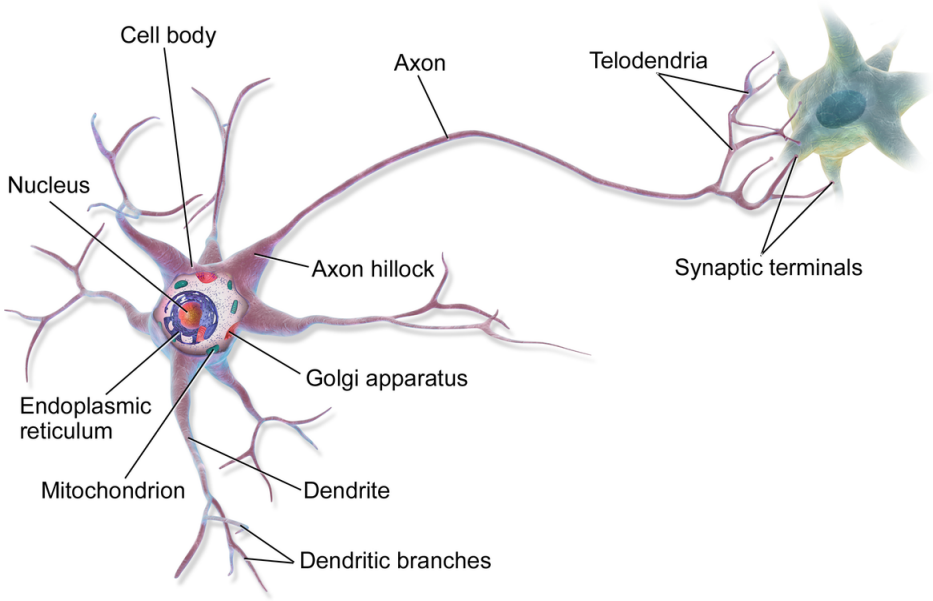
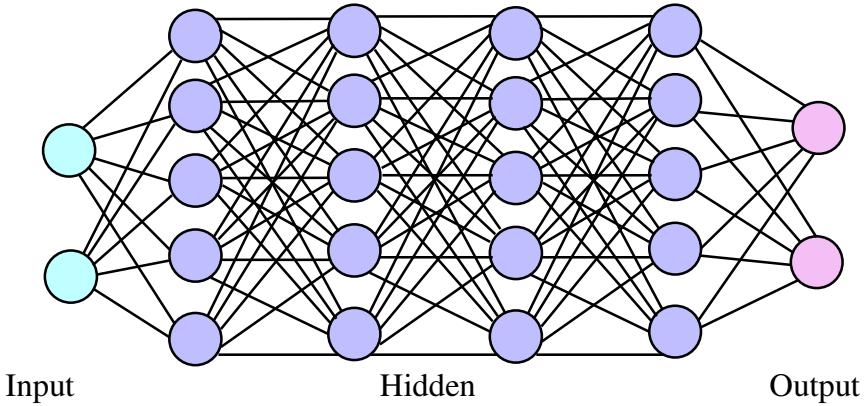
Machine Learning and Neural Networks

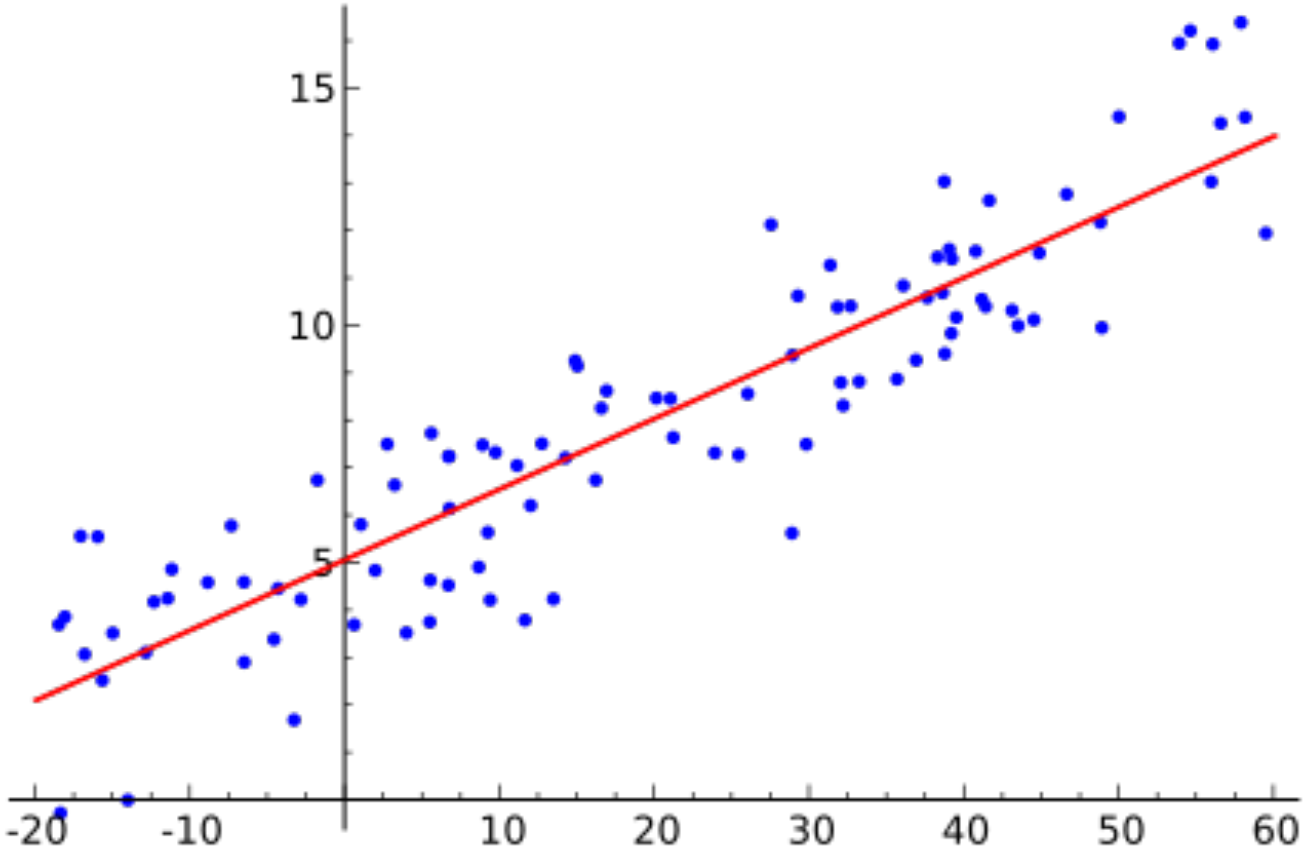
Introduction





Author: BruceBlaus, Wikipedia

Regression



Goals:

- Explain a set of observations with a “simple” model.
- Given this model, make predictions about new objects.

Regression basics

- Strategy: Model the **data generation process.**
- Usual model: Response variable $y \in \mathbb{R}$ (also called "target" t) is a noisy function of an input variable $\boldsymbol{x} \in \mathbb{R}^d$:

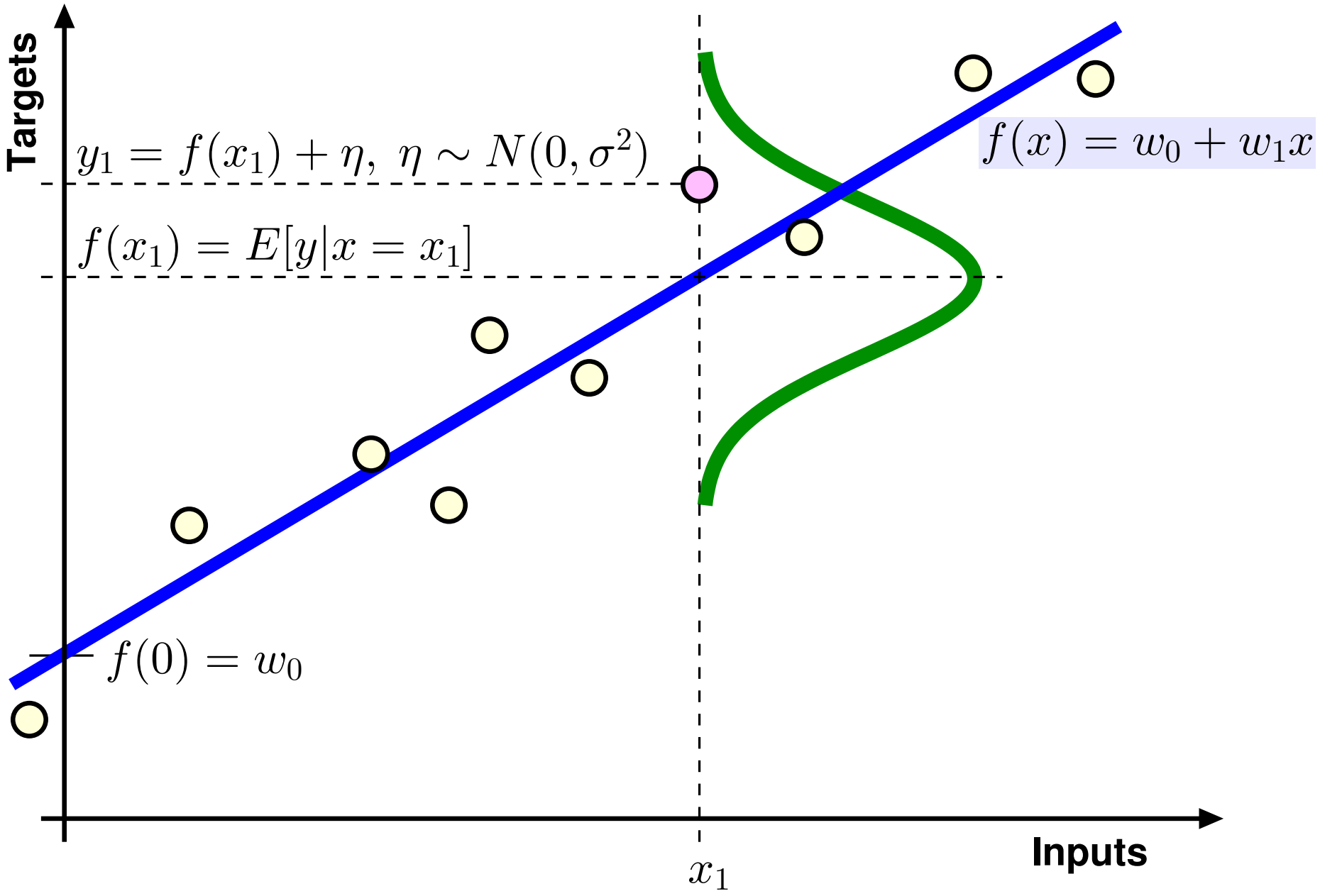
$$y = f(\boldsymbol{x}) + \eta.$$

- **Linear Gaussian regression:** $f(\boldsymbol{x}) = \boldsymbol{w}^t\boldsymbol{x} + \eta$, η has zero-mean Gaussian distribution with constant variance, $\eta \sim N(0, \sigma^2)$.
- Can equivalently be written as

$$p(y|\boldsymbol{x} = \boldsymbol{x}_k) = N(\mu(\boldsymbol{x}_k), \sigma^2), \text{ with } \mu(\boldsymbol{x}_k) = \boldsymbol{w}^t\boldsymbol{x}_k.$$

- In one dimension: $\mu(\boldsymbol{x}_k) = \boldsymbol{w}^t\boldsymbol{x}_k = w_0 + w_1 x_k$ and $\boldsymbol{x}_k = (1, x_k)^t$. w_0 is the **intercept** or bias term and w_1 is the **slope.**

Linear Gaussian regression



Least Squares and Maximum Likelihood

- Goal: Fit n input-target pairs $(\boldsymbol{x}_i, y_i)$ to a model that has $d+1$ parameters $w_j,\ j = 0, \ldots, d$.
- Notation: $\boldsymbol{x}, y$ are random variables (RV), $(\boldsymbol{x}_i, y_i)$ is one sample. Augmented $\boldsymbol{x}_i \leftarrow (1, \boldsymbol{x}_i) \rightsquigarrow w_0$ is the intercept.
- We assume that the n targets y_i are independent and identically distributed (**iid**), given their locations $\boldsymbol{x}_i$ in the input space.
- **Linear model:** $y_i = \boldsymbol{w}^t \boldsymbol{x}_i + \eta_i, \quad \eta_i \sim N(0, \sigma^2)$.
- Model predicts a linear relationship between the **conditional expectation of targets** and **inputs**:

$$E[y|\boldsymbol{x} = \boldsymbol{x}_i] = E[\boldsymbol{w}^t \boldsymbol{x}_i + \eta_i] = E[\boldsymbol{w}^t \boldsymbol{x}_i] + \underbrace{E[\eta_i]}_{0}$$

$$= \mu_i(\boldsymbol{x}_i) = \boldsymbol{w}^t \boldsymbol{x}_i = f(\boldsymbol{x}_i; \boldsymbol{w}).$$

Regression function = conditional expectation.

LS and Maximum Likelihood

- **Likelihood function:** conditional probability of all observed y_i given their explanation, treated as a function of $\boldsymbol{w}$:

$$\mathcal{L}(\boldsymbol{w}) \propto \prod_i p(y_i|\boldsymbol{x}_i, \boldsymbol{w}) \propto \prod_i \exp\left[-\frac{1}{2\sigma^2}(y_i - \overbrace{\boldsymbol{w}^t\boldsymbol{x}_i}^{\mu_i})^2\right]$$

- **Maximizing $\mathcal{L}$ = finding model that best explains observations:**

$$\begin{aligned}\hat{\boldsymbol{w}} &= \arg\max_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}) = \arg\min_{\boldsymbol{w}}[-\mathcal{L}(\boldsymbol{w})] = \arg\min_{\boldsymbol{w}}[-\log(\mathcal{L}(\boldsymbol{w}))] \\ &= \arg\min_{\boldsymbol{w}} \sum_i (y_i - \boldsymbol{w}^t\boldsymbol{x}_i)^2\end{aligned}$$

Least-squares fit = ML solution under Gaussian error model.

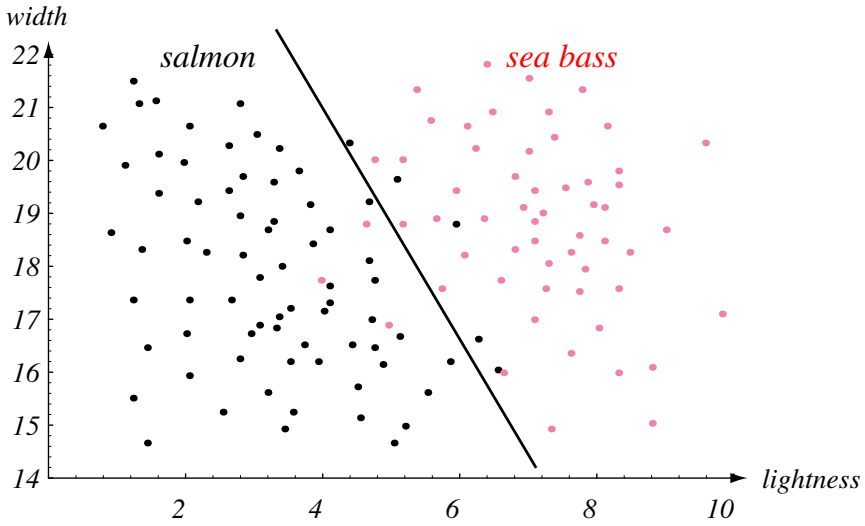
- $\hat{w}_{\text{MLE}}$ minimizes the **residual sum of squares**

$$RSS(\boldsymbol{w}) = \sum_{i=1}^{n} r_i^2 = \sum_{i=1}^{n} [y_i - f(\boldsymbol{x}_i; \boldsymbol{w})]^2 = \|\boldsymbol{y} - X\boldsymbol{w}\|^2.$$

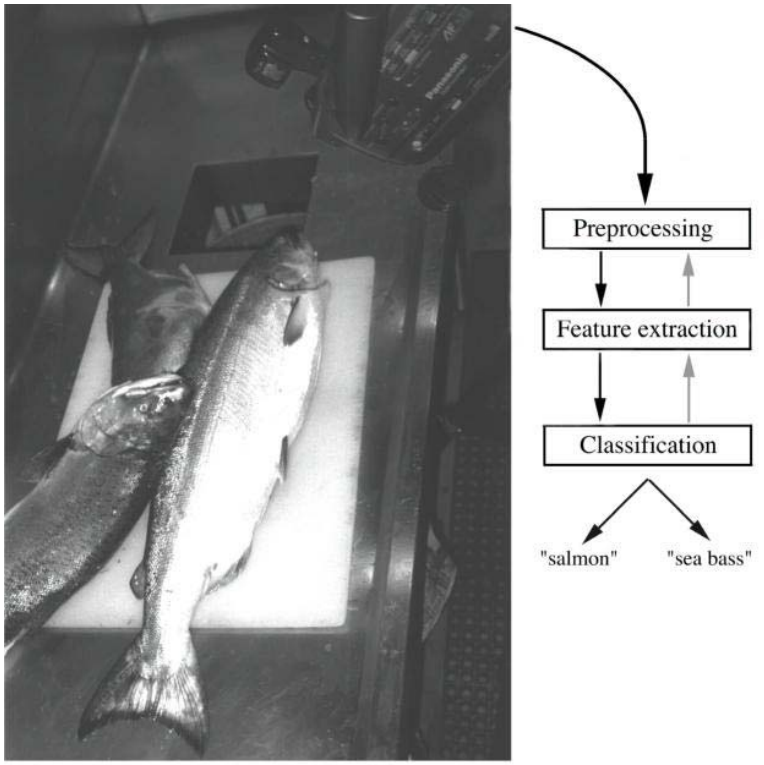
Classification

Classification: Find **class boundaries** based on training data $\{(\boldsymbol{x}_1, t_1), \ldots, (\boldsymbol{x}_n, t_n)\}$. Use boundaries to classify new items $\boldsymbol{x}_*$. Here, the target t_i is a discrete class indicator (or "label").

Example: Fish-packing plant wants to automate the process of sorting fish on conveyor belt using optical sensing.

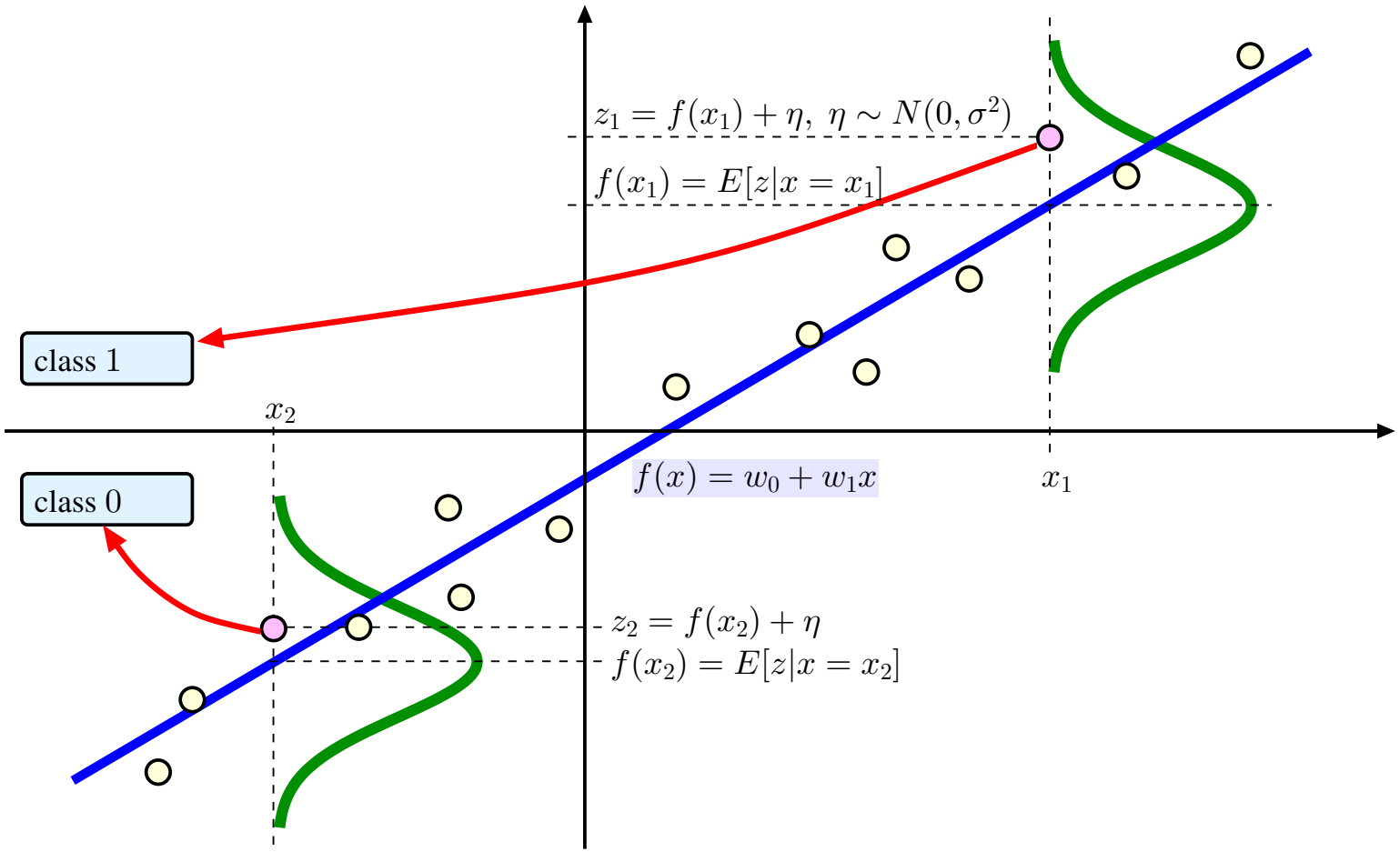


(Duda, Hart, Stork: *Pattern classification*. Wiley, 2001)



(Duda, Hart, Stork: *Pattern classification*. 2001)

Classification



Again a generative model: First generate latent $z \sim N(f(\boldsymbol{x}), \sigma^2)$, then choose t based on sign(z): $t = 1$, if $z > 0$, and $t = 0$ else.
Probit regression, a specific **generalized linear model.**

Basis functions and additive models

- Nonlinear generalization: Replacing $\boldsymbol{x}$ with some non-linear function of the inputs, $\boldsymbol{\phi}(\boldsymbol{x})$:

$$p(t|\boldsymbol{x}) = N(\boldsymbol{w}^t\boldsymbol{\phi}(\boldsymbol{x}), \sigma^2).$$

- Components of $\boldsymbol{\phi}$ might be interpreted as fixed basis functions $\boldsymbol{\phi}(\boldsymbol{x}) = \{g_0(\boldsymbol{x}), g_1(\boldsymbol{x}), \ldots, g_m(\boldsymbol{x})\}$, with $g_i(\boldsymbol{x}) : \mathbb{R}^d \mapsto \mathbb{R}$. Intercept: $g_0(\boldsymbol{x}) = 1$. Example: Polynomial basis functions:

$$f(\boldsymbol{x}; \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2.$$

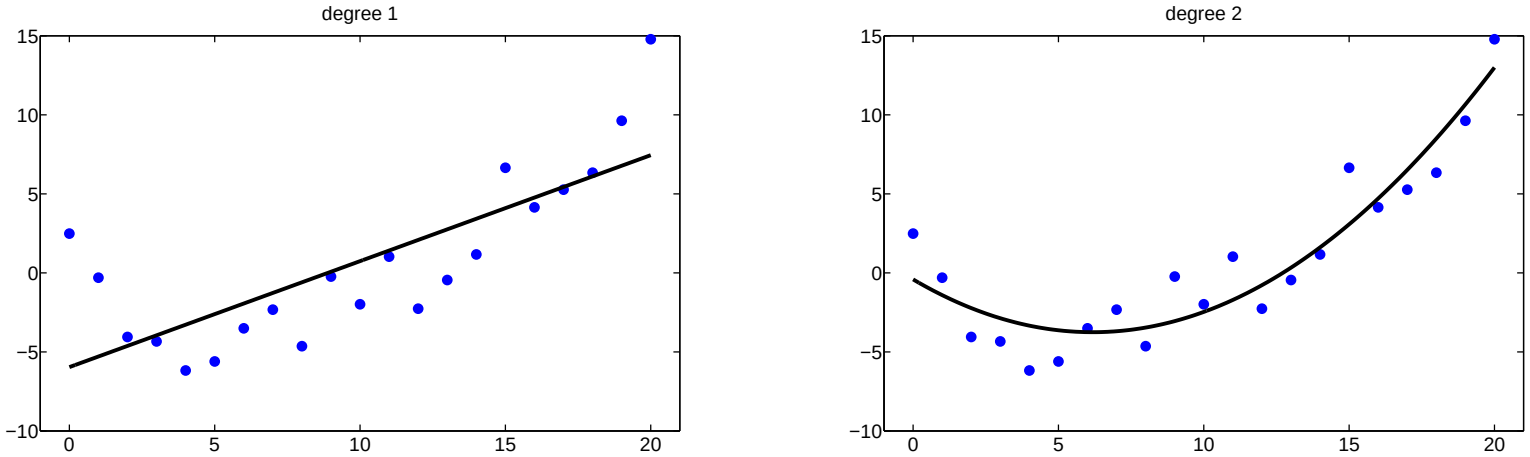
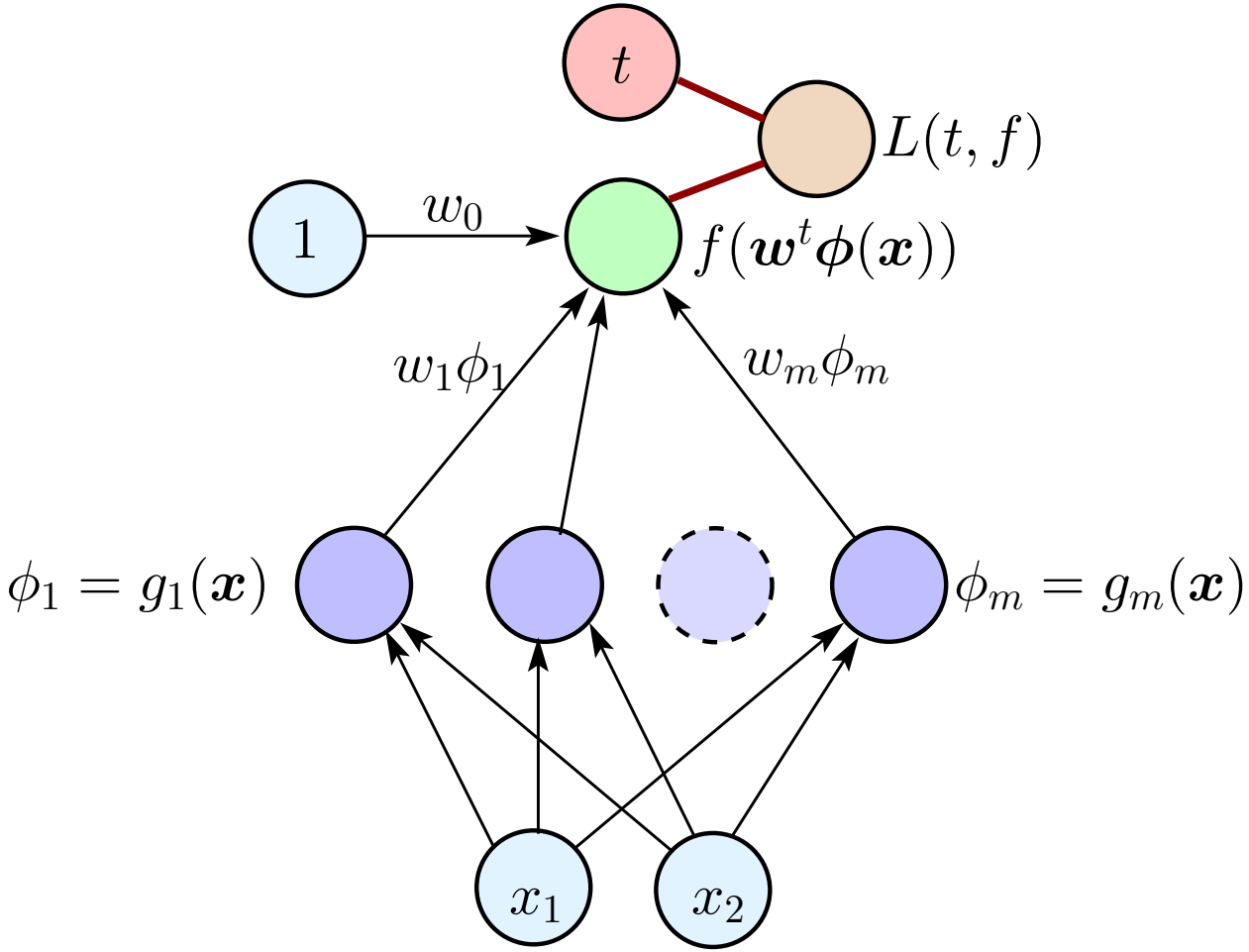


Fig 1.7 in K. Murphy: *Machine Learning*. MIT press, 2012

Additive models as graphs

Graphical representation in terms of **units** and **weights**.



This is “almost” an **artificial neural network.**

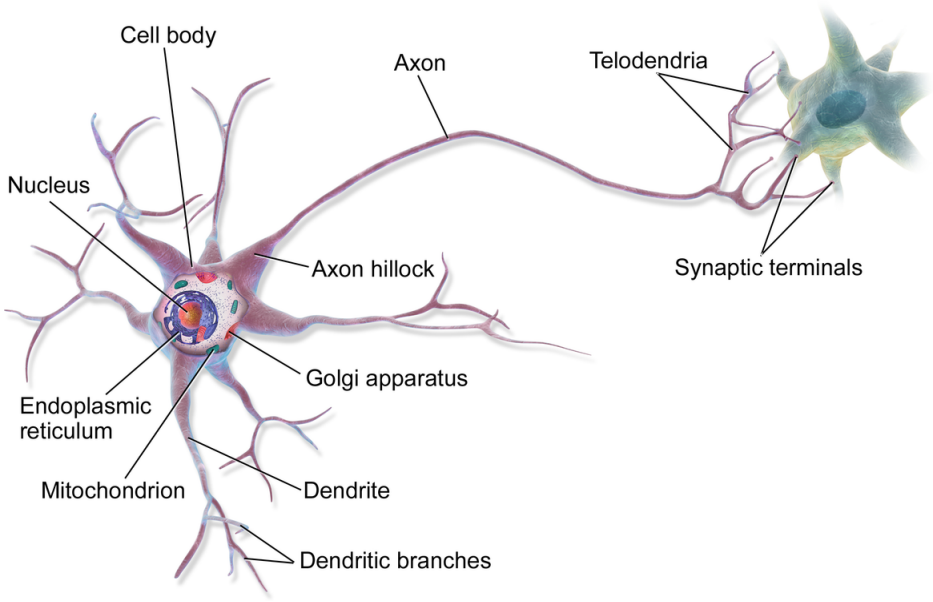
There, the basis functions also have adjustable parameters.

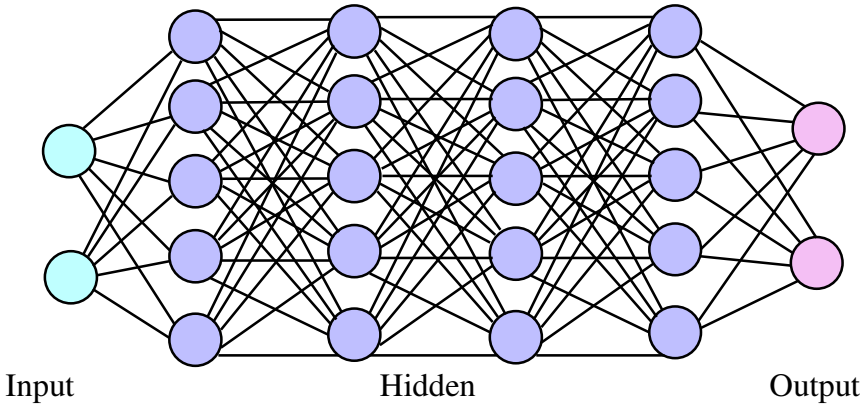
Summary

- Model the data generation process with a parametrized model, such as *"target = function of $\boldsymbol{x}$ plus noise"*, $t = f(\boldsymbol{x}; \boldsymbol{w}) + \eta$
- Infer parameters $\boldsymbol{w}$:
  - Maximize likelihood $\mathcal{L} \rightsquigarrow$ point estimate $\hat{\boldsymbol{w}} = \boldsymbol{w}_{\text{MLE}}$, equivalent to minimizing a loss function $L(t, f(\boldsymbol{x}; \boldsymbol{w})) = -\log \mathcal{L}$,
  - or estimate a probability density over parameters $p(\boldsymbol{w}|\boldsymbol{x})$ $\rightsquigarrow$ Bayesian inference.
- Use model(s) to predict the target t_* for a new $\boldsymbol{x}_*$
- Most neural networks basically follow the MLE idea:
  - they implement the **forward path** $\boldsymbol{x} \to f(\boldsymbol{x}; \boldsymbol{w})$,
  - mapped values $f(\boldsymbol{x})$ are compared with targets t via a loss function,
  - parameters ("weights") $\boldsymbol{w}$ are trained by minimizing the loss.

Machine Learning and Neural Networks

Neural networks: Biological and artificial





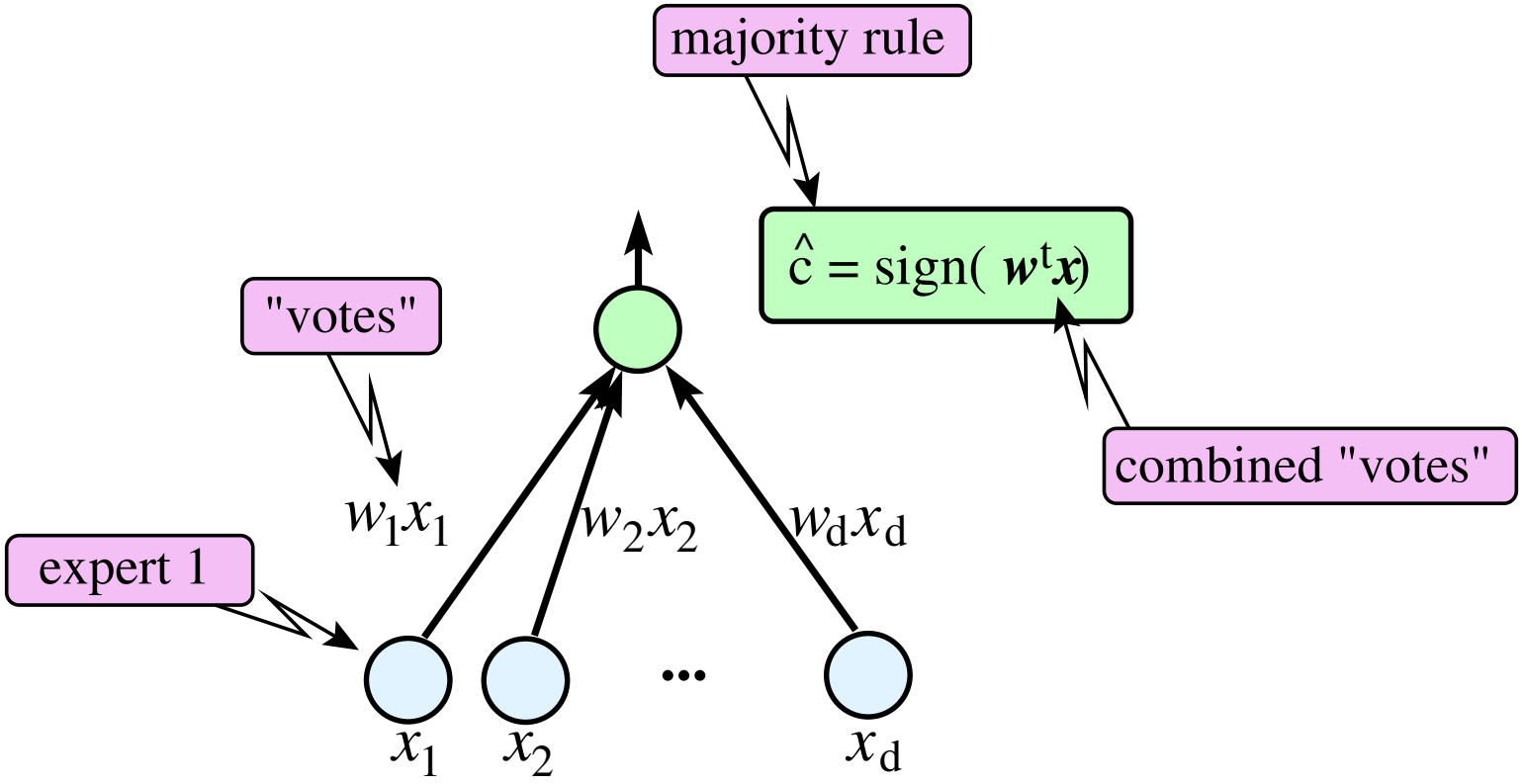
Author: BruceBlaus, Wikipedia

Linear classifier

We can understand the simple **linear classifier**

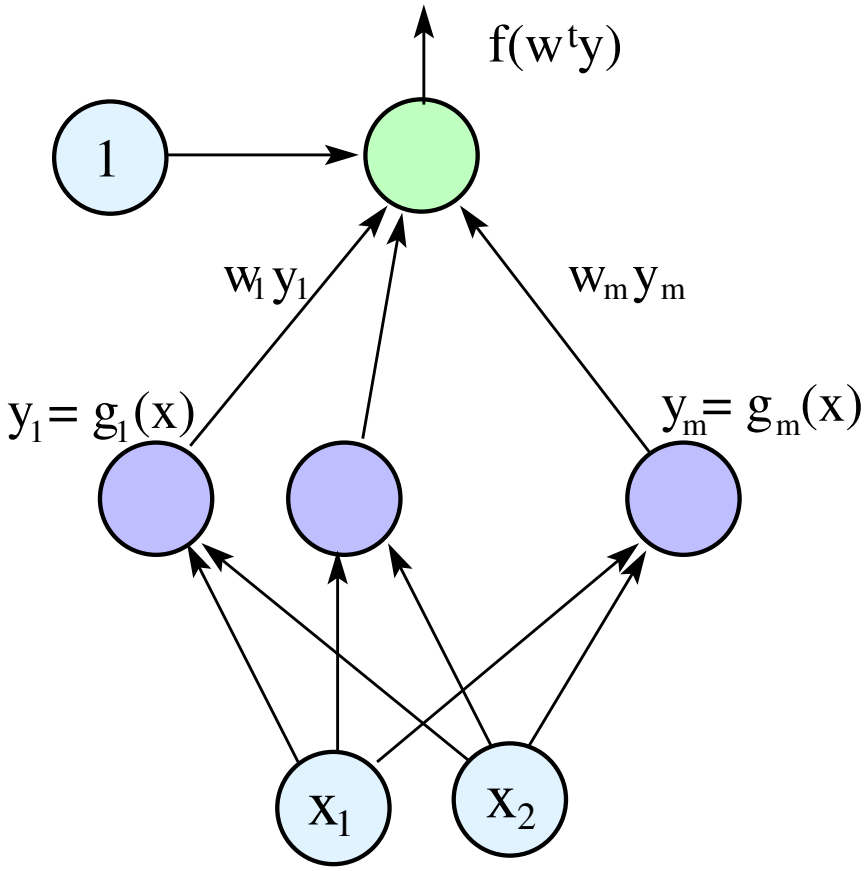
$$\hat{c} = \text{sign}(\boldsymbol{w}^t \boldsymbol{x}) = \text{sign}(w_1 x_1 + \cdots + w_d x_d)$$

as a way of **combining expert opinion**



Additive models cont'd

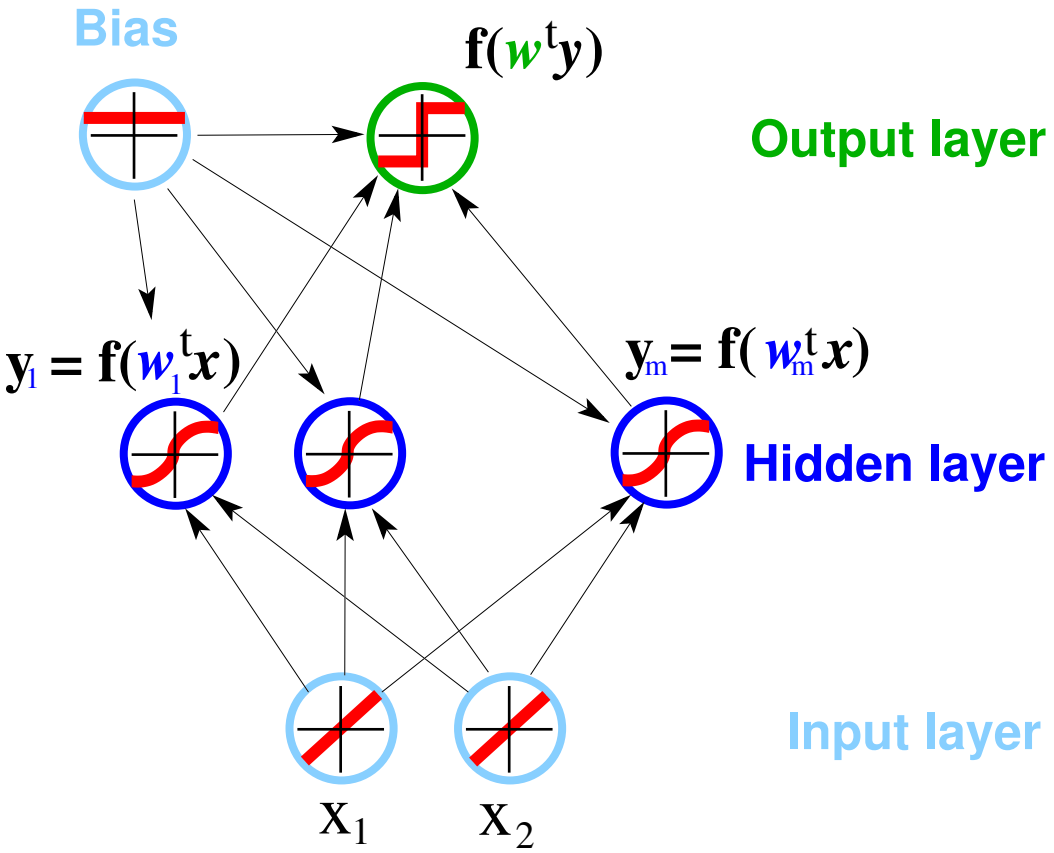
View additive models graphically in terms of **units** and **weights**.



In **neural networks** the basis functions themselves have adjustable parameters.

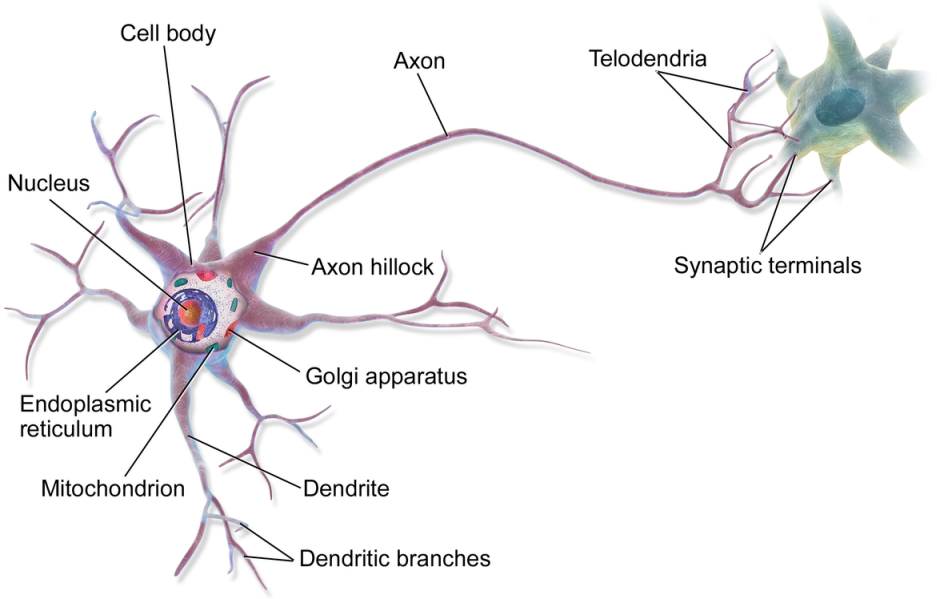
From Additive Models to Multilayer Networks

Separate units ($\rightsquigarrow$ artificial **neurons**) with **activation** f(net activation), where **net activation** is the weighted sum of all inputs, $\text{net}_j = \boldsymbol{w}_j^t \boldsymbol{x}$.



Biological neural networks

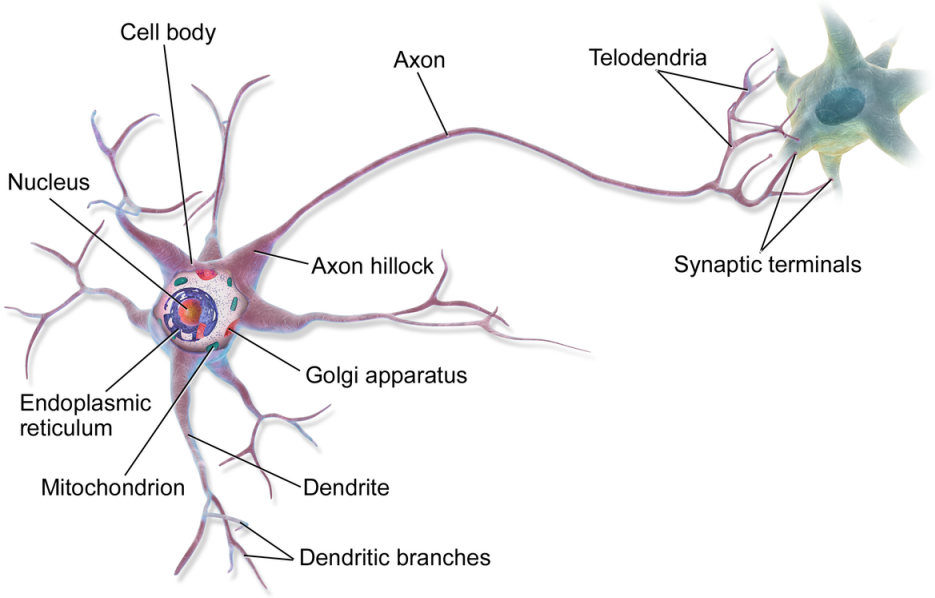
- Neurons (nerve cells): core components of **brain** and **spinal cord**. Information processing via **electrical and chemical signals**.
- Connected neurons form **neural networks**.
- Neurons have a cell body (**soma**), **dendrites**, and an **axon**.
- **Dendrites** are thin structures that arise from the soma, branching multiple times, forming a **dendritic tree**.
- **Dendritic tree collects input from other neurons.**



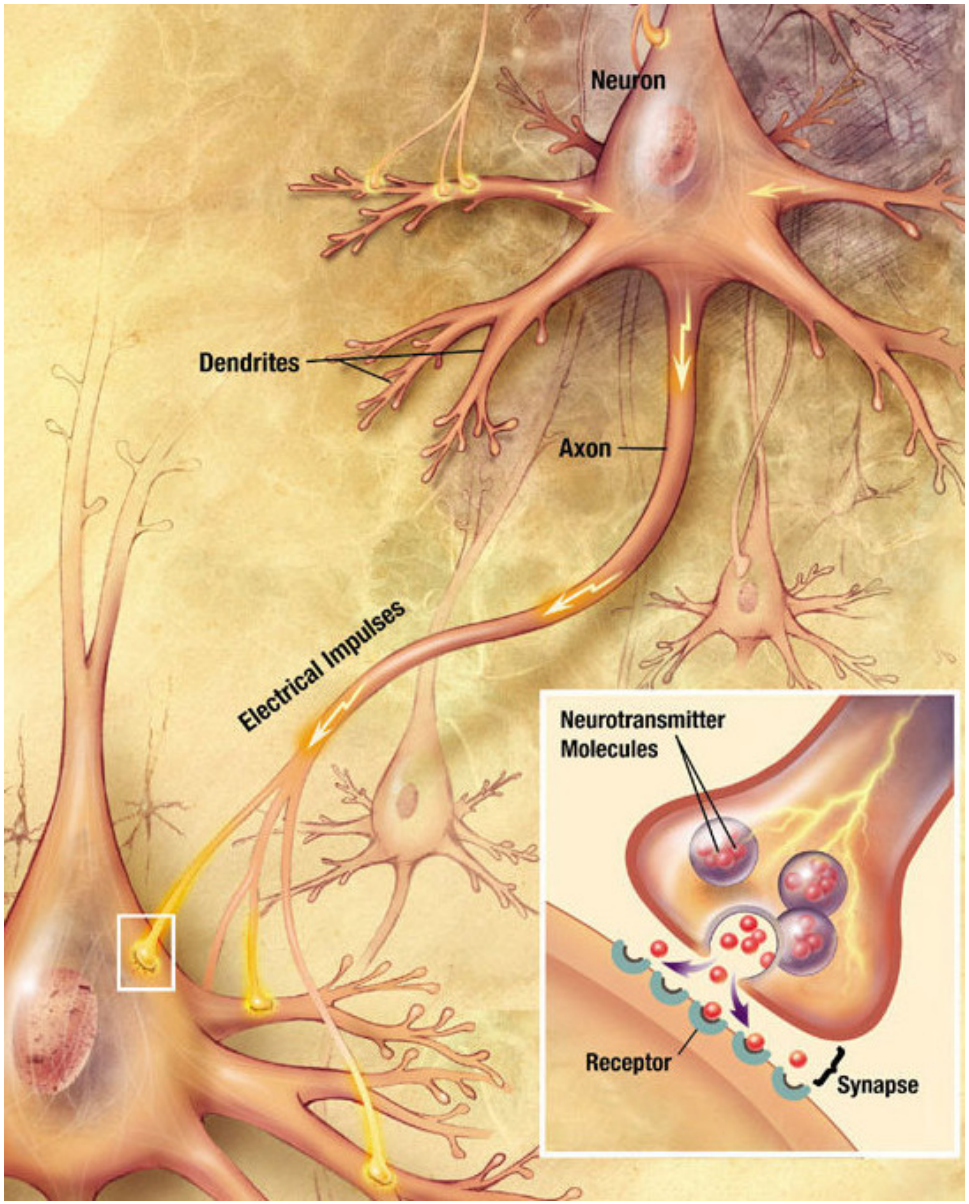
Author: BruceBlaus, Wikipedia

A typical cortical neuron

- **Axon:** cellular extension, contacts dendritic trees at **synapses.**
- **Spike of activity** in the axon
  $\rightsquigarrow$ charge injected into **post-synaptic neuron**
  $\rightsquigarrow$ chemical **transmitter molecules** released
  $\rightsquigarrow$ they **bind to receptor molecules** $\rightsquigarrow$ **in-/outflow** of ions.
- The effect of inputs is controlled by a **synaptic weight.**
- **Synaptic weights adapt $\rightsquigarrow$ whole network learns**

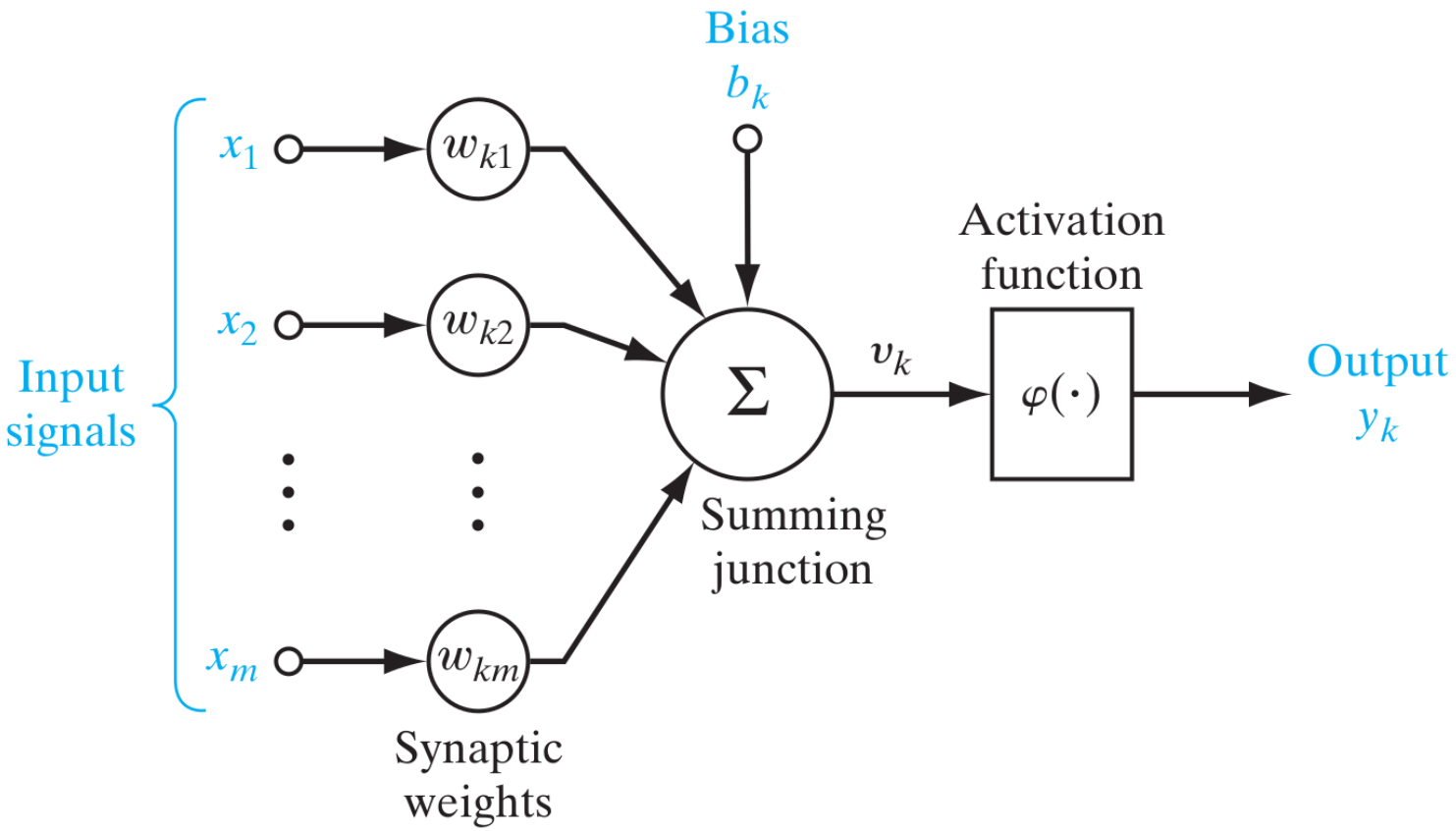


Author: BruceBlaus, Wikipedia



By user:Looie496 created file, US National Institutes of Health, National Institute on Aging created original - http://www.nia.nih.gov/alzheimers/publication/alzheimers-disease-unraveling-mystery/preface, Public Domain, https://commons.wikimedia.org/w/index.php?curid=8882110

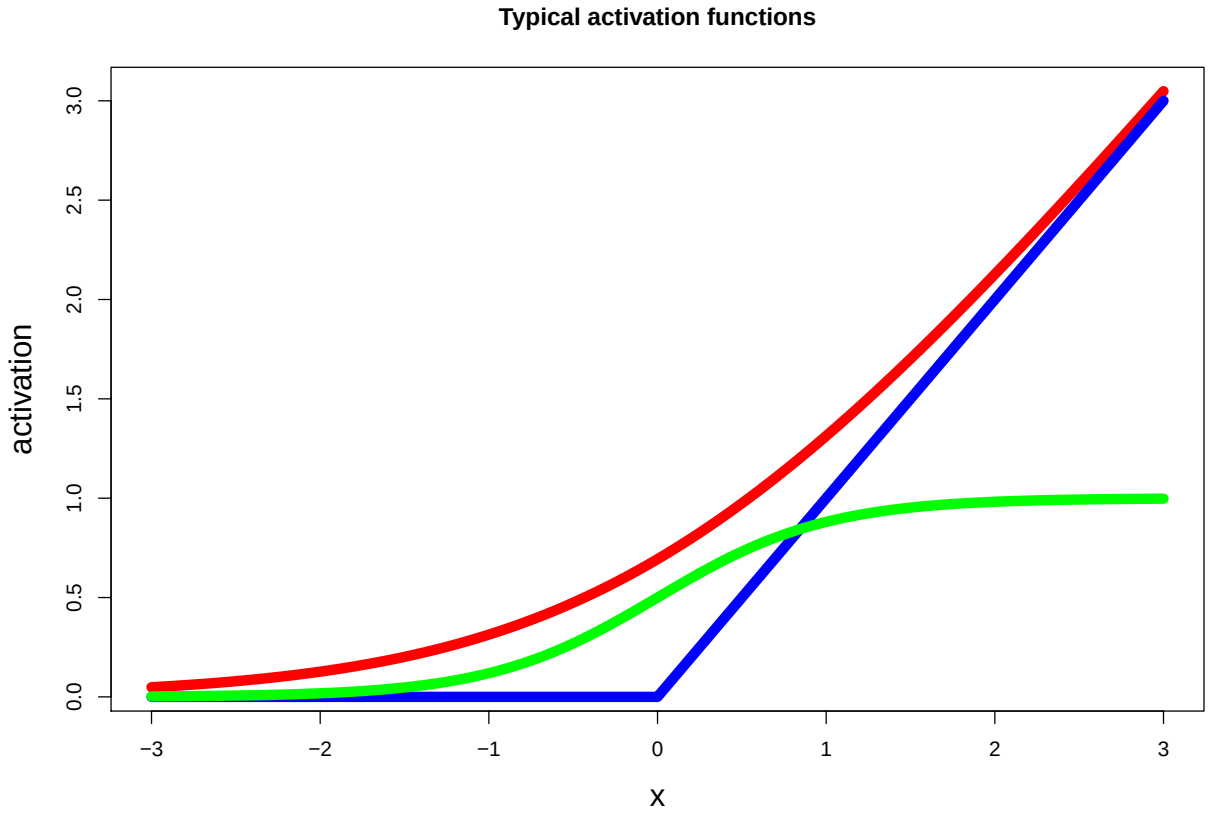
Idealized Model of a Neuron



from (Haykin, Neural Networks and Learning Machines, 2009)

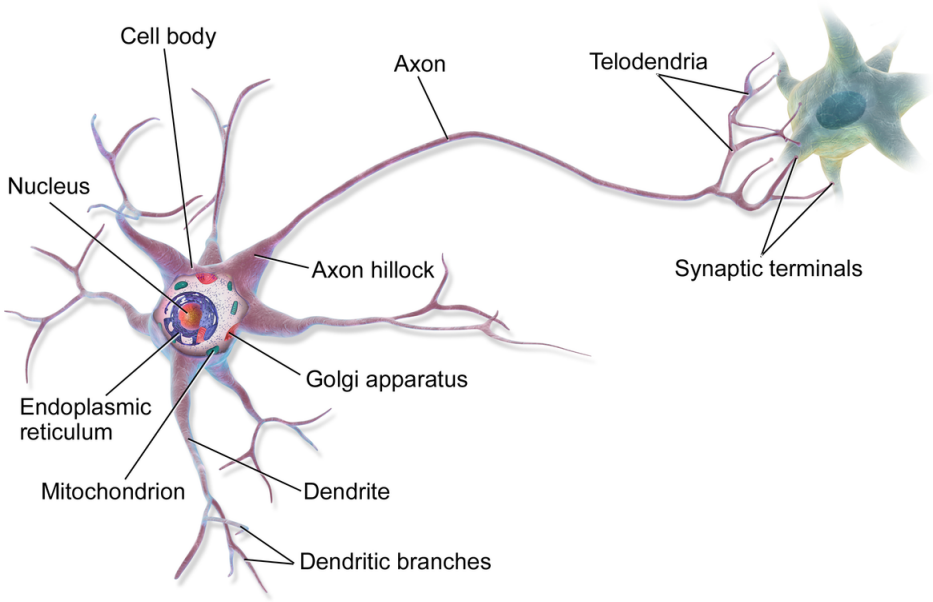
Hyperbolic tangent / Rectified / Softplus Neurons

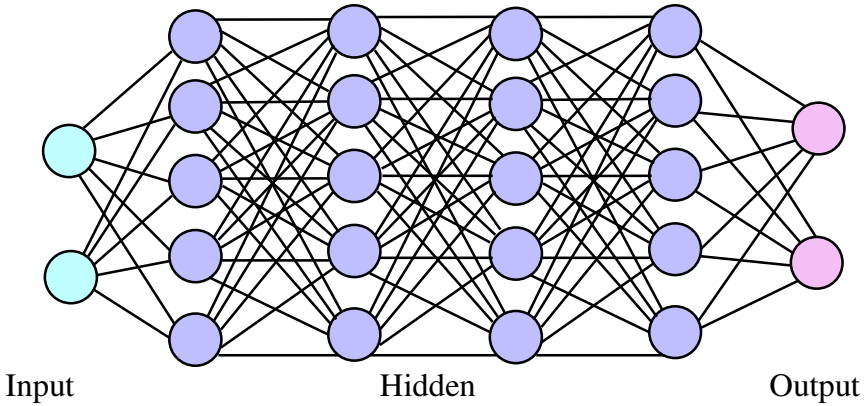
- "Classical" activations are **smooth and bounded**, such as **tanh.**
- In modern networks **unbounded** activations are more common, like **rectifiers** ("plus"): $f(x) = x^+ = \max(0, x)$ or **softplus** $f(x) = \log(1 + \exp(x))$.



Machine Learning and Neural Networks

Convolutional neural networks

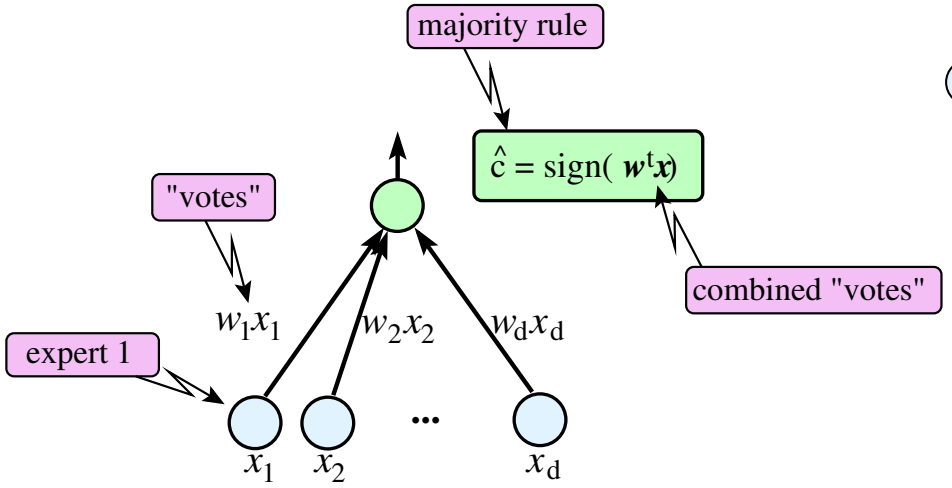




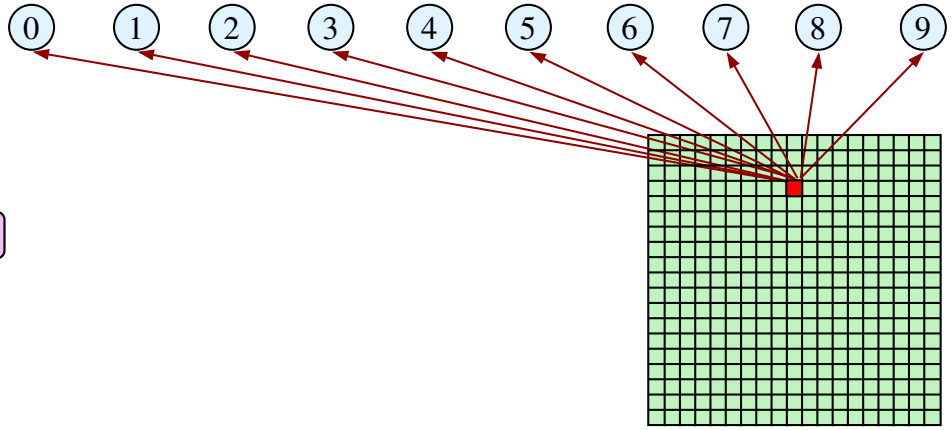
Author: BruceBlaus, Wikipedia

Simple NN for recognizing handwritten shapes

Two classes

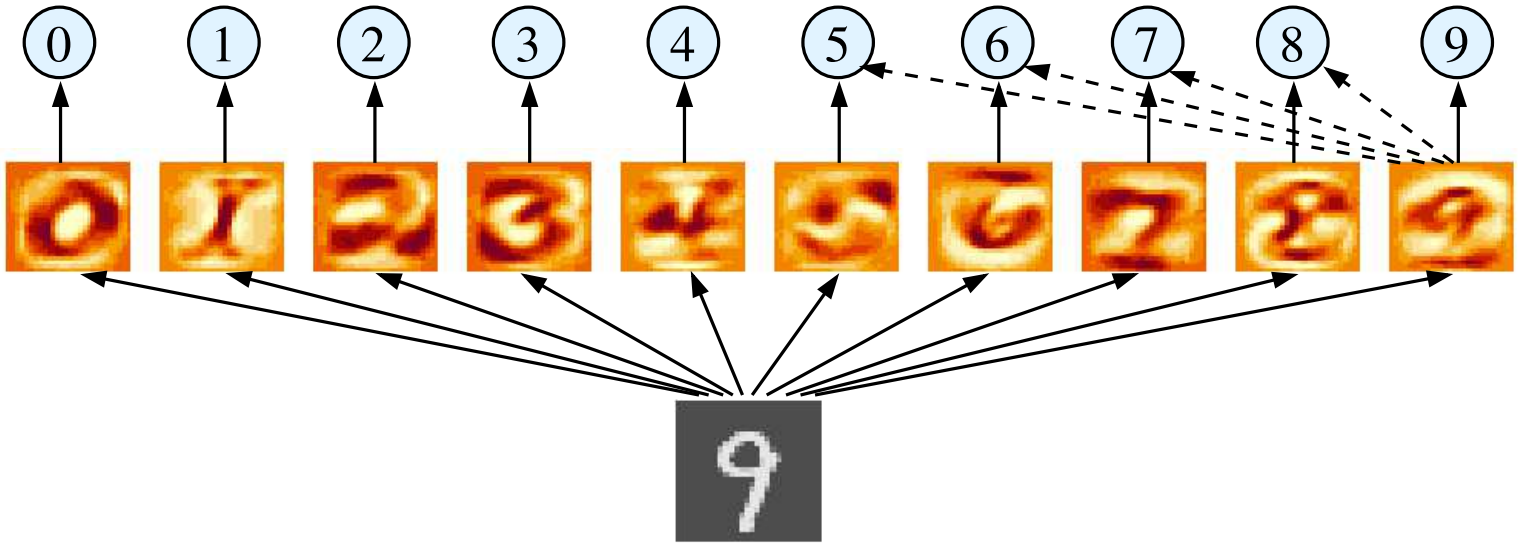


10 classes



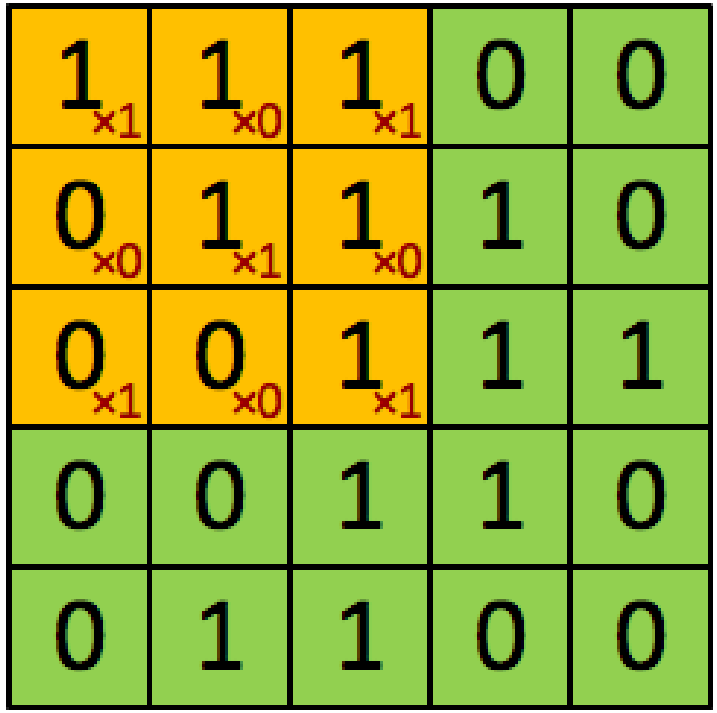
- Consider a neural network with **two layers** of neurons.
- Each pixel can vote for **several different shapes**.
- The shape that gets the **most votes** wins.

Why the simple NN is insufficient

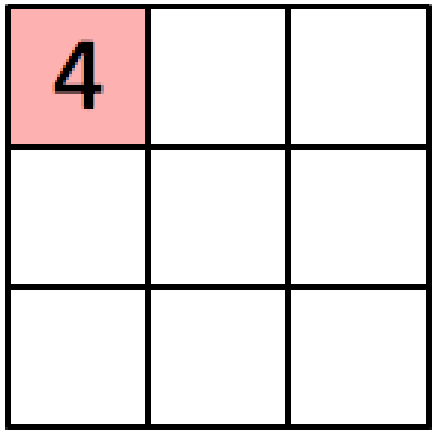


- Simple two layer network is essentially equivalent to having a **rigid template** for each shape.
- Hand-written digits vary in many complicated ways
  $\rightsquigarrow$ simple template matches of whole shapes are not sufficient.
- To capture all variations we need to **learn the features**
  $\rightsquigarrow$ **add more layers.**
- One possible way: learn different (linear) filters
  $\rightsquigarrow$ **convolutional neural nets (CNNs).**

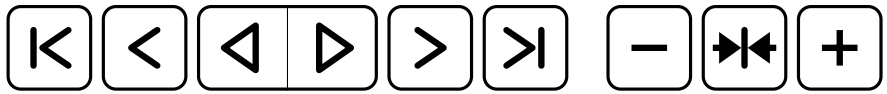
Convolutions



Image



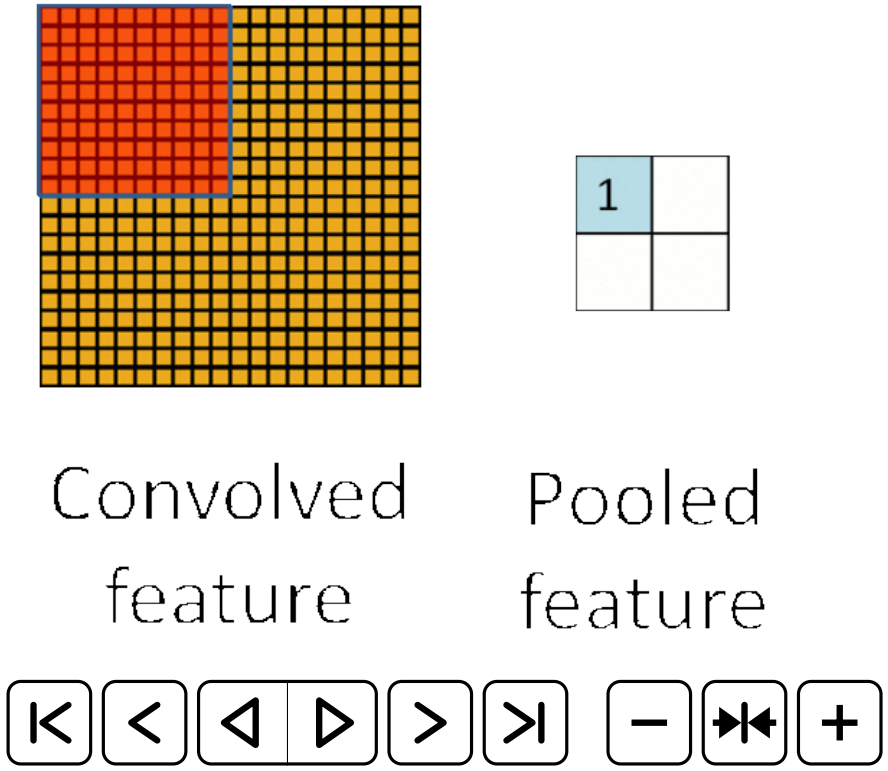
Convolved Feature



deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Pooling the outputs of replicated feature detectors

- **Averaging neighboring detectors**
  $\rightsquigarrow$ Some amount of **translational invariance.**
- **Reduces the number of inputs** to the next layer.
- Taking the **maximum** works slightly better in practice.



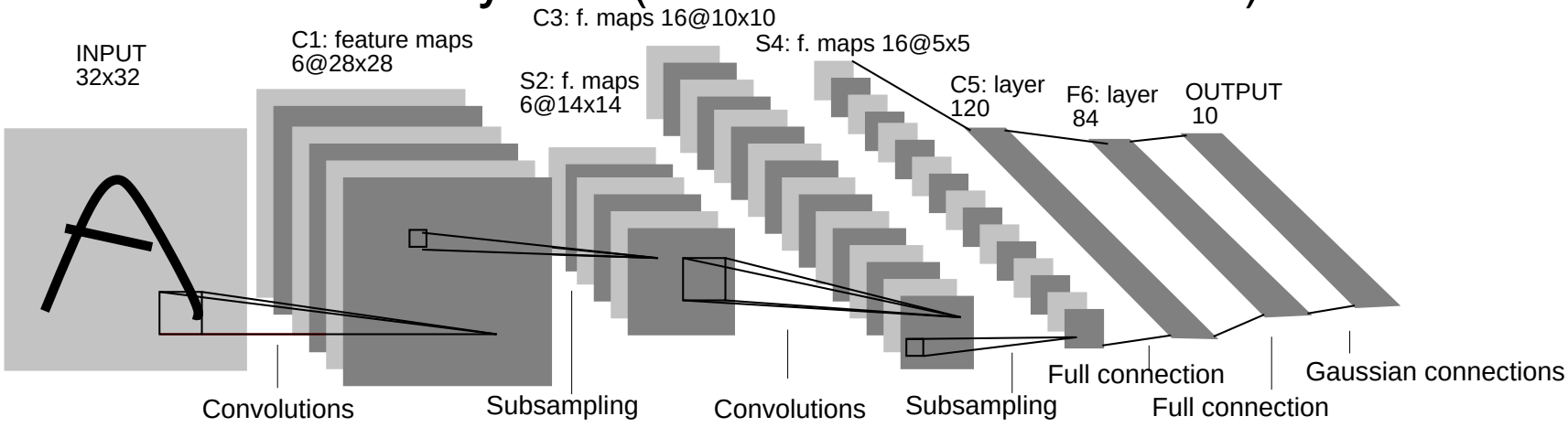
Source: deeplearning.stanford.edu/wiki/index.php/File:Pooling_schematic.gif

LeNet

Yann LeCun and his collaborators developed a really good recognizer for handwritten digits by using **backpropagation** in a feed-forward net with:

- many hidden layers
- many maps of replicated units in each layer.
- pooling of the outputs of nearby replicated units.

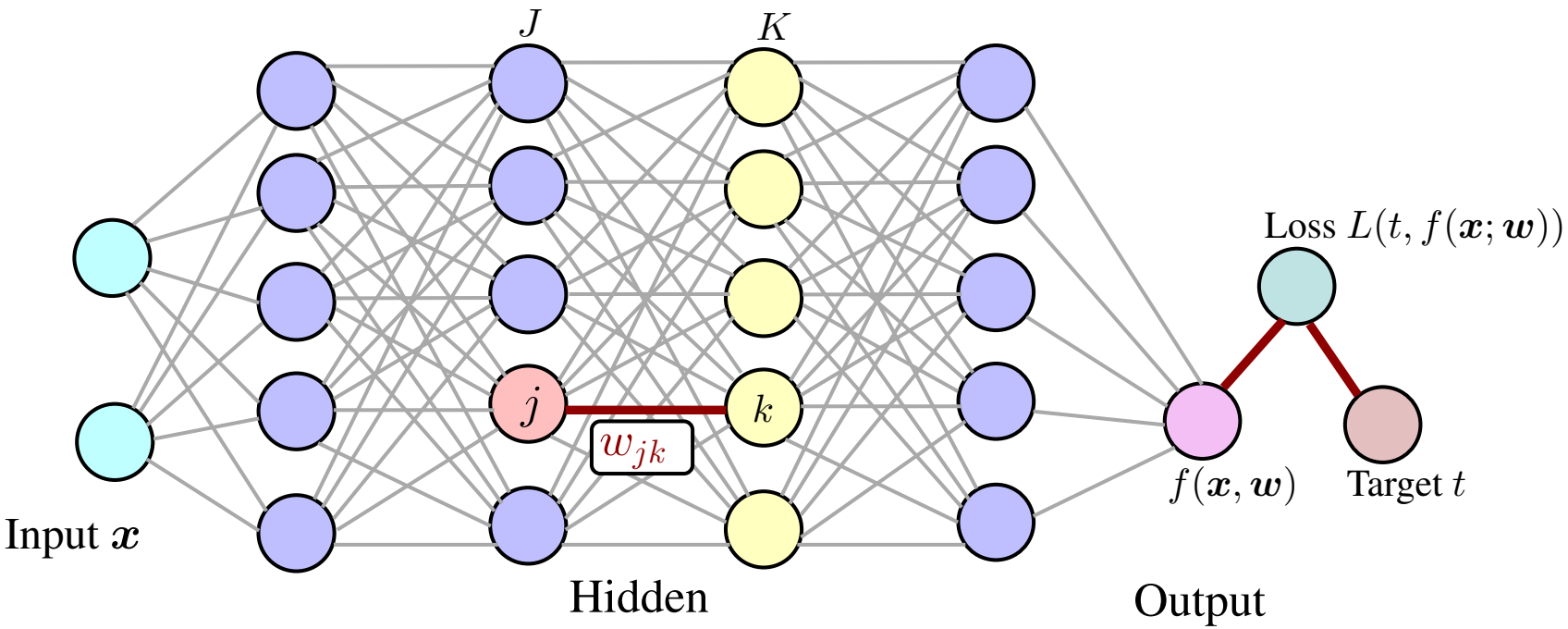
On the **US Postal Service** handwritten digit benchmark dataset the error rate was only 4% (human error $\approx 2 - 3\%$).



Original Image published in [LeCun et al., 1998]

Machine Learning and Neural Networks

Training & expressive power



Backpropagation

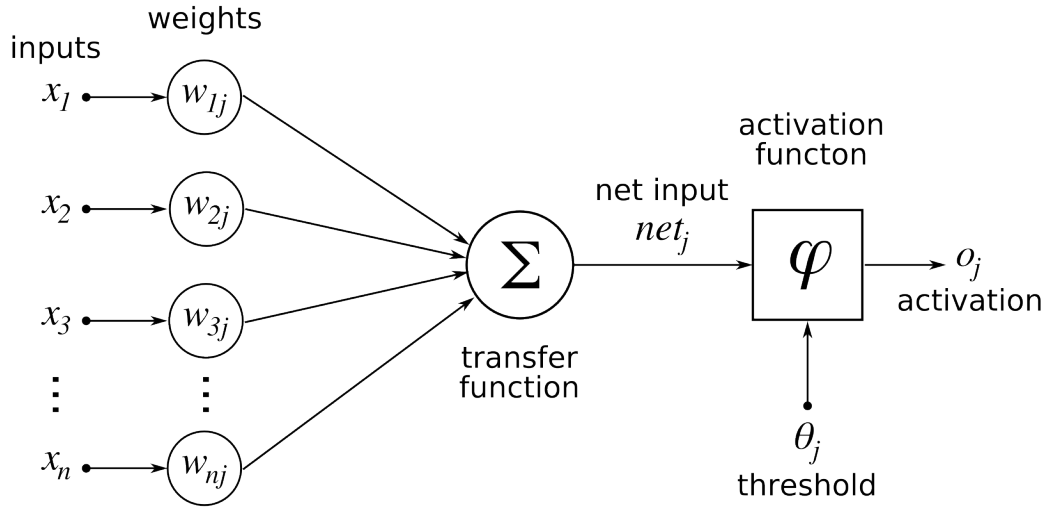
Assuming one output neuron, the error function is

$$E = \text{Loss}(t, f).$$

For each neuron j, its output o_j is defined as

$$o_j = \varphi(\text{net}_j) = \varphi\left(\sum_{i=1}^{n} w_{ij} o_i\right).$$

net_j = weighted sum of inputs (i.e. outputs of previous neurons).



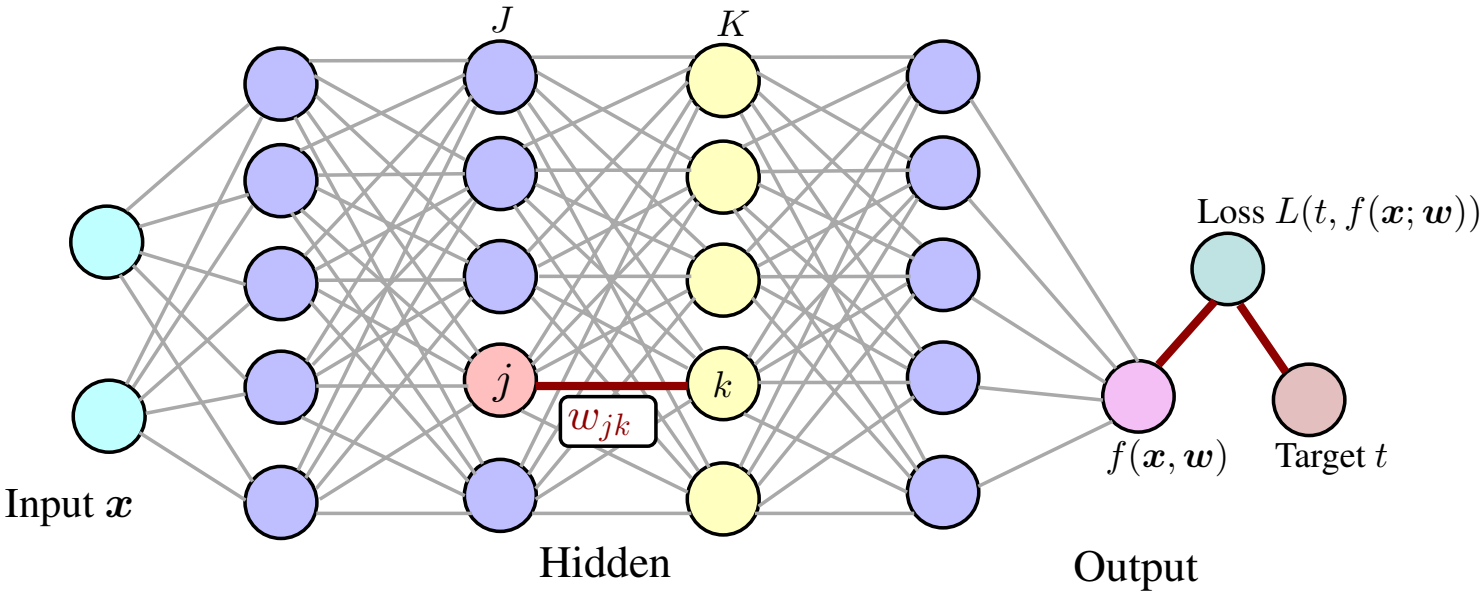
By Chrislb, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=224555

Backpropagation

Networks are usually trained by **gradient descent**

Gradient = derivatives of loss w.r.t. weights:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial w_{ij}} = \frac{\partial E}{\partial o_j}\overbrace{\frac{\partial o_j}{\partial \text{net}_j}}^{=\frac{\partial \varphi(\text{net}_j)}{\partial \text{net}_j}}\overbrace{\frac{\partial \text{net}_j}{\partial w_{ij}}}^{=o_i}$$



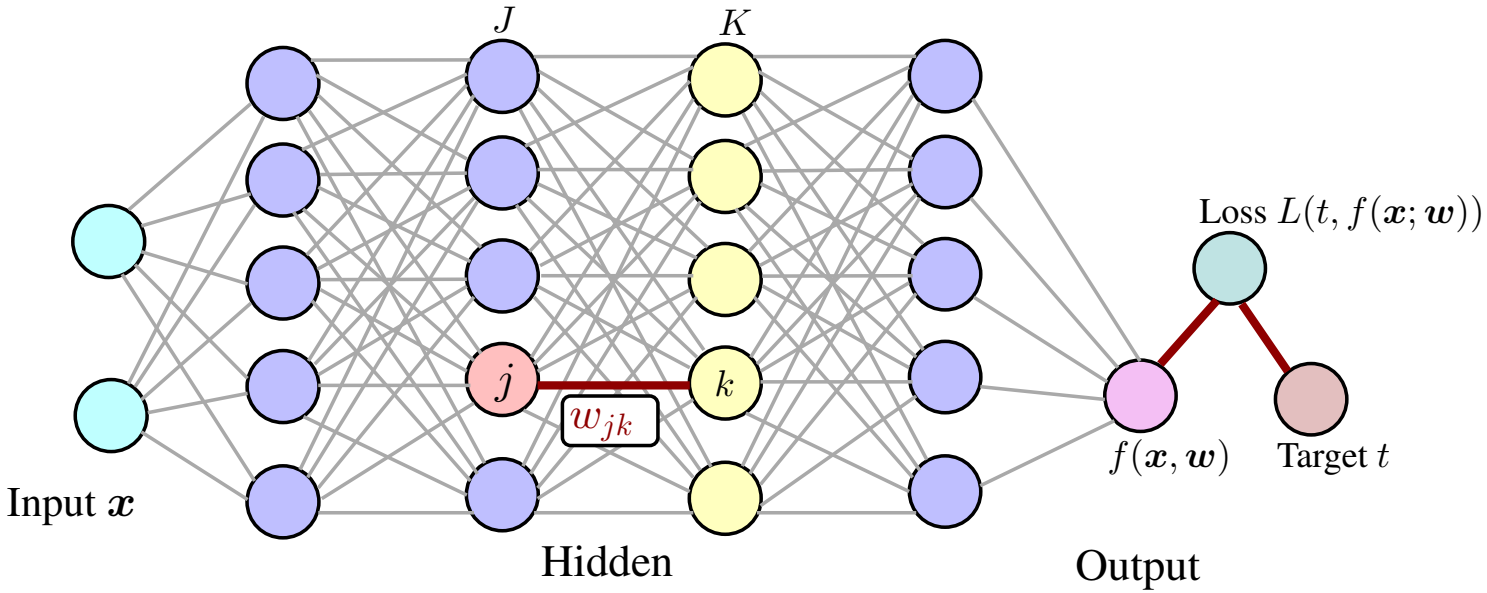
Backpropagation

First term: Consider $E(o_j)$ as a function of all neurons receiving input from neuron j (those in layer K): $E(\text{net}_{k_1}, \text{net}_{k_2}, \ldots, \text{net}_{k_n})$
Total derivative & chain rule $\rightsquigarrow$ recursive structure:

$$\frac{\partial E}{\partial o_j} = \sum_{k \in K} \left(\frac{\partial E}{\partial \text{net}_k} \frac{\partial \text{net}_k}{\partial o_j} \right) = \sum_{k \in K} \left(\frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial \text{net}_k} \frac{\partial \text{net}_k}{\partial o_j} \right) = \sum_{k \in K} \left(\frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial \text{net}_k} w_{jk} \right)$$

$\rightsquigarrow$ Derivatives in layer J depend only on those in next layer K

$\rightsquigarrow$ error is"backpropagated".



Expressive Power of Networks

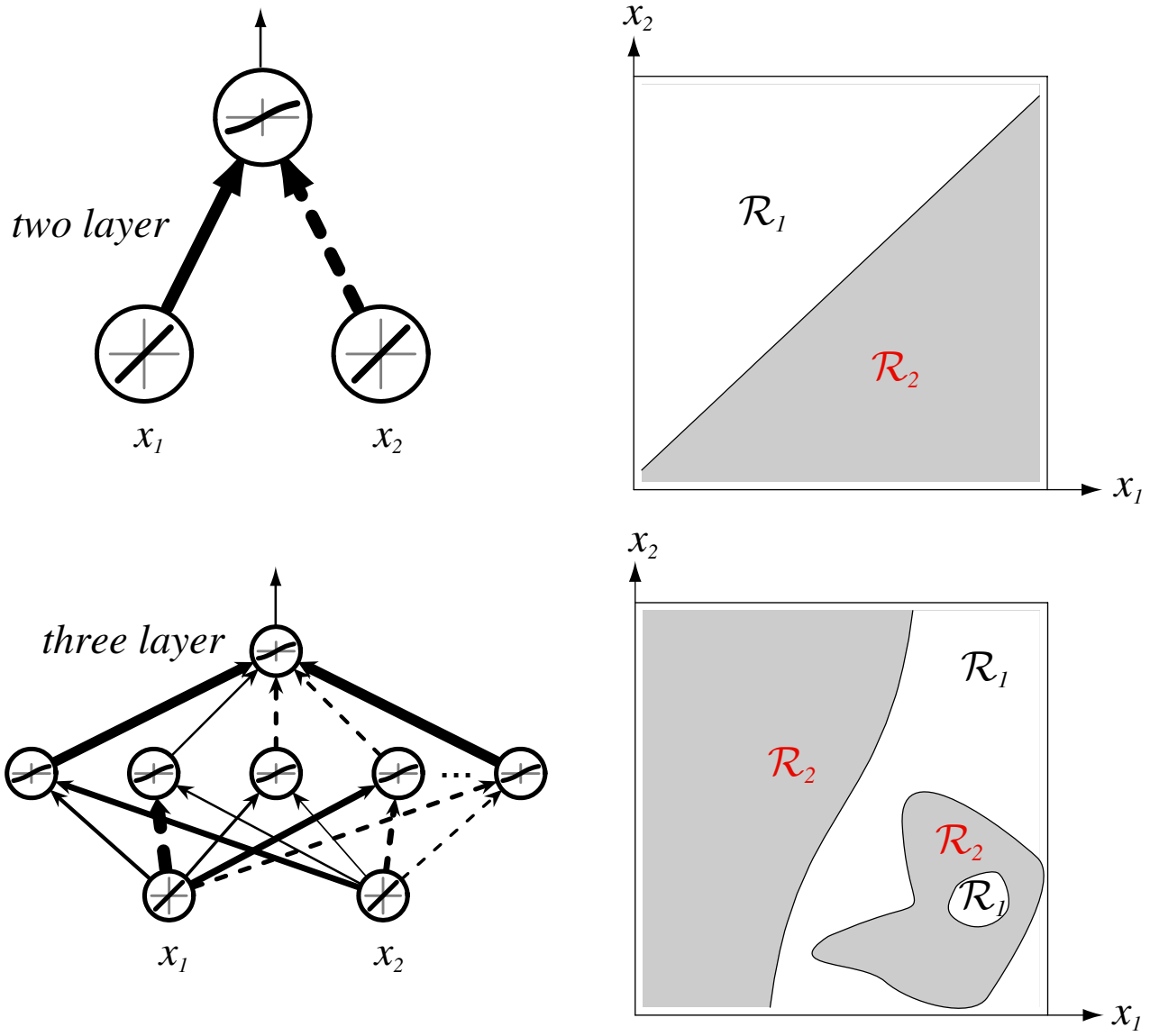


Fig 6.3 in (Duda, Hart & Stork)

Expressive Power of Networks

- **Question:** can every decision be implemented by a **three-layer network?**
- **Answer:** Basically yes – if the input-output relation is continuous and if there are **sufficiently many hidden units.**
- **Theorem** (Kolmogorov 61, Arnold 57, Lorentz 62): every continuous function $f(x)$ on the hypercube I^d $(I = [0, 1], d \geq 2)$ can be represented in the form

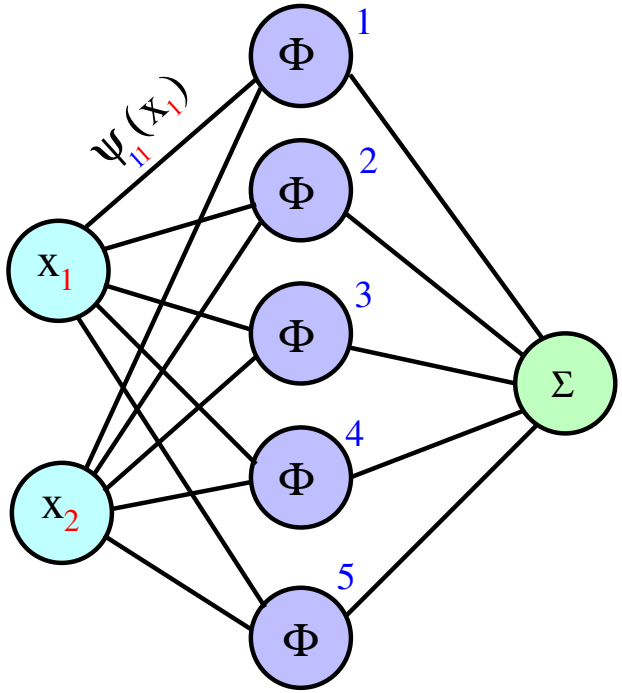
$$f(x) = \sum_{j=1}^{2d+1} \Phi \left(\sum_{i=1}^{d} \psi_{ji}(x_i) \right),$$

  for properly chosen functions Φ, ψ_{ji}.
- Note that we can always rescale the input region to lie in a hypercube.

Expressive Power of Networks

Relation to three-layer network:

- Each of $2d + 1$ hidden units takes as input a sum of d nonlinear functions, one for each input feature x_i.
- Each hidden unit emits a nonlinear function Φ of its total input.
- The output unit emits the sum of all contributions of the hidden units.



Problem: Theorem guarantees only existence

$\rightsquigarrow$ might be hard to find these functions.

Are there "simple" function families for Φ, ψ_{ji}?

Universal approximations by ridge functions

Theorem (Cybenko 89, Hornik 91, Pinkus 99). *Let $\varphi(\cdot)$ be a non-constant, bounded, and monotonically-increasing continuous function. Let I^d denote the unit hypercube $[0,1]^d$, and $C(I^d)$ the space of continuous functions on I^d. Then, given any $\varepsilon > 0$ and any function $f \in C(I^d)$, there exist an integer N, real constants $v_i, b_i \in \mathbb{R}$ and real vectors $\boldsymbol{w}_i \in \mathbb{R}^d$, $i = 1, \cdots, N$, such that we may define:*

$$F(x) = \sum_{i=1}^{N} v_i \varphi\left(\boldsymbol{w}_i^t \boldsymbol{x} + b_i\right)$$

as an approximate realization of the function f, i.e. $\|F - f\|_\infty < \varepsilon$.

In other words, functions of the form $F(x)$ are universal approximators for continuous functions.

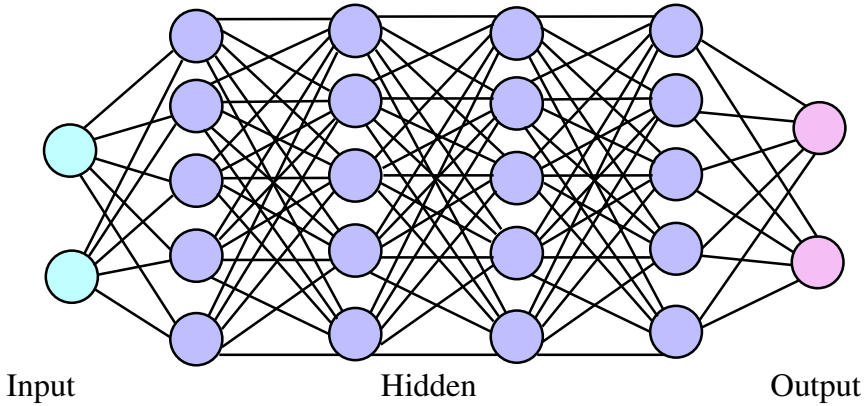
Artificial Neural Networks: Rectifiers

- Classic activation functions are indeed **bounded** and **monotonically-increasing continuous** functions like *tanh*.
- In practice, however, it is often better to use "simpler" activations.
- **Rectifier:** activation function defined as:

$$f(x) = x^+ = \max(0, x),$$

  where x is the input to a neuron.
- A unit employing the rectifier is called **rectified linear unit (ReLU).**
- What about approximation guarantees? Basically, we have the same guarantees, but at the price of **wider layers:**
  Theorem (Shekhtman (1982), using classical results from polygonal approximations due to Lebesgue (1898)). *Networks with one (wide enough) hidden layer of ReLU are universal approximators for continuous functions.*

Why should we use more hidden layers?

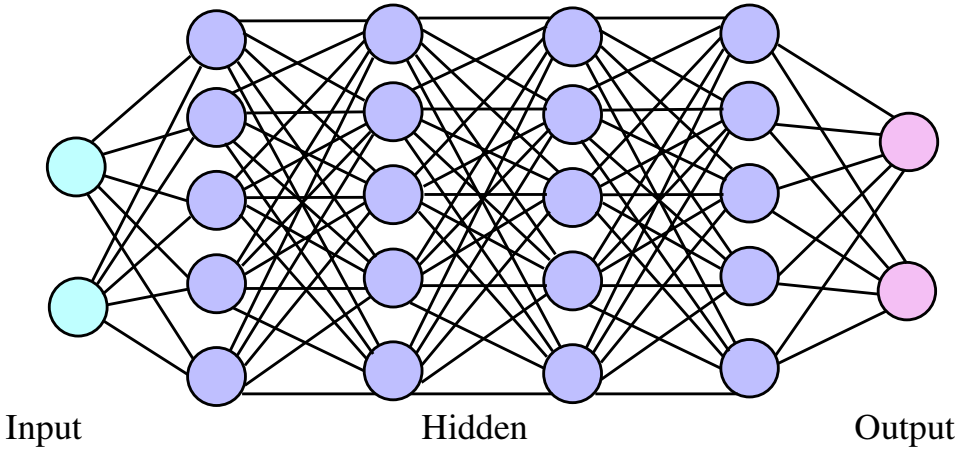


- Characterize the expressive power by counting into how many cells we can partition $\mathbb{R}^d$ with combinations of rectifying units.
- A rectifier is a piecewise linear function. It partitions $\mathbb{R}^d$ into two open half spaces (and a border face):

$$\begin{aligned} H^+ &= \boldsymbol{x} : \boldsymbol{w}^t \boldsymbol{x} + b > 0 \in \mathbb{R}^d \\ H^- &= \boldsymbol{x} : \boldsymbol{w}^t \boldsymbol{x} + b < 0 \in \mathbb{R}^d \end{aligned}$$

- Question: by linearly combining m rectified units, into how many cells is $\mathbb{R}^d$ maximally partitioned?
- Explicit formula (Zaslavsky 1975): An arrangement of m hyperplanes in $\mathbb{R}^n$ has at most $\sum_{i=0}^{n} \binom{m}{i}$ regions.

Deep Learning



Applied to ReLu networks:

Theorem (Montufar et al, 2014). *A rectifier neural network with d input units and L hidden layers of width $m \geq d$ can compute functions that have $\Omega\left(\left(\frac{m}{d}\right)^{(L-1)d} m^d\right)$ linear regions.*

Important insights:

- The number of linear regions of deep models grows **exponentially in L** and **polynomially in m**.
- This growth is much faster than that of **shallow networks** with the same number mL of hidden units in a single wide layer.

Recurrent Neural Networks

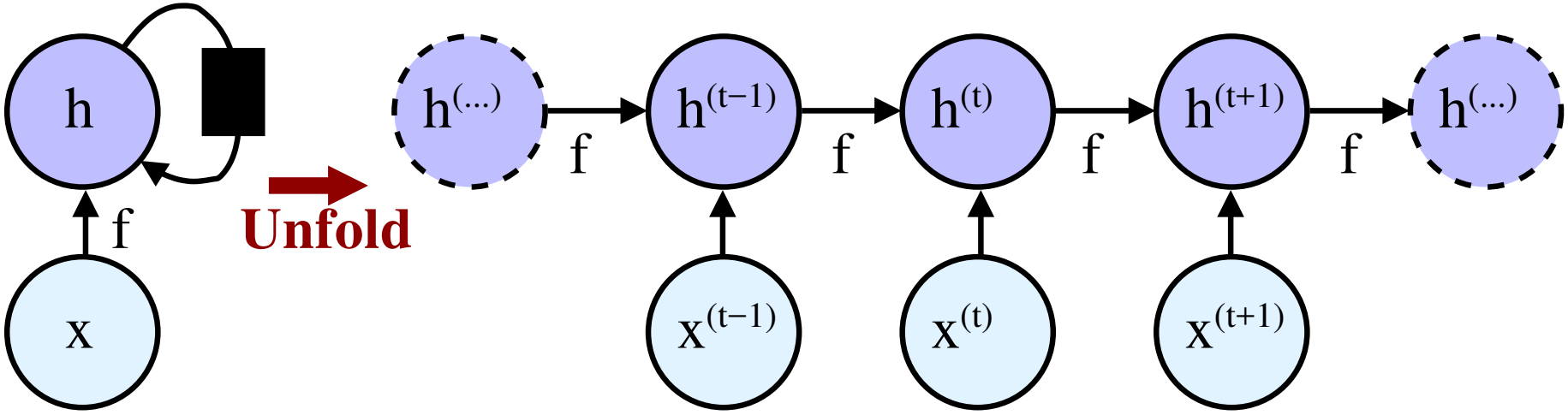
- Classical form of a **dynamical system:**

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}; \boldsymbol{\theta}),$$

where $\boldsymbol{h}^{(t)}$ is the **state** of the system.

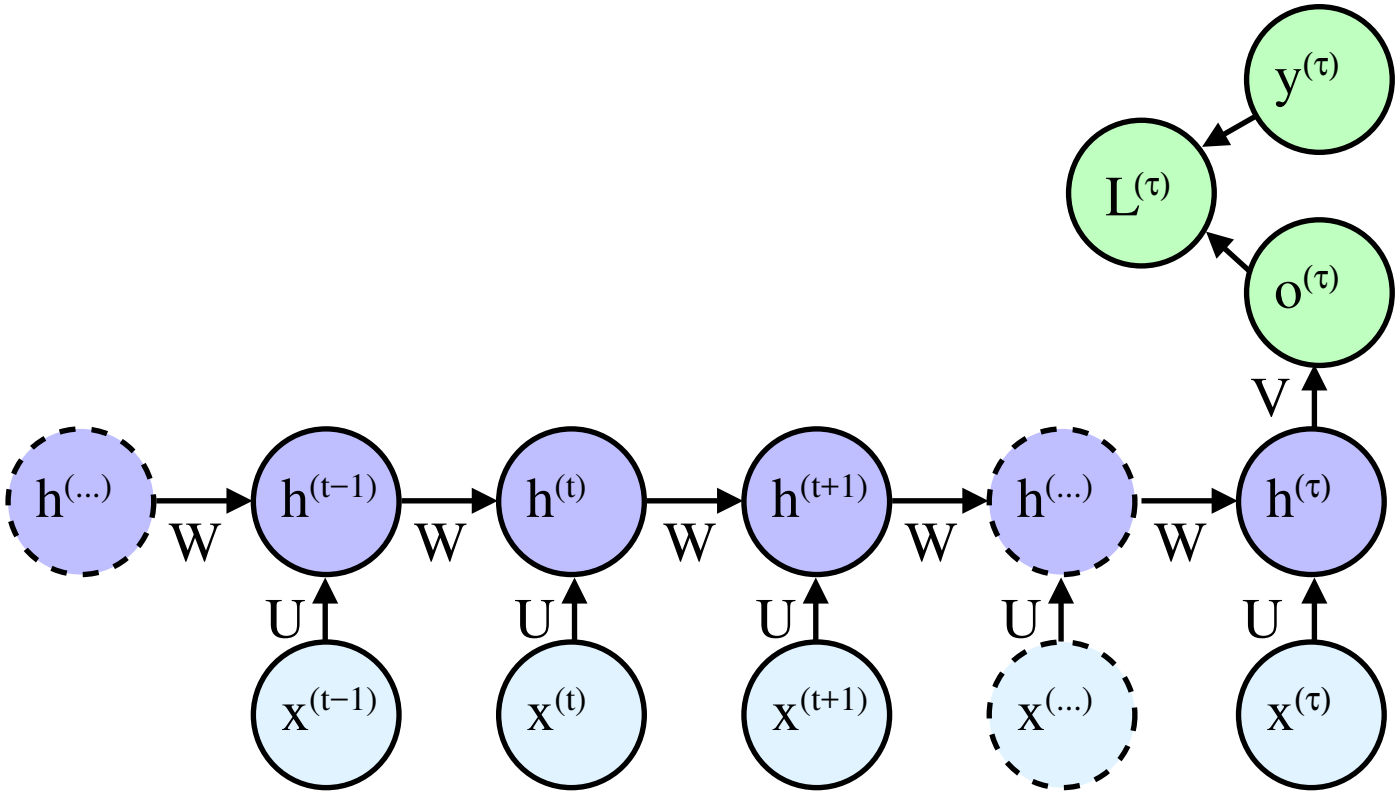
- Often, a dynamical system is driven by an **external signal:**

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}).$$



Unfolding Computational Graphs

- The network typically learns to use the fixed length state $\boldsymbol{h}^{(t)}$ as a **lossy summary of the task-relevant aspects of $\boldsymbol{x}^{(1:t)}$.**

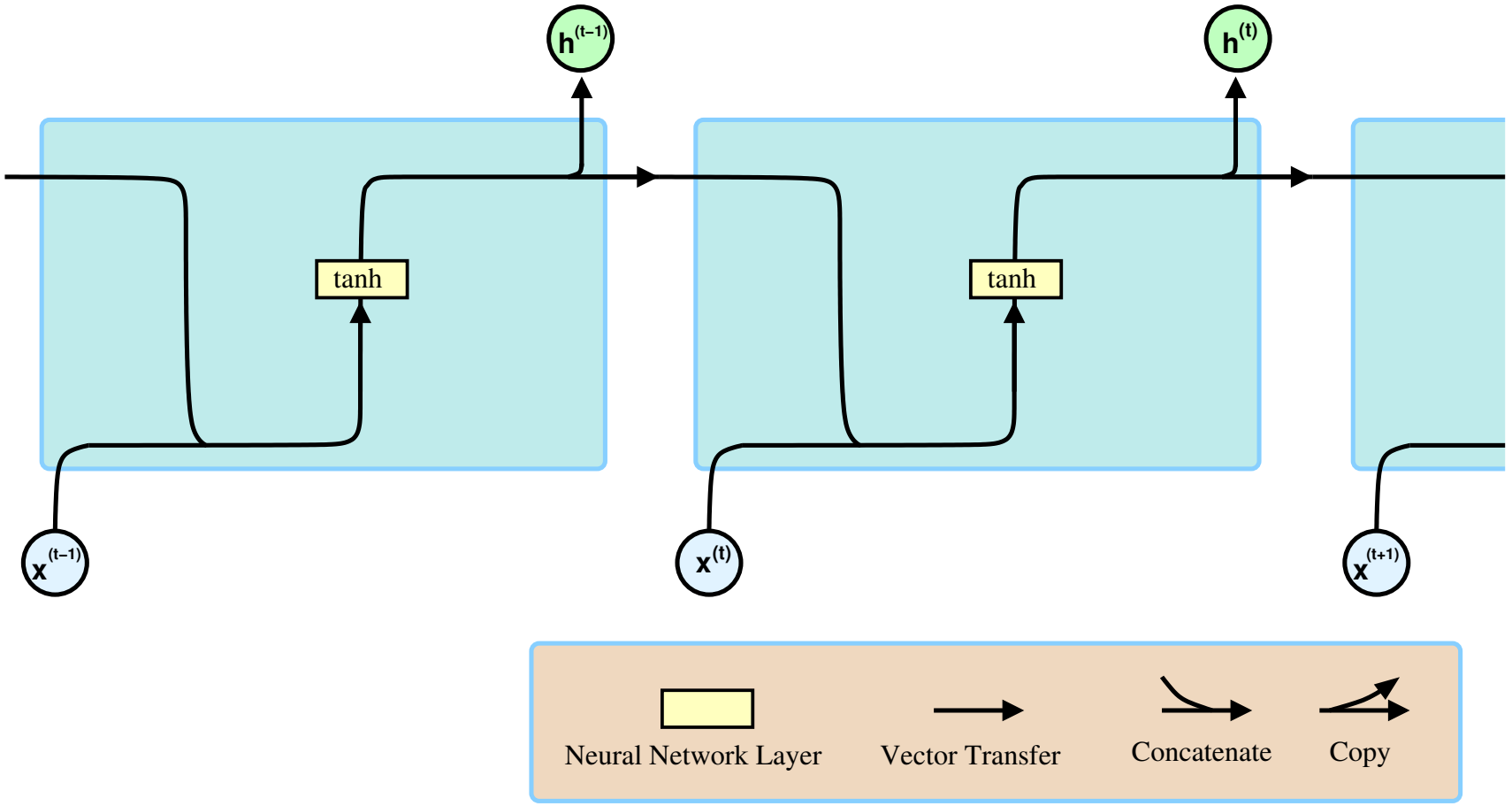


Such a network can be used to summarize a sequence and produce a **fixed-size representation for sequences of any length.**

Long short-term memory (LSTM) cells

- Theory: RNNs can keep track of **arbitrary long-term dependencies.**
- **Practical problem:** computations in finite-precision:
  $\rightsquigarrow$ **Gradients can vanish or explode.**
- RNNs using **LSTM units** partially solve this problem:
  LSTM units allow gradients to also **flow unchanged.**
  Exploding gradients may still occur $\rightsquigarrow$ use **gradient clipping.**
- Common architectures composed of a cell and three **regulators** or **gates:** input, output and forget gate.
- Variations: gated recurrent units (GRUs) do not have an output gate.
- **Input gate** controls to which extent a new value flows into the cell
- **Forget gate** controls to which extent a value remains in the cell
- **Output gate** controls to which extent the current value is used to compute the output activation.

Long short-term memory (LSTM) cells

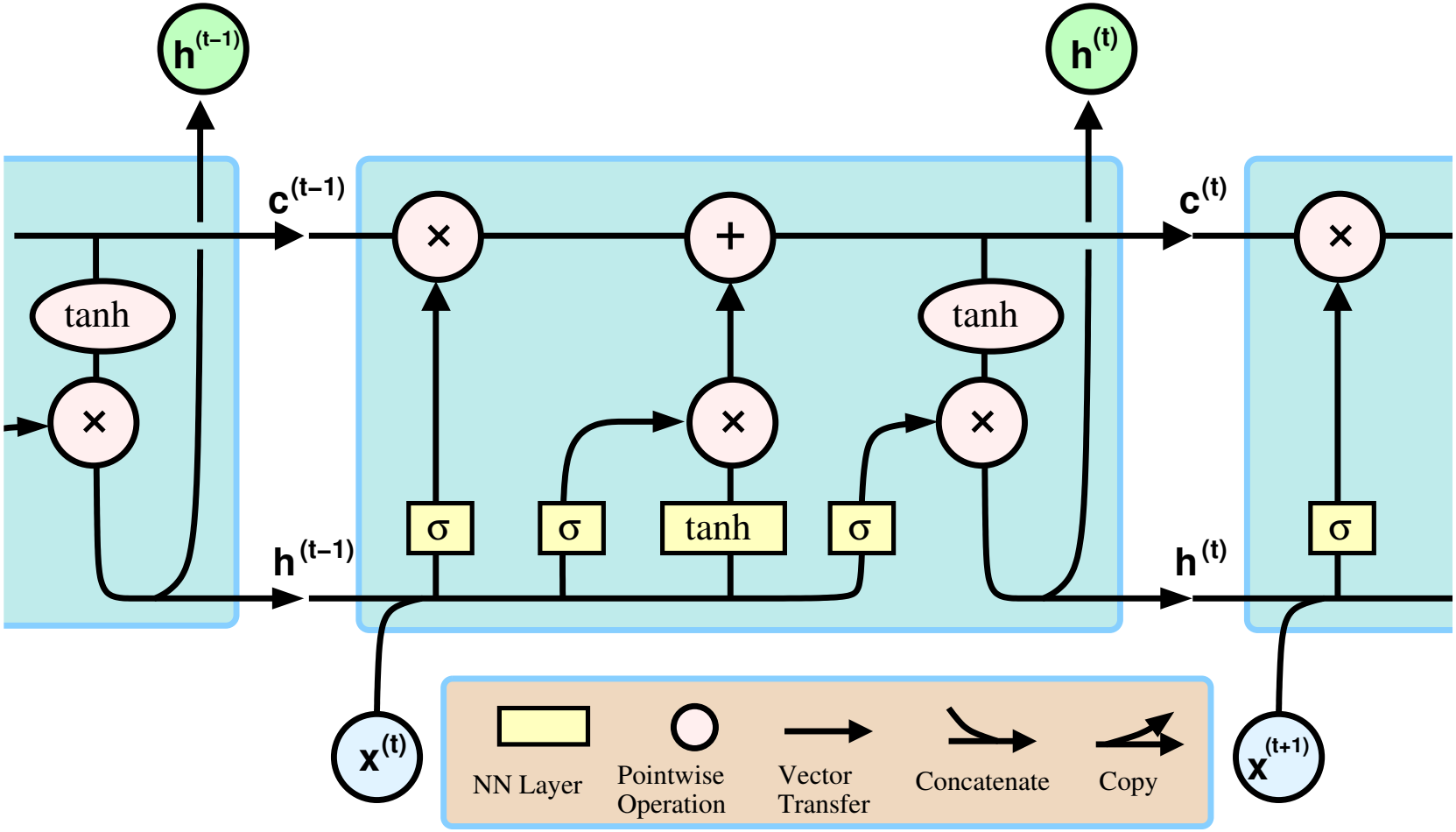


$$\boldsymbol{h}^{(t)} = \tanh(W[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b})$$

RNN cell takes current input $\boldsymbol{x}^{(t)}$ and outputs the hidden state $\boldsymbol{h}^{(t)}$

$\rightsquigarrow$ pass to the next RNN cell.

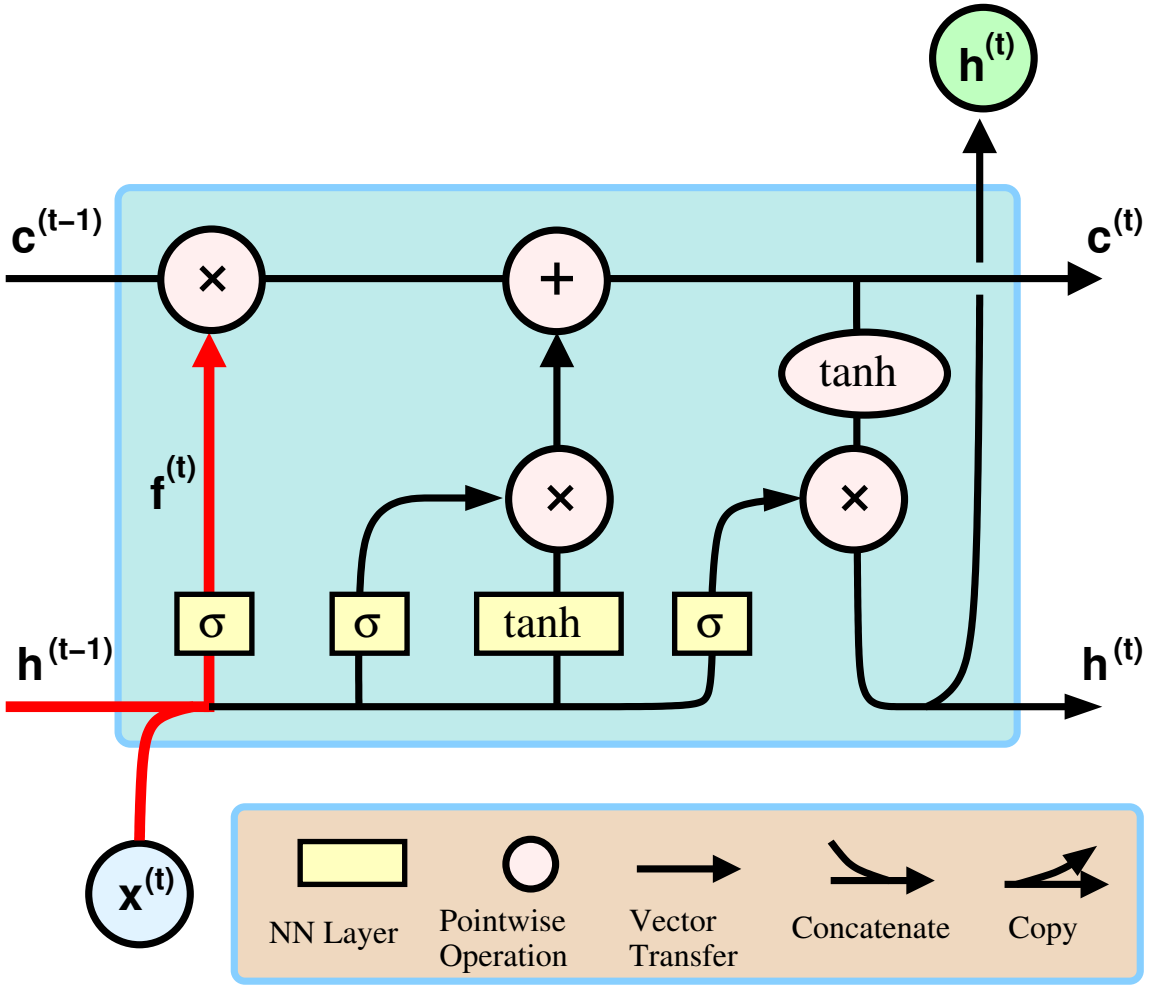
Long short-term memory (LSTM) cells



Cell states allows flow of unchanged information

⇝ helps preserving context, learning long-term dependencies.

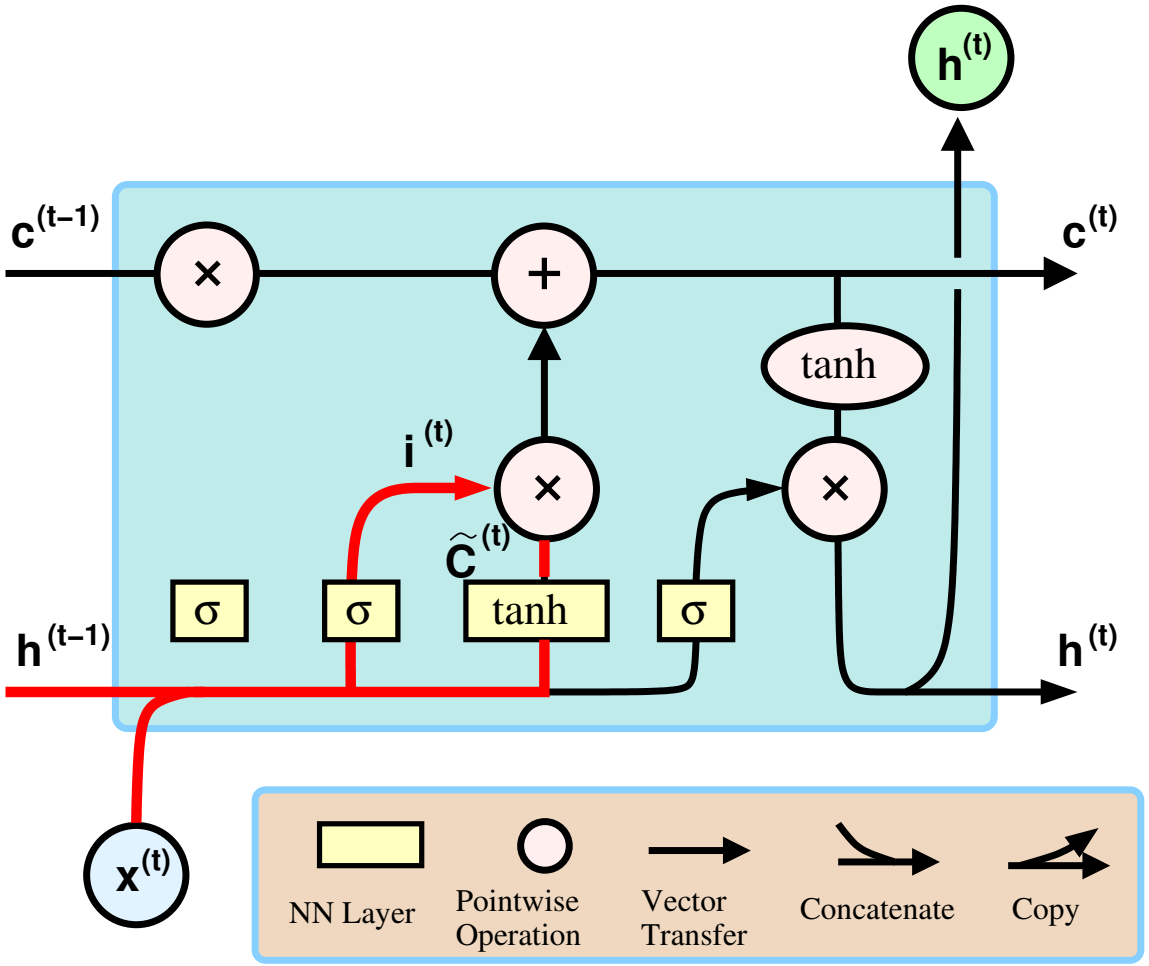
LSTM cells: Forget gate



$$\boldsymbol{f}^{(t)} = \sigma(W^f[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}^f)$$

Forget gate alters cell state based on current input $\boldsymbol{x}^{(t)}$ and output $\boldsymbol{h}^{(t-1)}$ from previous cell.

LSTM cells: Input gate

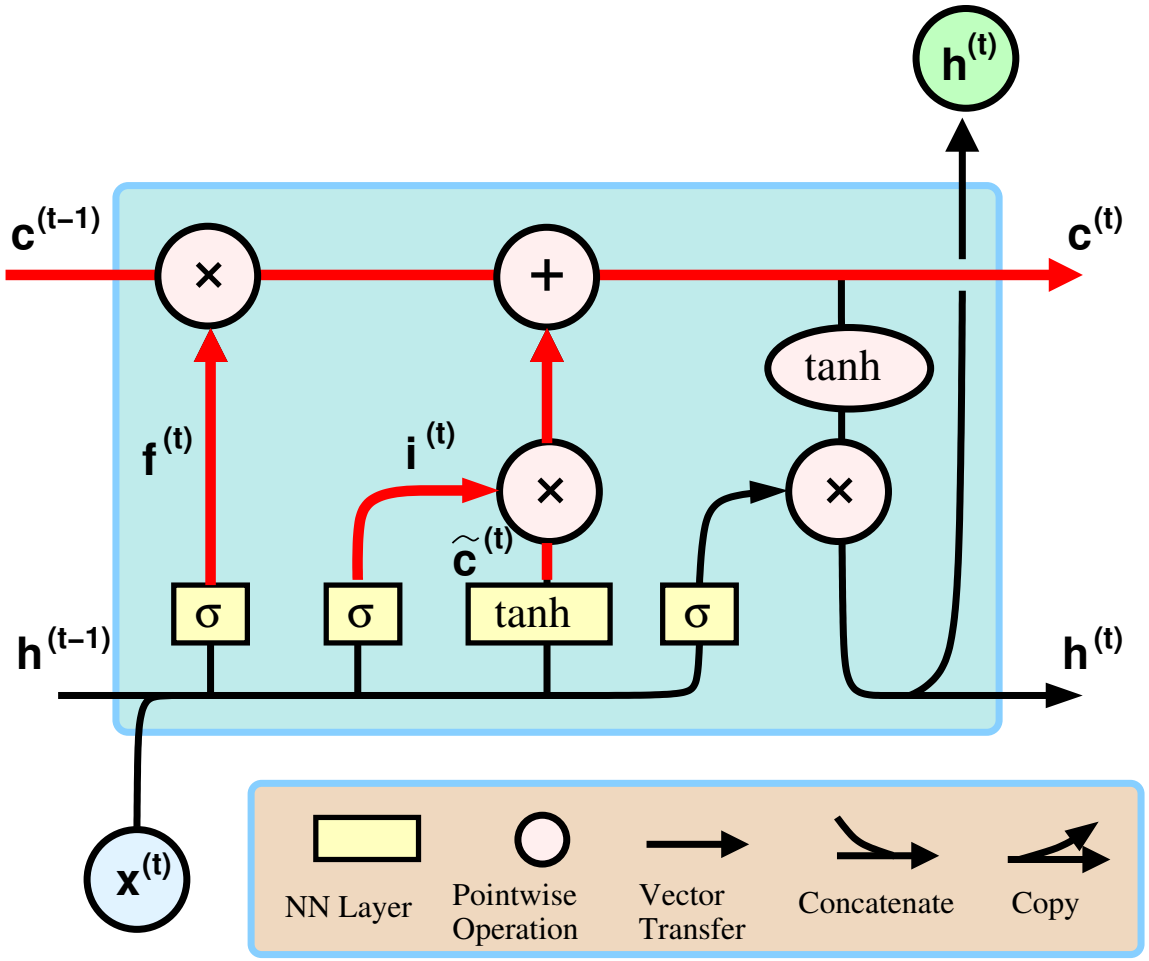


$$\boldsymbol{i}^{(t)} = \sigma(W^i[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}^i)$$

$$\tilde{\boldsymbol{c}}^{(t)} = \tanh(W^c[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}^c)$$

Input gate decides and computes values to be updated in the cell state.

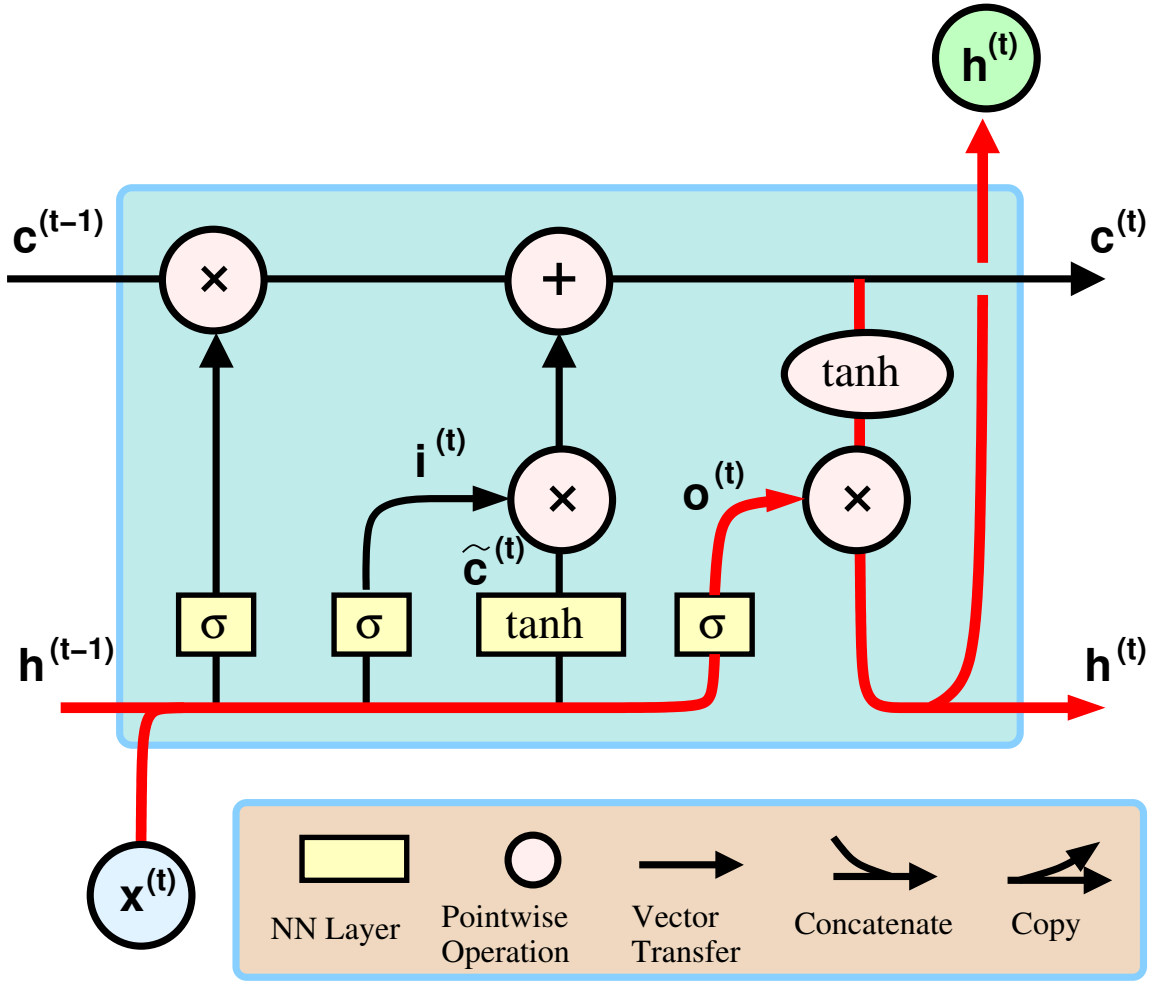
LSTM cells: Input gate (cont'd)



$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)}$$

Forget and input gate together update old cell state.

LSTM cells: Output gate

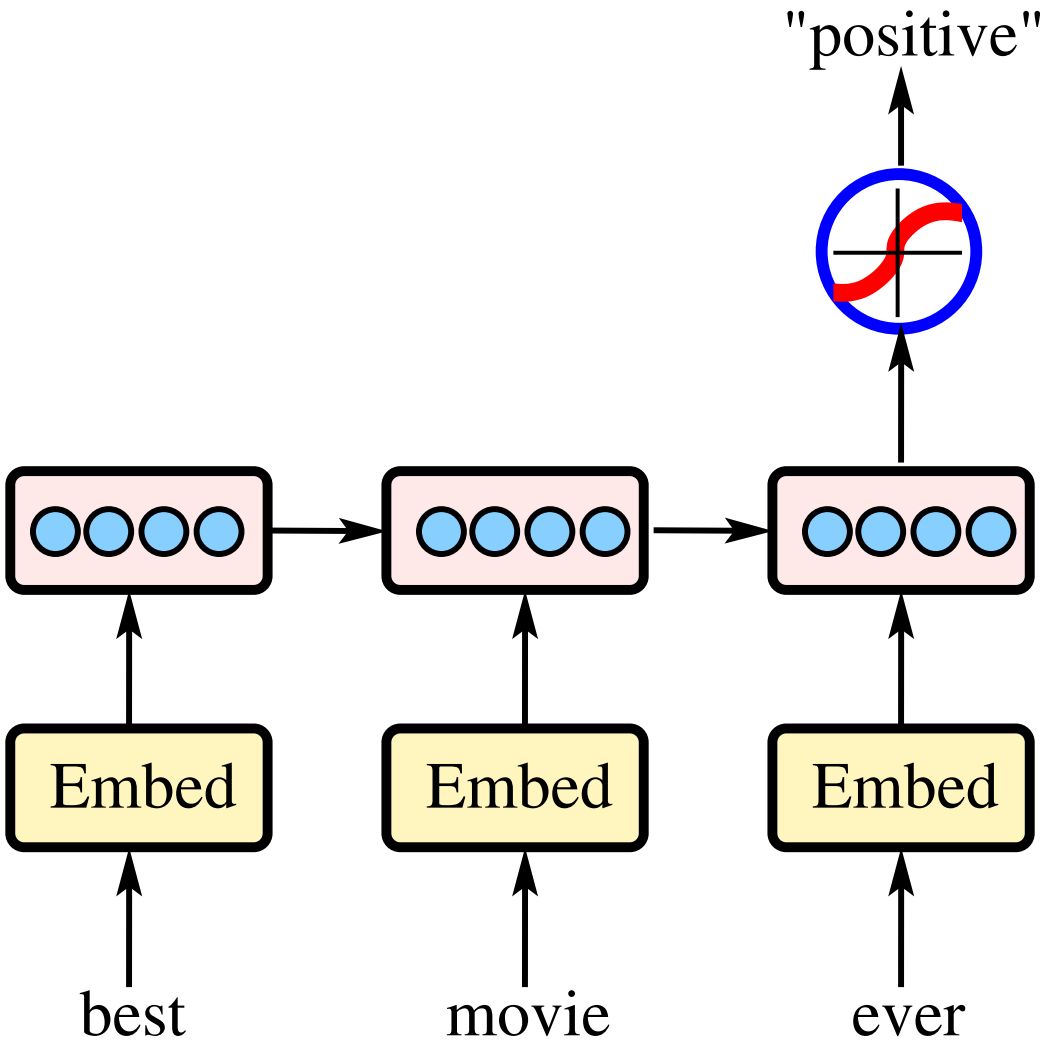


$$\boldsymbol{o}^{(t)} = \sigma(W^o[\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}] + \boldsymbol{b}^o)$$

$$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{c}^{(t)}) \circ \boldsymbol{o}^{(t)}$$

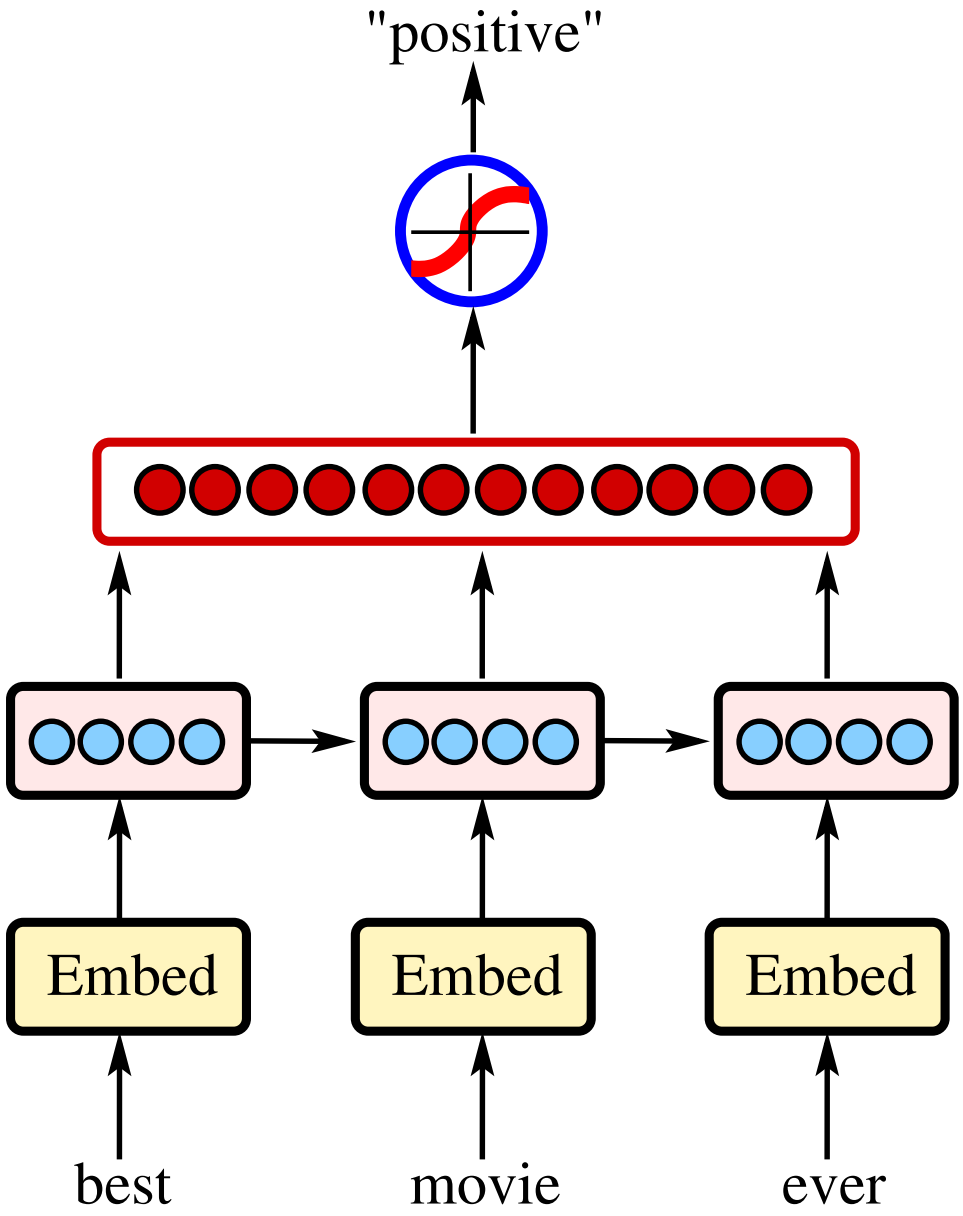
Output gate computes output from cell state to be sent to next cell.

Movie review example

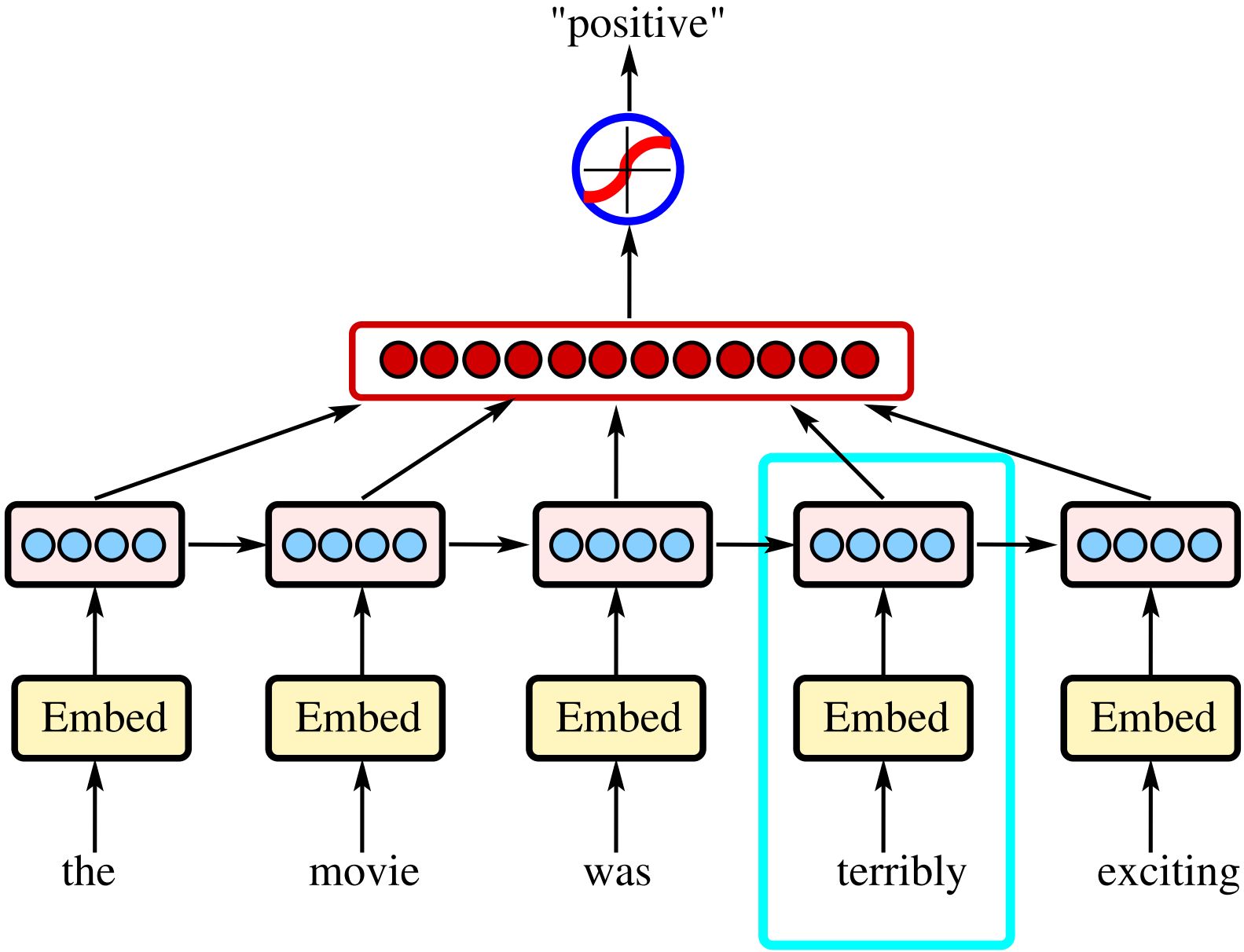


Inputs: words in a movie review

Movie review example: refined model

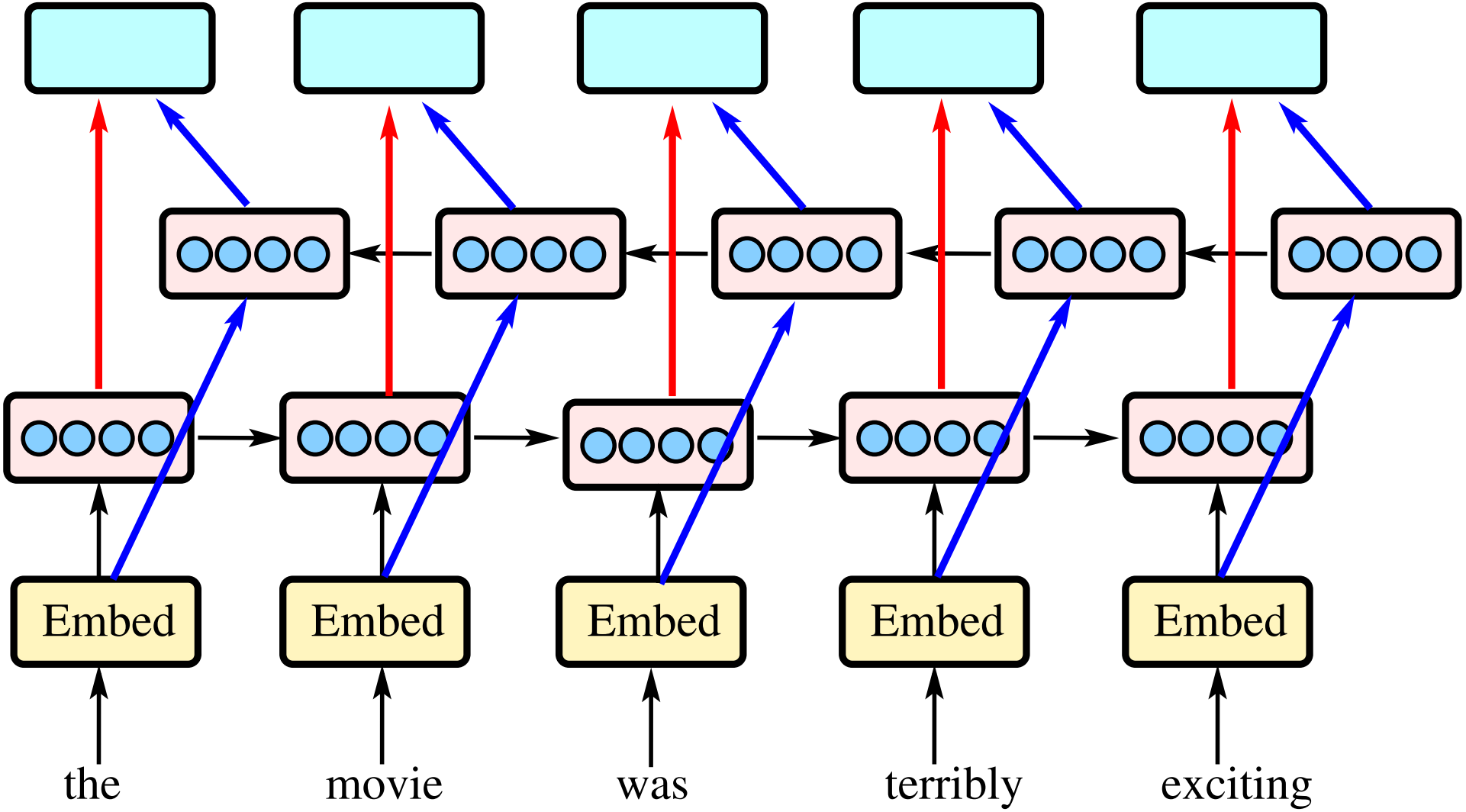


Left vs. right context

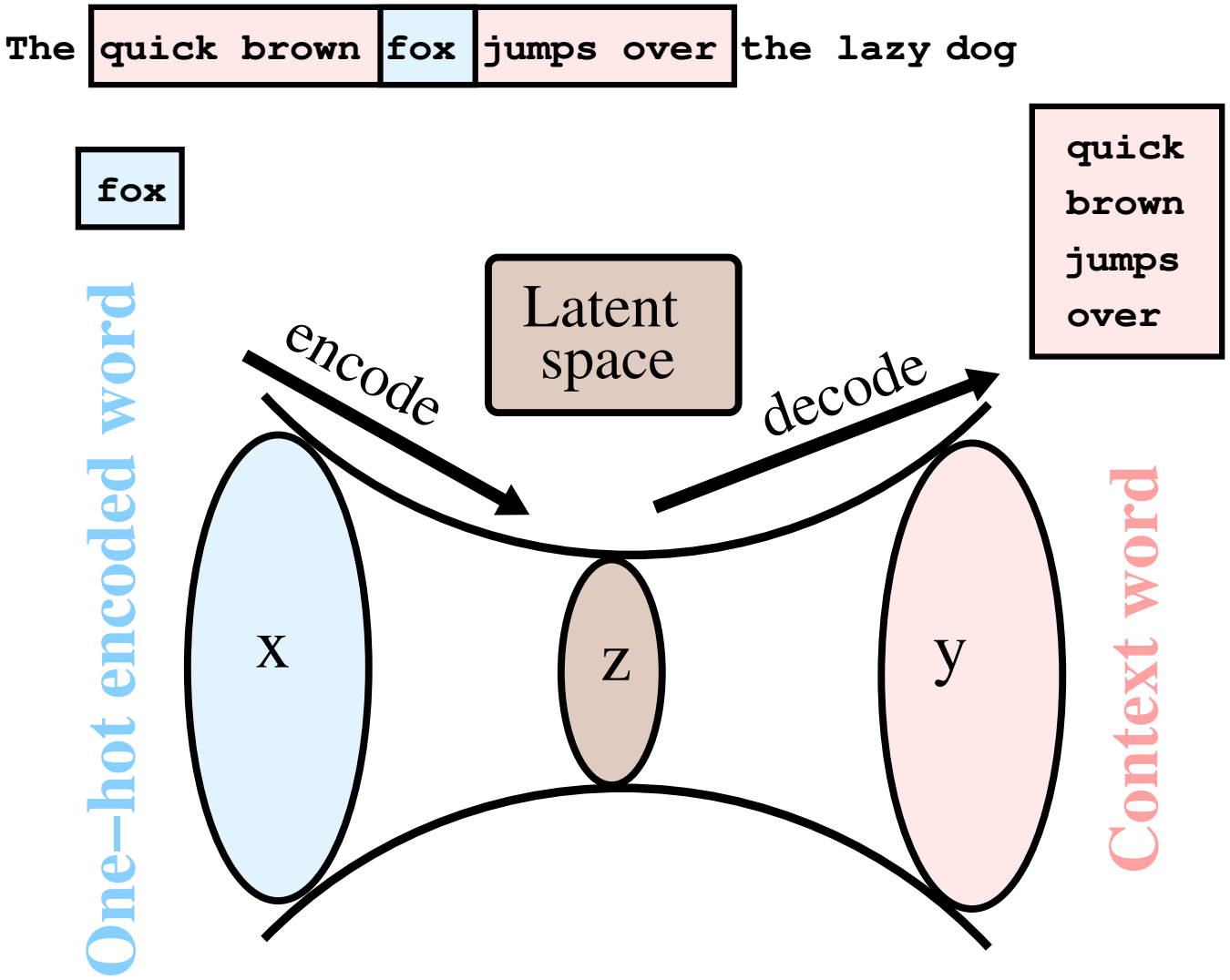


Bidirectional RNNs

Concatenated hidden states containing left and right context



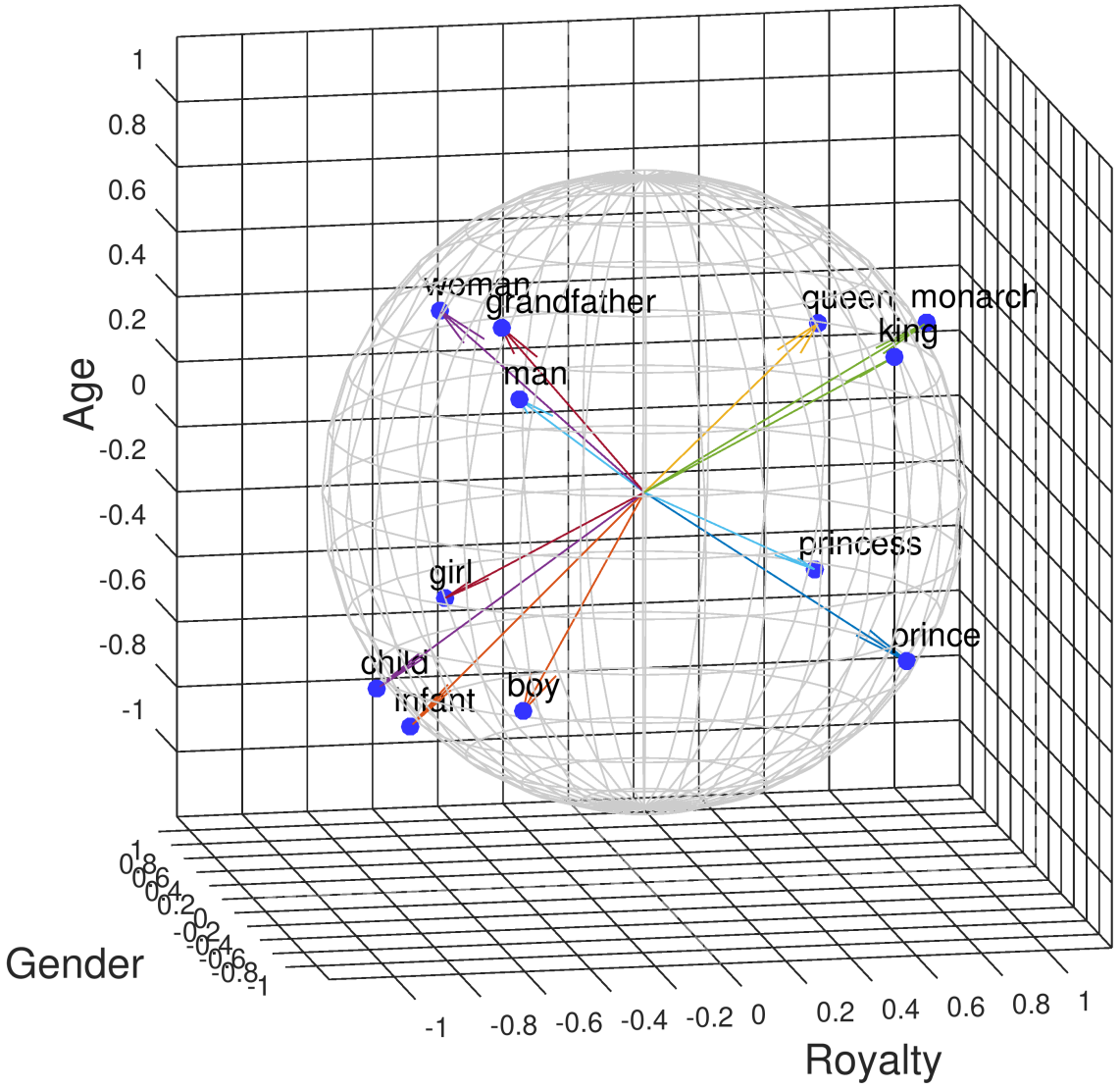
Word embeddings



Multidimensional, distributed representation of words in a vector space.

Word embeddings

Zero-Mean 3D Unit Vectors



https://www.cs.cmu.edu/~dst/WordEmbeddingDemo/tutorial.html

(Simple) neural Language Models

As the proctor started the clock, **the students opened their** .....

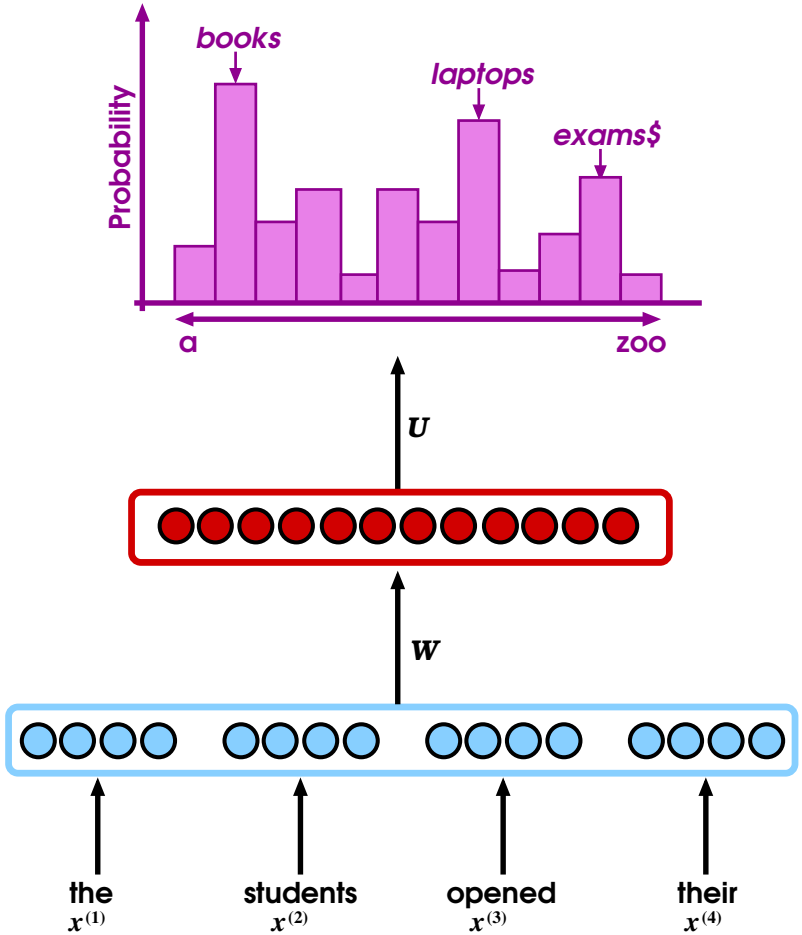
ignore

fixed window

Output layer:
probability for next word

Hidden layer:
enables flexible (non-linear) mappings

Word embeddings:
context-specific representation of words



We need an architecture that can process any length input.

Recurrent neural language models

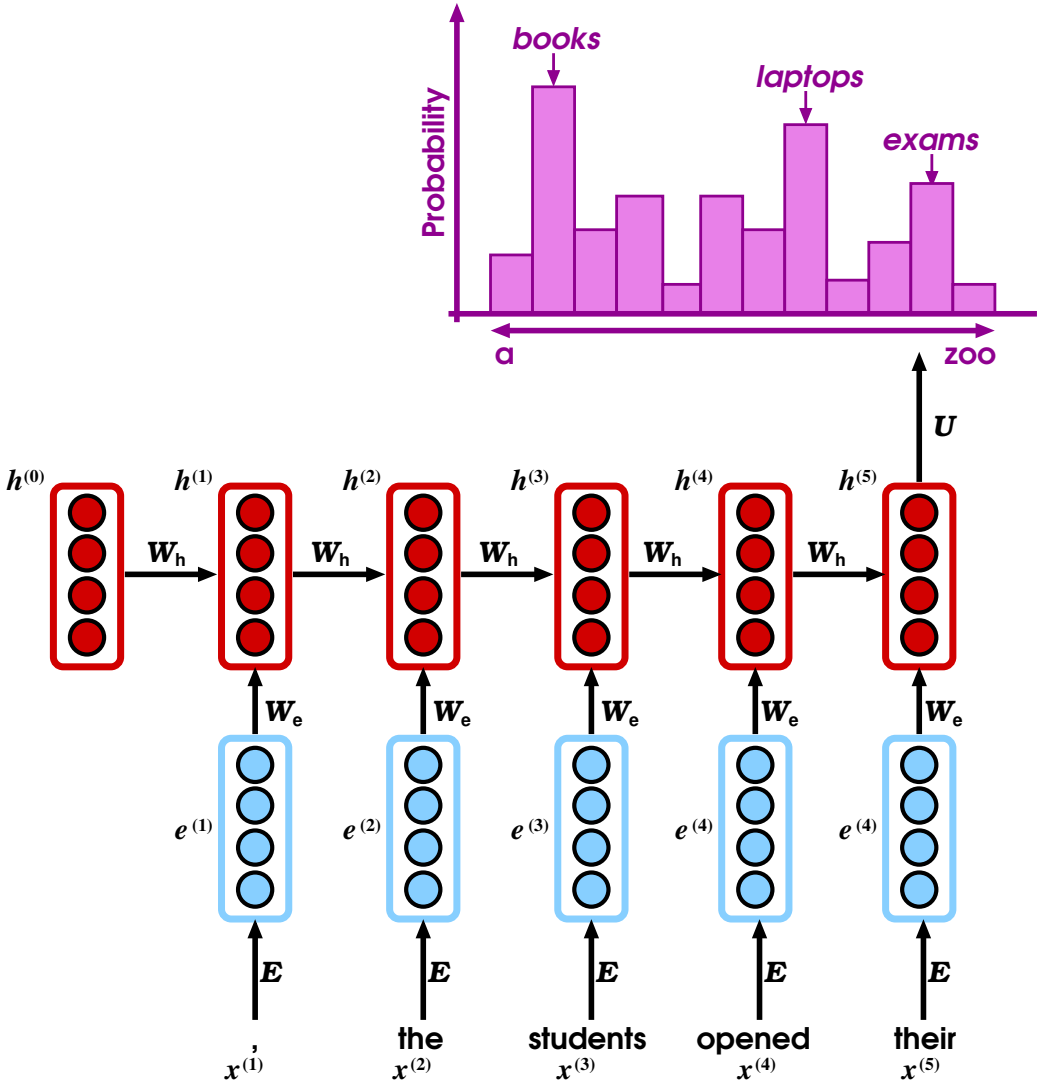
...started the clock, the students opened their .....

arbitrary window

Output layer:
probability for next word

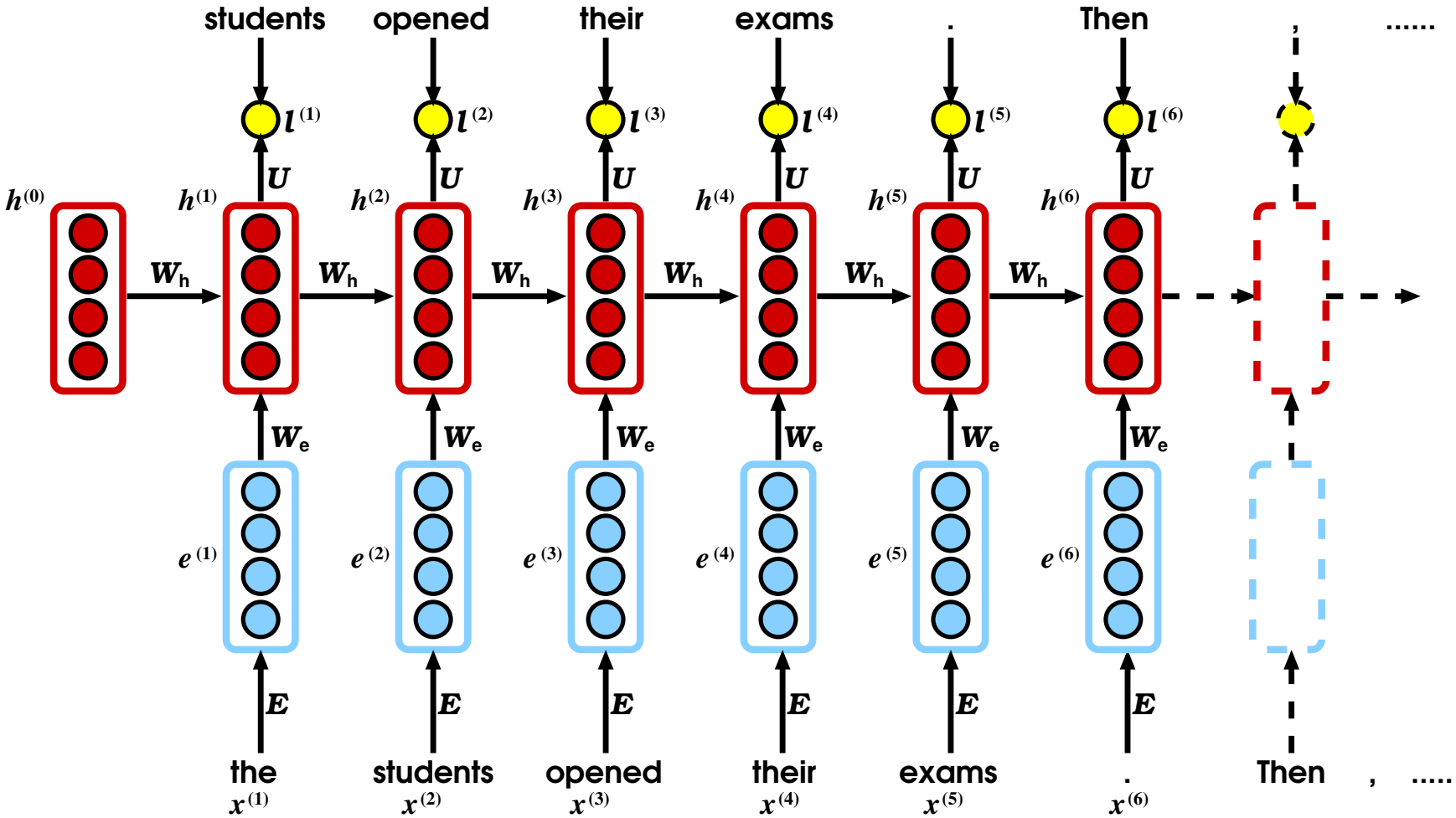
Hidden layer:
enables flexible (non-linear) mappings

Word embeddings:
context-specific representation of words



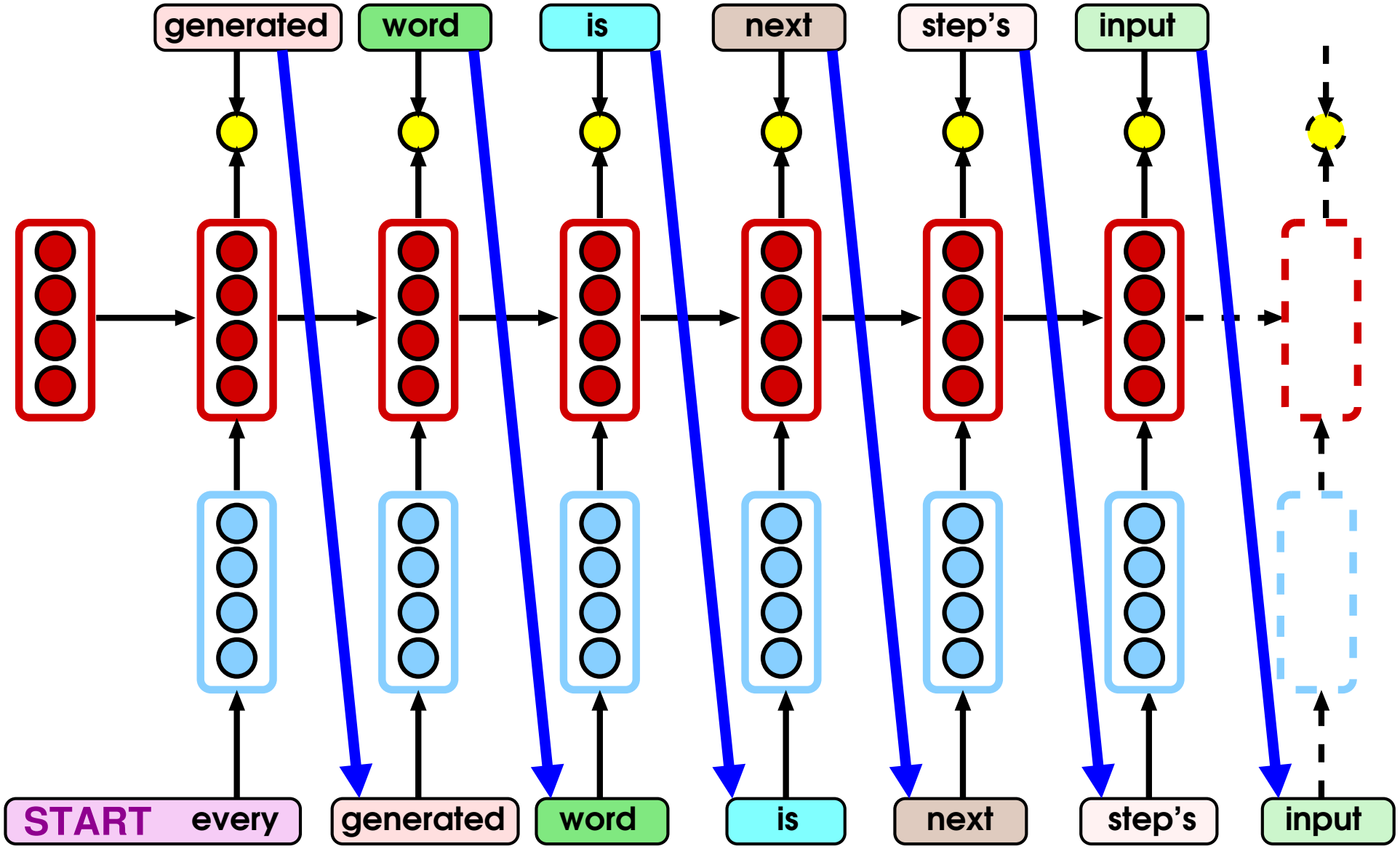
Recurrent neural language models: Training

Get a **big corpus of text** (long sequence of words, $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}\}$). For every step t: predict next word and compare with actual next word via a **loss function**. Averaged loss is minimized during training.



Recurrent neural language models

Generating new text



Recurrent neural language models

RNN trained on Obama speeches

https://commons.wikimedia.org/w/index.php?curid=23956389

RNN trained on Harry Potter

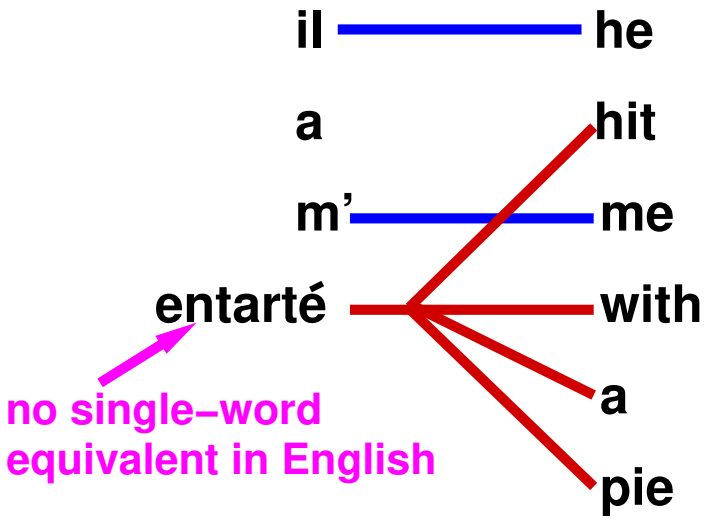
By Oxyman - Own work, CC BY 2.5

https://commons.wikimedia.org/w/index.php?curid=3225840

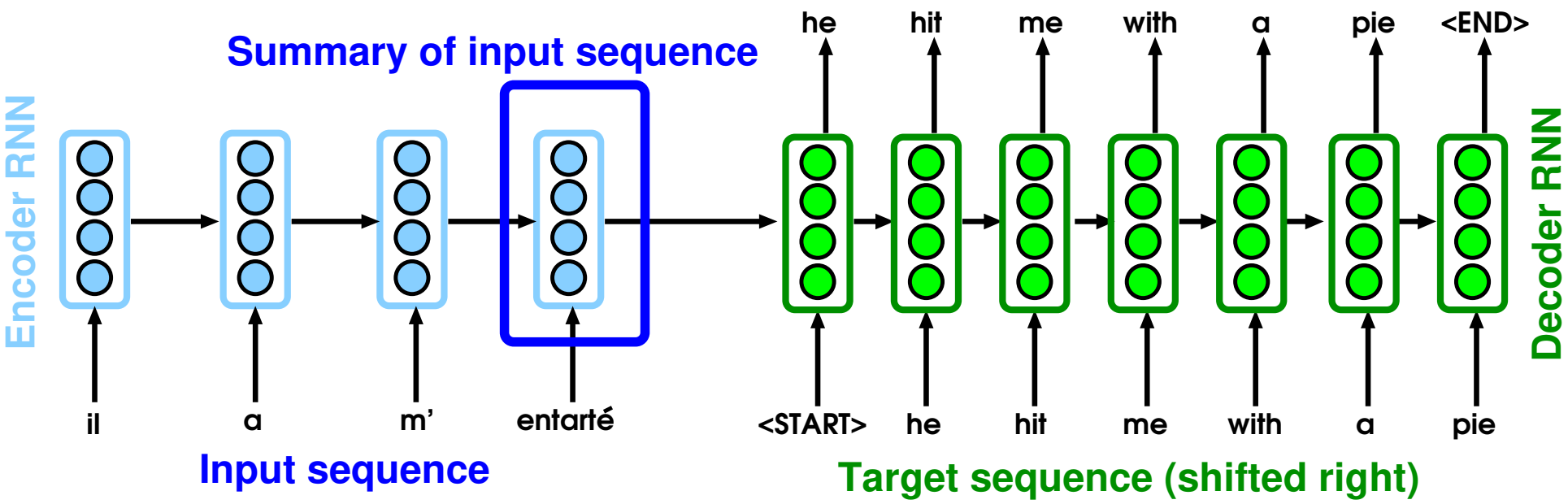
https://medium.com/@samim/obama-rnn-machine-generated-political-speeches-c8abd18a2ea0

https://medium.com/deep-writing/harry-potter-written-by-artificial-intelligence-8a9431803da6

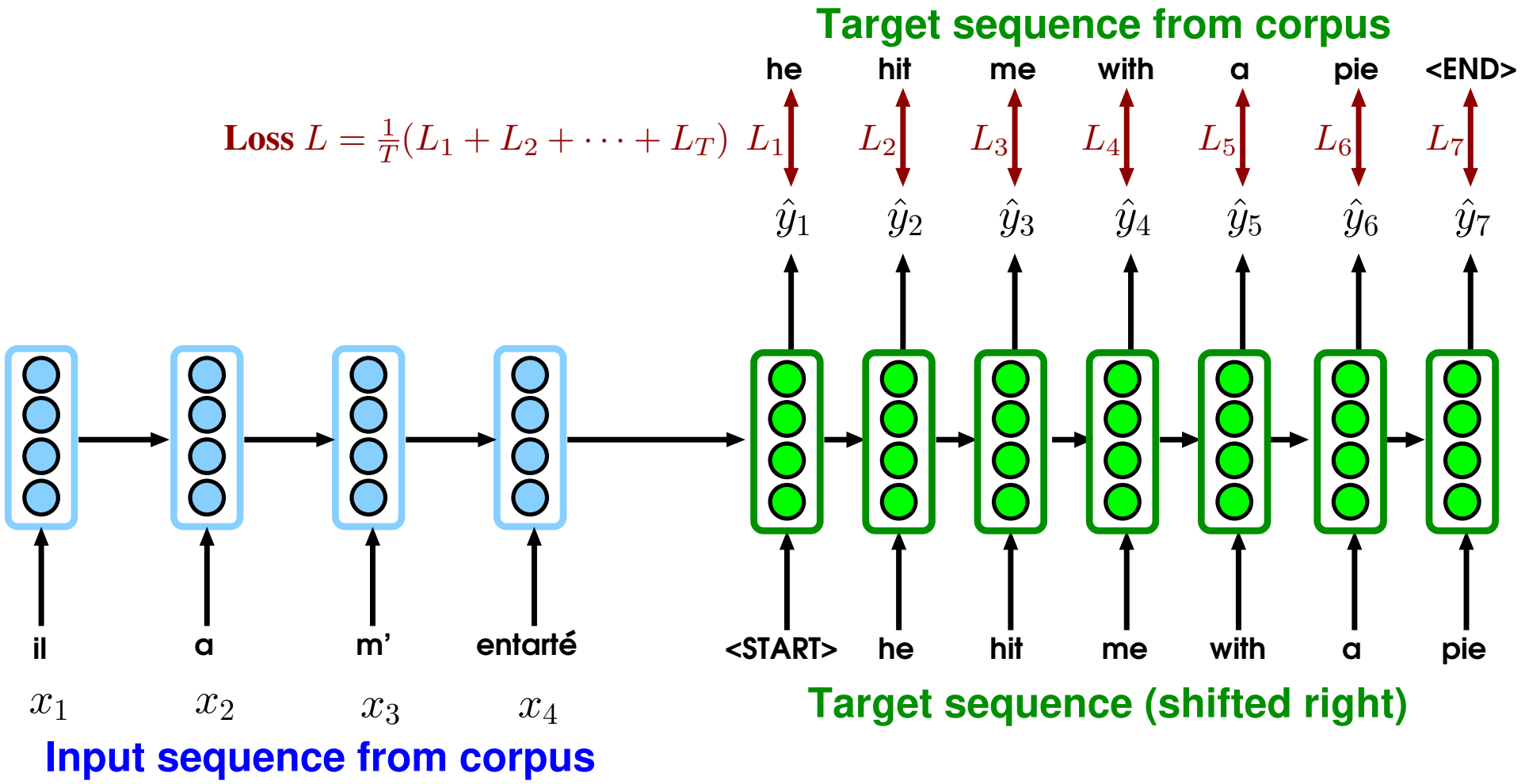
Neural Encoder/Decoder Models for Machine Translation



https://commons.wikimedia.org/w/index.php?curid=9105527

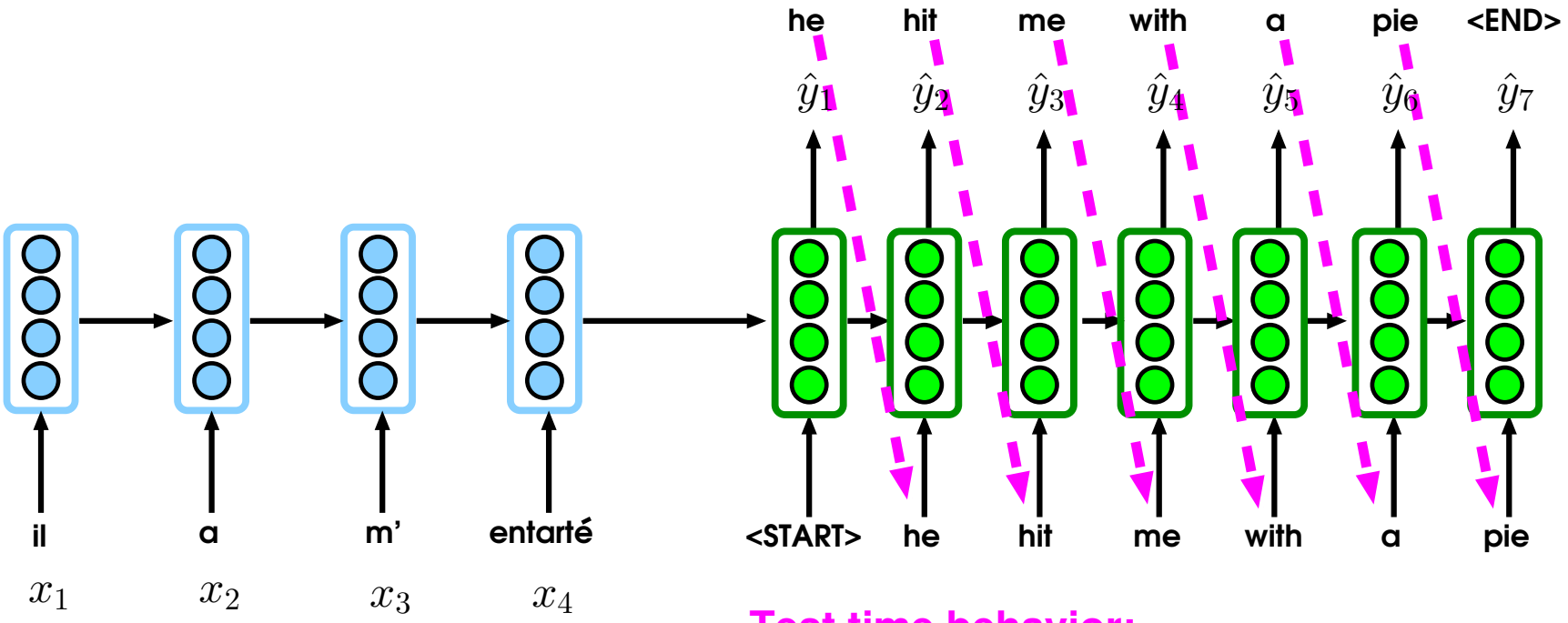


Sequence-to-sequence: Training



Get a **big parallel corpus** containing input/target sequence pairs!

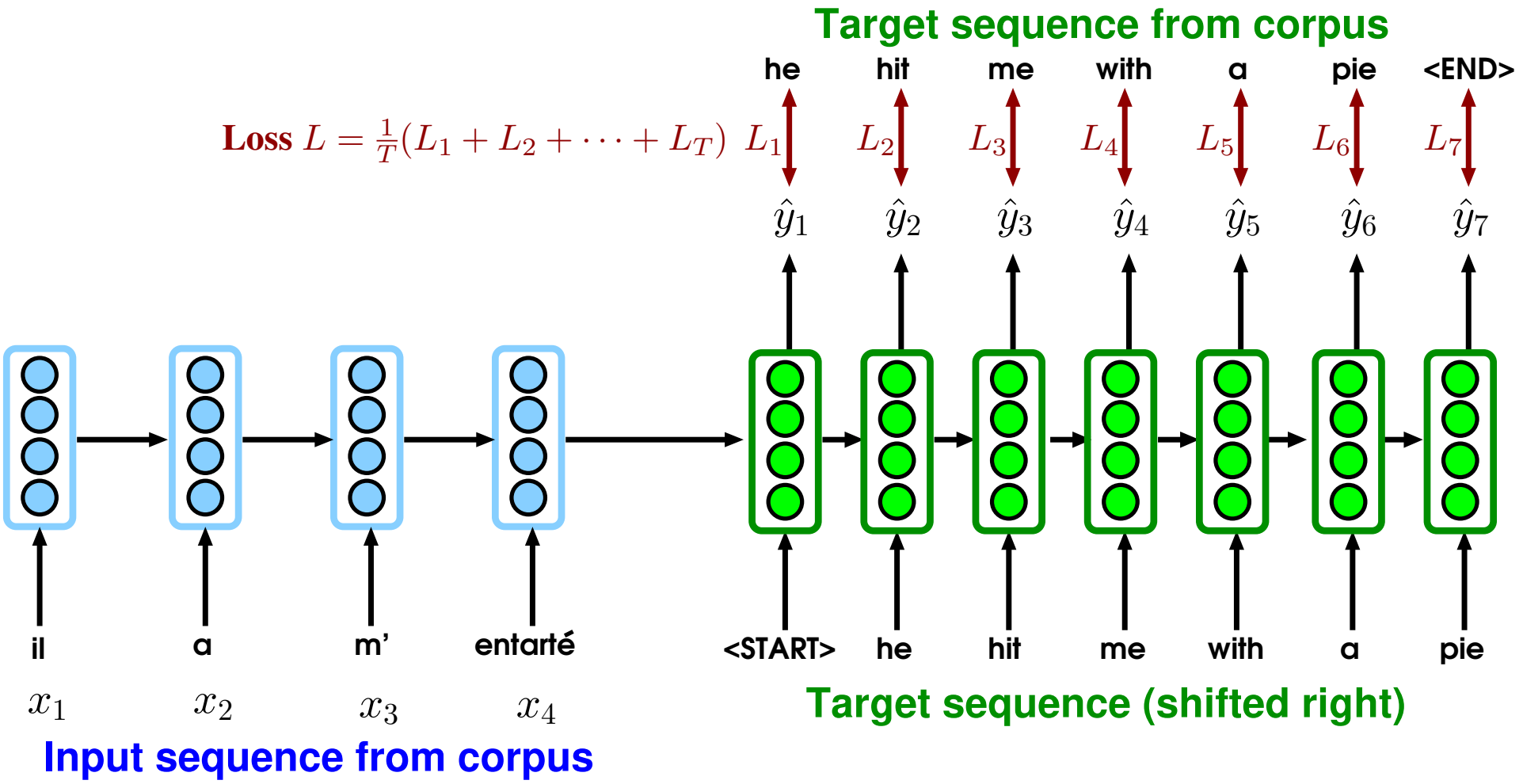
Sequence-to-sequence: Test time behavior



Input sequence from corpus

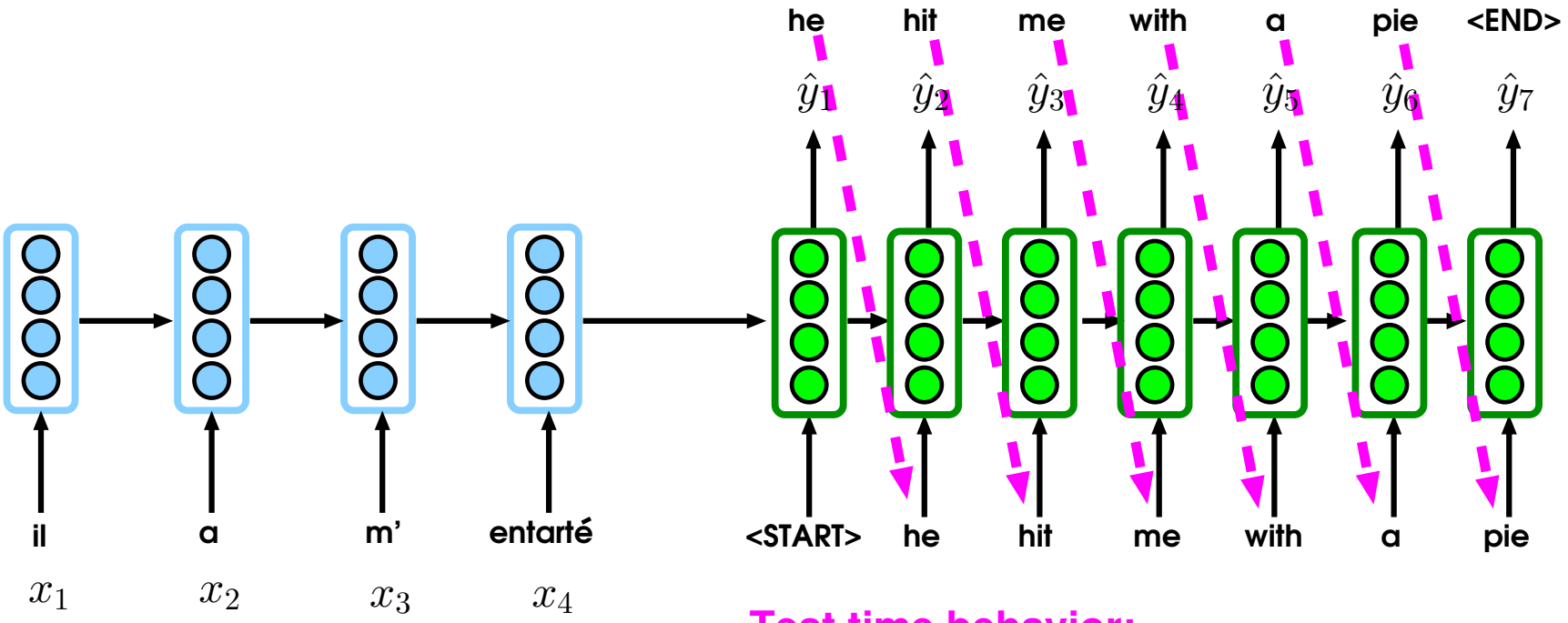
Test time behavior:
Last decoder output used as next step's input

Sequence-to-sequence: Training



Get a **big parallel corpus** containing input/target sequence pairs!

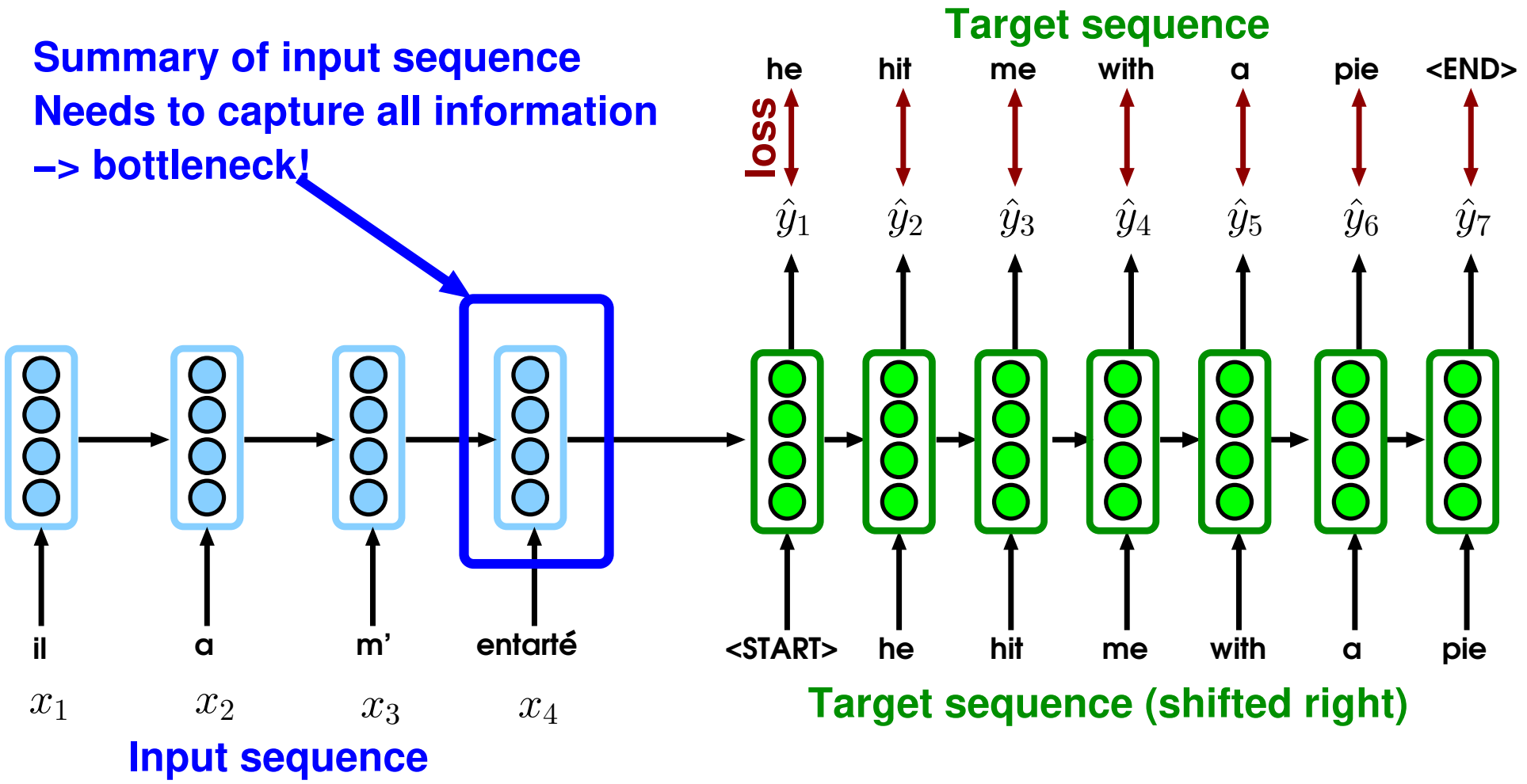
Sequence-to-sequence: Test time behavior



Input sequence from corpus

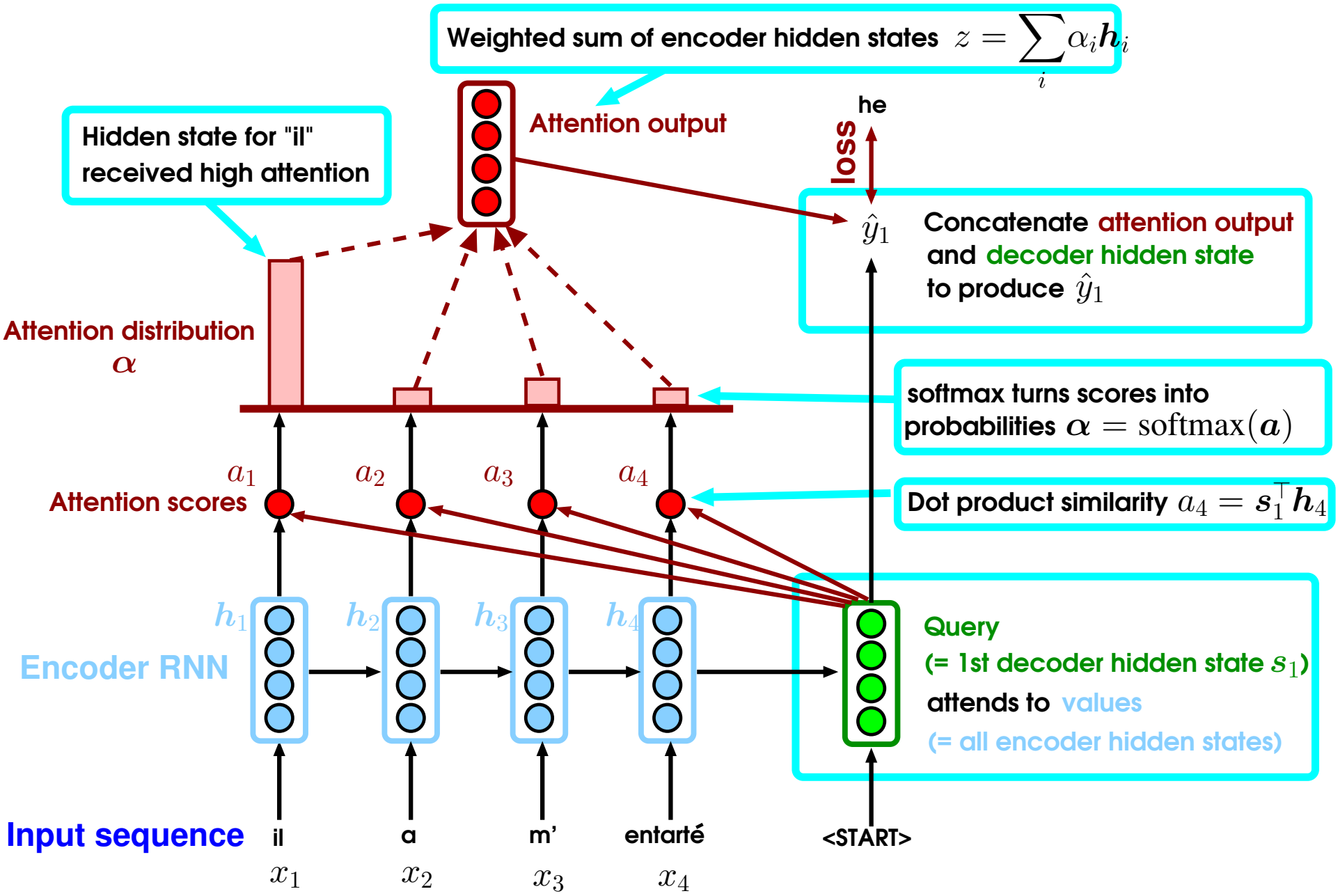
Test time behavior:
Last decoder output used as next step's input

Sequence-to-sequence: bottleneck problem

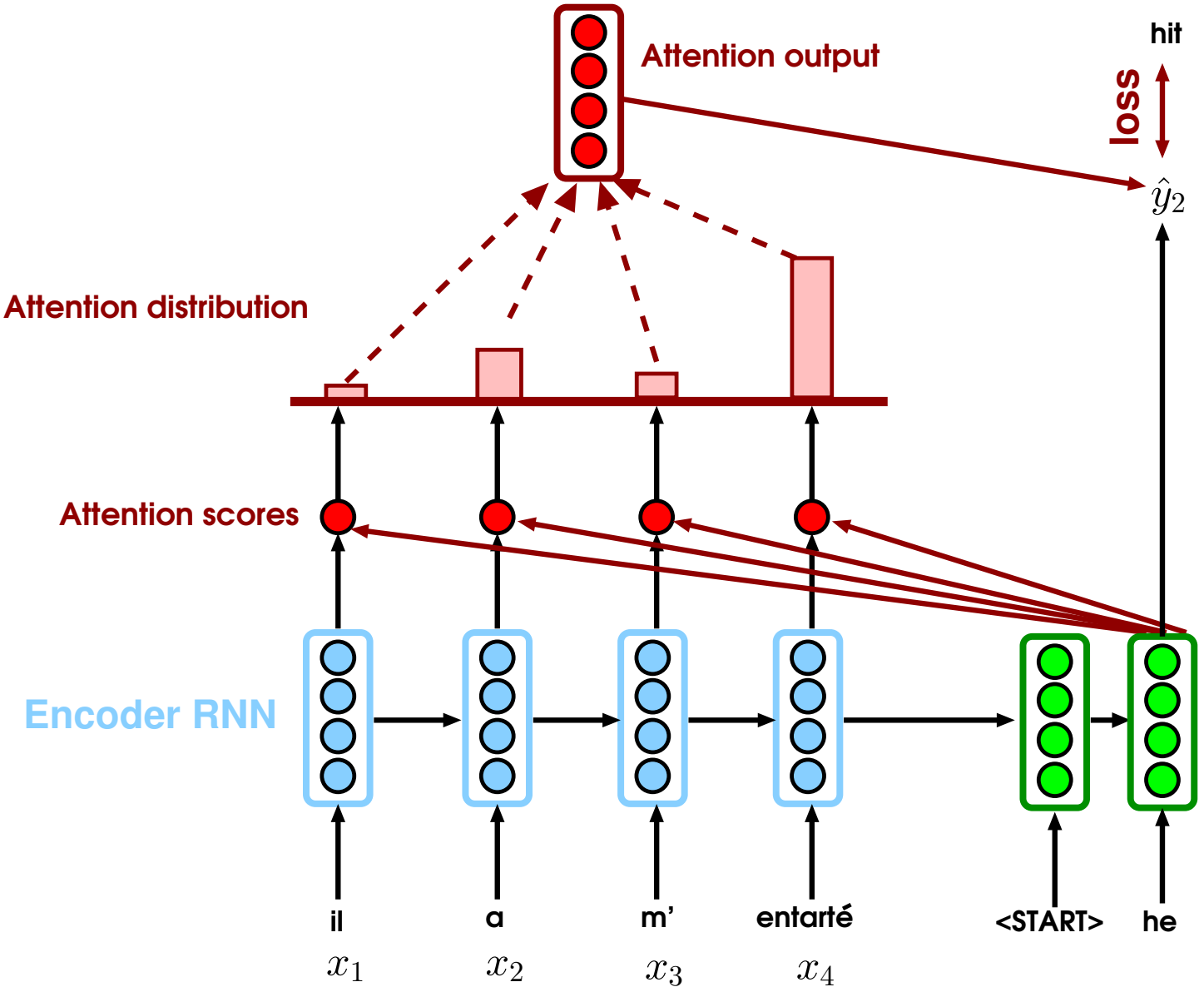


Idea: allow the decoder to look directly at input, bypass bottleneck.

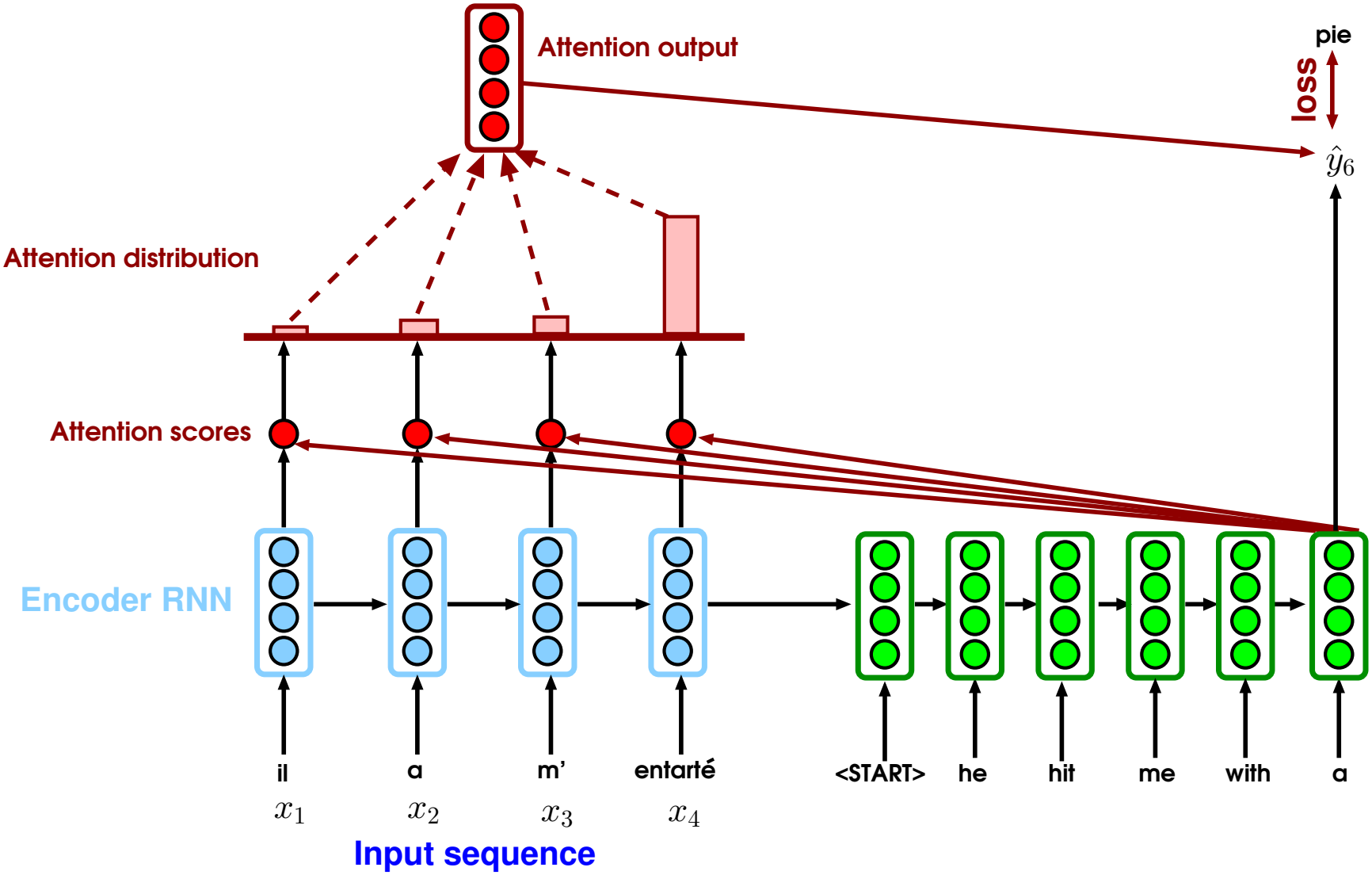
Sequence-to-sequence with attention



Sequence-to-sequence with attention



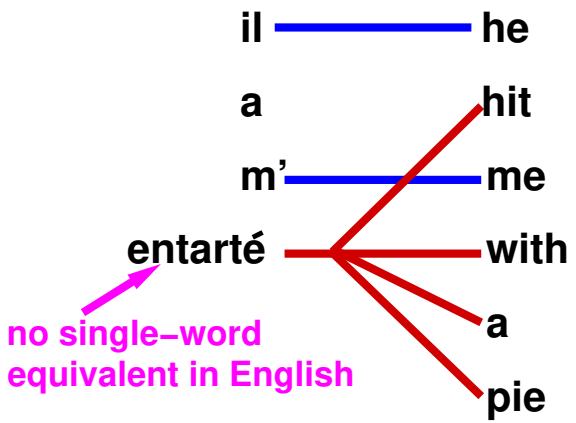
Sequence-to-sequence with attention



Attention is great

- Attention solves the bottleneck problem: It allows decoder to look directly at the input sequence, bypass bottleneck
- Attention helps with vanishing gradient problem: Provides shortcut to faraway states

- Interpretability: Attention distribution provides (probabilistic) word alignments for free!



- We never explicitly trained an alignment system, the network learned it by itself!

- **Attention is also the main builing block of transformers:** We "transform" queries s_t to attention outputs z_t, conditioned on inputs and previous queries.

Attention: General setting

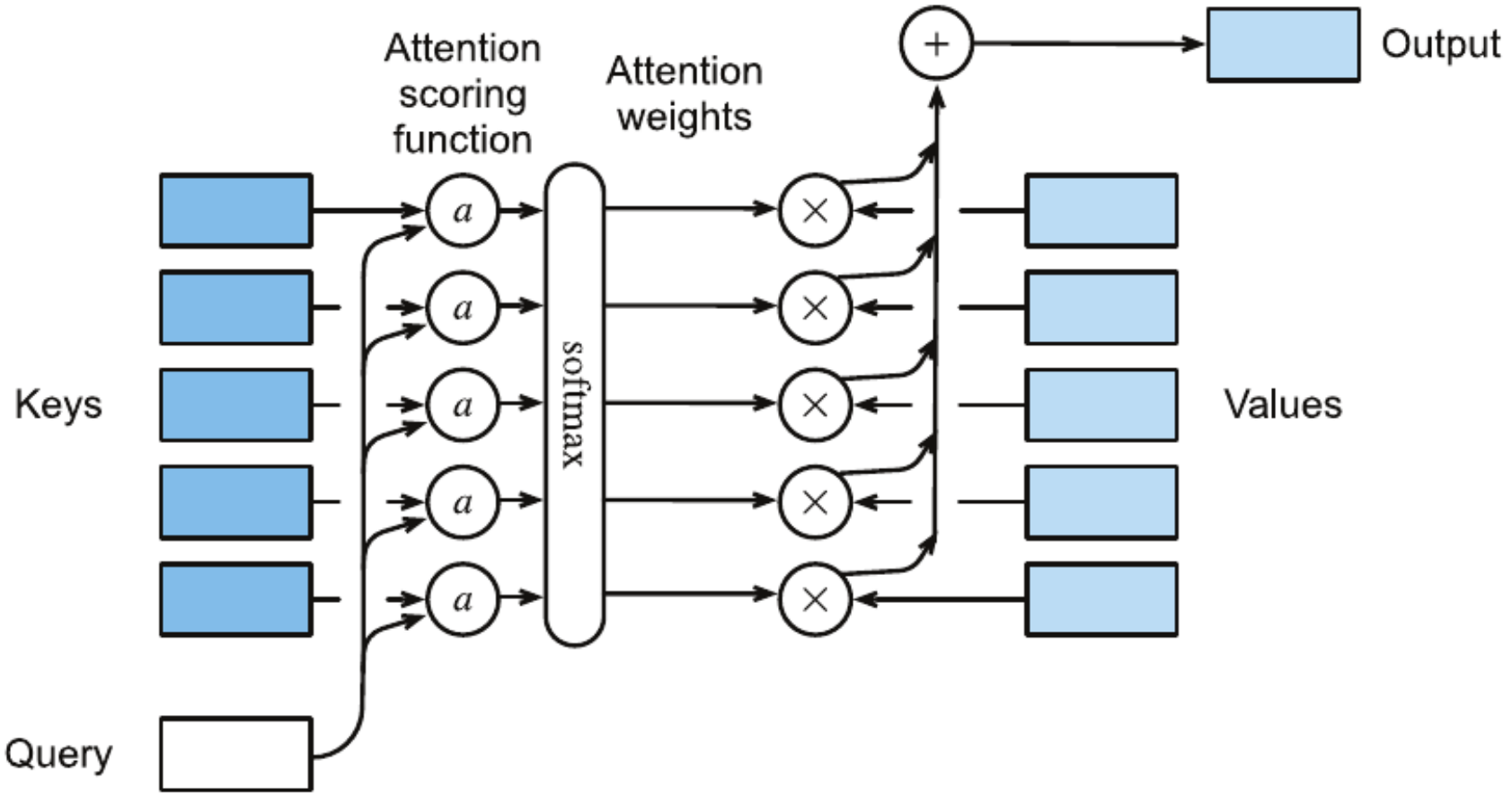
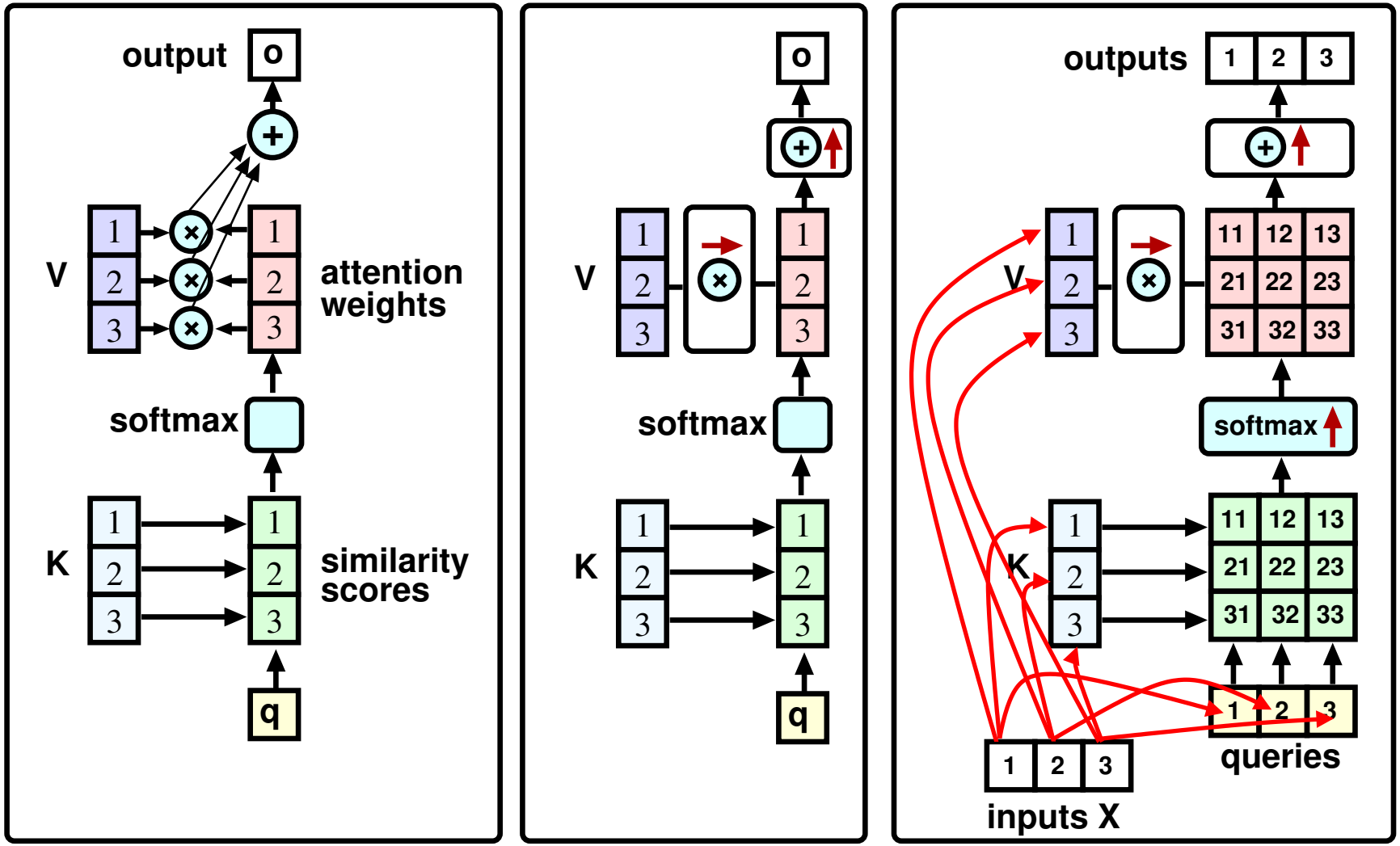


Figure 16.6 in K.Murphy: Probabilistic Machine Learning, Advanced Topics. MIT Press, 2023. Attention layer. (a) Mapping a single query q to a single output, given a set of keys and values.

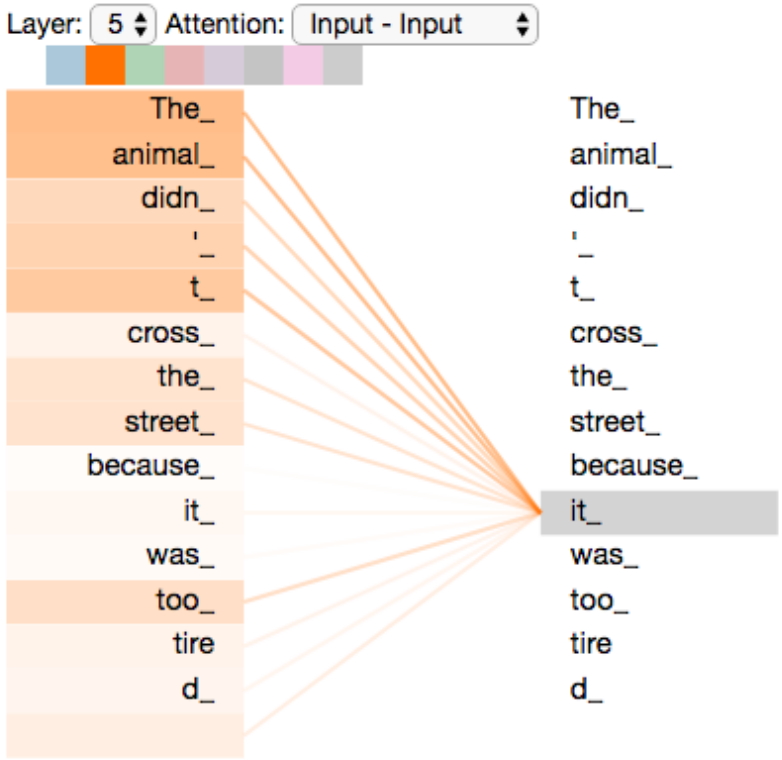
Attention and Self-Attention



Left: Mapping a single query q to a single output o, given a set of keys and values. Middel: simplified notation. Right: Mapping multiple queries to multiple outputs, either for given values and keys (without the red arrows and without inputs X), or in the self-attention case, where queries, values and keys are functions of inputs X (red arrows) .

Attention in Language Models

The animal didn't cross the street because it was too tired



https://jalammar.github.io/illustrated-transformer/

Attention distribution provides word alignments for free!

Attention is also the main building block of transformers.

"Transformer"-language models

- RNNs process one token at a time $\rightsquigarrow$ representation of word at t depends on hidden state s_t (summary of previous words).
- Alternative approach: use attention to compute representation directly as a function of all other words.
- This is the idea of a **encoder-only transformer**, used by LMs such as BERT (Bidirectional Encoder Representations from Transformers).

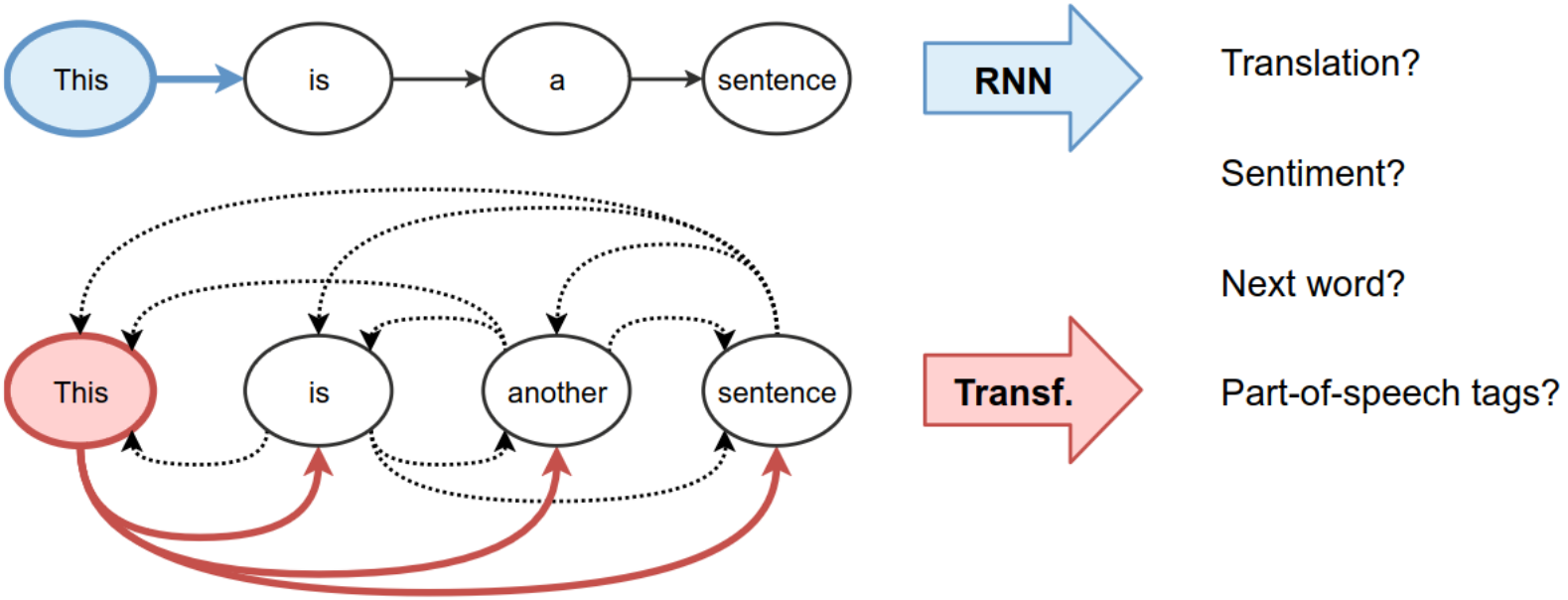


Fig. 16.16 in K.Murphy: Probabilistic Machine Learning, Advanced Topics. MIT Press, 2023. Original Image published in C. Joshi. Transformers are Graph Neural Networks. Tech. rep. 2020.

“Transformer”-language models

- Alternative: **Decoder-only transformer:** each output y_t only attends to all previously generated outputs, $y_{1:(t-1)}$.
- This can be implemented using **masked self-attention**, and is useful for **generative language models**, such as GPT.
- Combination: **Sequence-to-sequence models**, $p(y_{1:T_y}|x_{1:T_x})$.

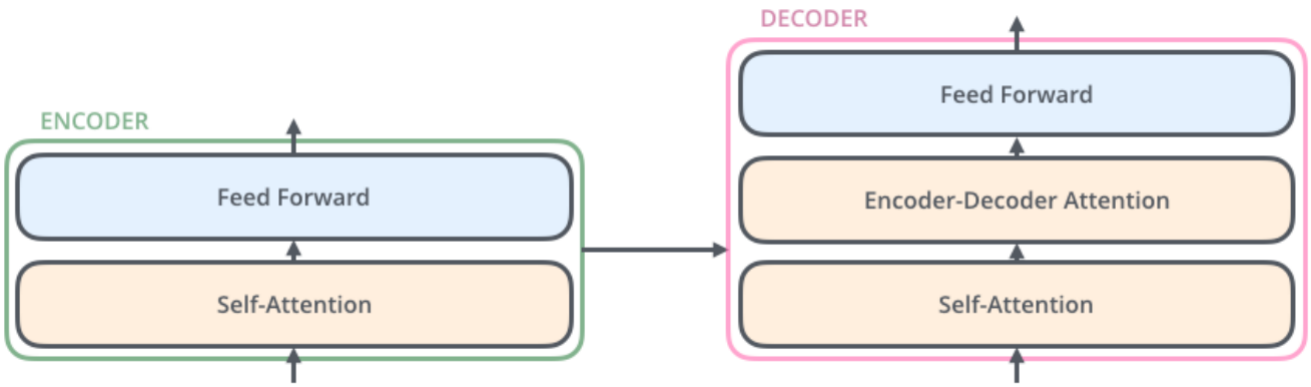
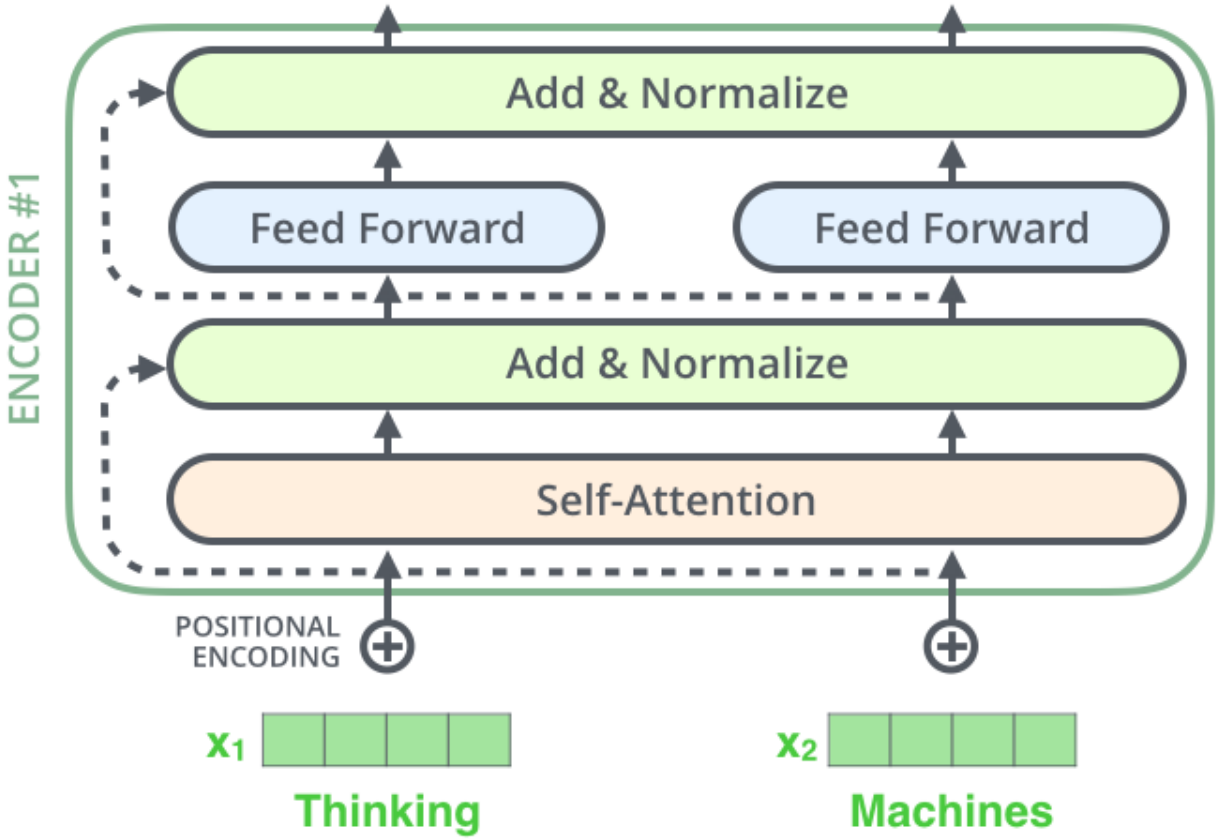


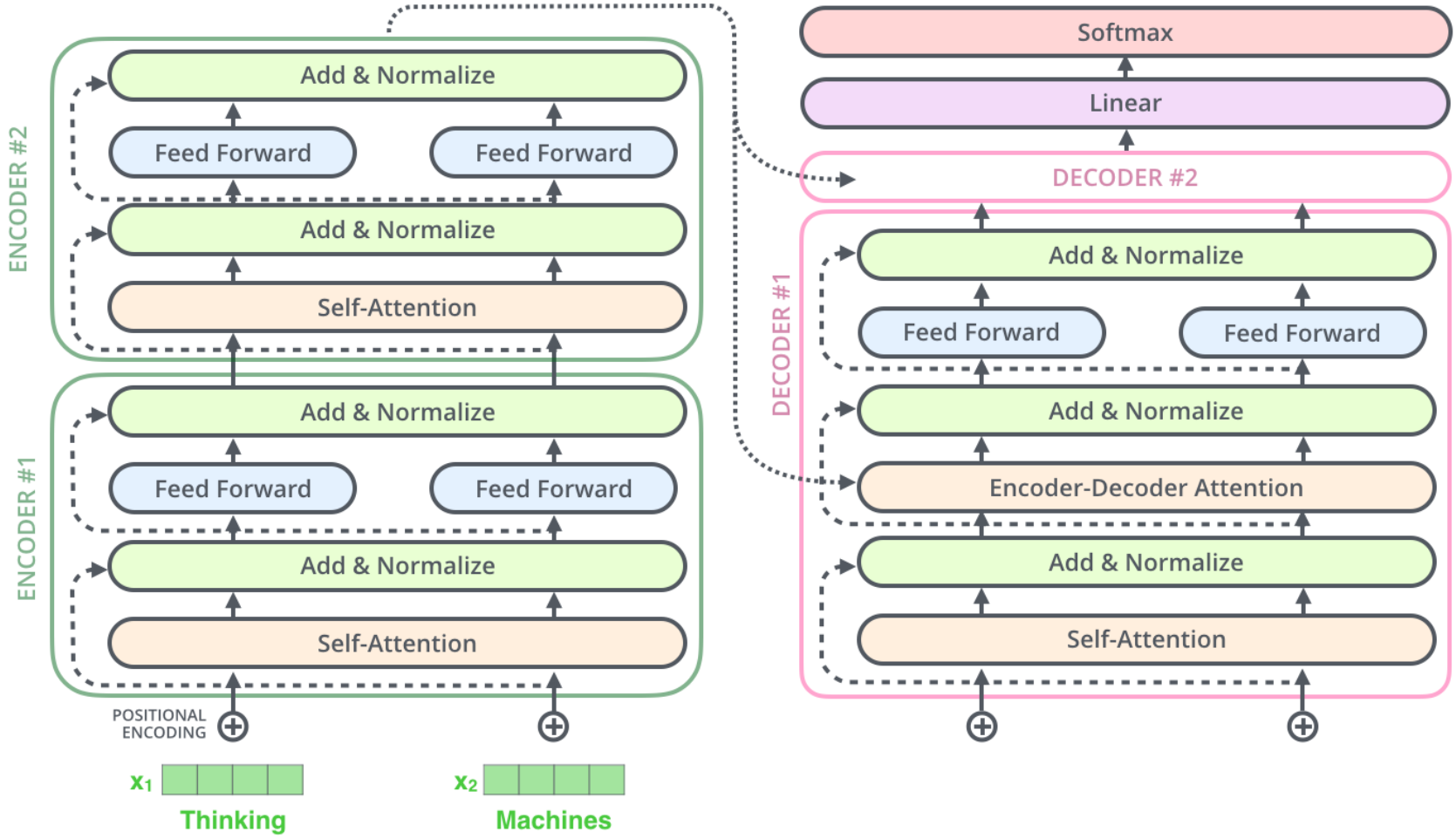
Figure 16.1 in the supplement of K.Murphy: Probabilistic Machine Learning, Advanced Topics. MIT Press, 2023. High level structure of the encoder-decoder transformer architecture. https://jalammar.github.io/illustrated-transformer/

Transformer: Encoder



https://jalammar.github.io/illustrated-transformer/

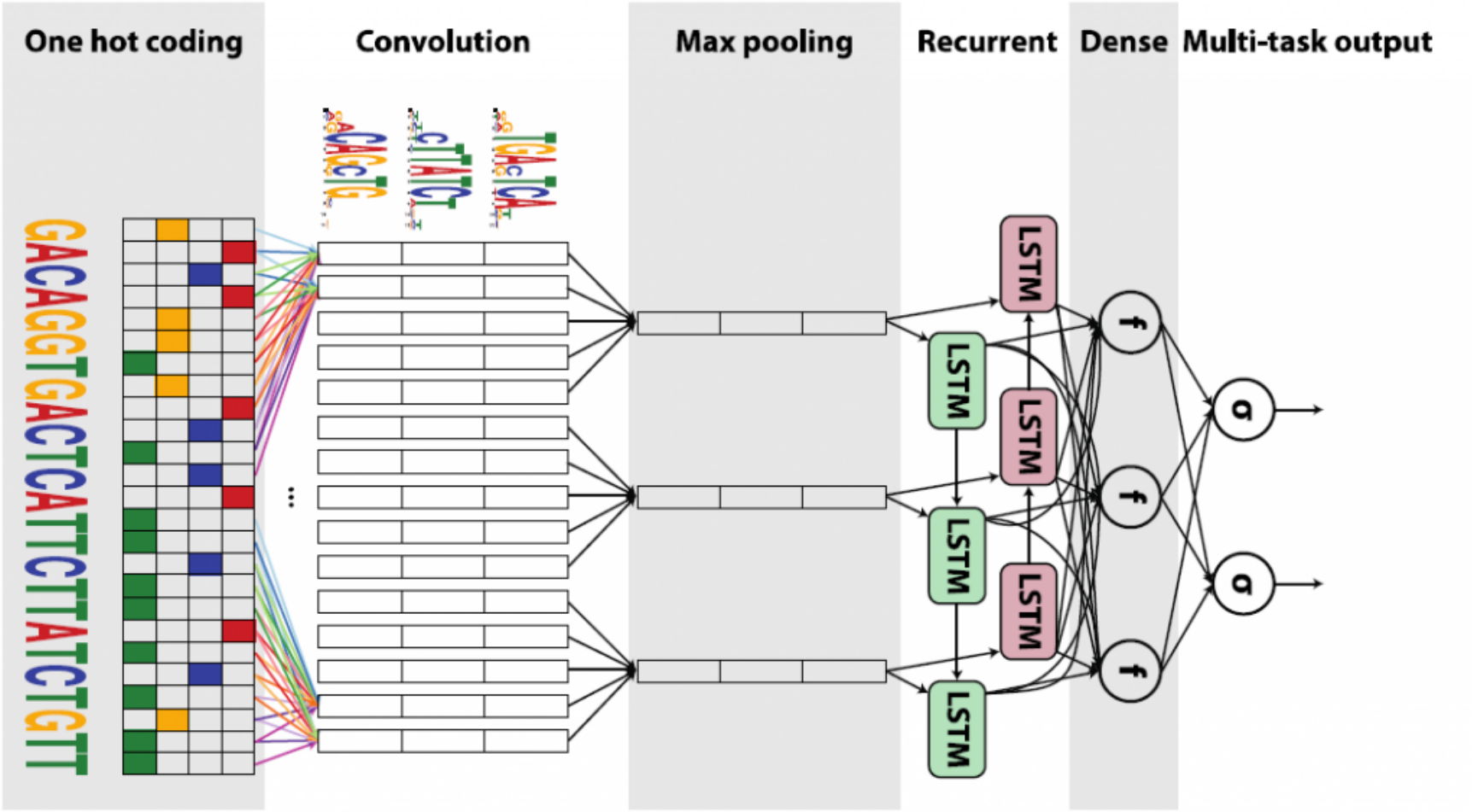
Seq2Seq with Transformers



https://jalammar.github.io/illustrated-transformer/

Example applications

DanQ: A hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences



D. Quang, X. Xie. Nucleic Acids Res. 2016 Jun 20;44(11):e107. doi: 10.1093/nar/gkw226.

DanQ: A hybrid CNN/RNN

- Goal: predict function directly from sequence, instead of from curated datasets (gene models, multiple alignments).
- $> 98\%$ of the human genome is non-coding. GWAS identified 6500 disease-related single-nucleotide polymorphisms (SNPs), 93% in non-coding regions.
- Example: Introns are non-coding sections of genes. Some have significant biological function (such as regulating RNA activity).

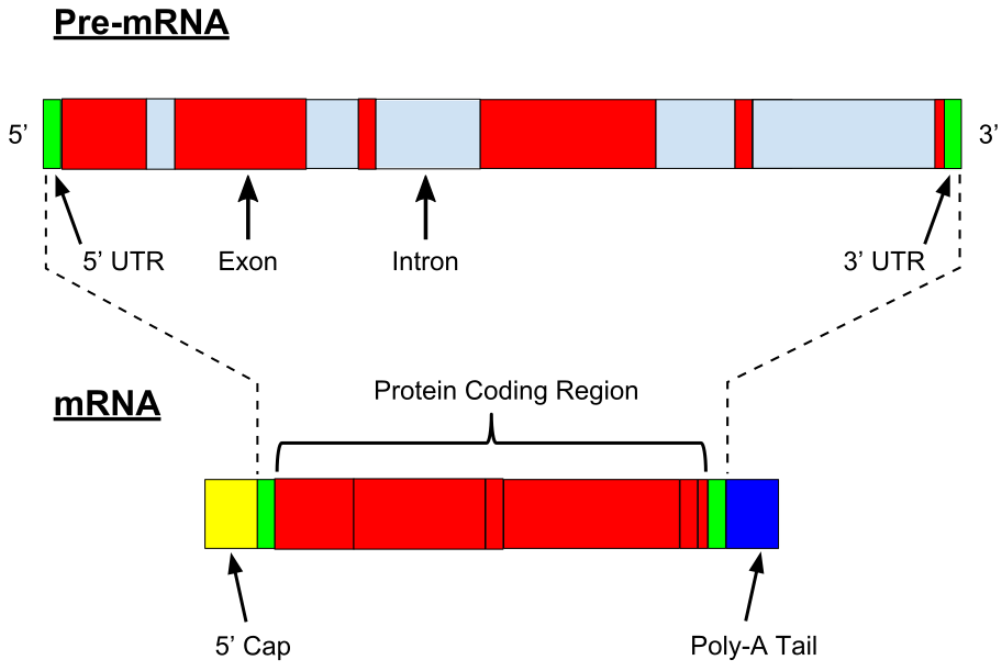
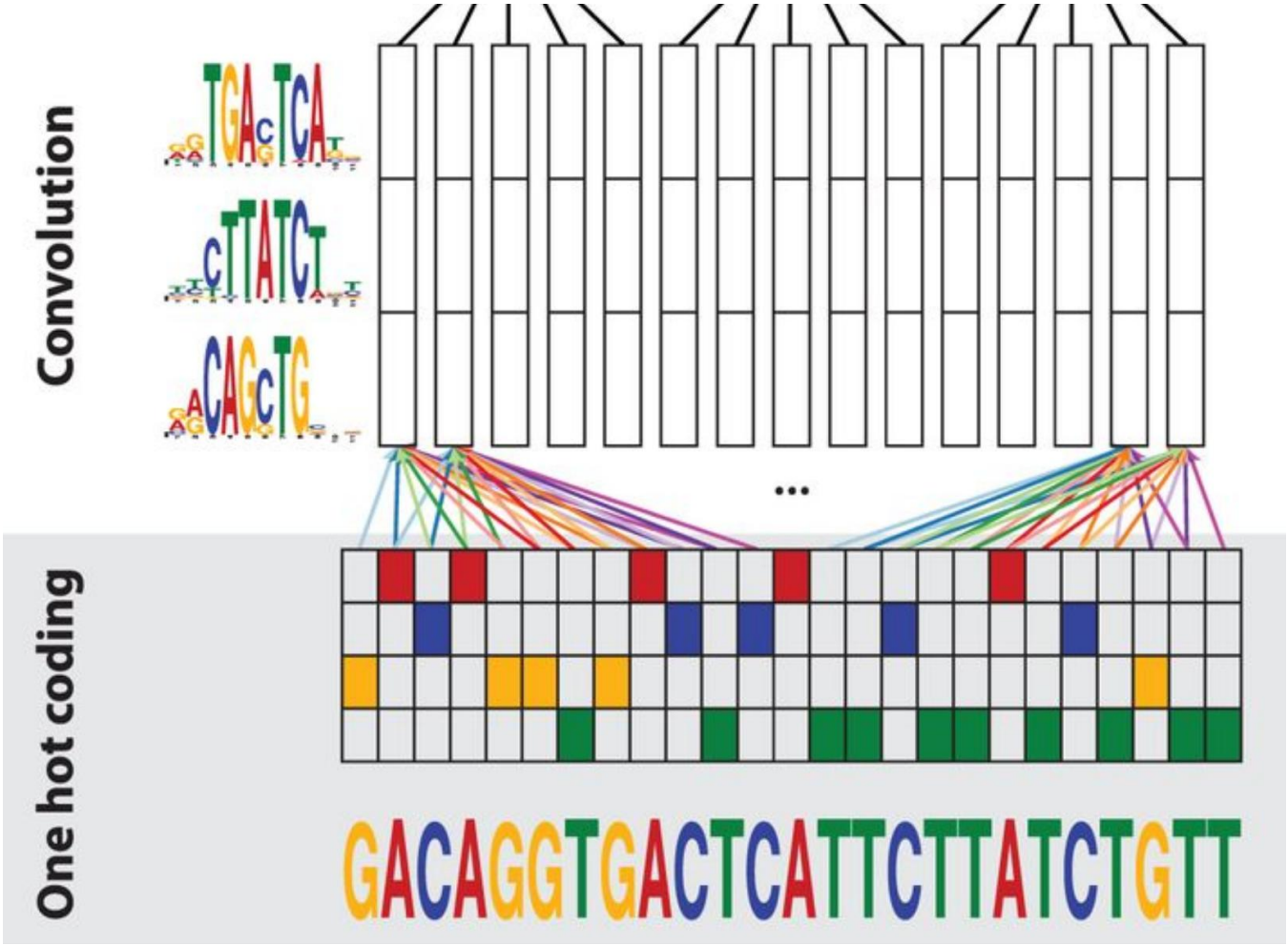


Illustration of an unspliced pre-mRNA precursor, with five introns and six exons (top). After the introns have been removed via splicing, the mature mRNA sequence is ready for translation (bottom). By Nastypatty - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=49282051

DanQ: A hybrid CNN/RNN

First layers: designed to scan sequences for motif sites through convolution filtering.



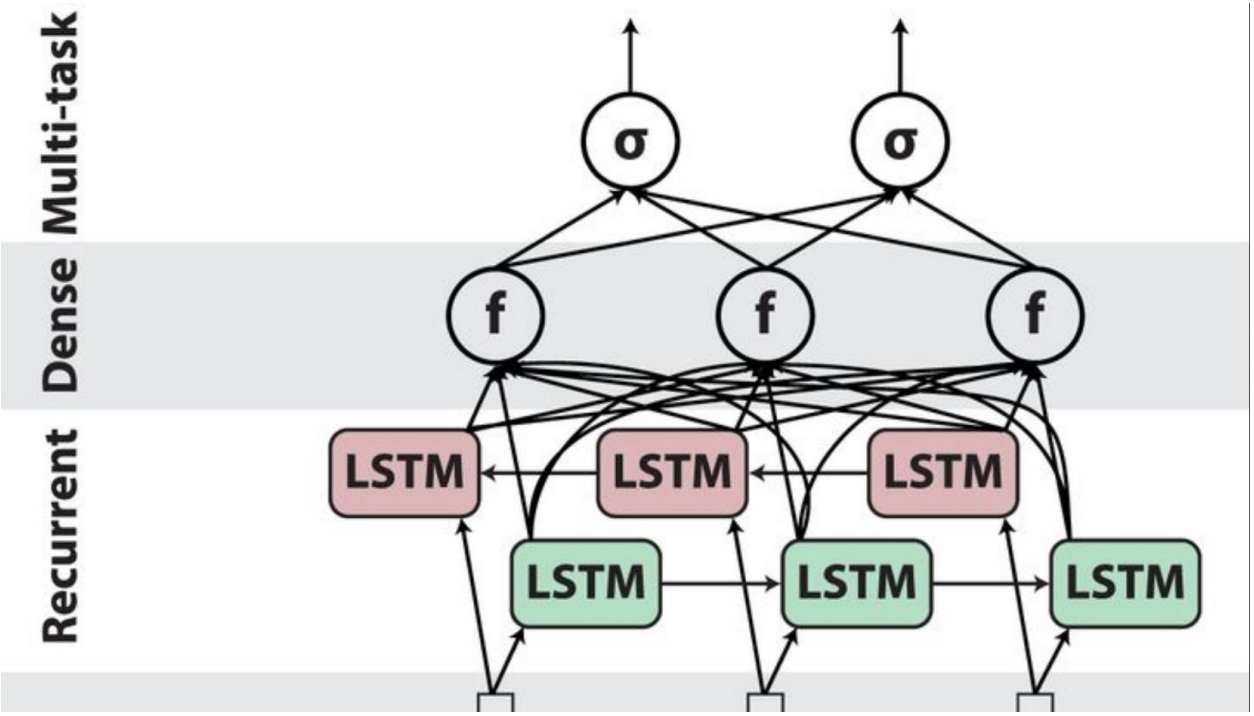
D. Quang, X. Xie. Nucleic Acids Res. 2016 Jun 20;44(11):e107. doi: 10.1093/nar/gkw226.

DanQ: A hybrid CNN/RNN

First layers: Scannning sequences for motif sites.

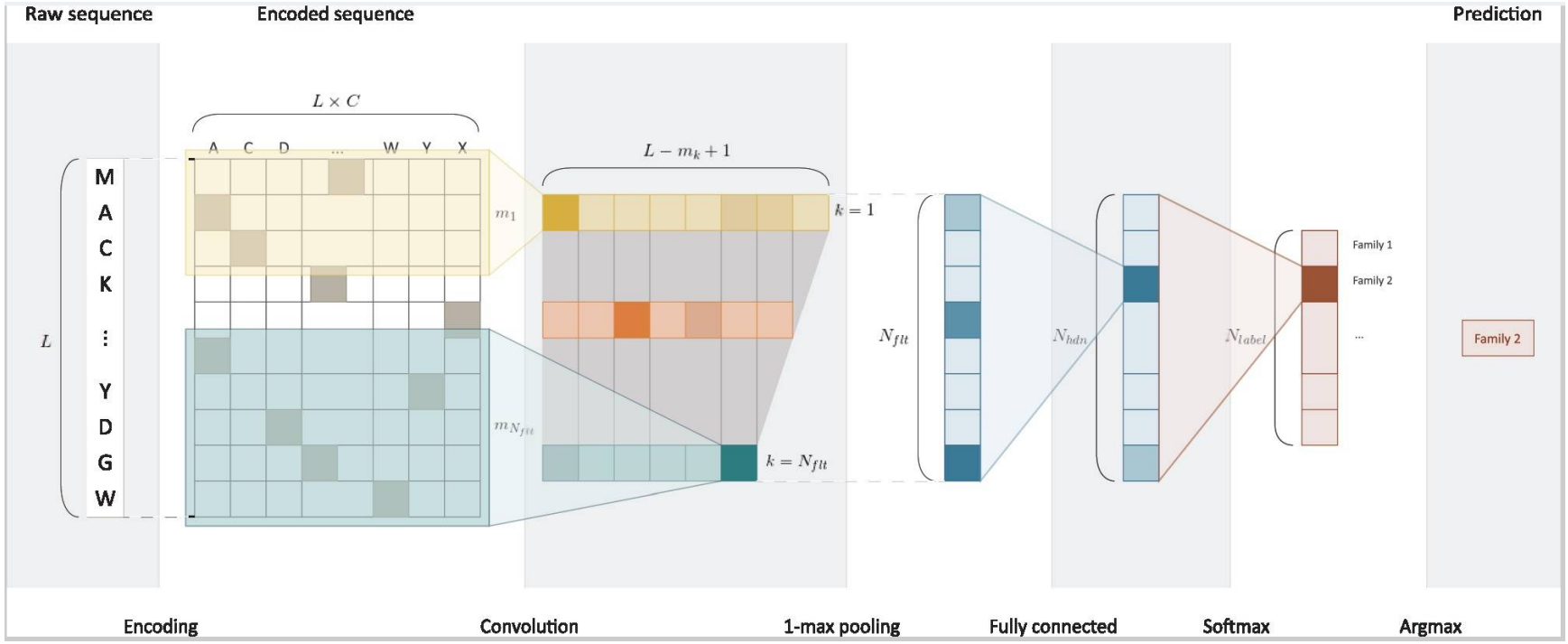
Bi-directional LSTM layer: Modelling spatial arrangements and frequencies of combinations of motifs.

Hybrid CNN-RNN architecture $\rightsquigarrow$ simultaneous learning of motifs and sequential arrangement of motifs.



D. Quang, X. Xie. Nucleic Acids Res. 2016 Jun 20;44(11):e107. doi: 10.1093/nar/gkw226.

DeepFam: Deep learning based alignment-free method for protein family modeling and prediction



S. Seo et al. Bioinformatics. 2018 Jul 1; 34(13): i254–i262. doi: 10.1093/bioinformatics/bty275

DeepFam: Motivation

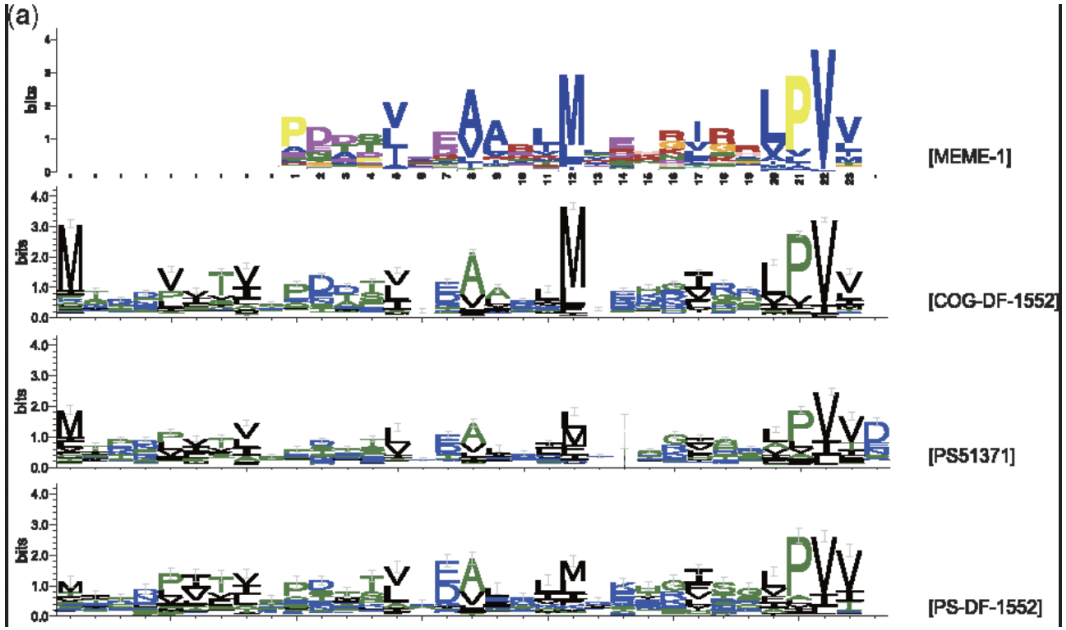
- Most widely adopted method to compare two protein sequences: Smith-Waterman algorithm
- Works less well for distant sequences.
- Protein family approaches are better. Two variants:
  - **Alignment-based: profile Hidden Markov Models.**
    Position-specific scoring + explicit insertion/deletion states.
  - **Alignment-free protein family modeling.**
    Usually based on frequencies of **k-mers.**
    Example: ATGG has two 3-mers: ATG and TGG.

```
 TGG
ATGC
ATG
```

    Problems: Order information is lost, requires exact matches.
    But: In some applications, better performance.

DeepFam: Architecture

- First layers: designed to scan sequences for motif sites through convolution filtering.
- Convolutional units: Position-specific, functioning similarly as position-specific scoring matrix in MSA.
- Interpretability: In experiments, convolution units correspond to well-known protein motifs.

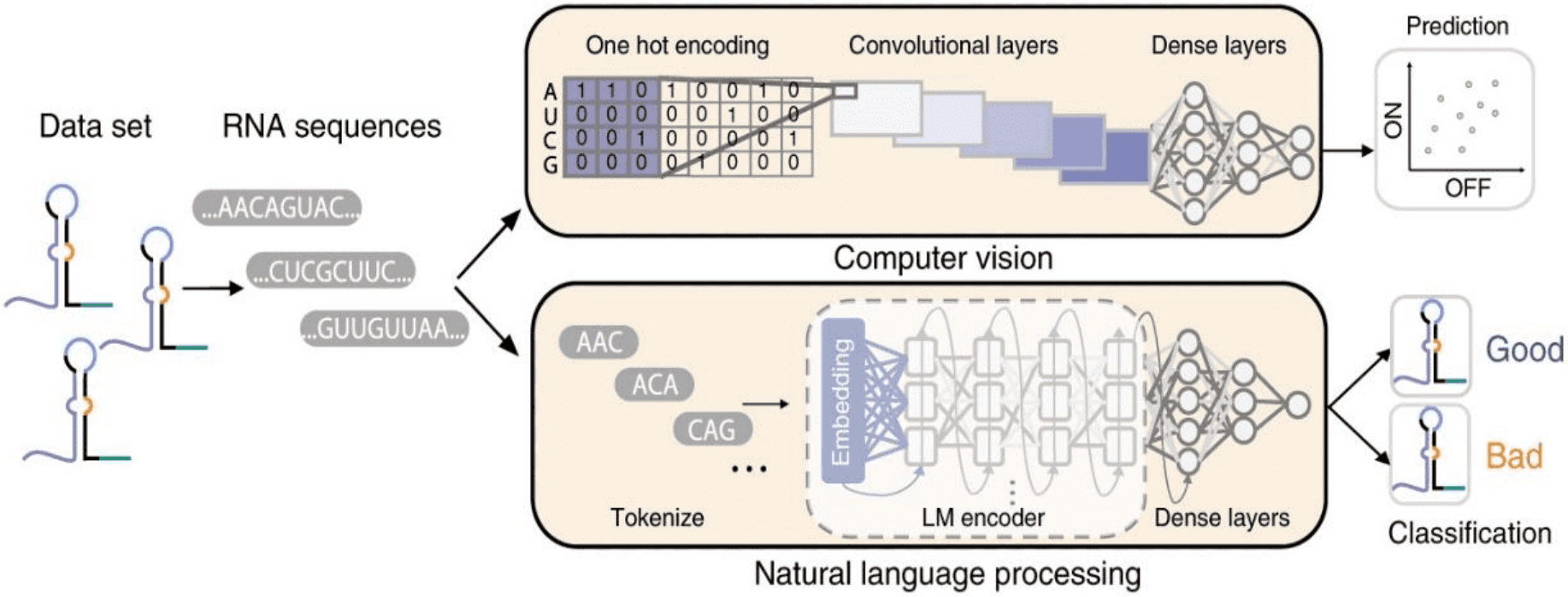


S. Seo et al. Bioinformatics. 2018 Jul 1; 34(13): i254–i262. doi: 10.1093/bioinformatics/bty275

Sequence-to-function deep learning

- Predicting phenotypes from genotypes: the aim is to both predict and classify **toehold switch performance (phenotype)** from **RNA sequences (genotype).**
- **Toehold switches** are RNA molecules of increased interest because they act as programmable sensors for precision diagnostics.
- One-hot encoding turns RNA into a $4 \times L$ tensor, where L is the length of the RNA sequence.
- For the prediction task, **CNNs** are used.
- On the other hand, the RNA sequences are tokenized into 3-mers and fed into a **LSTM-RNN** for binary classification.

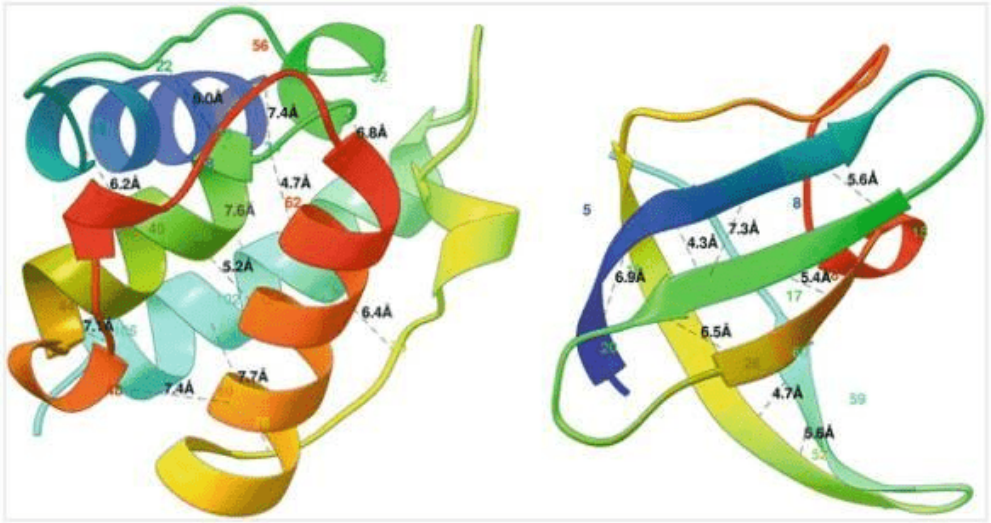
Sequence-to-function deep learning

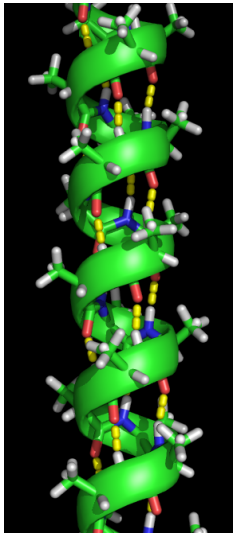


Sequence-to-function deep learning frameworks for engineered riboregulators, Valeri et al, Nat Commun. 2020 Oct 7;11(1):5058. doi: 10.1038/s41467-020-18676-2.

Protein 3D structure prediction

- Formally, protein structure prediction is the inference of the 3D structure of a protein from its amino acid sequence (input).
- In biological terms, the 3D structure is the secondary and tertiary structure (output).
- One can formulate the problem as a 3D contact prediction.



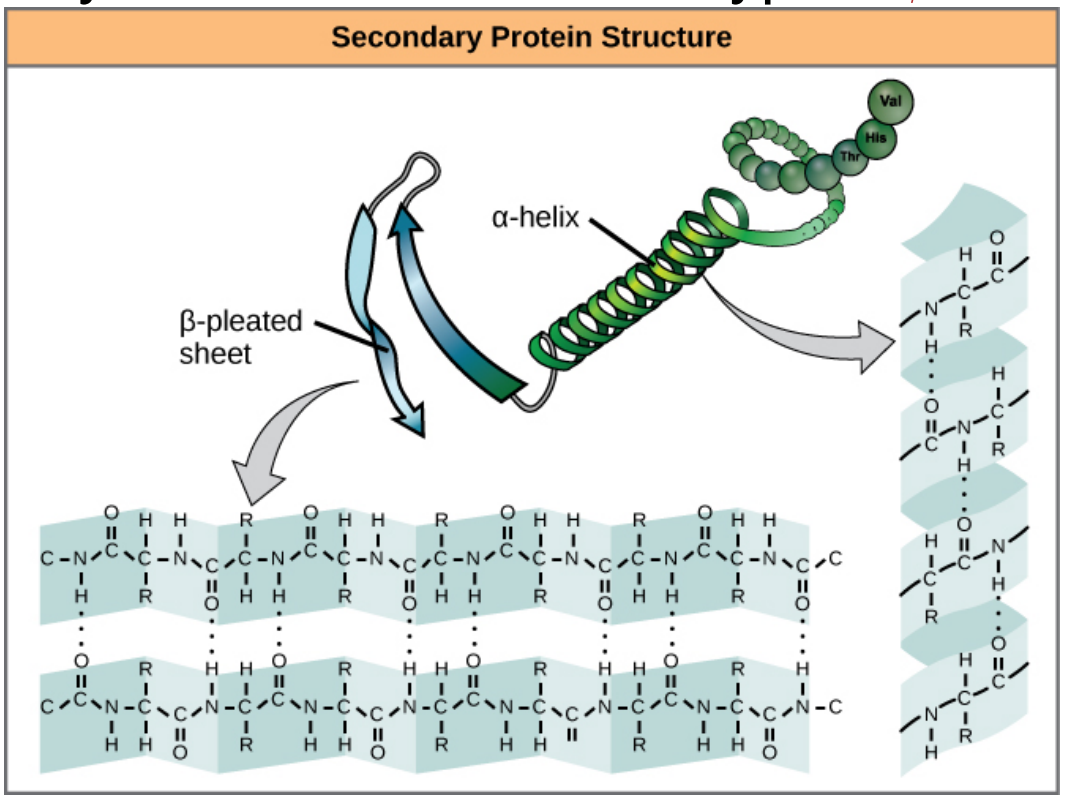


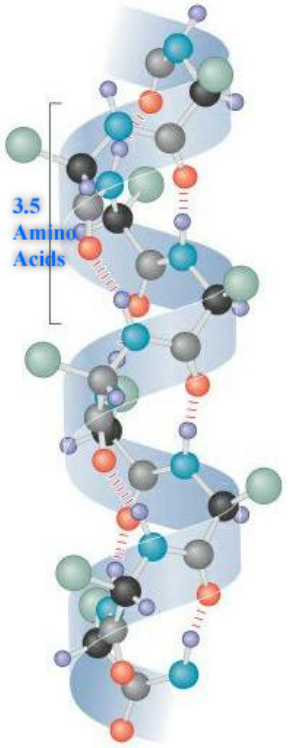
Left: Two globular proteins with some contacts in them shown in black dotted lines along with the contact distance in Armstrong. The alpha helical protein (left) has many long-range contacts, the beta sheet protein (right) has more short-/medium-range contacts. Source: Protein Residue Contacts and Prediction Methods.

Right: By en:User:Bikadi, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=16236442

Secondary Structure Prediction

Secondary structure: two main types: *β-sheet* and *α-helix*





The School of Biomedical Sciences Wiki

Short range interactions in the AA chain are important for the secondary structure: α-helix performs a $100°$ turn *per amino acid* $\rightsquigarrow$ full turn after 3.6 AAs.

Formation of a helix mainly depends on interactions in a *4 AA* window.

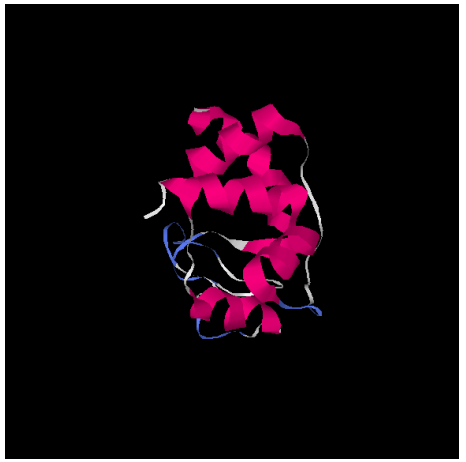
Example: Cytochrome C2 Precursor

```
Secondary structure (h=helix)
amino acid sequence

hhhhhhhhhhh
MKKGFLAAGVFAAVAFASGAALAEGDAAAGEKVSKKCLACHTFDQGGANKVGPNLFGVFE
hhhhhhhh hhhhhhhhh hhhhhhhhh
NTAAHKDDYAYSESYTEMKAKGLTWTEANLAAYVKDPKAFVLEKSGDPKAKSKMTFKLTK

hhhhhhhhhhhhh

DDEIENVIAYLKTLK
```



Given: Examples of known helices and non-helices in several proteins

$\rightsquigarrow$ **training set**

Goal: Predict, mathematically, the existence and position of α-helices in **new proteins.**

Classification of Secondary Structure

Idea: Use a **sliding window** to cut the AA chain into pieces. 4 AAs are enough to capture one full turn $\rightsquigarrow$ choose window of size 5.

Decision Problem: Find function $f(\dots)$ that predicts for each substring in a window the structure:

$$f(\texttt{AADTG}) = \begin{cases} \text{"Yes"}, & \text{if the central AA belongs to an } \alpha\text{-helix,} \\ \text{"No"}, & \text{otherwise} \end{cases}$$

Problem: How should we numerically encode a string like AADTG?

Simple encoding scheme: Count the number of occurrences of each AA in the window. First order approximation, neglects AA's position within the window.

Example

...RAADTGGSDP...

...xxxhhhhhhx...

...xxxhhhhhhx...

...xxxhhhhhhx...

(black ≙ structure info about central AA; green ≙ know secondary structure; red≙ sliding window)

A	C	D	...	G	...	R	S	T	...	Y	Label
2	0	1	0	0	0	1	0	1	0	0	"No"
2	0	1	0	1	0	0	0	1	0	0	"Yes"
1	0	1	0	2	0	0	0	1	0	0	"Yes"
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

This is a standard **binary classification problem.**

Alphafold 2 High-level overview

The Nature article diagram that outlines the different pieces of the architecture.

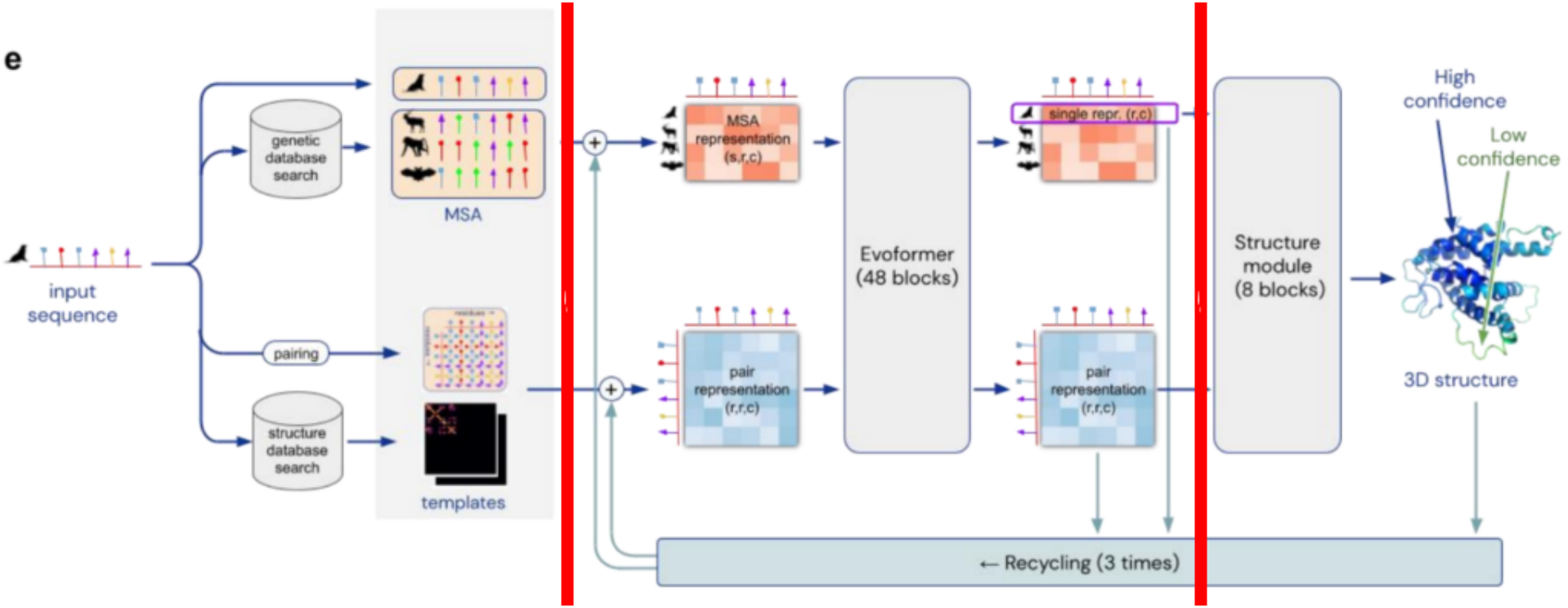


Diagram of AlphaFold 2 as published in the official Nature paper in July 2021. The added red lines divided the image into thirds which represent the three main parts.

Alphafold 2 High-level overview

The preprocessing module

- Input amino acid sequence is used to query several protein sequence databases $\rightsquigarrow$ **jackHMMER** $\rightsquigarrow$ **MSA.**
- MSA enables the determination of the parts of the sequence that are more likely to mutate, and allows us to detect correlations between them.
- AlphaFold 2 also tries to identify proteins that may have a similar structure to the input **(**$\rightsquigarrow$**templates)**, and constructs an **initial representation of the structure**, which it calls the **pair representation:** a model of which AAs are likely to be in contact with each other.

The transformer module

- AlphaFold 2 takes the MSA and the templates, and passes them through a **transformer.** Objective: Refine the representations for MSA and pair interactions, iteratively exchange information between them.
- A better model of the MSA will improve the network's characterization of the geometry $\rightsquigarrow$ helps refining the MSA.
- Process is organised in blocks that are repeated.

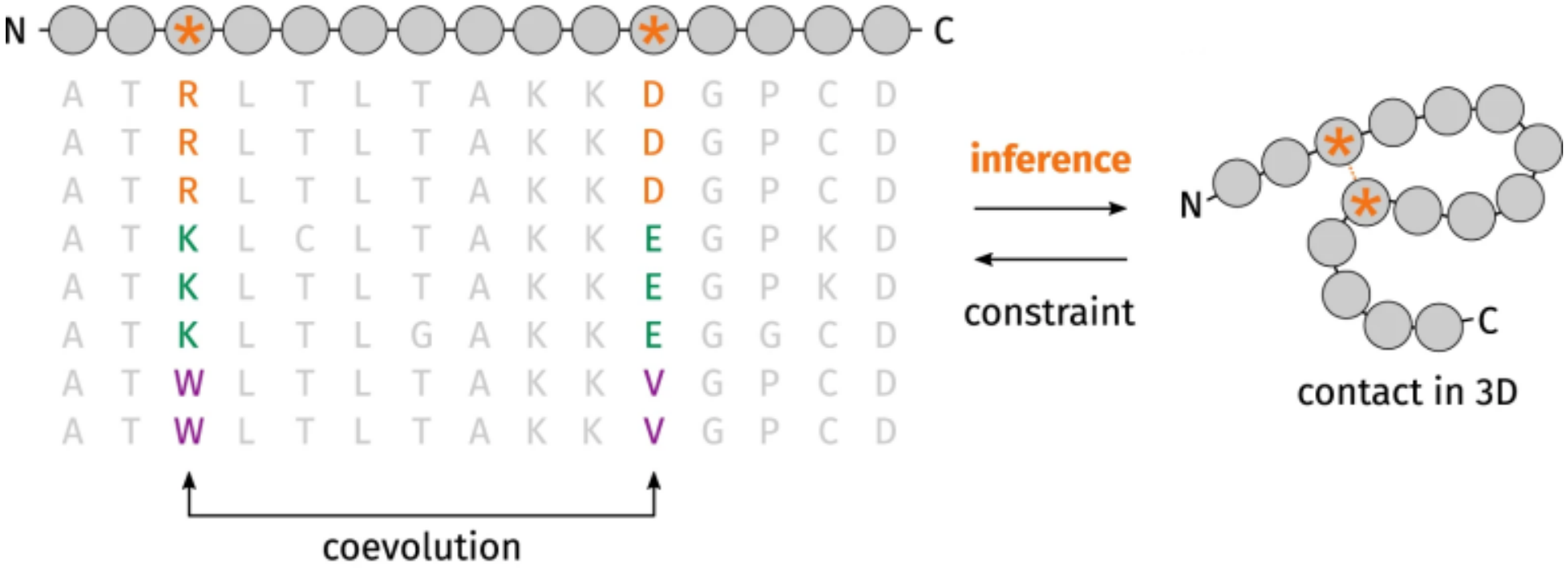
Alphafold 2 High-level overview

The structure module

- Takes the refined **MSA representation** and **pair representation**, and leverages them to construct a 3D-model of the structure.
- Does not use any energy-based optimisation algorithm: generates a static, final structure, in a single step.
- End result is a long list of coordinates representing the position of each atom of the protein.
- The model works iteratively. After generating a final structure, it will take all the information (i.e. MSA representation, pair representation and predicted structure) and pass it back to the beginning of the Evoformer blocks. This allows the model to refine its predictions.

Preprocessing: MSA

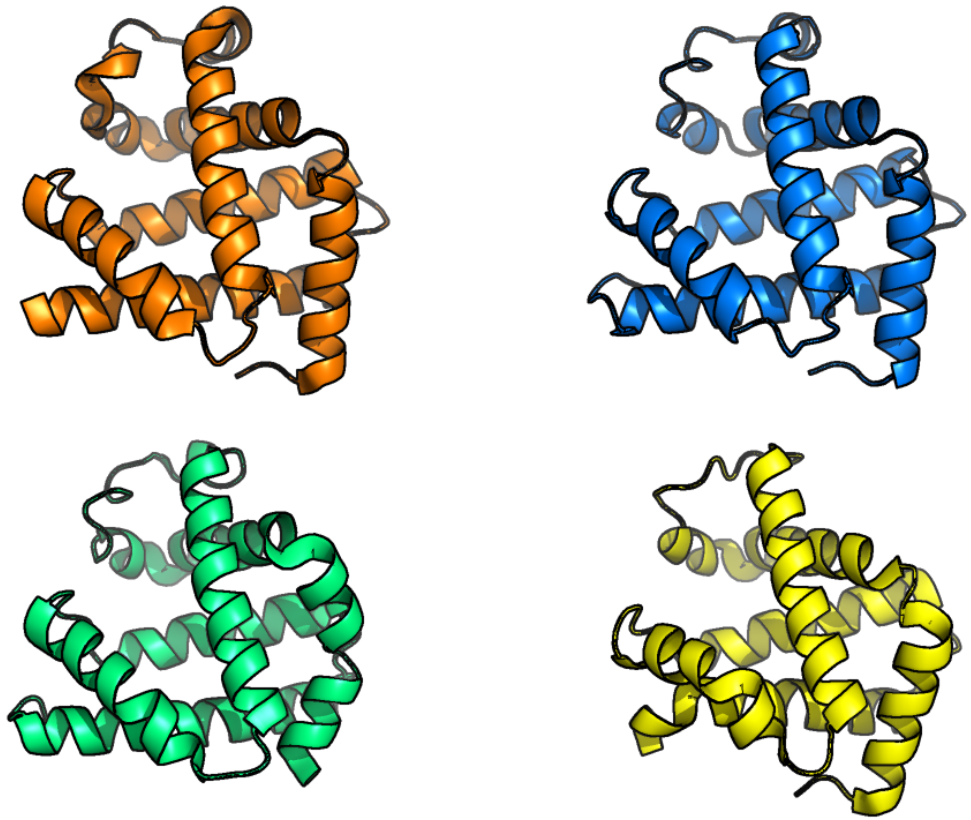
In an MSA, the sequence of the protein whose structure we intend to predict is compared across a large database. The underlying idea is that, if two amino acids are in close contact, mutations in one of them will be closely followed by mutations of the other, in order to preserve the structure.



Schematic of how co-evolution methods extract information about protein structure from a multiple sequence alignment (MSA). Image modified from doi: 10.5281/zenodo.1405369, which in turn was modified from doi: 10.1371/journal.pone.0028766.

Preprocessing: template sequences

- Proteins mutate and evolve, but their **structures tend to remain similar.**
- Example: structure of 4 myoglobin proteins from different organisms.
- They all look similar, but the **sequences are very different:** Bottom right to top left: only 25% common amino acids.



Protein structures of human myoglobin (top left), african elephant myoglobin (top right, 80% sequence identity), blackfin tuna myoglobin (bottom right, 45% sequence identity) and pigeon myoglobin (bottom left, 25% sequence identity).

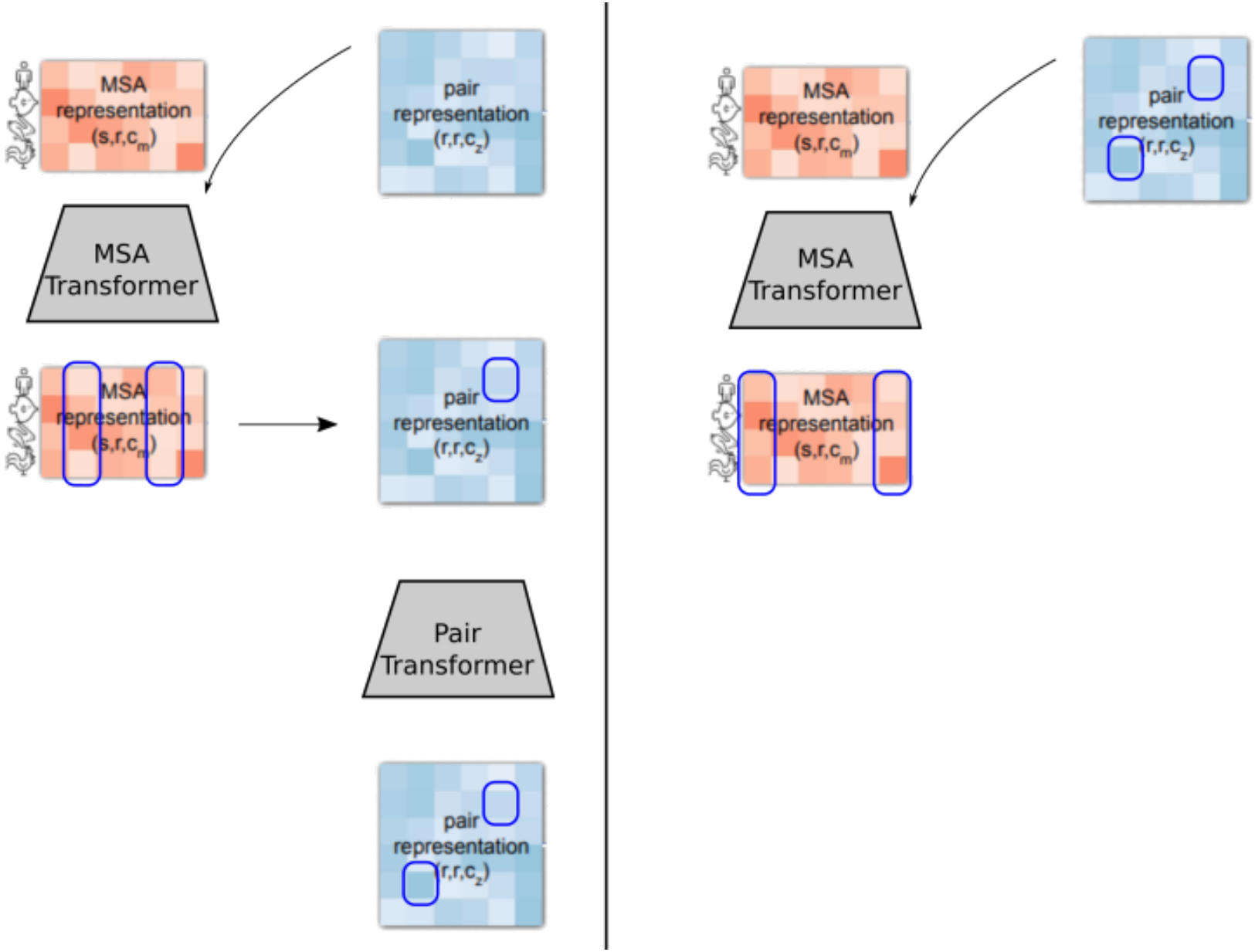
The Evoformer

- **Evoformer extracts information from the MSA and the templates.**
- Idea: information flows back and forth throughout the network.
- At every cycle: current structural hypothesis used to improve the assessment of the MSA $\rightsquigarrow$ **new structural hypothesis.**
- Both representations, sequence and structure, exchange information until the network reaches a solid inference.

Intuition:

- Suppose that you look at the MSA and notice a correlation between a pair of amino acids, A and B.
- Your hypothesisis: A and B are close $\rightsquigarrow$ translate this assumption into the structure model.
- You observe: since A and B are close, there is a good chance that C and D should be close.
- New hypothesis, based on the structure, which can be confirmed by searching for correlations between C and D in the MSA.
- By repeating this, you can build a good understanding of the structure.

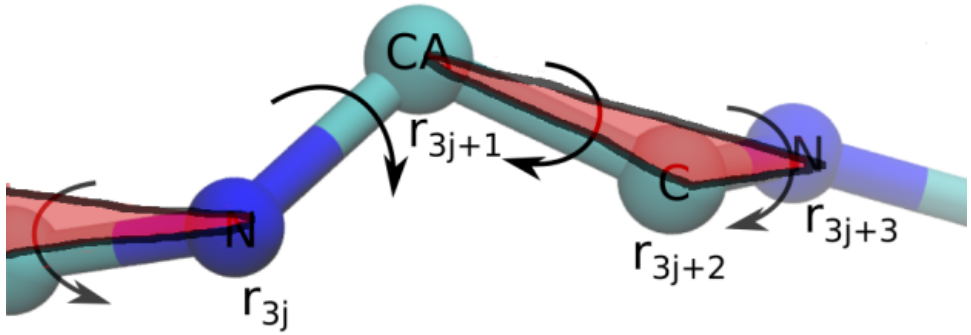
The Evoformer



Conceptualization of the Evoformer information. In the left diagram, the MSA transformer identifies a correlation between two columns of the MSA, each corresponding to a residue. This information is passed to the pair representation, where subsequently the pair representation identifies another possible interaction. In the right diagram, the information is passed back to the MSA. The MSA transformer receives an input from the pair representation, and observes that another pair of columns exhibits a significant correlation.

The structure module

- Recap: two “representations”: MSA (captures sequence variation); and “pairs of residues” (captures which residues are likely to interact).
- Q: how do we get a structure from these? $\rightsquigarrow$ **structure module.**
- Protein modeled as a **residue gas:** Every amino acid is modelled as a triangle, representing the 3 atoms of the backbone. Triangles float around in space, and are moved by the network to form the structure.



The “residue gas” approach. Image taken from the OpenFold 2 webpage, by Georgy Derevyanko.

- Transformations are parametrised as “affine matrices”, which are a mathematical way to represent translations and rotations in a single 4×4 matrix:

$$\mathbf{M} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Image taken from BrainVoyager.

The structure module

- At the beginning of the structure module, all of the residues are placed at the origin of coordinates.
- At every step of the iterative process, AlphaFold 2 produces a set of affine matrices that displace and rotate the residues in space.
- This representation does not reflect any physical or geometrical assumptions, and as a result the network has a tendency to generate structural violations.
- New flavour of attention devised specifically for working with 3D-structures: **Invariant Point Attention (IPA).**
- Main property: invariance to translations and rotations. The model knows that rotations and translations of the data lead to the same answer!
- Side chains: Positions parametrised by list of torsion angles.