

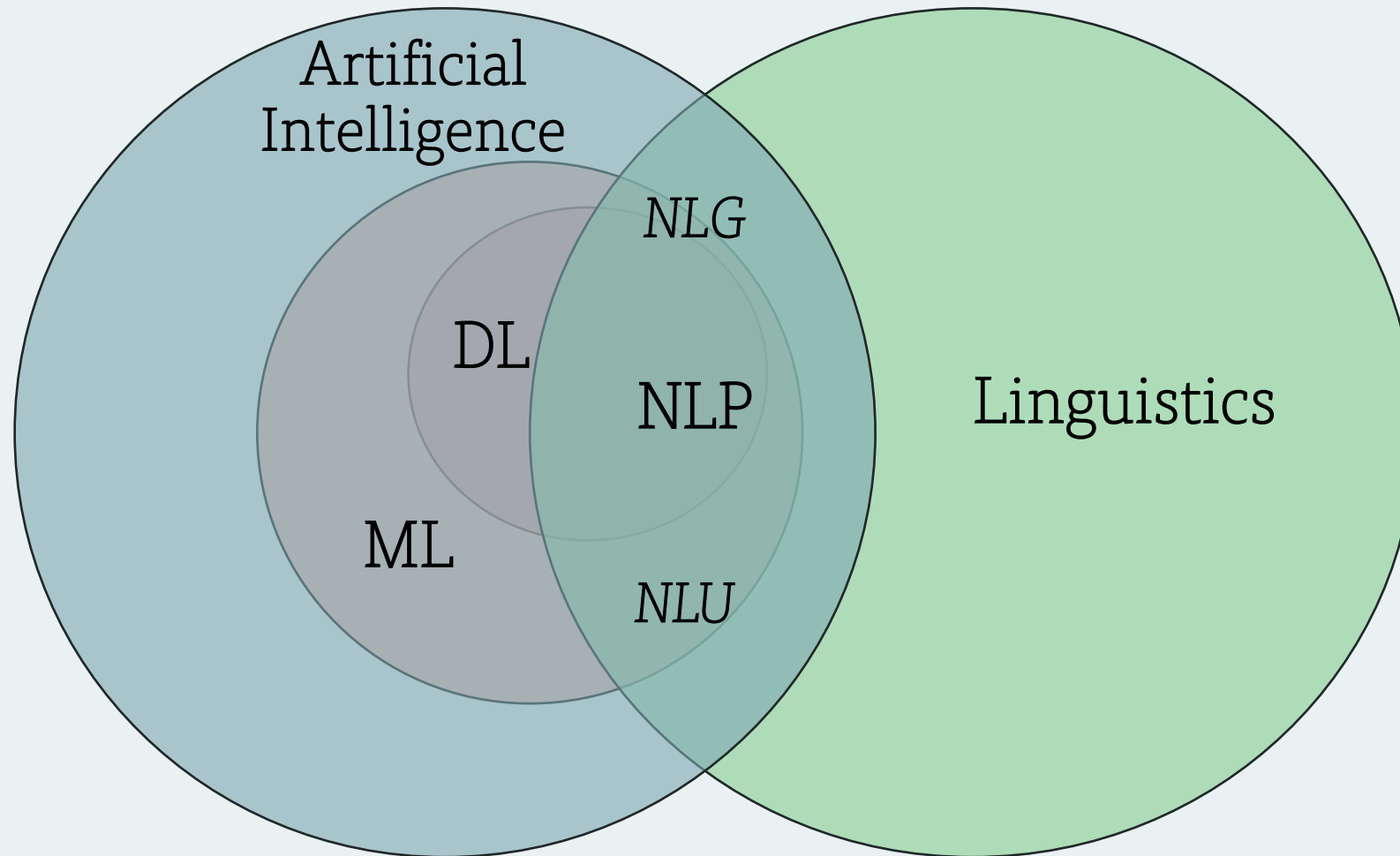
NLP for Social Sciences

5. Neural Networks and Language Modelling

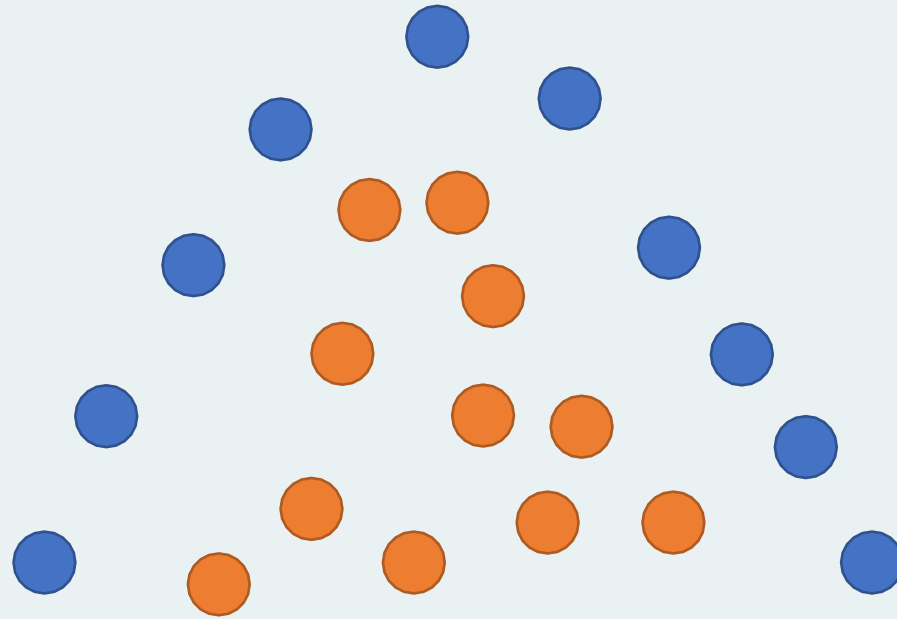
Irina Proskurina, Université de Lyon, de Lyon 2, Laboratoire ERIC

1. Neural Networks

Natural Language Processing

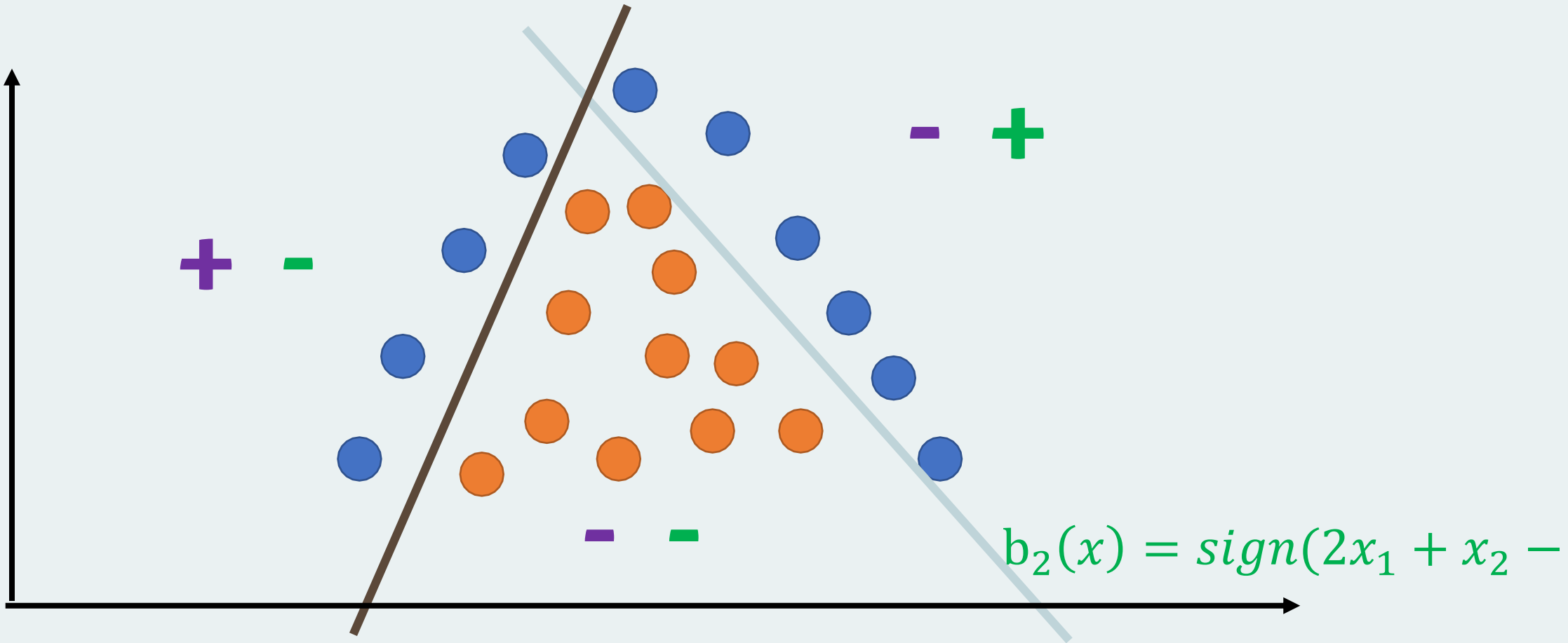


Neural Network Structure



Nonlinear patterns

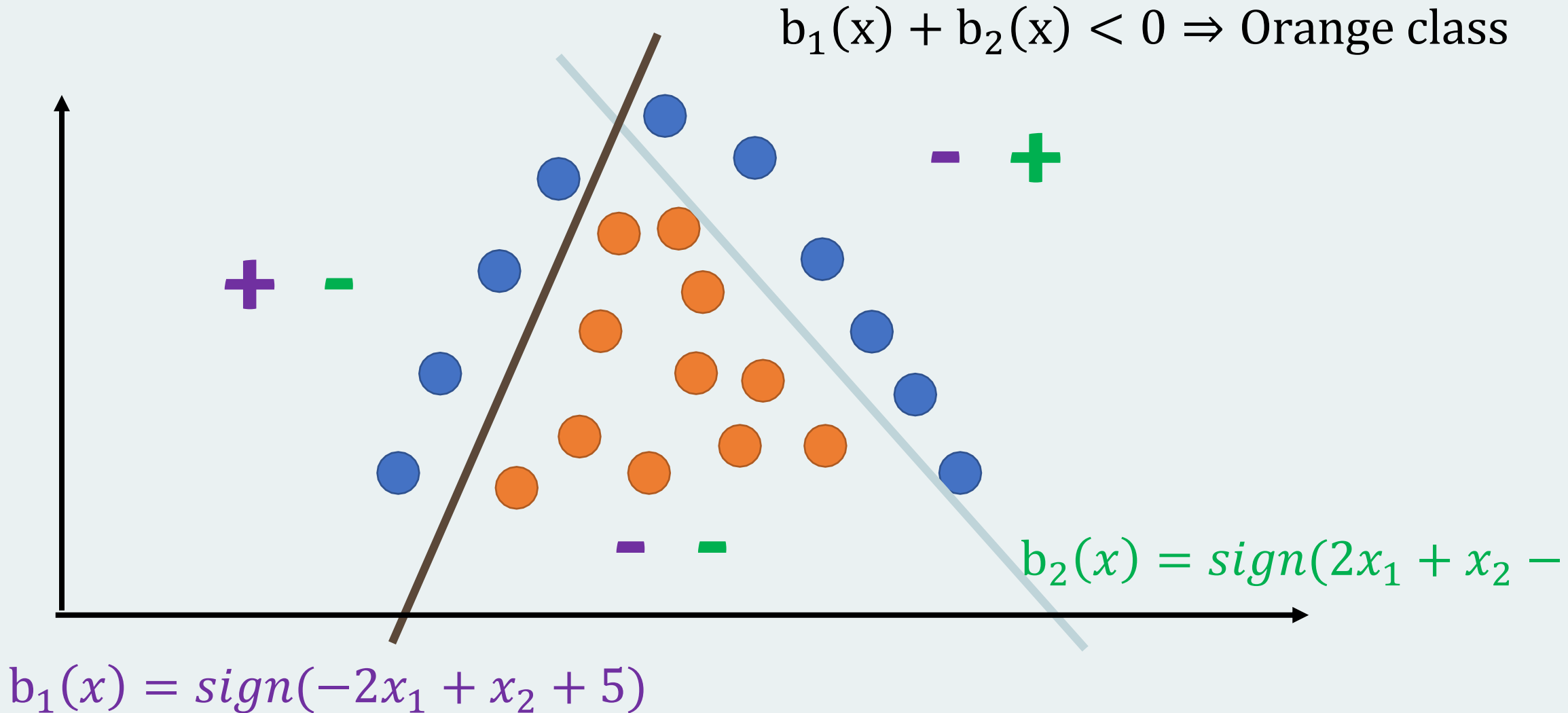
Neural Network Structure



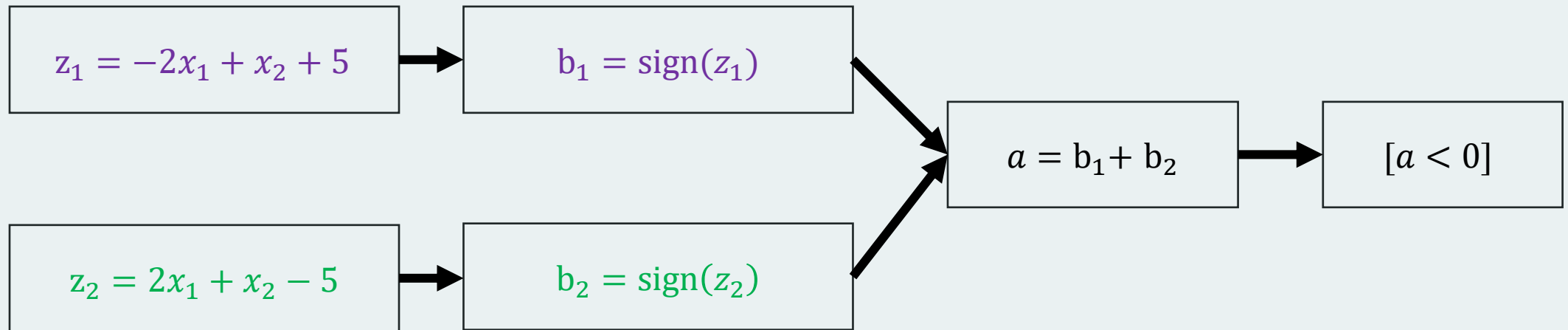
$$b_1(x) = \text{sign}(-2x_1 + x_2 + 5)$$

$$b_2(x) = \text{sign}(2x_1 + x_2 - 5)$$

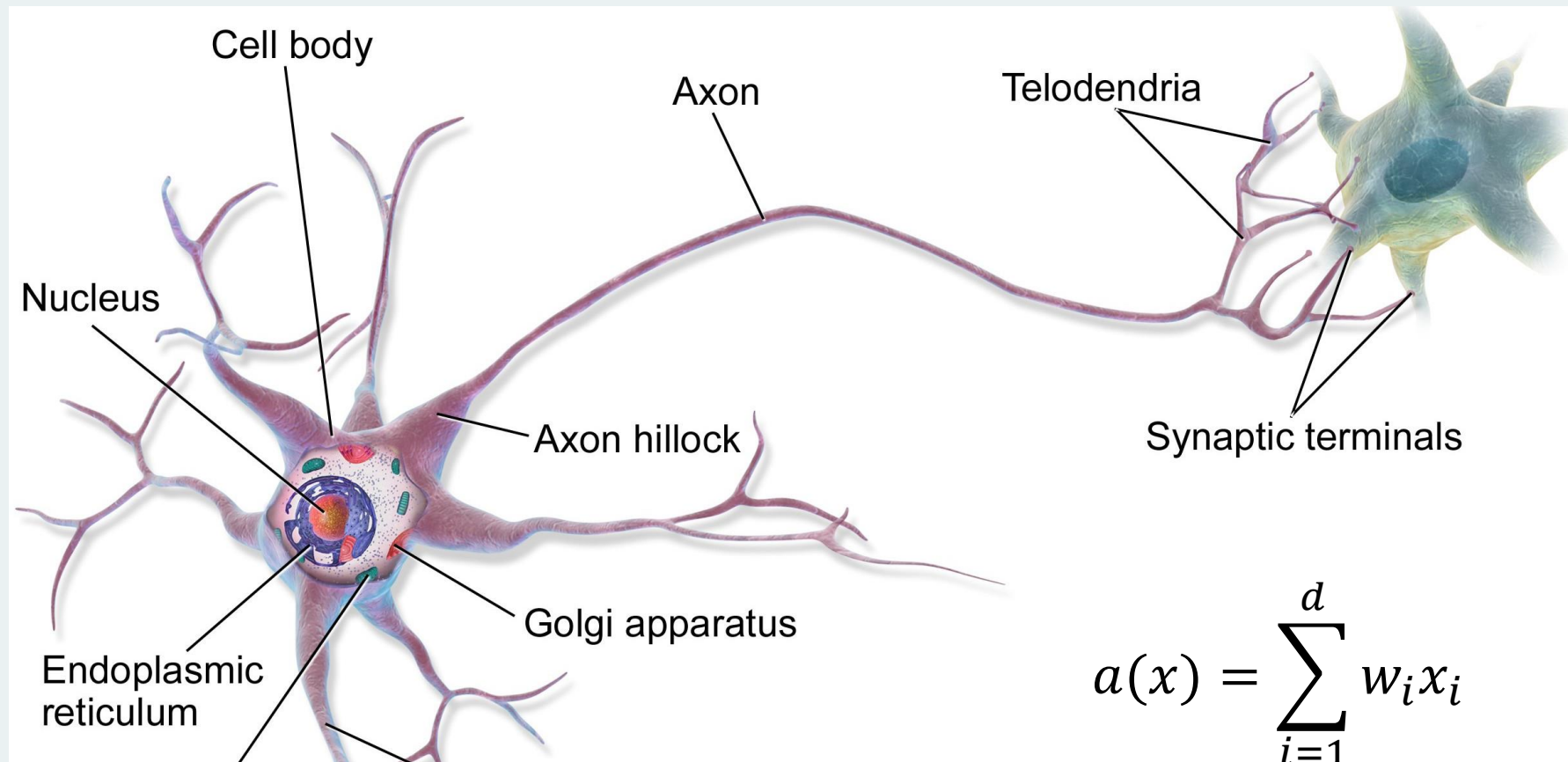
Neural Network Structure



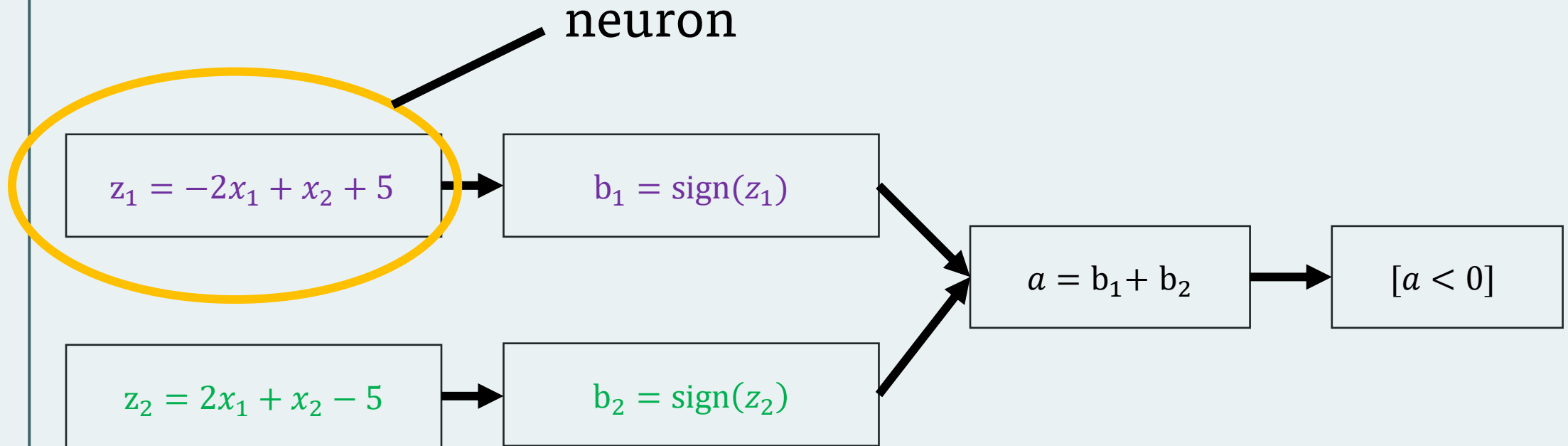
Neural Network Structure



Neuron

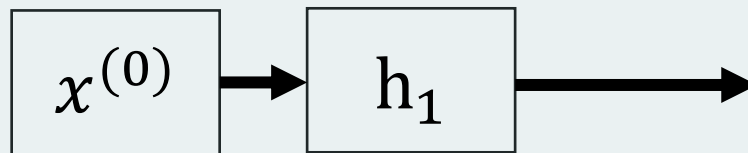


Neural Network Structure

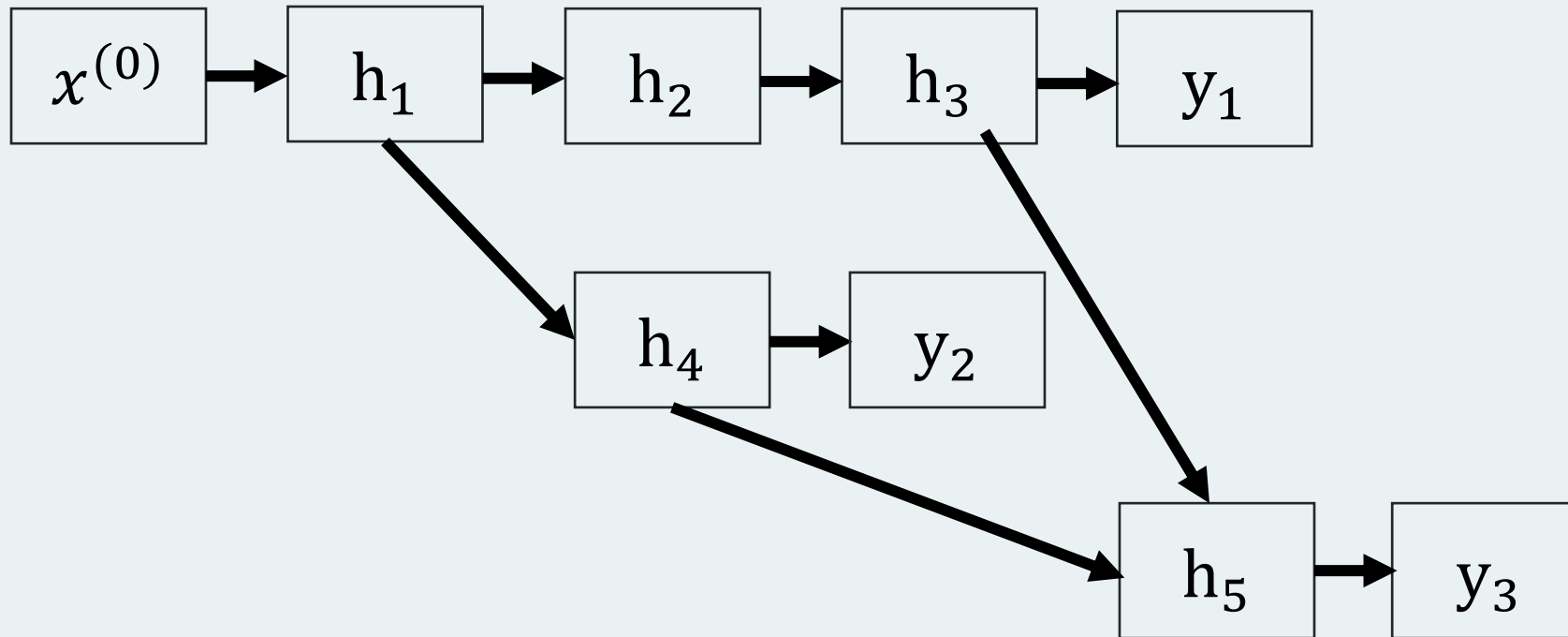


Computational graph (neural network)

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



Computational graph (neural network)



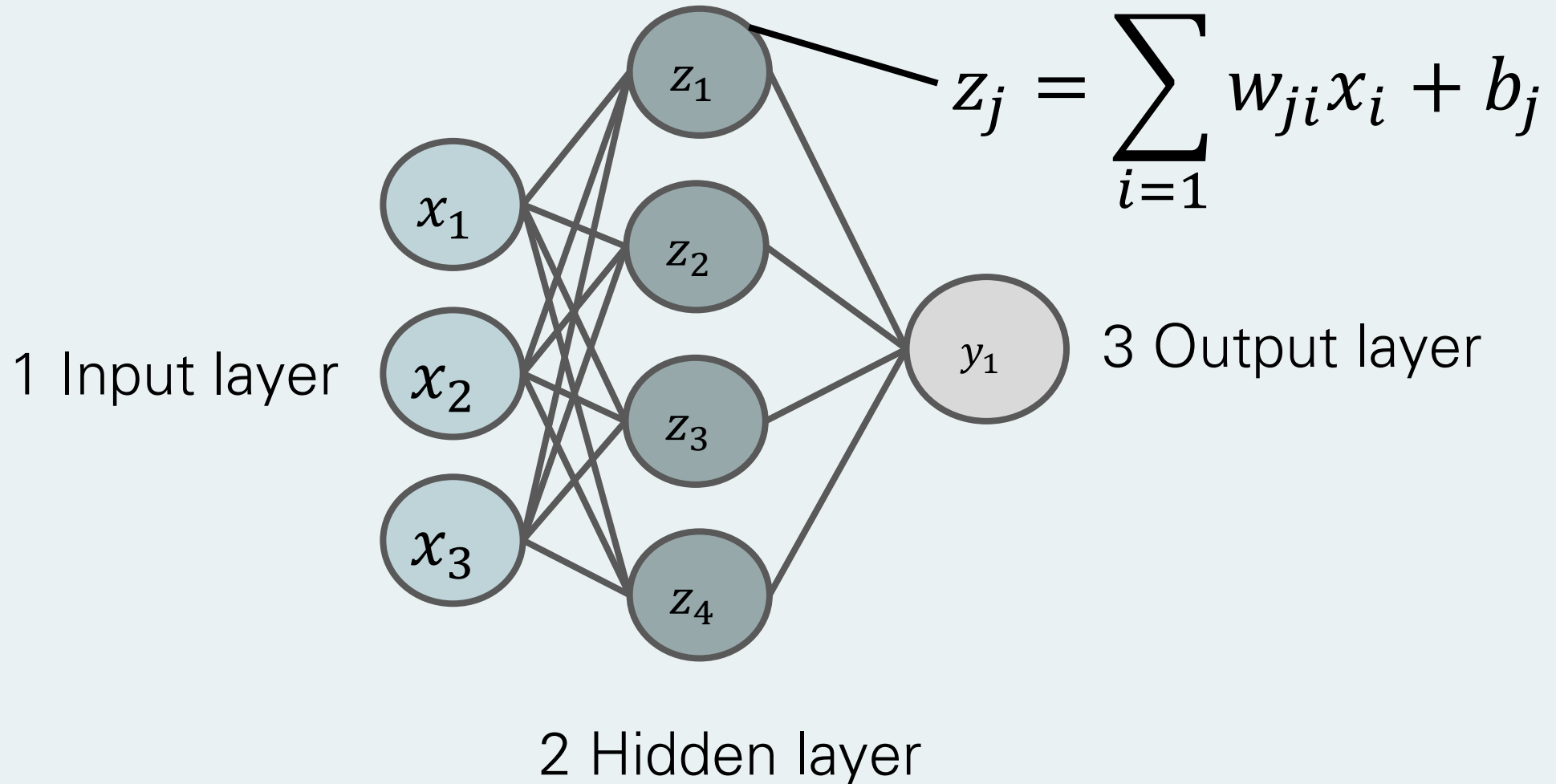
2. Fully connected layers

Fully connected layers

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



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

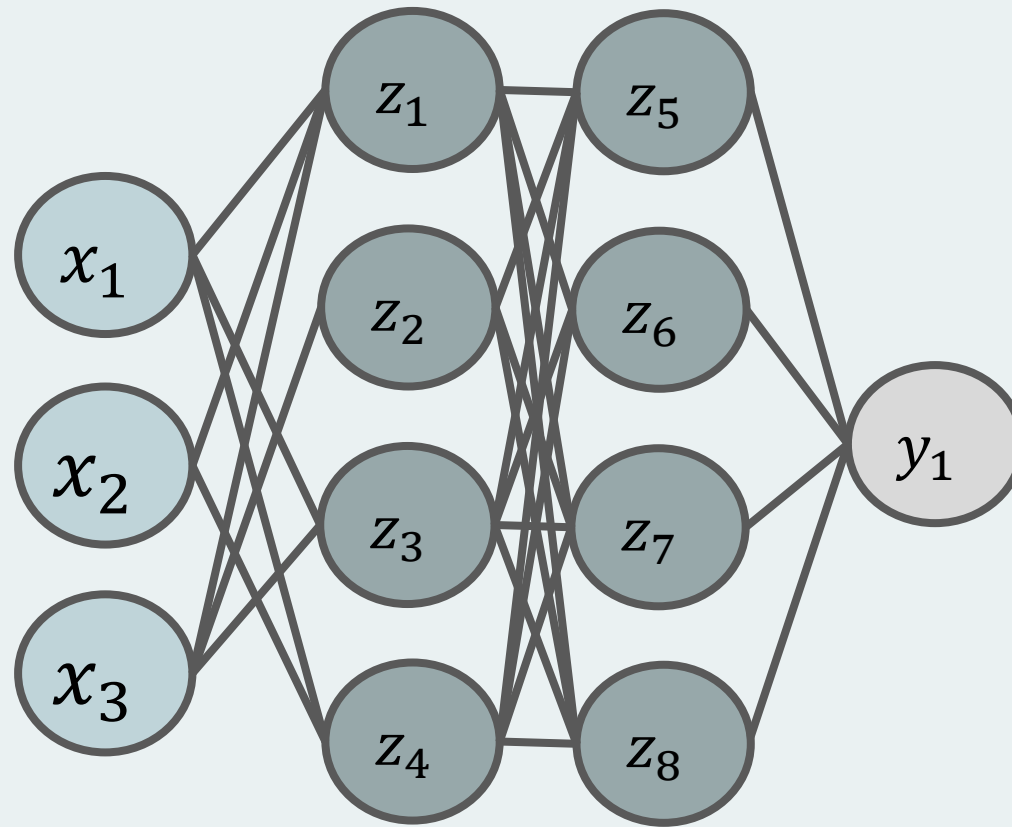
$$z_j = \sum_{i=1}^n w_{ji}x_i + b_j$$

- 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
- 16 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



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

Non-linearity

- Given 2 fully-connected layers

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

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

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

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

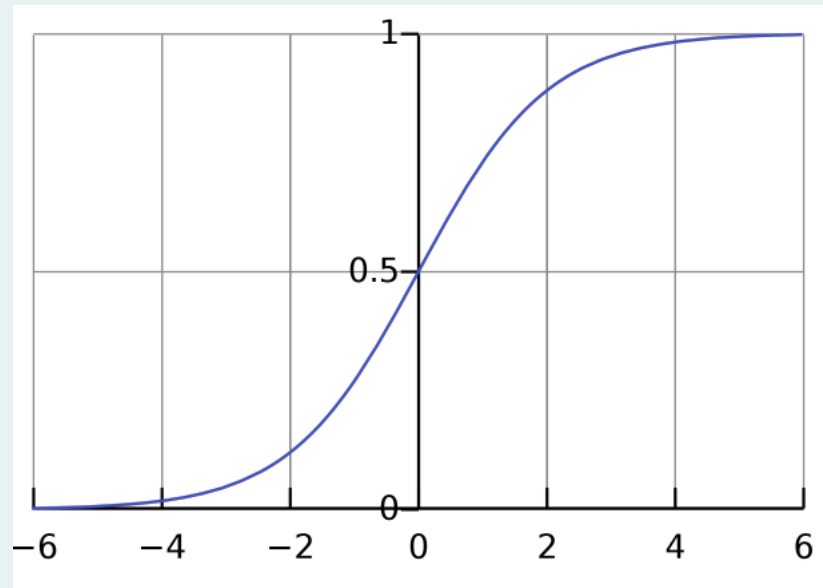
Activation functions

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

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

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

Logistic / Sigmoid



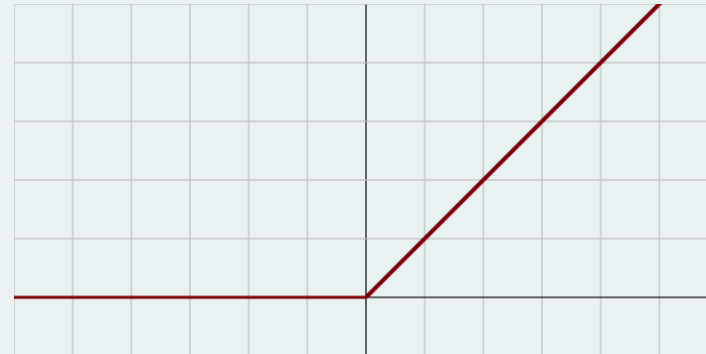
Activation functions

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


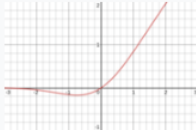

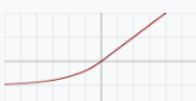
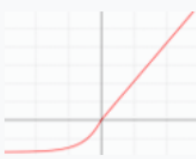
$$z_j = f\left(\sum_{i=1}^n w_{ji}x_i + b_j\right)$$

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

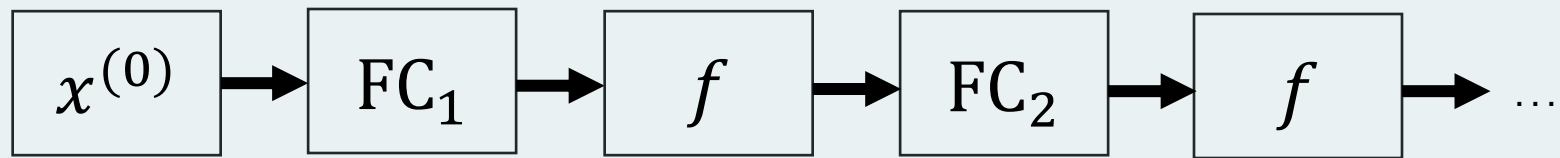
(ReLU, REctified Linear Unit)



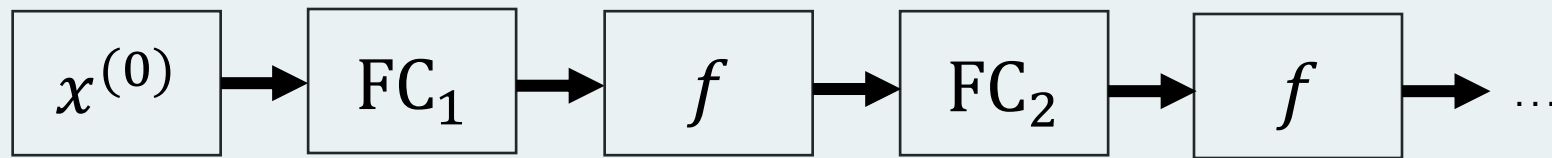
Activation Functions

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]		$\frac{1}{2}x \left(1 + \operatorname{erf} \left(\frac{x}{\sqrt{2}} \right) \right)$ $= x\Phi(x)$
Softplus ^[9]		$\ln(1 + e^x)$
Exponential linear unit (ELU) ^[10]		$\begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ <p>with parameter α</p>
Scaled exponential linear unit (SELU) ^[11]		$\lambda \begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ <p>with parameters $\lambda = 1.0507$ and $\alpha = 1.67326$</p>

A fully connected neural network

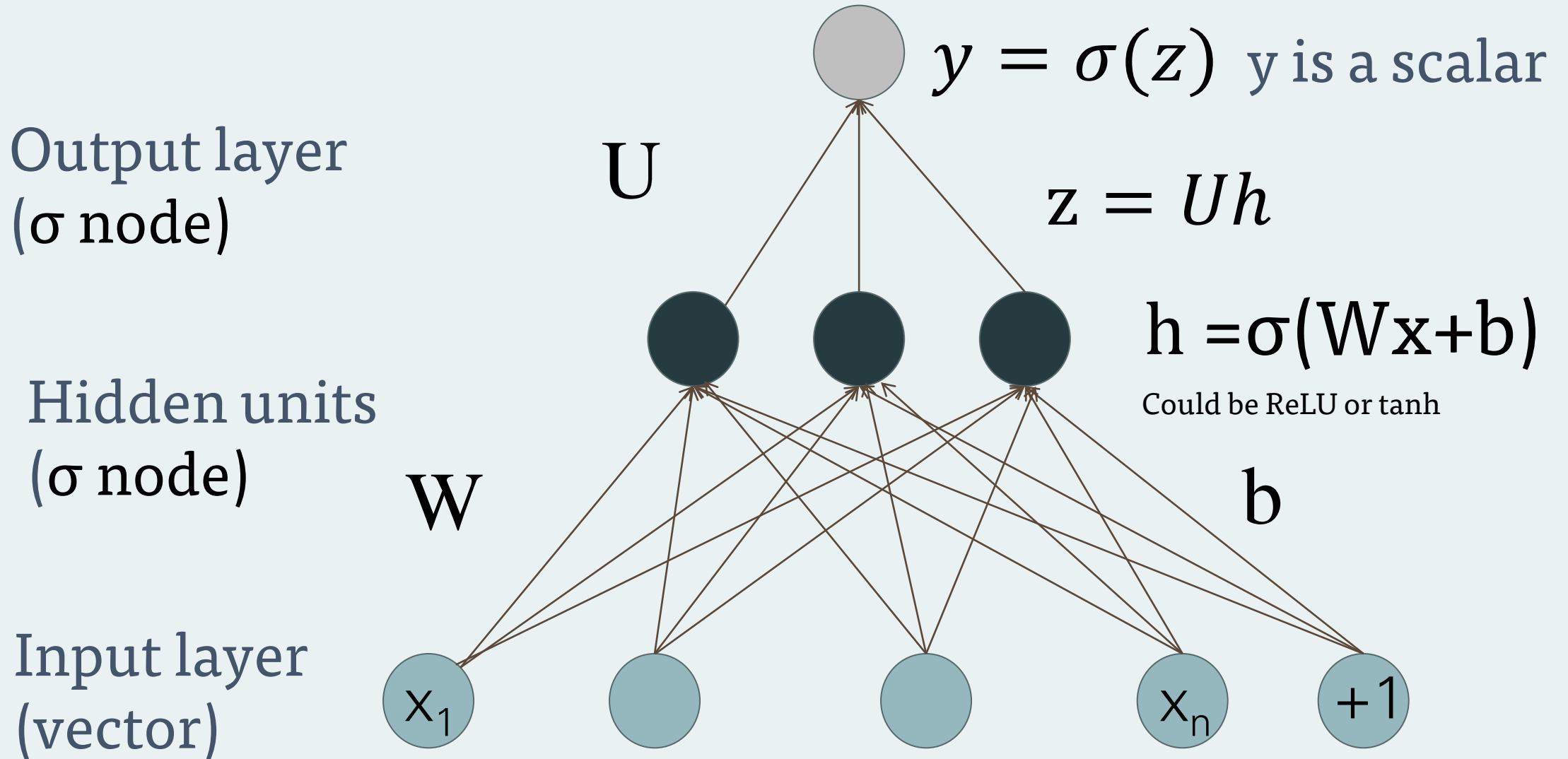


A fully connected neural network

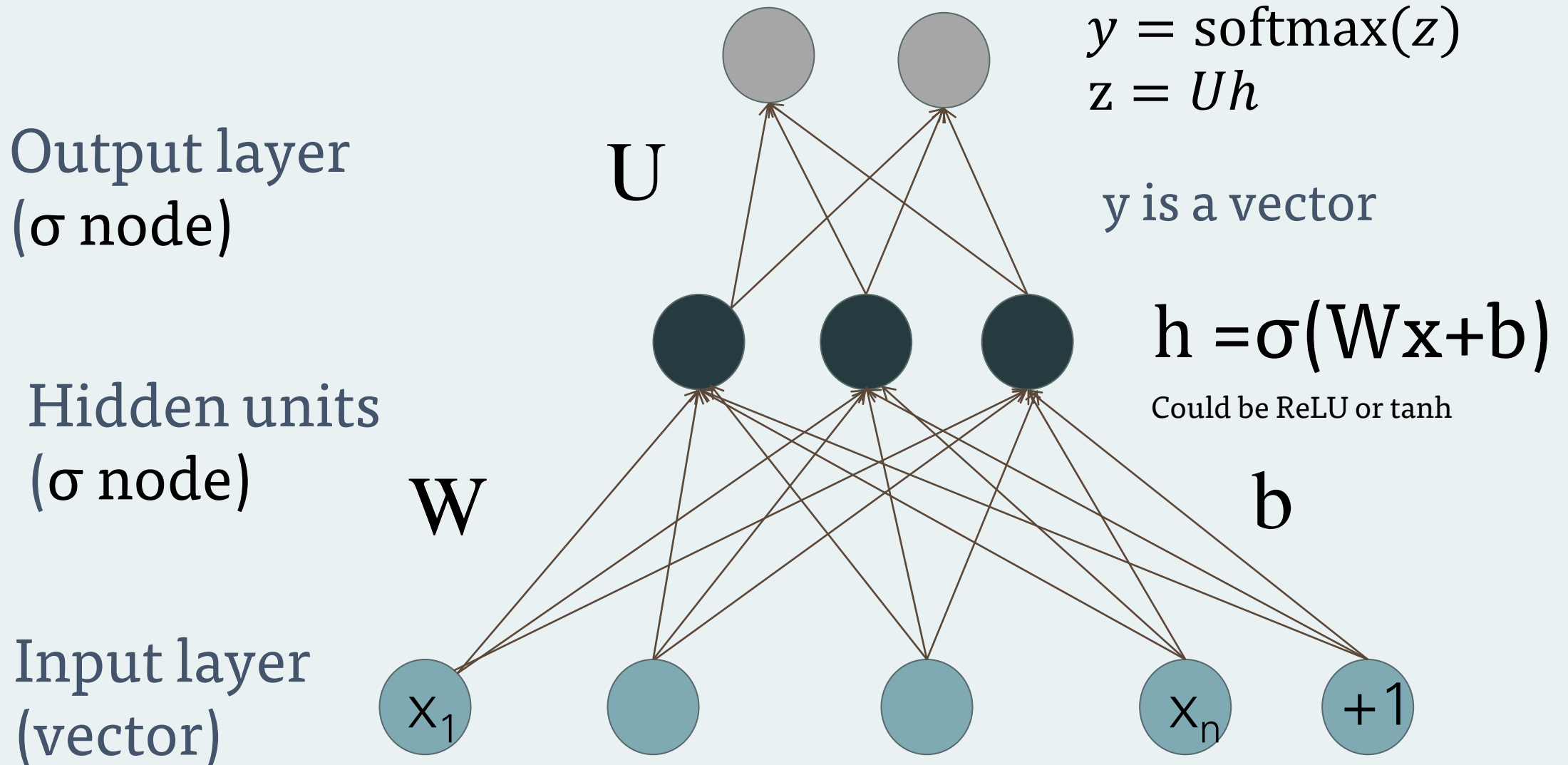


- Features are fed into the network
- The dimension of the last layer = # target variables (# classes)

Two-layer network with **scalar** output



Two-layer network with **softmax** output



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

Gradient Descent

- Repeat until convergence

$$w^t = w^{t-1} - \eta \nabla Q(w^t)$$

Updated
parameter



Learning Rate

Gradient at
the previous
point

Convergence

- Stopping rule 1

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

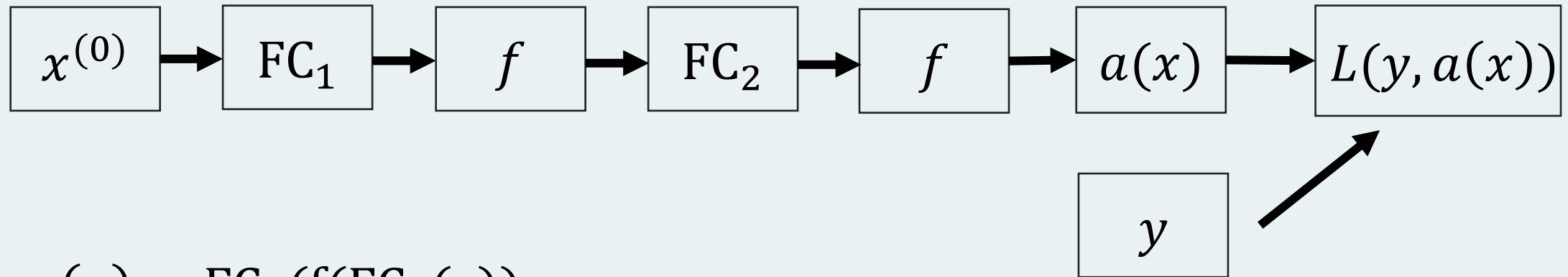
- Stopping rule 2

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

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

Training neural networks

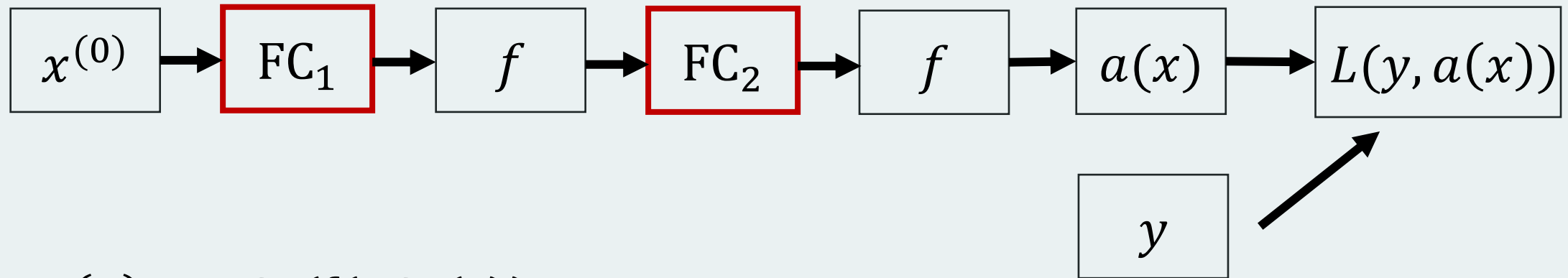
- All layers are usually possible to differentiate, so it's possible to compute derivatives with respect to all parameters



- $a(x) = FC_2(f(FC_1(x)))$
- 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



- $a(x) = FC_2(f(FC_1(x)))$
- Where are the **parameters**?

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

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:

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

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

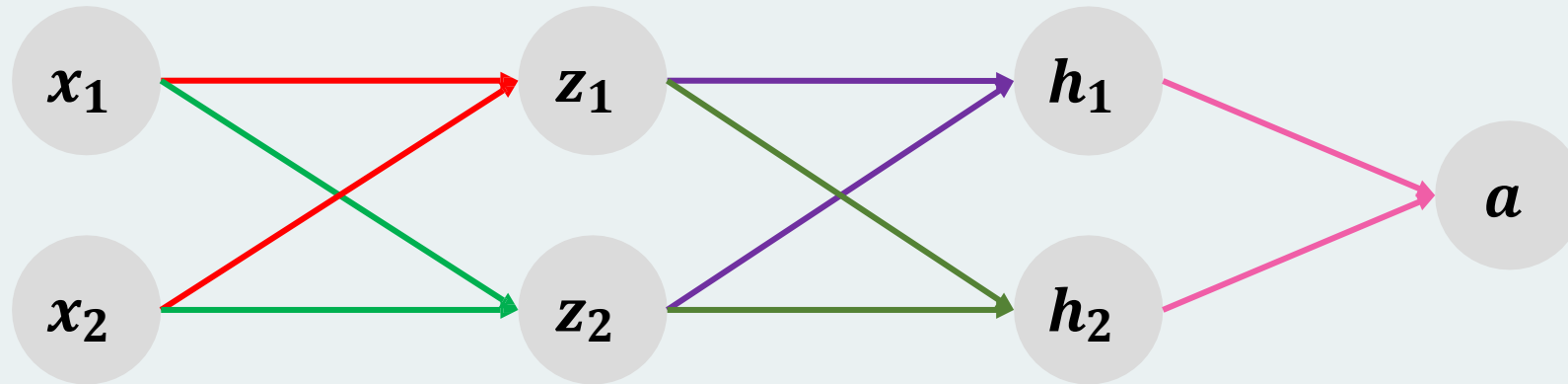
Computing derivatives

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

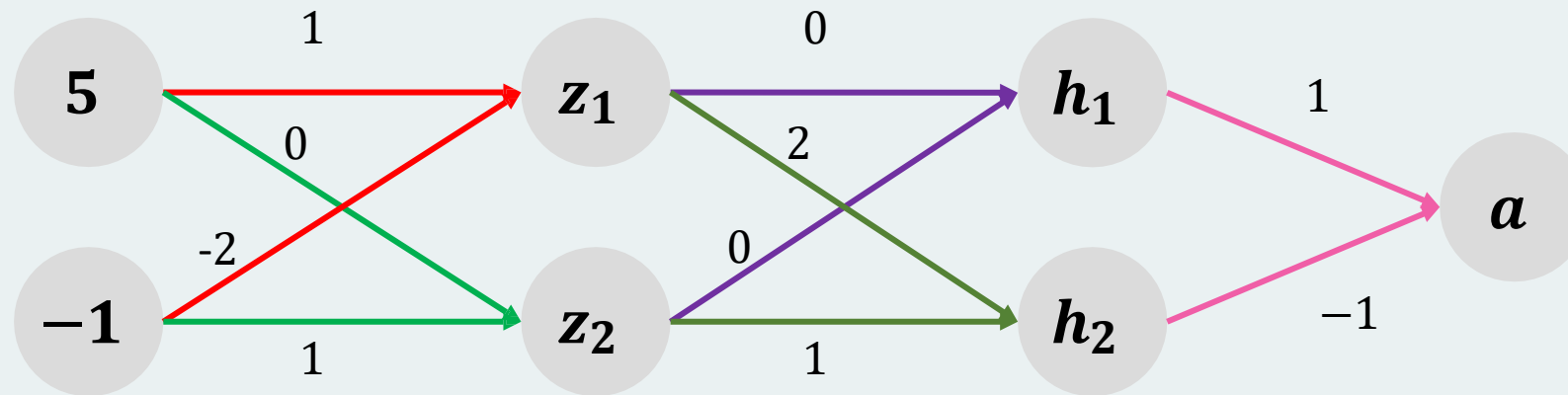
$$\frac{\partial}{\partial w_j} (a(x_i, w) - y_i)^2 = 2(a(x_i, w) - y_i) \frac{\partial}{\partial w_j} a(x_i, w)$$

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

How to compute derivatives?

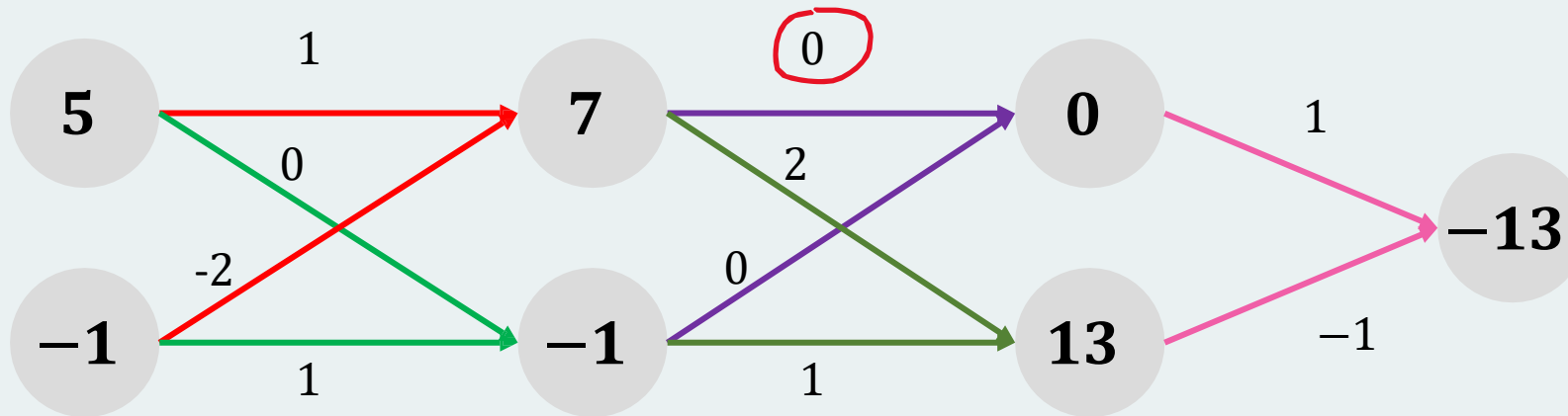


How to compute derivatives?

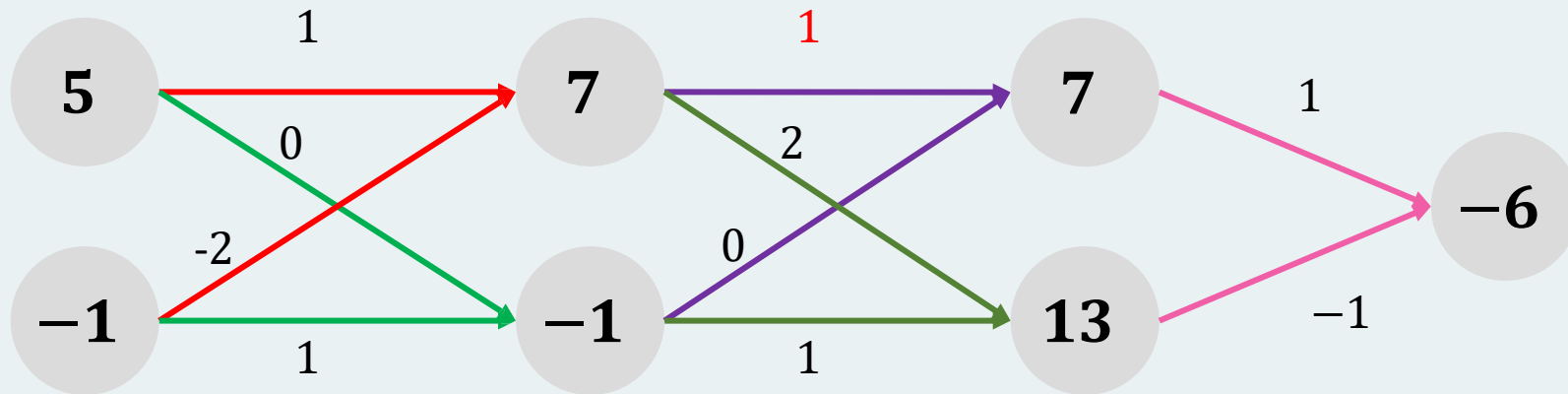


How to compute derivatives?

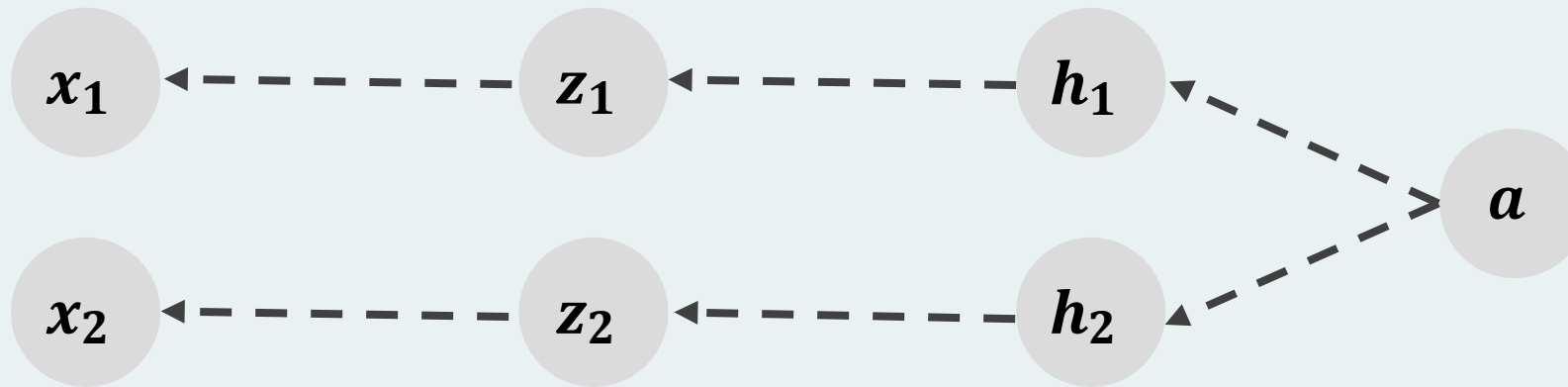
Will the output (-13) change if we change this weight from 0 to 1?



How to compute derivatives?



How to compute derivatives?



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

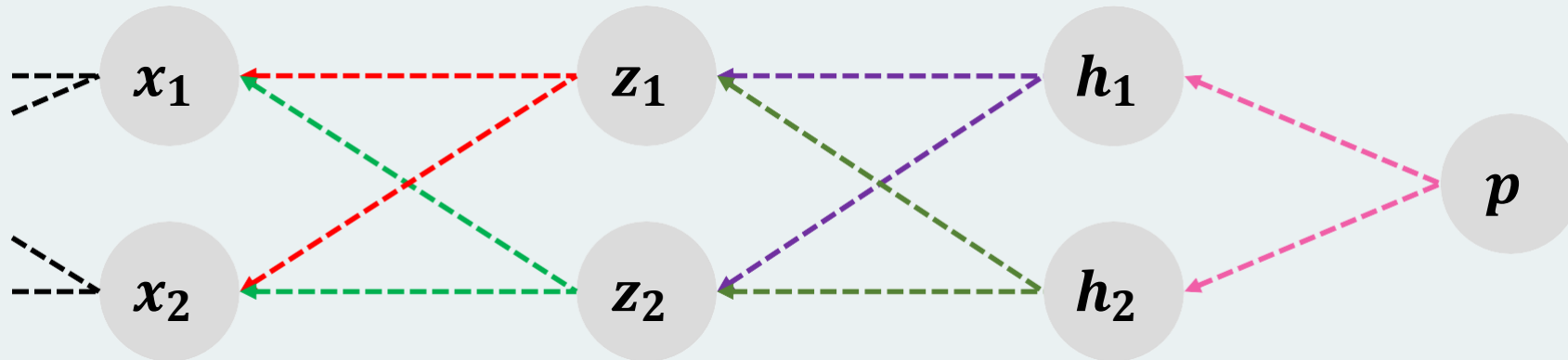
3: $\frac{\partial p}{\partial h_1} \quad \frac{\partial p}{\partial h_2}$

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

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

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

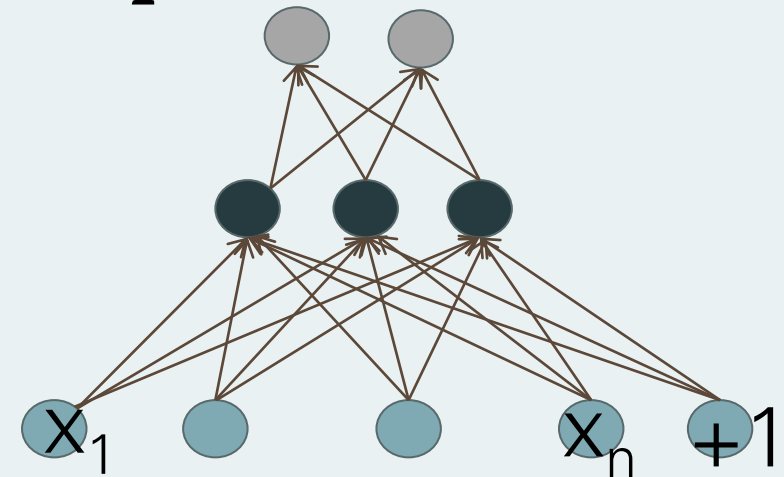
Sentiment Features

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 :

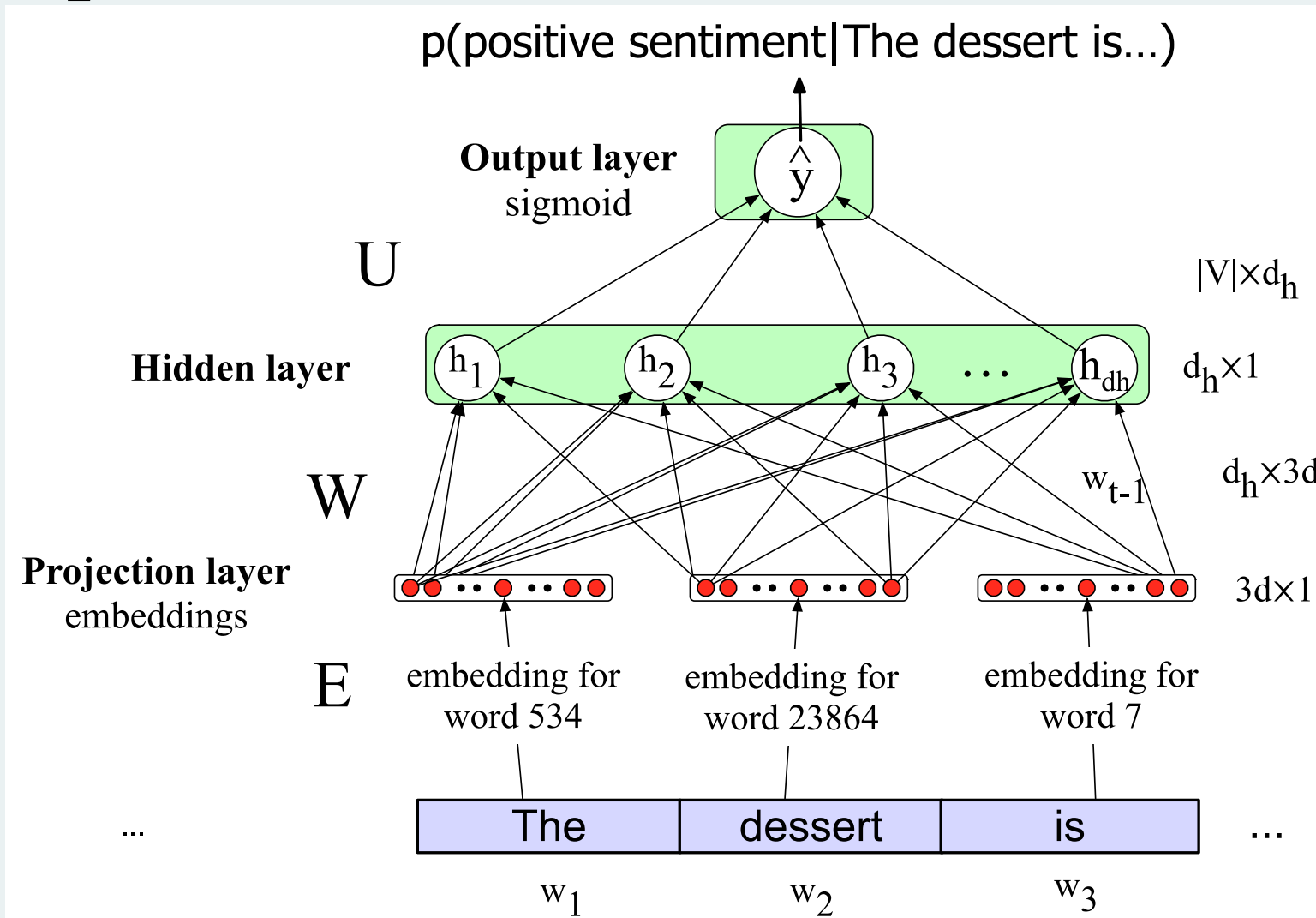
- allows the network to use non-linear interactions between features
- which may (or may not) improve performance
- Input: sentiment features
- Output: softmax layer



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!

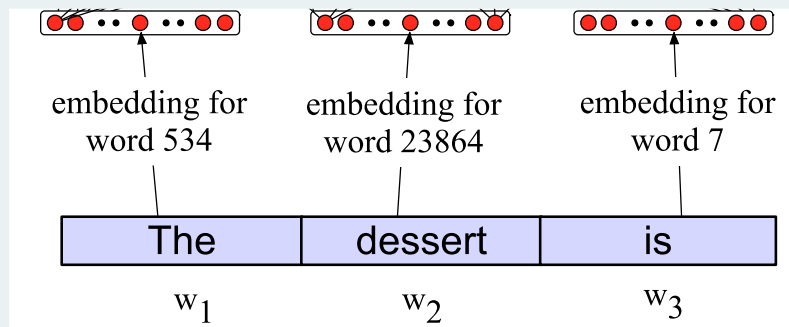


Issue: texts come in different sizes

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)

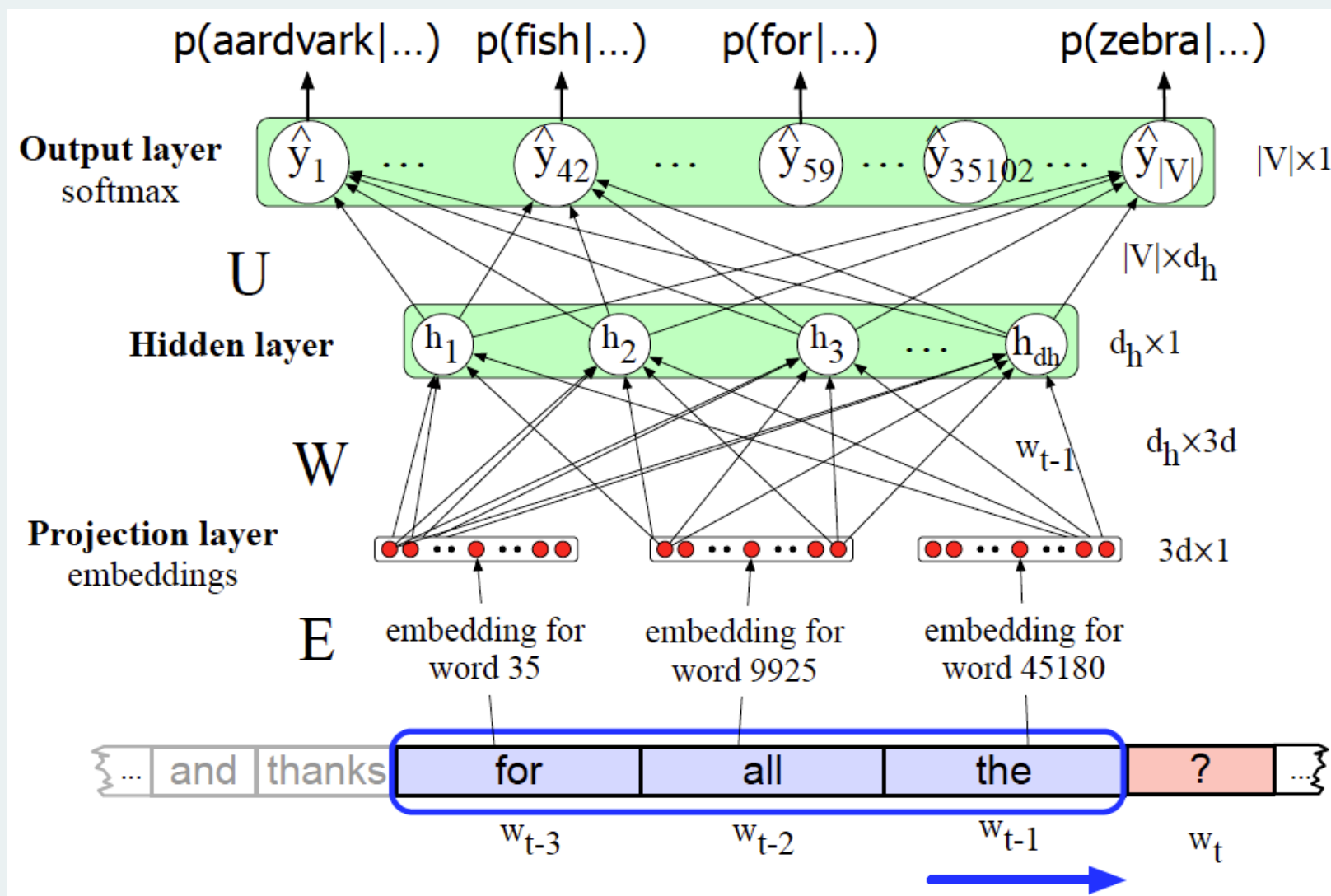
- **Language Modeling:** Calculating the probability of the next word in a sequence given some history.
- We've seen N-gram based LMs
- But neural network LMs far outperform n-gram language models
- State-of-the-art neural LMs are based on more powerful neural network technology like Transformers
- But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

Simple feedforward Neural Language Models

- **Task:** predict next word w_t , given prior words $w_{t-1}, w_{t-2}, w_{t-3}, \dots$
- **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

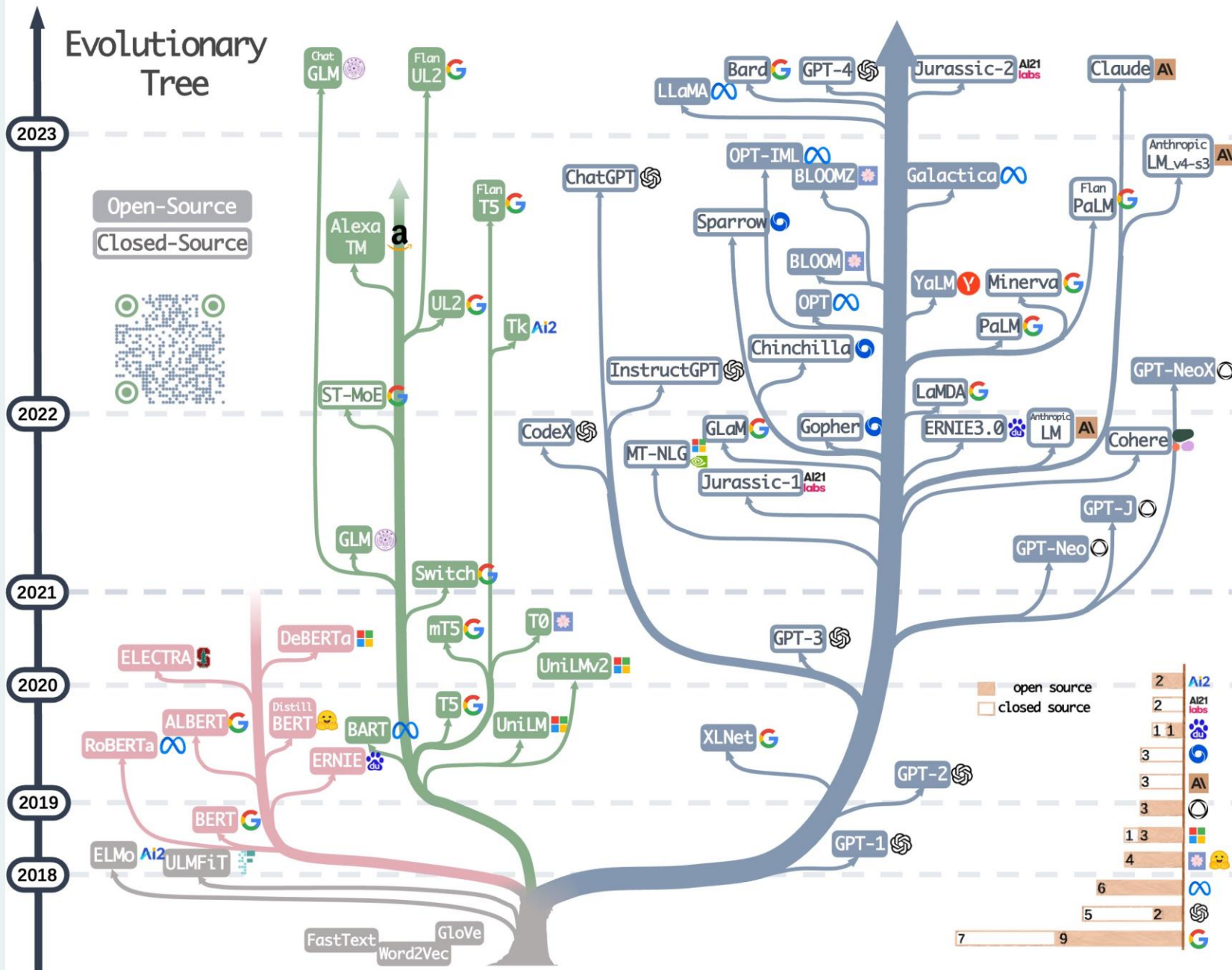
Advantages of neural networks

- **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

5. Transformer Language Models



> 1000 language models

Definition of language model

A language model is a probability distribution over sequences of token variables

$$W = (w_1, w_2 \dots w_n): P(W) = P(w_1, w_2 \dots w_n)$$

Given a word w_i , find the word, for which the conditional probability is the maximum:

$$P(w_{n+1}|w_1, w_2 \dots w_n)$$

$$P_\theta(W) = P_\theta(W_1, W_2, \dots W_n)$$

$$= P_\theta(W_1)P_\theta(W_2|W_1)P_\theta(W_3|W_1, W_2) \dots P_\theta(W_n|W_1, W_2, \dots, W_{n-1})$$

$$= \prod_{i=1}^n P_\theta(W_i|W_1, W_2, \dots, W_{i-1})$$

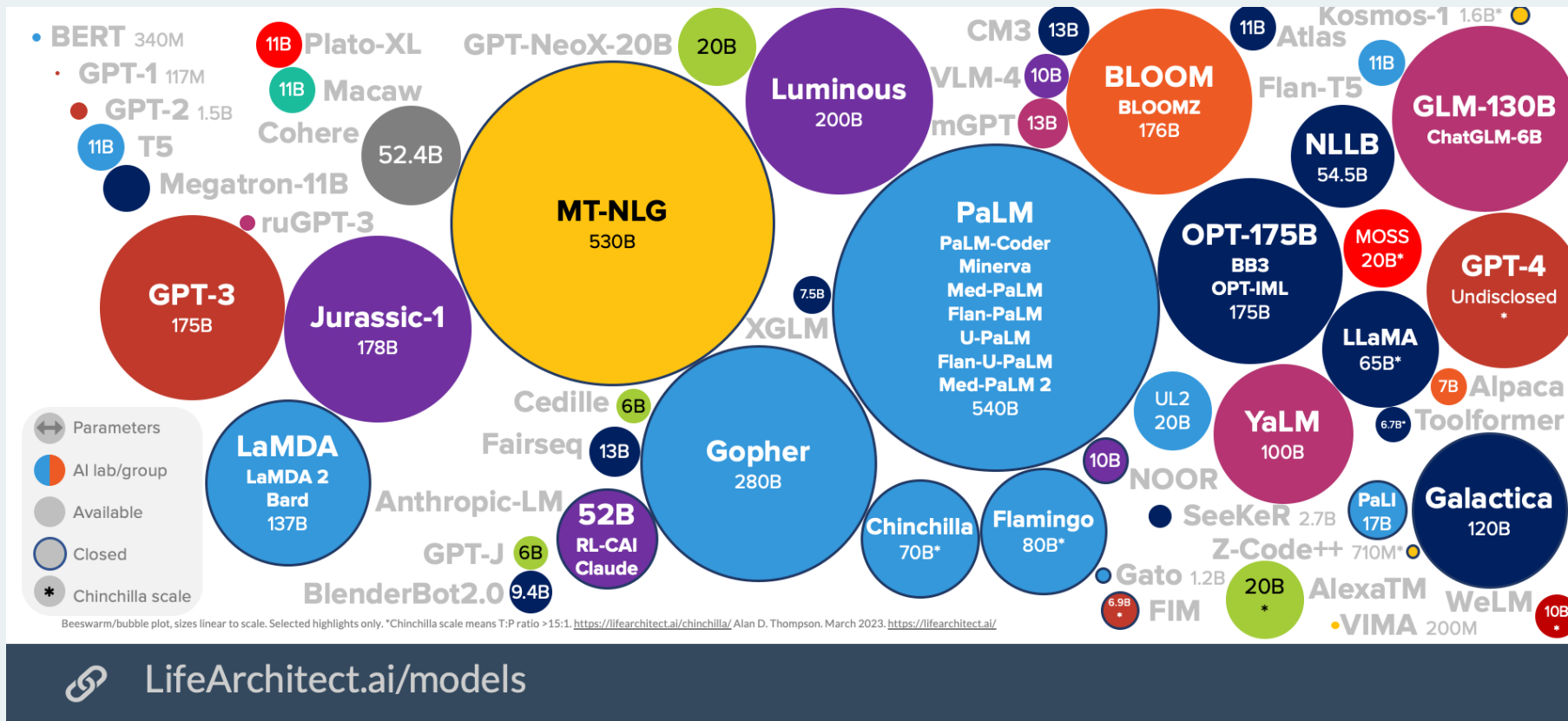
$$\theta^* = \operatorname{argmax}_{\theta} \sum_{i=1}^n \log P_\theta(w_i)$$

$$= \operatorname{argmax}_{\theta} \sum_{i=1}^n \sum_{j=1}^{m_i} \log P_\theta(w_{i,j}|w_{i,1}, w_{i,2}, \dots, w_{i,j-1})$$

m_i – the number of tokens in the sequence w_i ; $w_{i,j}$ is the j -th token in the sequence w_i

Large language models

- Large language models have $> 1\text{B}$ parameters θ



Decoder-only:

- PaLM
- MT-NLG
- GPT-3

LLMs are built out of transformers

- Transformer: a specific kind of network architecture, like a fancier feedforward network, but based on attention
-

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaizer@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

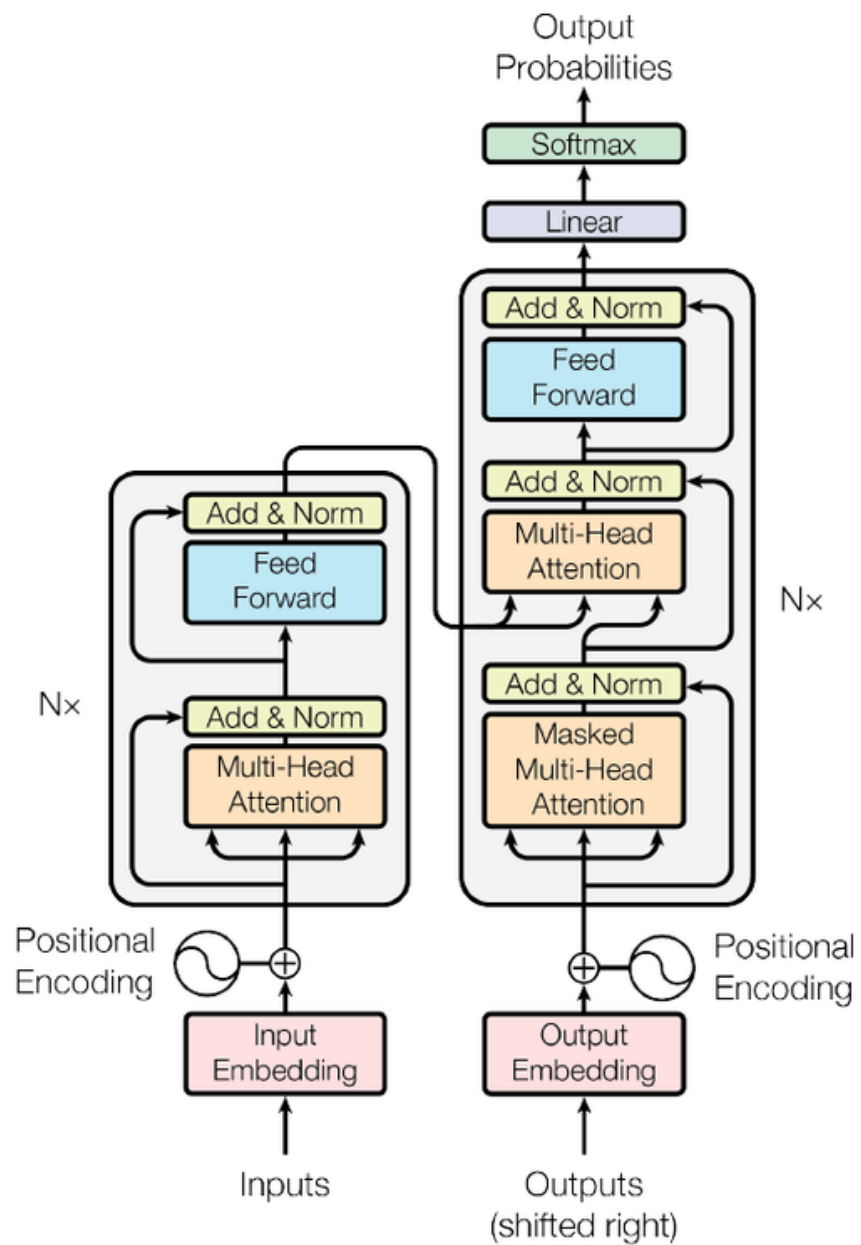
A very approximate timeline

- 1990 Static Word Embeddings
- 2003 Neural Language Model
- 2008 Multi-Task Learning
- 2015 Attention
- 2017 Transformer
- 2018 Contextual Word Embeddings and Pretraining
- 2019 Prompting

Generative AI and LLMs

- Large language models are models pre-trained on large corpus, they are used for many generation tasks.
- These models are generally fine-tuned to be adapted to specific needs.
- Recent results show that fine-tuning can be bypassed through appropriate prompts, paving the way for in-context learning or prompting.

Transformer LM



6. Attention layer

Problem with static embeddings (word2vec)

- They are static! The embedding for a word doesn't reflect how its meaning changes in context.
- The **chicken** didn't cross the road because **it** was too **tired**
- What is the meaning represented in the static embedding for "it"?

Contextual Embeddings

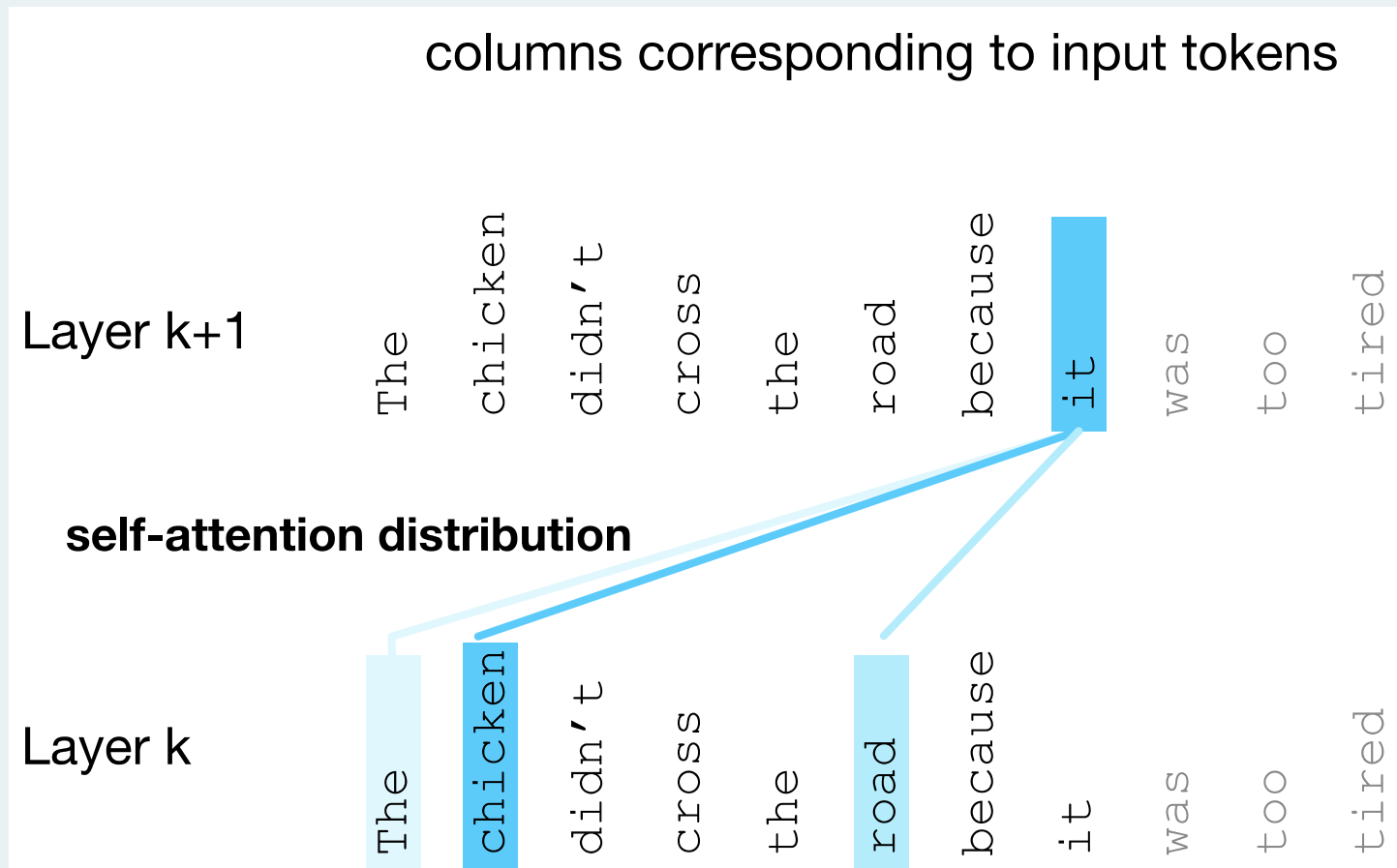
Intuition: a representation of meaning of a word should be different in different contexts!

Contextual Embedding: each word has a different vector that expresses different meanings depending on the surrounding words

How to compute contextual embeddings?

- **Attention**

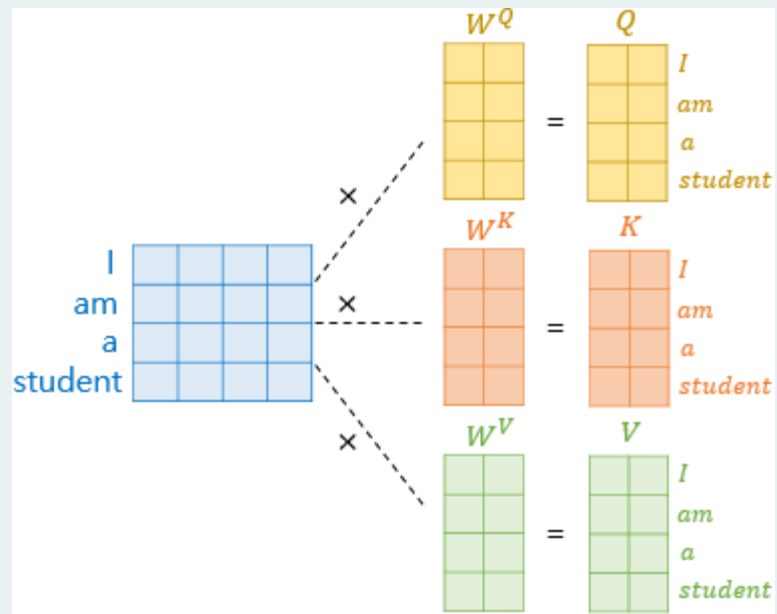
Intuition of attention:



Attention definition

- A mechanism for helping compute the embedding for a token by selectively attending to and integrating information from surrounding tokens
- More formally: a method for doing a weighted sum of vectors.

Attention is left-to-right



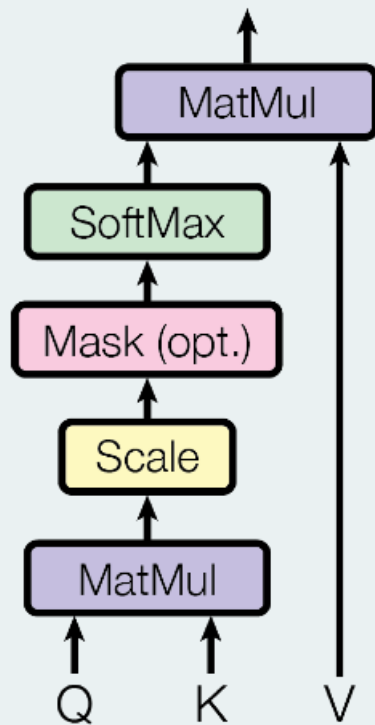
$$\text{softmax} \left(\frac{Q \times K^T}{\sqrt{d_k}} \right) \times V = \text{Attention Value Matrix } a$$

An Actual Attention Head: slightly more complicated

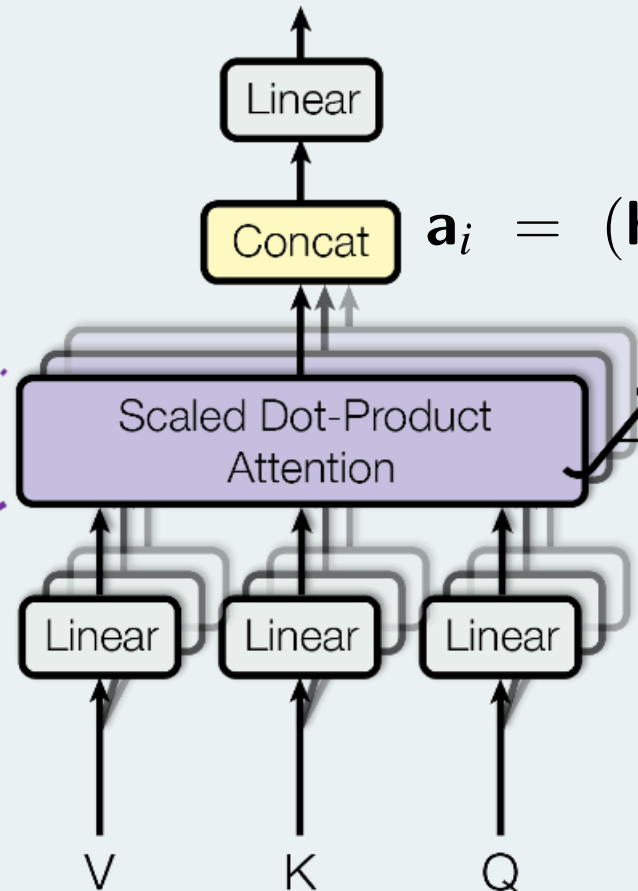
- High-level idea: instead of using vectors (like x_i and x_4) directly, we'll represent 3 separate roles each vector x_i plays:
- **query**: *As the current element* being compared to the preceding inputs.
- **key**: *as a preceding input* that is being compared to the current element to determine a similarity
- **value**: a value of a preceding element that gets weighted and summed

An Actual Attention Head: slightly more complicated

Scaled Dot-Product Attention



Multi-Head Attention



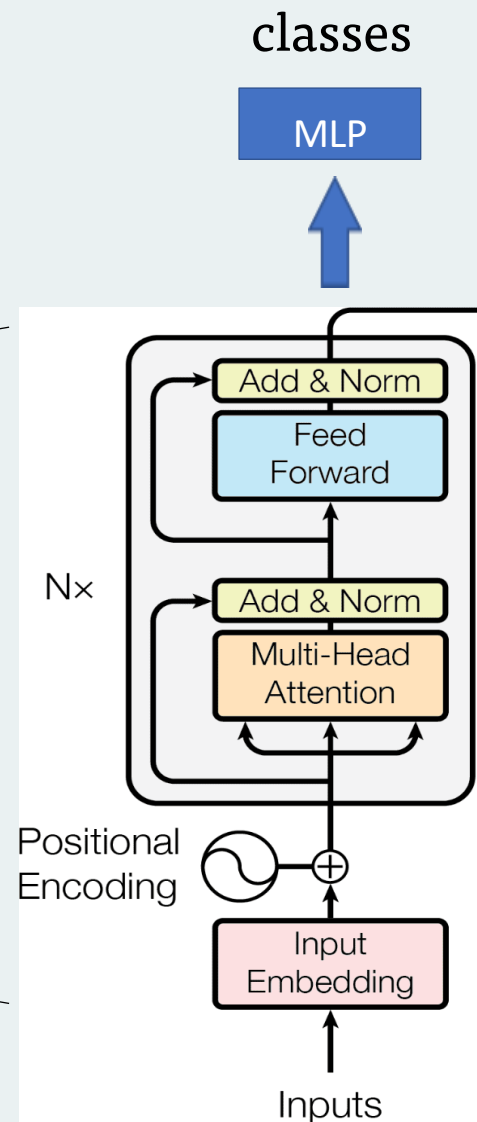
$$\mathbf{a}_i = (\mathbf{head}^1 \oplus \mathbf{head}^2 \dots \oplus \mathbf{head}^h) \mathbf{W}^O$$

$$\mathbf{head}_i^c = \sum_{j \leq i} \alpha_{ij}^c \mathbf{v}_j^c$$

7. Pre-trained large language models

BERT

encodeur

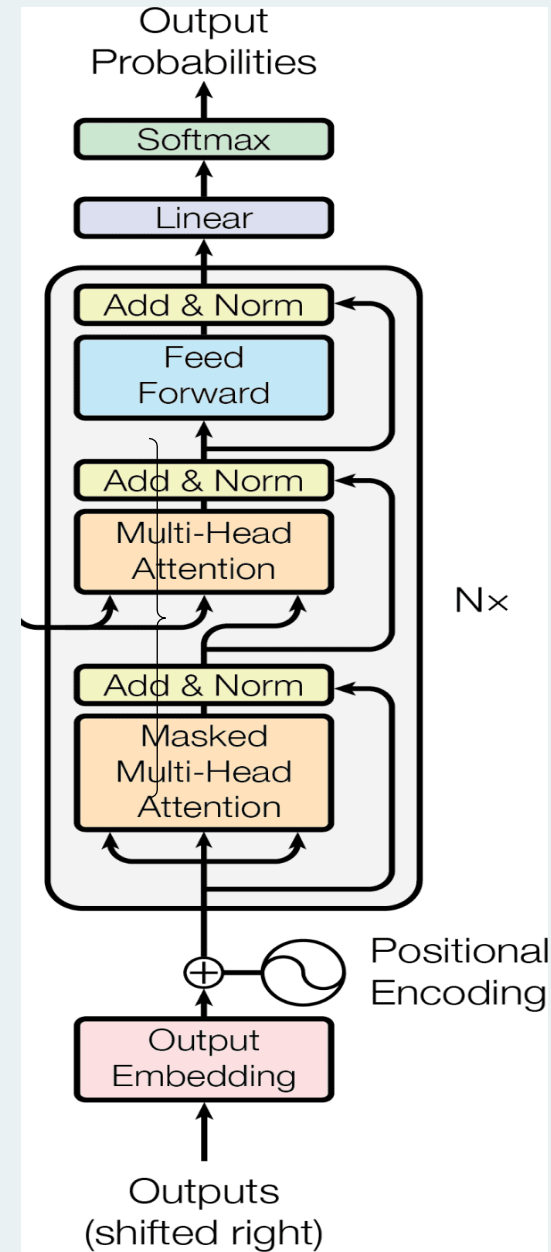


BERT(Devlin et al., 2018)

- Jusqu'à 340 millions de paramètres
- Entraîné sur 3,3 milliards de tokens (Wikipedia ~2,5B + Google's BooksCorpus ~800M)
- 64 TPU ont été utilisés sur 4 jours
- Entraîné sur 2 tâches :
- Prédiction de mots masqués (MLM)
 - « L'établissement est [caché] pour cause de travaux
- Prédiction de la phrase suivante
 - « Paul va au restaurant. Il commande un menu. » : OK
 - « Paul commande un café. Réduction sur le textile ! » : *pas* OK

GPT

Apprentissage : prochain mot



décodeur

GPT3 (Brown et al., 2020)

- Jusqu'à 175 milliards de paramètres
- Entraîné sur presque 500 milliards de tokens (version améliorée du CommonCrawl, WebText, books corpora, English-language Wikipedia)
- Tâche d'entraînement : prédiction du mot suivant (tâche auto-régressive)
- Résultats impressionnants en 0 / few-shot sur de nombreuses tâches : prédiction de mot, questions- réponses, traduction

GPT-3:

Language Models are Few-Shot Learners

- Scaling up language models greatly improves task-agnostic, few-shot performance. # decoder blocks=96, Text size=570GB
- 175B parameters

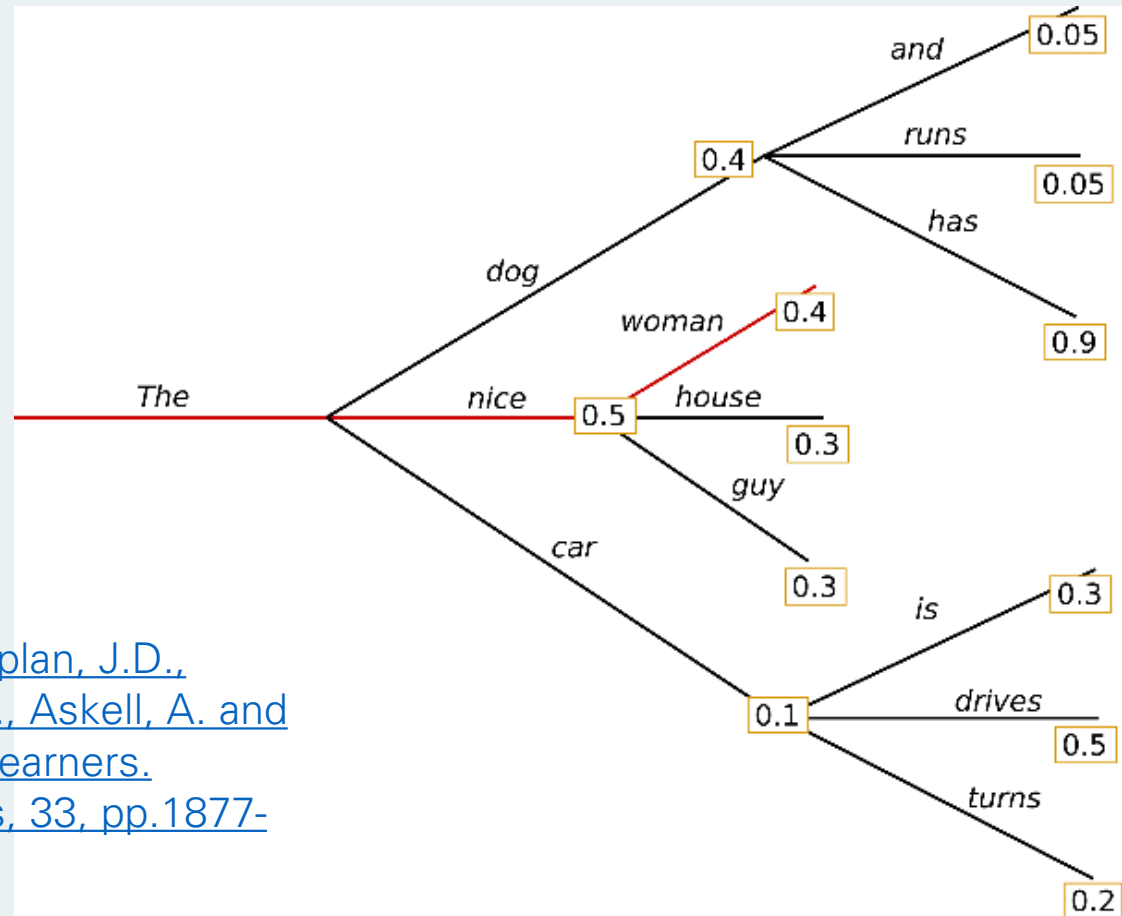
Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or “GPT-3”	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Table 2.1: Sizes, architectures, and learning hyper-parameters (batch size in tokens and learning rate) of the models which we trained. All models were trained for a total of 300 billion tokens.

[3] [Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A. and Agarwal, S., 2020. Language models are few-shot learners. Advances in neural information processing systems, 33, pp.1877-1901.](#)

GPT-3: Language Models are Few-Shot Learners

- Using Beam Search for predicting
 - Width=4



[3] [Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A. and Agarwal, S., 2020. Language models are few-shot learners. Advances in neural information processing systems, 33, pp.1877-1901.](#)

[4] <https://huggingface.co/blog/how-to-generate>

GPT-3:

Language Models are Few-Shot Learners

- Evaluation (=running, no fine-tuning here)

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
1  Translate English to French:  ← task description
2  cheese => .....           ← prompt
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
1  Translate English to French:  ← task description
2  sea otter => loutre de mer    ← example
3  cheese => .....             ← prompt
```

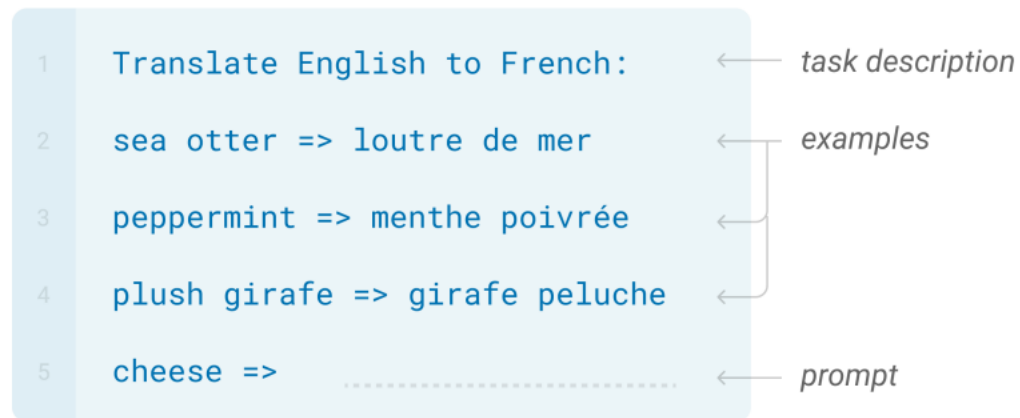
GPT-3:

Language Models are Few-Shot Learners

- Evaluation (=running, no fine-tuning here)

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.



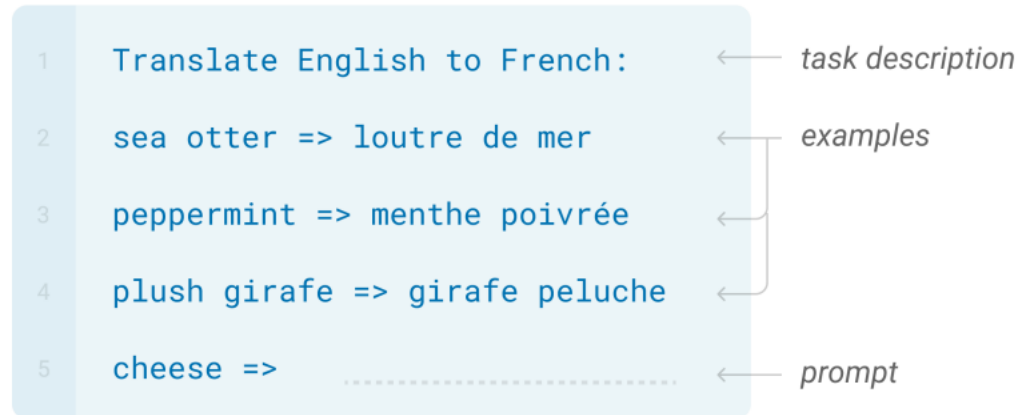
GPT-3:

Language Models are Few-Shot Learners

- Evaluation

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.



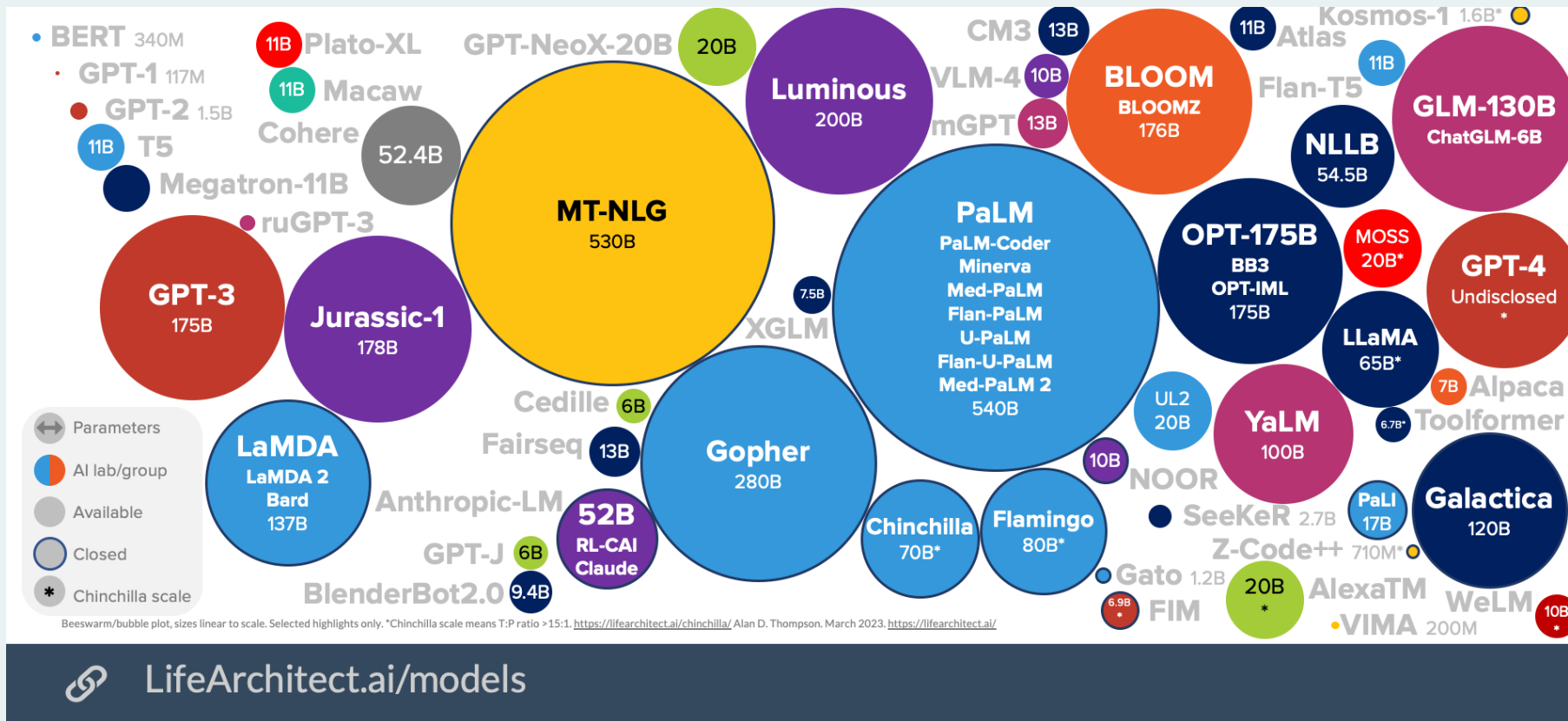
Fine-tuning

The model is trained via repeated gradient updates using a large corpus of example tasks.



Large language models

- Large language models have $> 1\text{B}$ parameters θ



Decoder-only:

- PaLM
- MT-NLG
- GPT-3

Une IA de confiance

- Limites des LLMs : effet « boîte noire », hallucinations
- Les solutions développées aujourd'hui :
 - Modèles « explicables » ou interprétables (**XAI**)
 - Solutions couplées à des techniques de recherche d'information sur des bases de données identifiées (**principe RAG** = *Retrieval Augmented Generation*)

Une IA éthique

- Limites des LLMs : apprentissage par cœur de **données sensibles** (vie privée, propriété intellectuelle), génération de textes **biaisés** vis-à-vis d'une certaine population, génération de textes potentiellement dangereux, racistes, etc.
- Les solutions développées aujourd'hui :
 - Mieux **encadrer** le développement et les usages des solutions d'IA (voir par ex. l'[AI act](#))
 - Recherches sur l'**évaluation** des modèles vis-à-vis des questions de *fairness* et l'**alignement** (cf. par ex. le [projet DIKé](#))