Computerlinguistik

Neuronale Netze

WS 2021/22
Vera Demberg

Neuronale Netze – Was ist das?

- Einer der **größten Fortschritte** in der **Sprachverarbeitung und Bildverarbeitung** der letzten Jahre:

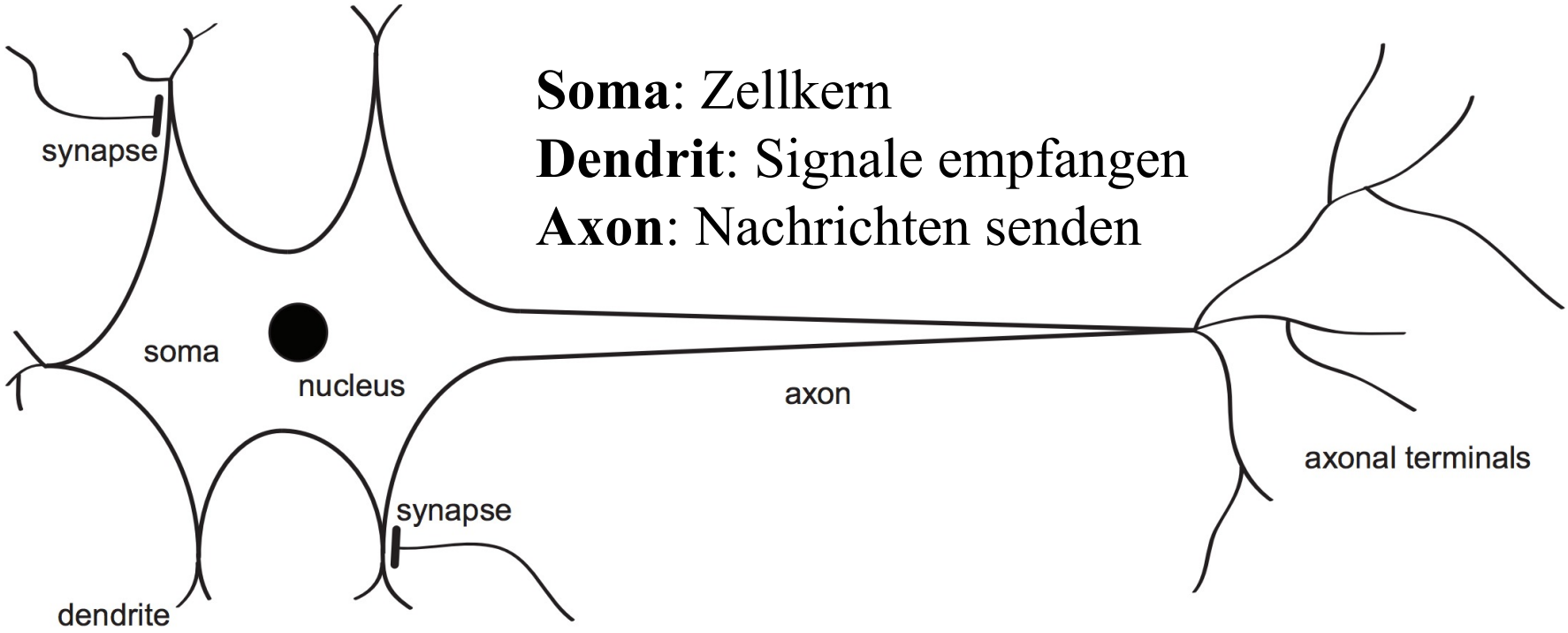
  NEURONALE NETZE

- Spracherkennungsfehlerrate soweit reduziert, dass freie Eingaben möglich sind.
- Gesichter / Katzen / Menschen können automatisch erkannt werden.

Neuronale Netze:
Das Katzenerkennungsneuron (Le et al., 2012)

Input: Standbilder von Youtube videos
Ein Neuron hat sich auf die Erkennung von Katzen spezialisiert.

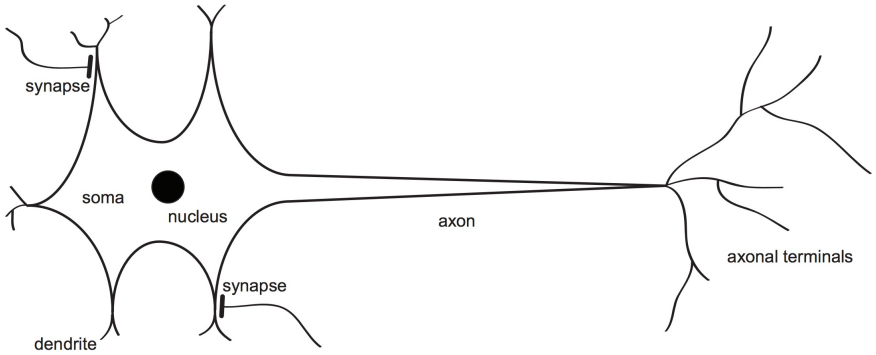
Neuronen im Gehirn



Soma: Zellkern
Dendrit: Signale empfangen
Axon: Nachrichten senden

schematisches Bild eines menschlichen Neurons im Gehirn

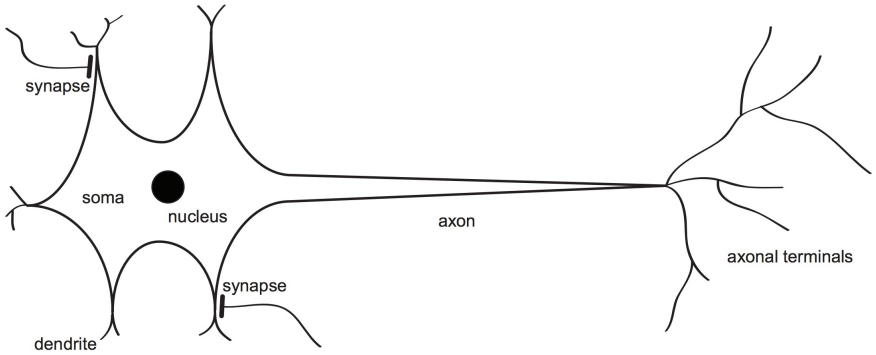
Neuronen im Gehirn



Wichtige relevante Fakten:

- Jedes Neuron hat viele **Dendriten** (Signale empfangen) und ein **Axon** (Signale senden) mit mehreren Terminalen.
- Das Axon eines Neurons **sendet ein Signal** an das Dendrit eines anderen Neurons.
- Axonterminal und Dendrit sind durch eine **Synapse** verbunden (kann Signal ***verstärken*** oder ***dämpfen***)
- Starkes einkommendes Signal → Neuron sendet selbst ein Signal (elektrische Aktivität)

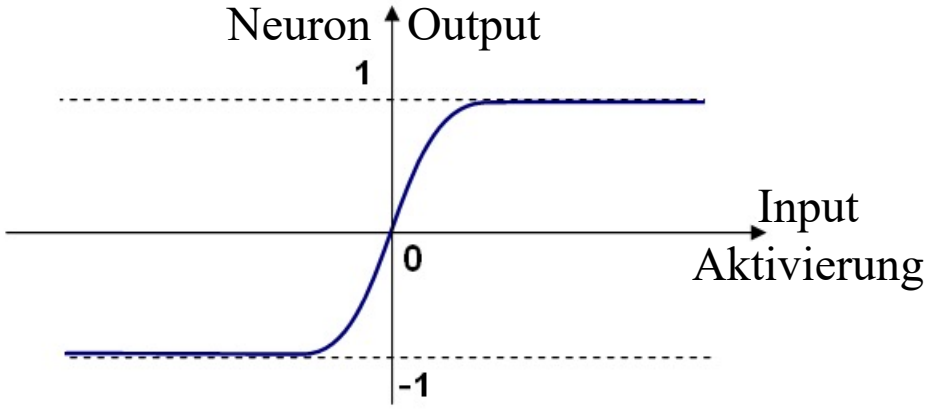
Neuronen im Gehirn



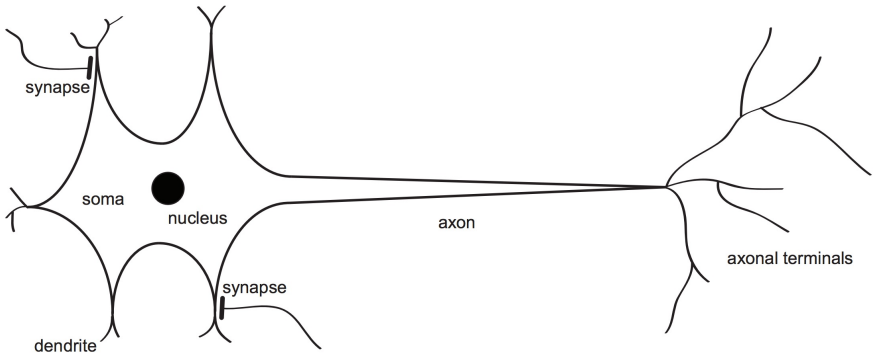
Wann feuert ein Neuron?

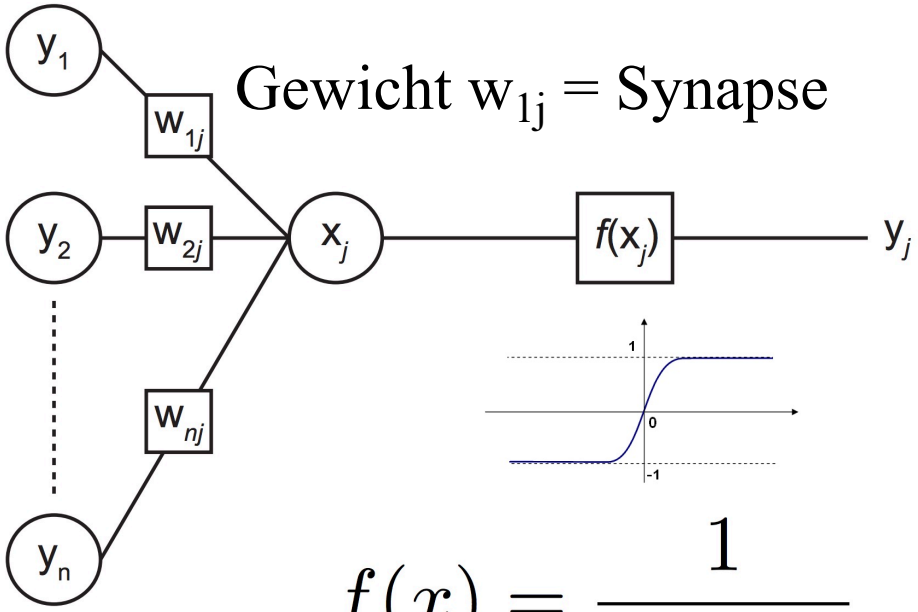
Aktivierung hängt ab von:

- Wieviele Signale gehen ein bei den Dendriten?
- Synapsen: verstärkend oder dämpfend?
- Zufall: Aktivierungsfunktion ist nicht deterministisch; höhere Aktivierung erhöht Wahrscheinlichkeit, dass das Neuron feuert.



Ein künstliches Neuron



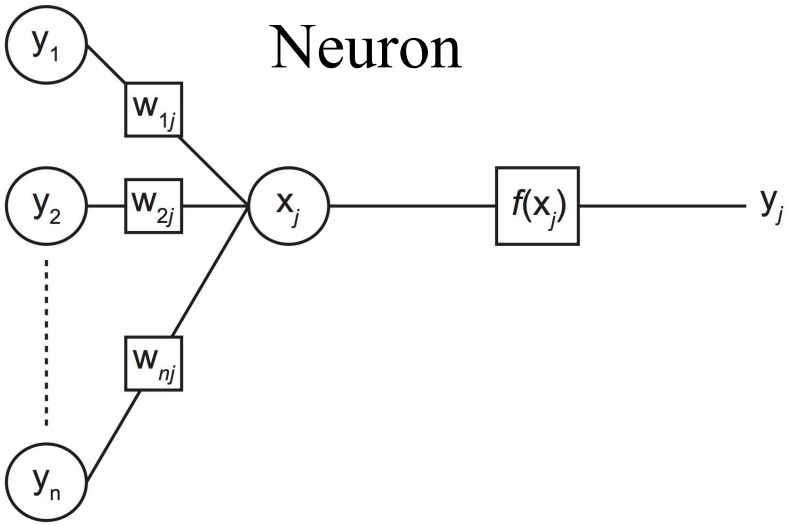


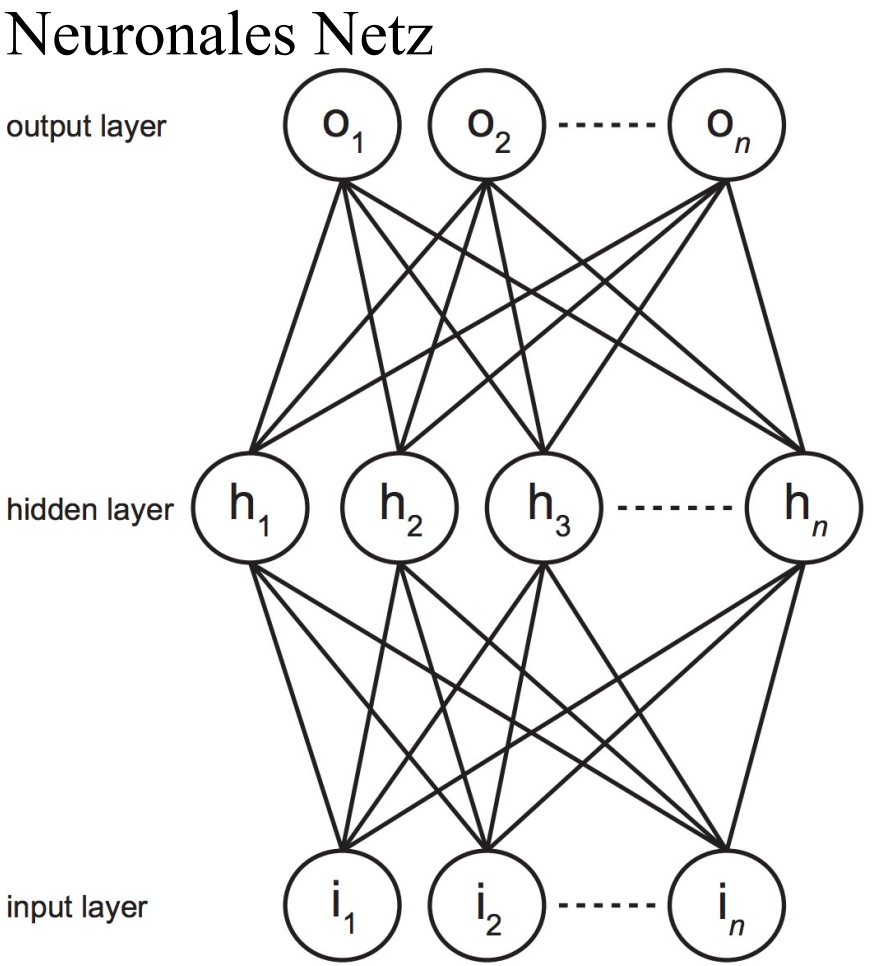
$$f(x) = \frac{1}{1 + e^{-x}}$$

Lernen:
Wenn sich die Synapsen-gewichte ändern.

künstliches Neuron aus einem Neuronalen Netz

Neuronale Netze





Input: Beobachtungen

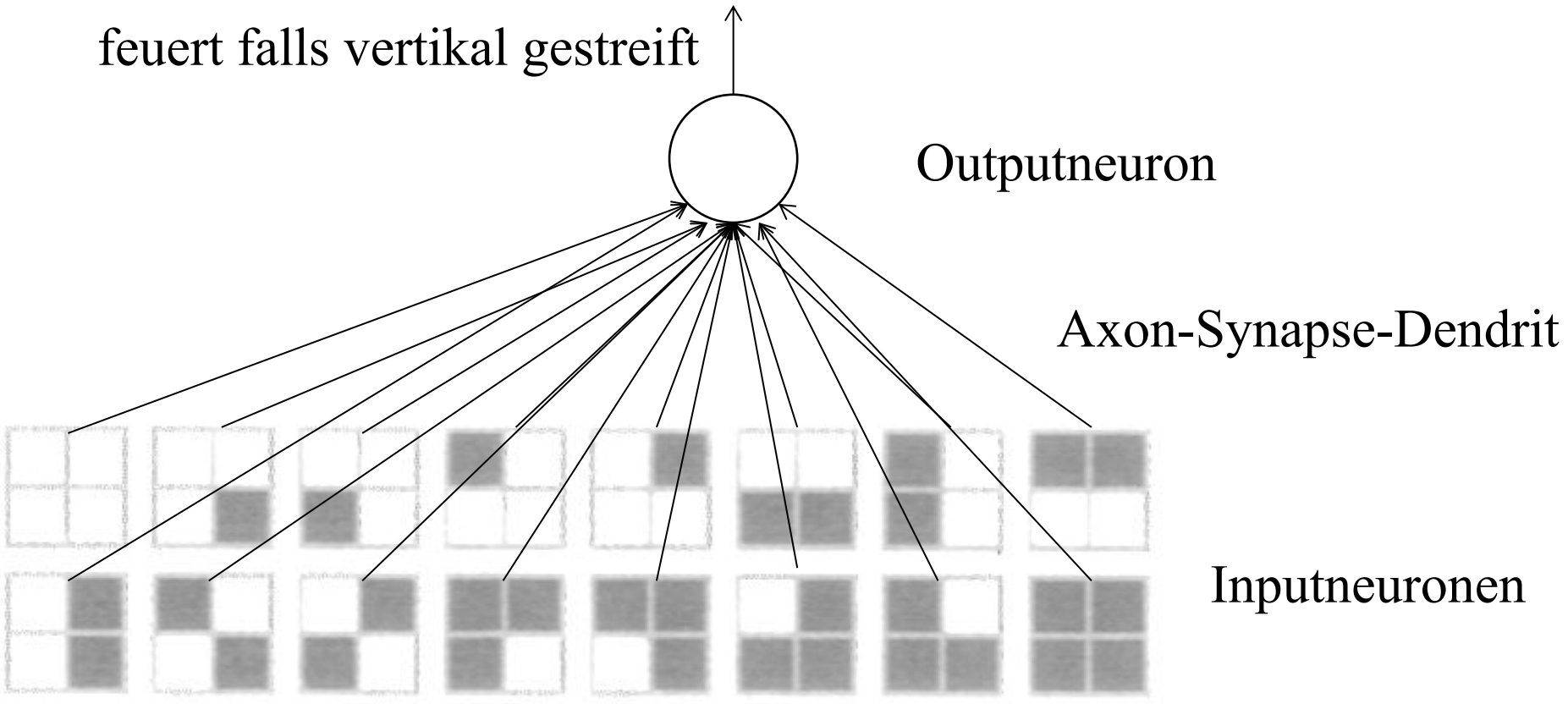
- Bilder
- Laute

Output: Erkannte Muster

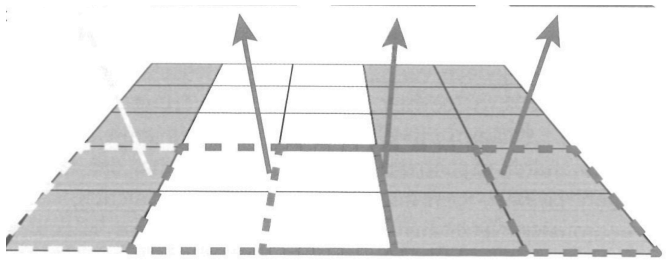
- Objekt
- Wort

Perzeptron

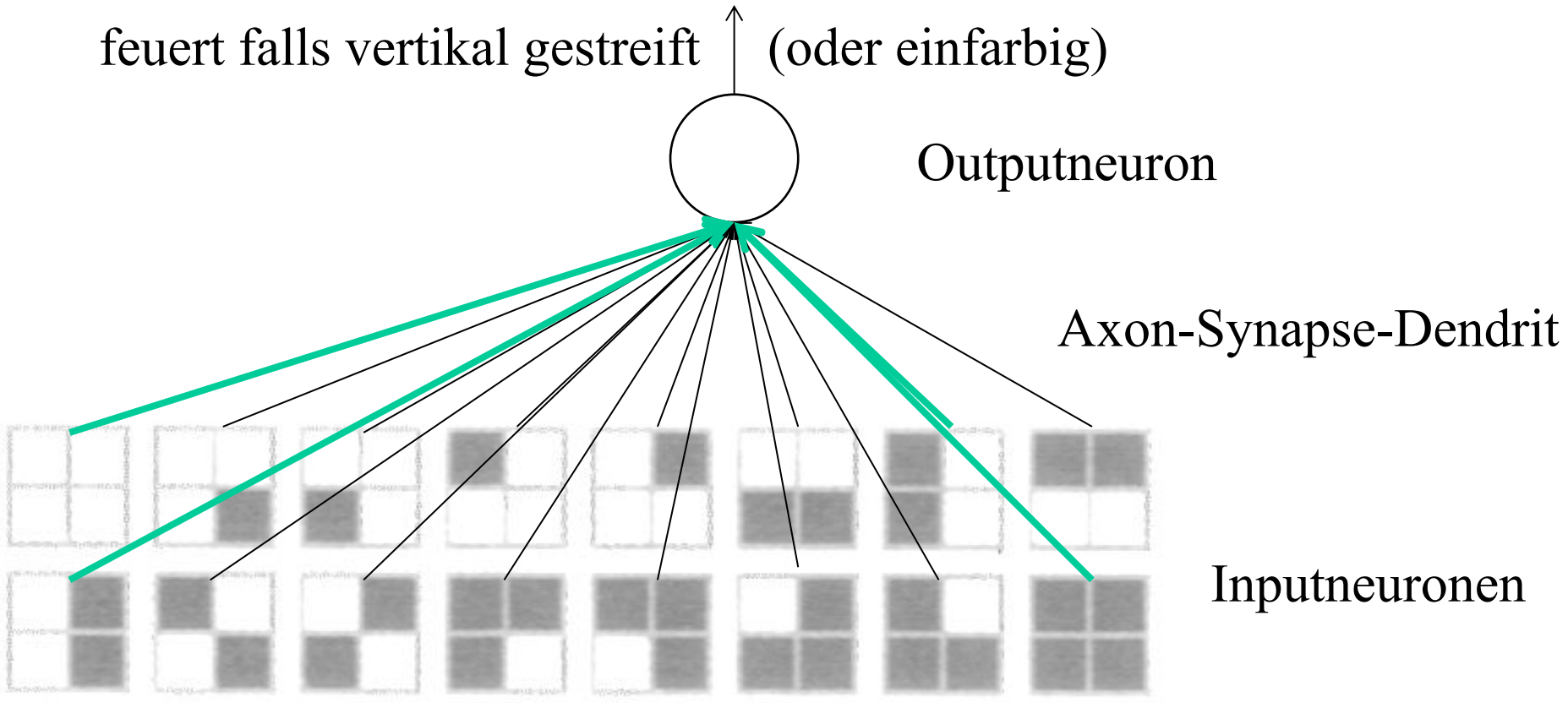
Erstes Beispiel: **Streifendetektor** (vertikale Streifen)



Perzeptron



Erstes Beispiel: **Streifendetektor** (vertikale Streifen)



Streifenperzeptron

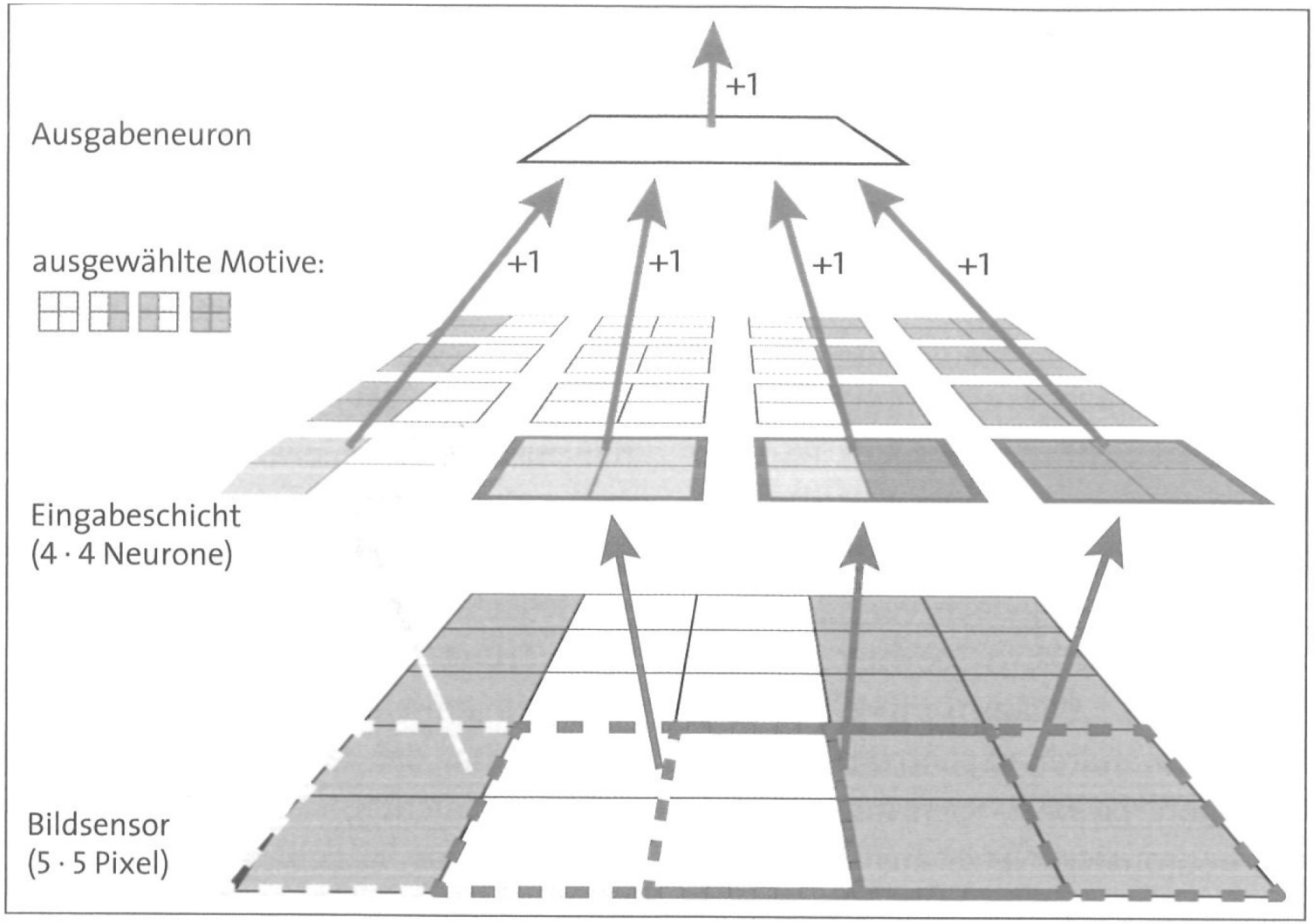


Bild aus Spektrum der Wissenschaft 08/14

Neuronale Netze sollen selbst lernen

- Streifenperzeptron: wir haben “programmiert” welche Inputneuronen wann feuern sollen, und die “Gewichte” selbst gesetzt (auf 1).

- Wir Menschen haben auch Neuronen in unserer primären Sehrinde, die auf Streifen reagieren.

- Aber Kindern muss man ja auch nicht erklären, wie sie einen Streifen sehen… kann das neuronale Netz das auch selbst lernen?

Lernende Neuronale Netze

Damit das Netz lernen kann, müssen wir

- ihm viele Bilder mit und ohne Streifen zeigen
- Feedback geben, auf welchen Bildern Streifen sind

Richtige Antwort → das Netz muss nichts tun

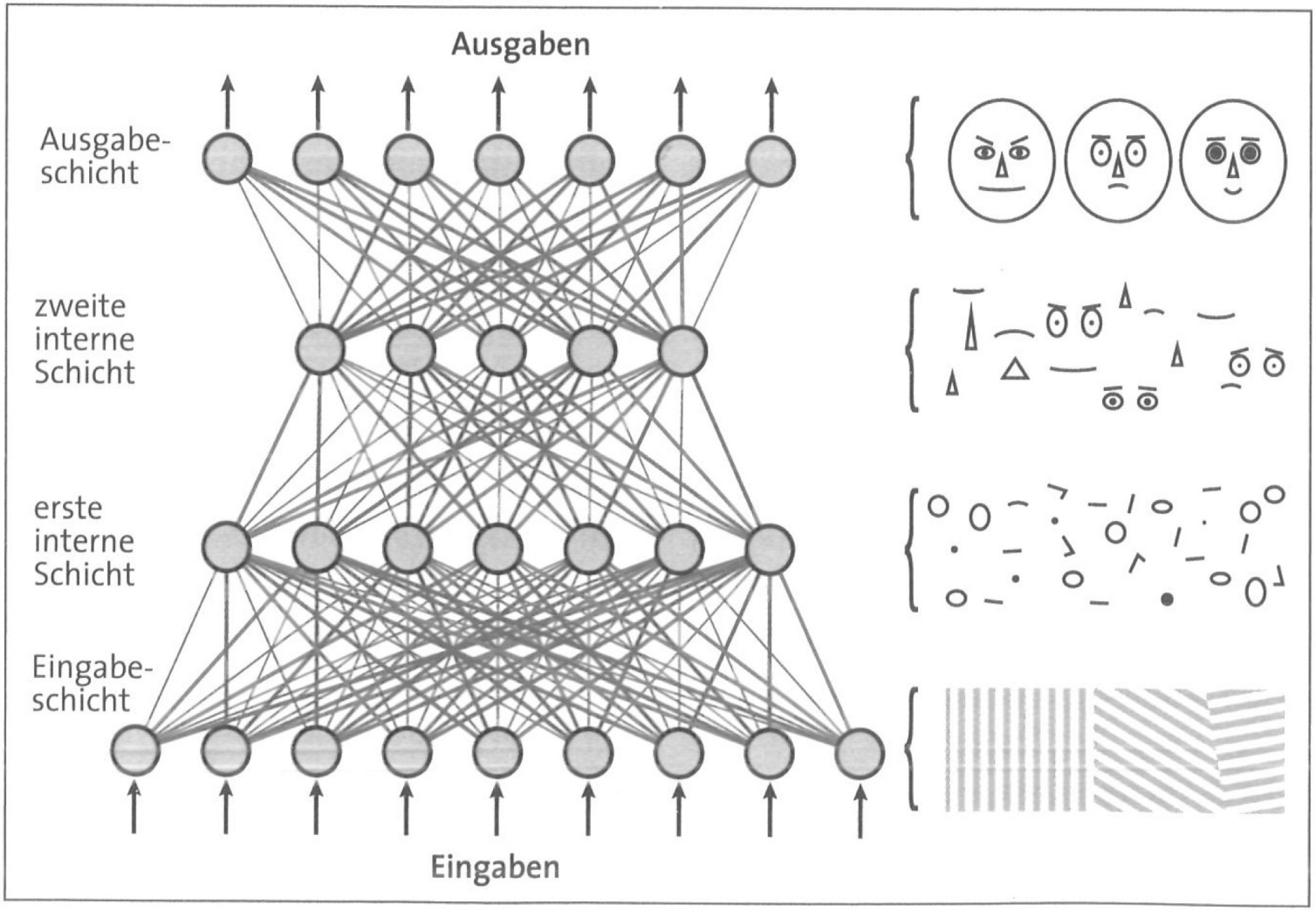
Falsche Antwort → das Netz muss mind. ein Neuron in der Eingabeschicht anweisen, gegenteilige Antwort zu geben

Streifenperzeptron → Neuronales Netz

Verbesserungsmöglichkeiten für einfaches Perzeptron:

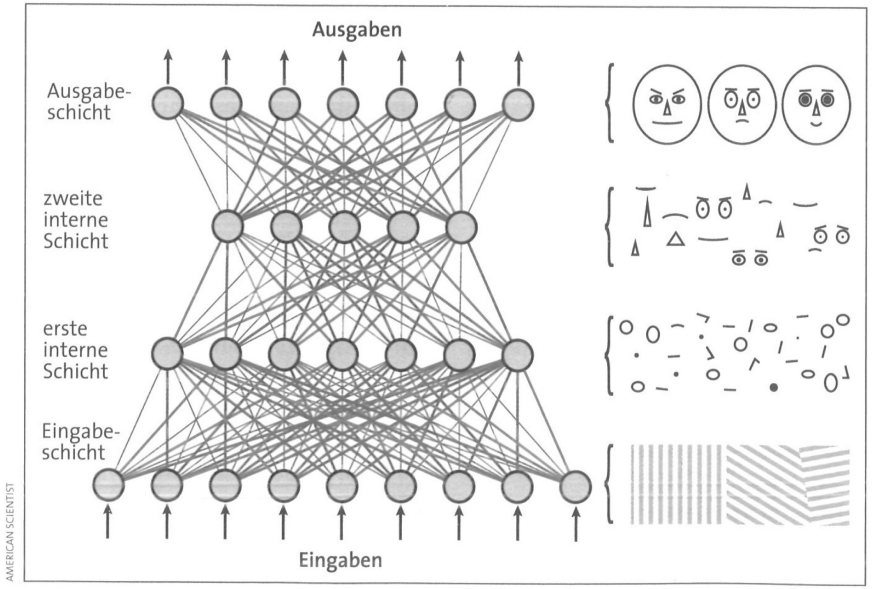
- Tiefere Struktur als input-output
  (eine oder mehr “hidden” layers)
- Gewichtung (wie Synapsen) anstelle von
  Erkennung ob ein bestimmtes Neuron gefeuert hat
  oder nicht
- Eingabeneuronen können viel mehr Input
  bekommen (z.B. gesamtes Bild geht an jedes
  Eingabeneuron)

Tiefe neuronale Netze



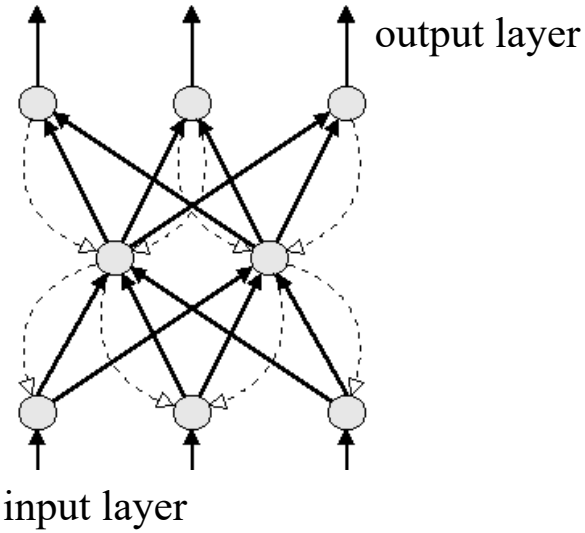
Lernen mit tiefen neuronalen Netzen

- Wie lernt man jetzt?
- In welcher Schicht liegt der Fehler, falls am Ende die Ausgabe falsch ist?

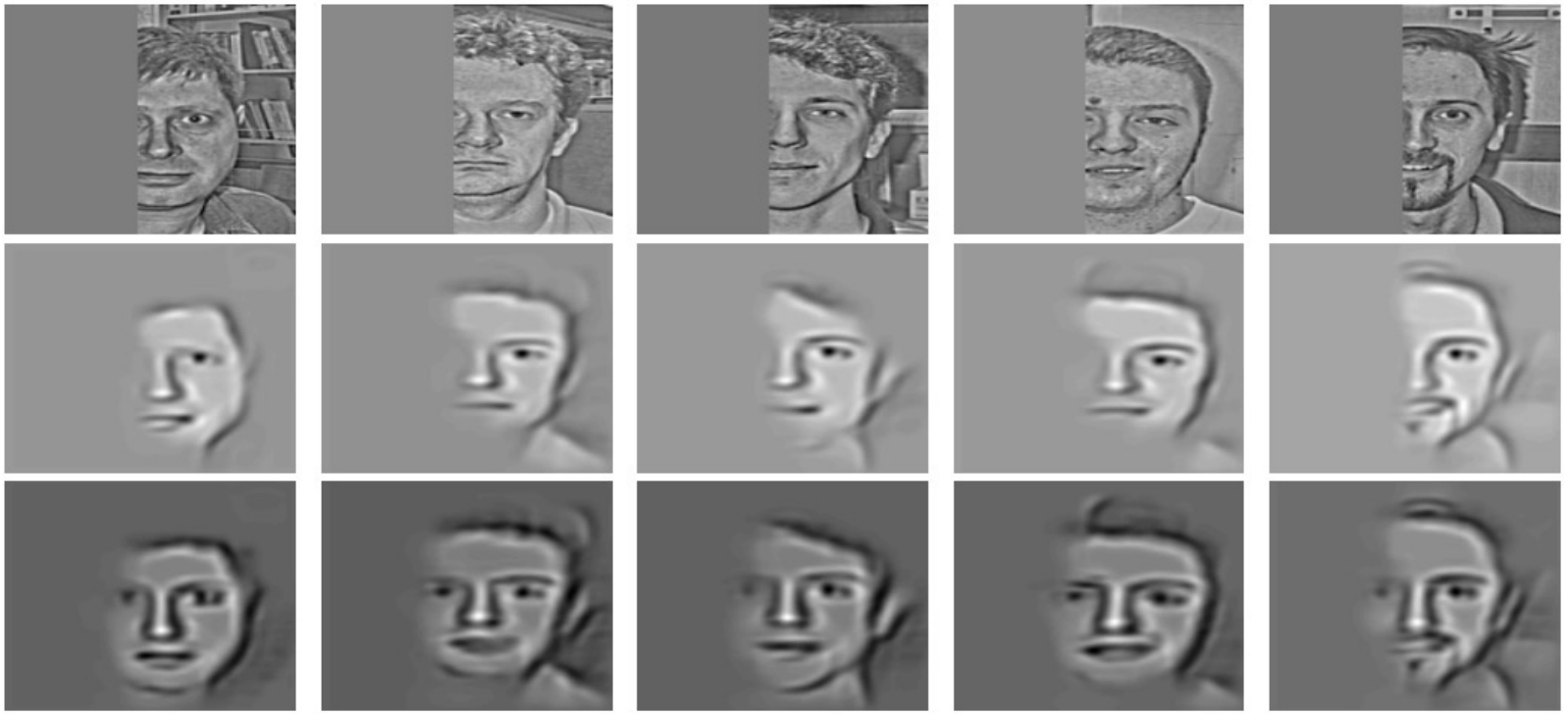


→ "Backpropagation"

jede Schicht meldet an die darunter weiter, falls ein Fehler gemacht wurde.



Model can reconstruct a partially occluded face

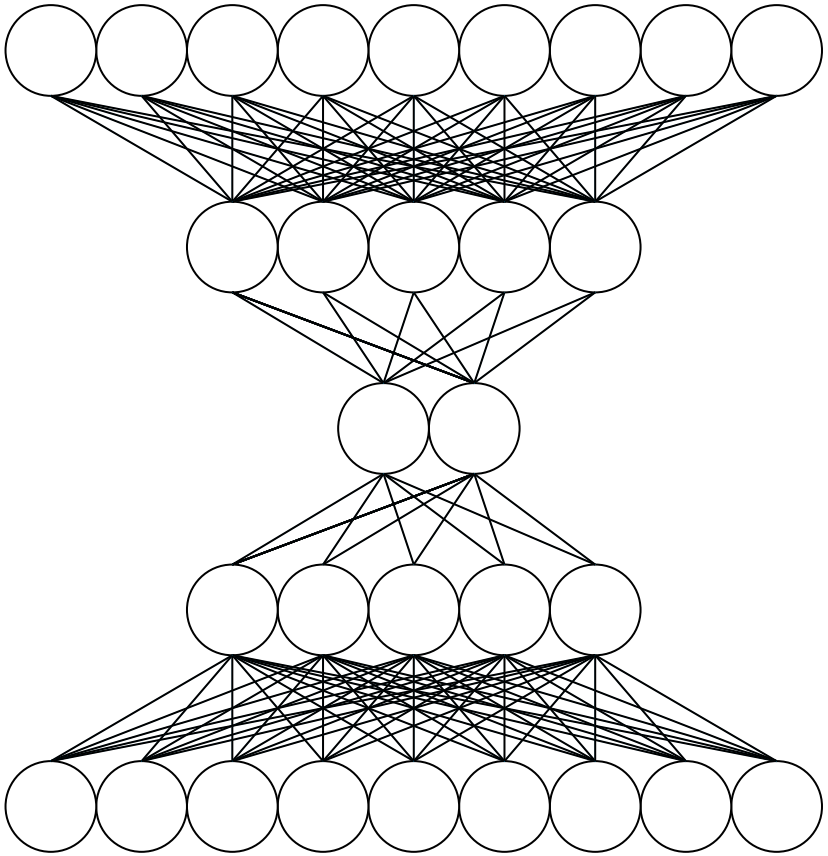


Lee, Grosse, Ranganath, Ng, 2009, ICML

Lernen ohne Lehrer

- Wir müssen nicht mehr genau sagen, wie ein Objekt zu erkennen ist.
- Aber: immernoch müssen wir dem Netz mitteilen, was auf dem Gesamtbild zu sehen ist.
  - für Gesichtererkennung müssten wir immer mitteilen, ob auf einem bestimmten Foto ein Gesicht drauf ist.
- Es geht! Google brain: 1 Millarde Verbindungen, 9 Schichten, 10 Millionen Bilder, bestimmte Form von Architektur eines neuronalen Netzes: **Autoencoder**

Autoencoder: optimiert für gute Komprimierung



output layer (reconstruct input)

hidden layer

bottleneck hidden layer

hidden layer

input layer

Autoencoder lernt, Katzen zu erkennen ohne Trainingsdaten oder Labels

"Katzenneuron"

Le et al., 2012 ICML

Bilder von denen der Autoencoder glaubt, dass es Katzen sind.

Figure 16. Top: most responsive stimuli on the test set for the cat neuron.

Neuronale Netze

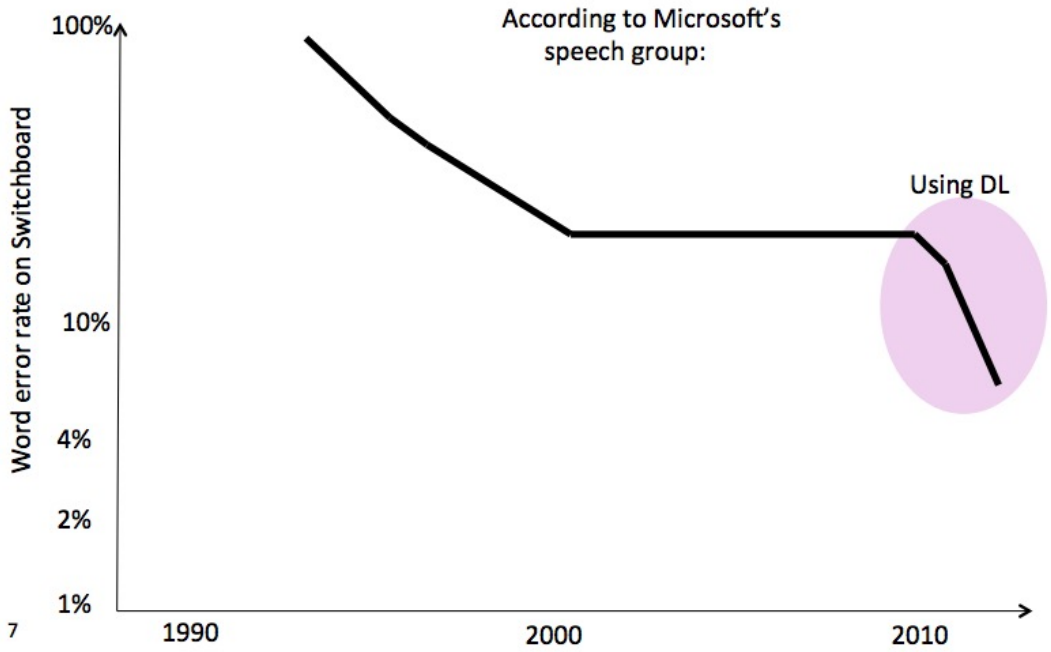
Es gibt auch **Nachteile**:

- Theoriedefizit!
- viele Parameter – aber nur Heuristiken, wie wir sie setzen sollen.
  - Anzahl Schichten???
  - Anzahl Neuronen???
  - Anfangsgewichte???
  - Lernprotokoll???

  → derzeit: ausprobieren!

Was hat das mit Coli zu tun???

- Objekterkennung nur als anschauliches Beispiel
- erster großer Erfolg: **Spracherkennung** (2009)
  - Fehlerrate sank um 25%: absolut phänomenal.



Was hat das mit Coli zu tun???

- Objekterkennung nur als anschauliches Beispiel
- erster großer Erfolg: **Spracherkennung** (2009)
  - Fehlerrate sank um 25%: absolut phänomenal.
- Verfahren ebenso erfolgreich angewendent auf
  - Language modelling
  - POS tagging, named entity recognition
  - Sprachsynthese
  - Document retrieval / Information retrieval
  - semantischer Ähnlichkeit, Paraphrasenerkennung
  - Maschinelle Übersetzung
  - Sentiment analysis

Zur Erinnerung (Video von letzter Woche): Language models

- How can we estimate the probability of word sequence $P(W) = P(w_1 w_2 \ldots w_n)$?
- We can estimate this from the frequency of word sequences in texts.
- But we still have a data sparsity problem:
  complete sentences have rarely been seen before; in fact, one can easily say a sentence that has never been said before.
- Chain rule allows us to reduce the joint probability $P(w_1 w_2 \ldots w_n)$ to conditional probabilities:

  $P(w_1 w_2 \ldots w_n)$

  $= P(w_1) * P(w_2|w_1) * P(w_3|w_1 w_2) * \ldots * P(w_n|w_1 w_2 \ldots w_{n-1})$
- *But this didn't solve the data sparsity problem:* $P(w_n|w_1 w_2 \ldots w_{n-1})$

n-grams

- n-gram method:
  - We approximate the probability of observing a word *w* in the context of the previous words by the probability of this word occurring given a limited-length context of previous words. ("Markov-assumption")
  - E.g.: A bigram is the probability of a word given the previous word $P(w_n|w_{n-1})$.
  - Usually, we use trigrams, 4-grams or 5-grams.
  - What do you think are the (dis)advantages of bigrams vs. 5-grams?

- Example for bigram approximation:
  - $P(w_n|w_1w_2... w_{n-1}) \approx P(w_n|w_{n-1})$

    $P(w_1w_2 ... w_n) \approx P(w_1)*P(w_2|w_1)*P(w_3|w_2)* ... P(w_n|w_{n-1})$

How to calculate n-grams from texts

Example for bigram approximation:

- $P(w_n|w_1w_2... w_{n-1}) \approx P(w_n|w_{n-1})$

  $P(w_1w_2 ... w_n) \approx P(w_1)*P(w_2|w_1)*P(w_3|w_2)* ... P(w_n|w_{n-1})$

We simply calculate the probability $P(w_3|w_2)$ as $P(w_2\, w_3)/P(w_2)$

And estimate probabilities from observed numbers of occurrences in texts.

$P(w_2\, w_3) = freq(w_2\, w_3)/\#bigrams\ in\ text$

$P(w_2) = freq(w_2)/\#words\ in\ text$

Hence $P(w_3 \mid w_2) = freq(w_2w_3)/freq(w_2)$

Try it for yourself

$P(w_3 \mid w_2) = freq(w_2 w_3)/freq(w_2)$

Example text:

A tall girl lived in a small house next to a tall tree. One day, the tall girl wanted to climb onto the tall tree.

Please calculate the bigram probability *P(girl|tall)*

The Era of Deep Learning in CL

Since 2015, Deep Learning (aka neural networks) has become the dominant paradigm in CL.

LM model	Model class	PTB test perplexity
old-school	5-grams with Kneser-Ney	125.7
Mikolov et al. 2011	neural (RNN)	101.0
Gong et al. 2018	neural (complex)	46.5

The Era of Deep Learning in CL

We will now take a look at how RNNs (and an improved version, called LSTMs) work.

LM model	Model class	PTB test perplexity
old-school	5-grams with Kneser-Ney	125.7
Mikolov et al. 2011	neural (RNN)	101.0
Gong et al. 2018	neural (complex)	46.5

Disadvantages of ngram models

Key observation regarding problems with n-gram models:

- You have to decide on a fixed length context (bigram, trigram, 5-gram)
- If a short context is chosen, many long distance dependencies are missed.
- If a long context is chosen, we have data sparsity issues (cannot estimate probabilities accurately because we haven't observed these exact contexts frequently enough).
- Dependencies in language can be arbitrarily long:
  - Syntactic dependencies
  - Topic-related dependencies

RNNs

If we use a neural network, we also need to make sure that the context of previous words is represented in the model. It therefore makes sense to design a neural network architecture that reflects this challenge.

Solution that (in principle) allows to model arbitrarily long context:

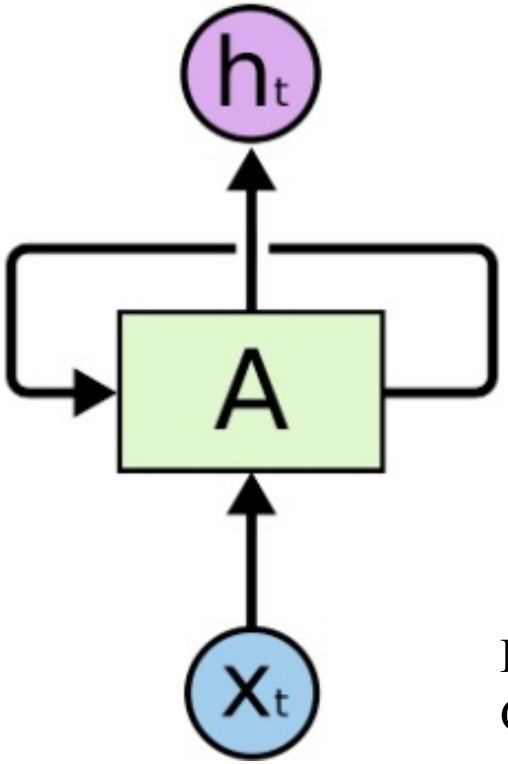
Recurrent Neural Network

x_t is the input word

h_t is the predicted next word

A is an internal hidden state

The network is “recurrent” because it contains a loop.



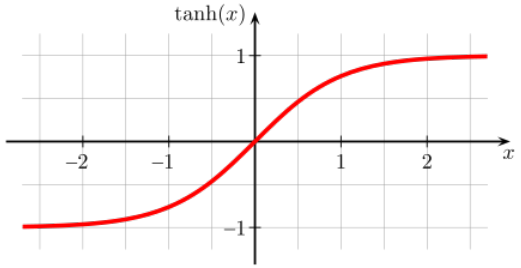
Picture credit: Christopher Olah

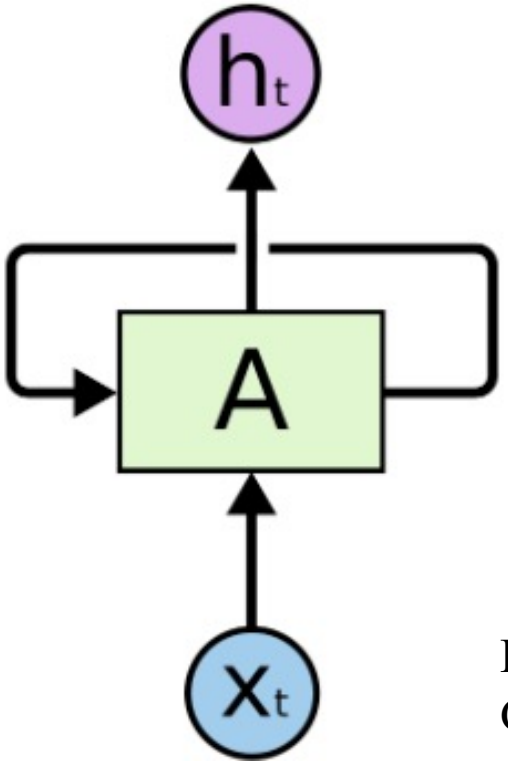
RNNs

If we use a neural network, we also need to make sure that the context of previous words is represented in the model. It therefore makes sense to design a neural network architecture that reflects this challenge.

$$A_t = \tanh (W_{AA}A_{t-1} + W_{xA}x_t)$$

$$h_t = W_{Ay}A_t$$





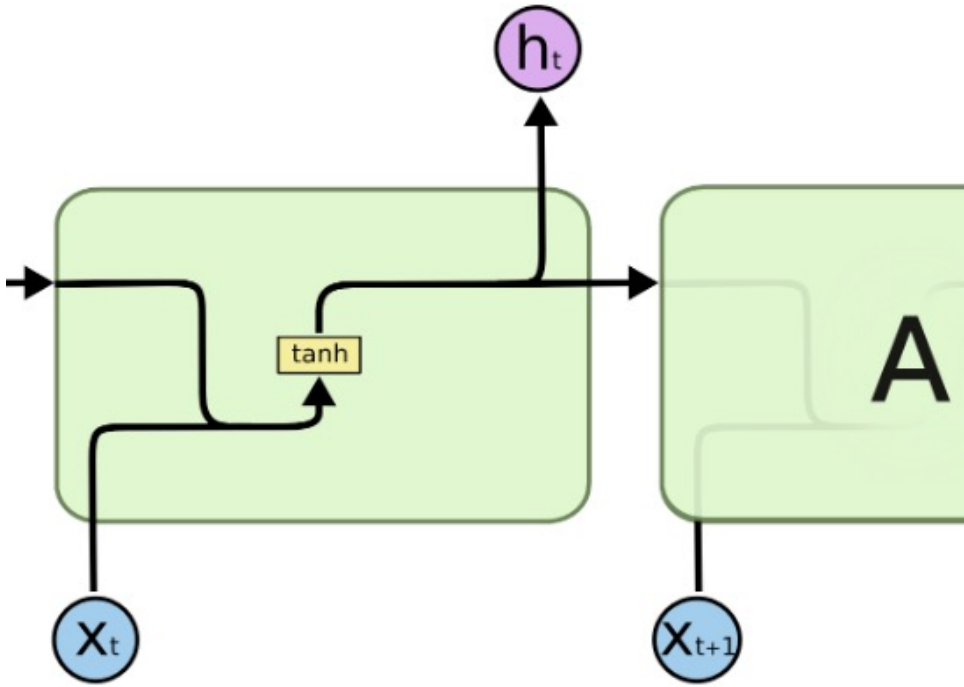
Picture credit:
Christopher Olah

RNNs

If we use a neural network, we also need to make sure that the context of previous words is represented in the model. It therefore makes sense to design a neural network architecture that reflects this challenge.

$$A_t = \tanh (W_{AA}A_{t-1} + W_{xA}x_t)$$

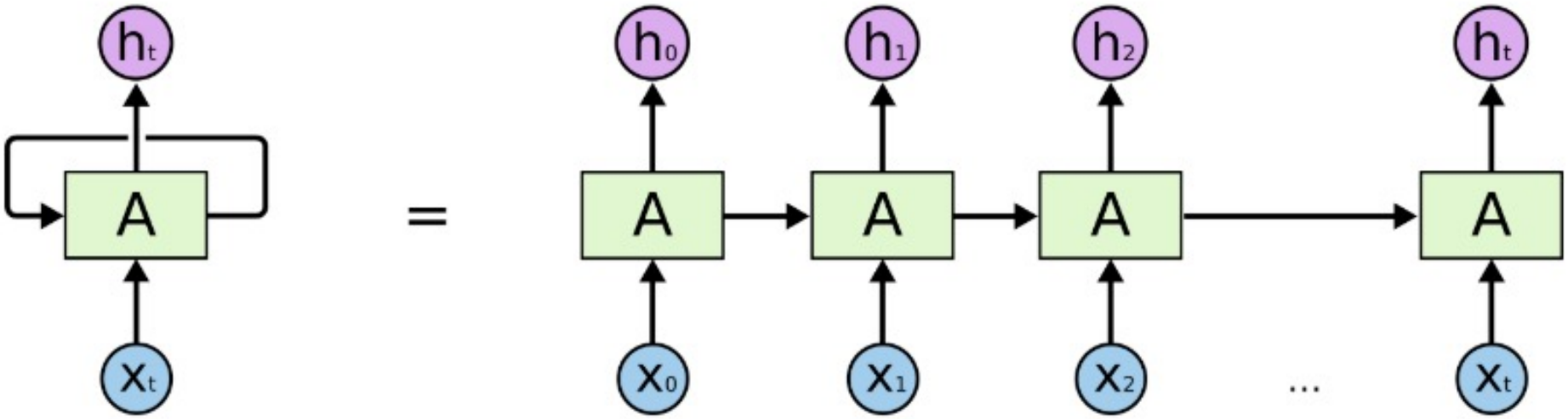
$$h_t = W_{Ay}A_t$$



Christopher Olah

RNNs

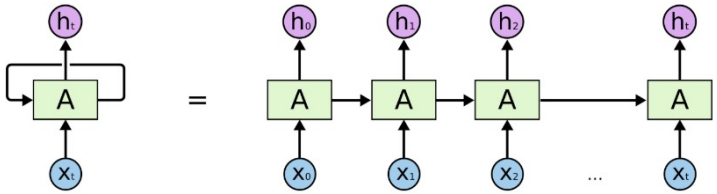
At word x_n, the network contains information about the new word and a representation of the previous words.

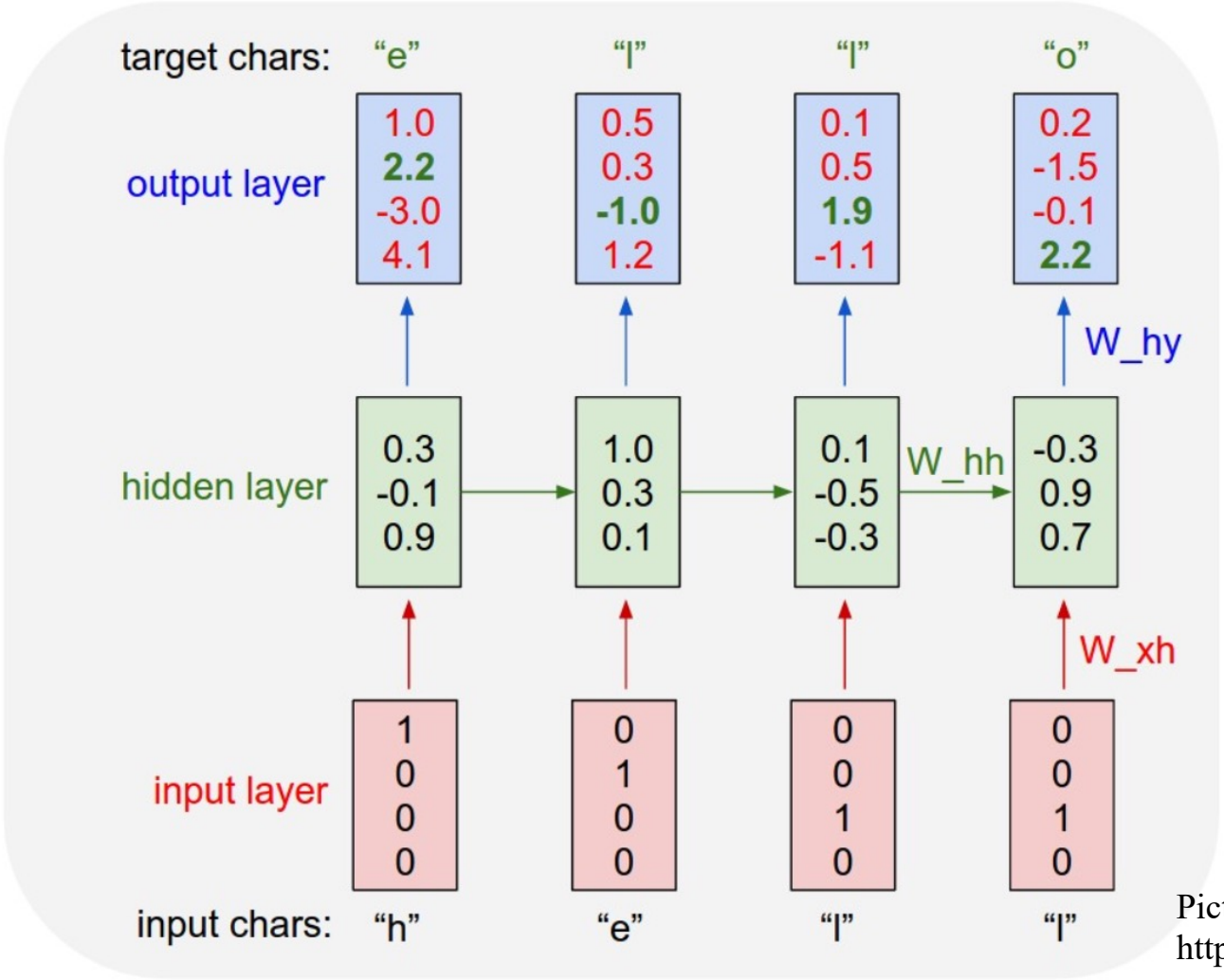


An unrolled recurrent neural network.

Picture credit: Christopher Olah

A simple example: character-based RNN

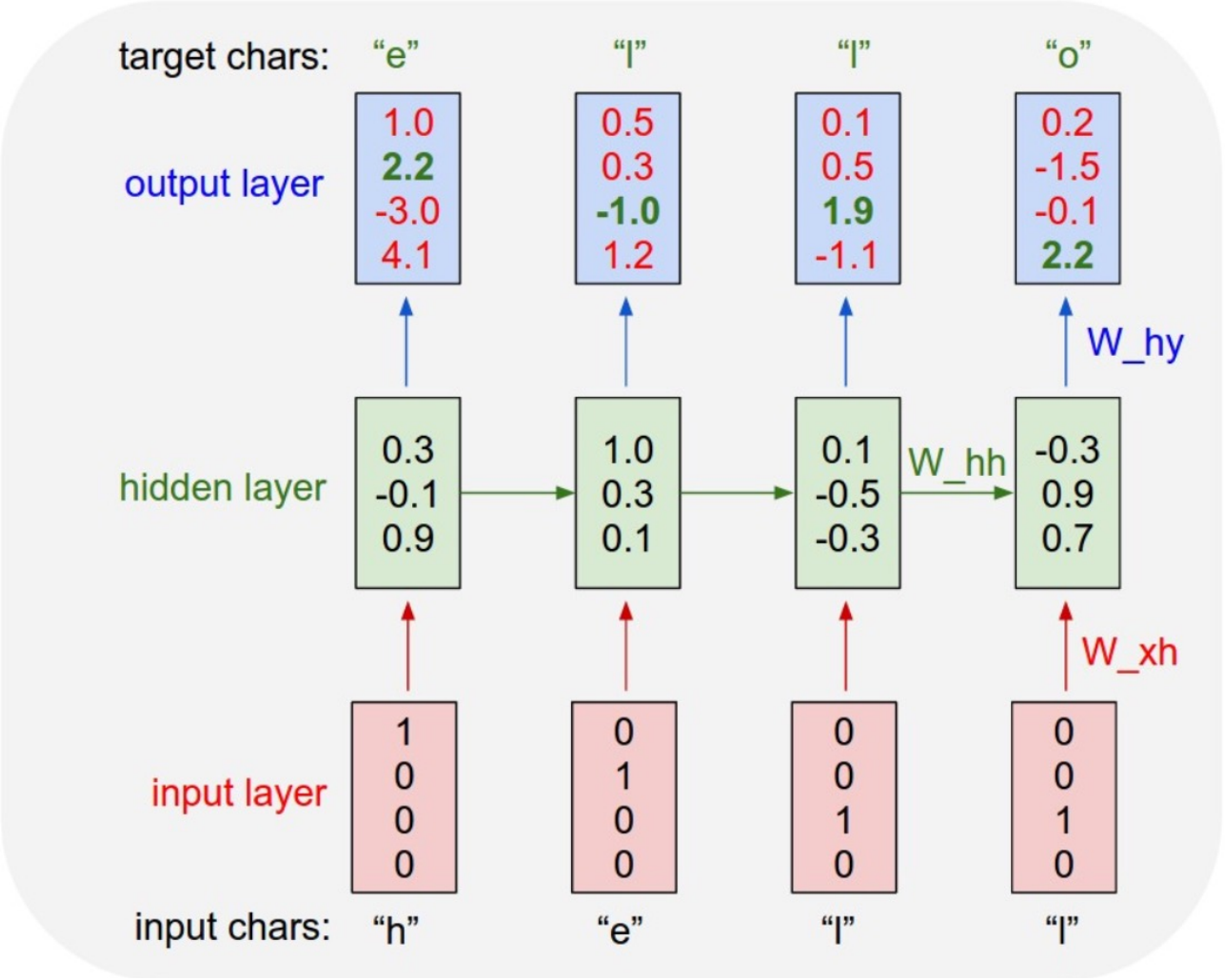




Picture from http://karpathy.github.io/

An example RNN with 4-dimensional input and output layers, and a hidden layer of 3 units (neurons). This diagram shows the activations in the forward pass when the RNN is fed the characters "hell" as input. The output layer contains confidences the RNN assigns for the next character (vocabulary is "h,e,l,o"); We want the green numbers to be high and red numbers to be low.

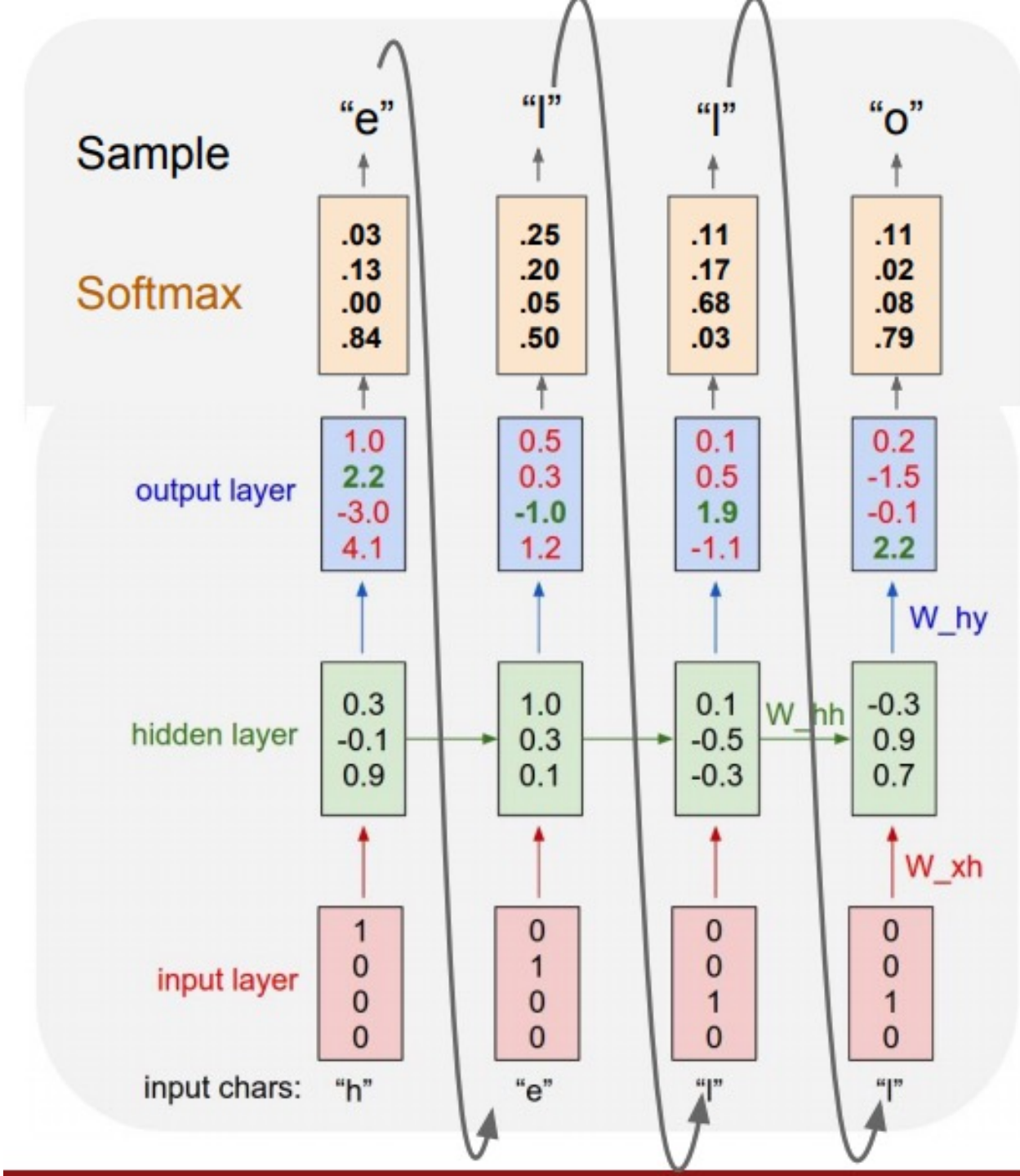
Training time



Use backpropagation to nudge the weights such that green numbers increase, and red numbers decrease.

Test tim

- Feed a character into RNN
- Obtain a distribution over (next) characters
- Sample from this distribution
- Feed the result in as the next input
- Repeat!

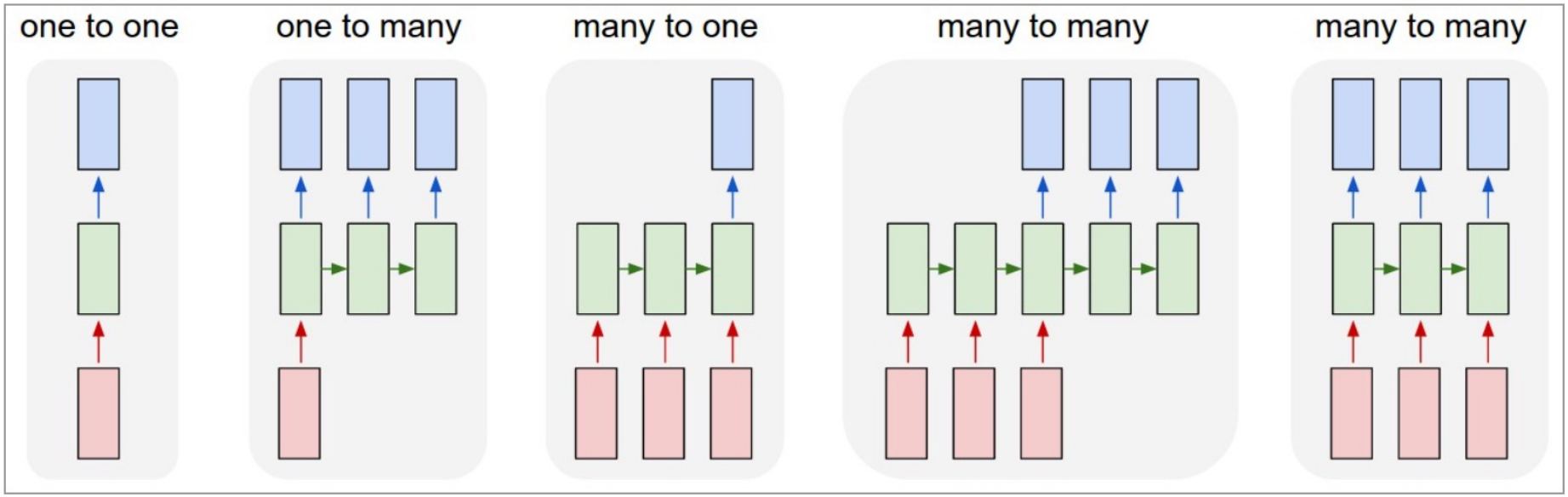


Shakespeare generated from an RNN

- Concatenate all works of Shakespeare into a file for training (4.4MB)
- 3-layer RNN
- 512 hidden nodes on each layer

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

Comparison to Convolutional NNs: RNNs allow to operate over sequences

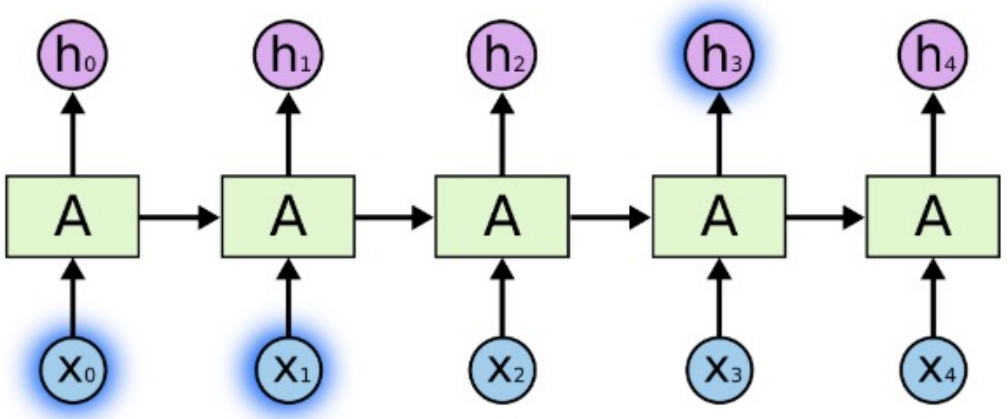


Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right: **(1)** Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). **(2)** Sequence output (e.g. image captioning takes an image and outputs a sentence of words). **(3)** Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). **(4)** Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). **(5)** Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.

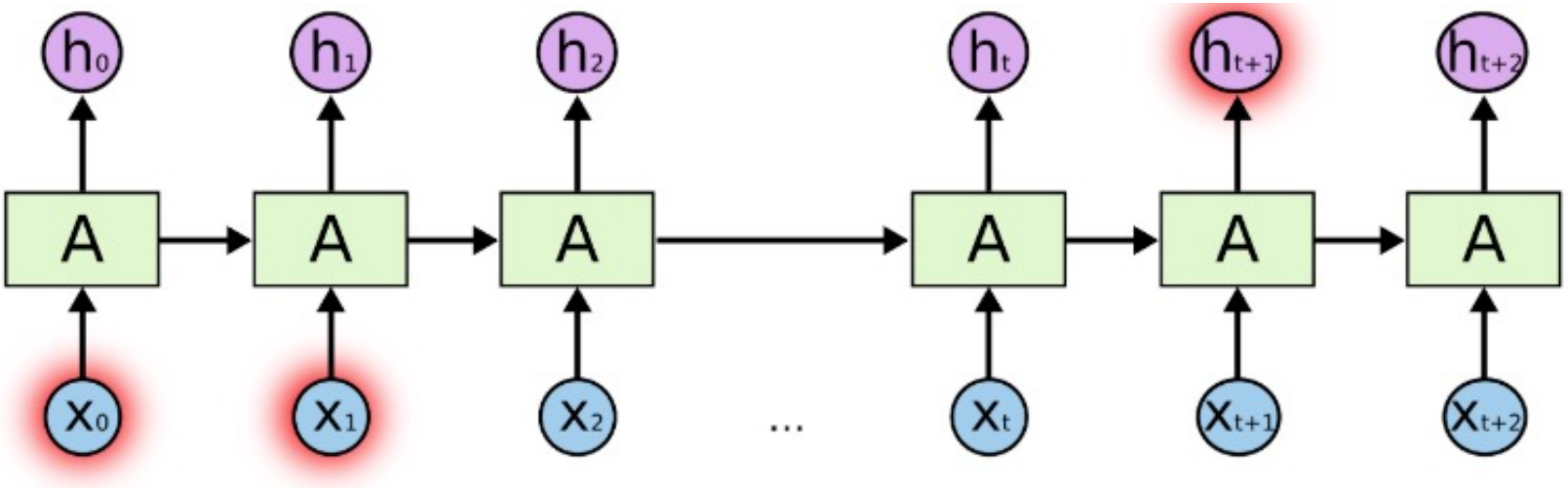
RNNs – how much context do they really capture?

Picture credit:
Christopher Olah

Short contexts are captured well.



For long gaps, we still have a problem.

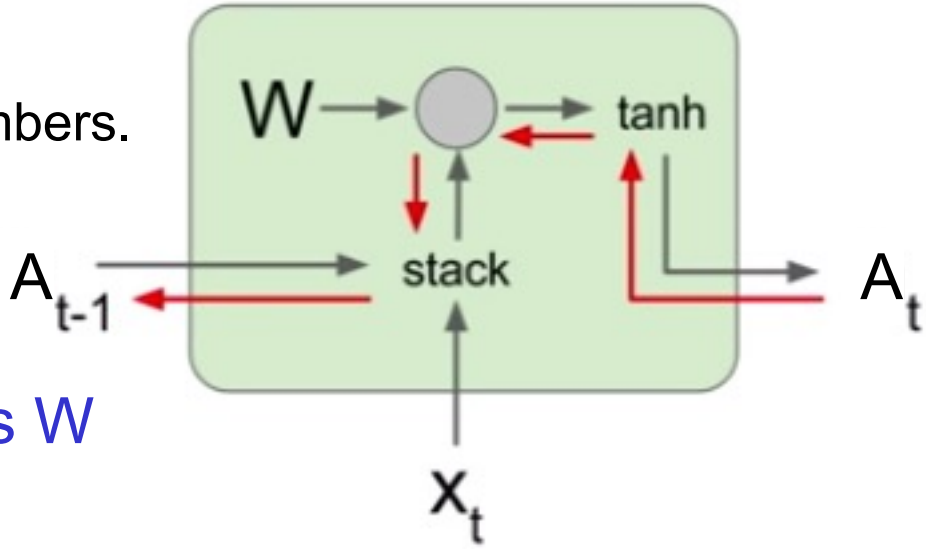


RNNs – how much context do they really capture?

Long contexts can get forgotten, because weights become too small during backpropagation (multiplying many small numbers).

Or we get „exploding gradients“ from multiplying many large numbers.

Picture credit: Justin Johnson



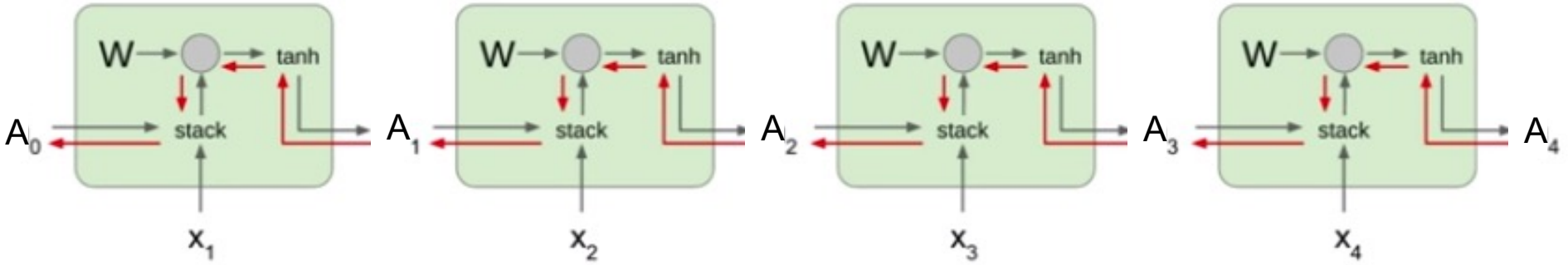
Backpropagation from A_t
To A_{t-1} multiplies by weights W (actually W^T)

RNNs – how much context do they really capture?

Long contexts can get forgotten, because weights become too small during backpropagation (multiplying many small numbers) => „vanishing gradients“.

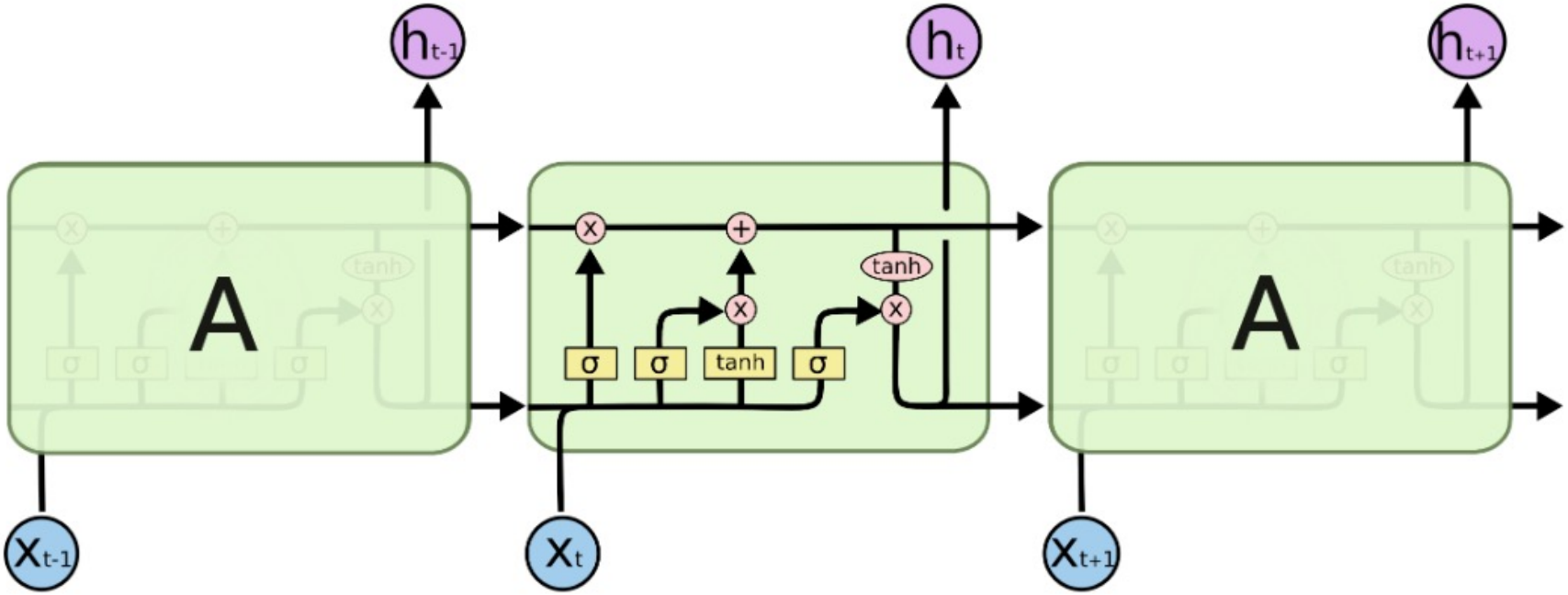
Or we get „exploding gradients“ from multiplying many large numbers.

Picture credit: Justin Johnson



Long Short Term Memory networks (LSTM)

- Proposed by Hochreiter & Schmidhuber (1997)
- An LSTM is a more complicated form of recurrent neural network
- Widely used for language modelling
- Explicitly designed to handle long-term dependencies



The repeating module in an LSTM contains four interacting layers.

Long Short Term Memory networks (LSTM)

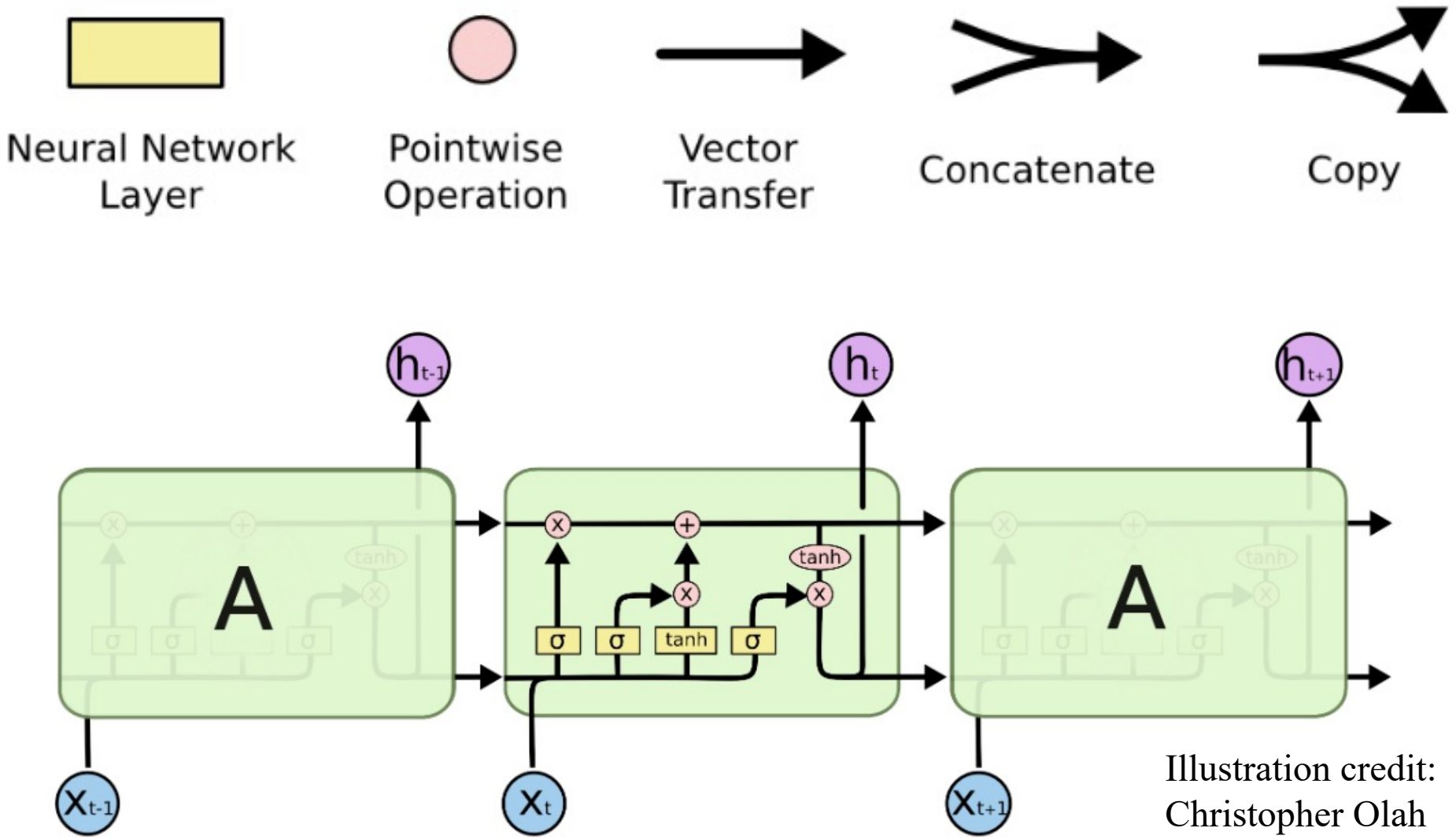


Illustration credit:
Christopher Olah

The repeating module in an LSTM contains four interacting layers.

Long Short Term Memory networks (LSTM)

Core idea:

Cell state C_t avoids the many multiplication by same weight matrix.

The LSTM can remove information from the cell state or add new information; this is regulated by the „gates“.

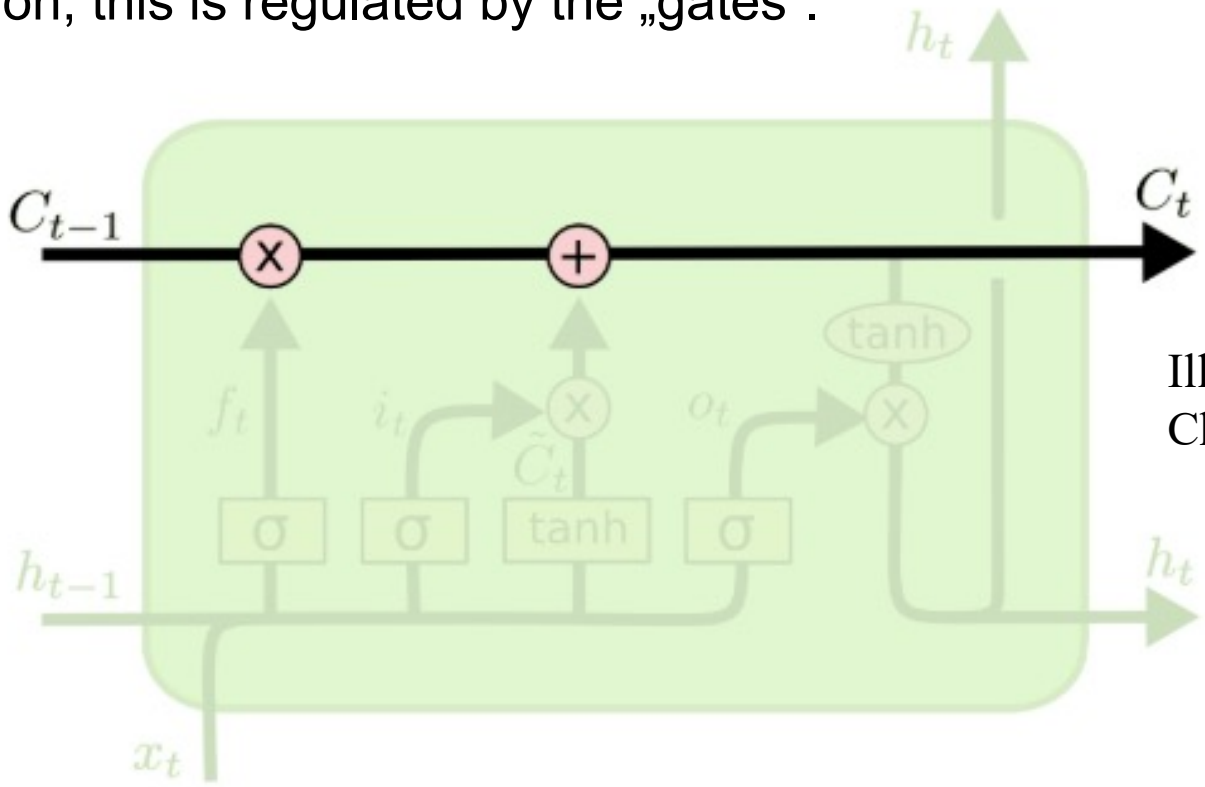


Illustration credit:
Christopher Olah

Long Short Term Memory networks (LSTM)

Gates are a way to control
what information gets through.
They consist of a **sigmoid neural net layer** and
a **pointwise multiplication** operation.

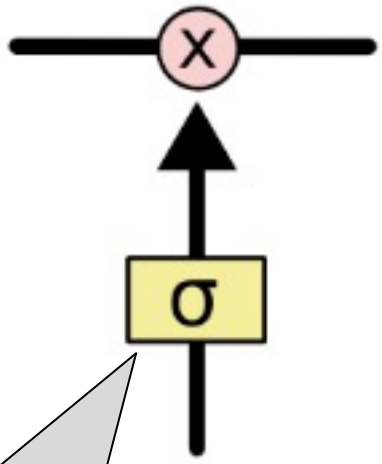
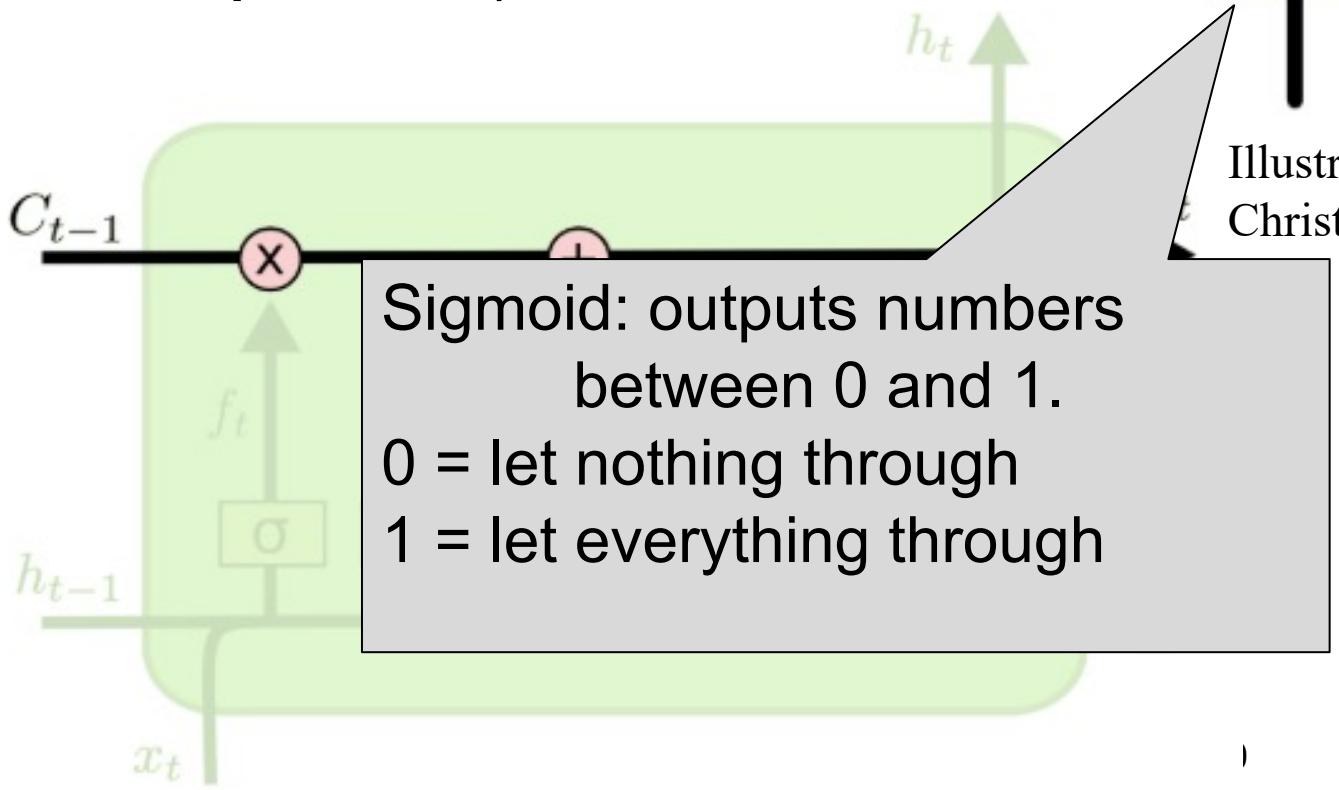


Illustration credit:
Christopher Olah

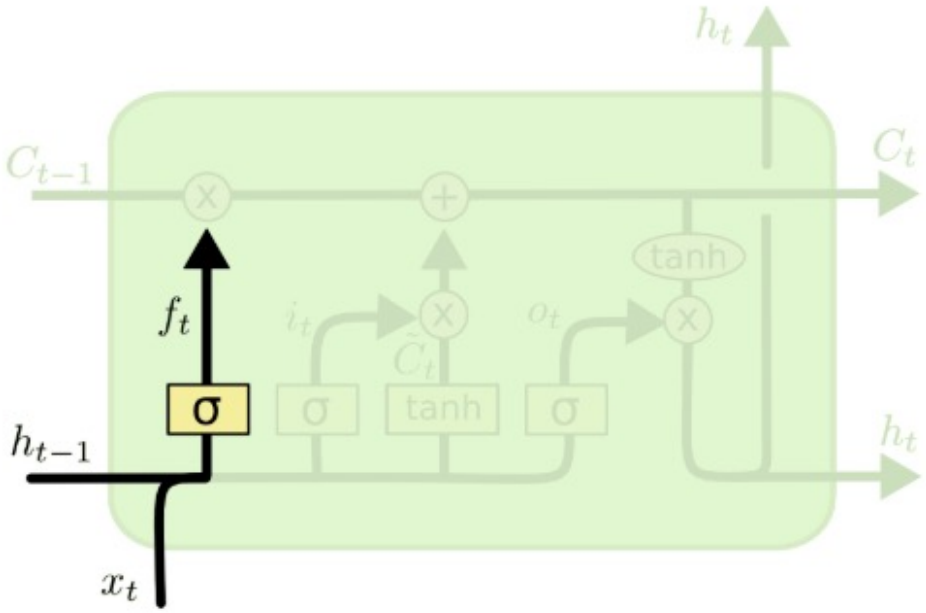


Sigmoid: outputs numbers
between 0 and 1.
0 = let nothing through
1 = let everything through

LSTM “forget gate”

What information from the state h_{t-1} should we forget vs. remember?

- e.g., forget gender of previous noun if we are encountering a new noun at x_t.



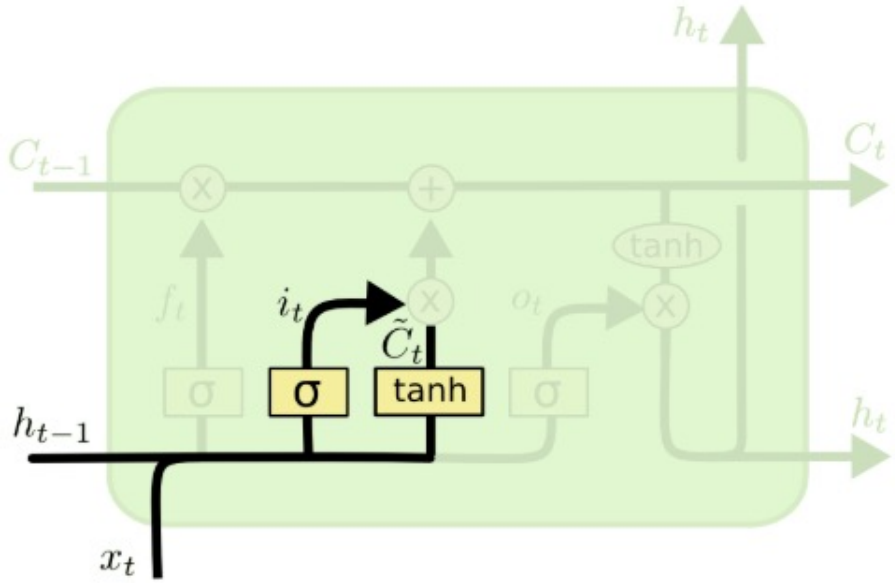
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Illustration credit:
Christopher Olah

LSTM “input gate”

What information from x_t should we add to C_{t-1} to obtain cell state C_t?

- e.g., add gender of new noun if we are encountering a new noun at x_t.



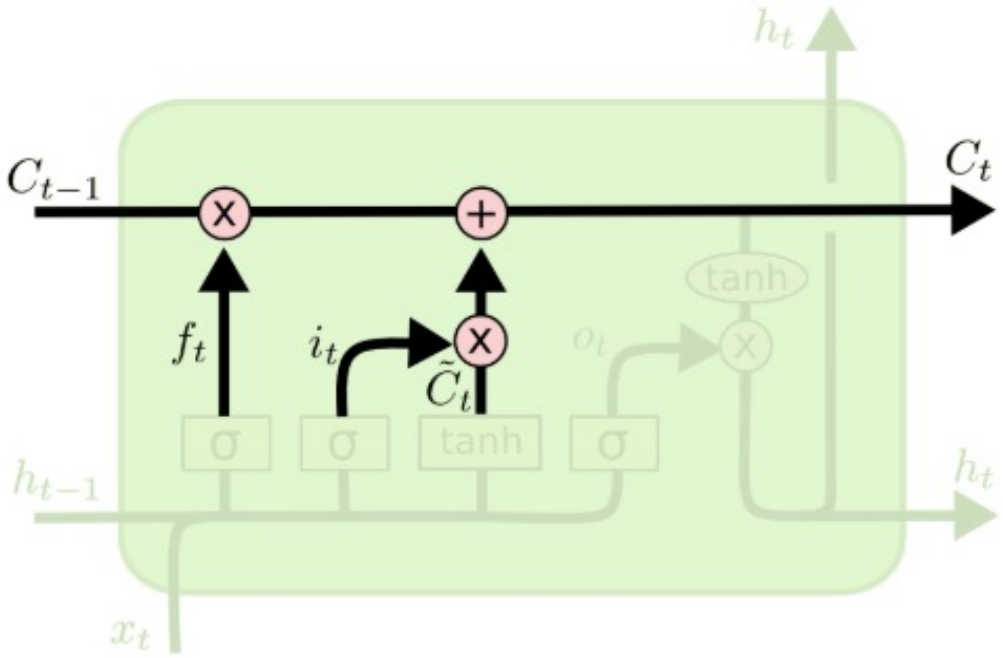
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Illustration credit:
Christopher Olah

LSTM update to cell state $C_{t-1} \rightarrow C_t$

1) Multiply old state by f_t (in order to remove what we want to forget)
2) Add the new contribution from x_t to the cell state.



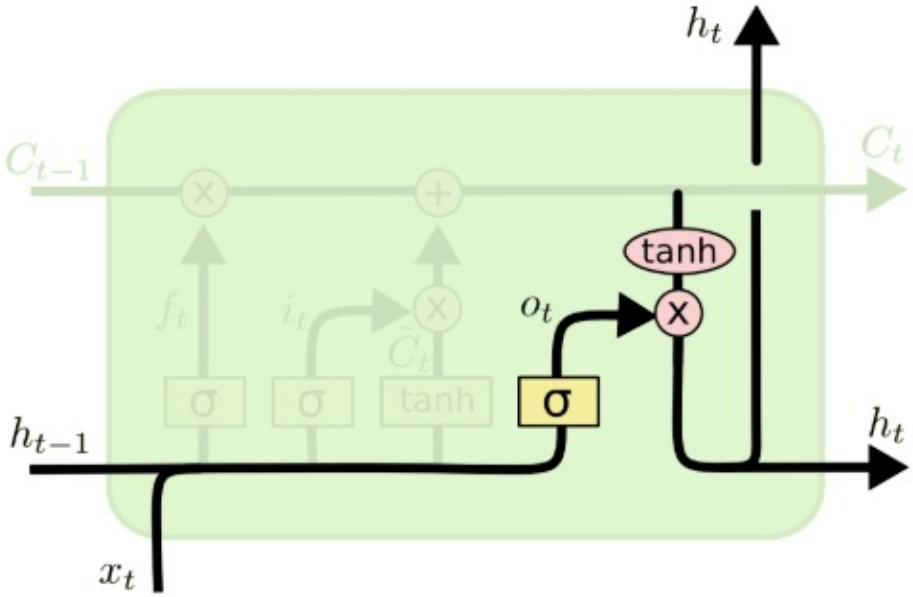
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Illustration credit:
Christopher Olah

LSTM “output gate”

What information from the new cell state should we hand on to predict the target output (and for flowing into the next cell state)?

- e.g., if we just encountered a new noun in subject role, might want to output information that‘s relevant for predicting the verb.



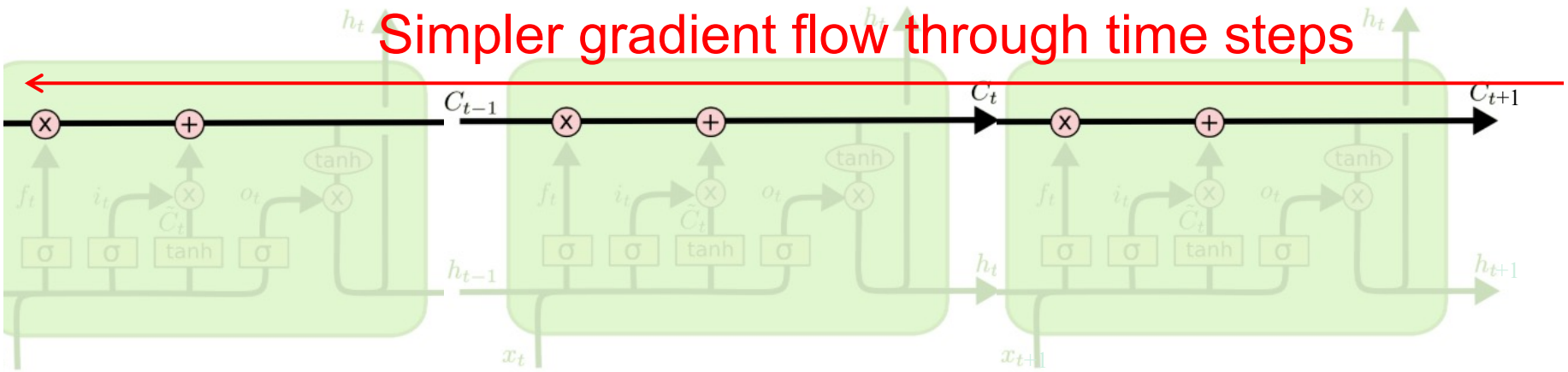
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Illustration credit:
Christopher Olah

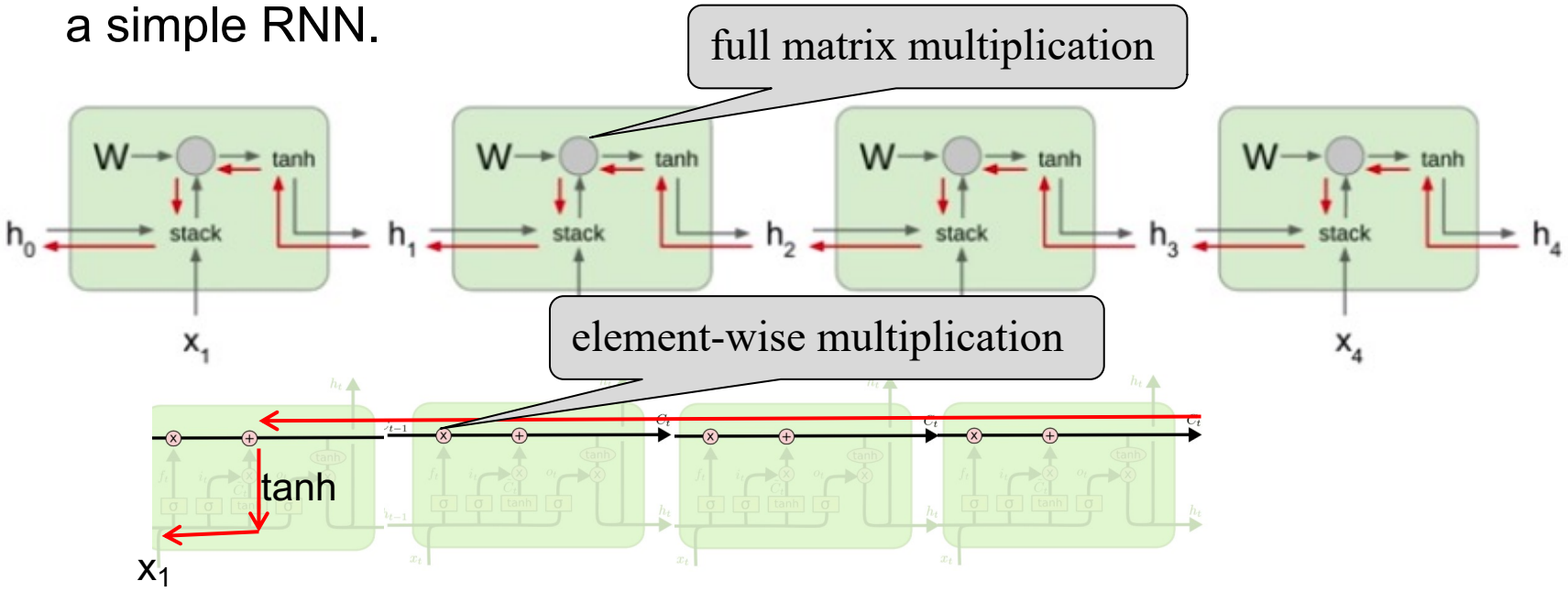
Long Short Term Memory networks (LSTM)

- During back-propagation, gradients flow through cell states with little modification: addition operation and multiply *element-wise* by forget gate
- Forget gate can vary by time stamp, therefore, less likely to have exploding or vanishing gradients.
- Doesn't have to go through *tanh* at each time step during back propagation (just once).
- Updates to weight matrices for gates are local.



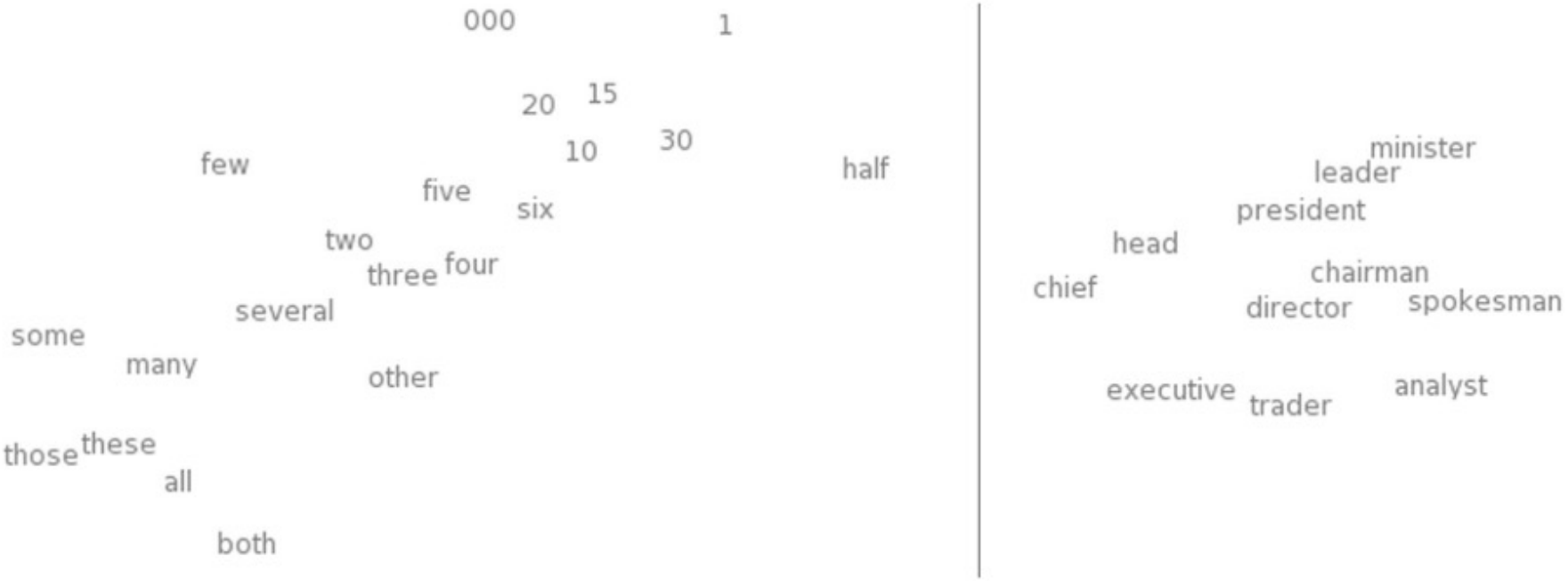
Summary simple RNN vs. LSTM

- RNNs generally allow to represent arbitrarily long contexts
- But a simple RNN has problems with vanishing and exploding gradients because it keeps multiplying with same weight matrix during back prop for each time step.
- LSTM avoids this problem by using the cell state and updating weight matrices more locally.
- LSTM has **a lot more parameters** that it needs to learn compared to a simple RNN.



Useful by-products: embeddings

- Training a simple RNN or an LSTM consists of learning the **weights** (in LSTMs, weight matrices for each of the gates)
- The learned weights can be extracted for each input word, yielding a vector of real numbers for each word, these are called **embeddings.**
- Similar words have been shown to have similar embeddings.



t-SNE visualizations of word embeddings. Left: Number Region; Right: Jobs Region. From Turian *et al.* (2010), see complete image.

Useful by-products: embeddings

- Training a simple RNN or an LSTM consists of learning the **weights** (in LSTMs, weight matrices for each of the gates)
- The learned weights can be extracted for each input word, yielding a vector of real numbers for each word, these are called **embeddings.**
- Similar words have been shown to have similar embeddings.

FRANCE	JESUS	XBOX	REDDISH	SCRATCHED	MEGABITS
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

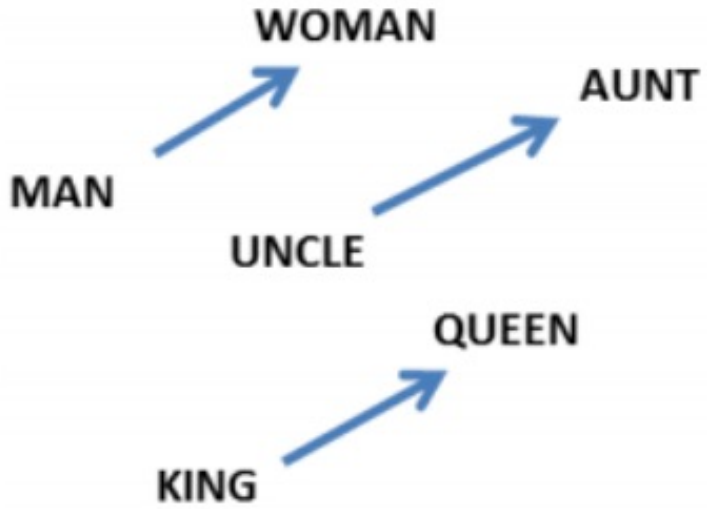
What words have embeddings closest to a given word? From Collobert *et al.* (2011)

Useful by-products: embeddings

- Training a simple RNN or an LSTM consists of learning the **weights** (in LSTMs, weight matrices for each of the gates)
- The learned weights can be extracted for each input word, yielding a vector of real numbers for each word, these are called **embeddings.**
- Similar words have been shown to have similar embeddings.

- This property can be exploited for analogy tasks:

  W("woman") – W("man") ≈
  W("queen") – W("king") ≈
  W("aunt") – W("uncle")



From Mikolov *et al.* (2013a)

Useful by-products: embeddings

Embeddings have been found to capture highly sophisticated relationships between words.

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Relationship pairs in a word embedding. From Mikolov *et al.* (2013b).

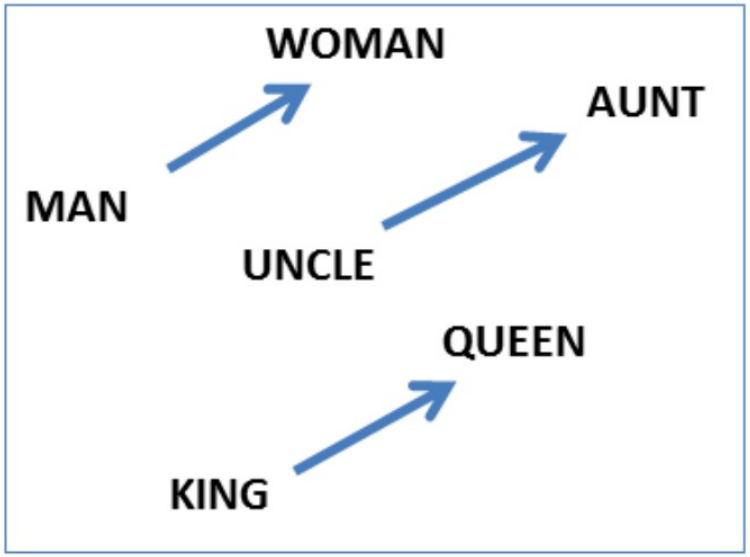
Useful by-products: embeddings

- Embeddings have been found to capture highly sophisticated relationships between words.
- They are therefore very useful for most NLP tasks, as they capture syntactic as well as semantic information about words.
- There exist context-independent embeddings for words (each word has one embedding independent of its context)
- and context-dependent word embeddings (these work better).
- Word embeddings are often used to initialize representations for words when learning a network for a new task.
- This saves a lot of compute time, and improves performance substantially if limited training data is available for the target task.

Syntaktische und Semantische Regularitäten in RNNs repräsentiert

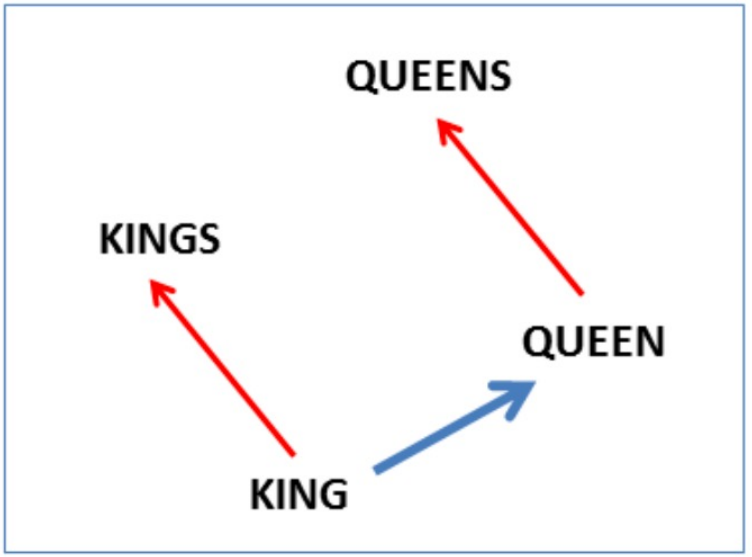
king – man + woman = ?

queen



queens – queen + king= ?

kings

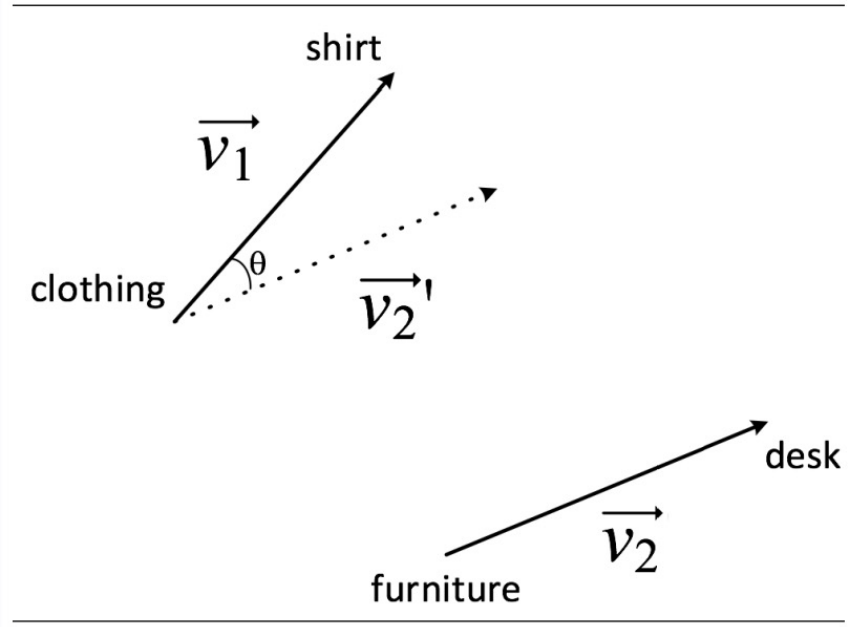


(Mikolov et al., 2013)

Syntaktische und Semantische Regularitäten in RNNs repräsentiert

Clothing is to shirt as furniture is to?

clothing – shirt + furniture = ? desk



(Mikolov et al., 2013; For a discussion, see Levy and Goldberg 2014)

If you'd like to learn more

I recommend Yoshua Bengio's book:

- http://www.deeplearningbook.org/