Seminar on language technologies: deep learning

Eneko Agirre, Gorka Azkune
Ander Barrena, Oier Lopez de Lacalle
@eagirre @gazkune @4nderB @oierldl #dl4nlp
http://ixa2.si.ehu.eus/eneko/dl4nlp

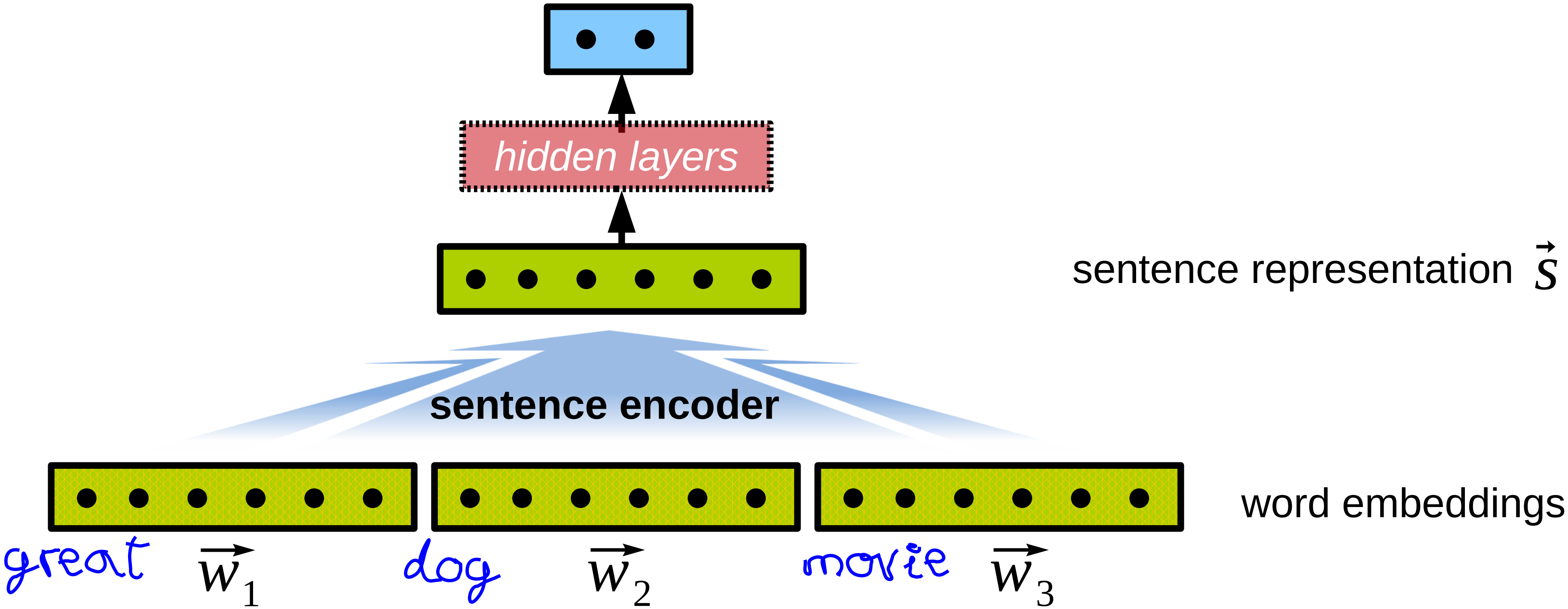
Sess. 4: Sequence-to-Sequence and Machine Translation

Plan for the course

- Introduction: machine learning and NLP
- Multilayer perceptron
- Word representation and
  Recurrent neural networks (RNN)
- **Sequence-to-Sequence (seq2seq) and Machine Translation**
- Attention, transformers and
  Natural language inference
- Pre-trained transformers, BERT, GPT
- Bridging the gap between natural languages and the visual world

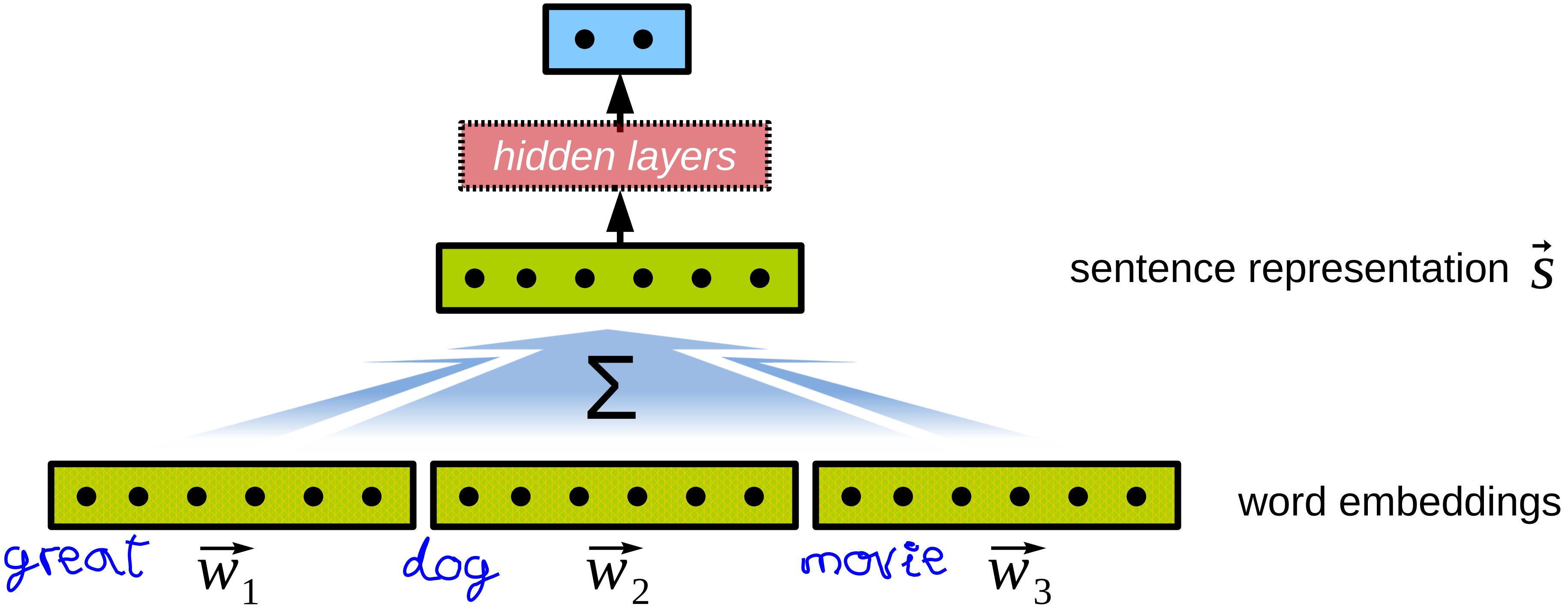
Previously...

Sentence encoder:
input a sequence of word embeddings $\vec{w}_i \in \mathbb{R}^D$
output a sentence representation $\vec{s} \in \mathbb{R}^{D'}$



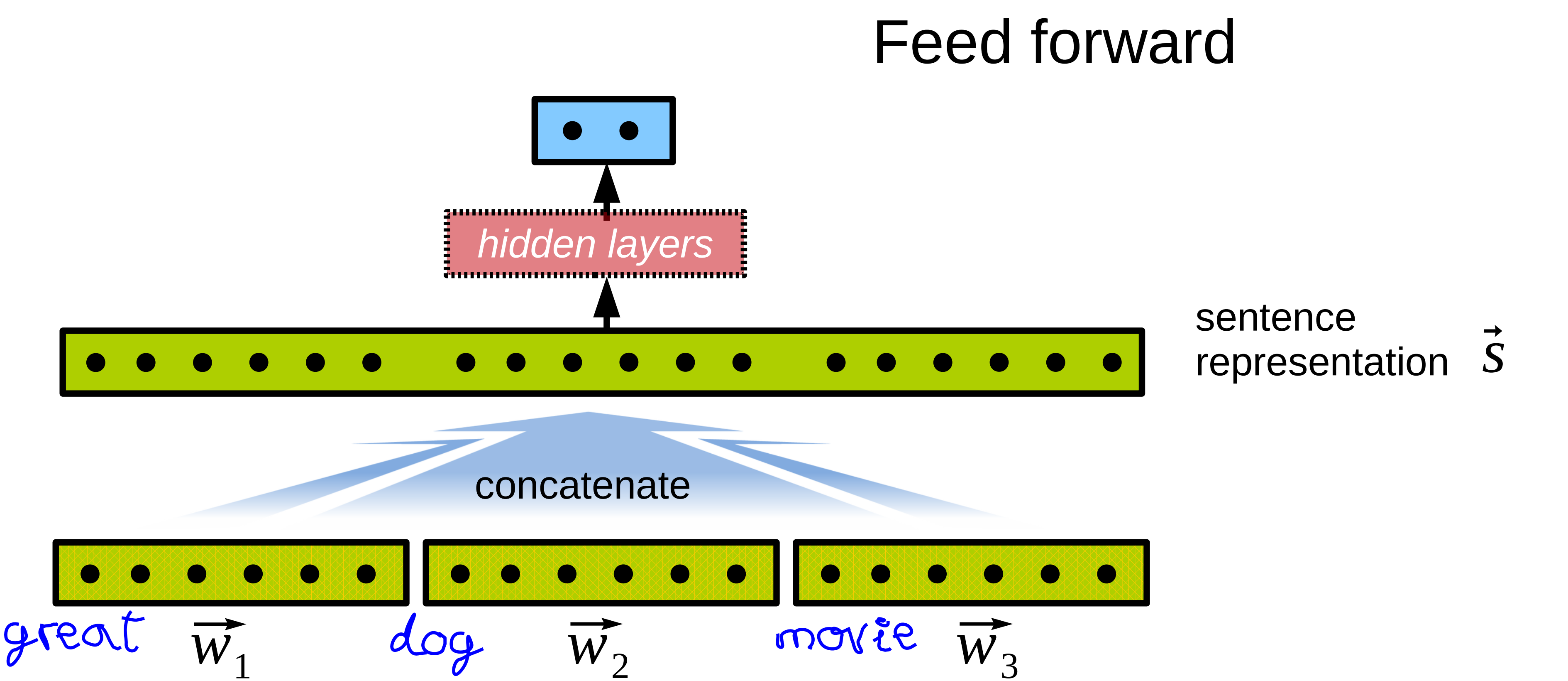
Previously…

Sentence encoder 1: Continuous bag of words

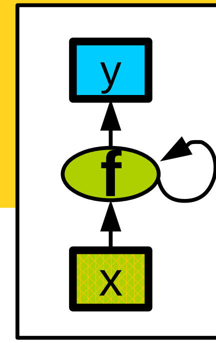


Previously...

Sentence encoder 2: Add word order with concatenation

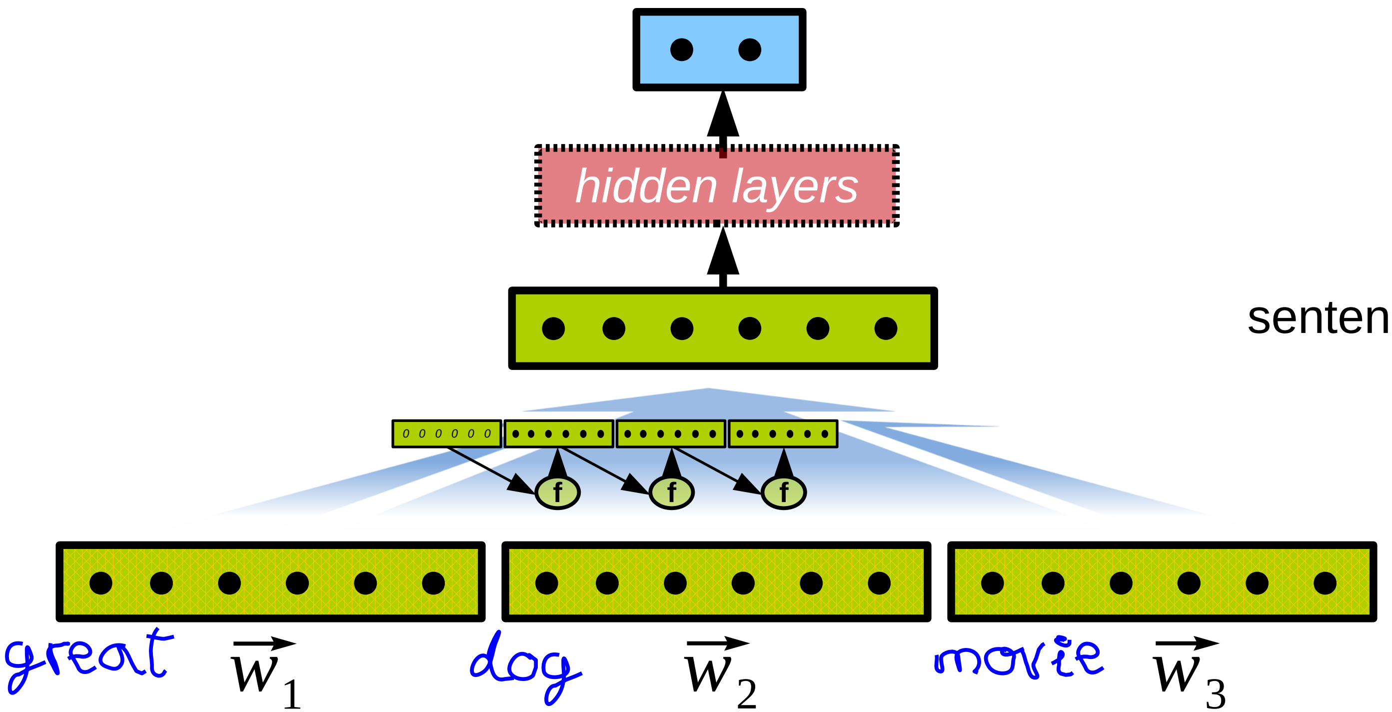


Previously…



Sentence encoder 3: RNN

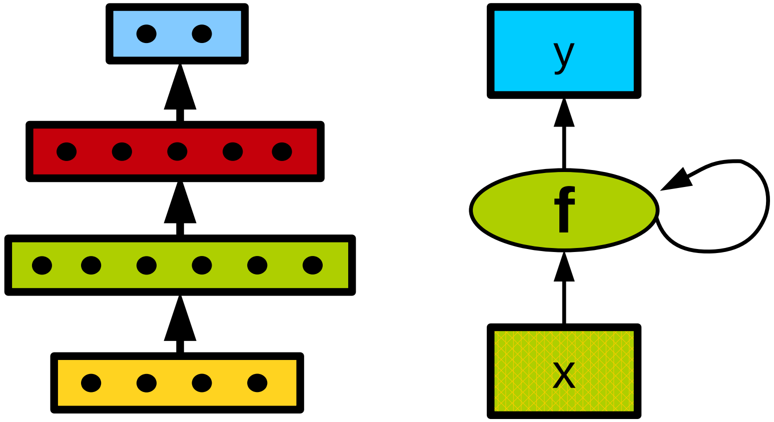
$$\vec{h}_t = \tanh\left(W\left[\vec{h}_{t-1}, \vec{w}_t\right] + \vec{b}\right)$$



sentence representation $\vec{s}$

word embeddings

Previously...



Topology: number and size of layers

Non-linearity

Optimizer

Learning-rate

Size of mini-batch

Weight of L2 regularization

Dropout rate

Plan for this session

- Application of RNN:
  - Language Models (sentence encoders)
  - Language Generation (sentence decoders)
  - Sequence to sequence models & Neural Machine Translation (I)
- Problems with gradients in RNN LSTM and GRU

Language Models

- Goal: assign probability to a sentence
- Useful for:
  - Machine Translation: ordering, word choice
    p(the cat is small) > p(small the is cat)
    p(high winds tonight) > p(large winds tonight)
  - Spell correction:
    p(15 minutes from my house) > p(15 minuets from my house)
  - Speech recognition:
    p(I saw a van) > p(eyes awe of an)
  - PRE-TRAINING and transfer learning!

Language Models

Estimate probability of a sequence of words

$$p(w_1, \ldots, w_m) = \prod_{t=1}^{m} p(w_t | w_1, \ldots, w_{t-1})$$

p(“language models are cool”) =

p(language| <s>) x

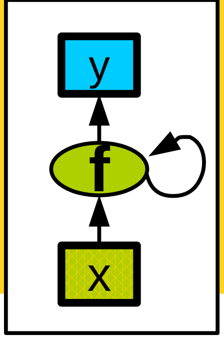
p(models | <s>, language) x

p(are | <s>, language, models) x

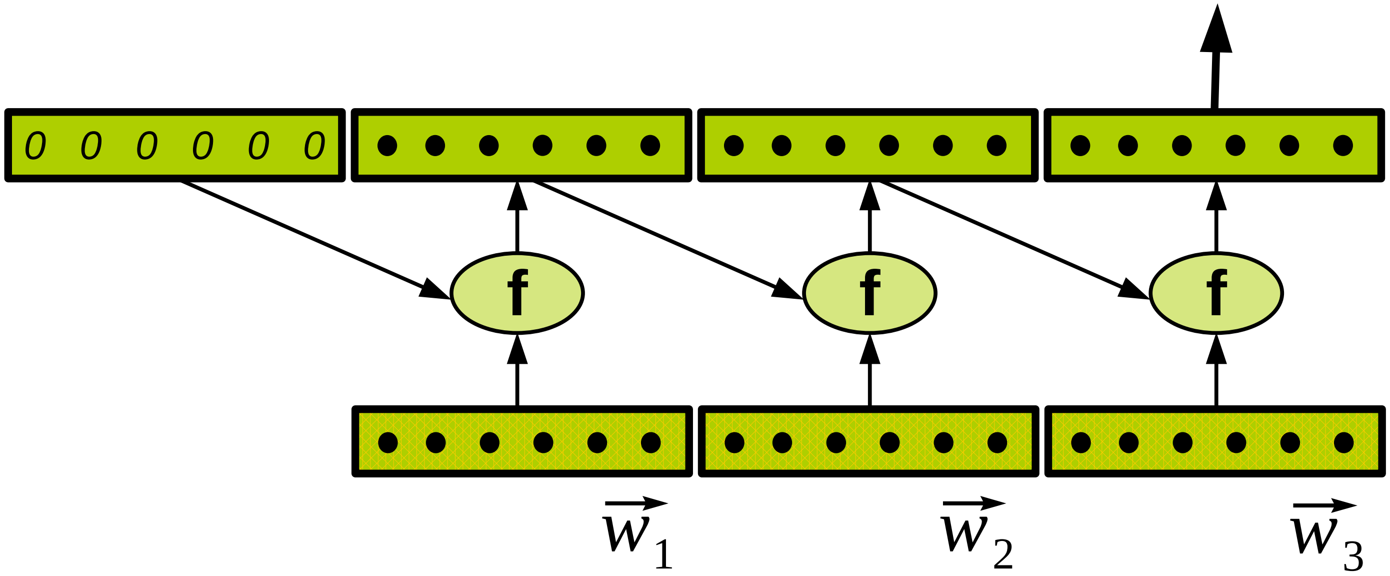
p(cool | <s>, language, models, are) x

p(</s> | <s>, language, models, are, cool)

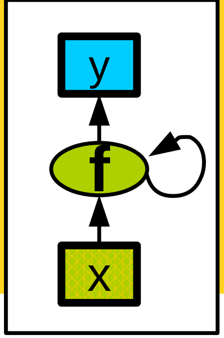
LM with RNN



Add classifier
at each step:

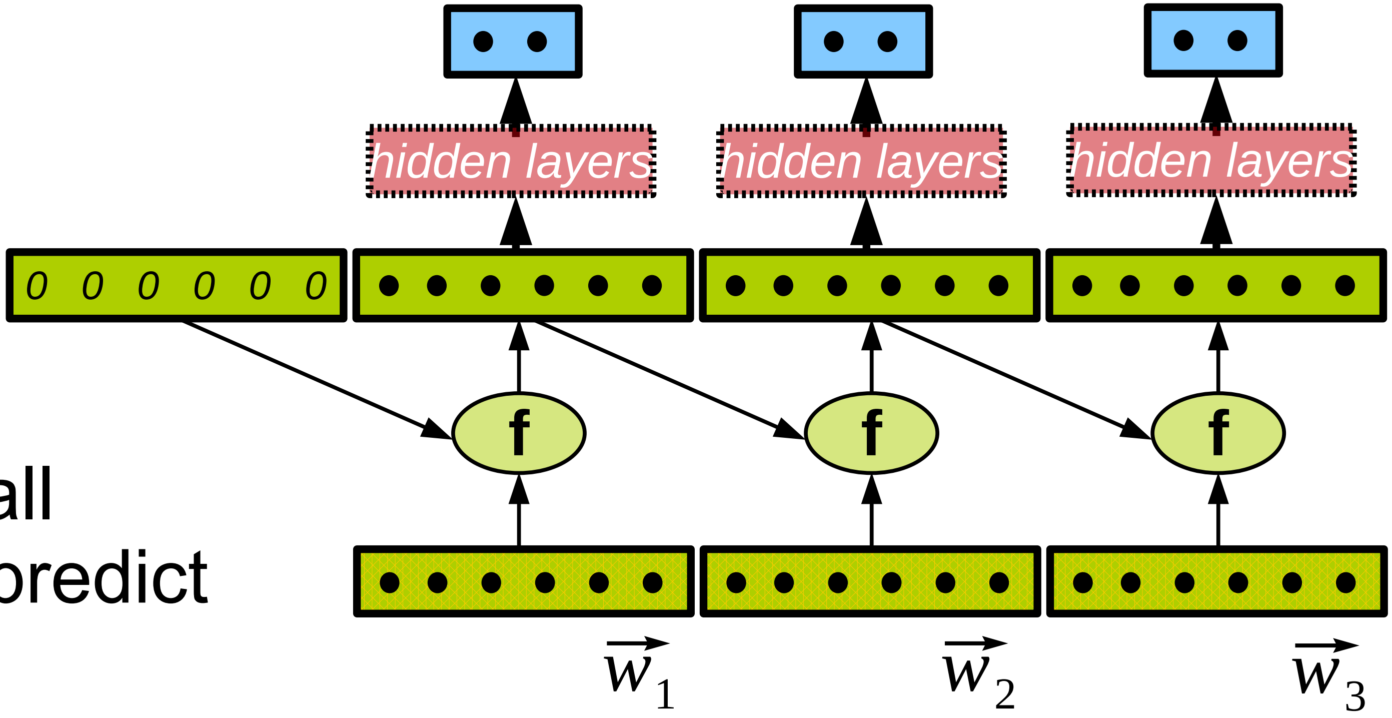


LM with RNN

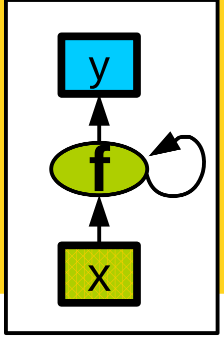


Add classifier at each step:

Softmax over all vocabulary to predict next word

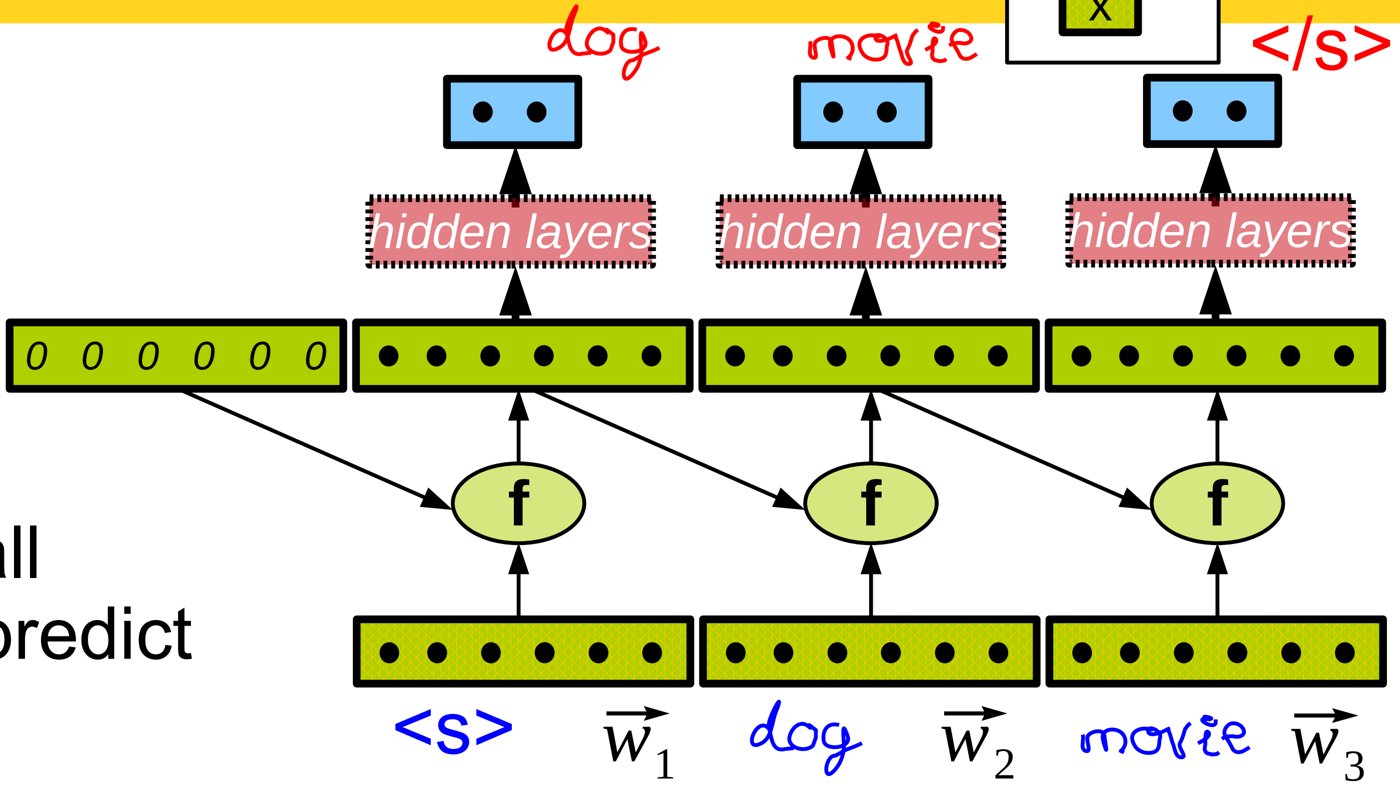


LM with RNN



Add classifier at each step:

Softmax over all vocabulary to predict next word



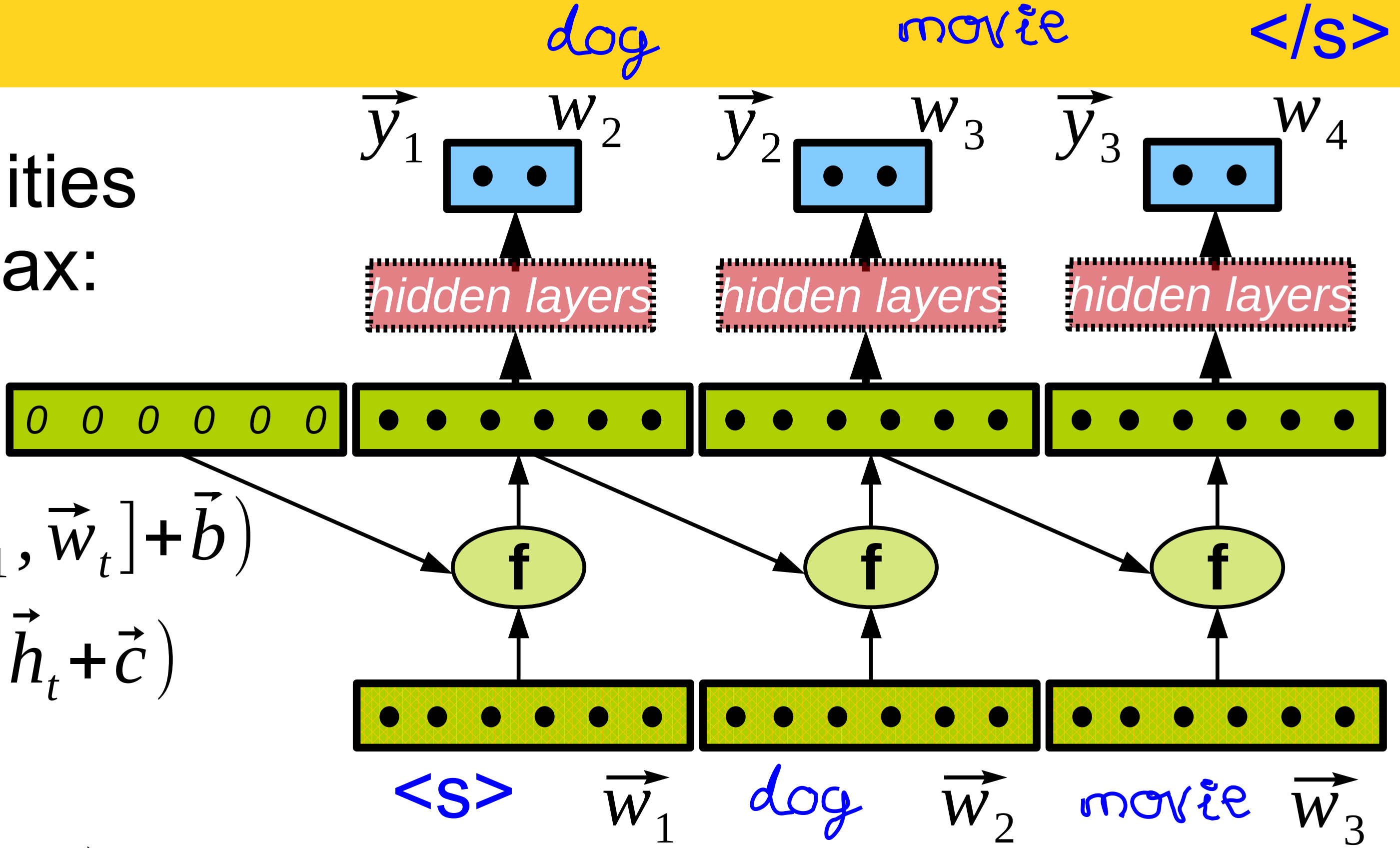
Add sentence boundary tokens: <s> and </s>

LM with RNN

Token probabilities given by softmax:

$$\vec{h}_t = \tanh(W[\vec{h}_{t-1}, \vec{w}_t] + \vec{b})$$

$$\hat{y}_t = softmax(W^S \vec{h}_t + \vec{c})$$

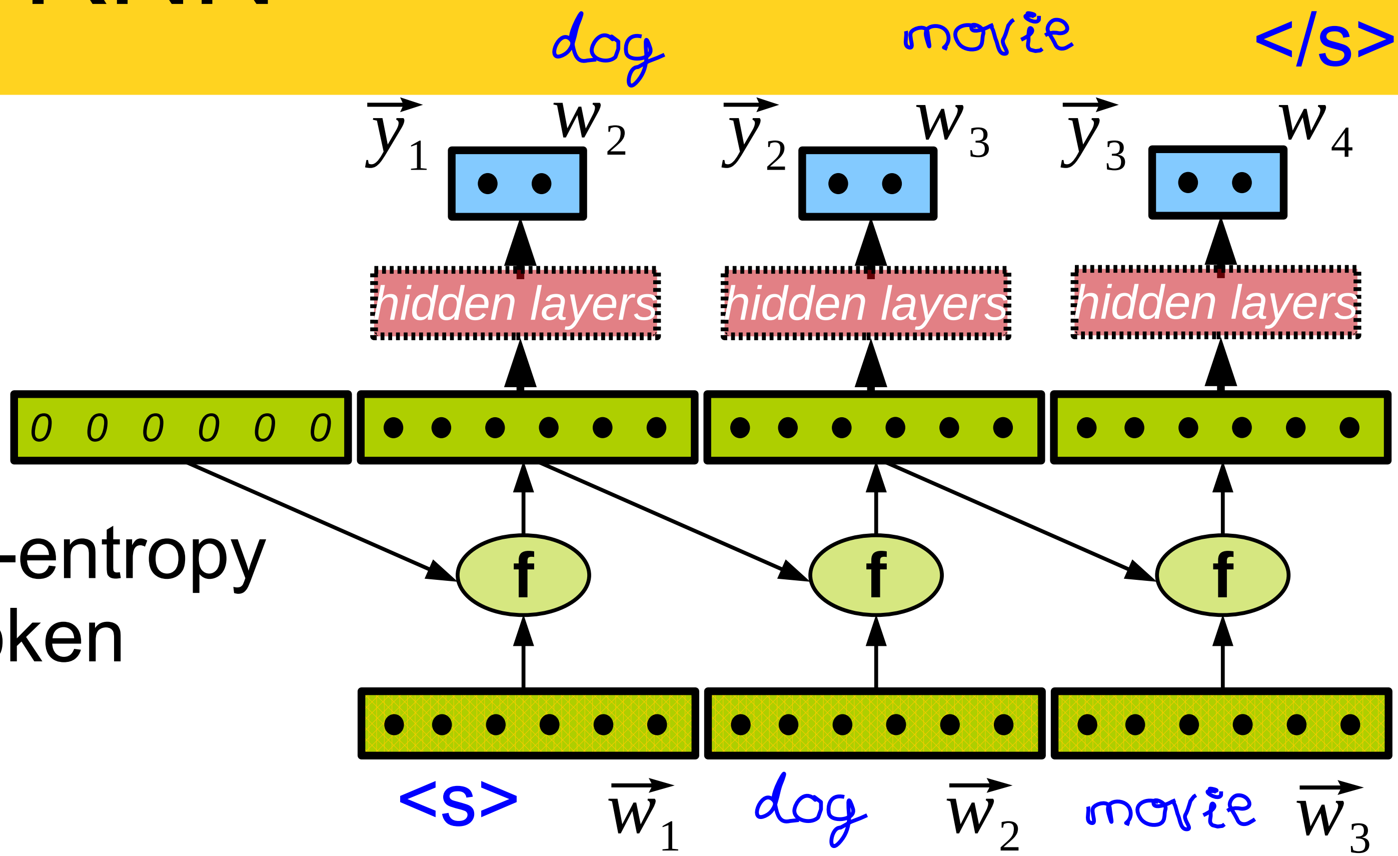


$$p(w_{t+1} = \hat{w}_j | w_1, \ldots, w_t) = \hat{y}_{t,j}$$

$$p(w_1, \ldots, w_m) \approx \prod_{t=0}^{m-1} \hat{y}_{t,j}$$

LM with RNN

Loss function:



Mean of cross-entropy loss of each token

$$J_{\text{sentence}} = -\frac{1}{T}\sum_{t=1}^{m} \log \hat{y}_{t,correct}$$

LM with RNN

- Does it really work better?
  Evaluation with perplexity 2^J (lower is better)

Model	Parameters	Validation	Test
Mikolov & Zweig (2012) - KN-5	2M‡	–	141.2
Mikolov & Zweig (2012) - KN5 + cache	2M‡	–	125.7
Mikolov & Zweig (2012) - RNN	6M‡	–	124.7
Mikolov & Zweig (2012) - RNN-LDA	7M‡	–	113.7
Mikolov & Zweig (2012) - RNN-LDA + KN-5 + cache	9M‡	–	92.0

Table 1. Single model perplexity on validation and test sets for the Penn Treebank language modeling task.

Source: (Merity et al. 2018)

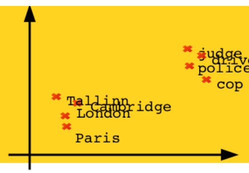
LM with RNN

- Mikolov again, is there any connection with word embeddings models like CBOW and skip-gram?

LM with RNN

- Mikolov again, is there any connection with word embeddings models like CBOW and skip-gram?

Word embeddings



General task with large quantities of data: **guess the missing word** (language models)

CBOW: given context guess middle word

… people who keep pet dogs or ***cats*** *exhibit better mental and physical health …*

SKIP-GRAM given middle word guess context

… people who keep pet dogs or ***cats*** *exhibit better mental and physical health …*

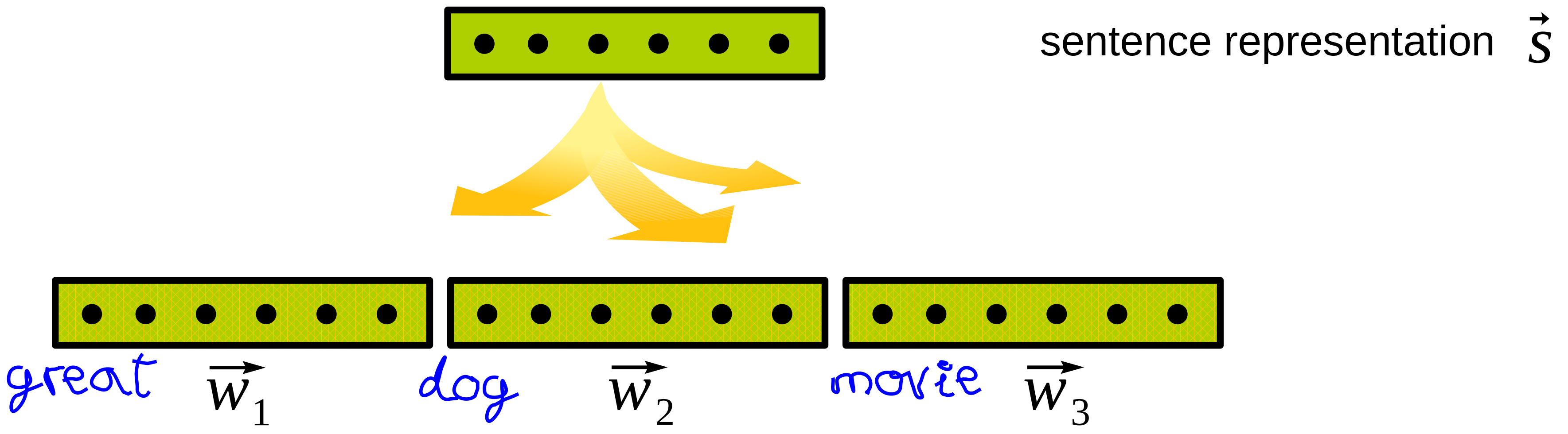
Proposed by Mikolov et al. (2013)

Plan for this session

- Application of RNN:
  - Language Models (sentence encoders)
  - **Language Generation (sentence decoders)**
  - Sequence to sequence models &
    Neural Machine Translation (I)
- Problems with gradients in RNN
  LSTM and GRU

Language generation: sentence decoders

- Can we reverse language models and generate language?

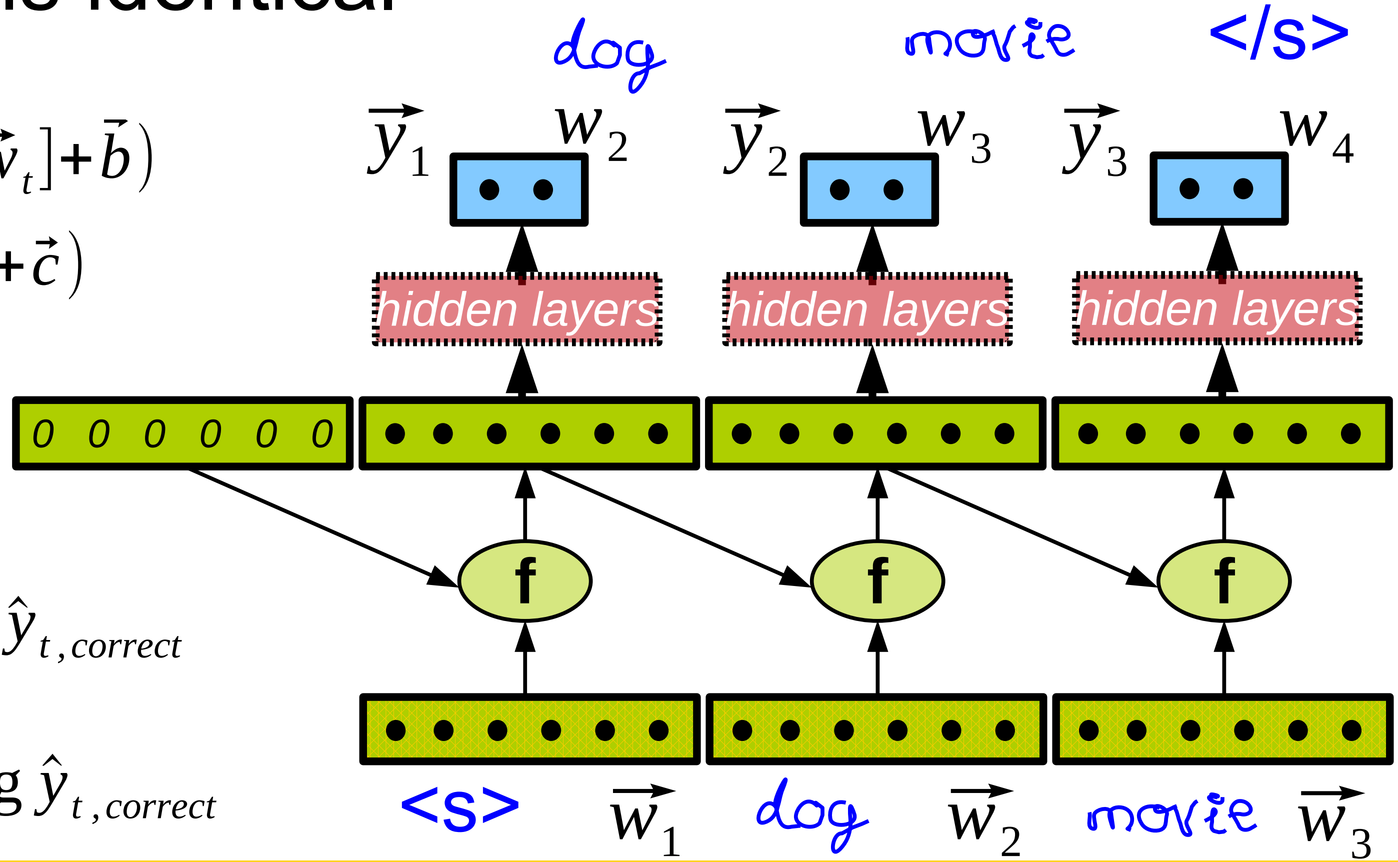


Language generation: sentence decoders

- We can reuse language models
  Training is identical

$$\vec{h}_t = \tanh\left(W[\vec{h}_{t-1}, \vec{w}_t] + \vec{b}\right)$$

$$\hat{y}_t = softmax\left(W^S \vec{h}_t + \vec{c}\right)$$



$$p(w_{1,} \ldots, w_m) \approx \prod_{t=0}^{m-1} \hat{y}_{t,correct}$$

$$J_{\text{sentence}} = -\frac{1}{T} \sum_{t=1}^{m} \log \hat{y}_{t,correct}$$

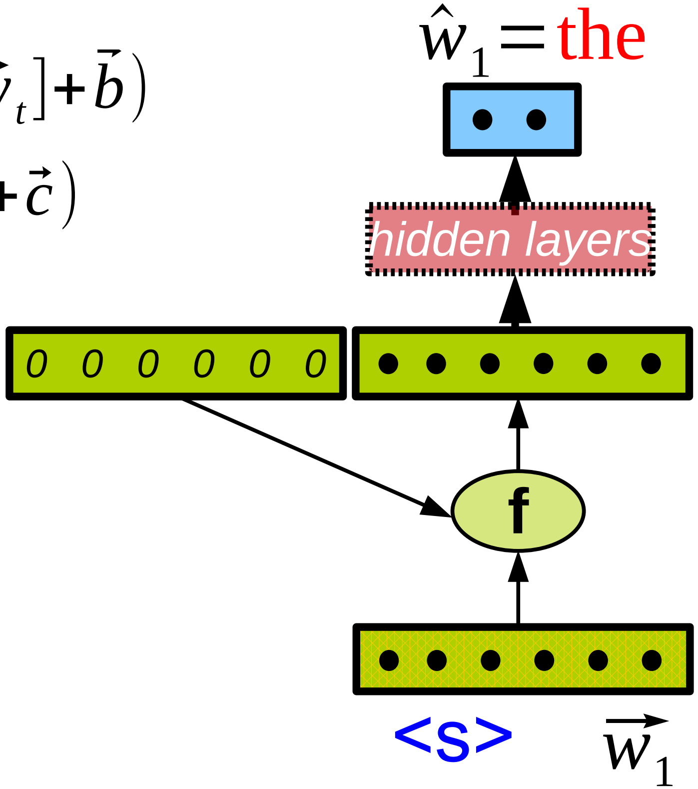
Language generation: sentence decoders

- Initialize history and run RNN one by one

$$\vec{h}_t = \tanh(W[\vec{h}_{t-1}, \vec{w}_t] + \vec{b})$$

$$\hat{y}_t = softmax(W^S \vec{h}_t + \vec{c})$$

$$\hat{w}_t = \underset{i}{argmax}\, \hat{y}_{t,i}$$



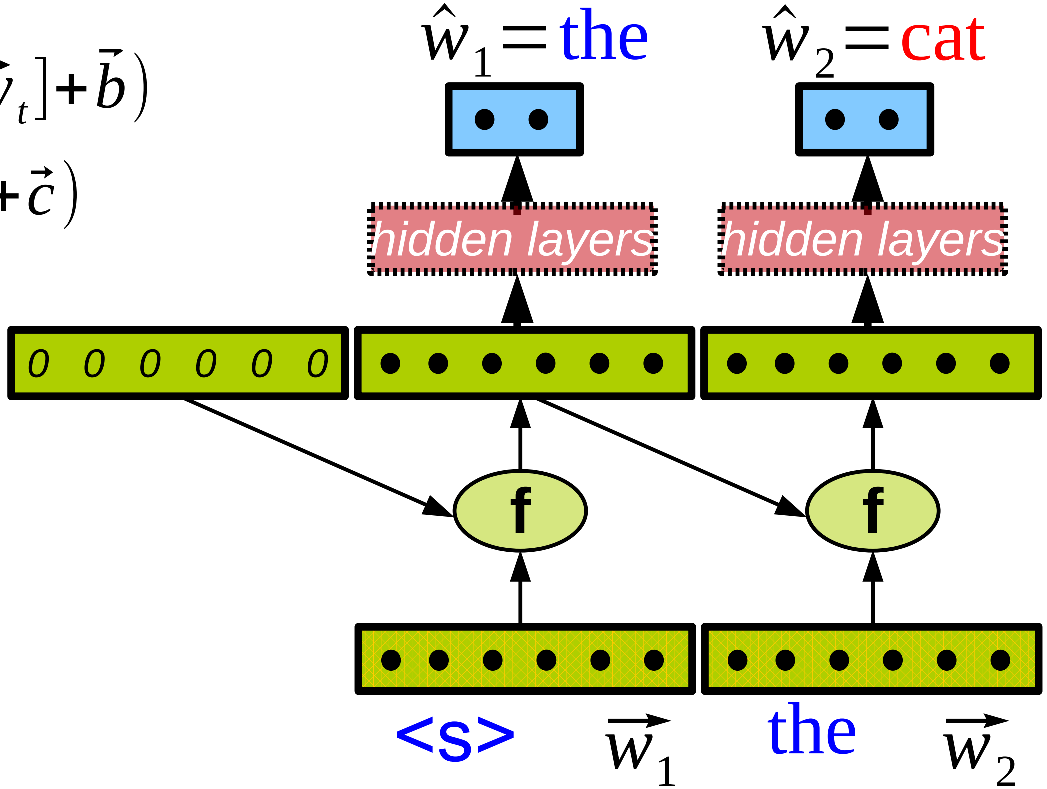
Language generation: sentence decoders

- Run RNN one by one

$$\vec{h}_t = \tanh(W[\vec{h}_{t-1}, \vec{w}_t] + \vec{b})$$

$$\hat{y}_t = softmax(W^S \vec{h}_t + \vec{c})$$

$$\hat{w}_t = \underset{i}{argmax}\, \hat{y}_{t,i}$$



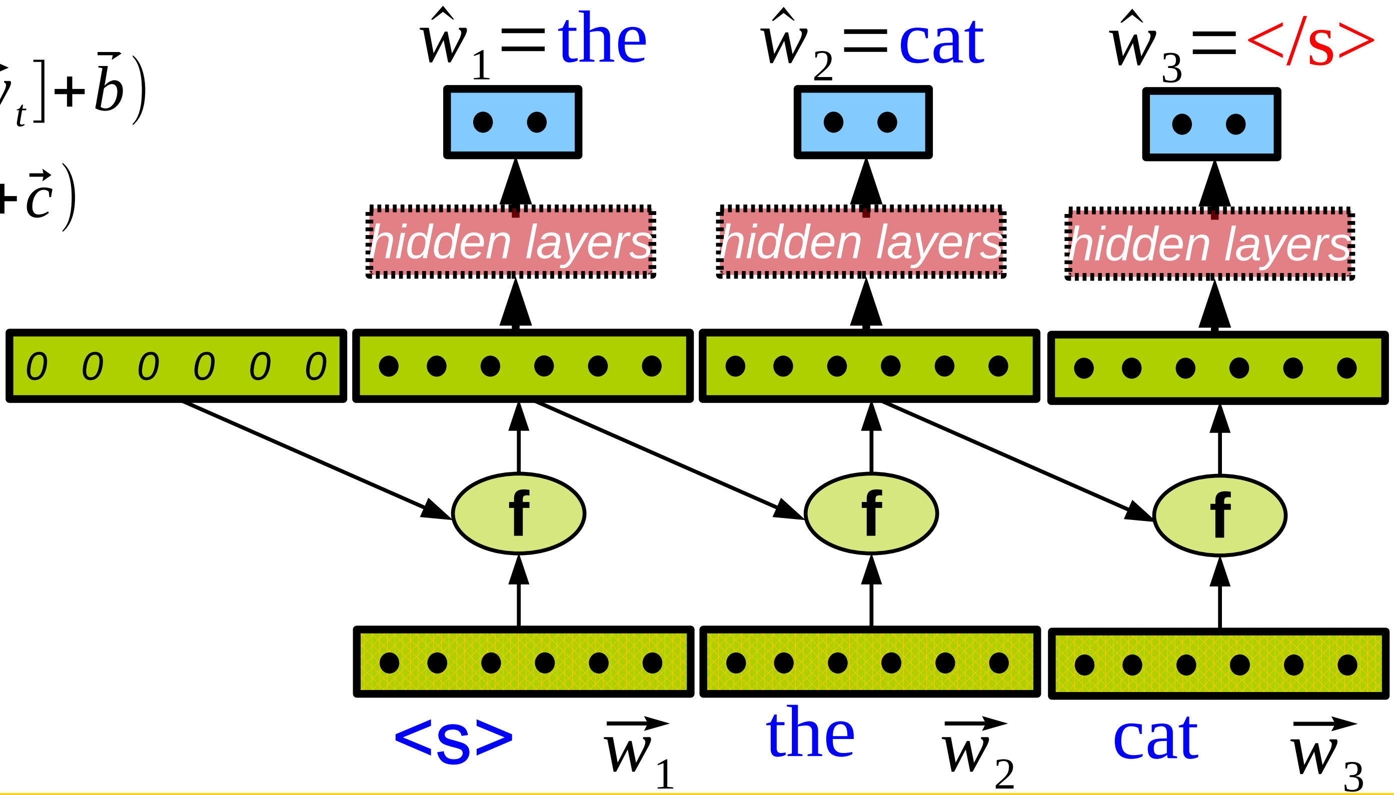
Language generation: sentence decoders

- Run RNN one by one

$$\vec{h}_t = \tanh(W[\vec{h}_{t-1}, \vec{w}_t] + \vec{b})$$

$$\hat{y}_t = softmax(W^S \vec{h}_t + \vec{c})$$

$$\hat{w}_t = \underset{i}{argmax}\, \hat{y}_{t,i}$$



Language generation: sentence decoders

- Sentence decoder
  1) Train a language model
  2) Initialize history
  3) First word is <s>
  4) Run the RNN cell
  5) Generate w, the word with max. softmax weight
  6) Set current word to w
  7) If w is </s> stop, otherwise goto 4

Language generation: sentence decoders

- Sentence decoder
  1) Train a language model
  2) Initialize history
  3) First word is <s>
  4) Run the RNN cell
  5) Generate w, the word with max. softmax weight
  6) Set current word to w
  7) If w is </s> stop, otherwise goto 4
- It always generates the same sequence!

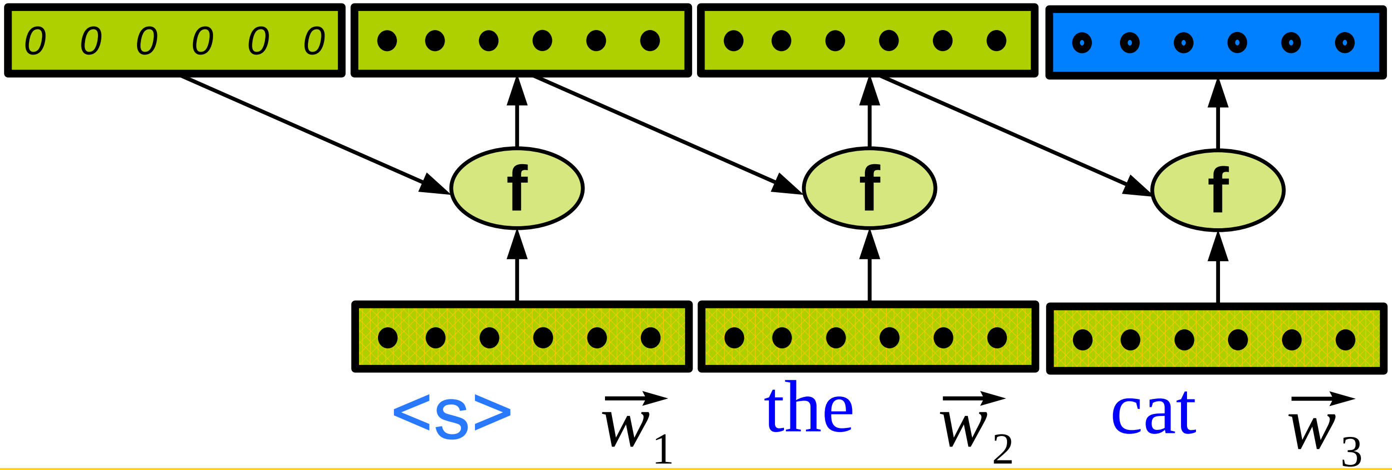
➔ Generate w by sampling (instead of max.)

Language generation: sentence decoders

- What if we put something meaningful in the starting history?
  - A history after reading a sentence:
    - **Language generation after a cue** (sentence completion, etc.)

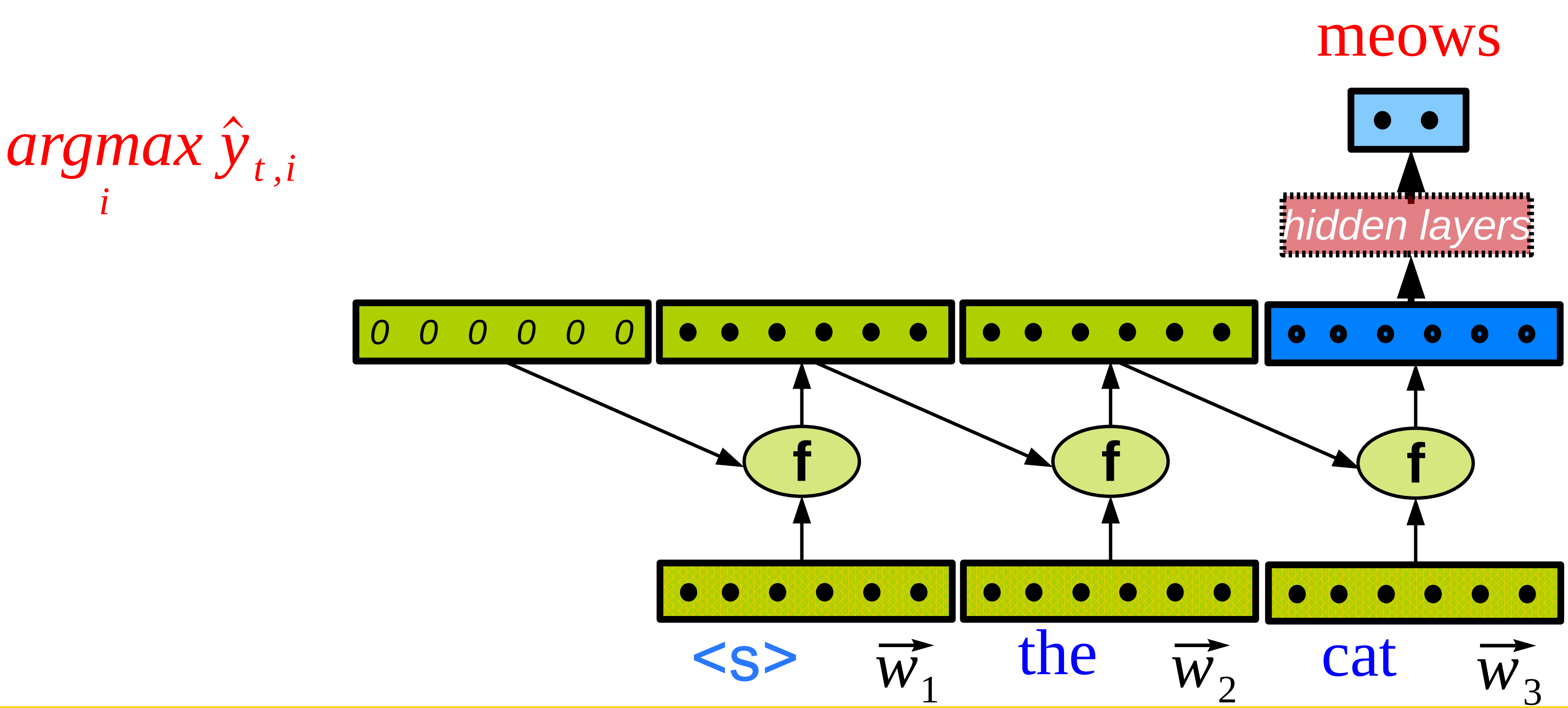
Language generation: conditional recurrent language model

- Feed input to LM in encoder mode



Language generation: conditional recurrent language model

- Feed input to LM in encoder mode
- Run LM in decoder mode from there



Language generation: sentence decoders

- What if we put something meaningful in the starting history?
  - A history after reading a sentence:
    - Language generation after a cue (sentence completion, etc.)
  - A dense representation of an image:
    - Caption generation!
  - A dense representation of a sentence in a foreign language:
    - Machine Translation!
  - …
- Conditional recurrent language models
  - Conditioned on input representation (start history)
  - It needs to be trained with the appropriate inputs and outputs
  - Regular decoder, but the start history is input in every time step

Language generation: Conditional recurrent language model

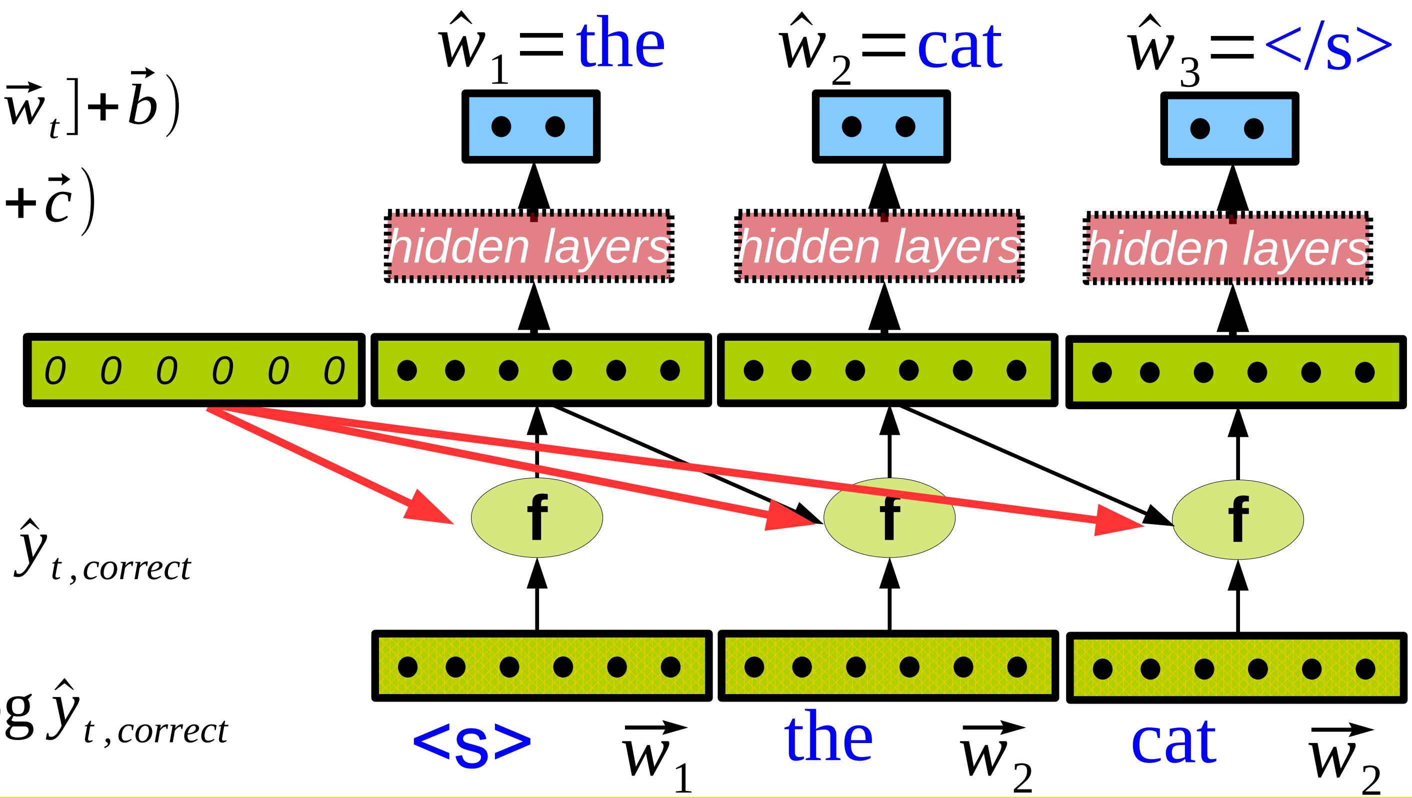
- Train (unconditional)

$$\vec{h}_t = \tanh(W[\vec{h}_{t-1}, \vec{w}_t] + \vec{b})$$

$$\hat{y}_t = softmax(W^S \vec{h}_t + \vec{c})$$

$$p(w_{1,} \ldots, w_m) \approx \prod_{t=0}^{m-1} \hat{y}_{t,correct}$$

$$J_{\text{sentence}} = -\frac{1}{T} \sum_{t=1}^{m} \log \hat{y}_{t,correct}$$



Language generation: Conditional recurrent language model

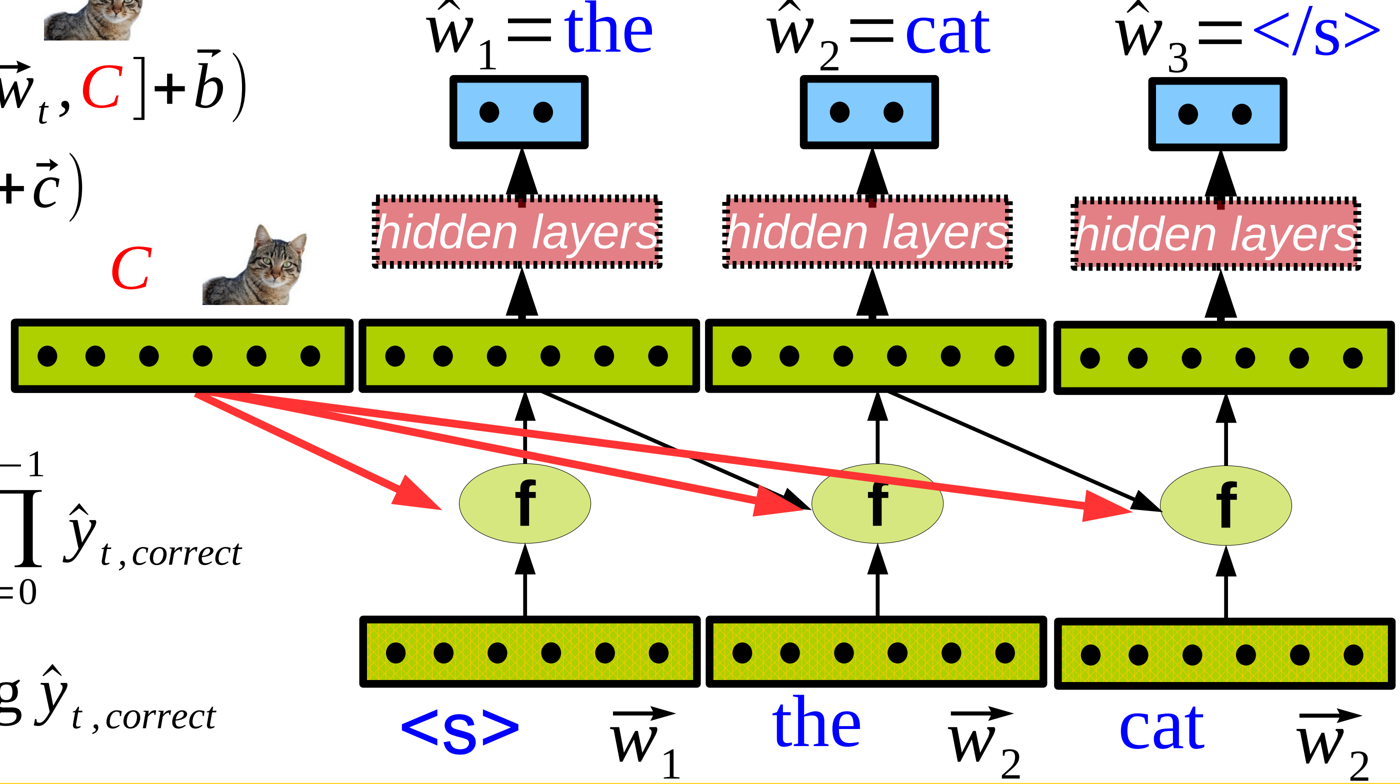
- Train (assuming *C* is given) 

Language generation:
Conditional recurrent language model

- Train (assuming *C* is given) 

$$\vec{h}_t = \tanh(W[\vec{h}_{t-1}, \vec{w}_t, C] + \vec{b})$$

$$\hat{y}_t = softmax(W^S \vec{h}_t + \vec{c})$$



$$p(w_{1,}\dots,w_m|C) \approx \prod_{t=0}^{m-1} \hat{y}_{t,correct}$$

$$J_{sentence} = -\frac{1}{T}\sum_{t=1}^{m} \log \hat{y}_{t,correct}$$

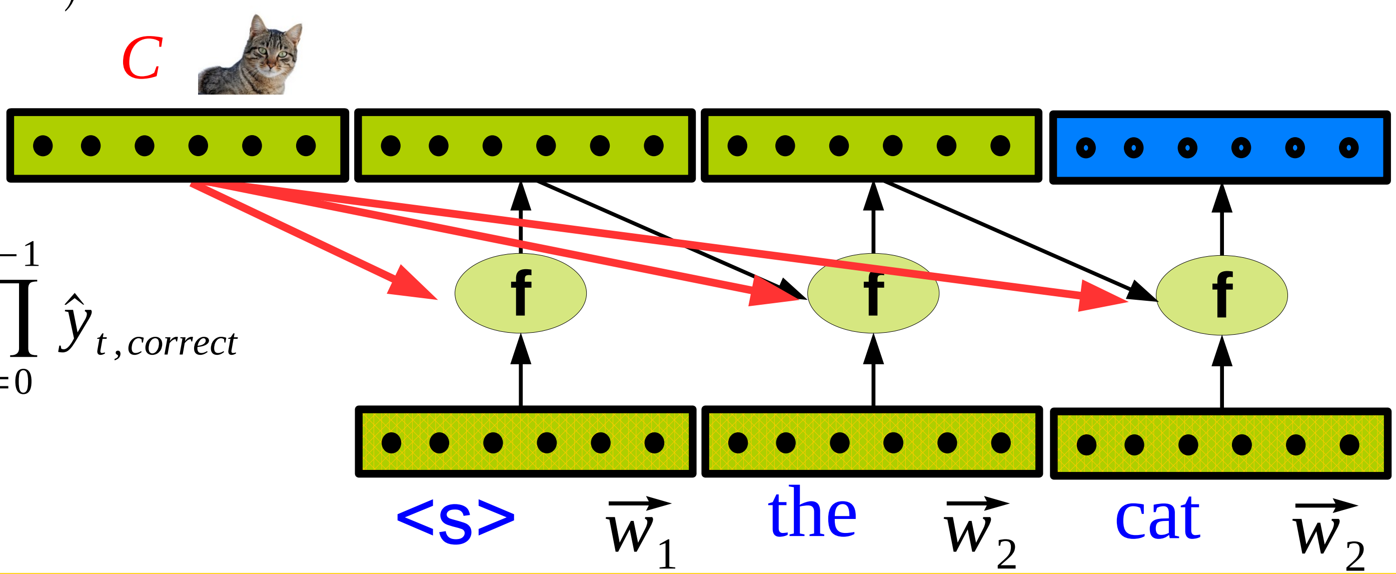
Language generation:
Conditional recurrent language model

- Use as encoder or decoder

$$\vec{h}_t = \tanh\left(W[\vec{h}_{t-1}, \vec{w}_t, C] + \vec{b}\right)$$

$$\hat{y}_t = softmax\left(W^S \vec{h}_t + \vec{c}\right)$$

$$p(w_{1,} \ldots, w_m | C) \approx \prod_{t=0}^{m-1} \hat{y}_{t,correct}$$



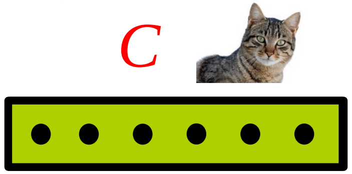
Language generation:
Conditional recurrent language model

- Use as encoder or decoder

$$\vec{h}_t = \tanh\left(W[\vec{h}_{t-1}, \vec{w}_t, C] + \vec{b}\right)$$

$$\hat{y}_t = softmax\left(W^S \vec{h}_t + \vec{c}\right)$$

C



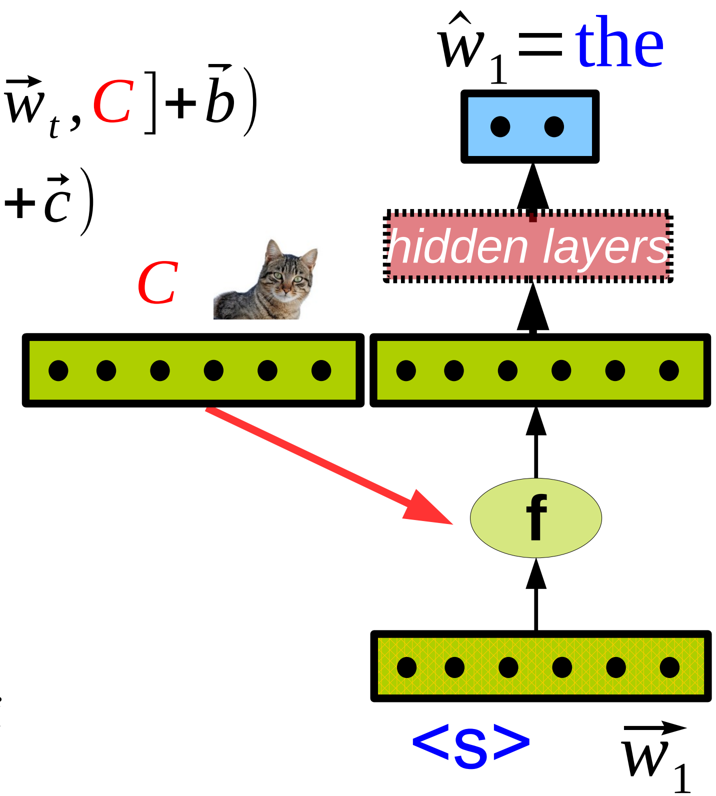
$$\hat{w}_t = \underset{i}{argmax}\, \hat{y}_{t,i}$$

Language generation:
Conditional recurrent language model

- Use as encoder or decoder

$$\vec{h}_t = \tanh(W[\vec{h}_{t-1}, \vec{w}_t, C] + \vec{b})$$

$$\hat{y}_t = softmax(W^S \vec{h}_t + \vec{c})$$



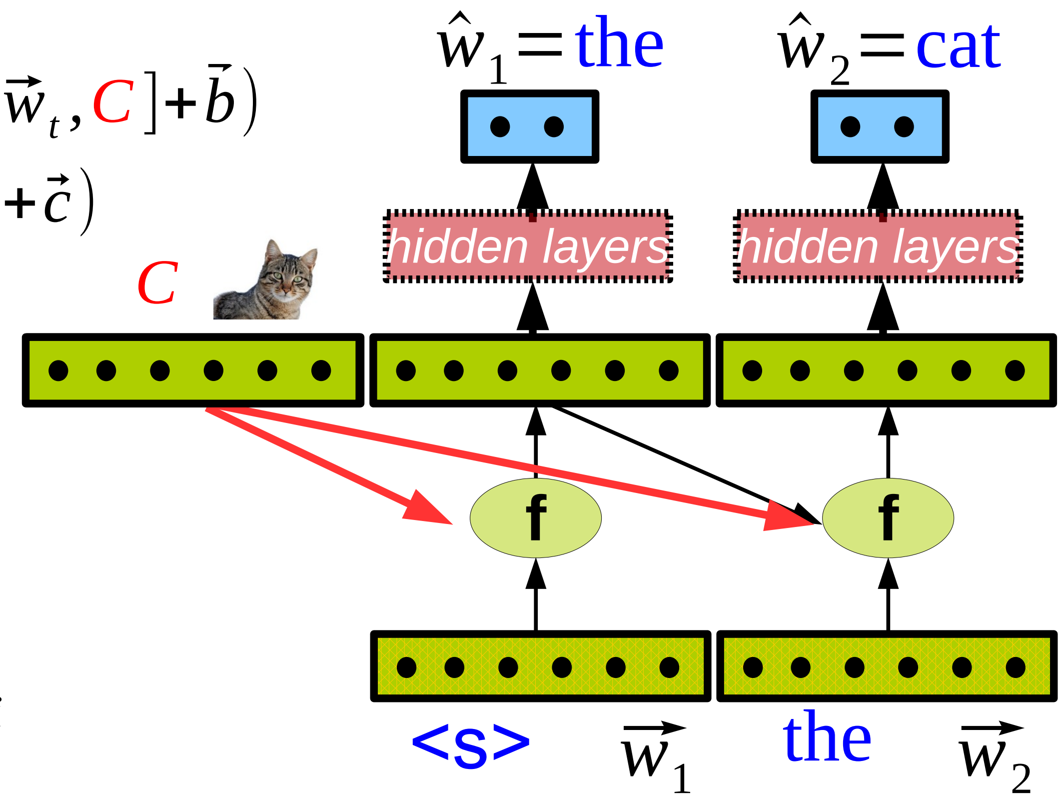
$$\hat{w}_t = \underset{i}{argmax}\, \hat{y}_{t,i}$$

Language generation: Conditional recurrent language model

- Use as encoder or decoder

$$\vec{h}_t = \tanh(W[\vec{h}_{t-1}, \vec{w}_t, C] + \vec{b})$$

$$\hat{y}_t = softmax(W^S \vec{h}_t + \vec{c})$$



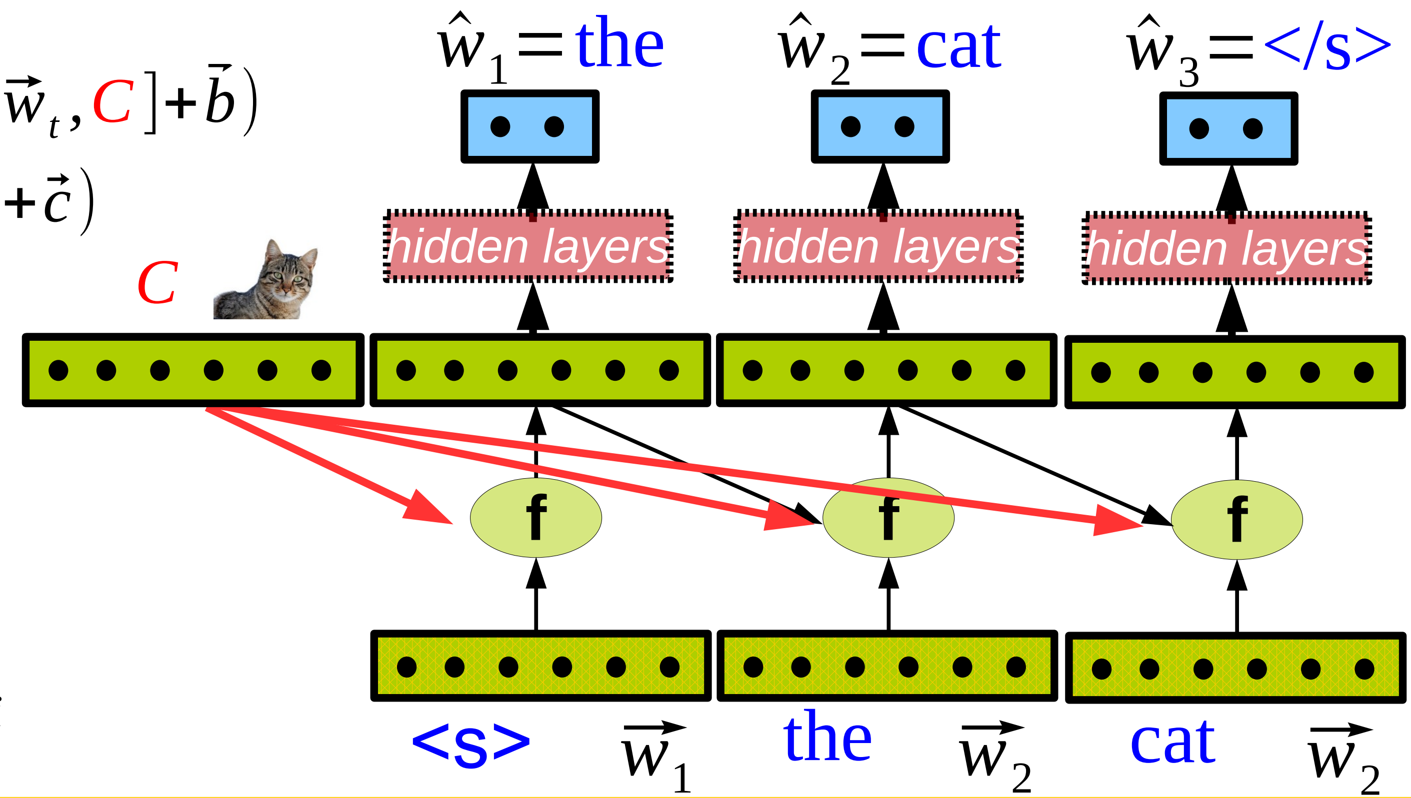
$$\hat{w}_t = \underset{i}{argmax}\, \hat{y}_{t,i}$$

Language generation:
Conditional recurrent language model

- Use as encoder or decoder

$$\vec{h}_t = \tanh(W[\vec{h}_{t-1}, \vec{w}_t, C] + \vec{b})$$

$$\hat{y}_t = softmax(W^S \vec{h}_t + \vec{c})$$



$$\hat{w}_t = \underset{i}{argmax}\, \hat{y}_{t,i}$$

Plan for this session

- Application of RNN:
  - Language Models (sentence encoders)
  - Language Generation (sentence decoders)
  - **Sequence to sequence models & Neural Machine Translation (I)**
- Problems with gradients in RNN LSTM and GRU

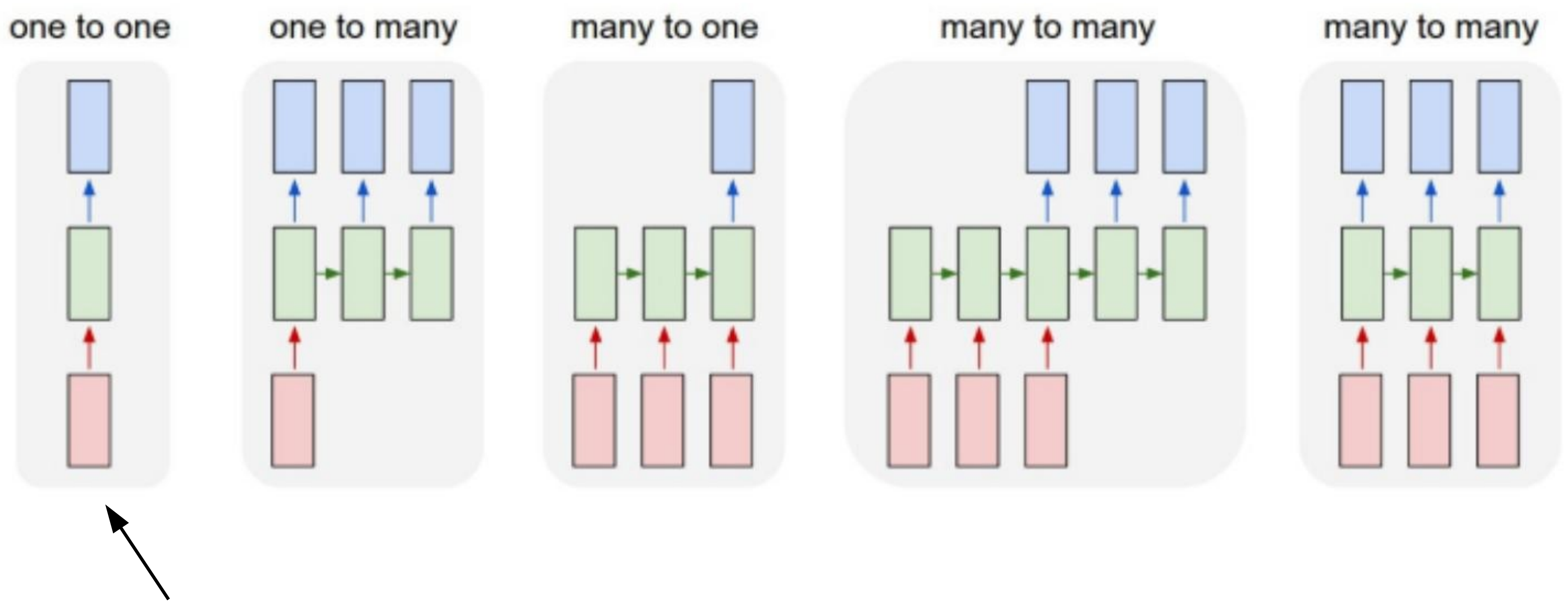
Sequence to sequence models

- For any problem that can be interpreted as a transformation from one sequence to another:
  - Model it as a **pair of RNNs:**
  - An **encoder** that reads the input sentence, outputs nothing
  - A **decoder** whose starting hidden state is the last hidden state of the encoder, and that generates a sentence (RNN language model)
  - Give it lots of data….
    Success is guaranteed (Sutskever et al. 2014)

Sequence to sequence models

- They are state-of-the-art in many problems, some of them fairly novel
  - Speech → text (and viceversa)
  - Foreign sentences → translation
  - Emails → simple replies (Gmail)
  - Python functions → result
  - English sentences → parsing instructions
  - Question → answer (Chatbots, Google Duplex)
  - Image → textual description
  - Video → textual descriptions
  - ...

Sequence to sequence models



MLP

Source: Fei-Fei Li & Andrej Karpathy and Justin Johnson

Sequence to sequence models

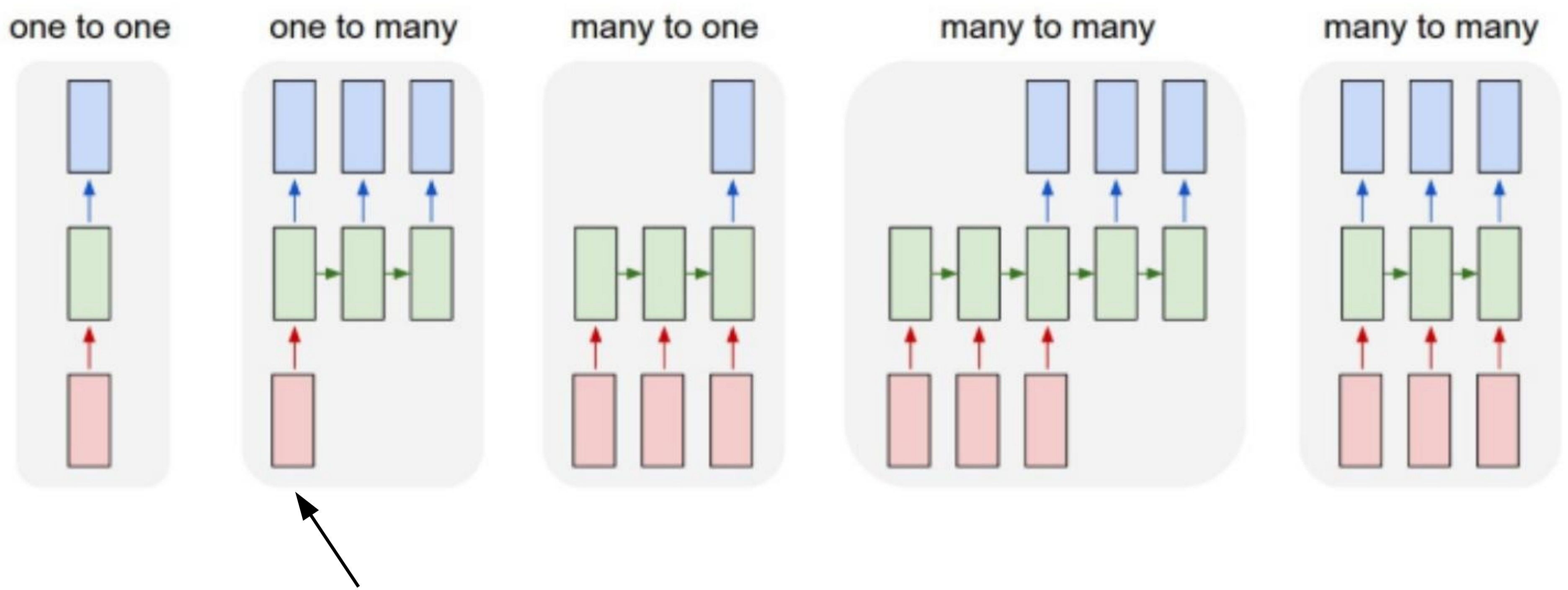
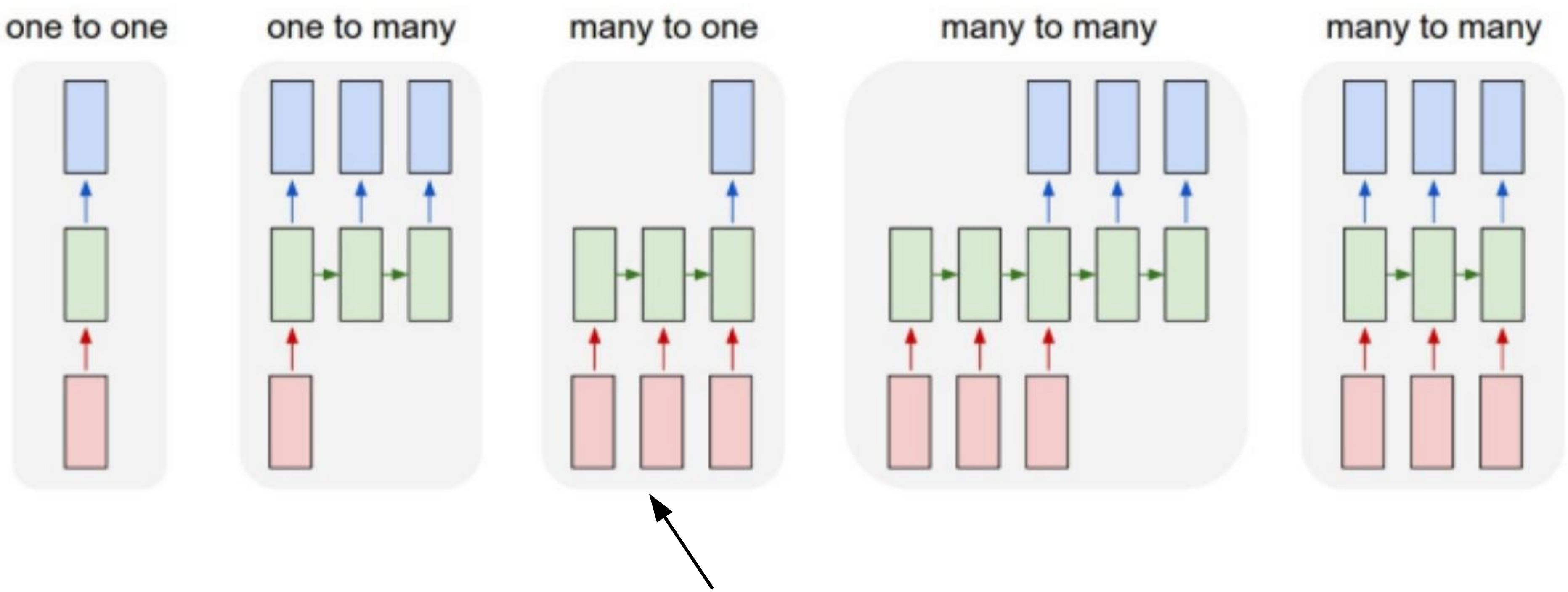


Image Captioning

Source: Fei-Fei Li & Andrej Karpathy and Justin Johnson

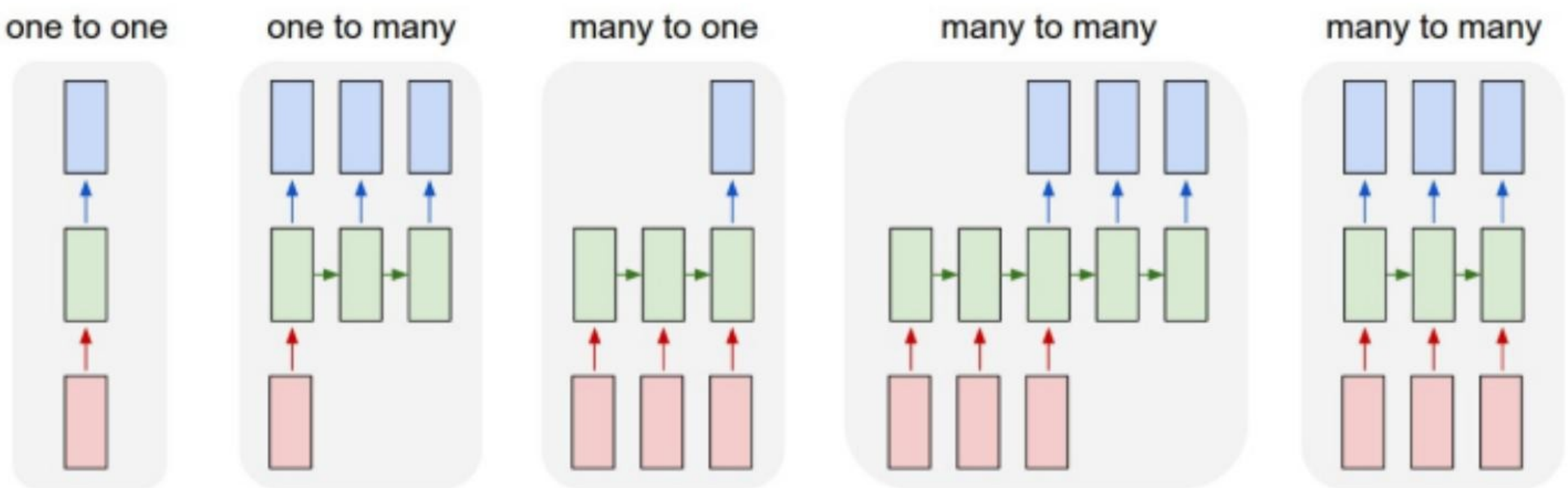
Sequence to sequence models



Sentiment classification

Source: Fei-Fei Li & Andrej Karpathy and Justin Johnson

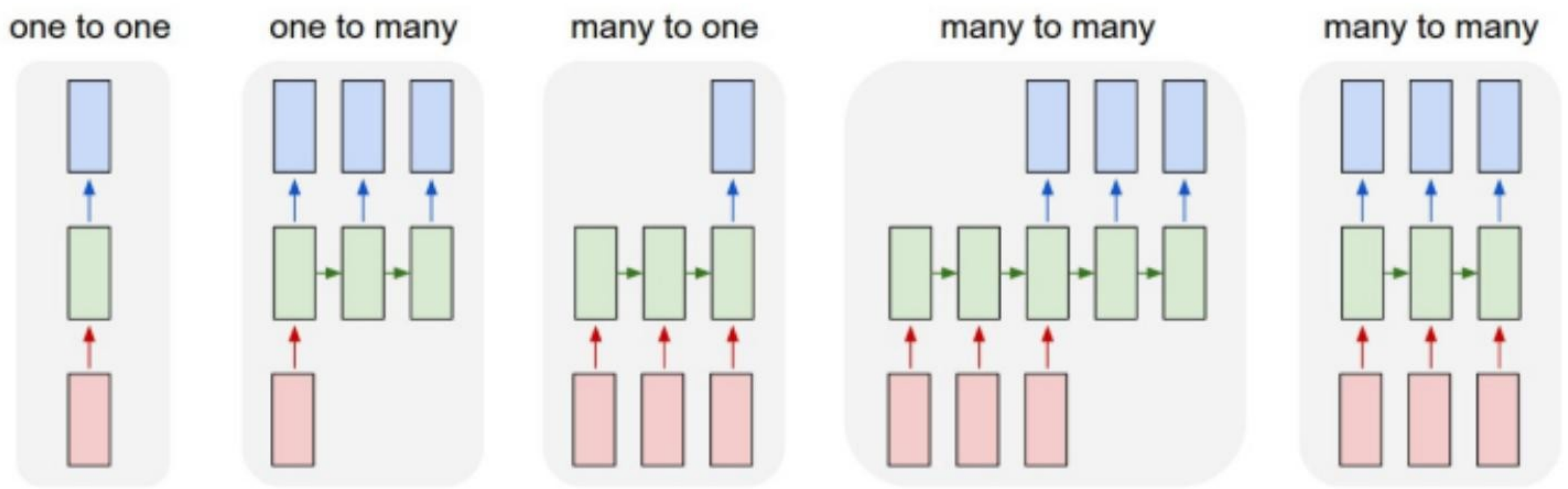
Sequence to sequence models



Machine translation

Source: Fei-Fei Li & Andrej Karpathy and Justin Johnson

Sequence to sequence models



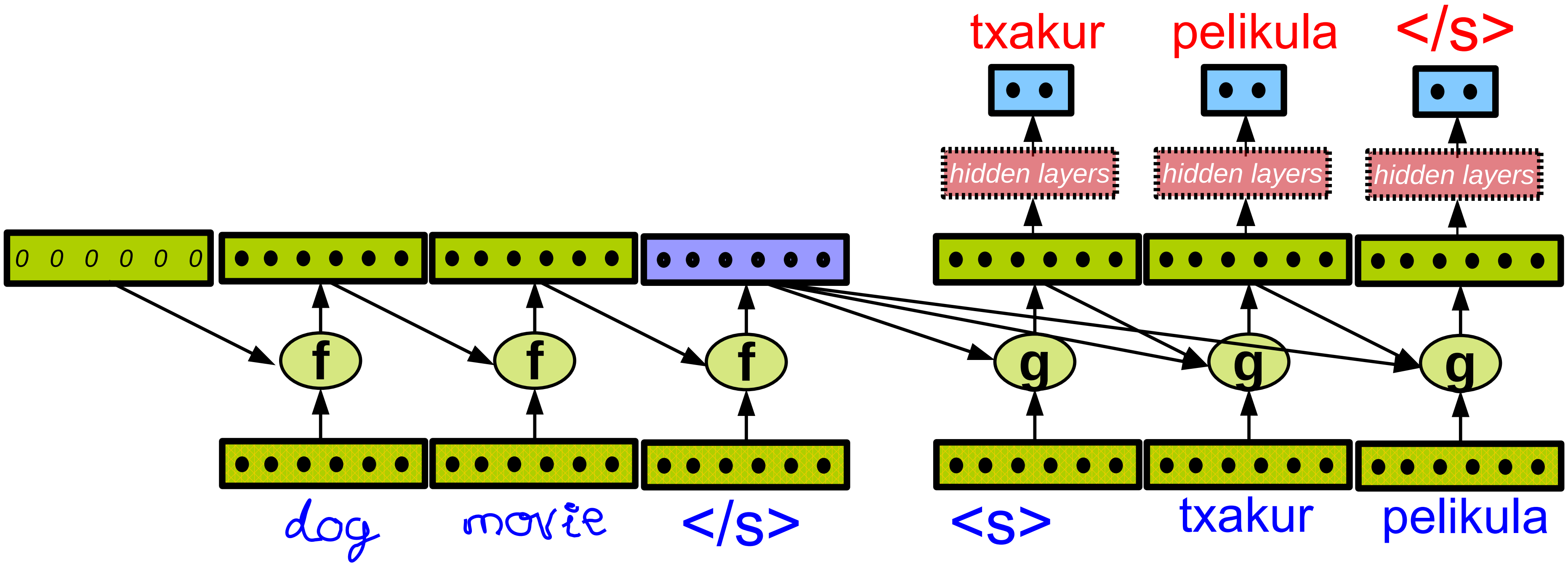
Video classification at frame level

Source: Fei-Fei Li & Andrej Karpathy and Justin Johnson

Sequence to sequence for MT

Combine two RLM: encoder and decoder

Train as regular RLM



Sequence to sequence for MT

Combine two RLM

Train as in regular RLM

- Note that decoder is conditioned on last hidden state from encoder:

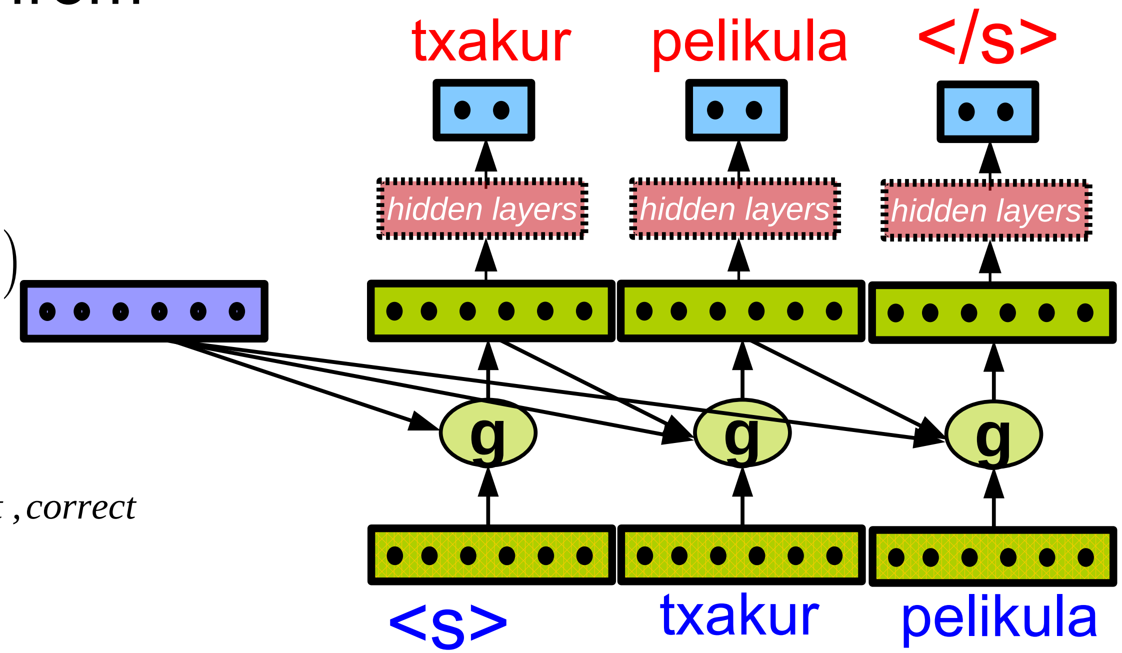
$$C = \vec{\boldsymbol{h}}_{\boldsymbol{encoder}}$$

$$\vec{h}_t = \tanh\left(W\left[\vec{h}_{t-1}, \vec{w}_t, C\right] + \vec{b}\right)$$

$$\hat{y}_t = softmax\left(W^S \vec{h}_t + \vec{c}\right)$$

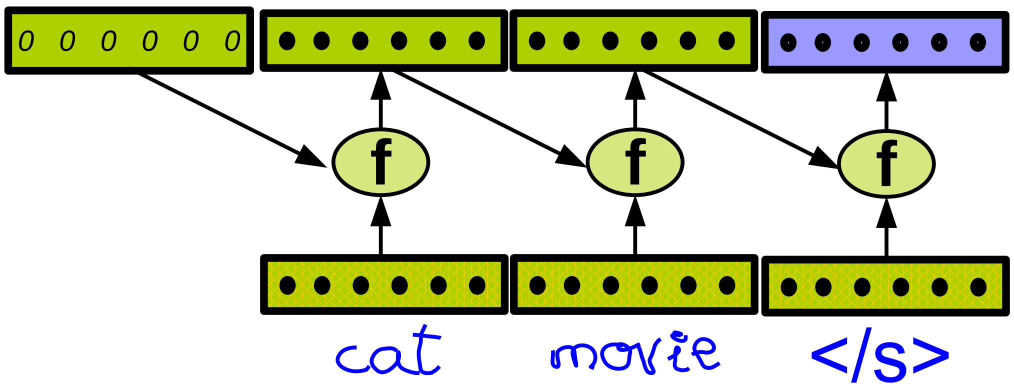
$$p\left(w_{1,} \ldots, w_m \middle| \boldsymbol{h}_{\mathbf{encoder}}\right) \approx \prod_{t=0}^{m-1} \hat{y}_{t,correct}$$

$$J_{\text{sentence}} = -\frac{1}{T} \sum_{t=1}^{m} \log \hat{y}_{t,correct}$$



Sequence to sequence for MT

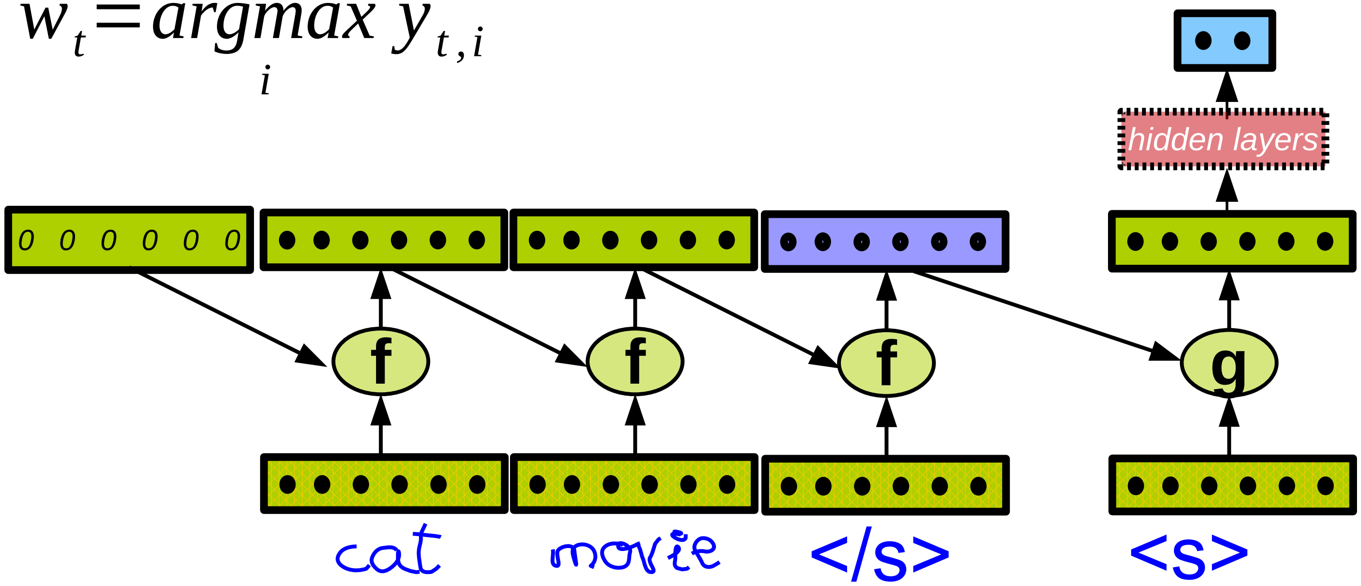
- Test as conditional RLM decoder
  - Compute sentence representation for input sentence



Sequence to sequence for MT

- Test as conditional RLM decoder
  - Compute sentence representation
  - Generate first word (after <s>)

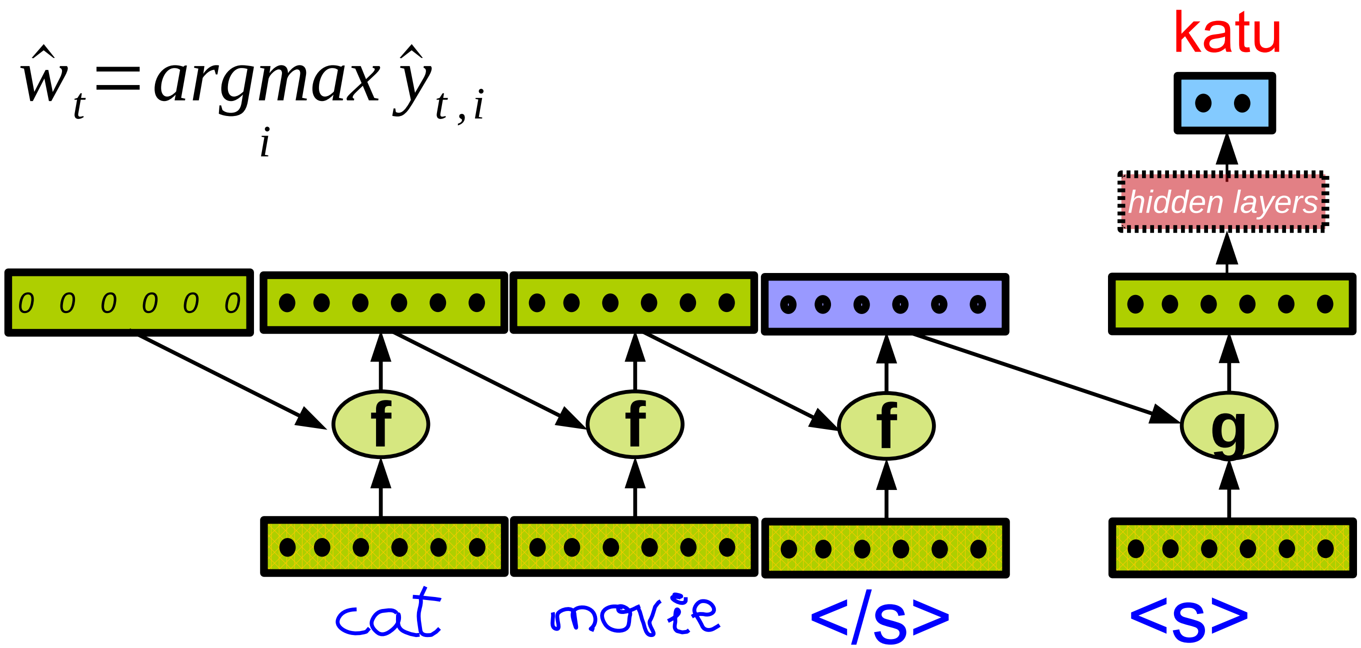
$$\hat{w}_t = \underset{i}{argmax}\, \hat{y}_{t,i}$$



Sequence to sequence for MT

- Test as conditional RLM decoder
  - Compute sentence representation
  - Generate first word (after <s>)

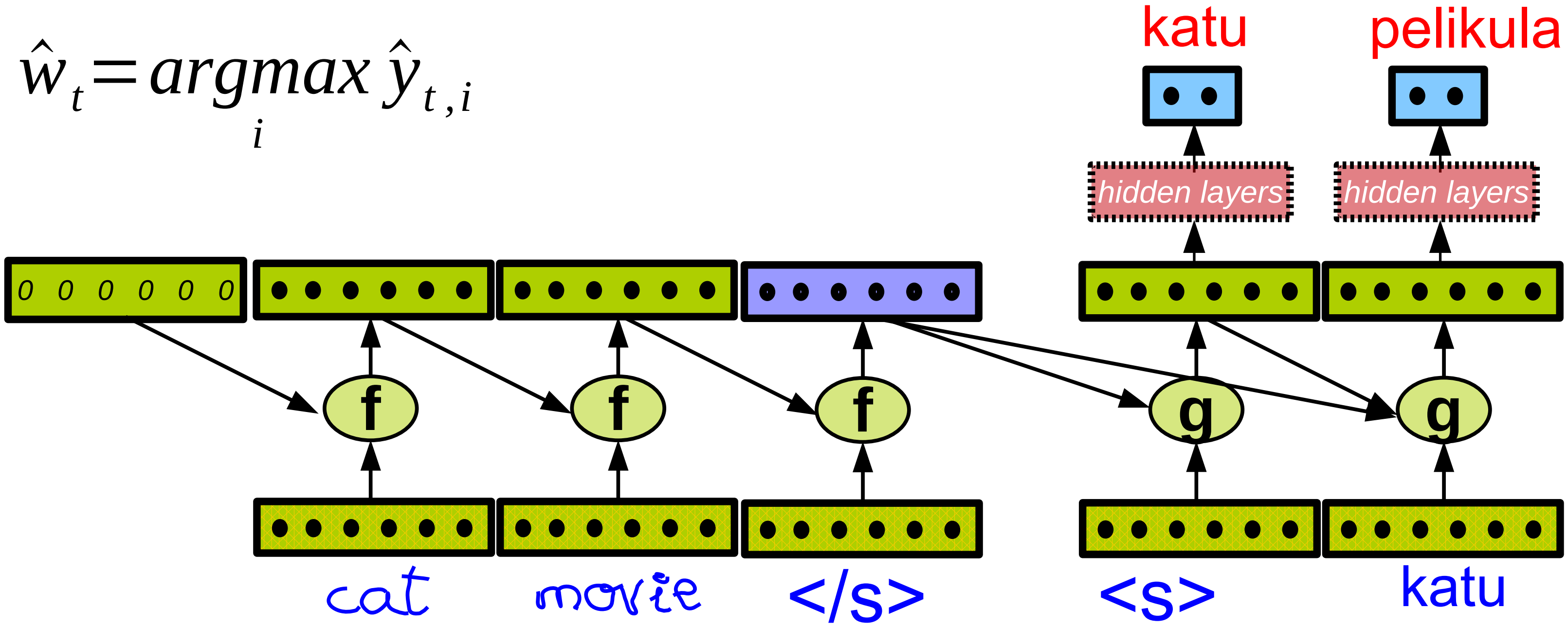
$$\hat{w}_t = \underset{i}{argmax}\, \hat{y}_{t,i}$$



Sequence to sequence for MT

- Test as conditional RLM decoder
  - Compute sentence representation
  - Generate second word

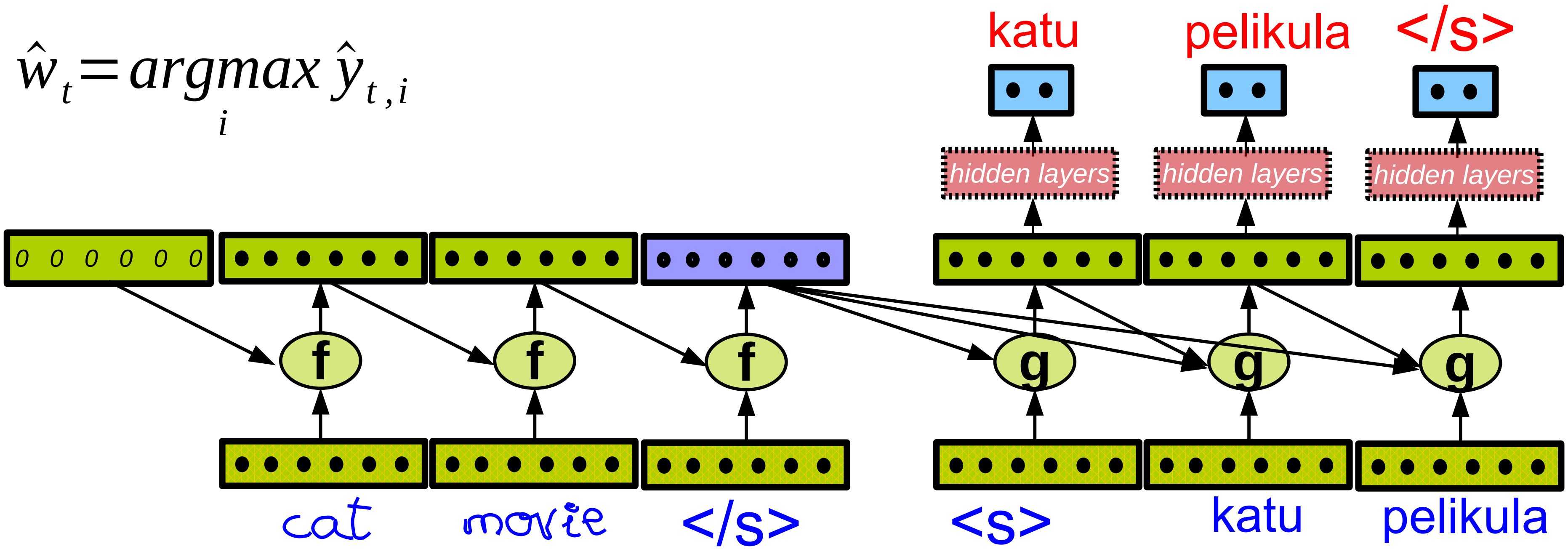
$$\hat{w}_t = \underset{i}{argmax}\, \hat{y}_{t,i}$$



Sequence to sequence for MT

- Test as conditional RLM decoder
  - Compute sentence representation
  - Generate next word, stop if </s>

$$\hat{w}_t = \underset{i}{argmax}\, \hat{y}_{t,i}$$



Sequence to sequence for MT

- Taking the maximum at each decoding step is called "**greedy decoding**"
  - It's not optimal: it depends a lot on the first word, and each decision is done independently (no turning back!)
- **Brute force**: Generate all possible sequences in target language and use conditional LM of the decoder to select maximum $P(x_1, \dots x_m \mid h)$
  - Optimal but NP-complete
- **Beam search**: At each decoding step keep the sequences with top probability so far
  - Best in practice, even with small beams (e.g. 10)

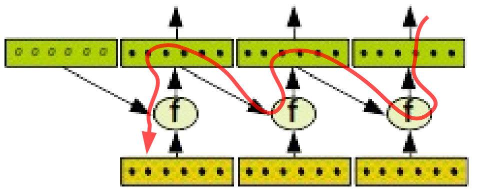
Sequence to sequence for MT

- These models do not beat statistical MT
- There is a huge loss of information in trying to cram all the meaning of the input sentence in a single vector
  - The decoder is missing key information like individual words in the input, word order information, length of input, etc.
  - We will see how to build a state-of-the-art neural MT as used by Facebook, Google, and elsewhere

Plan for this session

- Application of RNN:
  - Language Models (sentence encoders)
  - Language Generation (sentence decoders)
  - Sequence to sequence models &
    Neural Machine Translation (I)
- **Problems with gradients in RNN LSTM and GRU**

Training RNNs is hard

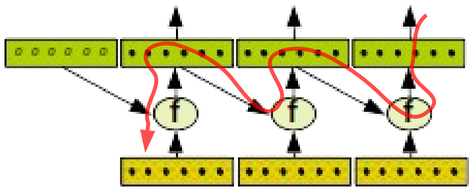


$$\vec{h}_t = \tanh(W[\vec{h}_{t-1}, \vec{w}_t] + \vec{b})$$

(*Basic*) RNNs are unstable, need to carefully initialize parameters

- RNNs are deep:
  layers between first token and last output
  - Actually deep MLPs have similar issues
- Forward: multiply same matrix each time
- Backward: the gradients for early tokens are extremely low

Training RNNs is hard



$\vec{h}_t = \tanh(W[\vec{h}_{t-1}, \vec{w}_t] + \vec{b})$

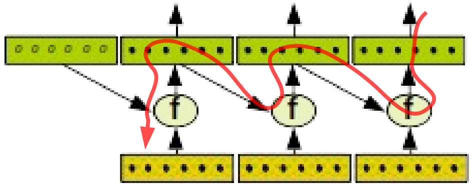
Vanishing / exploding gradients:

- When trained on long sequences (100 tokens) RNN gradients can vanish or explode
  - Careful initialization of weights
- Even with good initial weights, it is very hard for early tokens to influence later tokens (**long distance dependencies**, common in language)
  - RNN Lms have a difficult time with sentences like

I grew up in France, (…) so I speak fluent _____

Jane was waiting for John, so when he arrived Jane said hi (…) to ______

Training RNNs is hard



$$\vec{h}_t = \tanh(W[\vec{h}_{t-1}, \vec{w}_t] + \vec{b})$$

Solutions to vanishing / exploding gradients:

- Avoid multiplying history many times with W:
  - MLP: Residual networks and highway networks
  - RNN: use LSTM and GRU cells
- MLP: Unbounded non-linearities (e.g. Relu)
- Gradient clipping:

```
if ||gradient|| > threshold then
multiply gradient by threshold/||gradient||
```

Combinations!

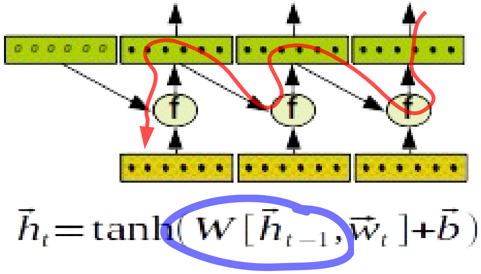
→ Easier to train RNNs

→ *Alleviate* the long-distance dependency problem.

Long Short-Term Memory LSTM

To remember tokens in the long past...

- … avoid multiplying history every step with parameter matrix



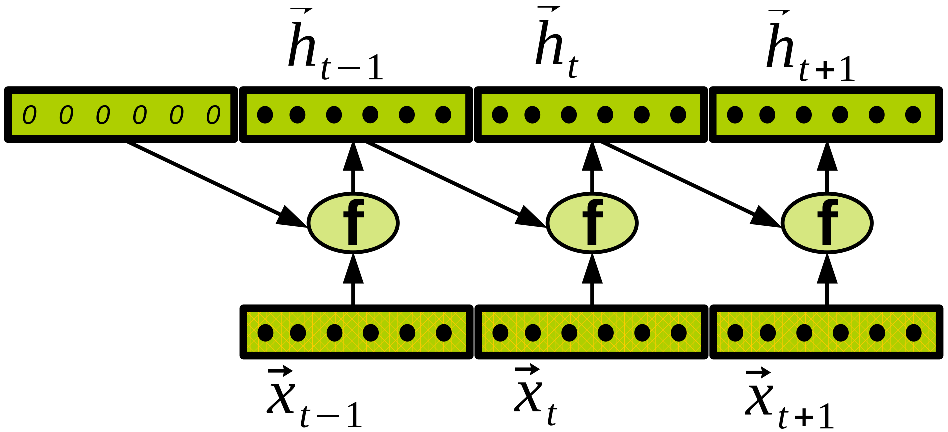
LSTM is a RNN composed of a **memory cell with gates**

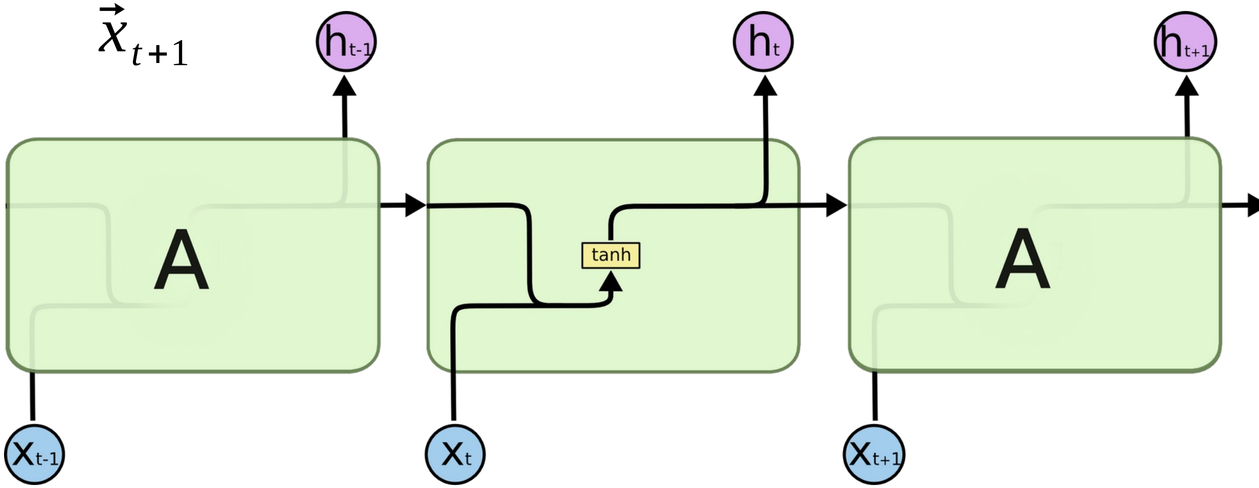
- Information gets into the cell whenever its “write” gate is on
- The information stays in the cell as long as its “keep” gate is on
- Information can be read from the cell by turning on its “read” gate

(Hochreiter and Schmidhuber, 1997)

Long Short-Term Memory LSTM

We introduce a new visualization for RNNs:

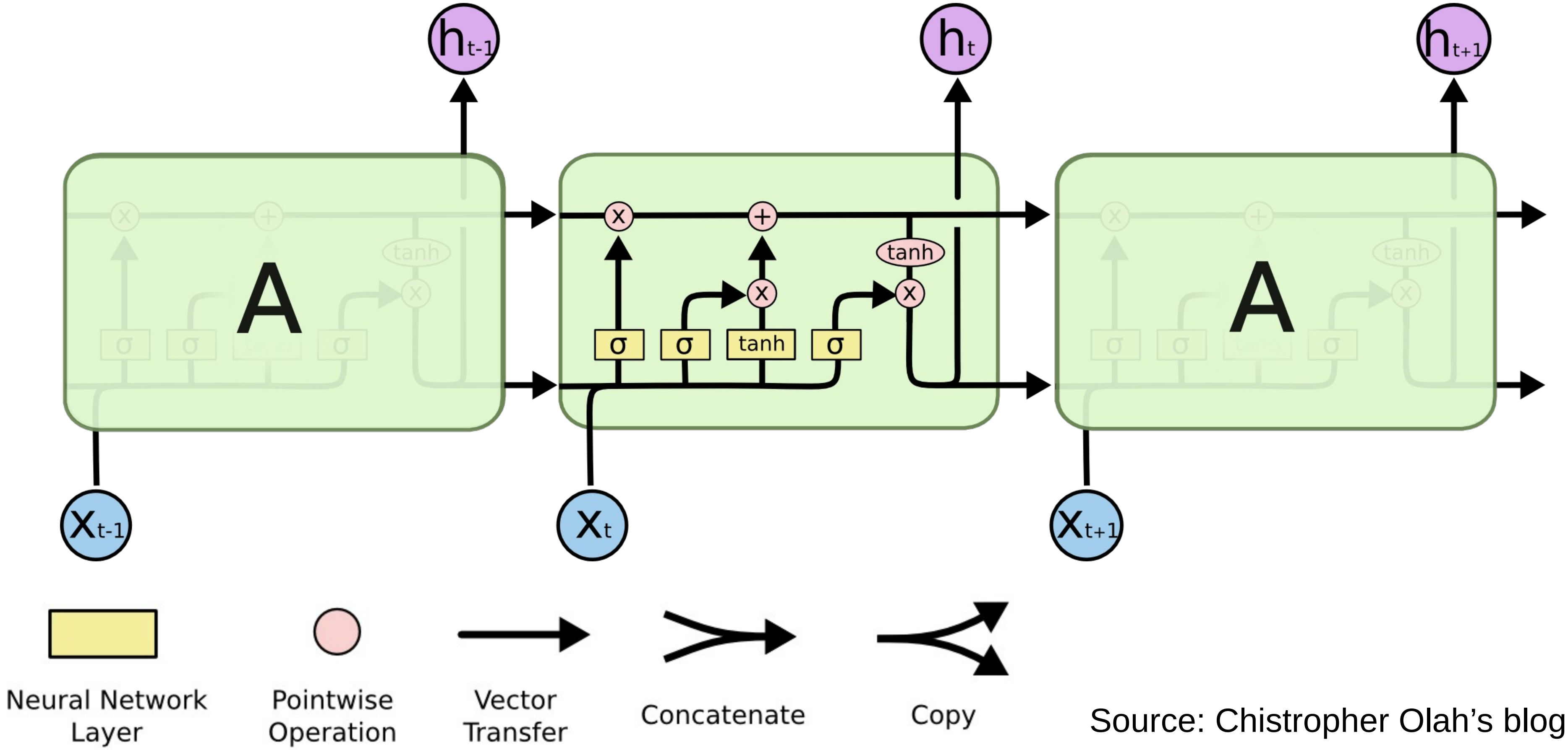




Source: Chistropher Olah's blog

Long Short-Term Memory LSTM

LSTM cells have a rich internal structure

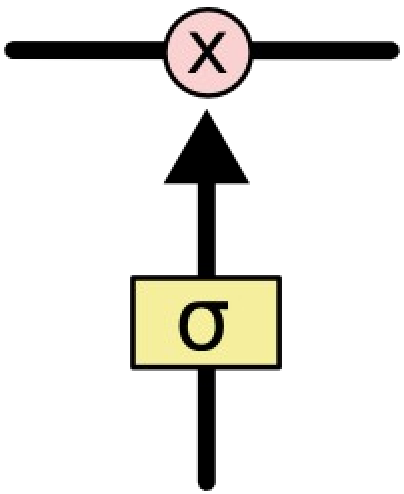


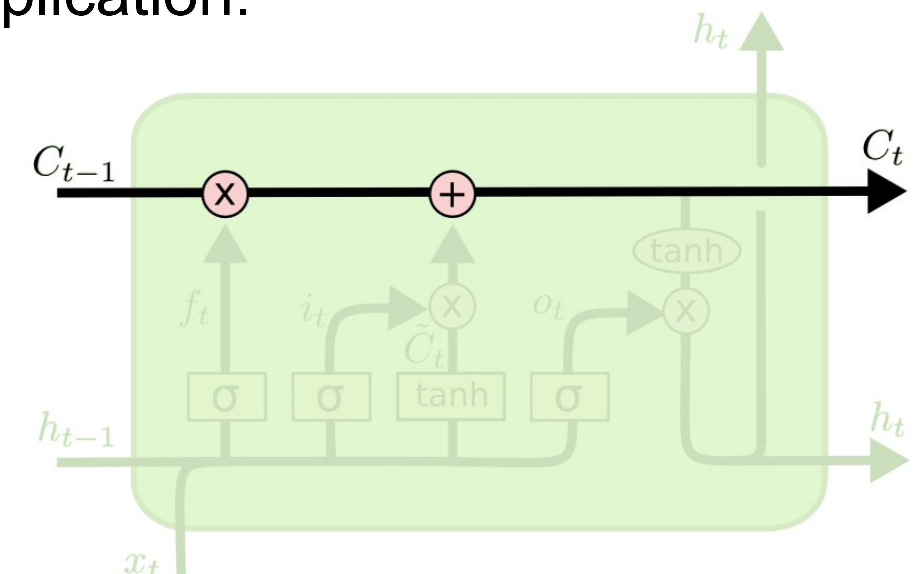
Source: Chistropher Olah’s blog

Long Short-Term Memory LSTM

There are three gates:

- Optionally let information through.
- Sigmoid layer, returning [0,1] per dimension, then pointwise multiplication.



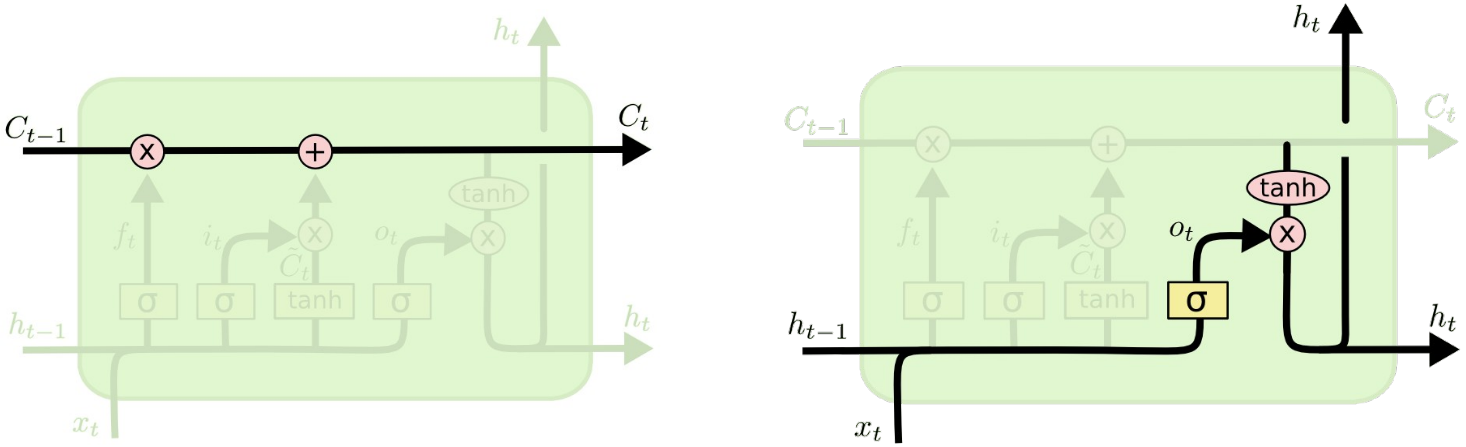


Source: Chistropher Olah's blog

Long Short-Term Memory LSTM

We keep two "histories":

- C_t cell state: flows from left to right, minor linear interactions, largely unchanged.
  Remove/add information via **forget** and **input** gates.
- h_t output state: filtered version of C_t (**output** gate).

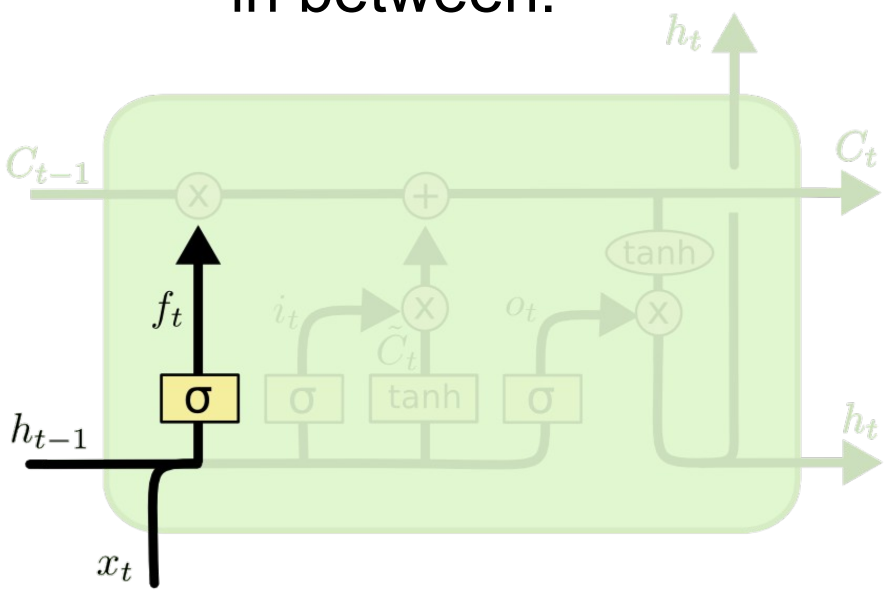


Source: Chistropher Olah's blog

Long Short-Term Memory LSTM

1st step: forget

- Forget gate $\boldsymbol{f_t}$ decides which information (dimension) to through away from $\boldsymbol{C_{t-1}}$: 1 to keep, 0 to discard and in between.



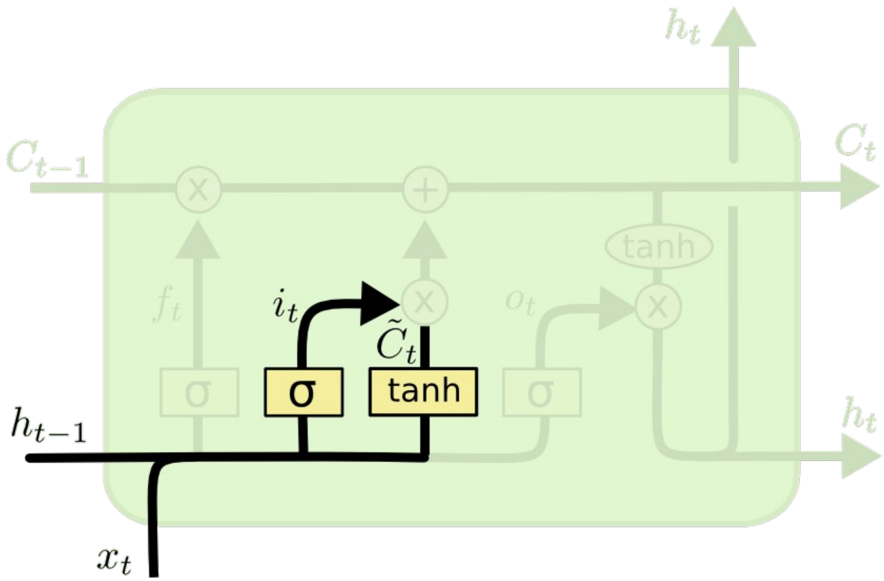
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Source: Chistropher Olah's blog

Long Short-Term Memory LSTM

2nd step: decide and prepare new information

- Input gate $\boldsymbol{i_t}$ decides which dimensions need updating information
- A tanh layer prepares candidate values $\tilde{C}_t$



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

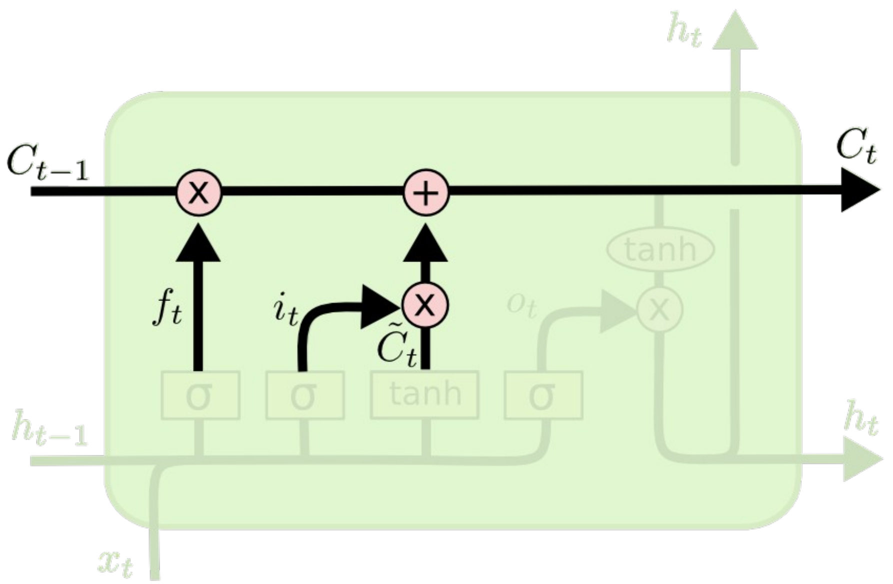
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \;+\; b_C)$$

Source: Chistropher Olah’s blog

Long Short-Term Memory LSTM

3rd step: update $\boldsymbol{C_{t-1}}$ into the new cell state $\boldsymbol{C_t}$

- Multiply by $\boldsymbol{f_t}$ and add $\boldsymbol{i_t * \tilde{C}_t}$



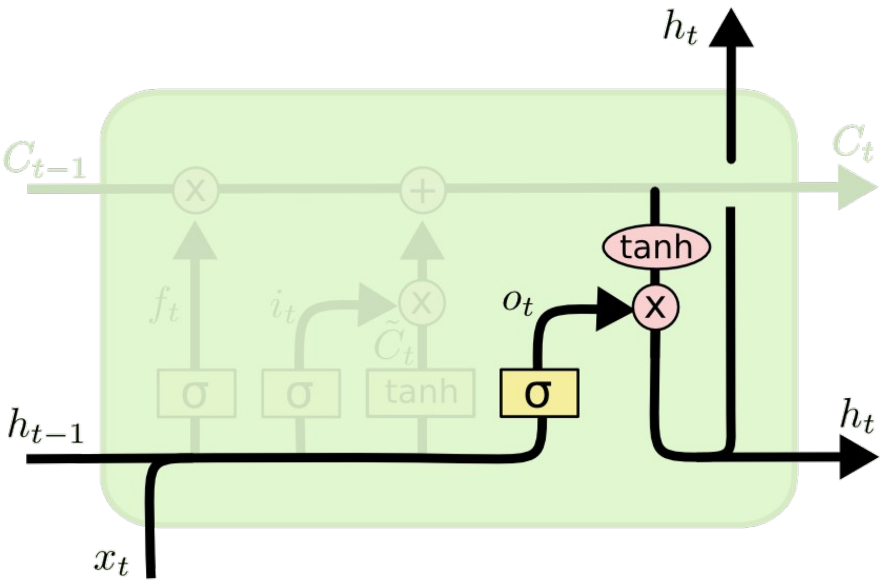
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Source: Chistropher Olah's blog

Long Short-Term Memory LSTM

4th step: decide and prepare output

- Push $\boldsymbol{C_t}$ through tanh into [-1,+1] values
- Multiply by output gate $\boldsymbol{f_t}$, which decides which parts of $\boldsymbol{C_t}$ to output



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

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

Source: Chistropher Olah's blog

Long Short-Term Memory LSTM

- Does it really work better?

Model	Parameters	Validation	Test
Mikolov & Zweig (2012) - KN-5	2M‡	–	141.2
Mikolov & Zweig (2012) - KN5 + cache	2M‡	–	125.7
Mikolov & Zweig (2012) - RNN	6M‡	–	124.7
Mikolov & Zweig (2012) - RNN-LDA	7M‡	–	113.7
Mikolov & Zweig (2012) - RNN-LDA + KN-5 + cache	9M‡	–	92.0
Zaremba et al. (2014) - LSTM (medium)	20M	86.2	82.7
Zaremba et al. (2014) - LSTM (large)	66M	82.2	78.4

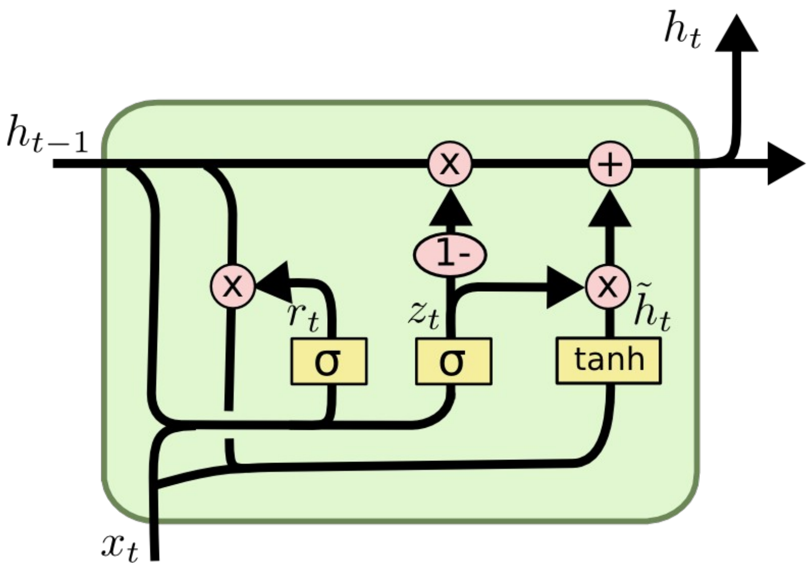
Table 1. Single model perplexity on validation and test sets for the Penn Treebank language modeling task.

Source: (Merity et al. 2018)

- Many variants and sophisticated models, but LSTMs are standing the test of time
- (Merity et al. 2018 – ICLR) for latest LM models and tricks

Variants of LSTM: GRU

- Gated Recurrent Unit (Cho et al. 2014)
- Simplification: single history, merge forget/input gates into update gate, …



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

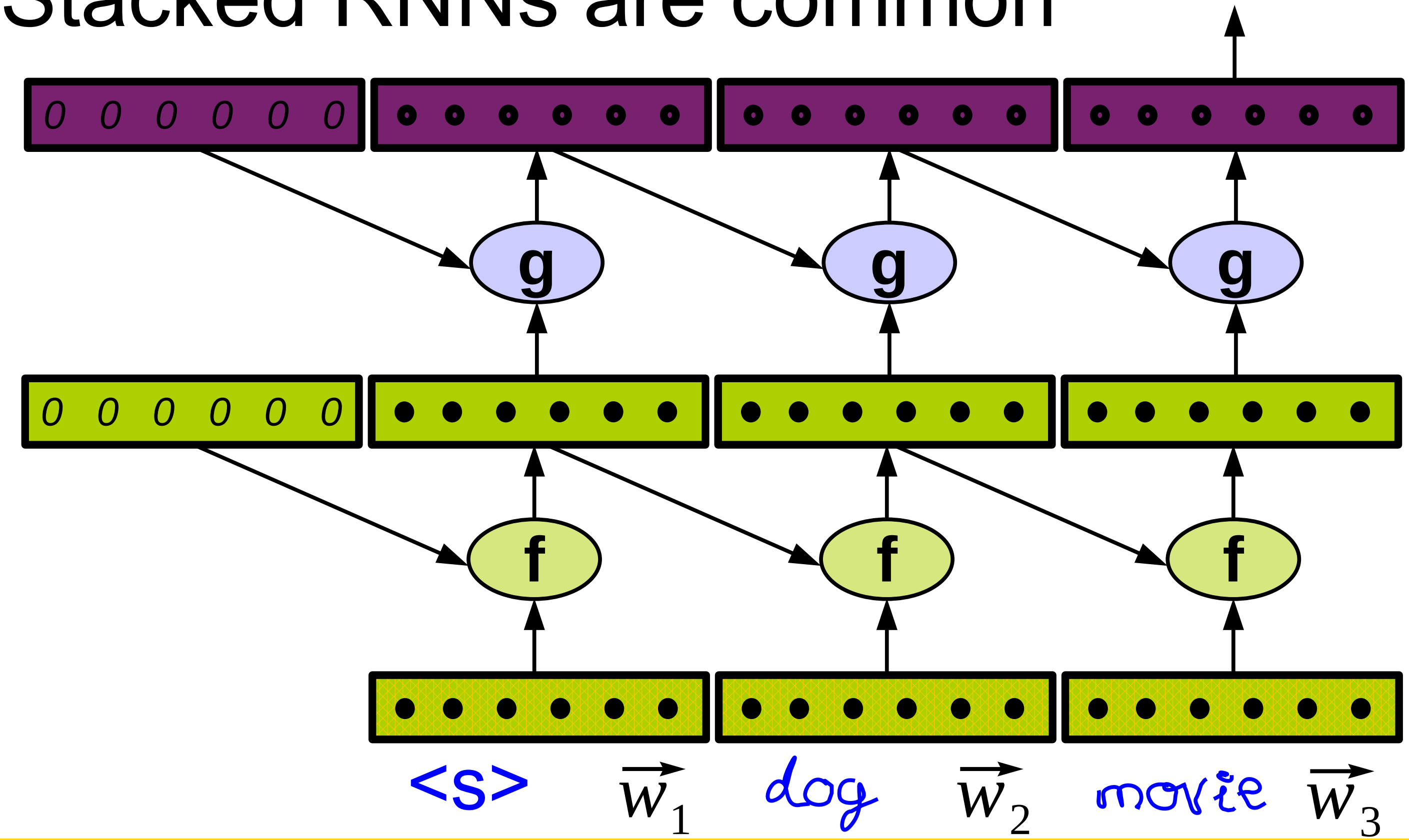
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- Advantage with respect to LSTM?

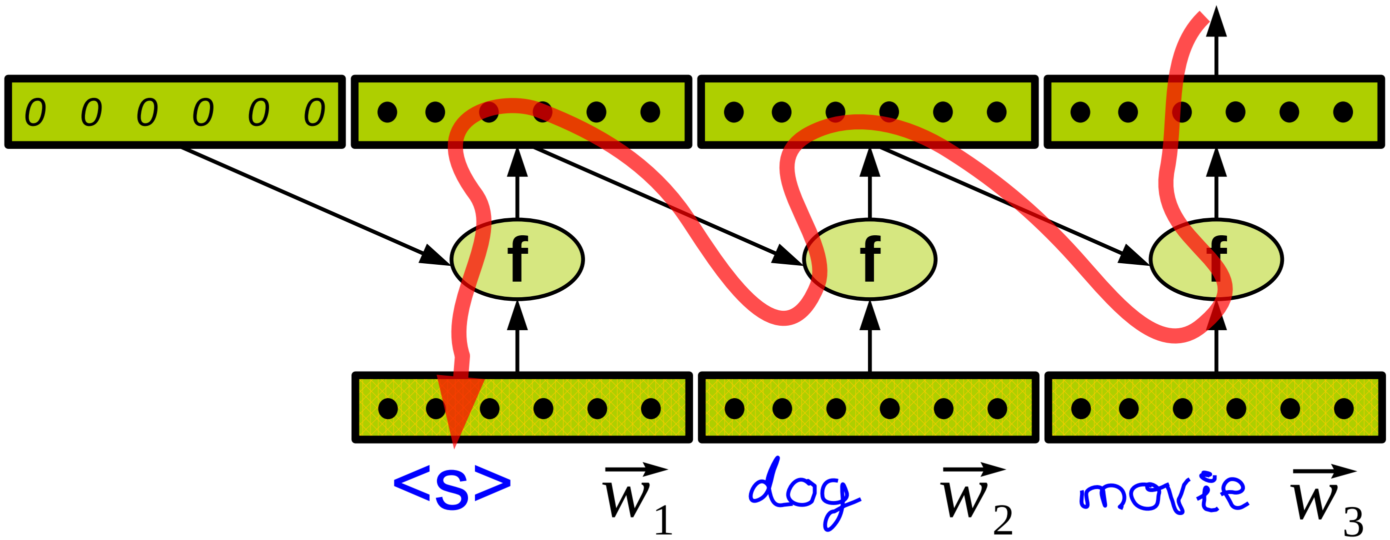
Depth and regularization

- If deep MLPs generalize better...
- Stacked RNNs are common



Depth and regularization

- Two notions of depth:
  - Number of layers between first input and last output



  - Number of layers with recurrent connections (stacked RNN layers)

Depth and regularization

Regularization depends:

- Vertical connections (feedforward):
  - Dropout
  - L2 regularization on parameters
- Horizontal connections (recurrent)
  - No dropout, although some positive results are also reported (Keras has recurrent_dropout) https://medium.com/@bingobee01/a-review-of-dropout-as-applied-to-rnns-72e79ecd5b7b

    https://medium.com/@bingobee01/a-review-of-dropout-as-applied-to-rnns-72e79ecd5b7b
  - L2 regularization on parameters

RNN cells and layers in tf

- tf.keras.layers.SimpleRNNCell
  - tf.keras.layers.LSTMCell
  - tf.keras.layers.GRUCell
  - tf.keras.layers.StackedRNNCells (as a single cell)
- tf.keras.layers.RNN
  - tf.keras.layers.LSTM
  - tf.keras.layers.GRU
- Dropout as argument

THANKS!

Acknowledgements:

- Overall slides:Sam Bowman (NYU), Chris Manning and Richard Socher (Stanford), Geofrey Hinton (Toronto, Google)
- All source url’s listed in the slides.
- Resources:
  https://cs224d.stanford.edu/notebooks/vanishing_grad_example.html
  https://cs224d.stanford.edu/notebooks/vanishing_grad_example.ipynb
  http://colah.github.io/posts/2015-08-Understanding-LSTMs/