Learning and deep neural networks

Xihan Li
Department of Computer Science,
University College London
xihan.li@cs.ucl.ac.uk
https://snowkylin.github.io

Feb 2024

Contents

- Lecture 1: Multiagent AI and basic game theory
- Lecture 2: Repeated games
- Lecture 3: Potential games
- Lecture 4: Solving (“Learning”) Nash Equilibria (1)
- Lecture 5: Solving (“Learning”) Nash Equilibria (2)
- **Lecture 6: Learning and deep neural networks**  We are here
- Lecture 7: Single-agent Learning (1)
- Lecture 8: Multi-agent Learning (1)
- Lecture 9: Single-agent Learning (2)
- Lecture 10: Multi-agent Learning (2)

Outline

Before diving into the details…

- Introduction
- History of Deep Learning

Cornerstones
(that you must know)

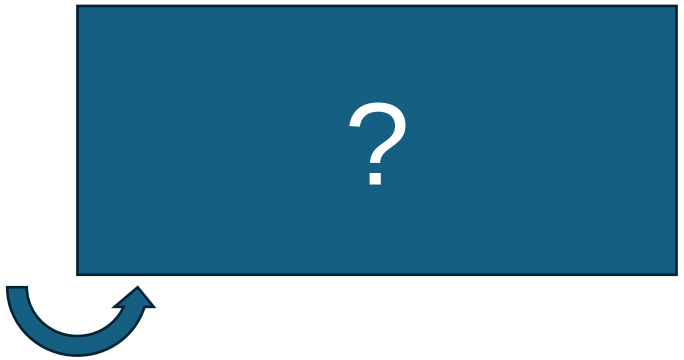
- Computational model of a neuron
- Gradient Descent
- Multilayer Neural Networks and Backpropagation

Simple but useful
(Old-fashioned? Maybe not)

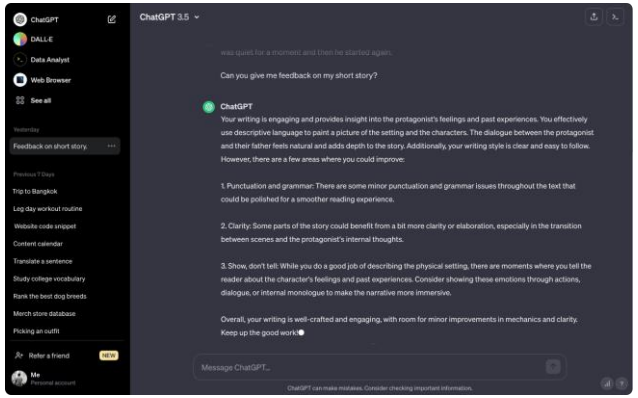
- Convolutional Neural Networks (CNN) for image
- Recurrent Neural Networks (RNN) for text

State of the art!
(super hot topic)

- Transformer
  (the technique behind ChatGPT/Gemini)



Deep Neural Networks are everywhere



Text (audio)
(Dialogue generation via ChatGPT)

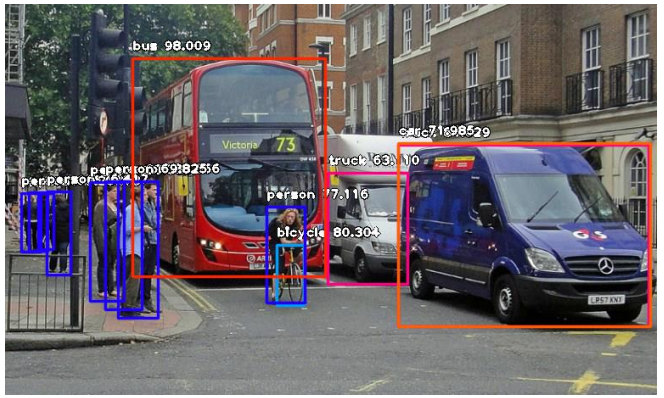
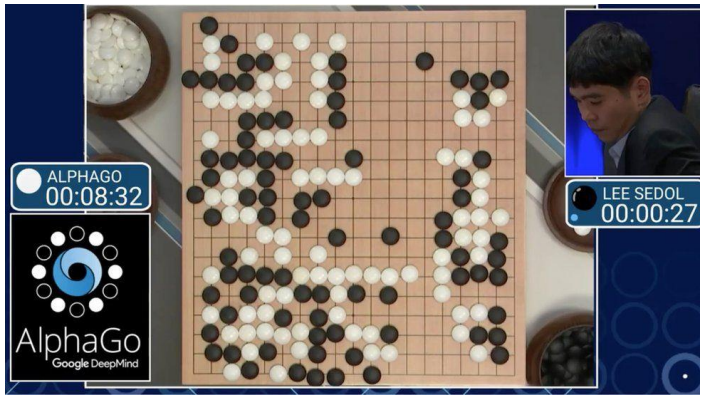
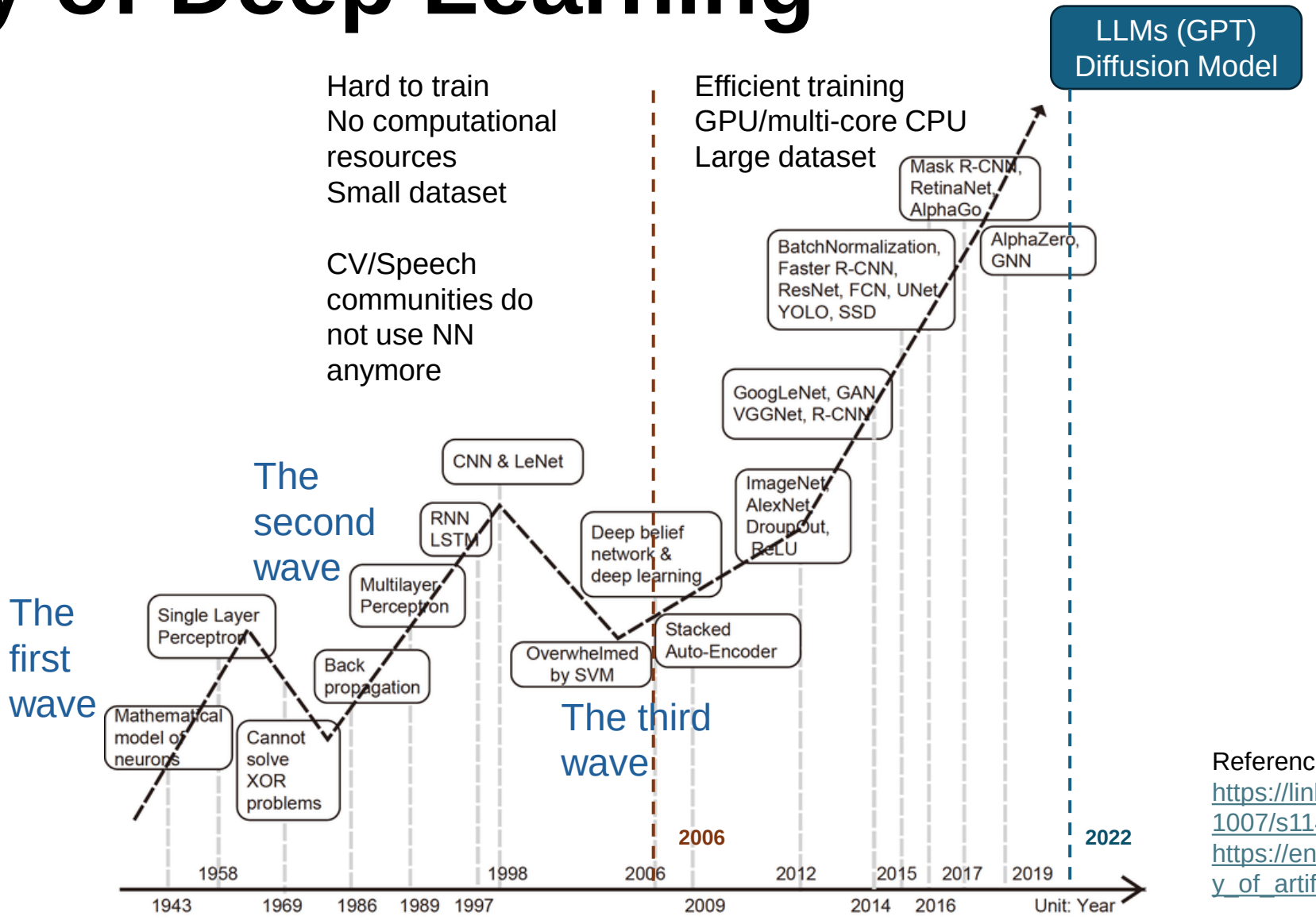


Image (video)
(Object Detection via YOLO)



Decision Making
(Playing Go via AlphaGo)

History of Deep Learning



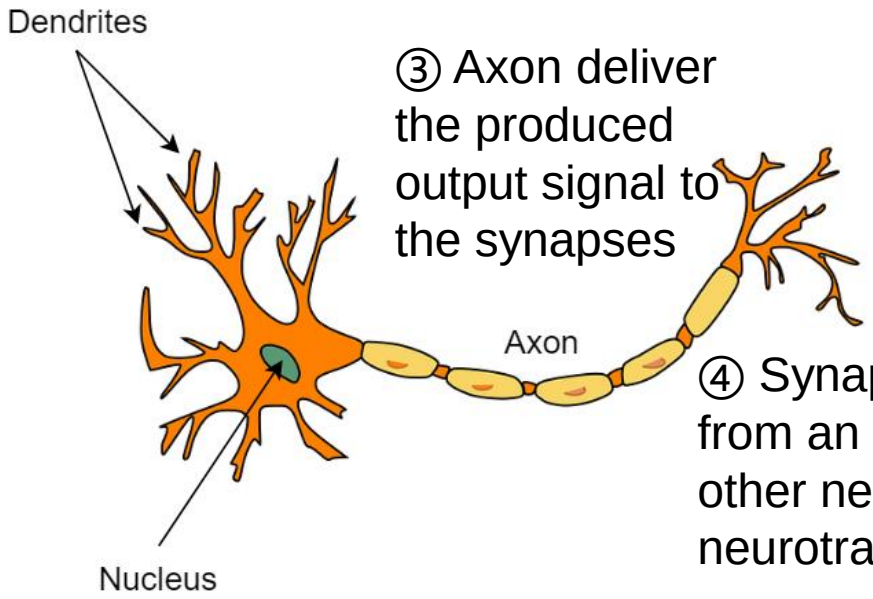
Reference:
https://link.springer.com/article/10.1007/s11430-019-9584-9
https://en.wikipedia.org/wiki/History_of_artificial_intelligence

Cornerstones

- Computational model of a neuron
- Gradient Descent
- Multilayer Neural Networks and Backpropagation

Computational model of a neuron

① Dendrites receive input signals from other neurons

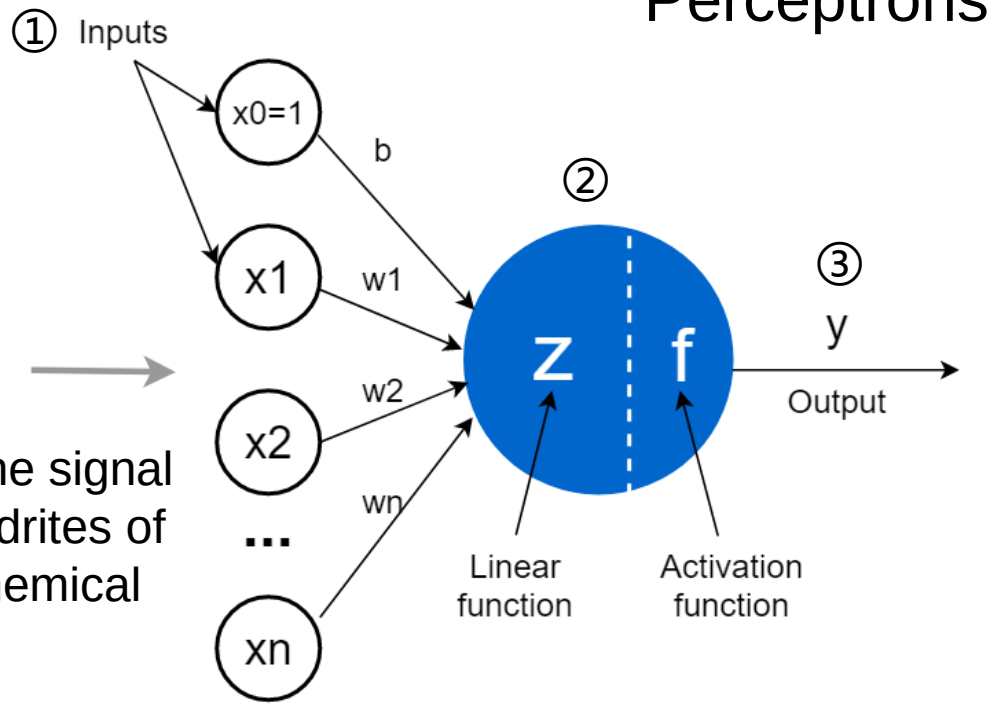


③ Axon deliver the produced output signal to the synapses

④ Synapses pass the signal from an axon to dendrites of other neurons via chemical neurotransmitter

② Nucleus process the input signals (different inputs have different importance) and produce an electrical output signal

Perceptrons



Input signals: $x = (x_1, \dots, x_n)$
Output signal: y
Parameters of the neuron model: $w = (w_1, \dots, w_n)$ and b

Computational process:

$$z = w_1 x_1 + \cdots + w_n x_n + b$$
$$y = f(z)$$

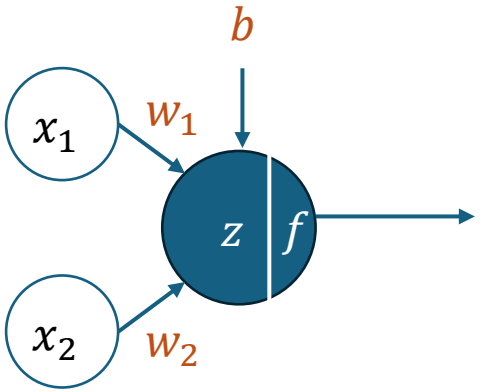
Reference:
https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc

Searching parameters

- Given a set of inputs with their corresponding "desired" outputs, finding the value of parameters w and b, so that the behavior of the perceptron model is aligned with the given data.

Desired behavior of the neuron

x_1	x_2	d
1	1	-1
1	2	-1
2	1	-1
2	2	1



$$z = w_1 x_1 + w_2 x_2 + b$$

$$f(x) = \begin{cases} 1, x \geq 0 \\ -1, otherwise \end{cases}$$

$$y = f(z)$$

Finding parameters w_1, w_2, b so that the above perceptron model behaves as in the left table

Solution: transforming it into an **optimization problem**

x_1	x_2	d	Model output y
1	1	-1	$f(w_1 + w_2 + b)$
1	2	-1	$f(w_1 + 2w_2 + b)$
2	1	-1	$f(2w_1 + w_2 + b)$
2	2	1	$f(2w_1 + 2w_2 + b)$

Loss function $L = \sum_i (d_i - y_i)^2$
We should find w_1, w_2, b that minimize L ! (ideally 0)

Optimization via gradient descent

	Gradient Descent	Linear Programming
Objective	Any differentiable function	Linear function
Constraint	N/A	Linear constraints

Gradient descent is another optimization technique to maximize/minimize a given function. which run two steps iteratively: ① compute gradient ② update variables guided by gradient

$y = f(x)$	$\frac{dy}{dx} = f'(x)$
k, any constant	0
x	1
x^2	$2x$
x^3	$3x^2$
x^n, any constant n	nx^{n-1}
e^x	e^x
e^{kx}	ke^{kx}
$\ln x = \log_e x$	$\frac{1}{x}$
$\sin x$	$\cos x$

Example: finding x and y that minimize $L = x^2 + y^2 + 2x - 2y$

Preparation: compute the gradient of L w.r.t. x and y

$\frac{\partial L}{\partial x} = 2x + 2$ (regard y as a constant), $\frac{\partial L}{\partial y} = 2y - 2$ (regard x as a constant)

Then initialize (x, y) randomly, and do the following iteratively:

① compute $g = \frac{\partial L}{\partial x}$ and $h = \frac{\partial L}{\partial y}$ (here they are $g = 2x + 2$ and $h = 2y - 2$)

② update x, y via $x \leftarrow x - \alpha g$ and $y \leftarrow y - \alpha h$ (α is a small learning rate)

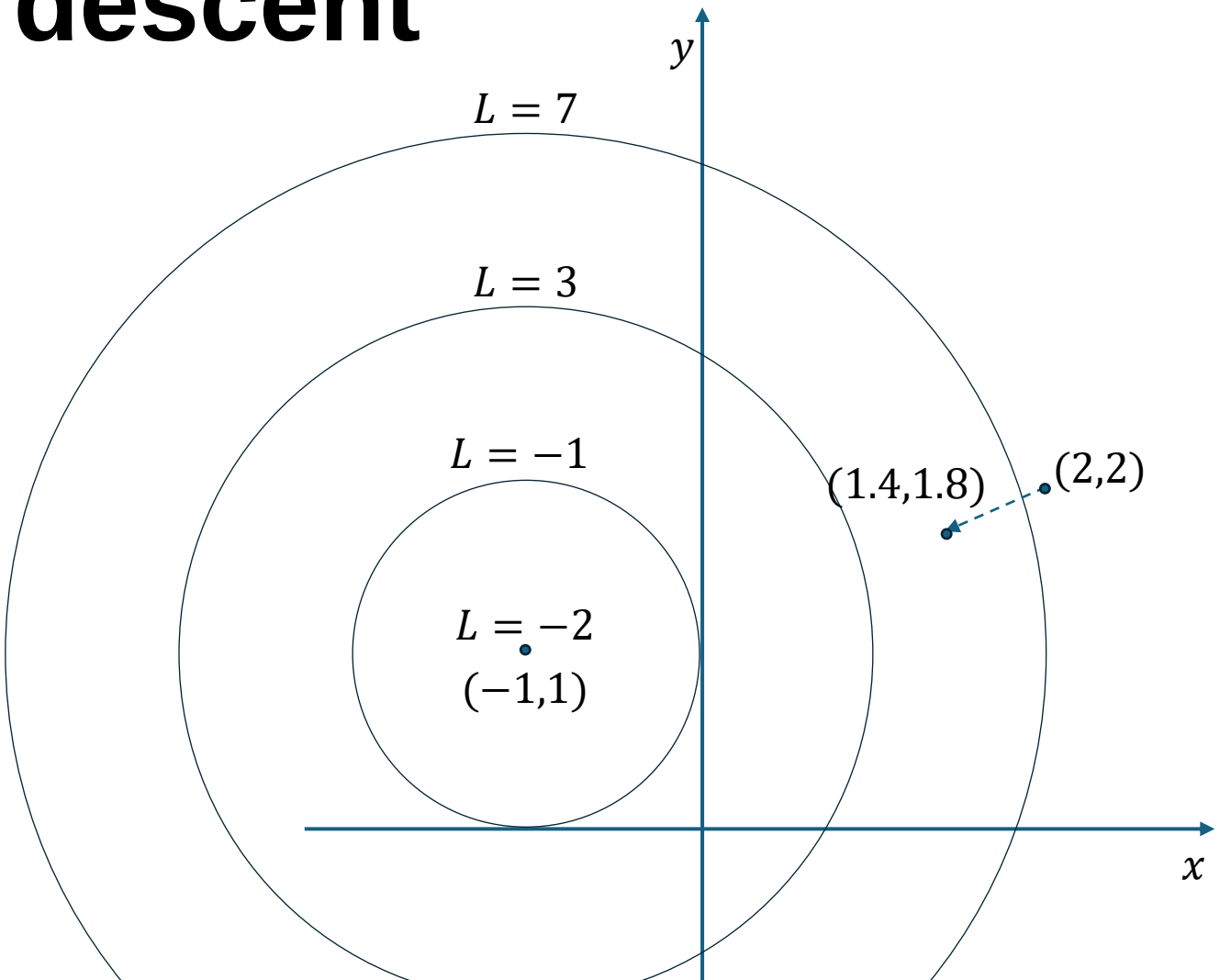
Until g and h are close to zero.

$\alpha = 0.1$

	x	y	$g = \frac{\partial L}{\partial x}$	$h = \frac{\partial L}{\partial y}$	L
0	2	2	6	2	8
1	1.4	1.8	5.6	1.6	4.4
	…	…	…	…	…
T	-1	1	0	0	-2

L is decreasing!

Optimization via gradient descent



$$L = x^2 + y^2 + 2x - 2y$$

$(-\frac{\partial L}{\partial x}, -\frac{\partial L}{\partial y})$ points to the direction that leads to fastest descent

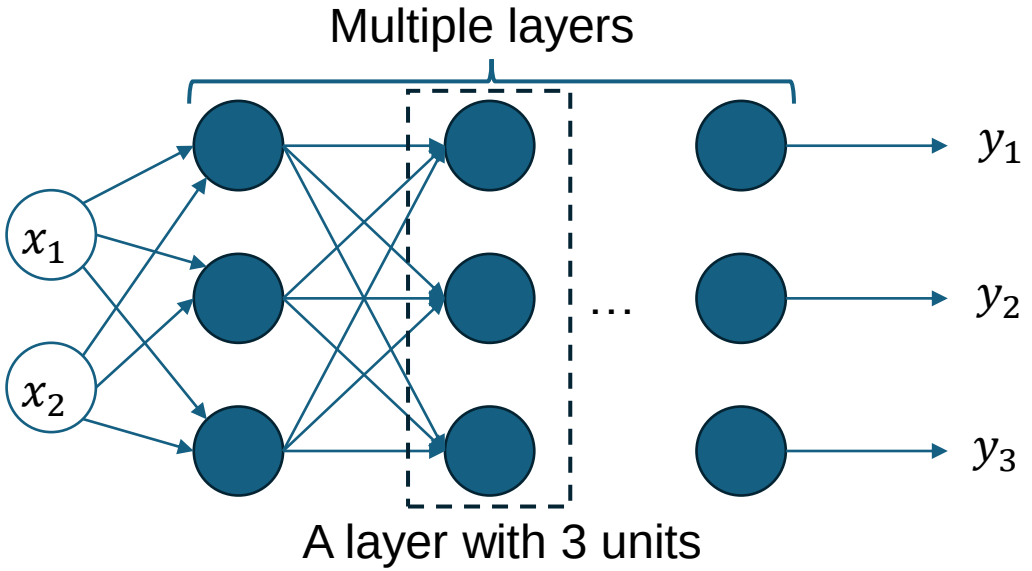
$\alpha = 0.1$

	x	y	$g = \frac{\partial L}{\partial x}$	$h = \frac{\partial L}{\partial y}$	L
0	2	2	6	2	8
1	1.4	1.8	5.6	1.6	4.4
	…	…	…	…	…
T	-1	1	0	0	-2

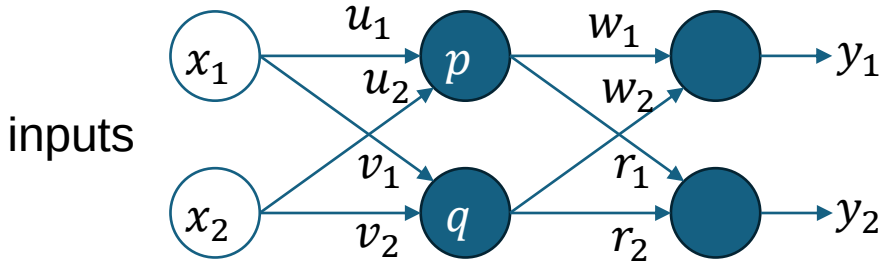
L is decreasing!

Multilayer Neural Networks

- Now we can find suitable parameters for a single neuron model, to mimic given expected behaviors.
- However, the capacity of a single neuron is very limited
  - (consider the XOR logic function, why a single neuron model cannot mimic it?)
- Solution: stack multiple neuron models horizontally and vertically!



Backpropagation



Here we omitted the bias term for simplicity

$$\frac{\partial(a+b)}{\partial x} = \frac{\partial a}{\partial x} + \frac{\partial b}{\partial x}$$
$$\frac{\partial f(y)}{\partial x} = \frac{\partial f(y)}{\partial y}\frac{\partial y}{\partial x}$$
$$\frac{\partial f(a,b)}{\partial x} = \frac{\partial f(a,b)}{\partial a}\frac{\partial a}{\partial x} + \frac{\partial f(a,b)}{\partial b}\frac{\partial b}{\partial x}$$

- Feedforward:
  - $p = f(u_1x_1 + u_2x_2), q = f(v_1x_1 + v_2x_2)$
  - $y_1 = f(w_1p + w_2q), y_2 = f(r_1p + r_2q)$
  - $L_1 = (y_1 - d_1)^2, L_2 = (y_2 - d_2)^2$
  - $L = L_1 + L_2$

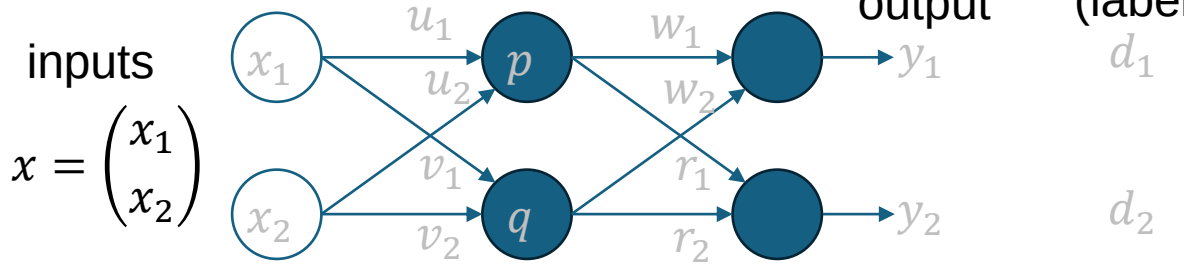
$$L = \underbrace{(f(w_1p + w_2q)}_{y_1} - d_1)^2 + \underbrace{(f(r_1p + r_2q)}_{y_2} - d_2)^2$$
$$L_1 \qquad\qquad L_2$$

- Backpropagation (finding the gradient of loss function L w.r.t variables w, r, u, v)
  - $\frac{\partial L}{\partial y_1} = 2(y_1 - d_1), \frac{\partial L}{\partial y_2} = 2(y_2 - d_2)$
  - $\frac{\partial L}{\partial w_1} = \frac{\partial L_1}{\partial w_1} + 0 = \frac{\partial L}{\partial y_1}\frac{\partial y}{\partial w_1} = \frac{\partial L}{\partial y_1}f'(w_1p + w_2q)p, \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial y_2}\frac{\partial y}{\partial w_2} = \frac{\partial L}{\partial y_2}f'(r_1p + r_2q)q$ (similar for $\frac{\partial L}{\partial r_1}$ and $\frac{\partial L}{\partial r_2}$)
  - $\frac{\partial L}{\partial p} = \frac{\partial L_1}{\partial p} + \frac{\partial L_2}{\partial p} = \frac{\partial L_1}{\partial y_1}\frac{\partial y_1}{p} + \frac{\partial L_2}{\partial y_2}\frac{\partial y_2}{p} = \frac{\partial L}{\partial y_1}f'(w_1p + w_2q)w_1 + \frac{\partial L}{\partial y_2}f'(r_1p + r_2q)r_1$ (similar for $\frac{\partial L}{\partial q}$)
  - $\frac{\partial L}{\partial u_1} = \frac{\partial L}{\partial p}\frac{\partial p}{\partial u_1} = \frac{\partial L}{\partial p}f'(u_1x_1 + u_2x_2)x_1, \frac{\partial L}{\partial u_2} = \frac{\partial L}{\partial p}\frac{\partial p}{\partial u_2} = \frac{\partial L}{\partial p}f'(u_1x_1 + u_2x_2)x_2$ (similar for $\frac{\partial L}{\partial v_1}$ and $\frac{\partial L}{\partial v_2}$)

Feedforward network in matrix form

Here we omitted the bias term for simplicity

$$V = \begin{pmatrix} u_1 & u_2 \\ v_1 & v_2 \end{pmatrix} \quad W = \begin{pmatrix} w_1 & w_2 \\ r_1 & r_2 \end{pmatrix}$$



$$z = \begin{pmatrix} p \\ q \end{pmatrix} = f(Vx) \quad y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = f(Wz)$$

In such a way we can simply write the feedforward process as

$$y = f\big(Wf(Vx)\big)$$

with parameters W and V

Training a feedforward neural network:

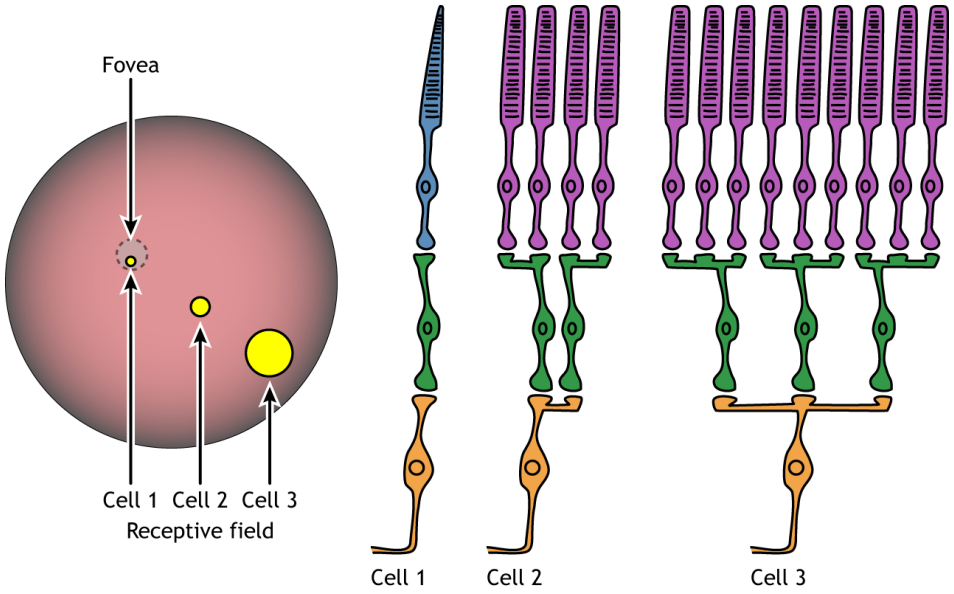
Given dataset (X, D), initialize parameters W, V
While not converged:
  sample data x, d from (X, D)
  compute model output $y = f\big(Wf(Vx)\big)$
  compute loss function $L = \|y - d\|^2$
  compute gradients $\frac{\partial L}{\partial W}$, $\frac{\partial L}{\partial V}$ via backpropagation
  update parameters via gradient descent
    $W \leftarrow W - \alpha \frac{\partial L}{\partial W}$, $V \leftarrow V - \alpha \frac{\partial L}{\partial V}$

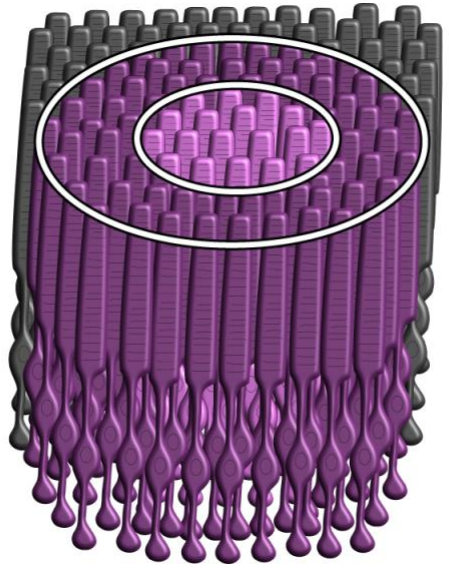
Basic Neural networks for image and text

- Convolutional Neural Networks (CNN) – spatial connection
- Recurrent Neural Networks (RNN) – temporal connection

Receptive field

- Different from the fully-connected case, neurons in the retina respond to light stimulus in **restricted regions** of the visual field

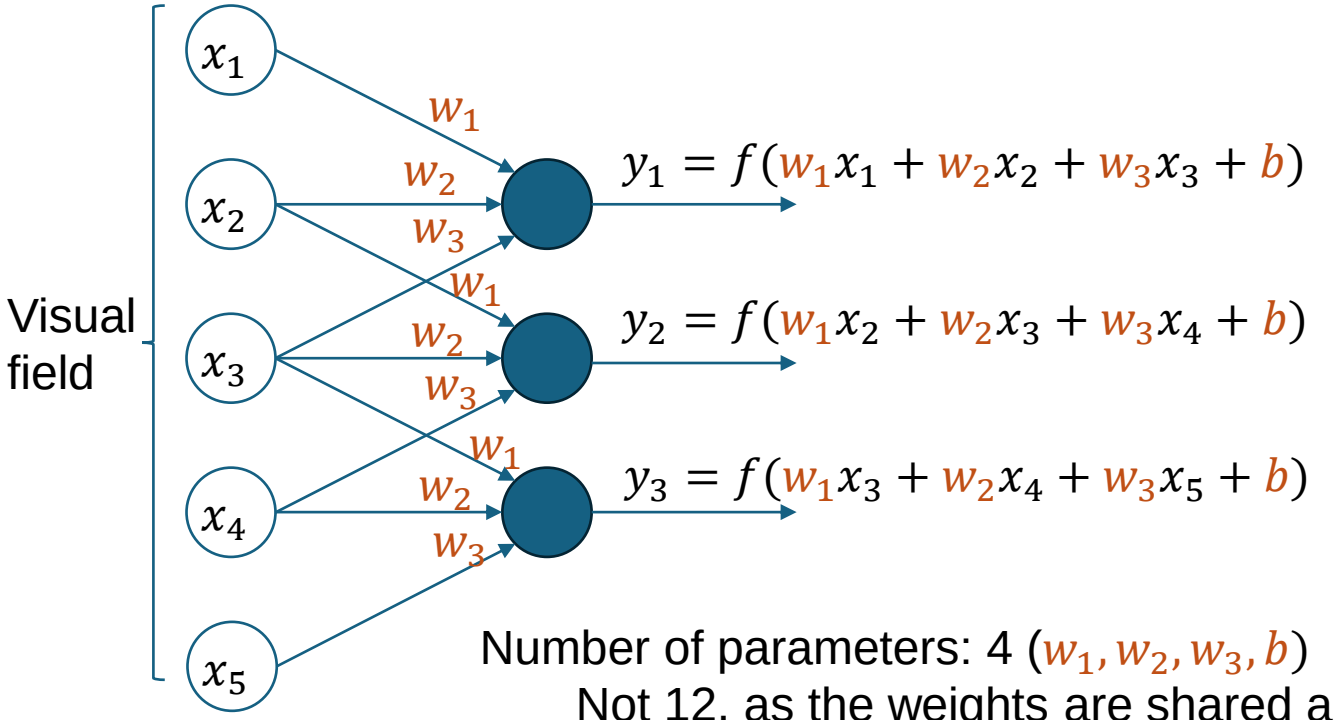




Reference:
https://openbooks.lib.msu.edu/neuroscience/chapter/vision-the-retina/

Convolutional layer (1D)

- To mimic the characteristic of retina neurons, we design a special way of connection that is
  - **Sparsely, local connected**: each output only connects to its nearest k inputs
  - **Shared weight**: the weight is replicated across the entire visual field
- We named it as a “filter”



Number of parameters: 4 (w_1, w_2, w_3, b)

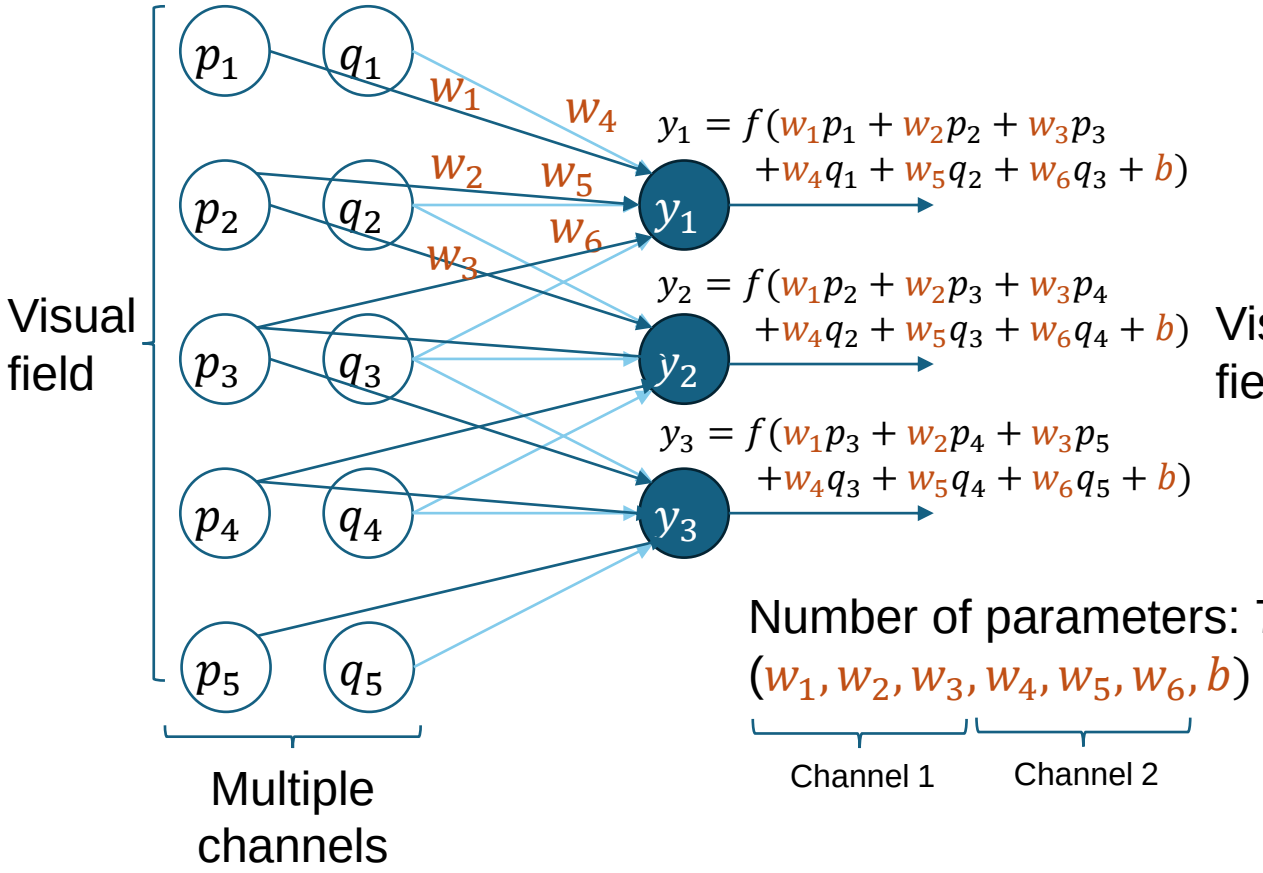
Not 12, as the weights are shared across the visual field

Size of the receptive field: 3

Each output connects to 3 inputs, not 5

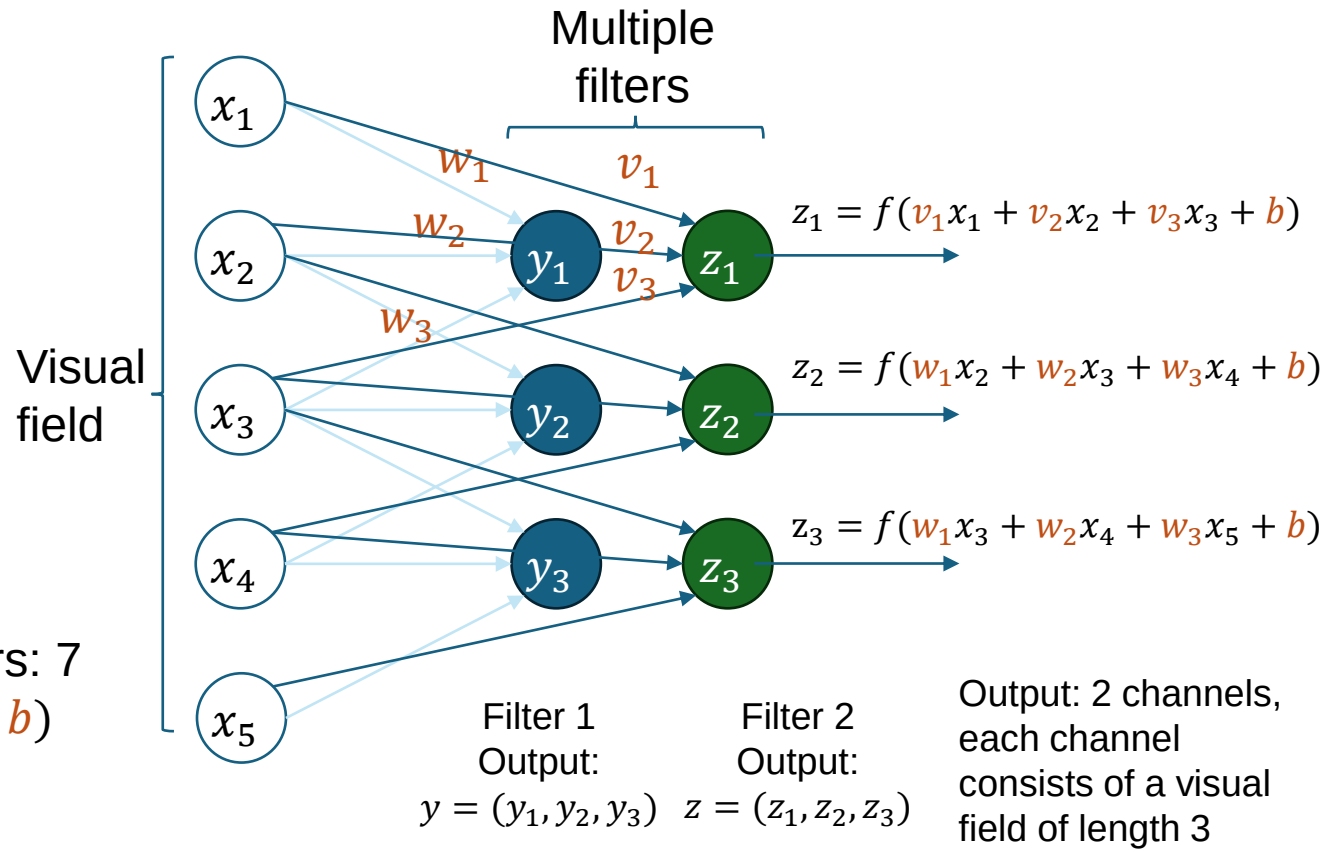
Convolutional layer (1D)

① One filter can process multiple channels of visual field



② Multiple filters can work simultaneously on the same visual field

A convolutional layer usually consists of multiple filters

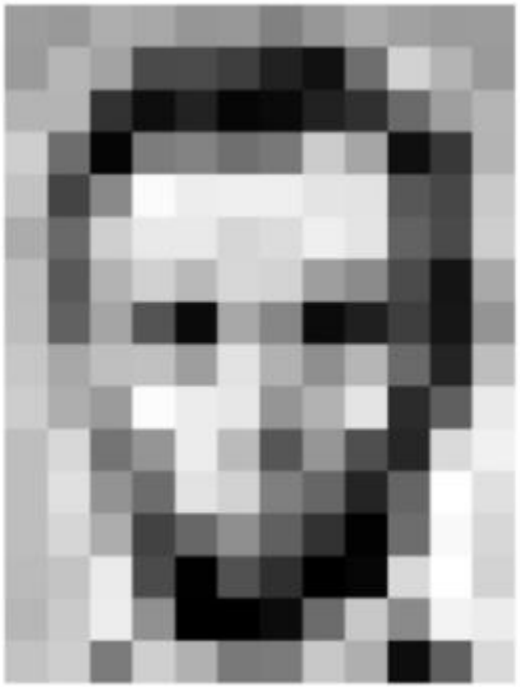


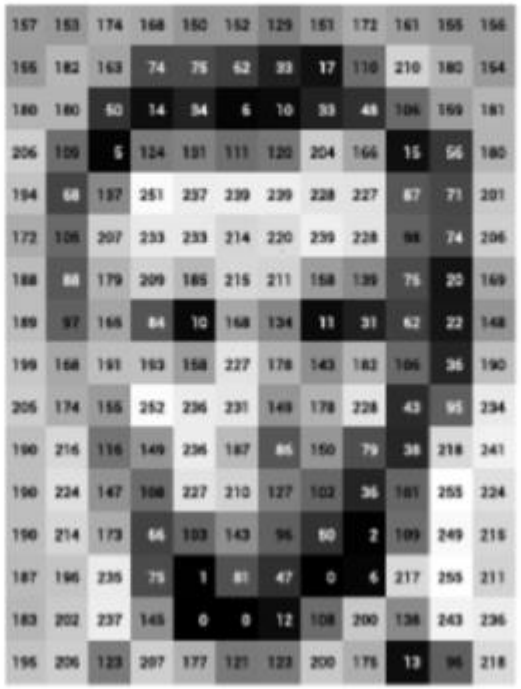
How to represent an image in a computer

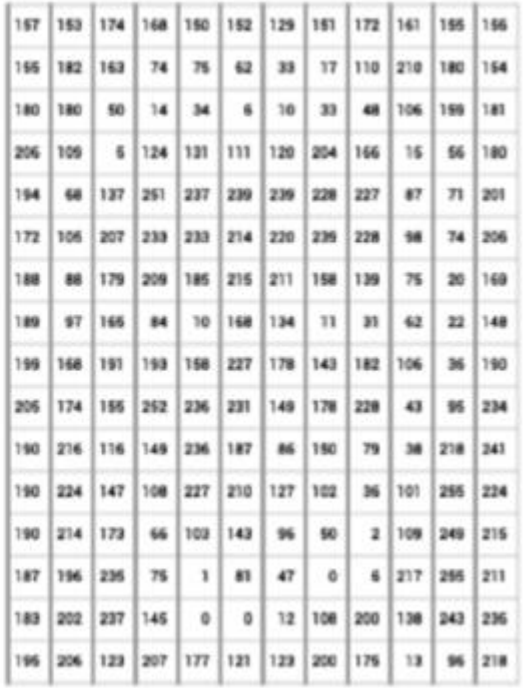
One or more 2D arrays.

Value typically from 0 (darkest) – 255 (brightest) (or 0.0 – 1.0 as a float)

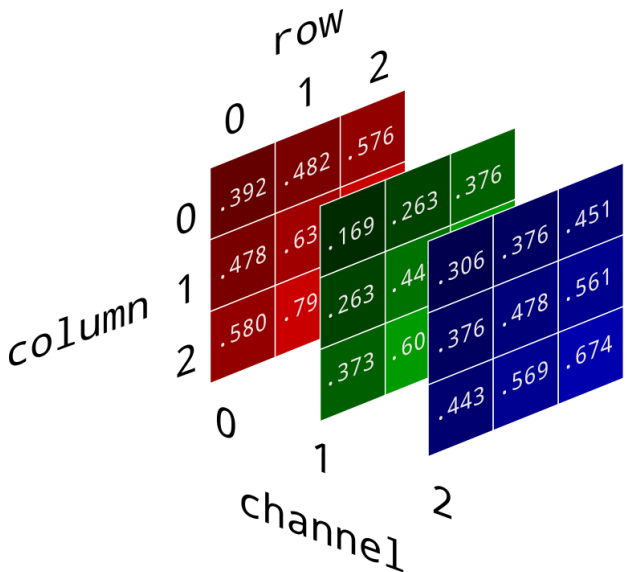
Colored image – three channels (red, green, blue)





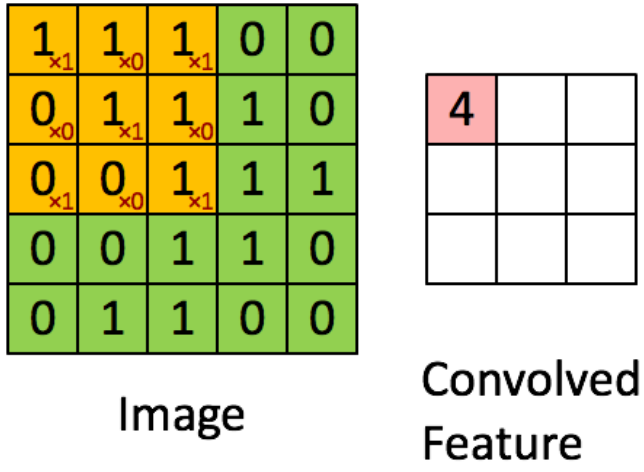


	Red	Green	Blue	Hexadecimal code
	0	0	0	#000000
	255	255	255	#FFFFFF
	255	0	0	#FF0000
	0	255	0	#00FF00
	0	0	255	#0000FF
	255	128	0	#FF8000
	255	255	0	#FFFF00
	128	128	128	#808080



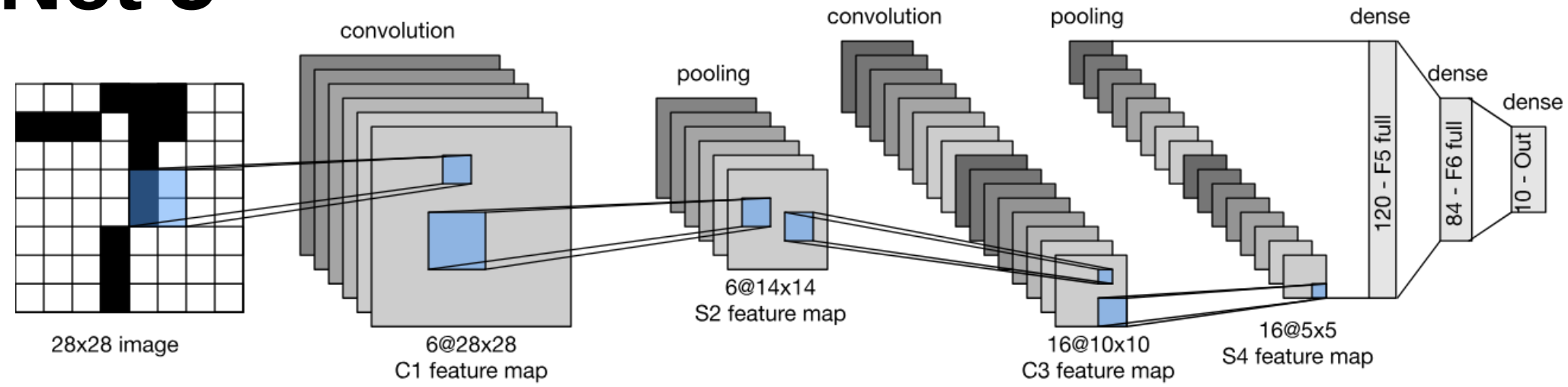
Convolutional layer (2D)

- A direct extension of the previous discussed filter, from 1D to 2D visual fields.
  - Input: from an 1D vector (size 5) to a 2D matrix (size 5×5)
  - Output: from an 1D vector (size 3) to a 2D matrix (size 3×3)
  - Receptive field: from an 1D sub-range (size 3) to a 2D sub-range (size 3×3)
- Other things are generally the same!



LeNet 5

Visual field = feature map
Size of receptive field = kernel size



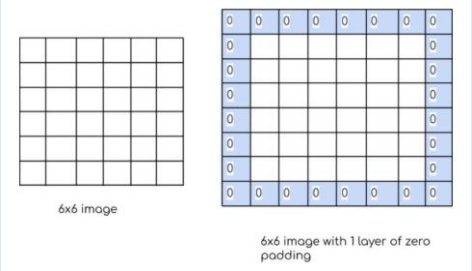
Receptive Field: 5×5
Number of filters: 6
Number of parameters:
$(5 \times 5 \times 1) \times 6$
With 2 paddings

Receptive Field: 5×5
Number of filters: 16
Number of parameters:
$(5 \times 5 \times 6) \times 16$
No padding

Reference:
https://d2l.ai/chapter_convolutional-neural-networks/lenet.html
Number of parameters excludes the bias term.

Padding

The size of the visual field will “shrink” after convolution
To recover the size, we add padding at the border of the visual field.



Pooling

A pooling layer slides a two-dimensional filter over each channel of visual field, and summarizes the value lying within the region covered by the filter.

1	2	2	3
2	1	3	2
2	3	1	2
3	2	2	1

1.5	2.5
2.5	1.5

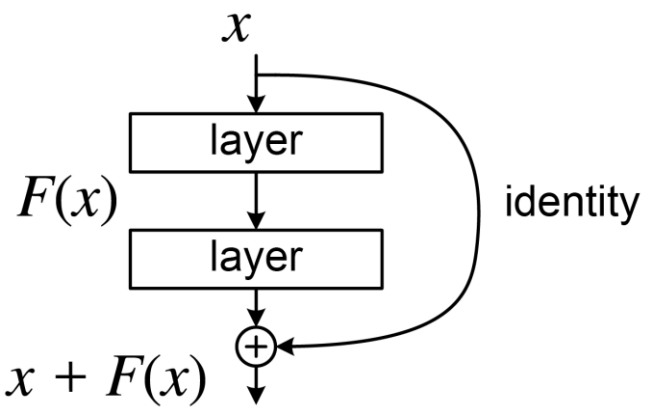
Average pooling

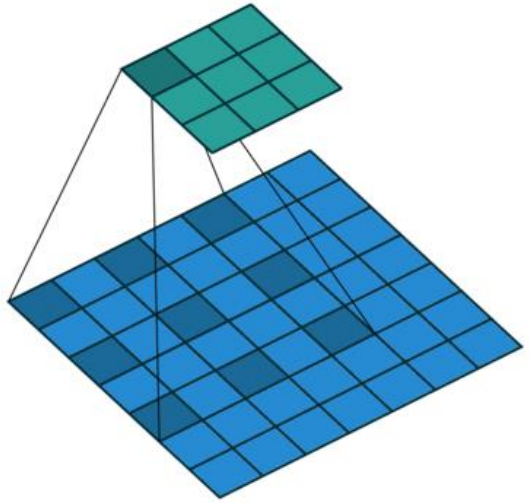
2	3
3	2

Max pooling

More about CNN

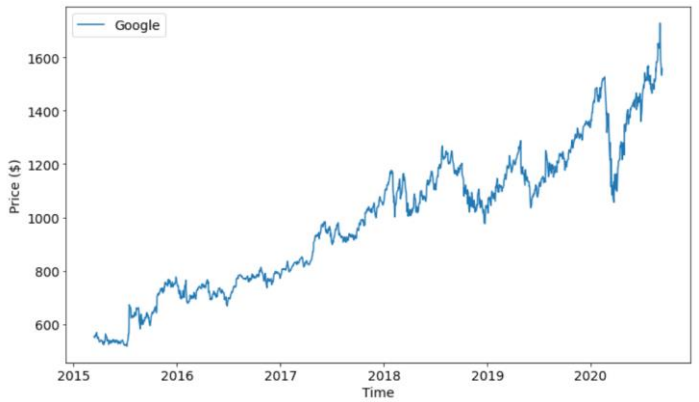
- Modern CNN (e.g., ResNet): https://d2l.ai/chapter_convolutional-modern/index.html
- Different types of convolution https://github.com/vdumoulin/conv_arithmetic

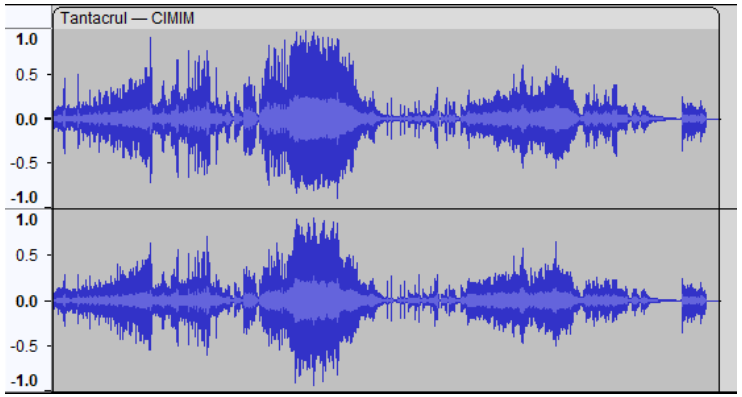




Sequential Data with temporal connections

- Time-series data (stock price)
- Audio
- …
- And the most common one, **text**

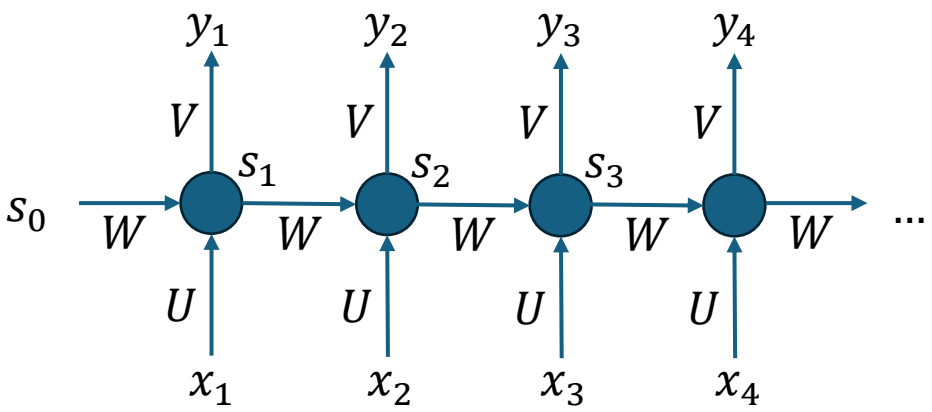




months that it's that it's really started to effect this but I know what it is that's because .
If you only want for the effect of being a clown. Yeah, I think you're
in Head and Shoulders it has the same effect as reversing it. I, I, a hairdresser told
t hold of say, the rainbow Yeah. effect of a Wurli I mean, that's the beauty abou
... 's what I mean. It may, if it's any effect at all it's very short lived I think. Mm
Yes. Oh yes. Lot of repetition. In effect. What's an ongoing topic? Politic
ll obviously, yeah. you know, for the effect and erm For the for the contrast, yea
nits finished in wooden set with marble effect roll topped work surface . Oh well that's y
t sure with my blades up it'll have much effect but we can try. Yeah, it would look nic
The trainer isn't. Just to get the full effect. Oh I was gonna turn this off Mm?
tually interview if I do effect all the Well you're all g
w them Without having a detrimental effect on the studying, you did what you could
'ell I would try and get something to that effect in writing. Yeah! Yeah. Where are the oth
ough, don't you agree? Or words to that effect, right, and I realize that you have to think
.. now that do have a, a, sort of a lasting effect. Yeah. I mean the majority of then
, and on London prices especially. This effect has been compounded by the natural fac
g he also gave his blessing to I what in effect proved to be the case I declaring the Tra
e wealthy which will have no significant effect on the economy and deepen the deficit.
rights of audience are put into practical effect as soon as the necessary conditions hav
y review nowhere considers the overall effect of the individual changes proposed, or h
from pure oxygen they found very little effect. Mike Roberts and colleagues at the
y Ian Snodin and Stuart McCall, to such effect during the second half that Steve Coppell
western with 'good demographics'. The effect is rather like an extended advertisement
looks even more refreshing, though its effect is that of a silver mallet. In the right plac
istorians have already raided it to good effect, notably Mark Girouard for his book on th
between bidders can have the opposite effect. Another recent auction in Leeds saw a ru
ing also creates an interesting highlight effect on the raised knitted details. The dye ten

Process sequential data with a recurrent neural network

- Assuming that the data is represented as $x_1, x_2, \ldots, x_T$ (each x_t is an n-dimensional vector)
- Initialize a state vector s of length h, and three parameters U, W, V (in matrix form)
- For t from 1 to T:
  - Update state: $s_t \leftarrow f(Ux_t + Ws_{t-1})$
  - Produce output: $y_t \leftarrow Vs_t$



Reference:
https://dennybritz.com/posts/wildml/recurrent-neural-networks-tutorial-part-1/

U: an $h \times n$ matrix transforming input x_t
W: an $h \times h$ matrix transforming previous input s_{t-1}
V: an $n \times h$ matrix transforming current input s_t
So we have $(2n + h)h$ parameters in an RNN (excluding bias)

How to represent a word for neural models?

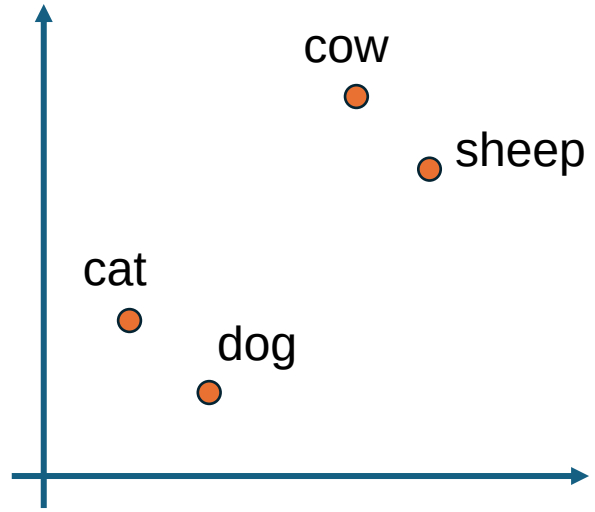
Neural models commonly use **vectors** to represent data

One-Hot Encoding

cat	1	0	0	0
dog	0	1	0	0
cow	0	0	1	0
sheep	0	0	0	1

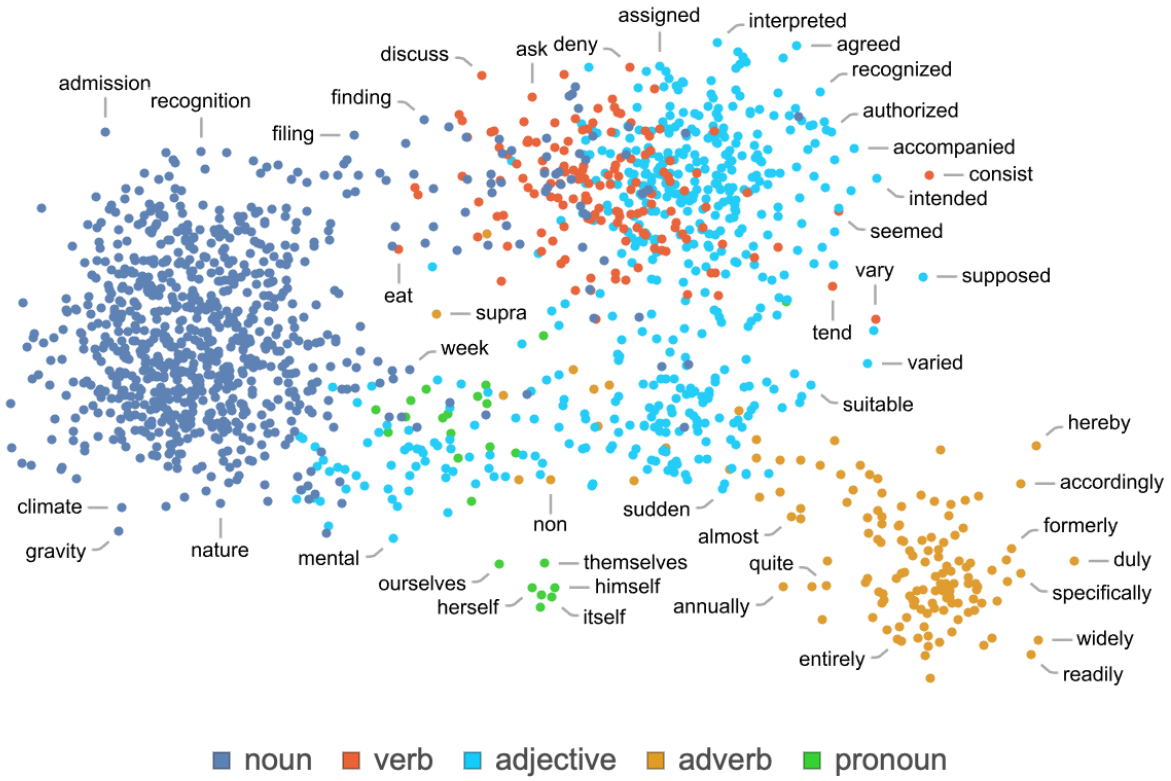
Word Embedding

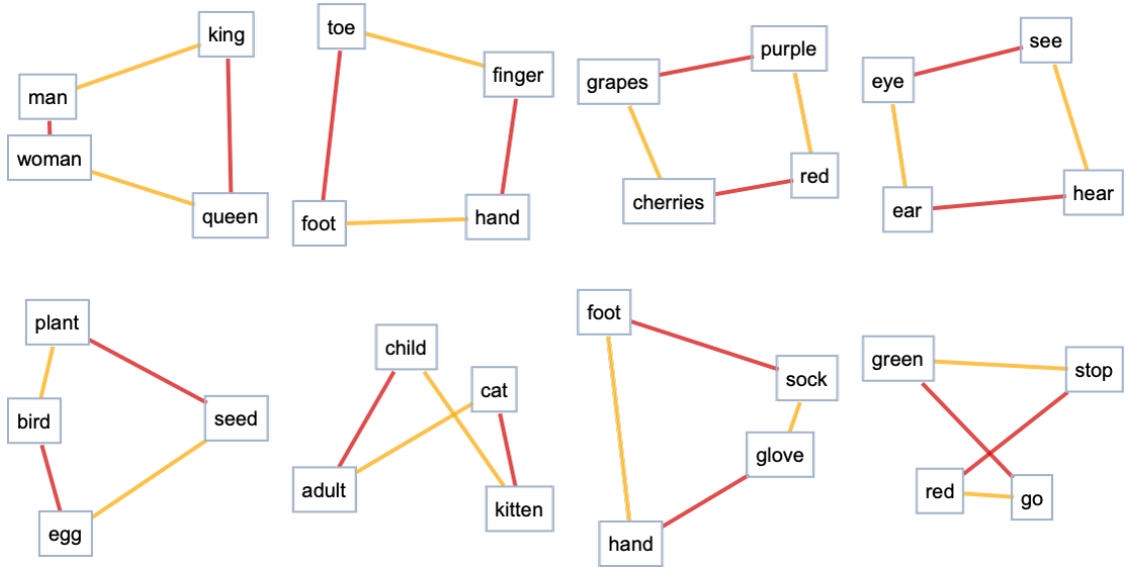
cat	0.1	0.2
dog	0.2	0.1
cow	0.8	0.7
sheep	0.7	0.8



"semantic space": for words with similar semantic meaning, their corresponding vectorized representations will also be closer

Word Embedding





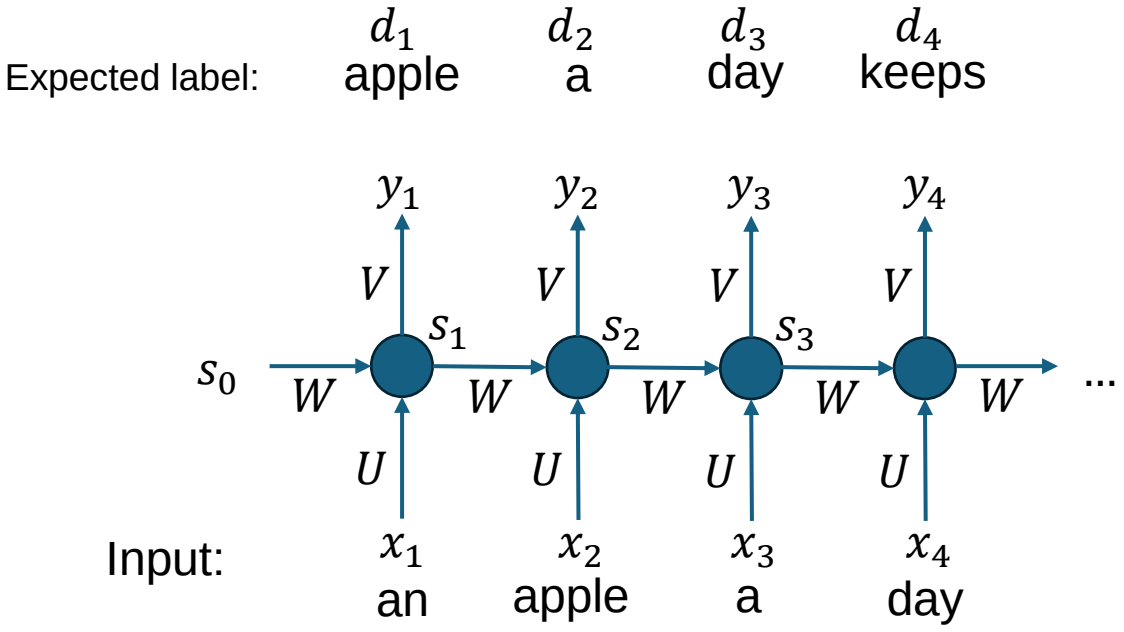
Reference:
https://writings.stephenwolfram.com/2023/02/what-is-chatgpt-doing-and-why-does-it-work/

RNN Example: next word prediction

Input $x_1, x_2, \ldots, x_T$

Expected label d_T

- An apple a day keeps ___ → the
- An apple a day keeps the ___ → doctor
- An apple a day keeps the doctor ___ → away



Training a recurrent neural network:

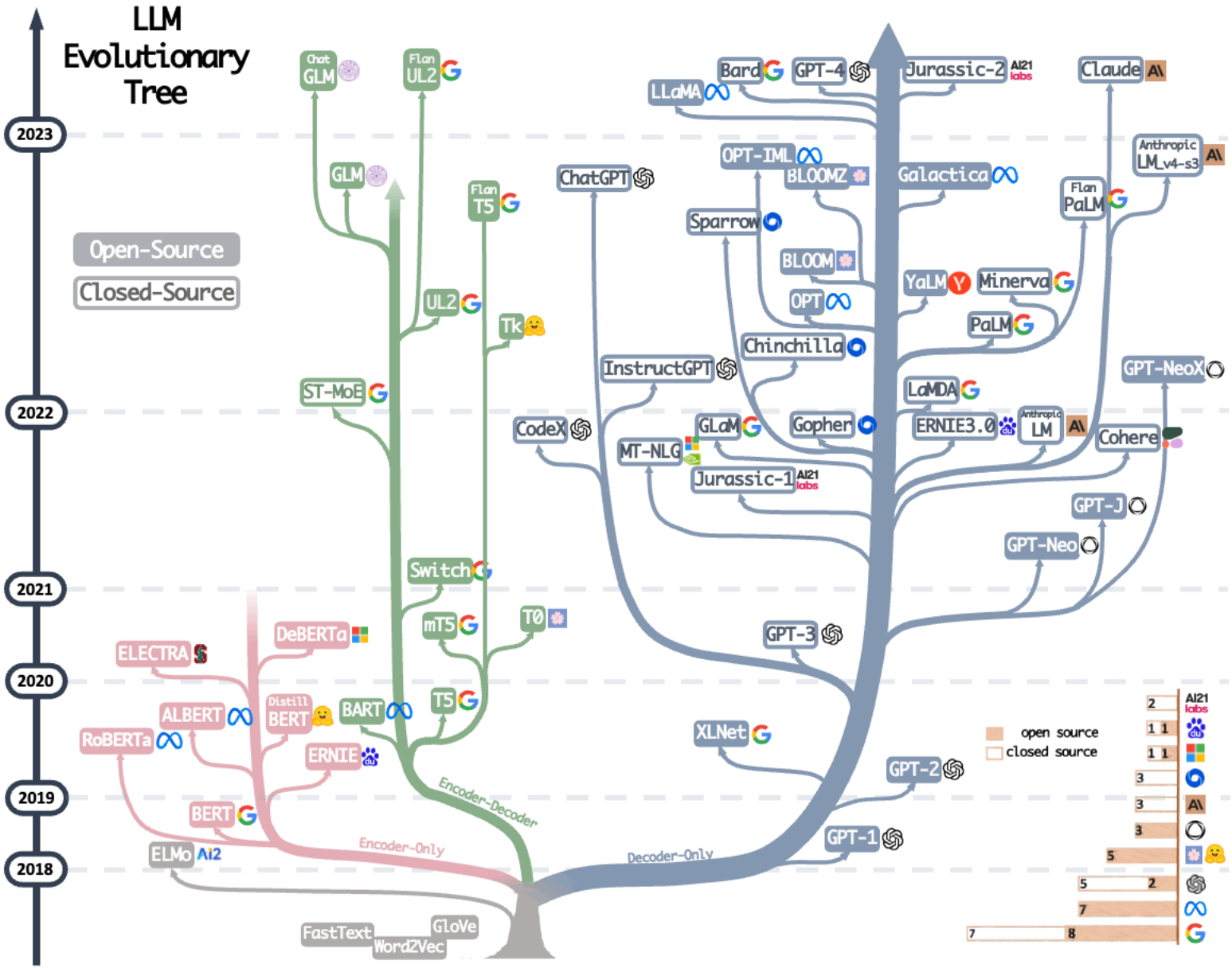
Given dataset (X, D), initialize parameters W, V
While not converged:
  sample data $x_1, x_2, \ldots, x_T, d$ from (X, D)
  For t from 1 to T:
    $s_t = f(Ux_t + Ws_{t-1}),\ y_t = Vs_t$
  compute loss function $L = \sum_t \|y_t - d_t\|^2$
  compute gradients $\frac{\partial L}{\partial U}, \frac{\partial L}{\partial W}, \frac{\partial L}{\partial V}$ via backpropagation
  update parameters via gradient descent
    $U \leftarrow U - \alpha \frac{\partial L}{\partial U},\ W \leftarrow W - \alpha \frac{\partial L}{\partial W},\ V \leftarrow V - \alpha \frac{\partial L}{\partial V}$

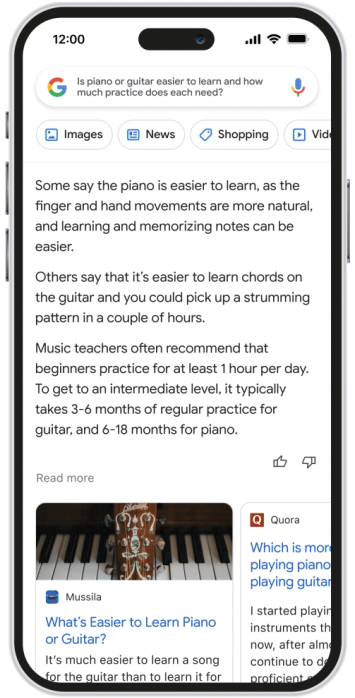
More about RNN

- Backpropagation Through Time https://dennybritz.com/posts/wildml/recurrent-neural-networks-tutorial-part-3/
- Vanishing Gradients and LSTM https://colah.github.io/posts/2015-08-Understanding-LSTMs/
- Sequence-to-Sequence Model (Seq2Seq) https://www.tensorflow.org/text/tutorials/nmt_with_attention

State of the art techniques

- Transformer (the technique behind ChatGPT and Gemini)

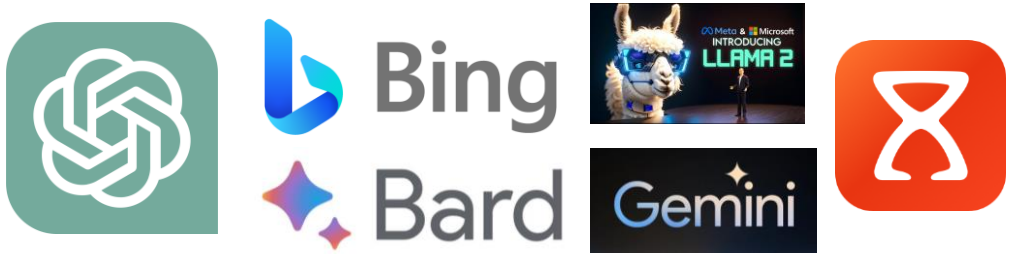




Reference: https://arxiv.org/abs/2304.13712

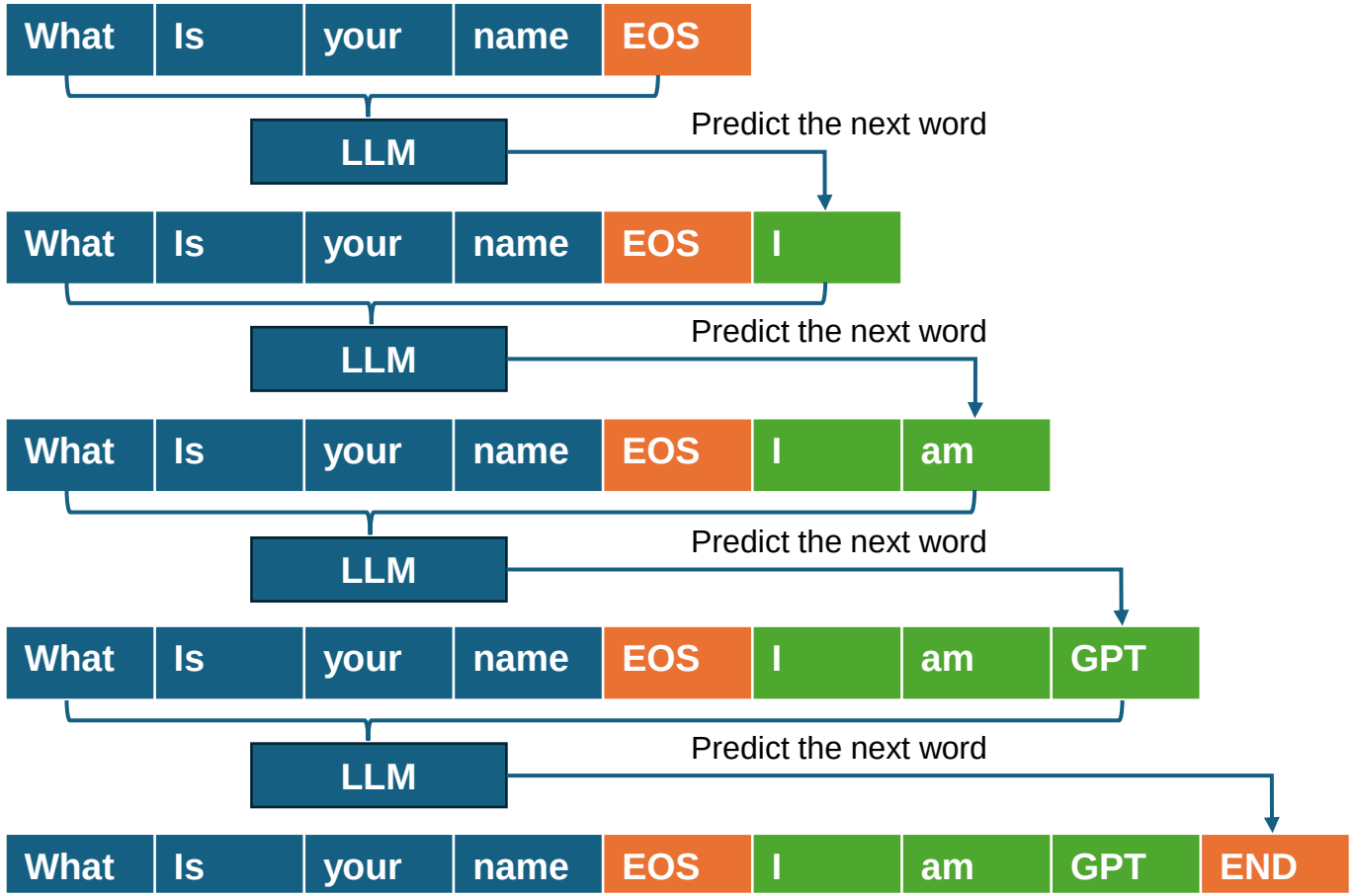
LLM as a next word predictor

Large language models (LLMs) perform incredibly good

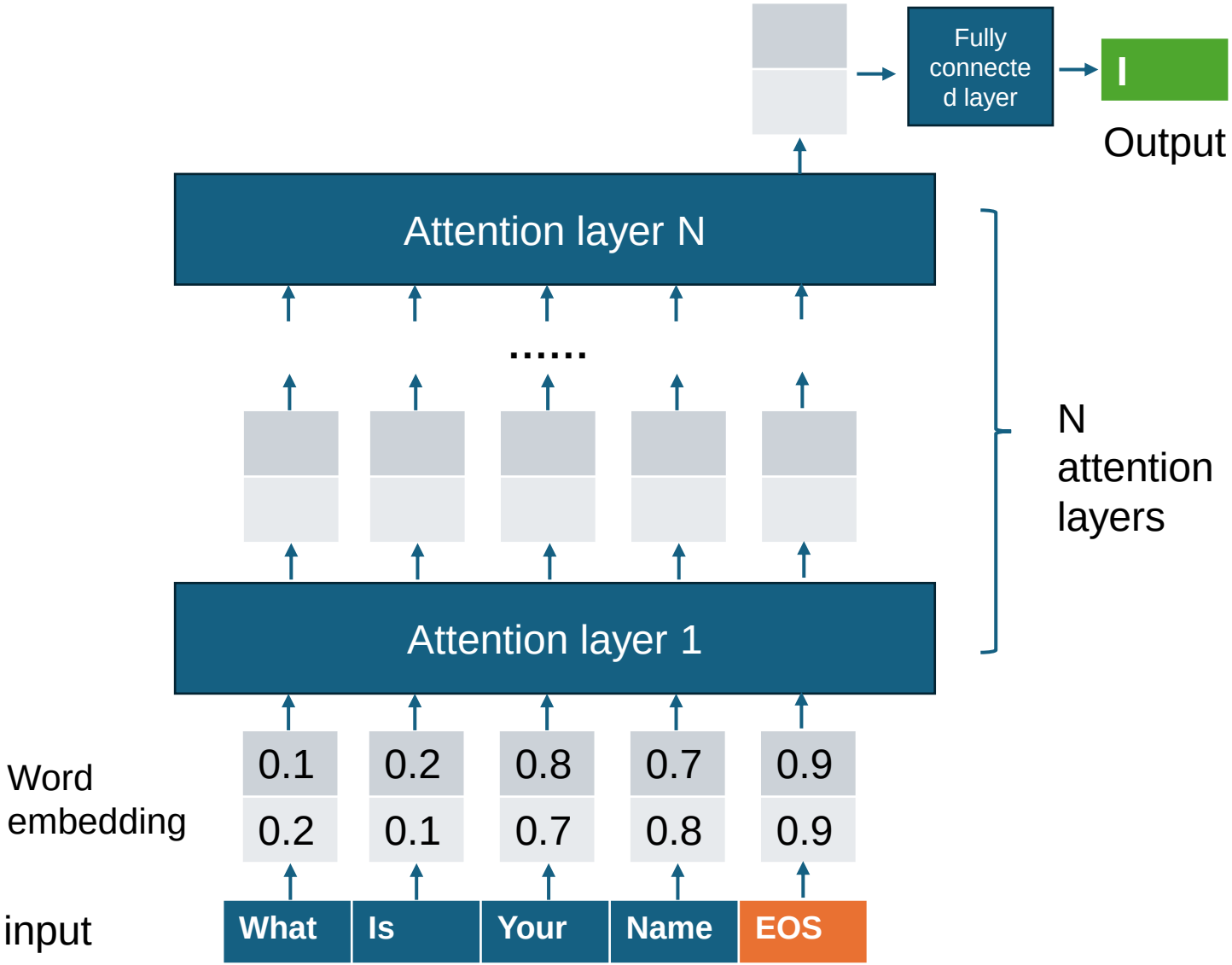


But their overall workflow (Transformer) is surprisingly simple

“just adding one word at a time”



Transformer at a glance



“Attention Is All You Need”

Attention Is All You Need

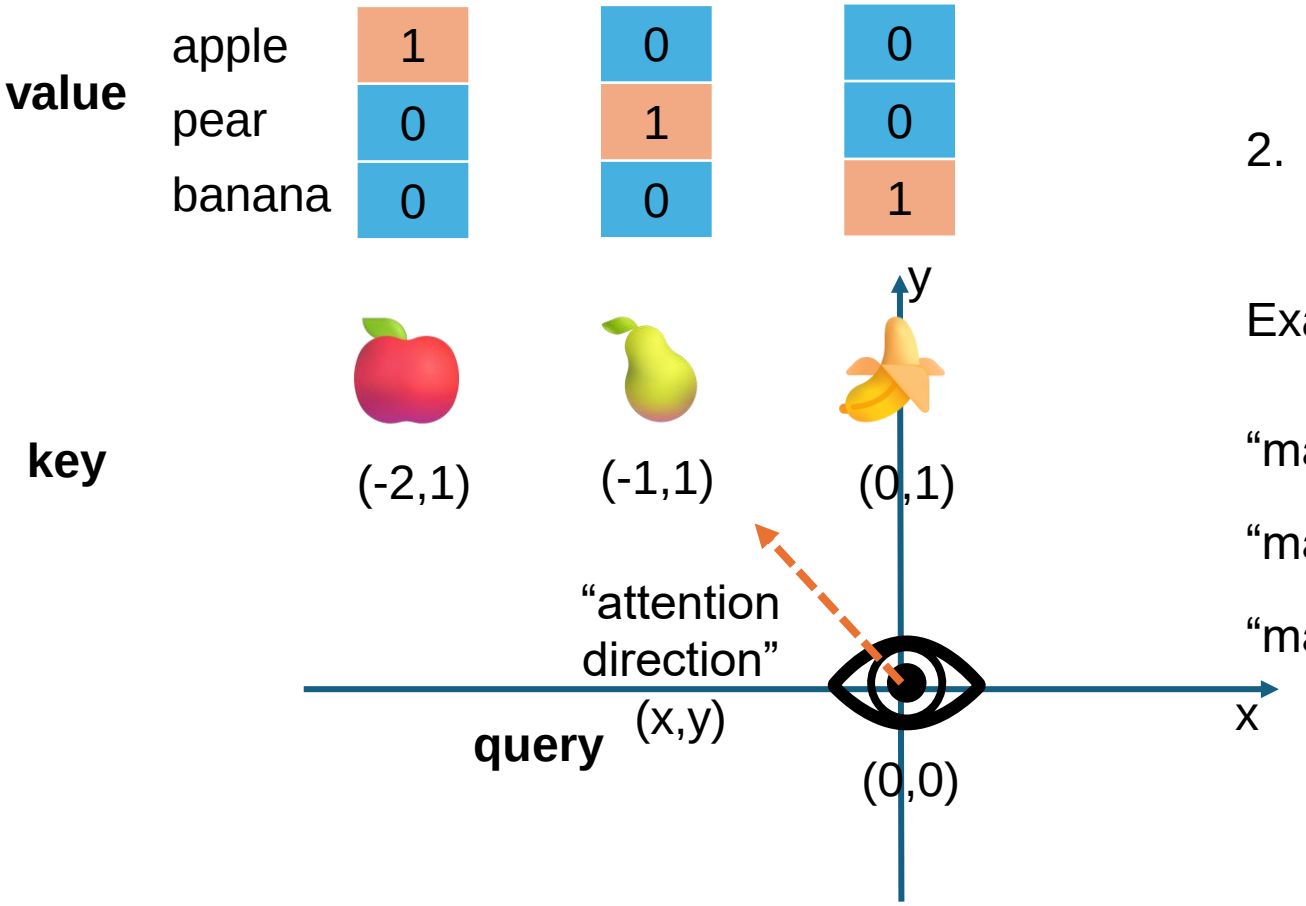
Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

https://arxiv.org/abs/1706.0376

Attention mechanism



Attention step:

1. Compute the “matching degree” or “similarity” between the attention direction (query) and the direction of items (key) – here we simply use $\cos(\theta)$
2. Compute the weight sum of each item’s value, according to the “matching degree” computed above

Example: when the attention direction is pointed to the pear

$$(x, y) = (-1,1)$$

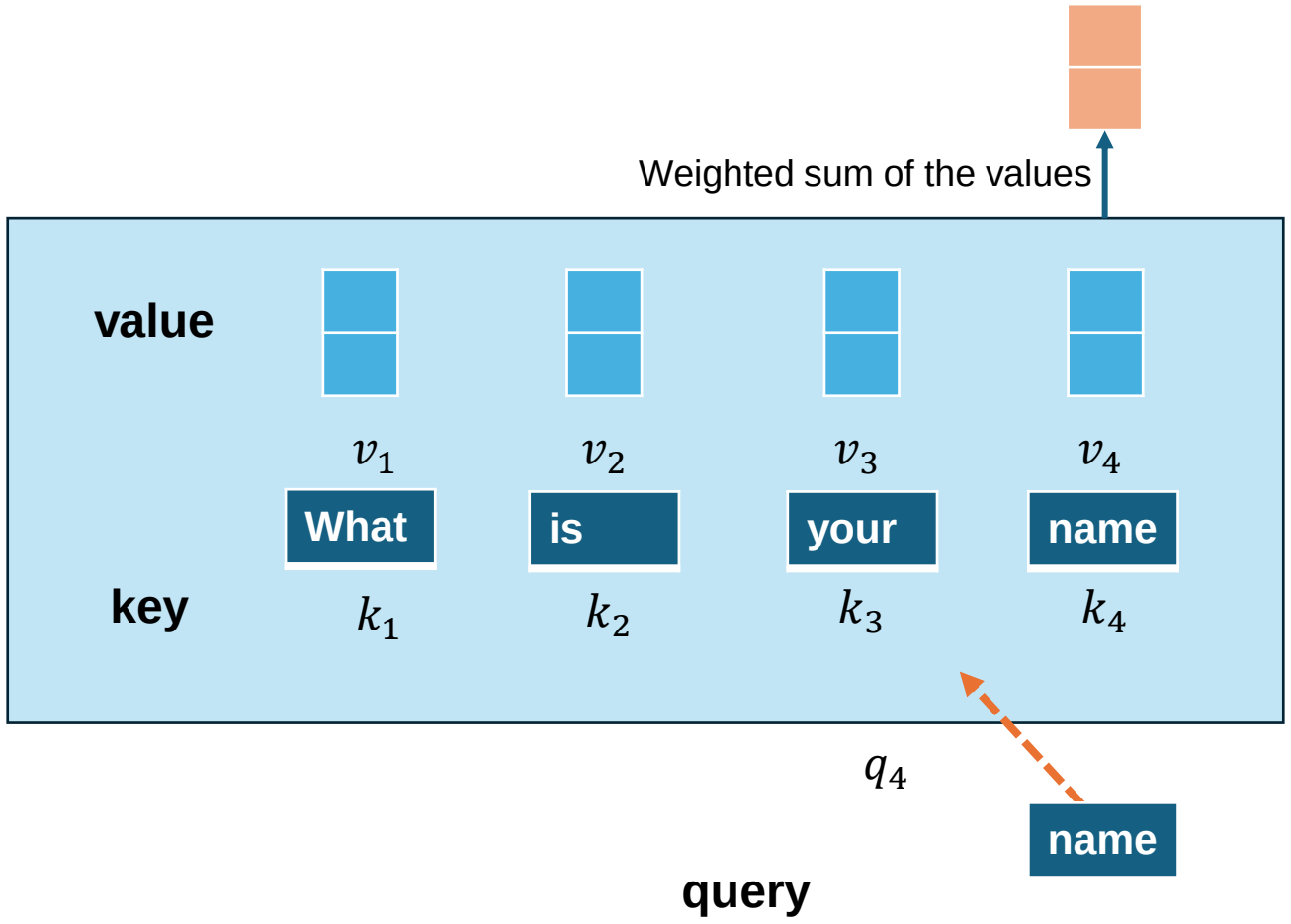
“matching degree” with the apple = $\frac{(-2,1)\times(-1,1)}{|(-2,1)||(-1,1)|} \approx 0.949$

“matching degree” with the pear = $\frac{(-1,1)\times(-1,1)}{|(-1,1)||(-1,1)|} = 1$

“matching degree” with the banana = $\frac{(0,1)\times(-1,1)}{|(0,1)||(-1,1)|} \approx 0.707$

$$\begin{matrix}1\\0\\0\end{matrix} \times 0.949 + \begin{matrix}0\\1\\0\end{matrix} \times 1 + \begin{matrix}0\\0\\1\end{matrix} \times 0.707 = \begin{matrix}0.949\\1\\0.707\end{matrix}$$

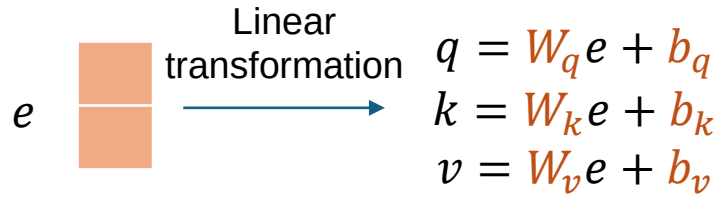
Attention mechanism



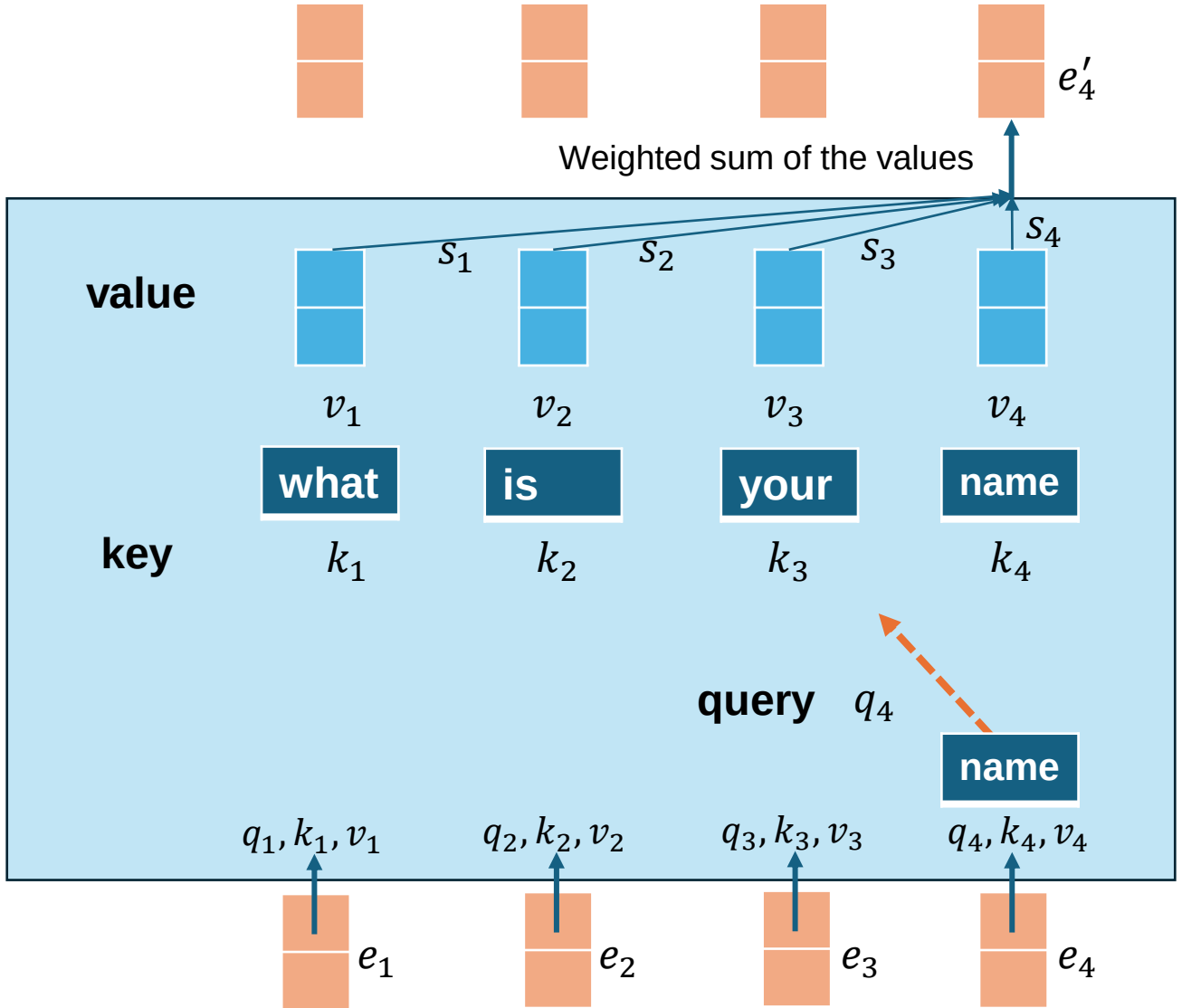
For words, we do exactly the same:

1. Compute the "matching degree" or "similarity" between the attention direction (query) and the direction of items (key)
2. Compute the weight sum of each item's value, according to the "matching degree" computed above

How to transform the word embedding of a word to the query, key and value vector?
– **This is what the model need to learn**



Attention mechanism



1. Transforming word embedding $e_1, \dots, e_t$ to query, key and value vectors

$$q_i = W_q e_i + b_q$$
$$k_i = W_k e_i + b_k$$
$$v_i = W_v e_i + b_v$$
$$i = 1, \dots, t$$

2. Compute "matching degree" s_i between the current word's query vector q_t and previous words' key vector $k_1 \cdots k_t$, and normalize it with softmax function

$$s_i = \text{sim}\,(q, k_i),\ i = 1, \dots, t$$
$$s_1, \dots, s_t \leftarrow \text{Softmax}(s_1, \dots, s_t)$$

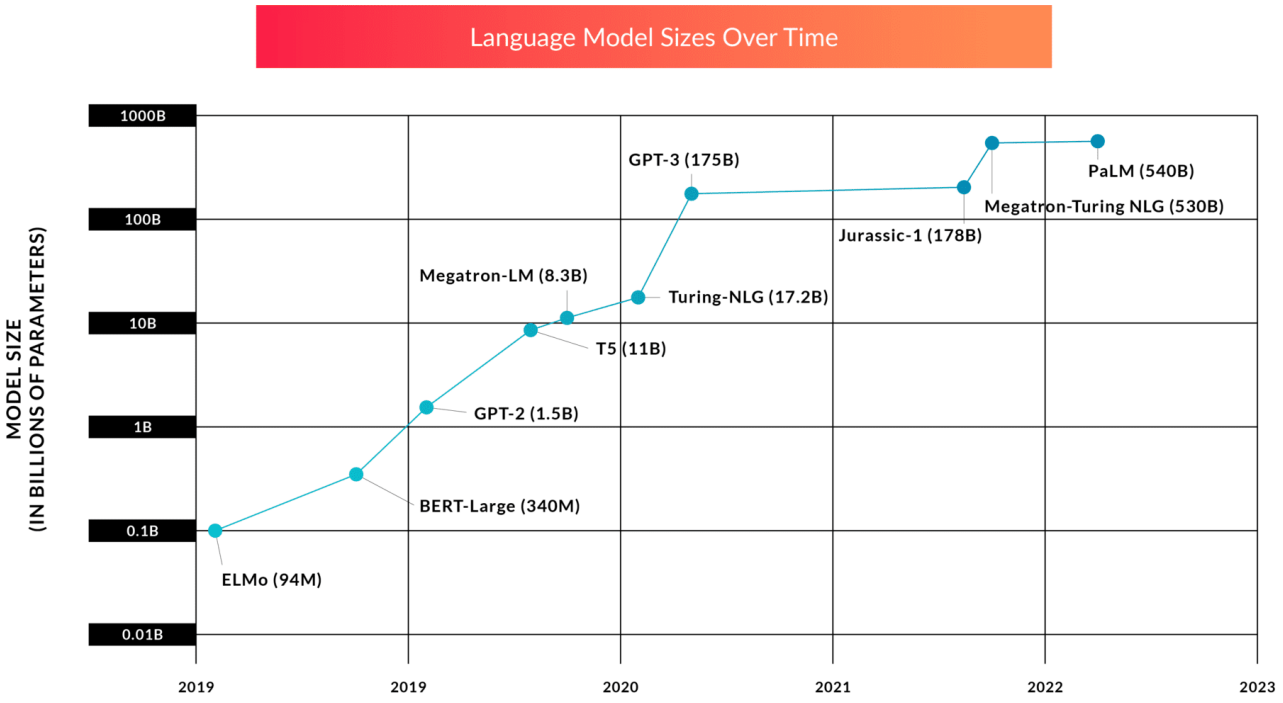
Here the $\text{sim}(\cdot,\cdot)$ function can be as simple as a dot product

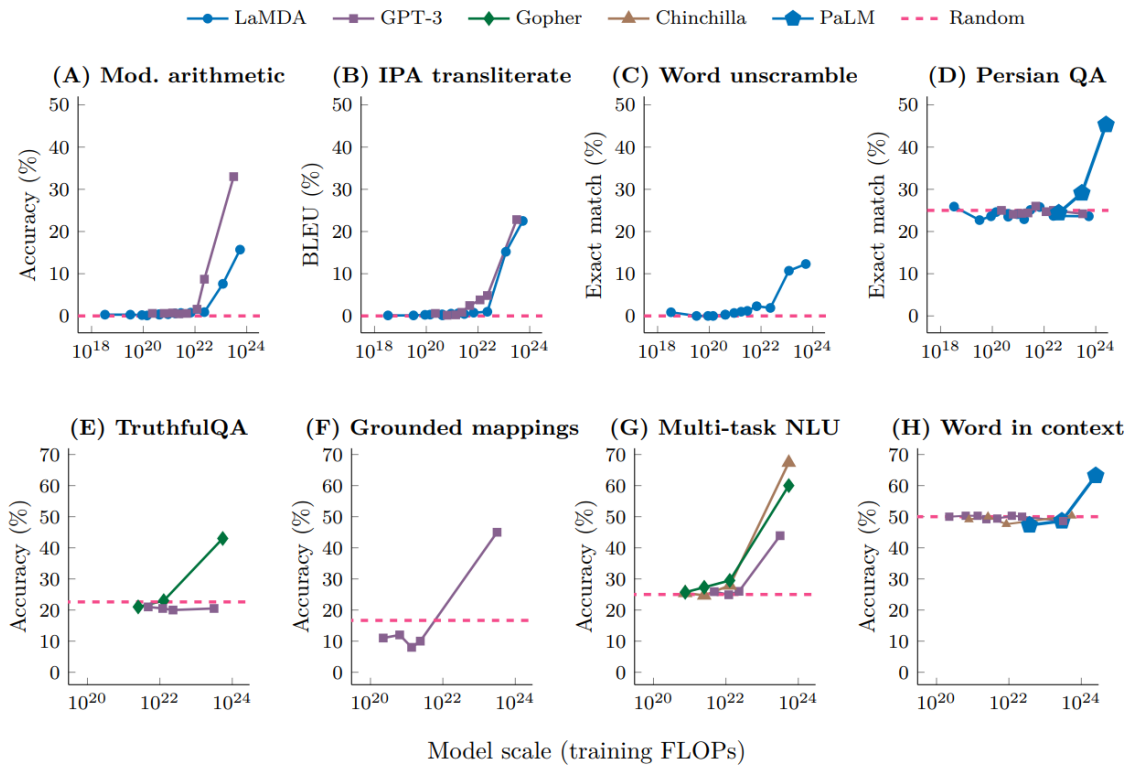
3. Compute the weight sum of the value vectors, according to s_i

$$e_t' = v_1 s_1 + \cdots + v_t s_t$$

The advance of recent language models

1. With the development of computational power, data, model and training technique, the trained language model becomes larger and larger – from “language model” to “large language model”
   We find that when the scale of the model exceeds certain “critical point”, new abilities emerge.





Reference: https://cmte.ieee.org/futuredirections/2023/04/24/how-much-bigger-can-should-llms-become/
Emergent Abilities of Large Language Models https://arxiv.org/abs/2206.07682

Emergent abilities

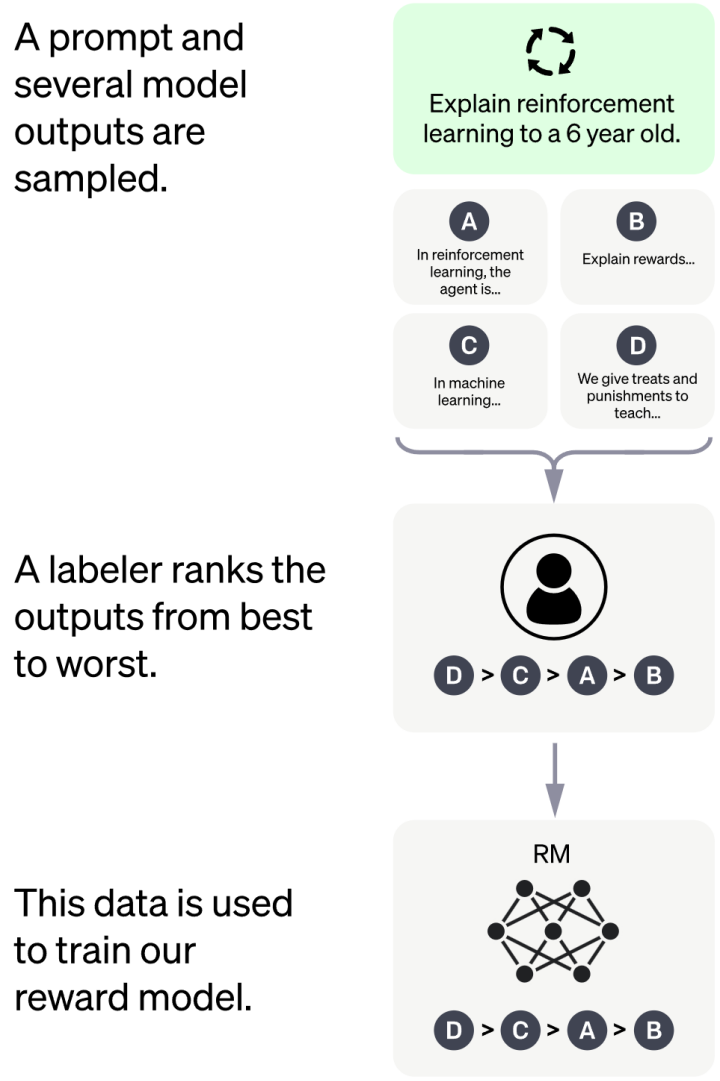
Model	Scale	Emergent abilities
BERT/GPT （2018）	12-layer Transformer 7000 books (4.6GB) 117 million parameters	**Pre-training** To accomplish certain tasks (e.g., translation), we just need to fine-tuning a pre-trained model, instead of training from scratch.
GPT-2（2019）	Same architecture Extend training data to 40GB (top articles in Reddit) 1.5 billion parameters	**Multi-task** The trained model can achieve good results on multiple language tasks, without any fine-tuning or parameter update.
GPT-3（2020） Codex（2021） GPT-3.5（2022）	Training data extended to 600GB Parameters extended to 175 billion Including programming code in training data(Codex) Instruction fine-tuning and RLHF (GPT-3.5)	**In-Context Learning** The trained model can accomplish certain tasks via providing examples in natural language. “Please output the number of legs: 1 chick = 2 legs, 2 chicken = 4 legs, 3 chicken =” **Chain of Thought** The trained model can output steps via adding “Let’s think step by step” in the prompt

The advance of recent language models

2. Techniques for aligning model output with people's expectation

- Instruction fine-tuning
  - Use supervised "instruction-answer" data pairs to fine-tuning the pre-trained model
  - Data collection is expensive, performs poor on open questions (e.g., write a story about…)
- Reinforcement Learning from Human Feedback, RLHF
  - Train a "reward" model to score the model's generated output
  - Fine-tuning the pre-trained model via reinforcement learning, encouraging high-reward outputs and repress low-reward outputs.

Reference: https://openai.com/blog/chatgpt



More about Transformer

- Natural Language Processing with Deep Learning CS224N/Ling284 https://web.stanford.edu/class/cs224n/slides/cs224n-2023-lecture11-prompting-rlhf.pdf
- Neural machine translation with a Transformer and Keras, https://www.tensorflow.org/text/tutorials/transformer
- The Illustrated GPT-2 http://jalammar.github.io/illustrated-gpt2/

Thank you!

Xihan Li
Department of Computer Science,
University College London
xihan.li@cs.ucl.ac.uk
https://snowkylin.github.io

Feb 2024