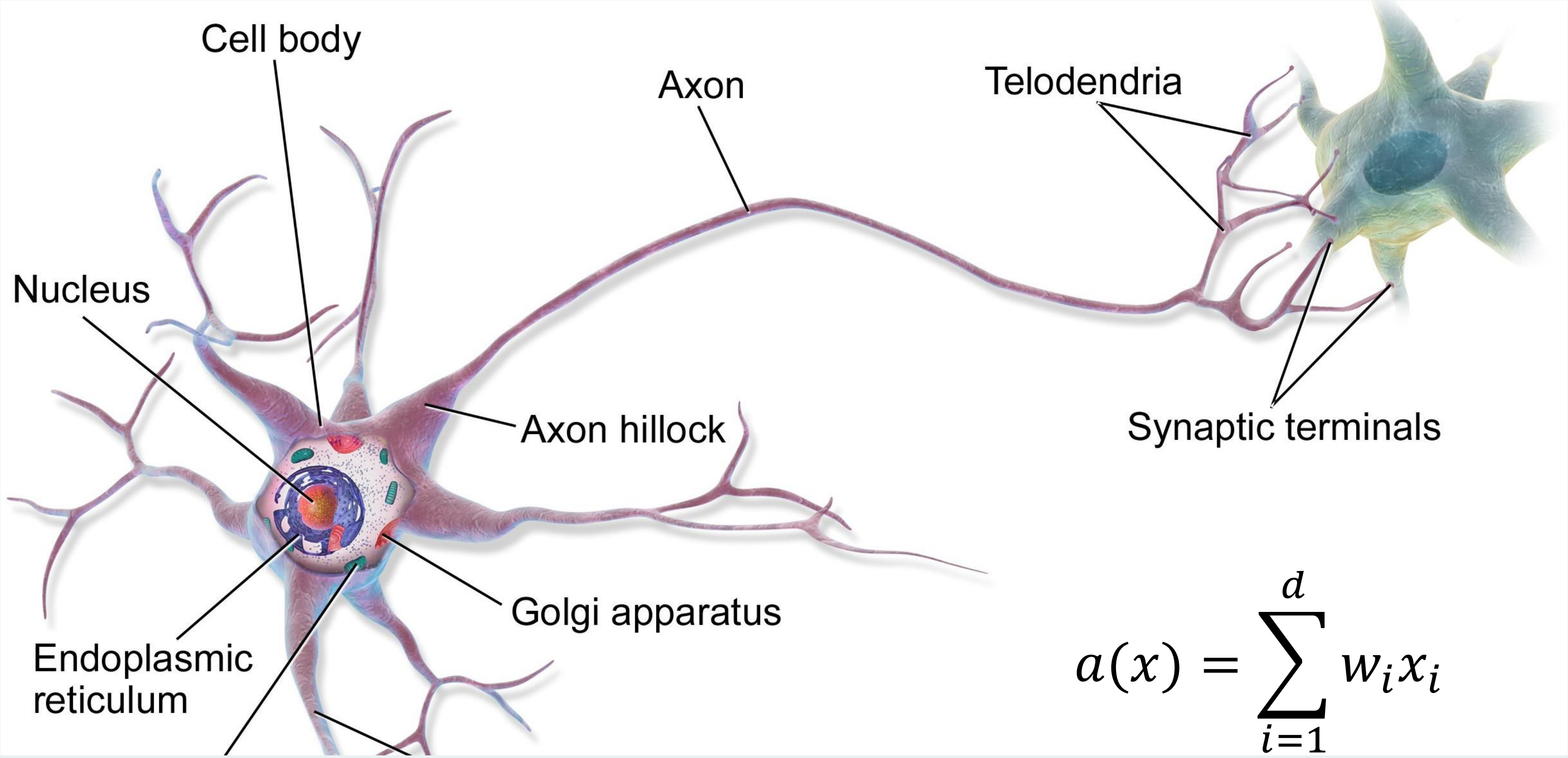
$$\mathrm{b}_2(x) = sign(2x_1 + x_2 -$$

$$\mathrm{b}_1(x) = sign(-2x_1 + x_2 + 5)$$

Neural Network Structure



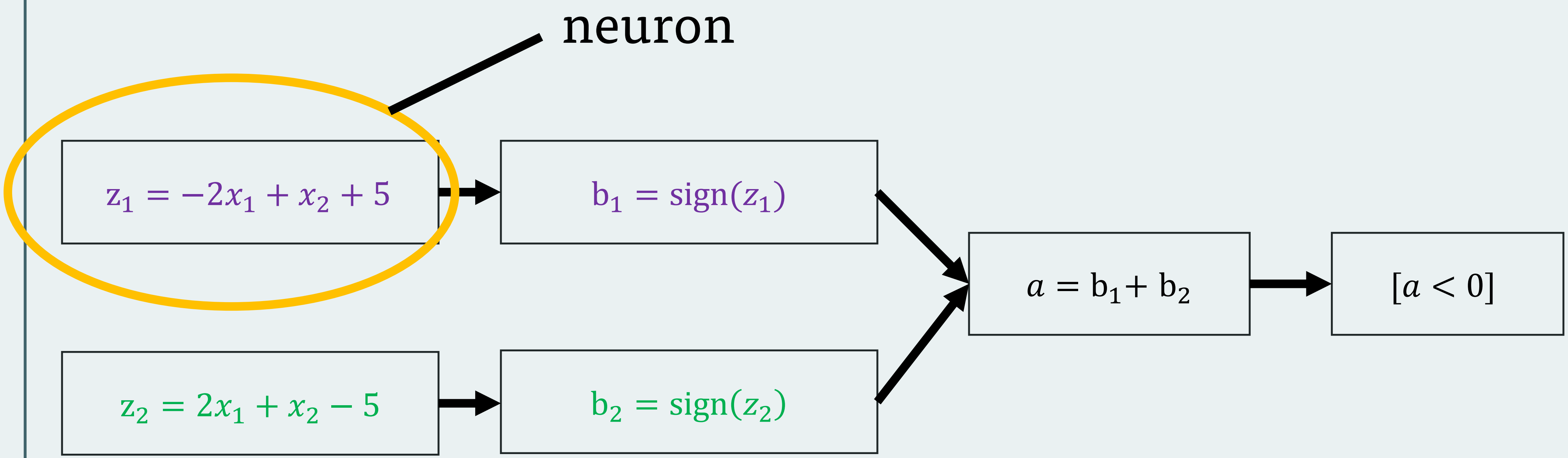
Neuron



$$a(x) = \sum_{i=1}^{d} w_i x_i$$

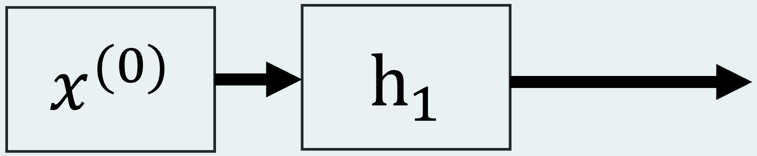
https://en.wikipedia.org/wiki/Neuron

Neural Network Structure

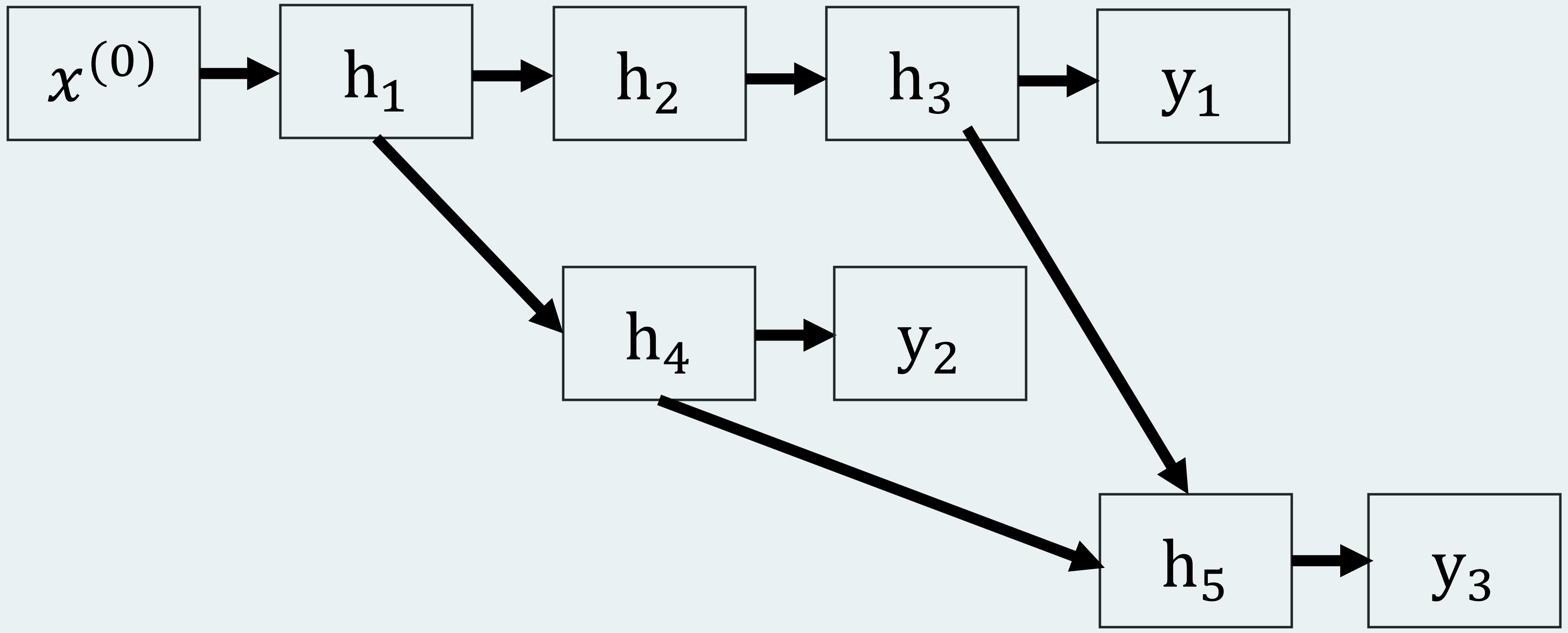


Computational graph (neural network)

- $x^{(0)}$ — input features (embedding)
- h_1 — hidden layer
- $x^{(1)}$ — output



Computational graph (neural network)



2. Fully connected layers

Fully connected layers

- The input consists of N values, and the output produces M values
- $x_1, \cdots x_N$ — input
- $y_1, \cdots y_M$— output

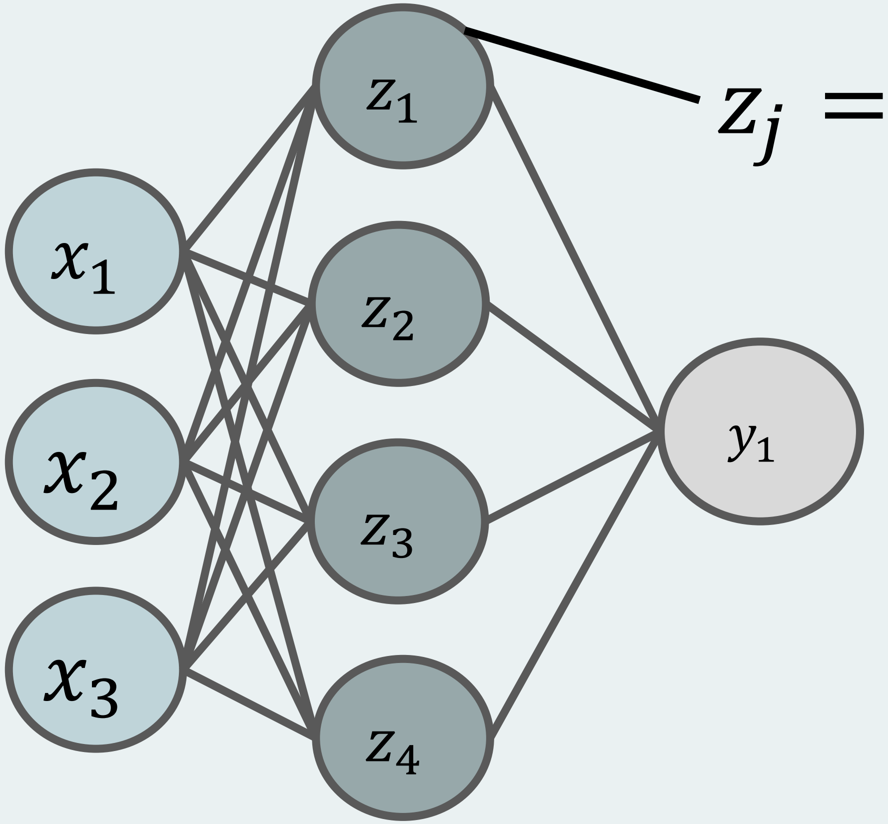
- Each output is the result of applying a linear model to the inputs.

$$z_j = \sum_{i=1}^{n} w_{ji} x_i + b_j$$

Fully connected layers

$$z_j = \sum_{i=1}^{n} w_{ji} x_i + b_j$$

1 Input layer



3 Output layer

2 Hidden layer

https://cs231n.github.io/convolutional-networks/

Fully connected layers

$$z_j = \sum_{i=1}^{n} w_{ji} x_i + b_j$$

- m linear models, each with $(n + 1)$ parameters
- in total, there are approximately mn parameters in a fully connected layer

torch.nn.Linear(20, 30)

keras.layers.Dense(64)

Fully connected layers

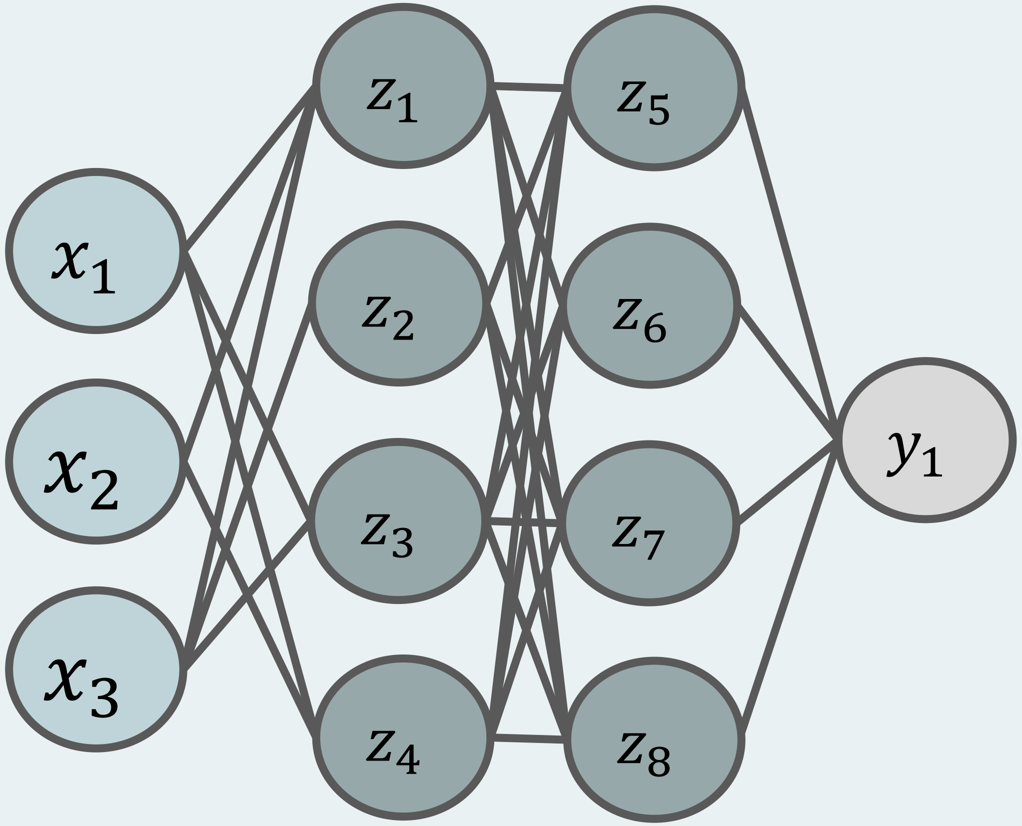
$$z_j = \sum_{i=1}^{n} w_{ji} x_i + b_j$$

- m linear models, each with $(n + 1)$ parameters
- in total, there are approximately mn parameters in a fully connected layer
- if we have 1,000,000 input features and 1000 outputs, it amounts to 1,000,000,000 parameters
- a substantial amount of data is needed for training

Important Questions in DL

How to build a useful model in deep learning?
Which layers to add?

Fully connected layers



- Can we have 2 fully-connected layers one after another?

Non-linearity

- Given 2 fully-connected layers

$$s_k = \sum_{j=1}^{m} v_{kj} z_j + c_k = \sum_{j=1}^{m} v_{kj} \sum_{j=1}^{m} w_{ji} x_i + \sum_{j=1}^{m} v_{kj} b_j + c_k =$$

$$= \sum_{j=1}^{m} (\sum_{j=1}^{m} \boldsymbol{v_{kj} w_{ji}} x_i + \boldsymbol{v_{kj} b_j} + \frac{\mathbf{1}}{\boldsymbol{m}} \boldsymbol{c_k})$$

$$\boxed{z_j = \sum_{i=1}^{n} \boldsymbol{w_{ji}} x_i + \boldsymbol{b_j}}$$

- **So, this is no better than a single fully connected layer**

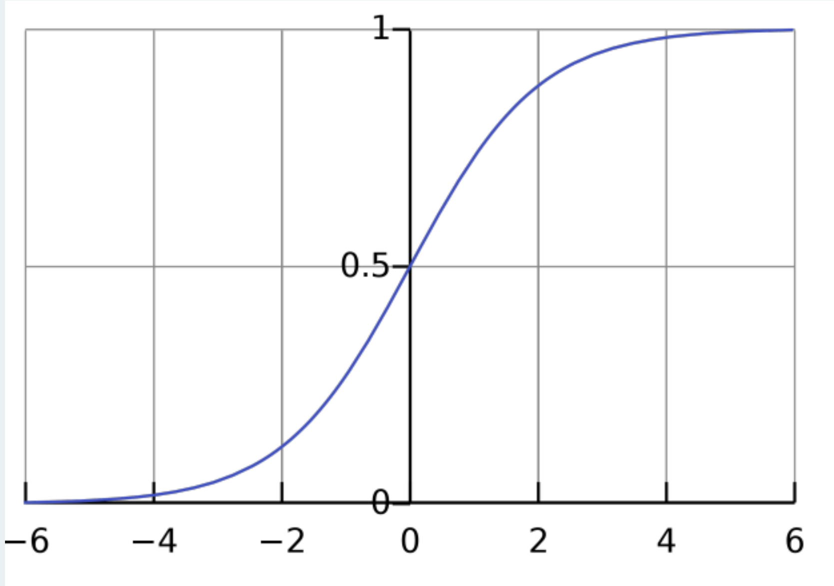
Activation functions

- It is necessary to add a non-linear activation function after the fully connected layer (`torch.nn.Sigmoid`)

$$z_j = f(\sum_{i=1}^{n} w_{ji}x_i + b_j)$$

1. $f(x) = \dfrac{1}{1 + \exp(-x)}$

Logistic / Sigmoid



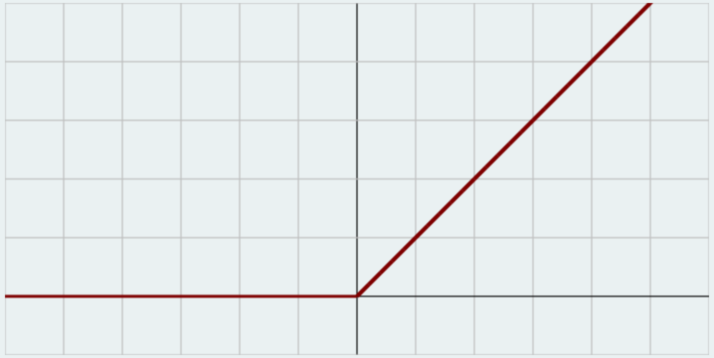
Activation functions

- It is necessary to add a non-linear activation function after the fully connected layer (`torch.nn.ReLU`)

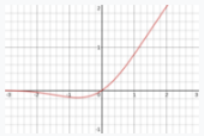
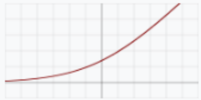
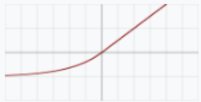
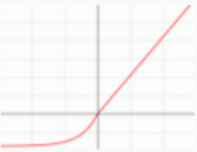
$$z_j = f(\sum_{i=1}^{n} w_{ji} x_i + b_j)$$

2. $f(x) = \max(0, x)$

(ReLU, REctified Linear Unit)



Activation Functions

Rectified linear unit (ReLU)[8]		$(x)^{+} \doteq \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} = \max(0, x) = x\mathbf{1}_{x>0}$
Gaussian Error Linear Unit (GELU)[2]		$\frac{1}{2}x\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right) = x\Phi(x)$
Softplus[9]		$\ln(1 + e^{x})$
Exponential linear unit (ELU)[10]		$\begin{cases} \alpha\left(e^{x} - 1\right) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α
Scaled exponential linear unit (SELU)[11]		$\lambda \begin{cases} \alpha(e^{x} - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ with parameters $\lambda = 1.0507$ and $\alpha = 1.67326$

https://en.wikipedia.org/wiki/Activation_function

A fully connected neural network

$$\boxed{x^{(0)}} \rightarrow \boxed{\mathrm{FC}_1} \rightarrow \boxed{f} \rightarrow \boxed{\mathrm{FC}_2} \rightarrow \boxed{f} \rightarrow \cdots$$

A fully connected neural network

$$\boxed{x^{(0)}} \rightarrow \boxed{\mathrm{FC}_1} \rightarrow \boxed{f} \rightarrow \boxed{\mathrm{FC}_2} \rightarrow \boxed{f} \rightarrow \cdots$$

- Features are fed into the network
- The dimension of the last layer = # target variables (# classes)

Two-layer network with **scalar** output

$y = \sigma(z)$ y is a scalar

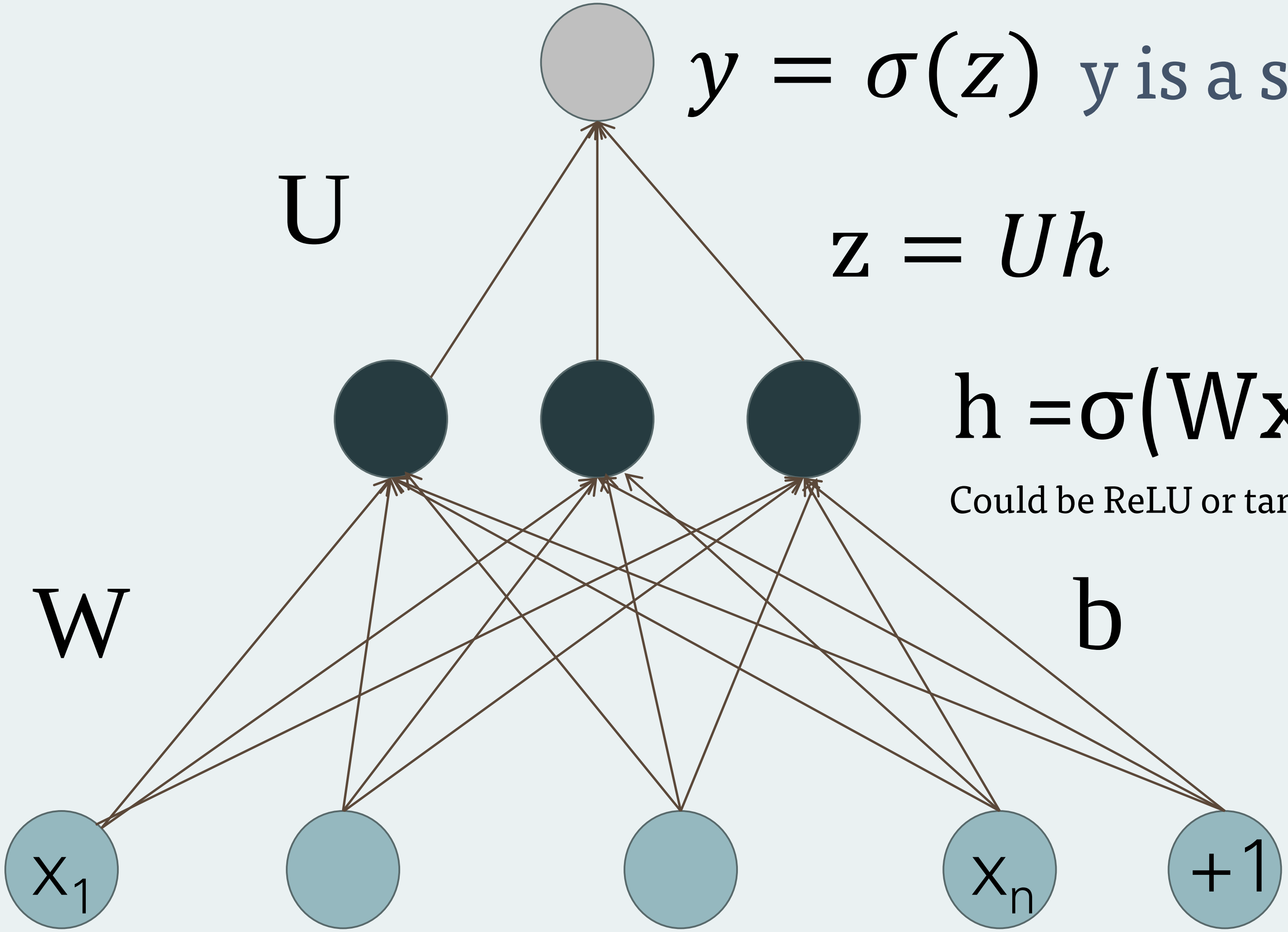
Output layer
(σ node)

$z = Uh$

$h = \sigma(Wx+b)$

Could be ReLU or tanh

Hidden units
(σ node)



Input layer
(vector)

Two-layer network with **softmax** output

$y = \text{softmax}(z)$
$z = Uh$

y is a vector

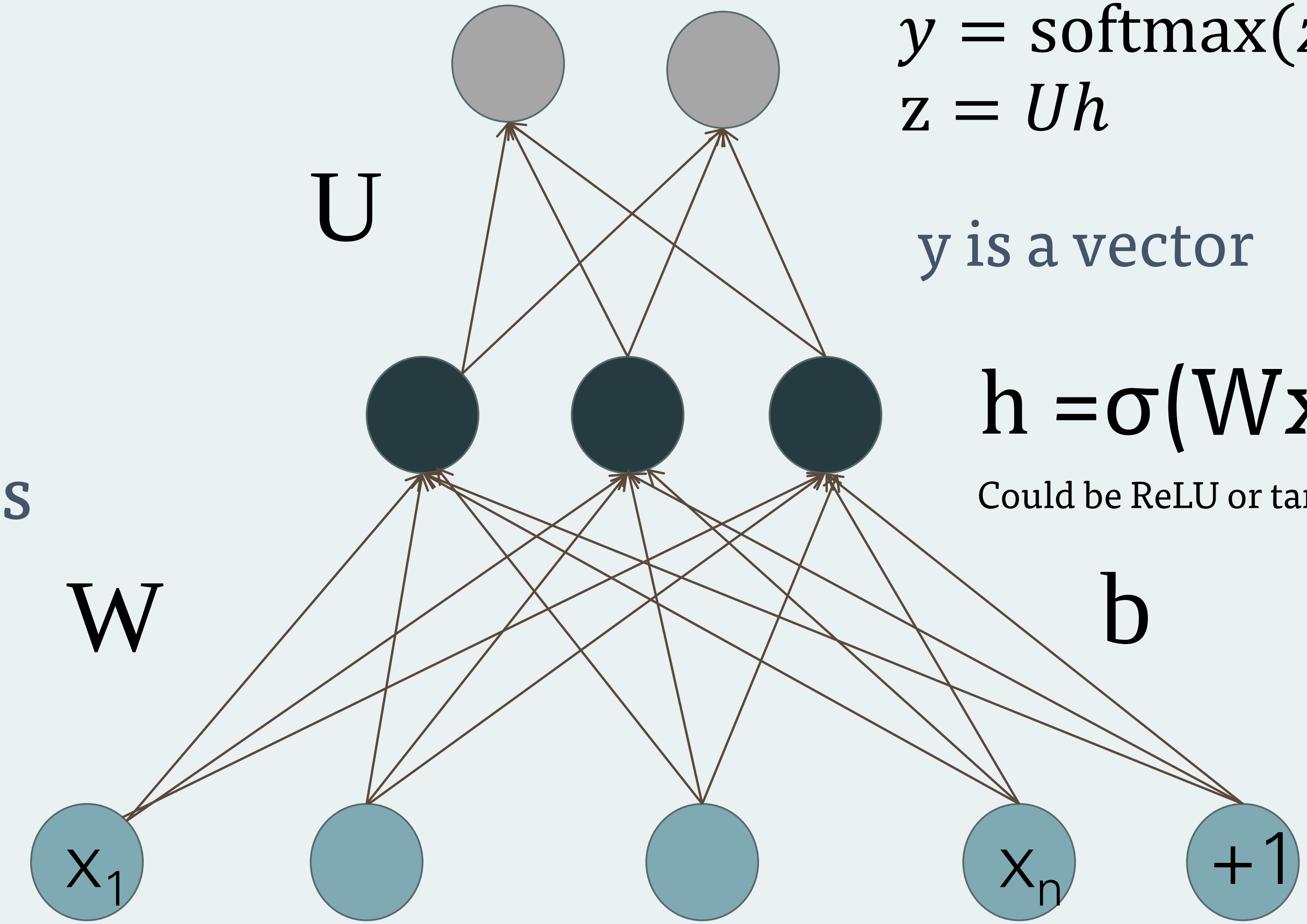
$h = \sigma(Wx+b)$

Could be ReLU or tanh

Output layer
(σ node)

Hidden units
(σ node)

Input layer
(vector)



The Cybenko universal approximation theorem

Summary:

- Let $g(x)$ be a continuous function. Then, it is possible to construct a two-layer neural network that approximates $g(x)$ with any predefined precision.
- In other words, two-layer neural networks are VERY powerful

3. Training neural networks

Warm-up

Which of this is the formula for a step in gradient descent?

1. $\mathrm{w}^{\mathrm{t}} = \mathrm{w}^{\mathrm{t}-1} + \eta\nabla Q(\mathrm{w}^{\mathrm{t}})$
2. $\mathrm{w}^{\mathrm{t}} = \mathrm{w}^{\mathrm{t}-1} - \eta\nabla Q(\mathrm{w}^{\mathrm{t}-1})$
3. $\mathrm{w}^{\mathrm{t}} = \mathrm{w}^{\mathrm{t}-1} - \eta\nabla Q(\mathrm{w}^{\mathrm{t}})$
4. $\mathrm{w}^{\mathrm{t}} = \mathrm{w}^{\mathrm{t}-1} + \eta\nabla Q(\mathrm{w}^{0})$

Convergence

- Stopping rule 1

$$\|w^t - w^{t-1}\| < \varepsilon$$

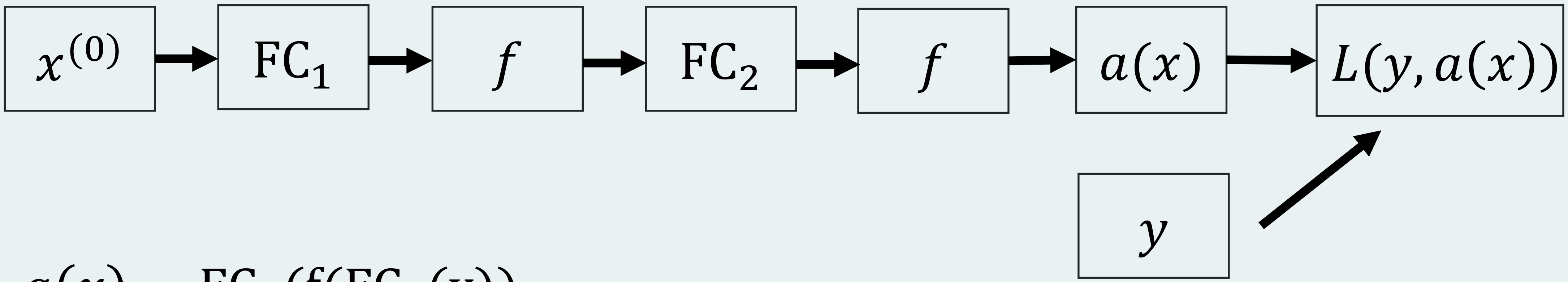
- Stopping rule 2

$$\|\nabla Q(w^t)\| < \varepsilon$$

- In DL: stop when the error on the test set stops decreasing

Training neural networks

- All layers are usually possible to differentiate, so it's possible to compute derivatives with respect to all parameters

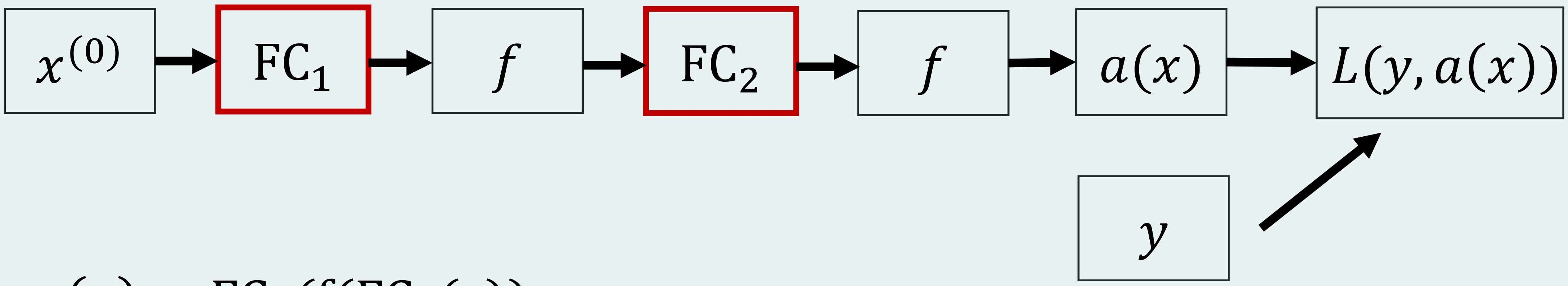


- $a(x) = \mathrm{FC}_2(\mathrm{f}(\mathrm{FC}_1(\mathrm{x}))$

- Where are the parameters in this neural network?

Training neural networks

- All layers are usually possible to differentiate, so it's possible to compute derivatives with respect to all parameters



- $a(x) = \mathrm{FC}_2(\mathrm{f}(\mathrm{FC}_1(\mathrm{x}))$
- Where are the parameters?

$$\frac{1}{\ell}\sum_{i=1}^{\ell} L(y_i, a(x_i)) \to \min_a$$

Computing derivatives

- For gradient descent, derivatives of the error with respect to the parameters are needed:

$$\frac{\partial}{\partial w_j}(a(x_i, w) - y_i)^2$$

$$\frac{\partial}{\partial w_j}(a(x_i, w) - y_i)^2 = 2(a(x_i, w) - y_i)\frac{\partial}{\partial w_j}a(x_i, w)$$

Computing derivatives

- For gradient descent, derivatives of the error with respect to the parameters are needed:

$$\frac{\partial}{\partial w_j}(a(x_i,w)-y_i)^2 = 2(a(x_i,w)-y_i)\frac{\partial}{\partial w_j}a(x_i,w)$$

- $a(x_i,w)=10, y_i=9.99$: $\quad 2*0.01*\frac{\partial}{\partial w_j}a(x_i,w)$
- $a(x_i,w)=10, y_i=1$: $\quad 2*9*\frac{\partial}{\partial w_j}a(x_i,w)$

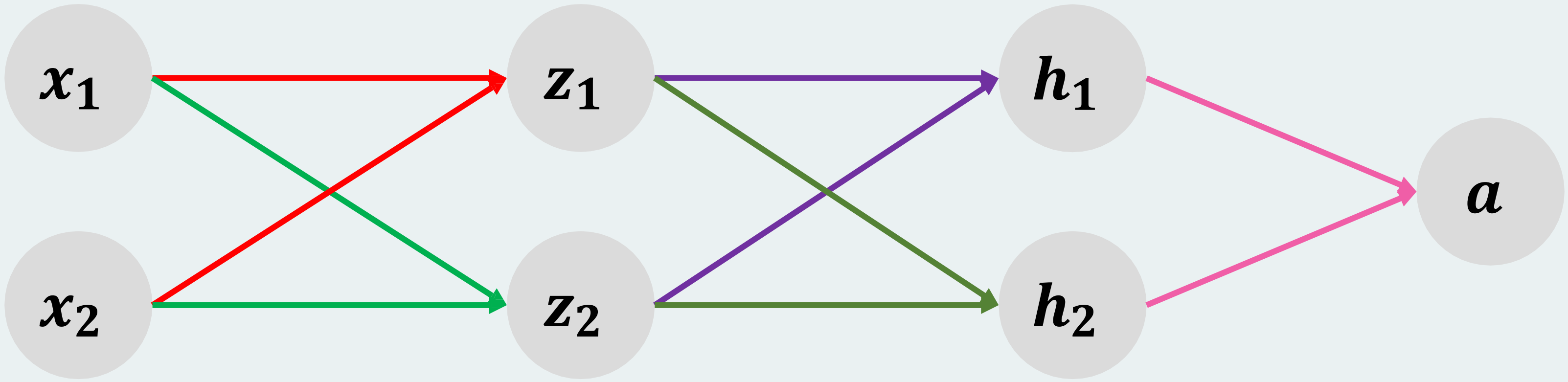
Computing derivatives

- For gradient descent, derivatives of the error with respect to the parameters are needed:

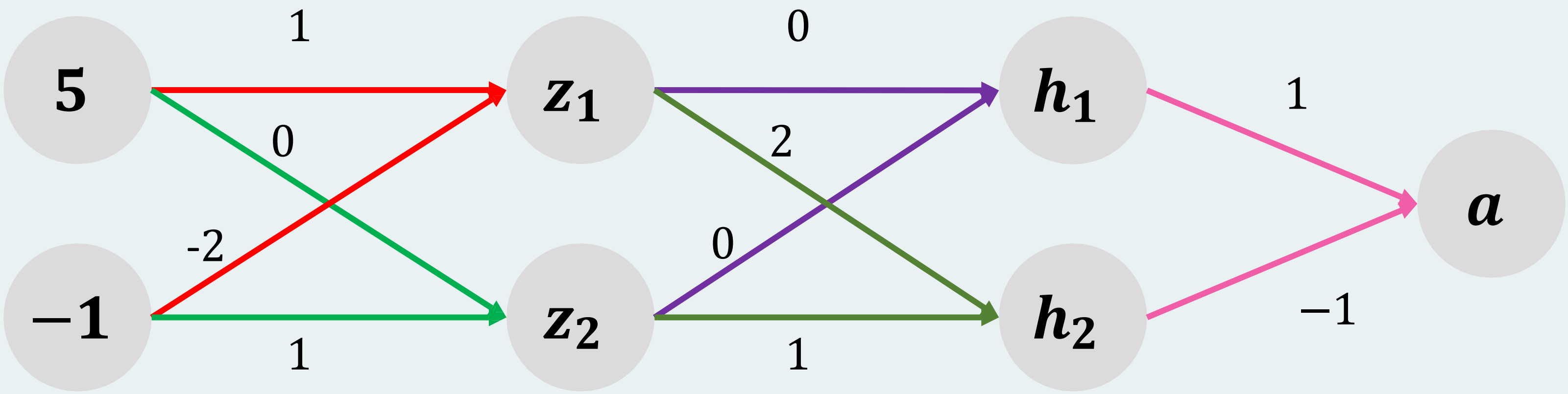
$$\frac{\partial}{\partial w_j}(a(x_i, w) - y_i)^2 = 2(a(x_i, w) - y_i)\frac{\partial}{\partial w_j}a(x_i, w)$$

$$\frac{\partial}{\partial w_j}L(y_i, a(x_i, w)) = \frac{\partial}{\partial z}L(y_i, z)\Big|_{z=a(x_i, w)}\frac{\partial}{\partial w_j}a(x_i, w)$$

How to compute derivatives?

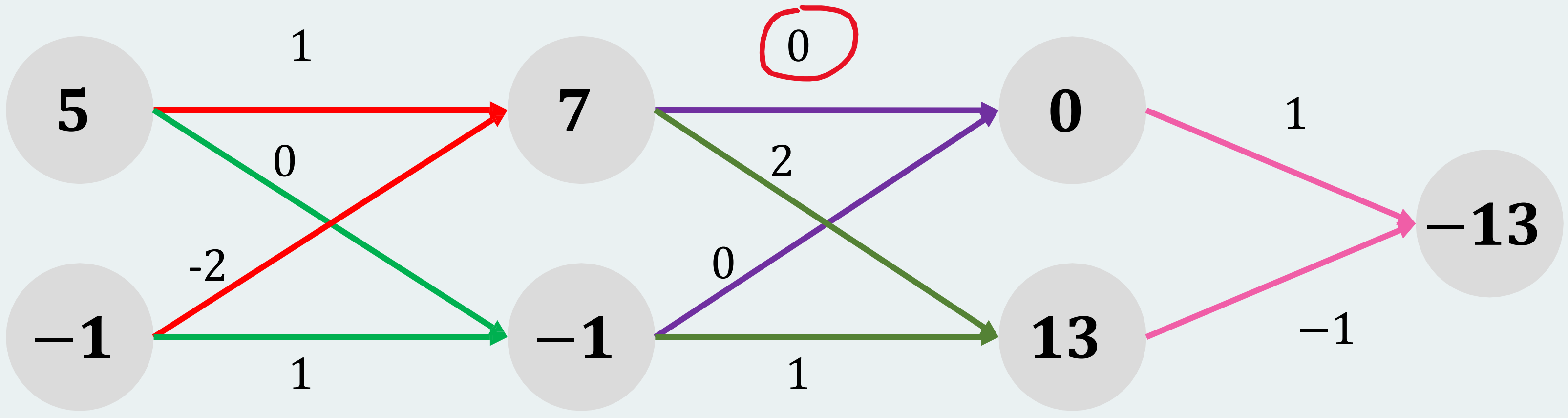


How to compute derivatives?

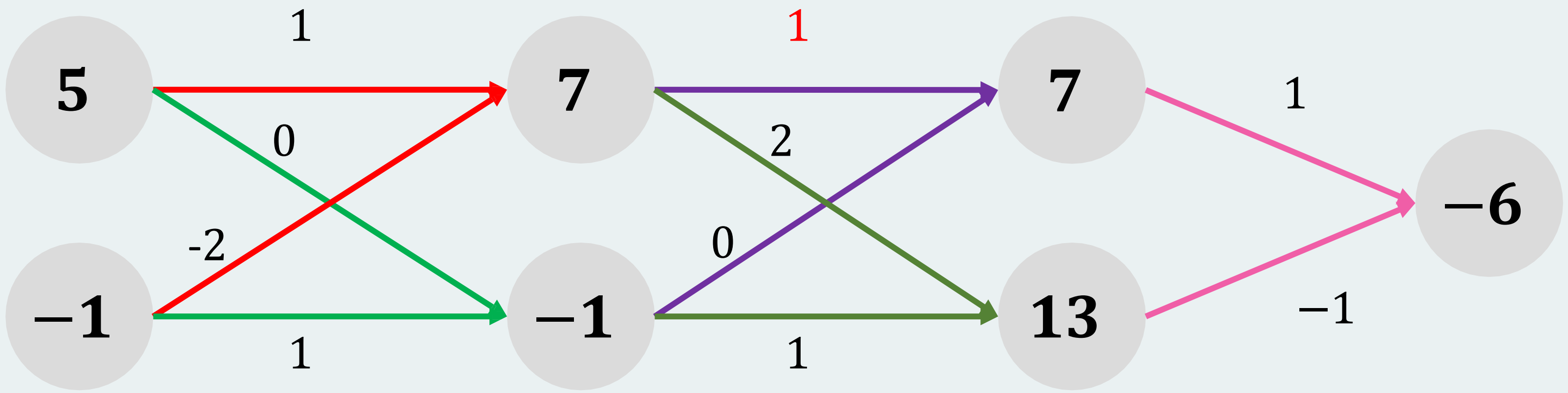


How to compute derivatives?

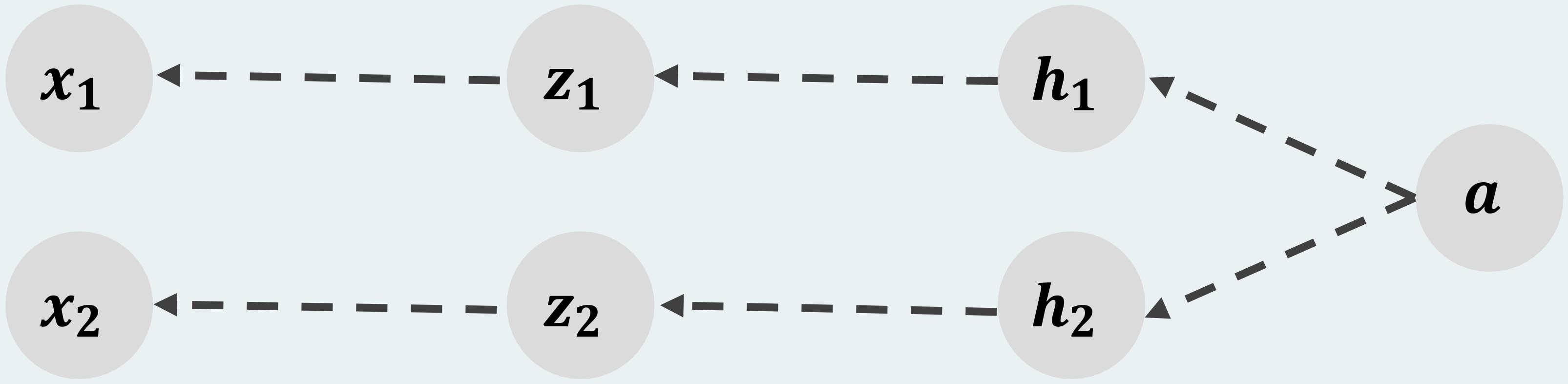
Will the output (-13) change if we change this weight from 0 to 1?



How to compute derivatives?



How to compute derivatives?



- We move in the opposite direction along the graph and calculate derivatives
- Backpropagation

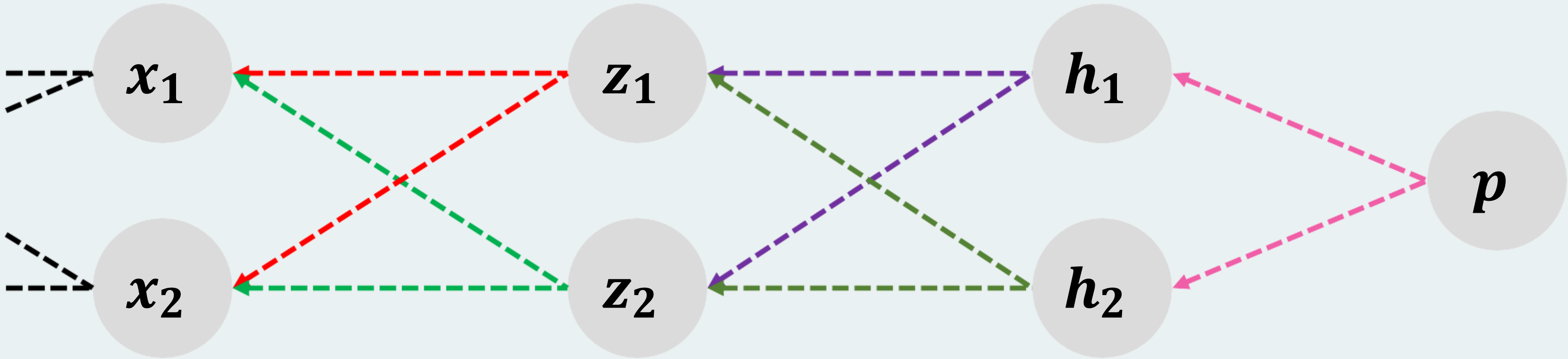
3: $\frac{\partial p}{\partial h_1}$ $\frac{\partial p}{\partial h_2}$

2: $\frac{\partial p}{\partial z_1} = \frac{\partial p}{\partial h_1}\frac{\partial h_1}{\partial z_1} + \frac{\partial p}{\partial h_2}\frac{\partial h_2}{\partial z_1}$ $\frac{\partial p}{\partial z_2} = \frac{\partial p}{\partial h_1}\frac{\partial h_1}{\partial z_2} + \frac{\partial p}{\partial h_2}\frac{\partial h_2}{\partial z_2}$

1:

$$\frac{\partial p}{\partial x_1} = \frac{\partial p}{\partial h_1}\frac{\partial h_1}{\partial z_1}\frac{\partial z_1}{\partial x_1} + \frac{\partial p}{\partial h_2}\frac{\partial h_2}{\partial z_1}\frac{\partial z_1}{\partial x_1} + \frac{\partial p}{\partial h_1}\frac{\partial h_1}{\partial z_2}\frac{\partial z_2}{\partial x_1} + \frac{\partial p}{\partial h_2}\frac{\partial h_2}{\partial z_2}\frac{\partial z_2}{\partial x_1}$$

$$\frac{\partial p}{\partial x_2} = \frac{\partial p}{\partial h_1}\frac{\partial h_1}{\partial z_1}\frac{\partial z_1}{\partial x_2} + \frac{\partial p}{\partial h_2}\frac{\partial h_2}{\partial z_1}\frac{\partial z_1}{\partial x_2} + \frac{\partial p}{\partial h_1}\frac{\partial h_1}{\partial z_2}\frac{\partial z_2}{\partial x_2} + \frac{\partial p}{\partial h_2}\frac{\partial h_2}{\partial z_2}\frac{\partial z_2}{\partial x_2}$$



Backpropagation

- Many formulas have the same derivatives
- In backpropagation, each partial derivative is computed only once—computing derivatives for layer N is reduced to multiplying the matrix of derivatives for layer N+1 by some vector

4. Neural Networks for NLP problems

Use cases for feedforward networks

Let's consider two sample tasks:

1. Text classification
2. Language modeling

State of the art systems use more powerful neural architectures, but simple models are still useful to consider!

Classification: Sentiment Analysis

- We could do exactly what we did with logistic regression
- Input layer are binary features as before
- Output layer is 0 or 1

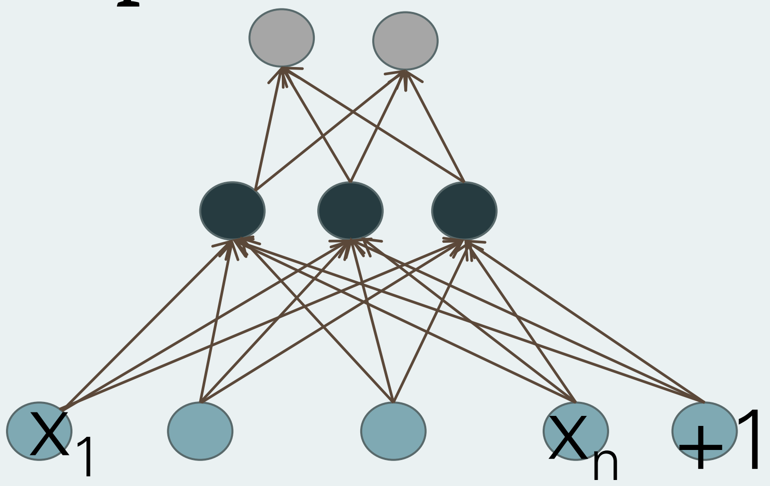
Sentiment Features

Var	Definition
x_1	count(positive lexicon) $\in$ doc)
x_2	count(negative lexicon) $\in$ doc)
x_3	$\begin{cases} 1 & \text{if ``no''} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_4	count(1st and 2nd pronouns $\in$ doc)
x_5	$\begin{cases} 1 & \text{if ``!''} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_6	log(word count of doc)

Feedforward nets for simple classification

Just adding a hidden layer to logistic regression :

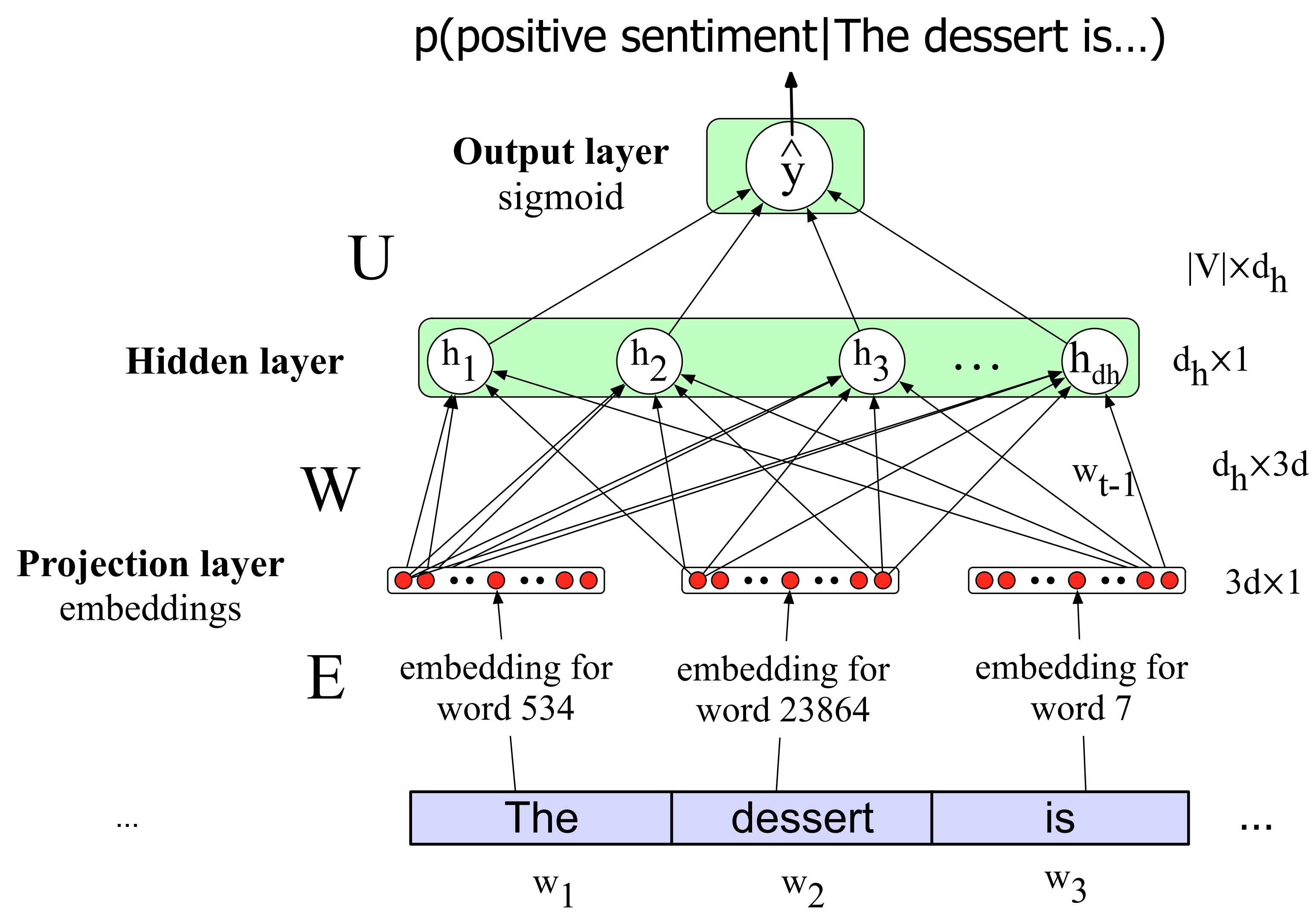
- allows the network to use non-linear interactions between features
- which may (or may not) improve performance
- Input: sentiment features
- Output: softmax layer



Even better: representation learning

- The real power of deep learning comes from the ability to **learn** features from the data
- Instead of using hand-built human-engineered features for classification
- Use learned representations like embeddings!
- Input: embeddings
- Output: softmax layer

Neural Net Classification with embeddings as input features!

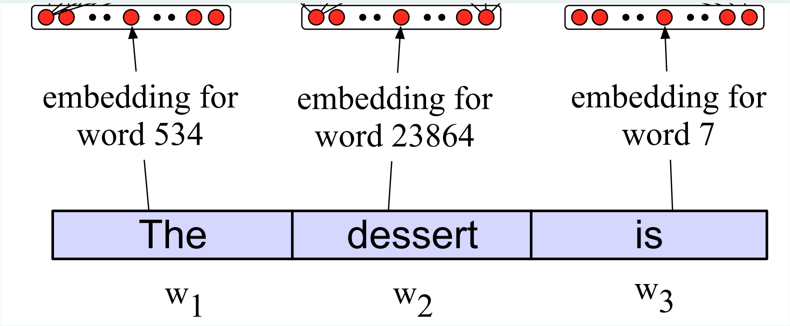


Issue: texts come in different sizes

This assumes a fixed size length (3)! Kind of unrealistic. Solutions:

Make the input the length of the longest review

- If shorter then **pad** with zero embeddings
- **Truncate** if you get longer reviews at test time



Reminder: Multiclass Outputs

- What if you have more than two output classes?
  - Add more output units (one for each class)
  - And use a "softmax layer"

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad 1 \leq i \leq D$$

Neural Language Models (LMs)

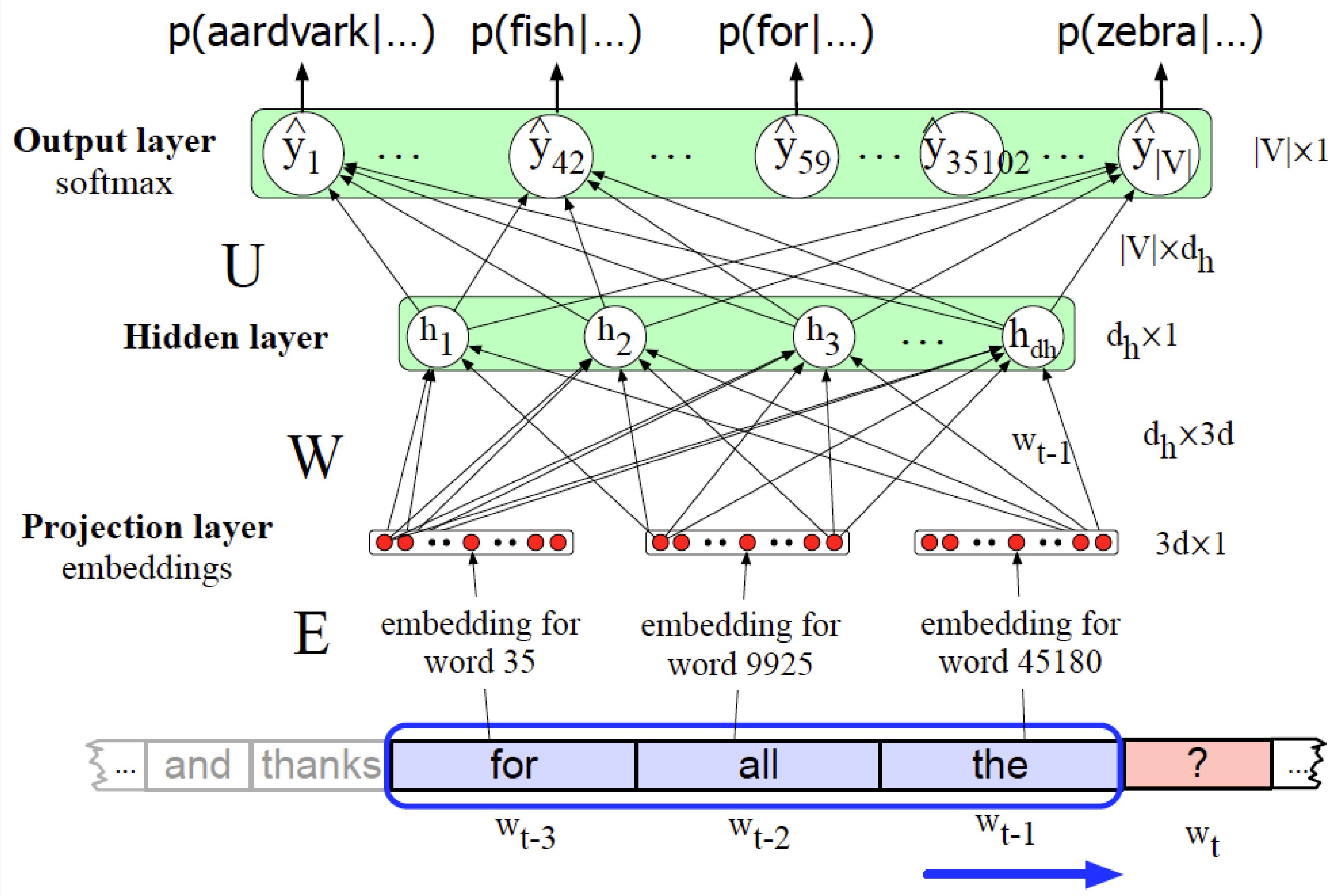
- **Language Modeling**: Calculating the probability of the next word in a sequence given some history.
- We've seen N-gram based LMs
- But neural network LMs far outperform n-gram language models
- State-of-the-art neural LMs are based on more powerful neural network technology like Transformers
- But **simple feedforward LMs** can do almost as well!

Simple feedforward Neural Language Models

- **Task**: predict next word w_t,given prior words w_{t-1}, w_{t-2}, w_{t-3}, …
- **Problem**: Now we're dealing with sequences of arbitrary length.
- **Solution**: Sliding windows (of fixed length)

$$P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-N+1}^{t-1})$$

Neural Language Model



Why Neural LMs work better than N-gram LMs

Training data:

- We've seen: I have to make sure that the cat gets fed.
- Never seen: dog gets fed

Test data:

- I forgot to make sure that the dog gets ___
- N-gram LM can't predict "fed"!
- Neural LM can use similarity of "cat" and "dog" embeddings to generalize and predict "fed" after dog

Advantages of neural networks

- Parallelism
- Generalization Abilities: Often remarkable, particularly in pattern recognition for images and text (but not limited to these areas!).
- Ease of Implementation: Simple to set up for a wide range of problems.
- Value in low-information contexts and low-resource languages: Generally effective for problems where little prior information is available.

Limitations

- **Training**: Neural networks require significant resources for training.
  Hugging Face model size estimator:
  https://huggingface.co/docs/accelerate/usage_guides/model_size_estimator
- "**Black Box**" Nature: No explanation for predictions (Shapley values, LIME)
- **Network architecture selection**: Choosing the right architecture and hyperparameters is challenging
- **Complex Optimization Problem**: The search space is large, and finding the global optimum is difficult