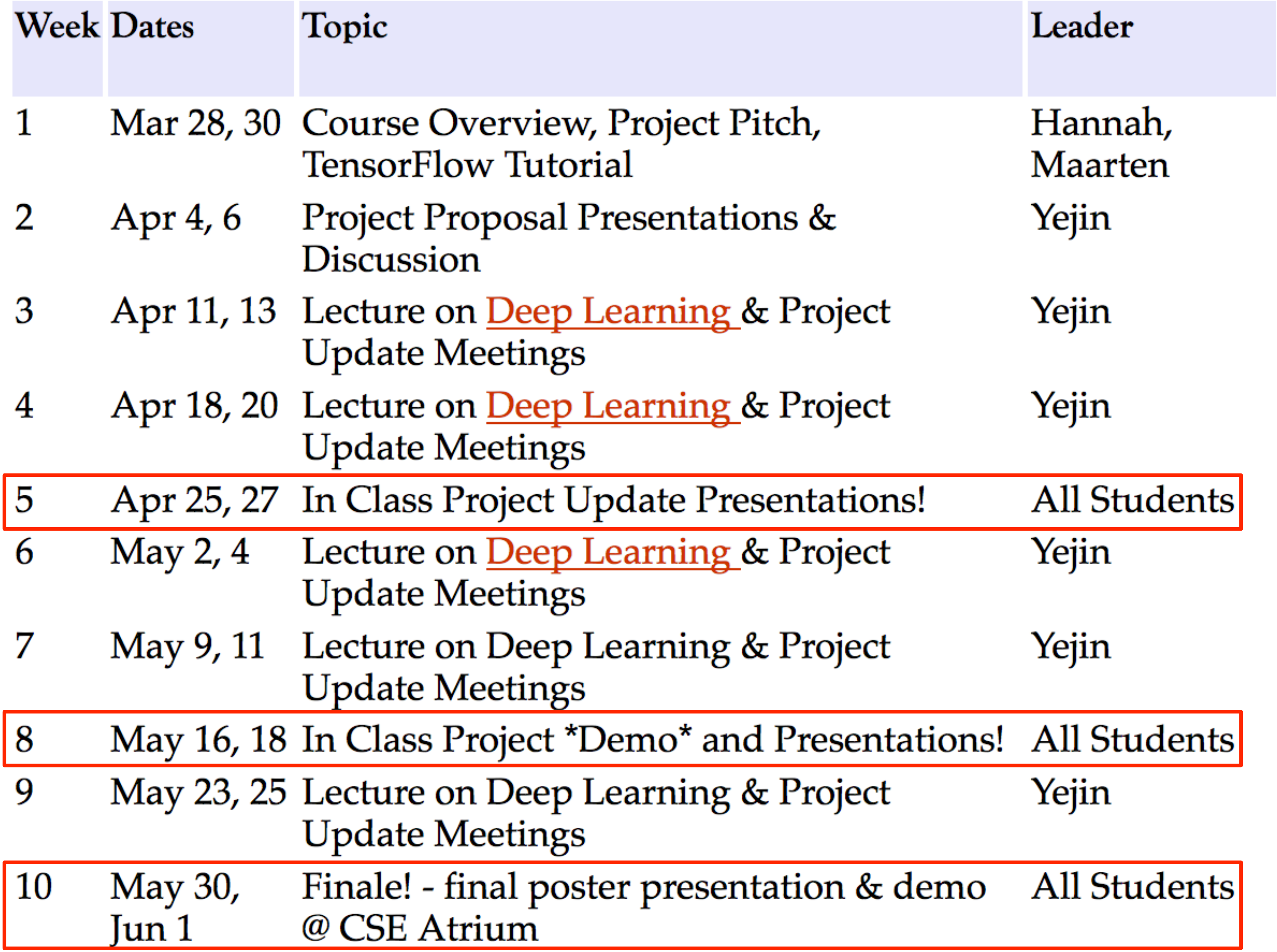
CSE 481: NLP Capstone
Spring 2017

Yejin Choi
University of Washington

Office Hour News

- Hannah:
  - Wed 2 - 3pm @ CSE 220
- Maarten:
  - Wed 2 - 3pm @ CSE 220
- Yejin:
  - Tue 2pm - 3:30pm
  - Wed 5pm - 5:30pm @ CSE 578
- All:
  - Thu 12pm – 1:25pm @ ??? for some weeks
- Google doc sign up required



Week	Dates	Topic	Leader
1	Mar 28, 30	Course Overview, Project Pitch, TensorFlow Tutorial	Hannah, Maarten
2	Apr 4, 6	Project Proposal Presentations & Discussion	Yejin
3	Apr 11, 13	Lecture on Deep Learning & Project Update Meetings	Yejin
4	Apr 18, 20	Lecture on Deep Learning & Project Update Meetings	Yejin
5	Apr 25, 27	In Class Project Update Presentations!	All Students
6	May 2, 4	Lecture on Deep Learning & Project Update Meetings	Yejin
7	May 9, 11	Lecture on Deep Learning & Project Update Meetings	Yejin
8	May 16, 18	In Class Project *Demo* and Presentations!	All Students
9	May 23, 25	Lecture on Deep Learning & Project Update Meetings	Yejin
10	May 30, Jun 1	Finale! - final poster presentation & demo @ CSE Atrium	All Students

GPU NEWS!

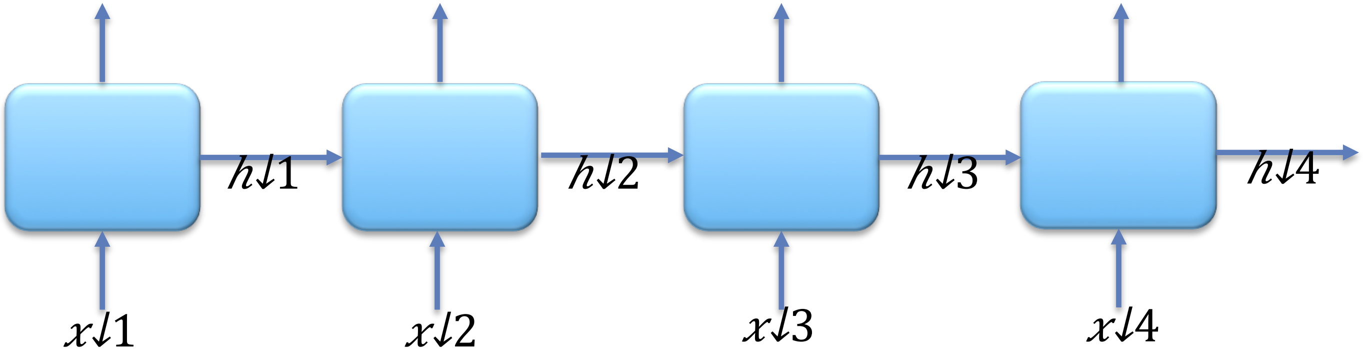
GPU NEWS!

1. Back in stock!
   - desktop with 2 GPUs can be set up at $4000
2. Microsoft Azure kindly agreed to donate free GPU cycles for the class!!!!!
3. You can sign up to Azure today for free $200 credits

RECURRENT NEURAL NETWORKS

Recurrent Neural Networks (RNNs)

- Each RNN unit computes a new hidden state using the previous state and a new input $h_t = f(x_t, h_{t-1})$
- Each RNN unit (optionally) makes an output using the current hidden state $y_t = \mathrm{softmax}(V h_t)$
- Hidden states $h_t \in R^D$ are continuous vectors
  - Can represent very rich information
  - Possibly the entire history from the beginning
- Parameters are shared (tied) across all RNN units (unlike feedforward NNs)



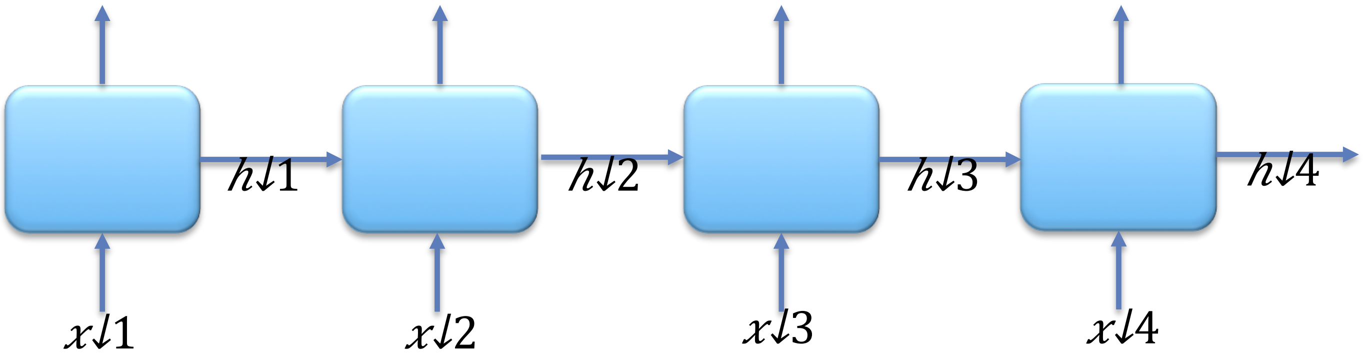
Recurrent Neural Networks (RNNs)

- Generic RNNs:

$$h_t = f(x_t, h_{t-1})$$
$$y_t = \text{softmax}(V h_t)$$

- Vanilla RNN:

$$h_t = \tanh(U x_t + W h_{t-1} + b)$$
$$y_t = \text{softmax}(V h_t)$$

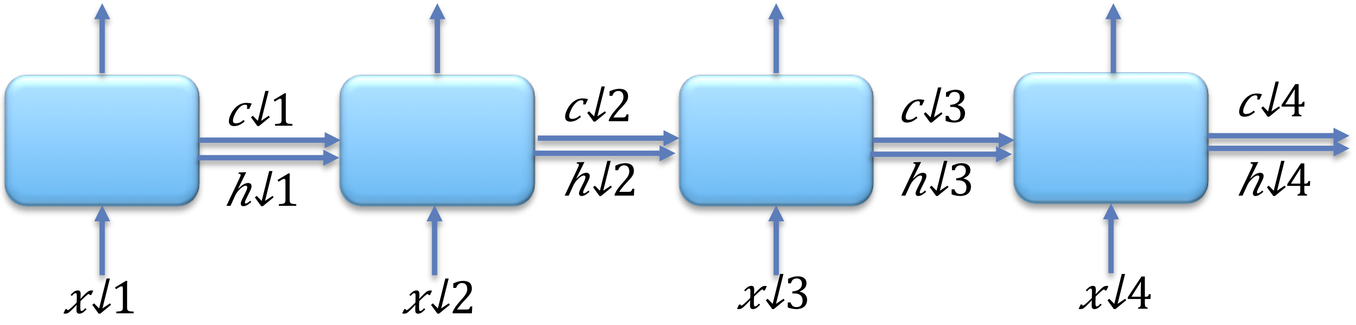


Recurrent Neural Networks (RNNs)

- Generic RNNs: $h_t = f(x_t, h_{t-1})$
- Vanilla RNNs: $h_t = \tanh(Ux_t + Wh_{t-1} + b)$
- LSTMs (Long Short-term Memory Networks):

$$
\begin{aligned}
i_t &= \sigma(U^{(i)}x_t + W^{(i)}h_{t-1} + b^{(i)}) \\
f_t &= \sigma(U^{(f)}x_t + W^{(f)}h_{t-1} + b^{(f)}) \\
o_t &= \sigma(U^{(o)}x_t + W^{(o)}h_{t-1} + b^{(o)}) \\
\tilde{c}_t &= \tanh(U^{(c)}x_t + W^{(c)}h_{t-1} + b^{(c)}) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \tanh(c_t)
\end{aligned}
$$

There are many known variations to this set of equations!



c↓t: cell state

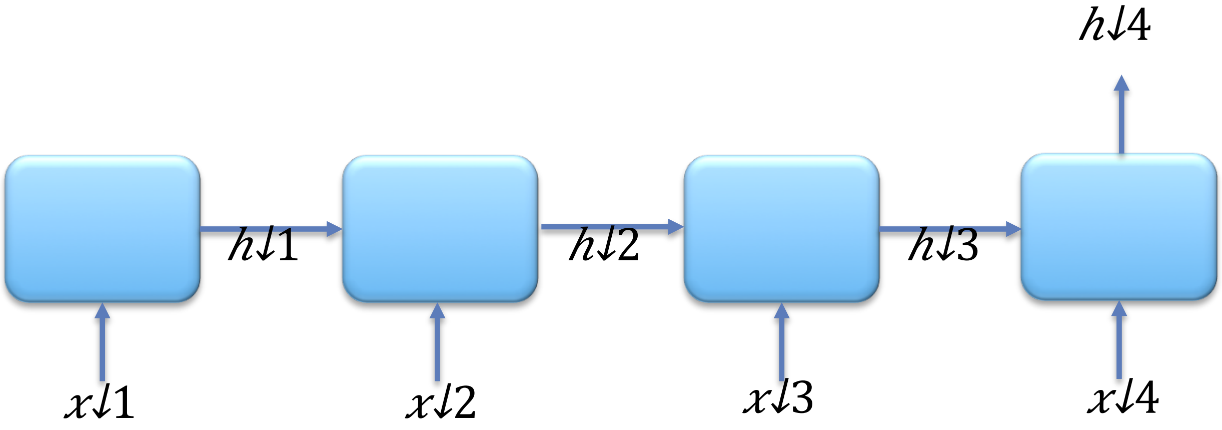
h↓t: hidden state

Many uses of RNNs
1. Classification (seq to one)

- Input: a sequence
- Output: one label (classification)
- Example: sentiment classification

$$h_t = f(x_t, h_{t-1})$$

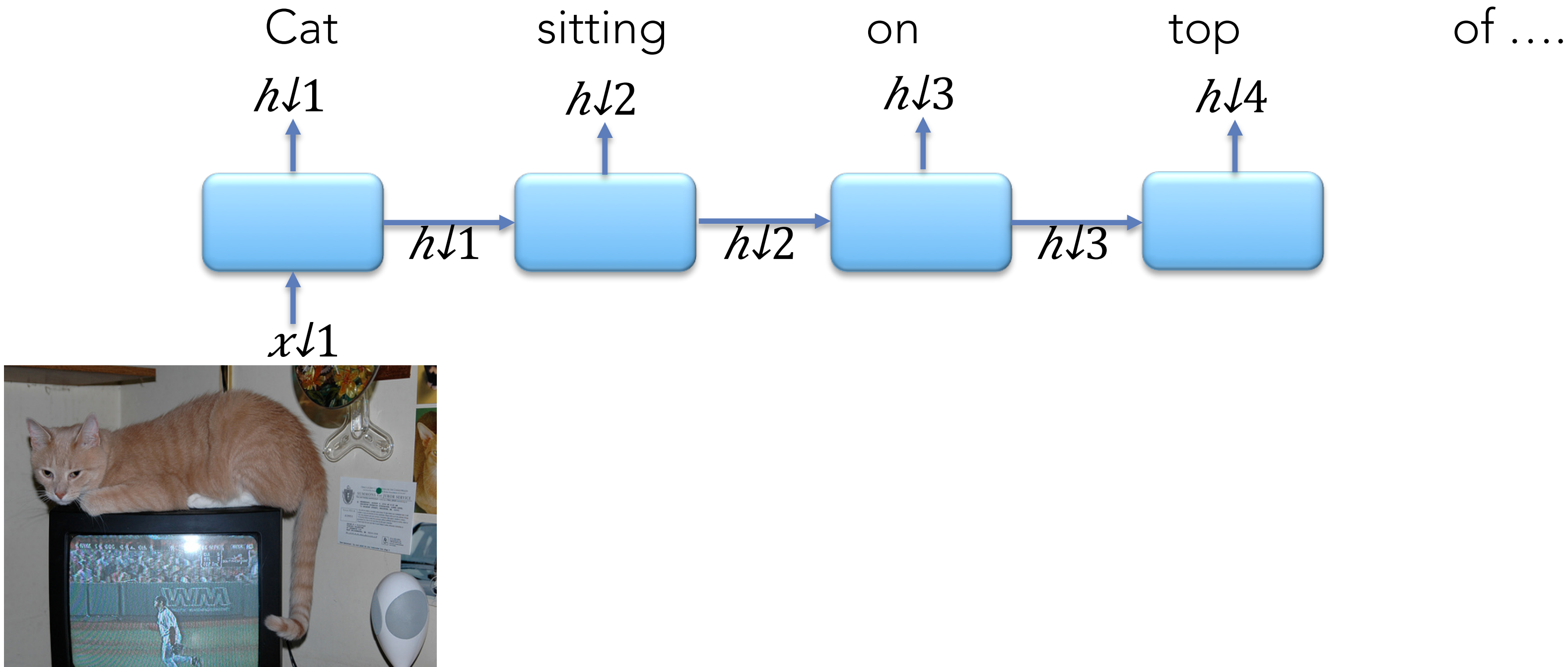
$$y = \mathrm{softmax}(V h_n)$$



Many uses of RNNs
2. one to seq

- Input: one item
- Output: a sequence
- Example: Image captioning

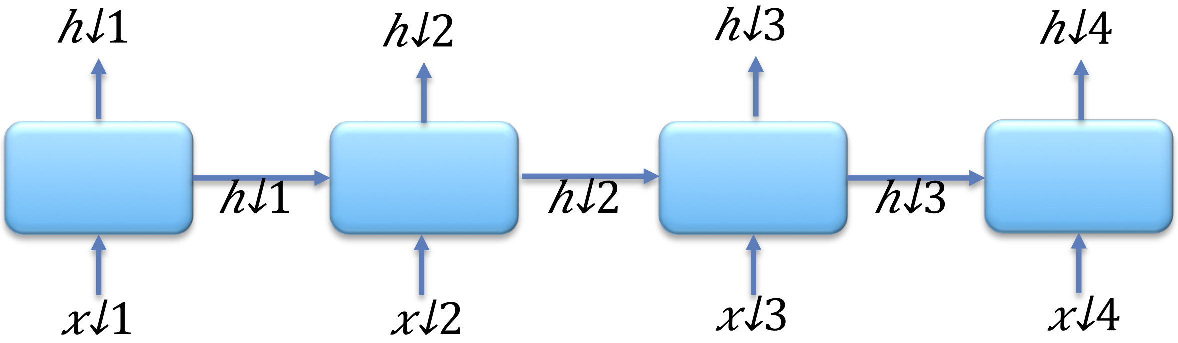
$$h_t = f(x_t, h_{t-1})$$
$$y_t = \text{softmax}(Vh_t)$$



Many uses of RNNs
3. sequence tagging

- Input: a sequence
- Output: a sequence (of the same length)
- Example: POS tagging, Named Entity Recognition
- How about Language Models?
  - Yes! RNNs can be used as LMs!
  - RNNs make markov assumption: T/F?

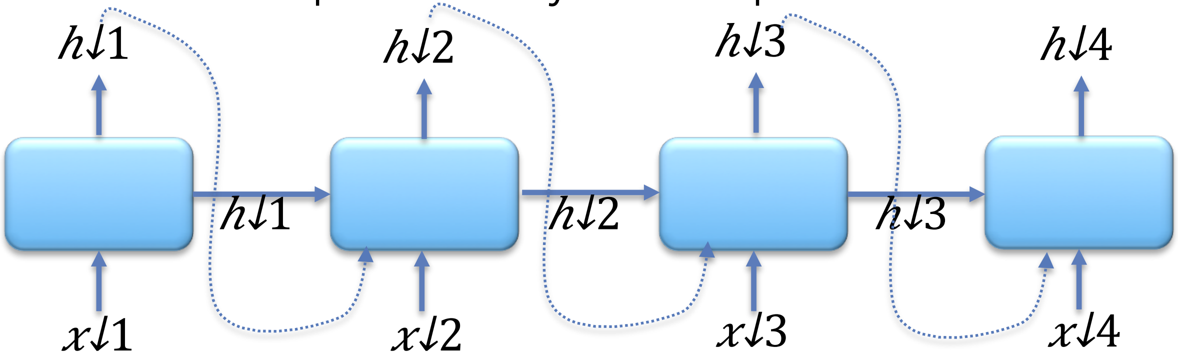
$$h_t = f(x_t, h_{t-1})$$
$$y_t = \text{softmax}(V h_t)$$



Many uses of RNNs
4. Language models

$$h_t = f(x_t, h_{t-1})$$
$$y_t = \text{softmax}(Vh_t)$$

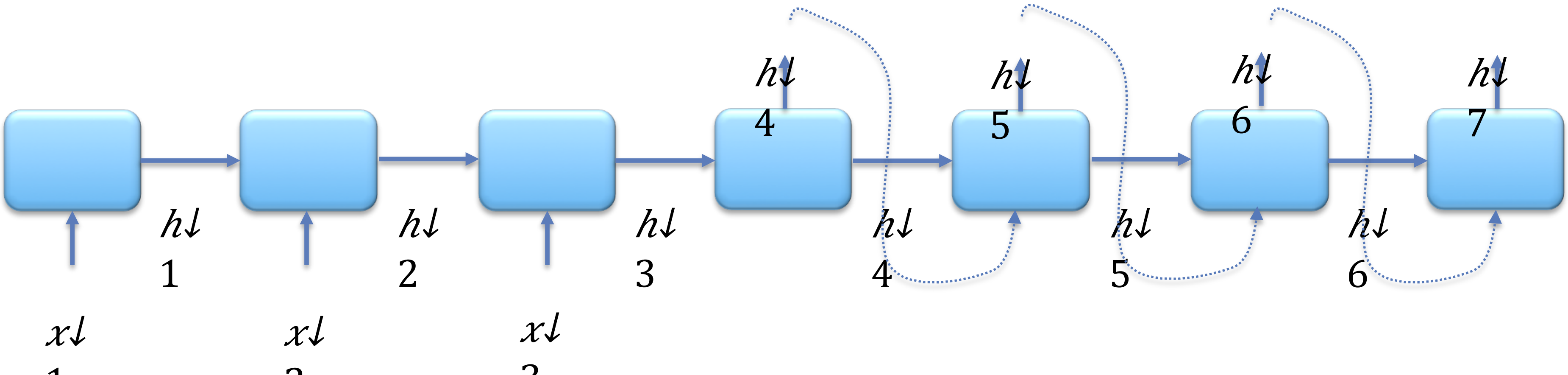
- Input: a sequence of words
- Output: one next word
- Output: or a sequence of next words
- During training, x_t is the actual word in the training sentence.
- During testing, x_t is the word predicted from the previous time step.
- Does RNN LMs make Markov assumption?
  - i.e., the next word depends only on the previous N words



Many uses of RNNs
5. seq2seq (aka "encoder-decoder")

- Input: a sequence
- Output: a sequence (of *different* length)
- Examples?

$$h_t = f(x_t, h_{t-1})$$
$$y_t = \text{softmax}(Vh_t)$$



Many uses of RNNs
4. seq2seq (aka “encoder-decoder”)

- Conversation and Dialogue
- Machine Translation

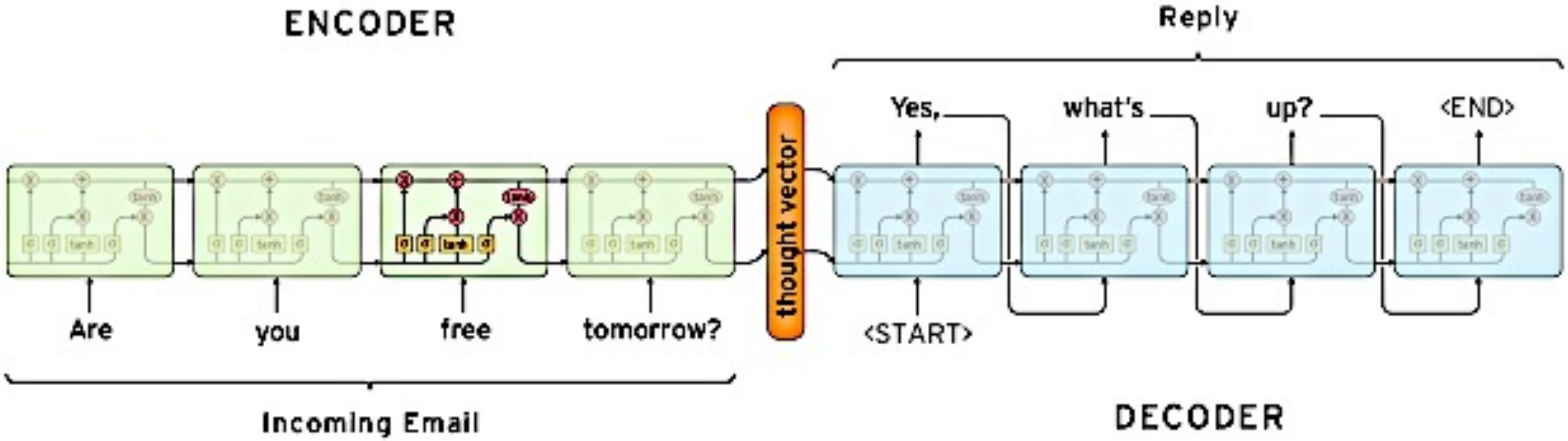


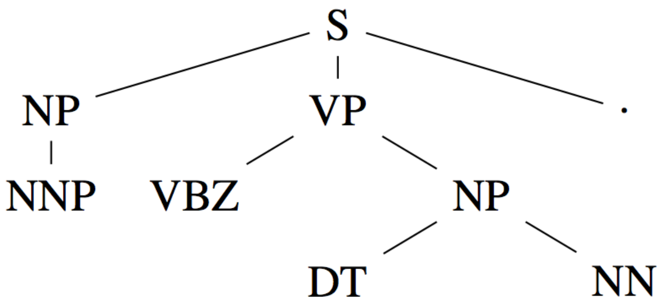
Figure from http://www.wildml.com/category/conversational-agents/

Many uses of RNNs

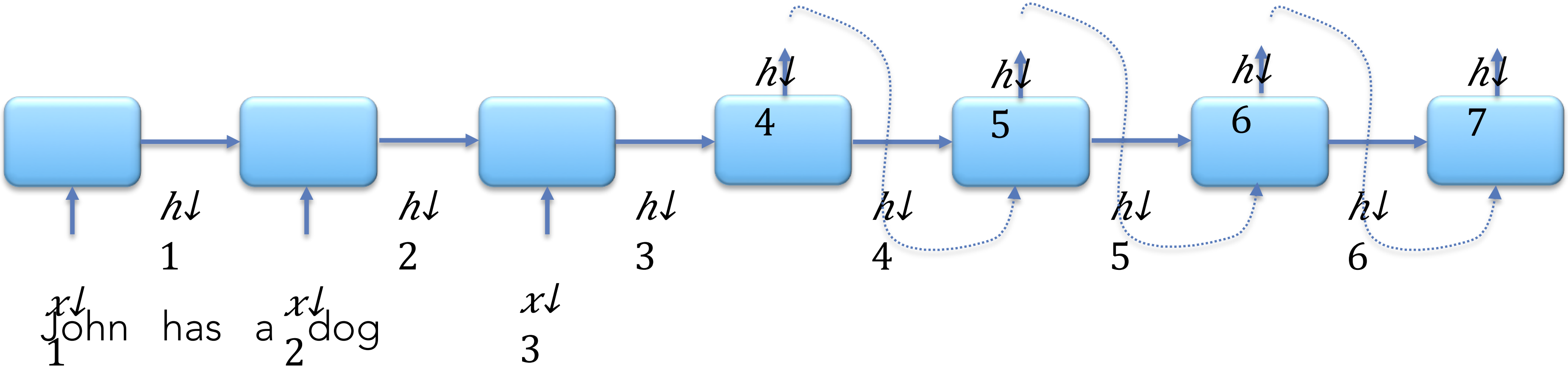
4. seq2seq (aka “encoder-decoder”)

Parsing!

- *“Grammar as Foreign Language”* (Vinyals et al., 2015)



(S (NP NNP $)_{NP}$ (VP VBZ (NP DT NN $)_{NP}$ $)_{VP}$. $)_{S}$



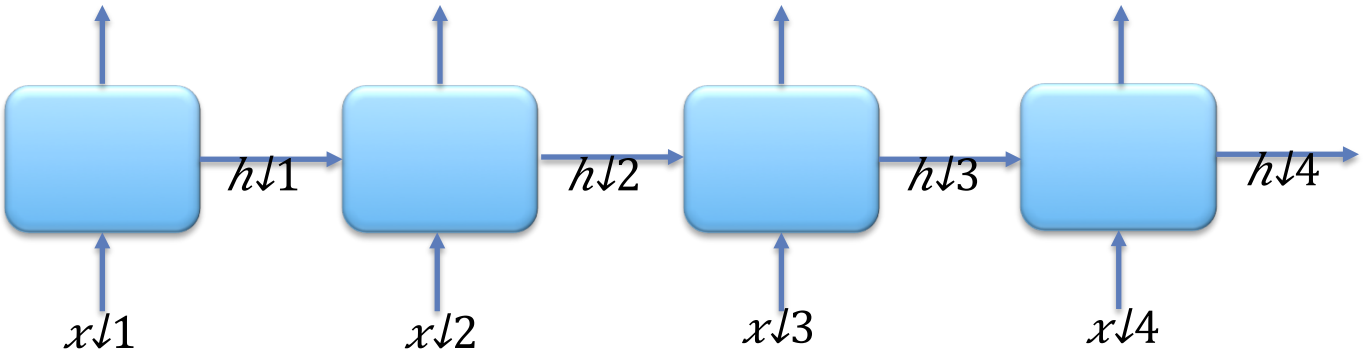
Recurrent Neural Networks (RNNs)

- Generic RNNs: $h_t = f(x_t, h_{t-1})$

  $y_t = \text{softmax}(V h_t)$

- Vanilla RNN: $h_t = \tanh(U x_t + W h_{t-1} + b)$

  $y_t = \text{softmax}(V h_t)$

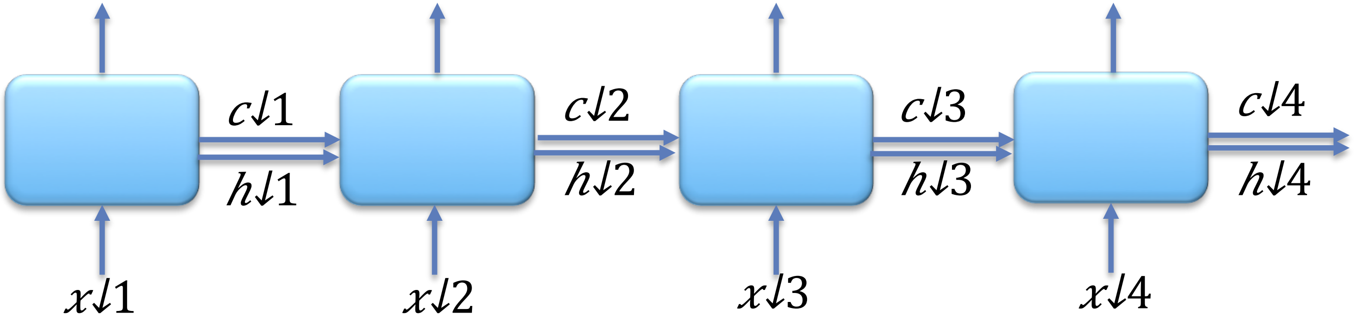


Recurrent Neural Networks (RNNs)

- Generic RNNs: $h_t = f(x_t, h_{t-1})$
- Vanilla RNNs: $h_t = \tanh(Ux_t + Wh_{t-1} + b)$
- LSTMs (Long Short-term Memory Networks):

$$\begin{aligned}
i_t &= \sigma(U^{(i)}x_t + W^{(i)}h_{t-1} + b^{(i)})\\
f_t &= \sigma(U^{(f)}x_t + W^{(f)}h_{t-1} + b^{(f)})\\
o_t &= \sigma(U^{(o)}x_t + W^{(o)}h_{t-1} + b^{(o)})\\
\tilde{c}_t &= \tanh(U^{(c)}x_t + W^{(c)}h_{t-1} + b^{(c)})\\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t\\
h_t &= o_t \circ \tanh(c_t)
\end{aligned}$$

There are many known variations to this set of equations!



c↓t: cell state

h↓t: hidden state

LSTMS (LONG SHORT-TERM MEMORY NETWORKS

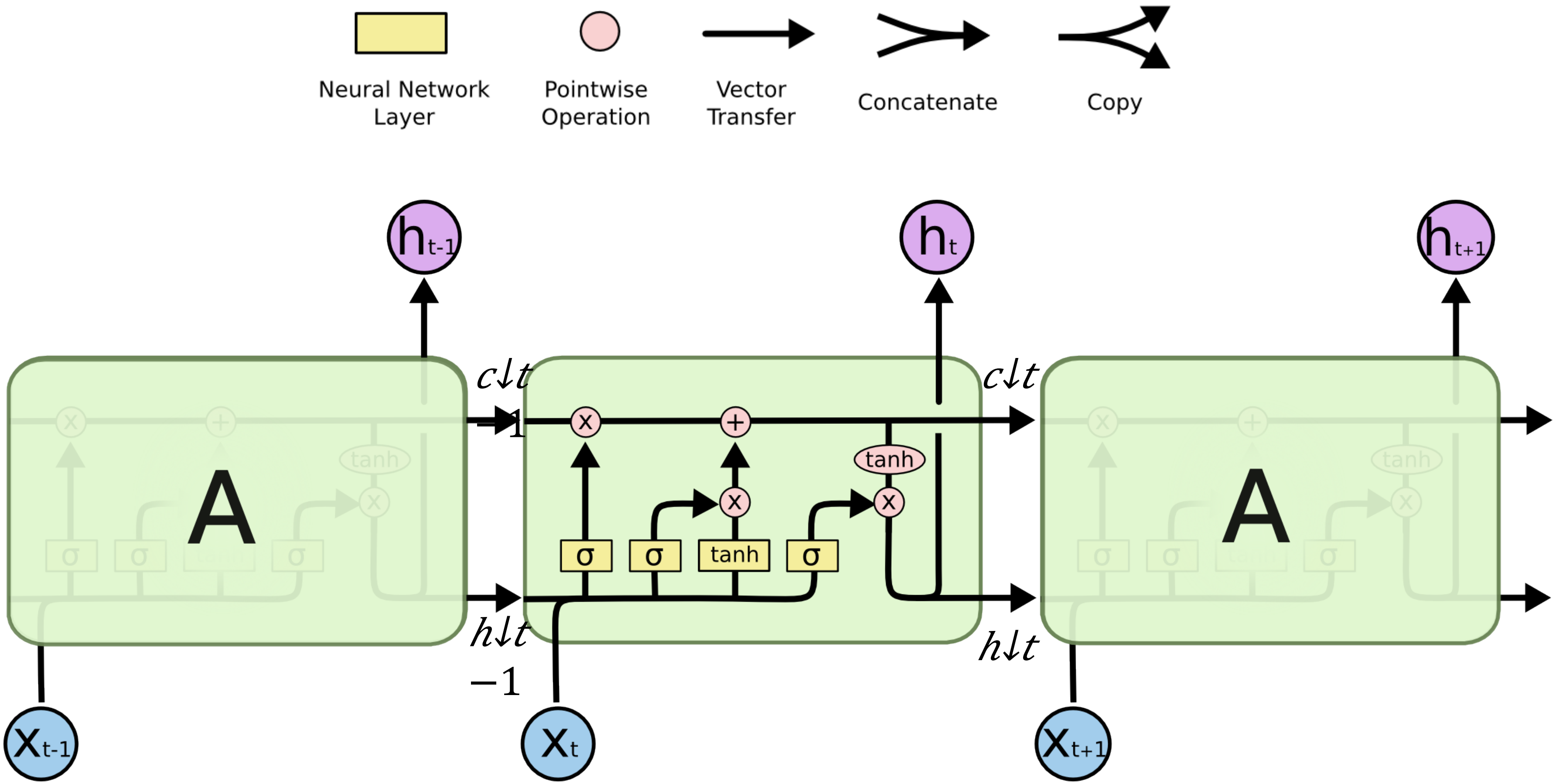
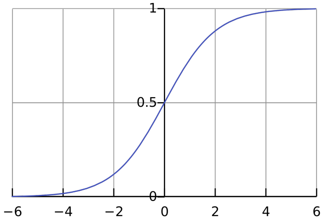


Figure by Christopher Olah (colah.github.io)

LSTMS (LONG SHORT-TERM MEMORY NETWORKS

sigmoid: [0,1]



Forget gate: forget the past or not

$$f_t = \sigma(U^{(f)}x_t + W^{(f)}h_{t-1} + b^{(f)})$$

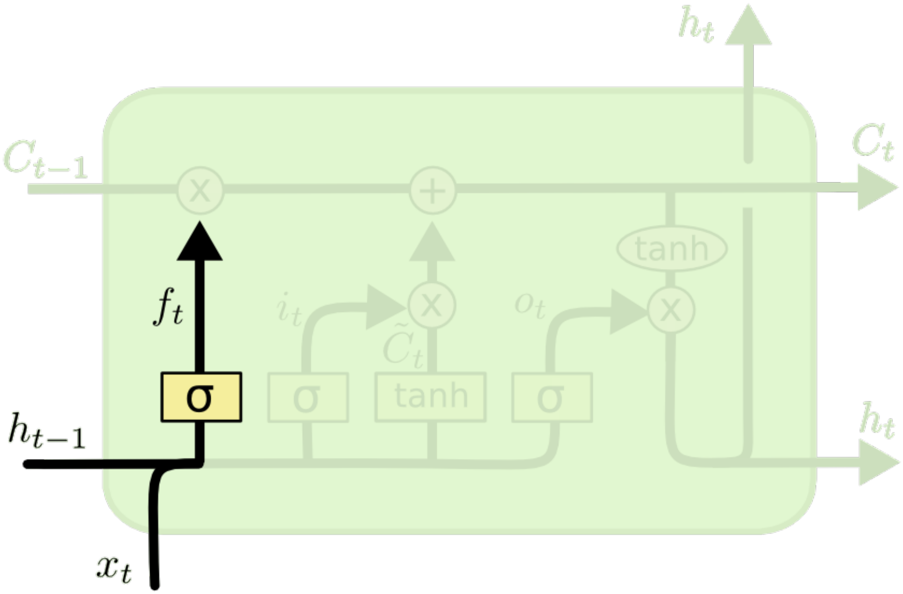
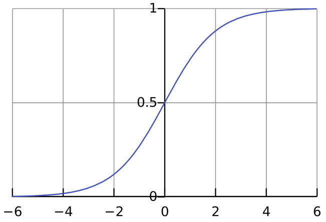


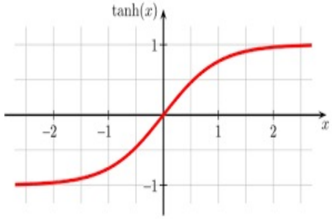
Figure by Christopher Olah (colah.github.io)

LSTMS (LONG SHORT-TERM MEMORY NETWORKS

sigmoid:
[0,1]



tanh:
[-1,1]



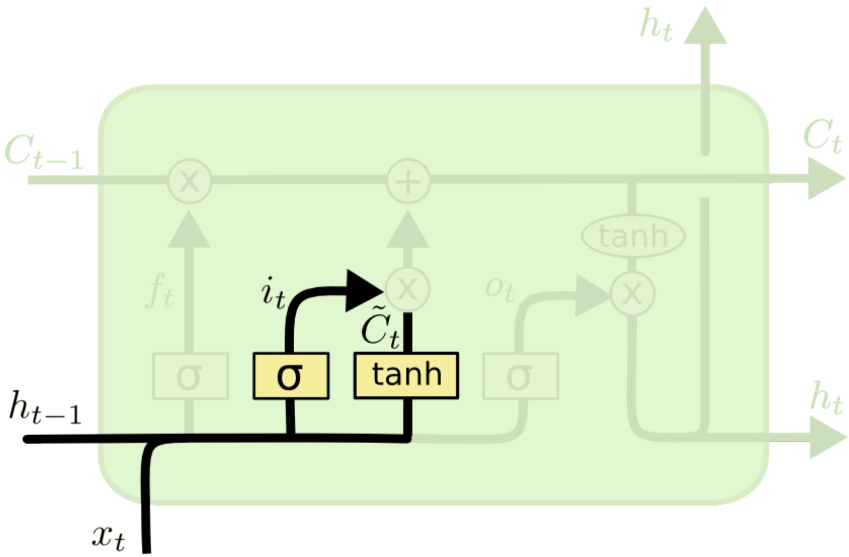


Figure by Christopher Olah (colah.github.io)

Forget gate: forget the past or not

$$f_t = \sigma(U^{(f)}x_t + W^{(f)}h_{t-1} + b^{(f)})$$

Input gate: use the input or not

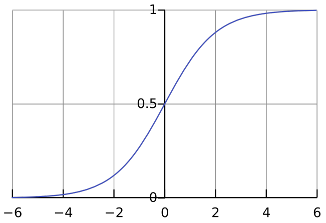
$$i_t = \sigma(U^{(i)}x_t + W^{(i)}h_{t-1} + b^{(i)})$$

New cell content (temp):

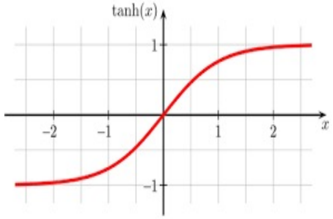
$$\tilde{c}_t = \tanh(U^{(c)}x_t + W^{(c)}h_{t-1} + b^{(c)})$$

LSTMS (LONG SHORT-TERM MEMORY NETWORKS

sigmoid:
[0,1]



tanh:
[-1,1]



Forget gate: forget the past or not

$$f_t = \sigma(U^{(f)}x_t + W^{(f)}h_{t-1} + b^{(f)})$$

Input gate: use the input or not

$$i_t = \sigma(U^{(i)}x_t + W^{(i)}h_{t-1} + b^{(i)})$$

New cell content (temp):

$$\tilde{c}_t = \tanh(U^{(c)}x_t + W^{(c)}h_{t-1} + b^{(c)})$$

New cell content:

- mix old cell with the new temp cell

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

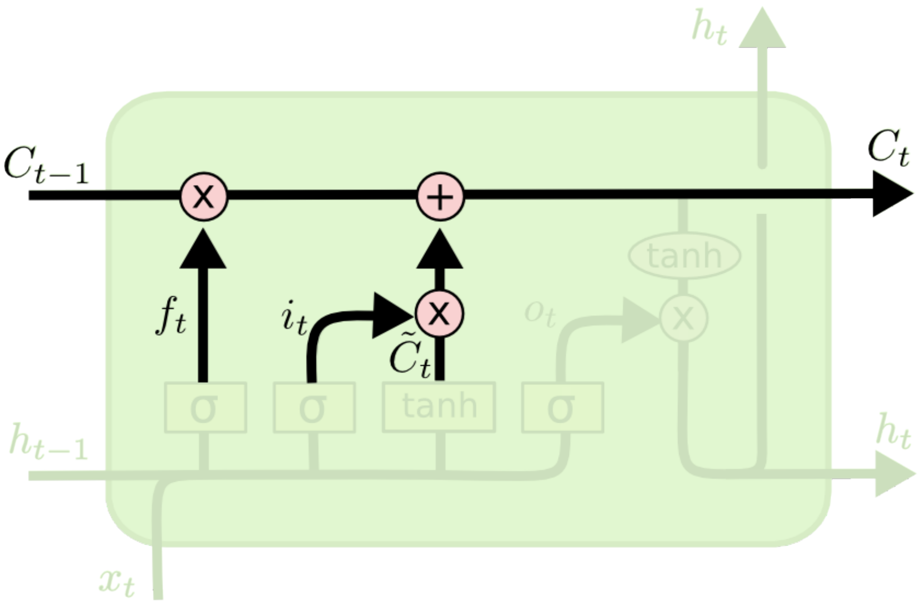


Figure by Christopher Olah (colah.github.io)

LSTMS (LONG SHORT-TERM MEMORY NETWORKS

Output gate: output from the new cell or not

$$o_t = \sigma(U^{(o)}x_t + W^{(o)}h_{t-1} + b^{(o)})$$

Hidden state:

$$h_t = o_t \circ \tanh(c_t)$$

Forget gate: forget the past or not

$$f_t = \sigma(U^{(f)}x_t + W^{(f)}h_{t-1} + b^{(f)})$$

Input gate: use the input or not

$$i_t = \sigma(U^{(i)}x_t + W^{(i)}h_{t-1} + b^{(i)})$$

New cell content (temp):

$$\tilde{c}_t = \tanh(U^{(c)}x_t + W^{(c)}h_{t-1} + b^{(c)})$$

New cell content:

- mix old cell with the new temp cell

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

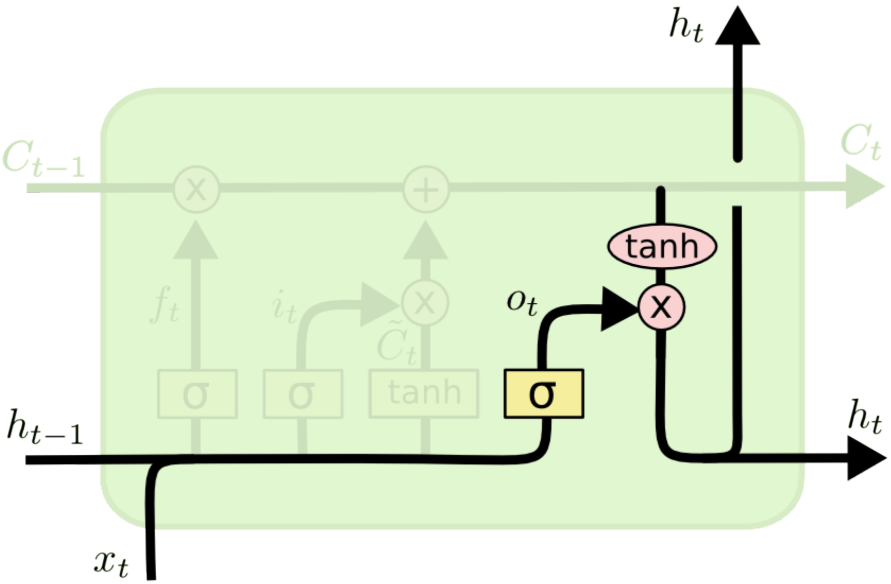


Figure by Christopher Olah (colah.github.io)

LSTMS (LONG SHORT-TERM MEMORY NETWORKS

Forget gate: forget the past or not

$$f_t = \sigma(U^{(f)}x_t + W^{(f)}h_{t-1} + b^{(f)})$$

Input gate: use the input or not

$$i_t = \sigma(U^{(i)}x_t + W^{(i)}h_{t-1} + b^{(i)})$$

Output gate: output from the new cell or not

$$o_t = \sigma(U^{(o)}x_t + W^{(o)}h_{t-1} + b^{(o)})$$

New cell content (temp):

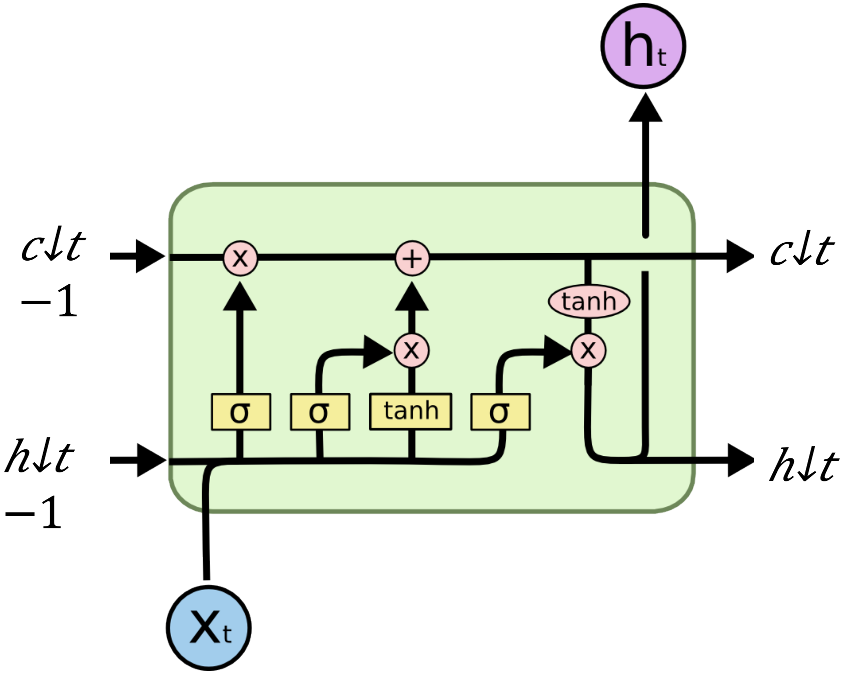
$$\tilde{c}_t = \tanh(U^{(c)}x_t + W^{(c)}h_{t-1} + b^{(c)})$$

New cell content:
- mix old cell with the new temp cell

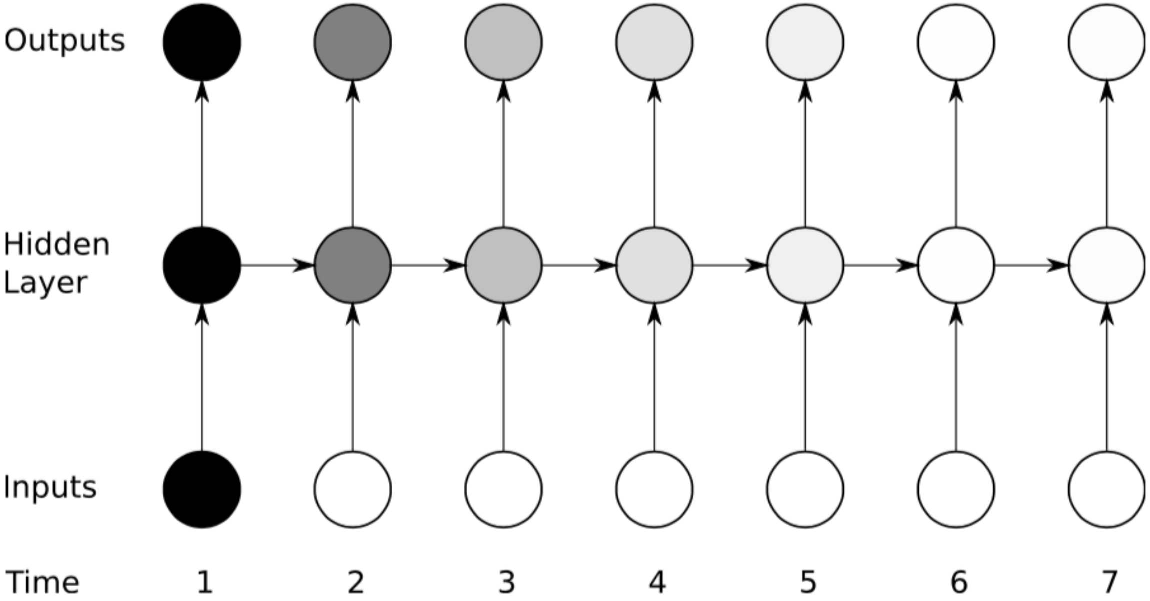
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

Hidden state:

$$h_t = o_t \circ \tanh(c_t)$$



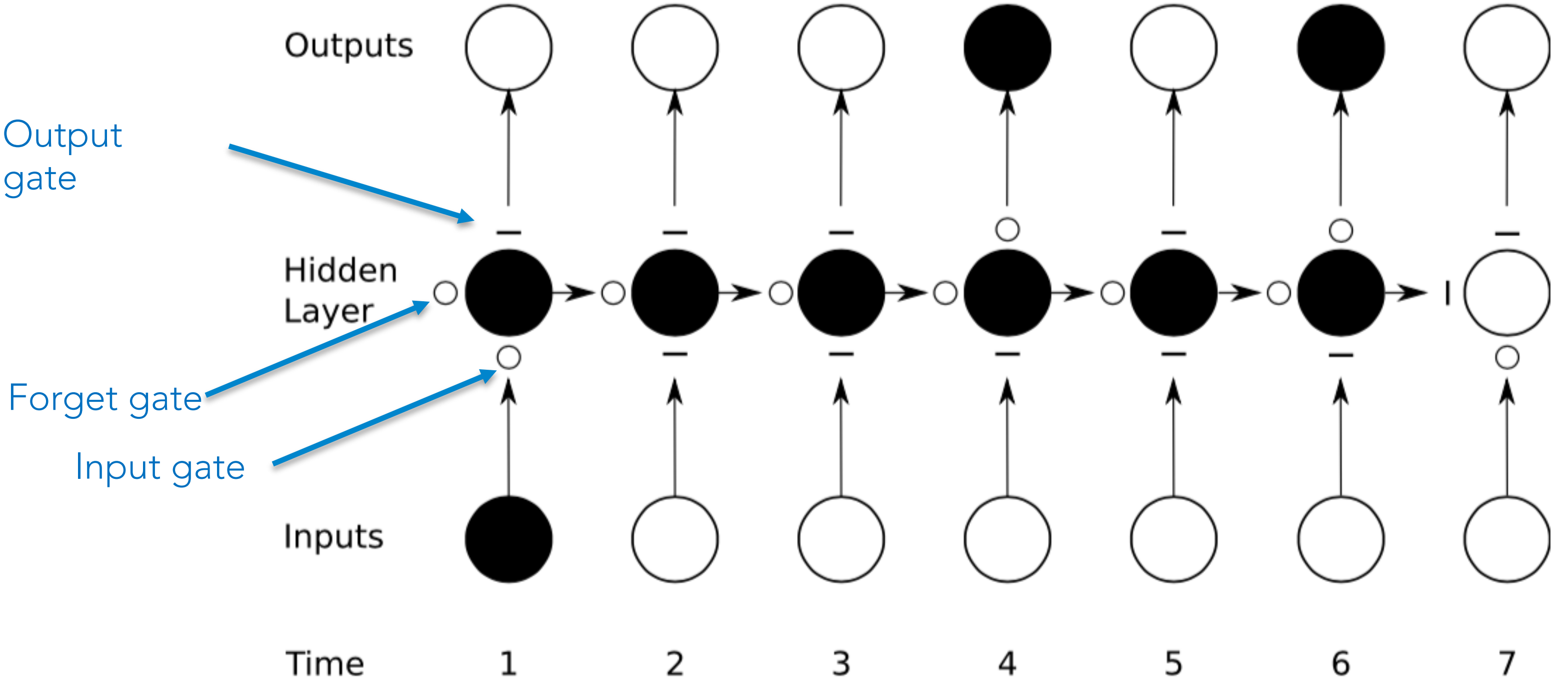
vanishing gradient problem for RNNs.



- The shading of the nodes in the unfolded network indicates their sensitivity to the inputs at time one (the darker the shade, the greater the sensitivity).
- The sensitivity decays over time as new inputs overwrite the activations of the hidden layer, and the network 'forgets' the first inputs.

Example from Graves 2012

Preservation of gradient information by LSTM



- For simplicity, all gates are either entirely open ('O') or closed ('—').
- The memory cell 'remembers' the first input as long as the forget gate is open and the input gate is closed.
- The sensitivity of the output layer can be switched on and off by the output gate without affecting the cell.

Example from Graves 2012

Recurrent Neural Networks (RNNs)

- Generic RNNs: $h_t = f(x_t, h_{t-1})$
- Vanilla RNNs: $h_t = \tanh(Ux_t + Wh_{t-1} + b)$
- GRUs (Gated Recurrent Units):

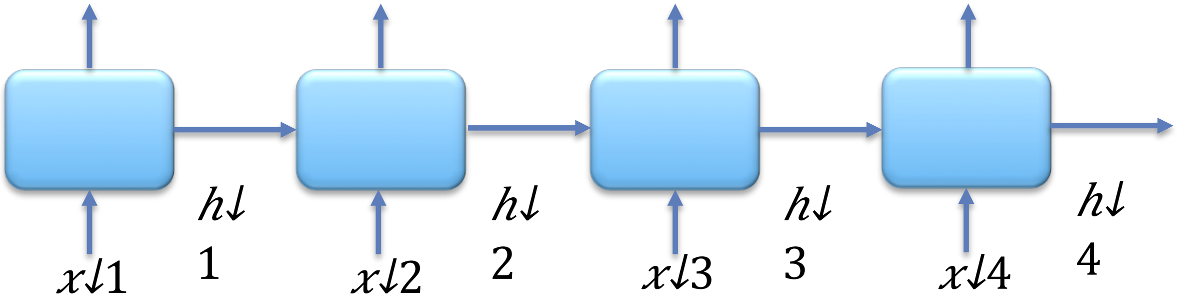
$$z_t = \sigma(U^{(z)}x_t + W^{(z)}h_{t-1} + b^{(z)})$$

$$r_t = \sigma(U^{(r)}x_t + W^{(r)}h_{t-1} + b^{(r)})$$

$$\tilde{h_t} = \tanh(U^{(h)}x_t + W^{(h)}(r_t \circ h_{t-1}) + b^{(h)})$$

$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \tilde{h_t}$$

Z: Update gate
R: Reset gate

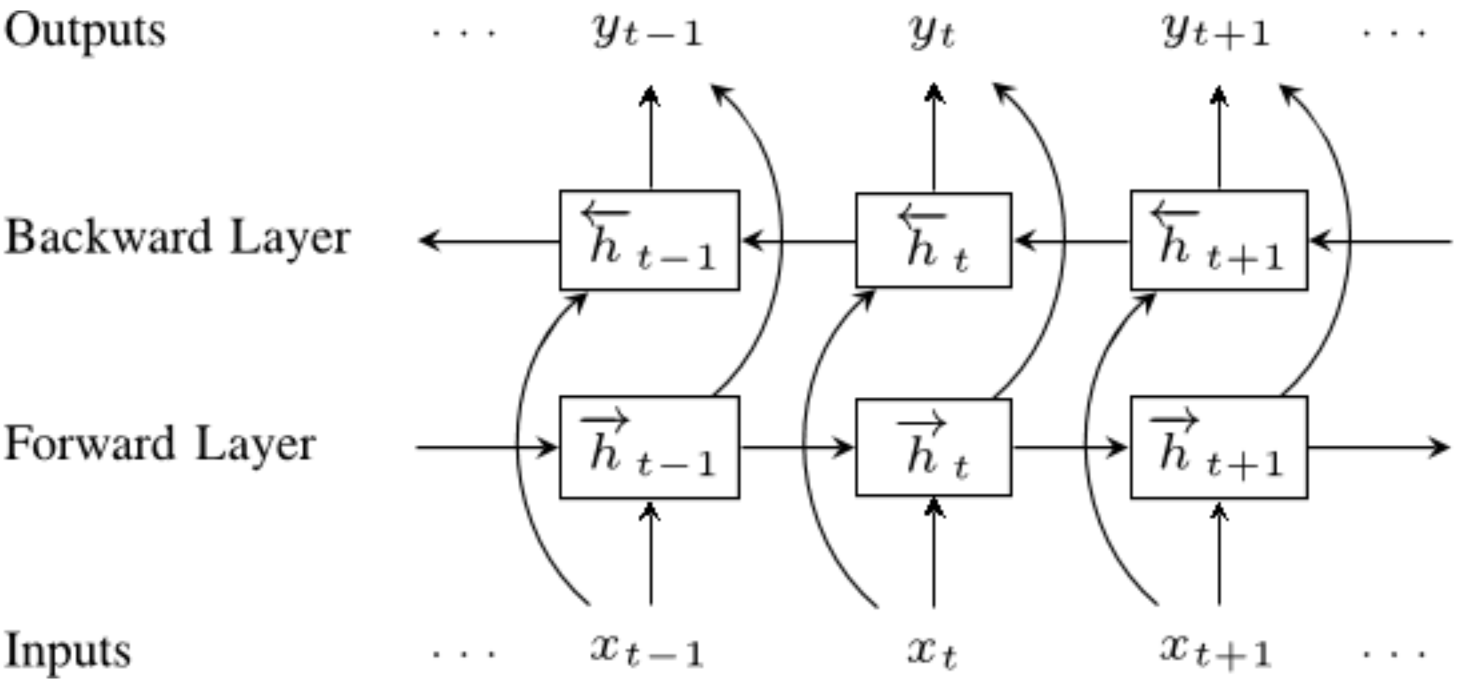


Less parameters than LSTMs. Easier to train for comparable performance!

Gates

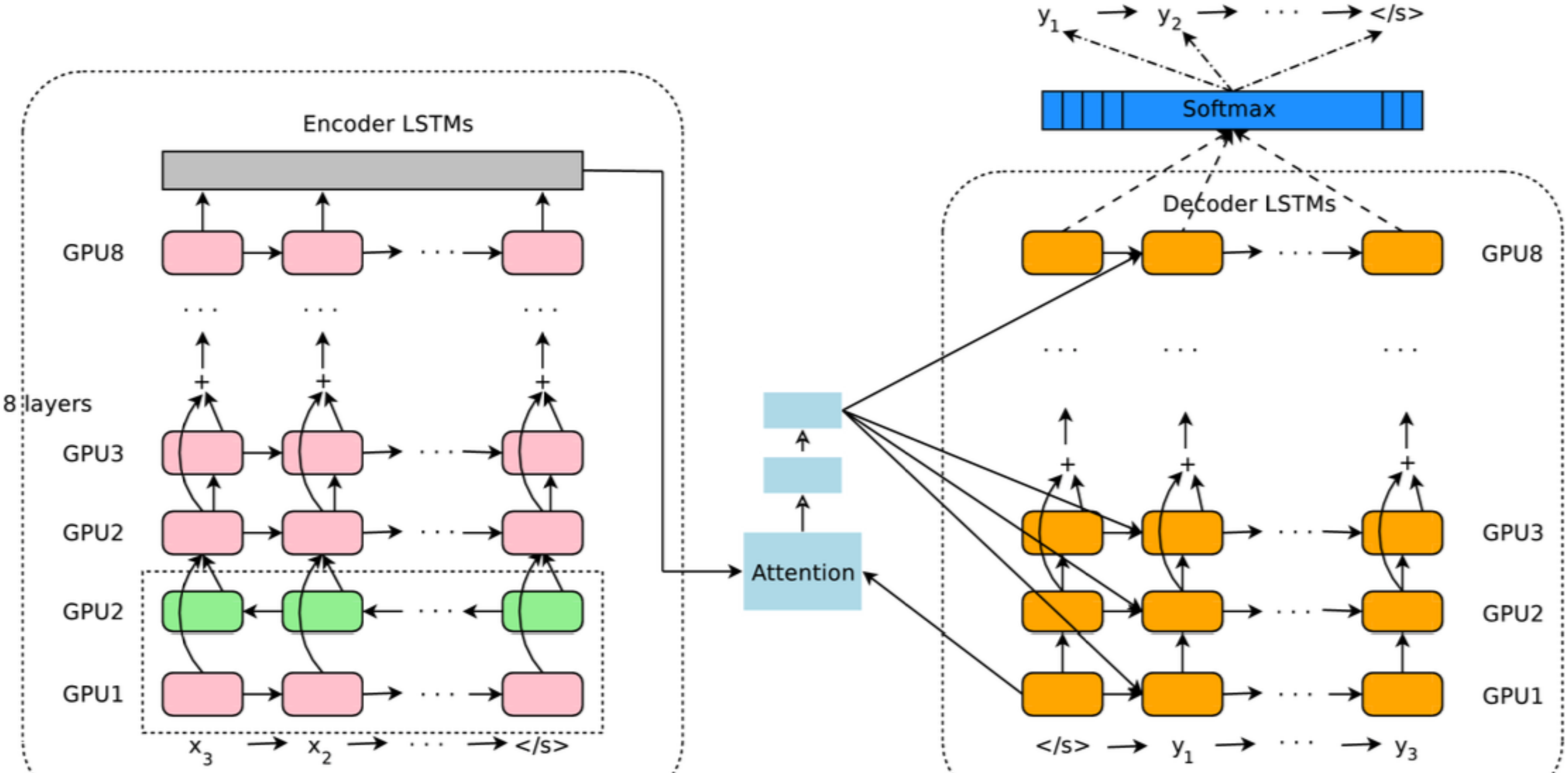
- Gates contextually control information flow
- Open/close with sigmoid
- In LSTMs and GRUs, they are used to (contextually) maintain longer term history

Bi-directional RNNs



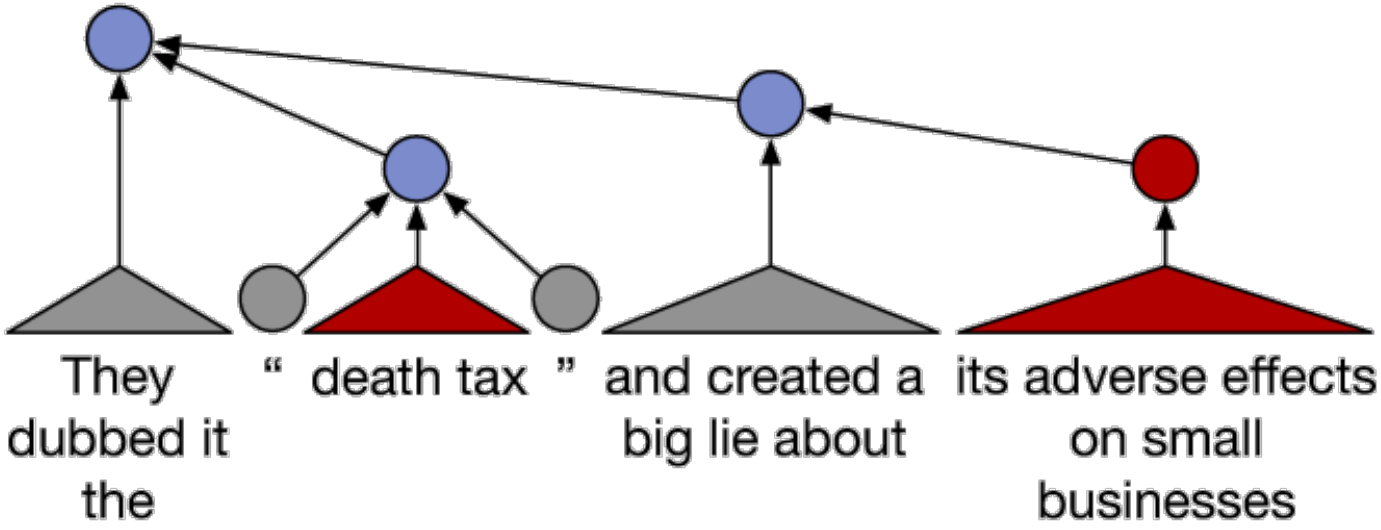
- Can incorporate context from both directions
- Generally improves over uni-directional RNNs

Google NMT (Oct 2016)



Recursive Neural Networks

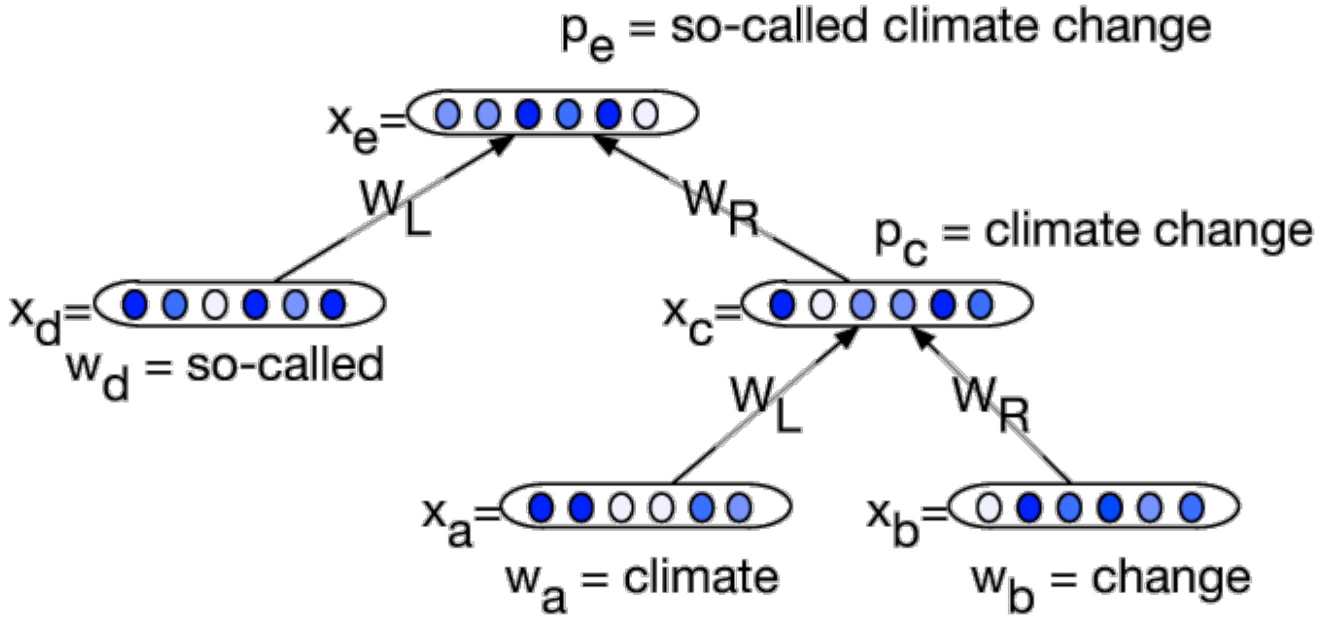
- Sometimes, inference over a tree structure makes more sense than sequential structure
- An example of compositionality in ideological bias detection (red → conservative, blue → liberal, gray → neutral) in which modifier phrases and punctuation cause polarity switches at higher levels of the parse tree


They dubbed it the
“
death tax
”
and created a big lie about
its adverse effects on small businesses

Example from Iyyer et al., 2014

Recursive Neural Networks

- NNs connected as a tree
- Tree structure is fixed a priori
- Parameters are shared, similarly as RNNs



Example from Iyyer et al., 2014

Tree LSTMs

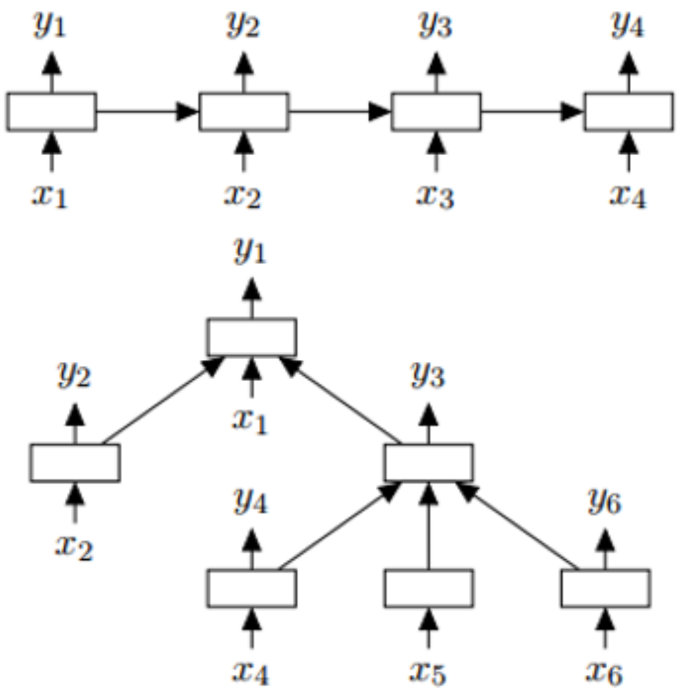
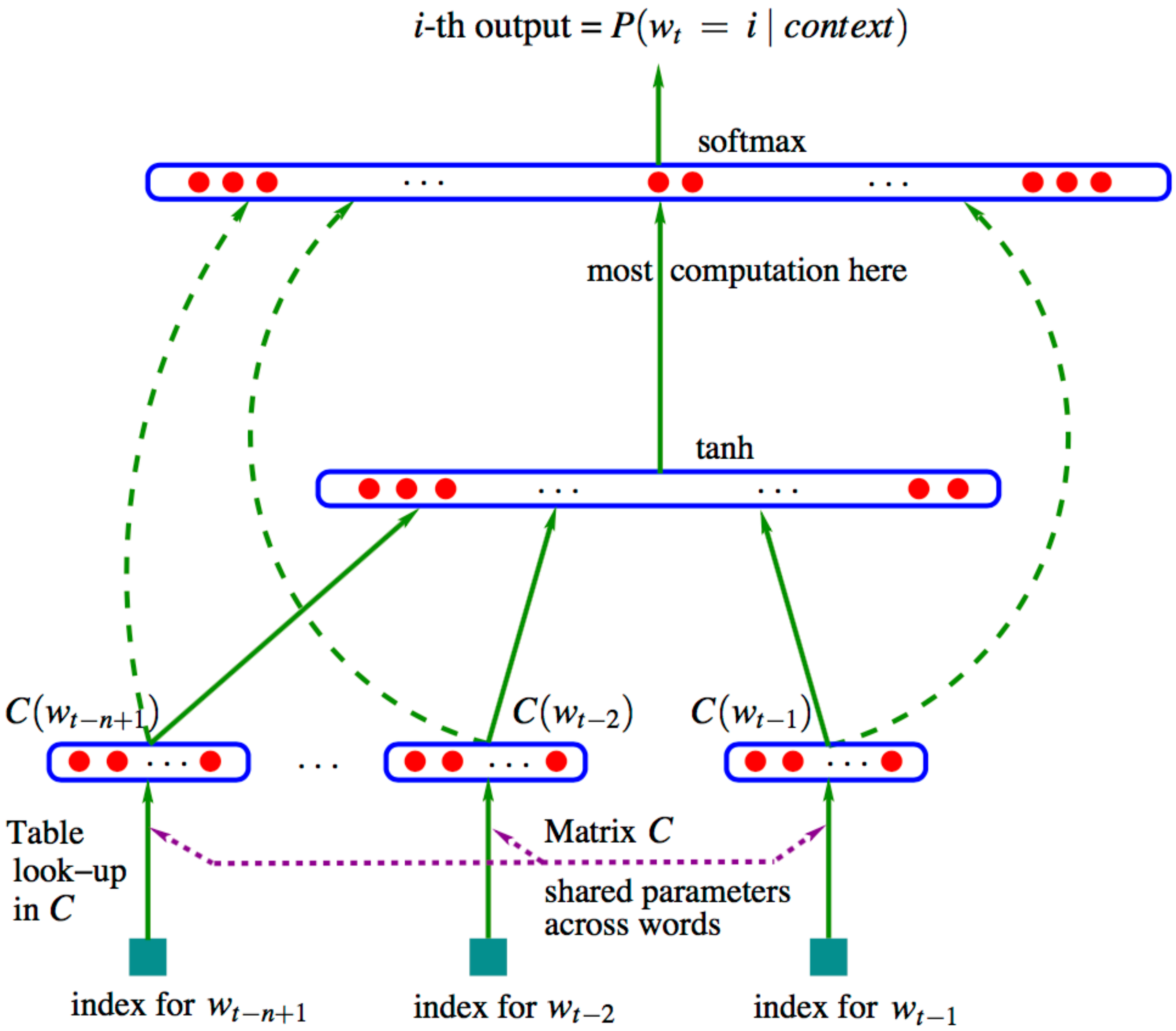


Figure 1: **Top:** A chain-structured LSTM network. **Bottom:** A tree-structured LSTM network with arbitrary branching factor.

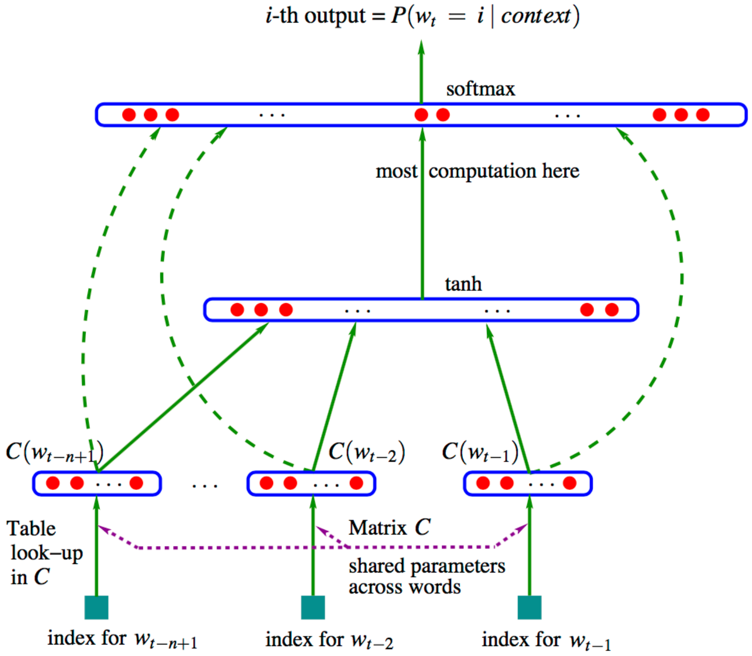
- Are tree LSTMs more expressive than sequence LSTMs?
- I.e., *recursive* vs *recurrent*

- When Are Tree Structures Necessary for Deep Learning of Representations? Jiwei Li, Minh-Thang Luong, Dan Jurafsky and Eduard Hovy. EMNLP, 2015.

Neural Probabilistic Language Model (Bengio 2003)



Neural Probabilistic Language Model (Bengio 2003)



- Each word prediction is a separate feed forward neural network
- Feedforward NNLM is a Markovian language model
- Dashed lines show optional direct connections

$$NN_{DMLP1}(\mathbf{x}) = [\tanh(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1), \mathbf{x}]W^2 + \mathbf{b}^2$$

- $\mathbf{W}^1 \in \mathbb{R}^{d_{\text{in}} \times d_{\text{hid}}}$, $\mathbf{b}^1 \in \mathbb{R}^{1 \times d_{\text{hid}}}$; first affine transformation
- $\mathbf{W}^2 \in \mathbb{R}^{(d_{\text{hid}} + d_{\text{in}}) \times d_{\text{out}}}$, $\mathbf{b}^2 \in \mathbb{R}^{1 \times d_{\text{out}}}$; second affine transformation

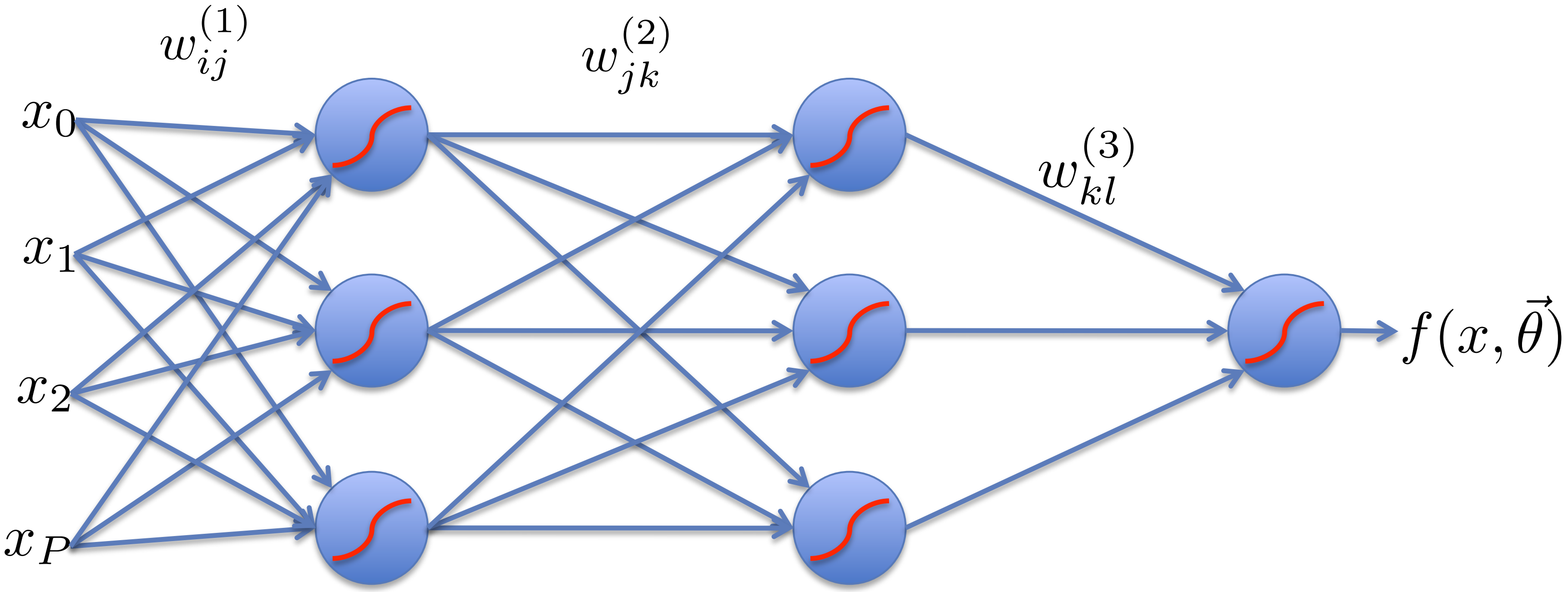
LEARNING: BACKPROPAGATION

Next 10 slides on back propagation are adapted from Andrew Rosenberg

Error Backpropagation

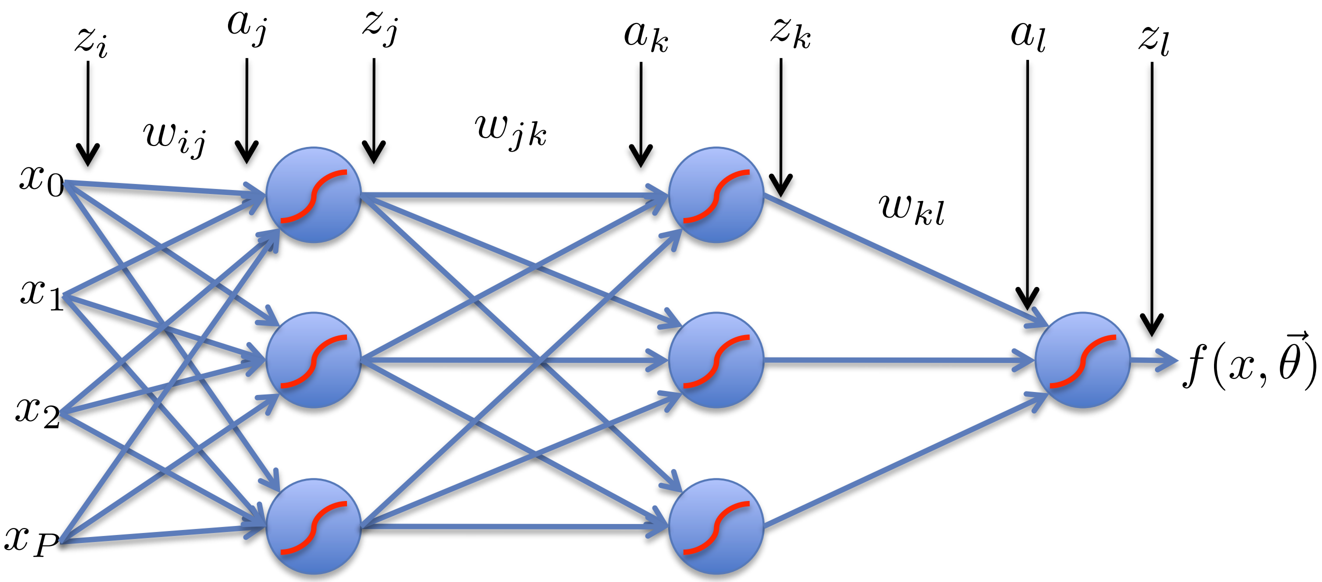
- Model parameters: $\vec{\theta} = \{w_{ij}^{(1)}, w_{jk}^{(2)}, w_{kl}^{(3)}\}$

for brevity: $\vec{\theta} = \{w_{ij}, w_{jk}, w_{kl}\}$



Learning: Gradient Descent

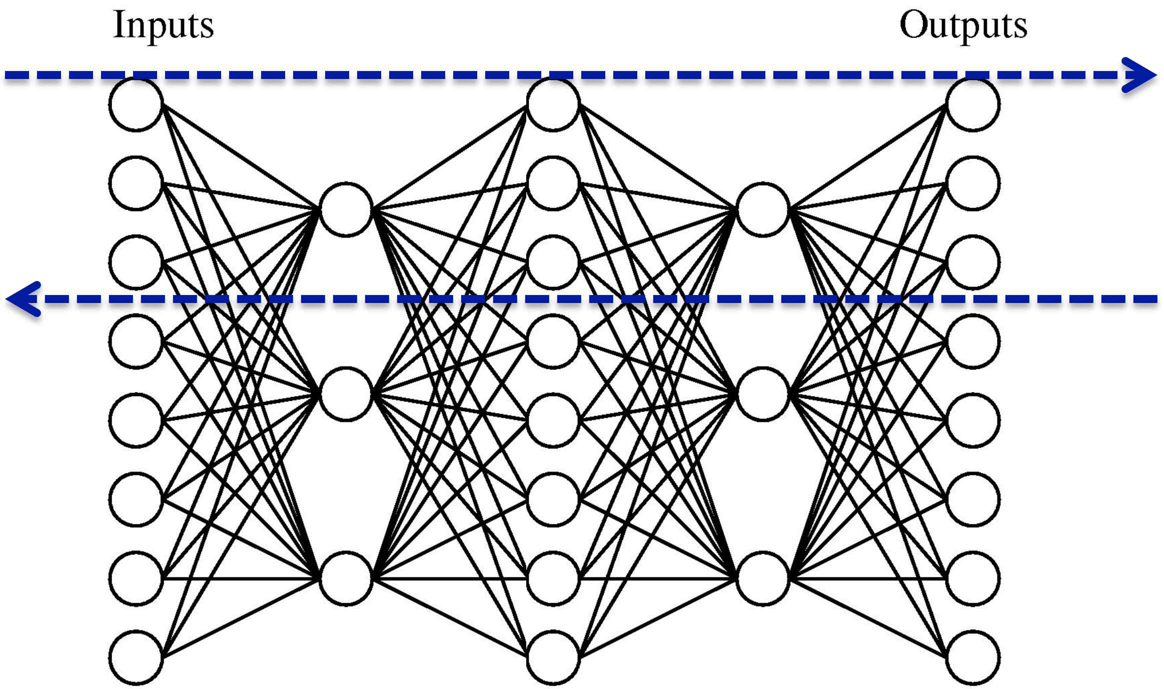
$$
\begin{aligned}
w_{ij}^{t+1} &= w_{ij}^{t} - \eta \frac{\partial R}{w_{ij}} \\
w_{jk}^{t+1} &= w_{jk}^{t} - \eta \frac{\partial R}{w_{kl}} \\
w_{kl}^{t+1} &= w_{kl}^{t} - \eta \frac{\partial R}{w_{kl}}
\end{aligned}
$$



Backpropagation

- Starts with a forward sweep to compute all the intermediate function values z_i
- Through backprop, computes the partial derivatives recursively δ_j $\frac{\partial R}{\partial w_{ij}}$
- A form of dynamic programming
  - Instead of considering exponentially many paths between a weight w_ij and the final loss (risk), store and reuse intermediate results.
- A type of automatic differentiation. (there are other variants e.g., recursive differentiation only through forward propagation

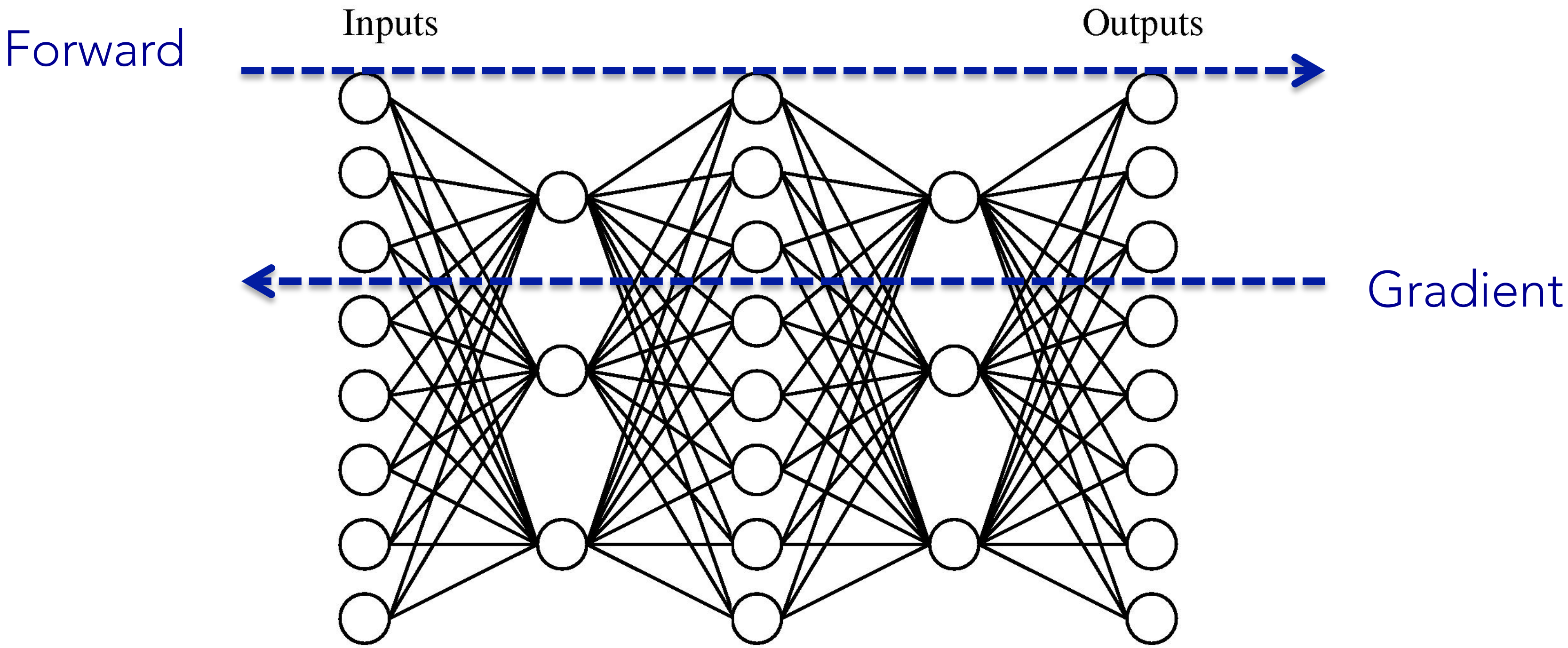
Forward



Gradient

Backpropagation

	Primary Interface Language
TensorFlow (https://www.tensorflow.org/)	Python
Torch (http://torch.ch/)	Lua
Theano (http://deeplearning.net/software/theano/)	Python
CNTK (https://github.com/Microsoft/CNTK)	C++
cnn (https://github.com/clab/cnn)	C++
Caffe (http://caffe.berkeleyvision.org/)	C++



Cross Entropy Loss (aka log loss, logistic loss)

- Cross Entropy

$$H(p,q) = -\sum_{y} p(y)\ \log\ q(y)$$

Predicted prob

True prob

- Related quantities
  - Entropy

$$H(p) = \sum_{y} p(y) \log\ p(y)$$

  - KL divergence (the distance between two distributions p and q)

$$D_{KL}(p||q) = \sum_{y} p(y)\ \log\ \frac{p(y)}{q(y)}$$

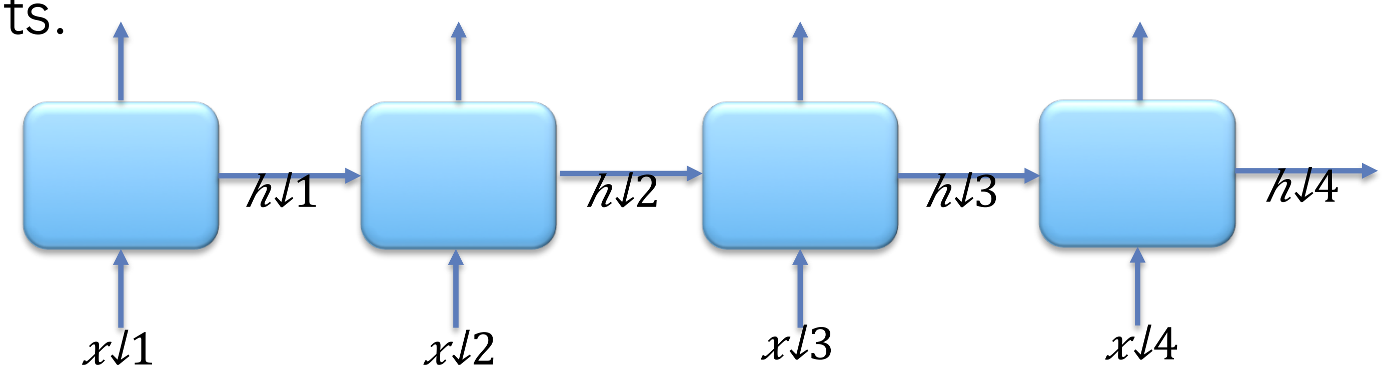
$$H(p,q) = E_p[-\log\ q] = H(p) + D_{KL}(p||q)$$

- Use Cross Entropy for models that should have more probabilistic flavor (e.g., language models)
- Use Mean Squared Error loss for models that focus on correct/incorrect predictions

$$\mathrm{MSE} = \frac{1}{2}(y - f(x))^2$$

RNN Learning: Backprop Through Time (BPTT)

- Similar to backprop with non-recurrent NNs
- But unlike feedforward (non-recurrent) NNs, each unit in the computation graph repeats the exact same parameters…
- Backprop gradients of the parameters of each unit as if they are different parameters
- When updating the parameters using the gradients, use the average gradients throughout the entire chain of units.



LEARNING: TRAINING DEEP NETWORKS

Vanishing / exploding Gradients

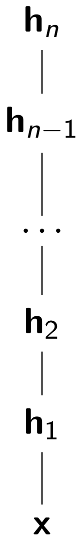
- Deep networks are hard to train
- Gradients go through multiple layers
- The multiplicative effect tends to lead to *exploding* or *vanishing* gradients
- Practical solutions w.r.t.
  - network architecture
  - numerical operations

Vanishing / exploding Gradients

- Practical solutions w.r.t. network architecture
  - Add skip connections to reduce distance
    - Residual networks, highway networks, …
  - Add gates (and memory cells) to allow longer term memory
    - LSTMs, GRUs, memory networks, …

Gradients of deep networks

$$NN_{layer}(\mathbf{x}) = \text{ReLU}(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1)$$



- Can have similar issues with vanishing gradients.

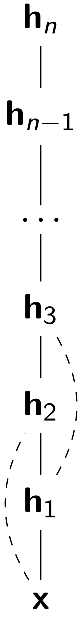
$$\frac{\partial L}{\partial h_{n-1,j_{n-1}}} = \sum_{j_n} \mathbf{1}(h_{n,j_n} > 0) W_{j_{n-1},j_n} \frac{\partial L}{\partial h_{n,j_n}}$$

Diagram borrowed from Alex Rush

Effects of Skip Connections on Gradients

- Thought Experiment: Additive Skip-Connections

$$NN_{sl1}(\mathbf{x}) = \frac{1}{2}\,\mathrm{ReLU}(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) + \frac{1}{2}\mathbf{x}$$



$$\begin{aligned}\frac{\partial L}{\partial h_{n-1,j_{n-1}}} = \quad & \tfrac{1}{2} \quad (\sum_{j_n} \mathbf{1}(h_{n,j_n} > 0) W_{j_{n-1},j_n} \frac{\partial L}{\partial h_{n,j_n}}) + \\ & \tfrac{1}{2} \quad (h_{n-1,j_{n-1}} \frac{\partial L}{\partial h_{n,j_{n-1}}})\end{aligned}$$

Diagram borrowed from Alex Rush

Effects of Skip Connections on Gradients

- Thought Experiment: Dynamic Skip-Connections

$$\begin{aligned} NN_{sl2}(\mathbf{x}) &= (1-t)\,\mathrm{ReLU}(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1) + t\mathbf{x} \\ t &= \sigma(\mathbf{x}\mathbf{W}^t + b^t) \\ \mathbf{W}^1 &\in \mathbb{R}^{d_{\mathrm{hid}} \times d_{\mathrm{hid}}} \\ \mathbf{W}^t &\in \mathbb{R}^{d_{\mathrm{hid}} \times 1} \end{aligned}$$

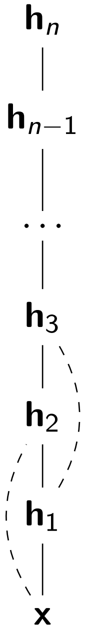


Diagram borrowed from Alex Rush

Highway Network (Srivastava et al., 2015)

- A plain feedforward neural network:

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}).$$

  - H is a typical affine transformation followed by a non-linear activation

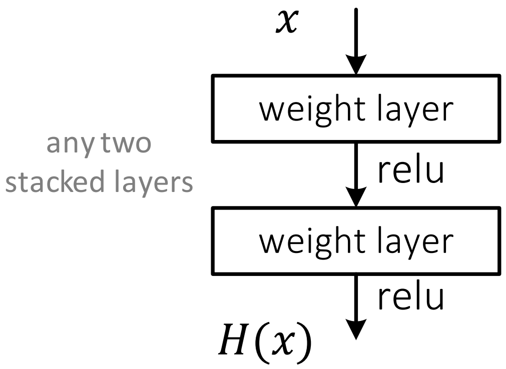
- Highway network:

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot C(\mathbf{x}, \mathbf{W_C}).$$

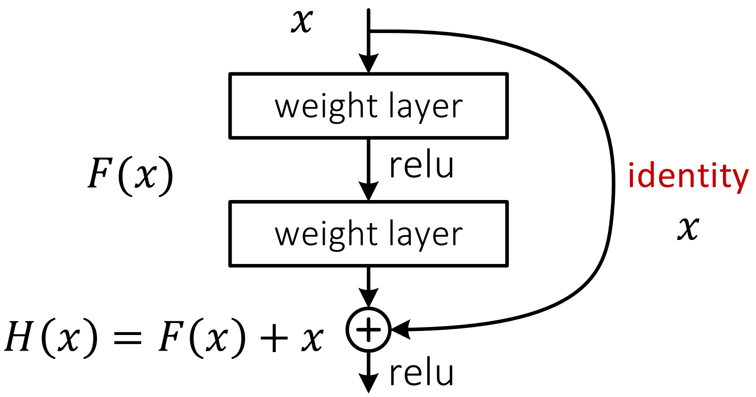
  - T is a "transform gate"
  - C is a "carry gate"
  - Often C = 1 – T for simplicity

Residual Networks

- Plaint net



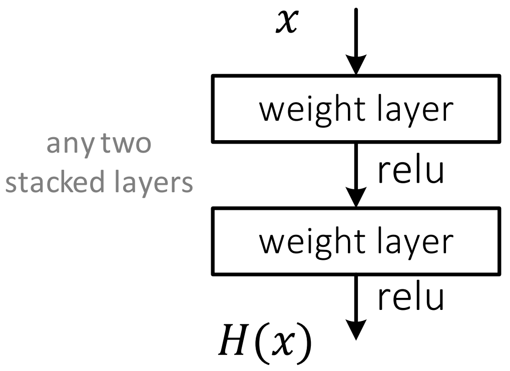
- Residual net



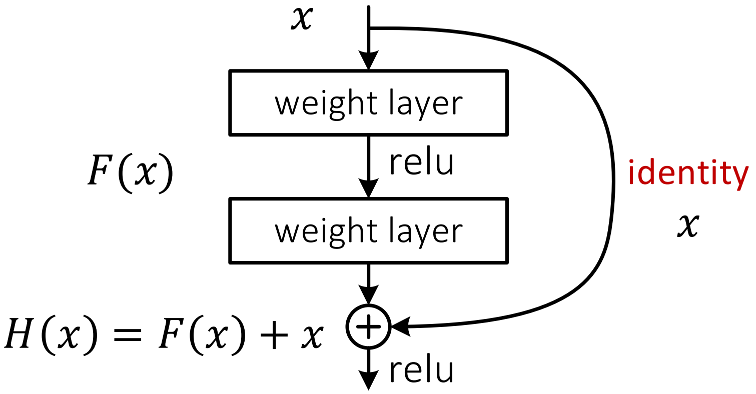
- ResNet (He et al. 2015): first very deep (152 layers) network successfully trained for object recognition

Residual Networks

- Plaint net



- Residual net

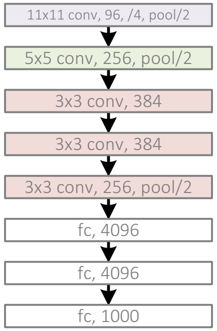


- F(x) is a residual mapping with respect to identity
- Direct input connection +x leads to a nice property w.r.t. back propagation --- more direct influence from the final loss to any deep layer
- In contrast, LSTMs & Highway networks allow for long distance input connection only through "gates".

Residual Networks

Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)


11x11 conv, 96, /4, pool/2

5x5 conv, 256, pool/2

3x3 conv, 384

3x3 conv, 384

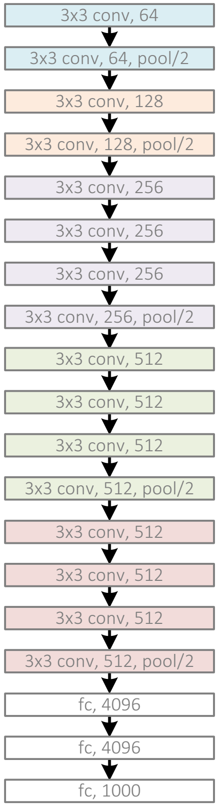
3x3 conv, 256, pool/2

fc, 4096

fc, 4096

fc, 1000

VGG, 19 layers
(ILSVRC 2014)


3x3 conv, 64

3x3 conv, 64, pool/2

3x3 conv, 128

3x3 conv, 128, pool/2

3x3 conv, 256

3x3 conv, 256

3x3 conv, 256

3x3 conv, 256, pool/2

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512, pool/2

3x3 conv, 512

3x3 conv, 512

3x3 conv, 512

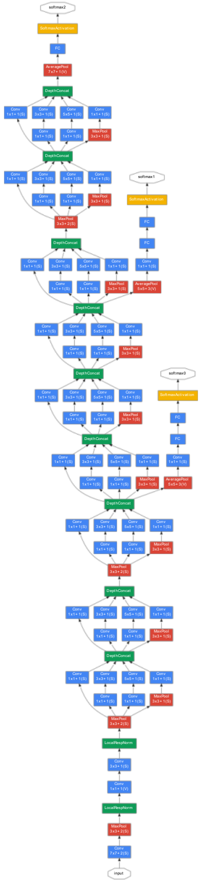
3x3 conv, 512, pool/2

fc, 4096

fc, 4096

fc, 1000

GoogleNet, 22 layers
(ILSVRC 2014)



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. CVPR 2016

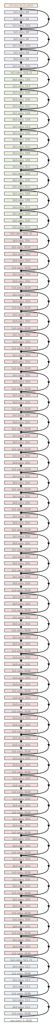
Residual Networks

Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)

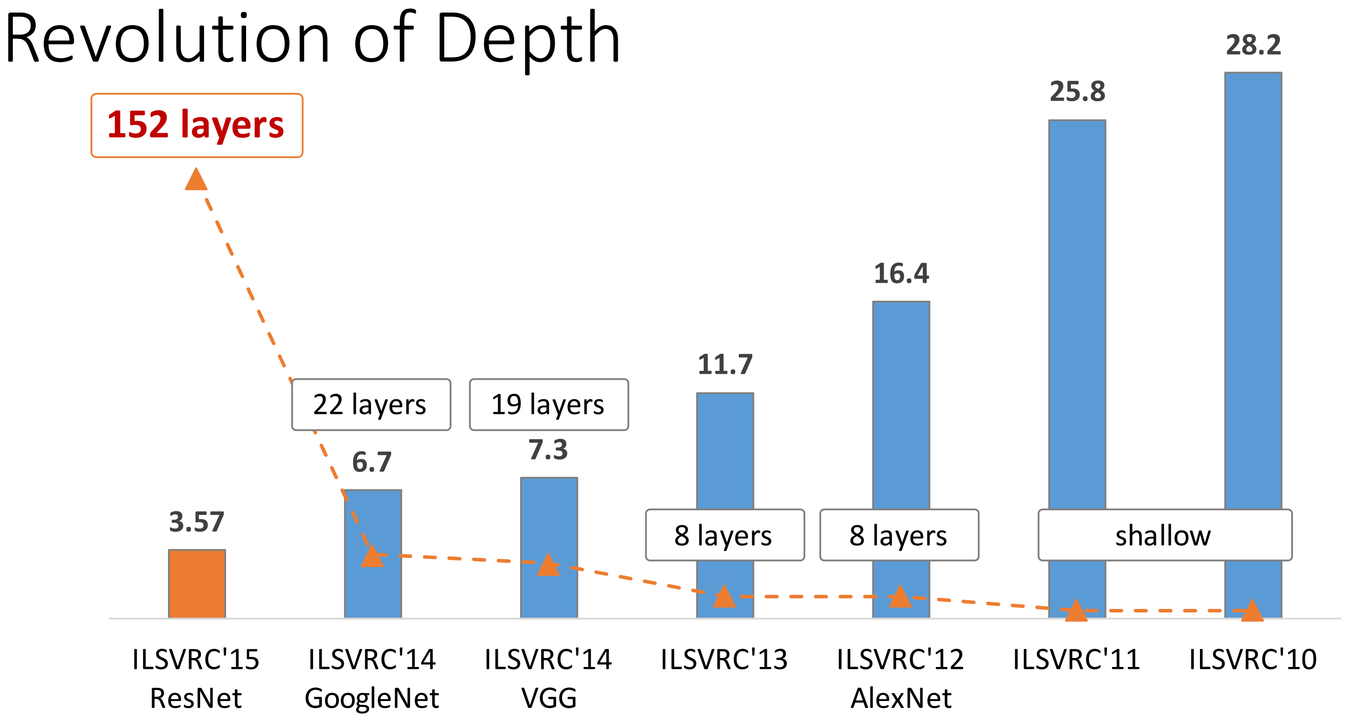
VGG, 19 layers
(ILSVRC 2014)

ResNet, 152 layers
(ILSVRC 2015)



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Residual Networks



ImageNet Classification top-5 error (%)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Highway Network (Srivastava et al., 2015)

- A plain feedforward neural network:

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}).$$

  - H is a typical affine transformation followed by a non-linear activation

- Highway network:

$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot C(\mathbf{x}, \mathbf{W_C}).$$

  - T is a “transform gate”
  - C is a “carry gate”
  - Often C = 1 – T for simplicity

@Schmidhubered

Vanishing / exploding Gradients

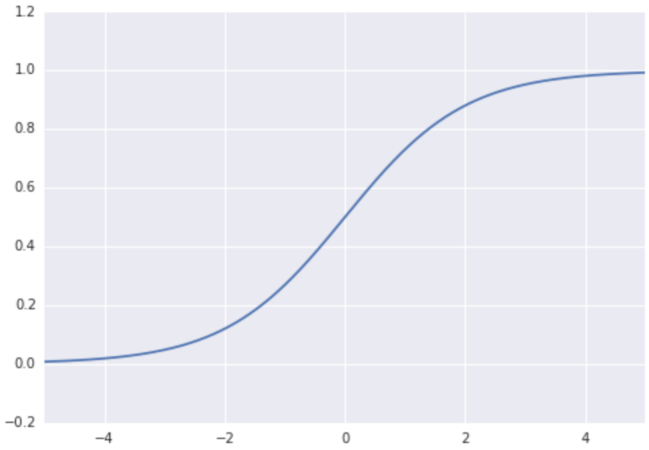
- Practical solutions w.r.t. numerical operations
  - Gradient Clipping: bound gradients by a max value
  - Gradient Normalization: renormalize gradients when they are above a fixed norm
  - Careful initialization, smaller learning rates
  - Avoid saturating nonlinearities (like tanh, sigmoid)
    - ReLU or hard-tanh instead

Sigmoid

- Often used for gates
- Pro: neuron-like, differentiable
- Con: gradients saturate to zero almost everywhere except x near zero => vanishing gradients
- Batch normalization helps

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

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$



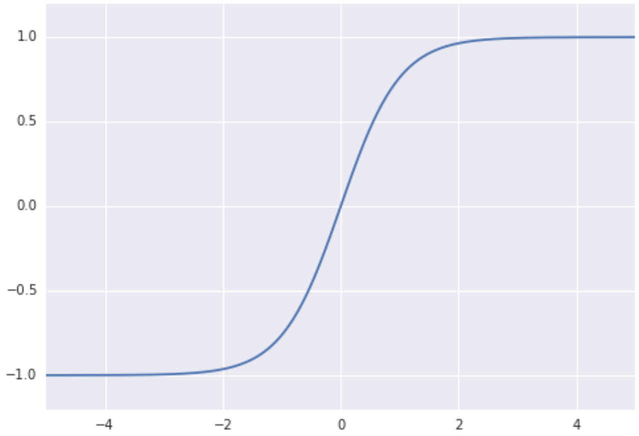
Tanh

- Often used for hidden states & cells in RNNs, LSTMs
- Pro: differentiable, often converges faster than sigmoid
- Con: gradients easily saturate to zero => vanishing gradients

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

$$\text{tanh'(x)} = 1 - \tanh^2(x)$$

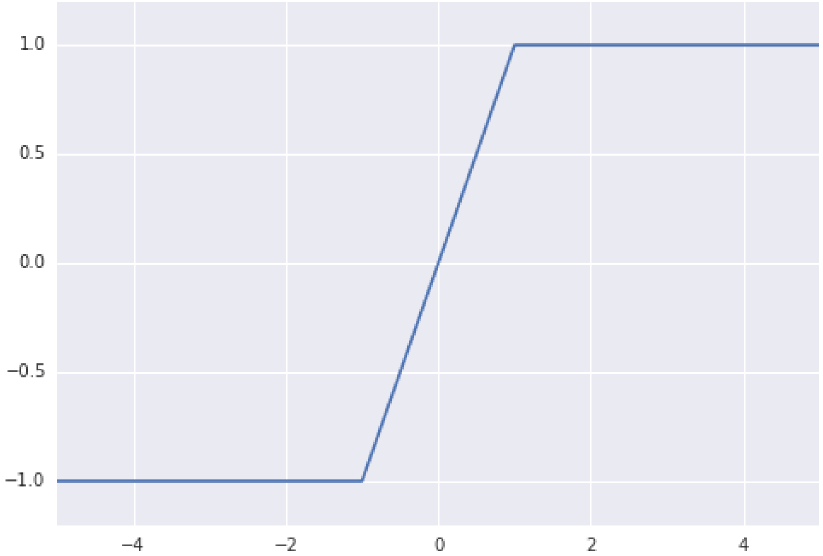
$$\tanh(x) = 2\sigma(2x) - 1$$



Hard Tanh

- Pro: computationally cheaper
- Con: saturates to zero easily, doesn't differentiate at 1, -1

$$\text{hardtanh}(t) = \begin{cases} -1 & t < -1 \\ t & -1 \le t \le 1 \\ 1 & t > 1 \end{cases}$$

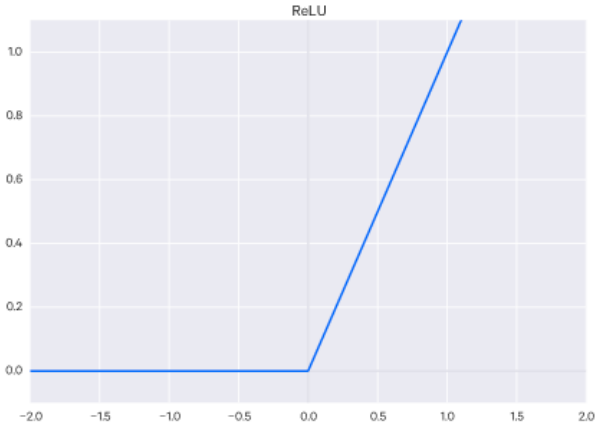


ReLU

- Pro: doesn’t saturate for x > 0, computationally cheaper, induces sparse NNs
- Con: non-differentiable at 0
- Used widely in deep NN, but not as much in RNNs
- We informally use subgradients:

$$\mathrm{ReLU}(x) = \max(0, x)$$

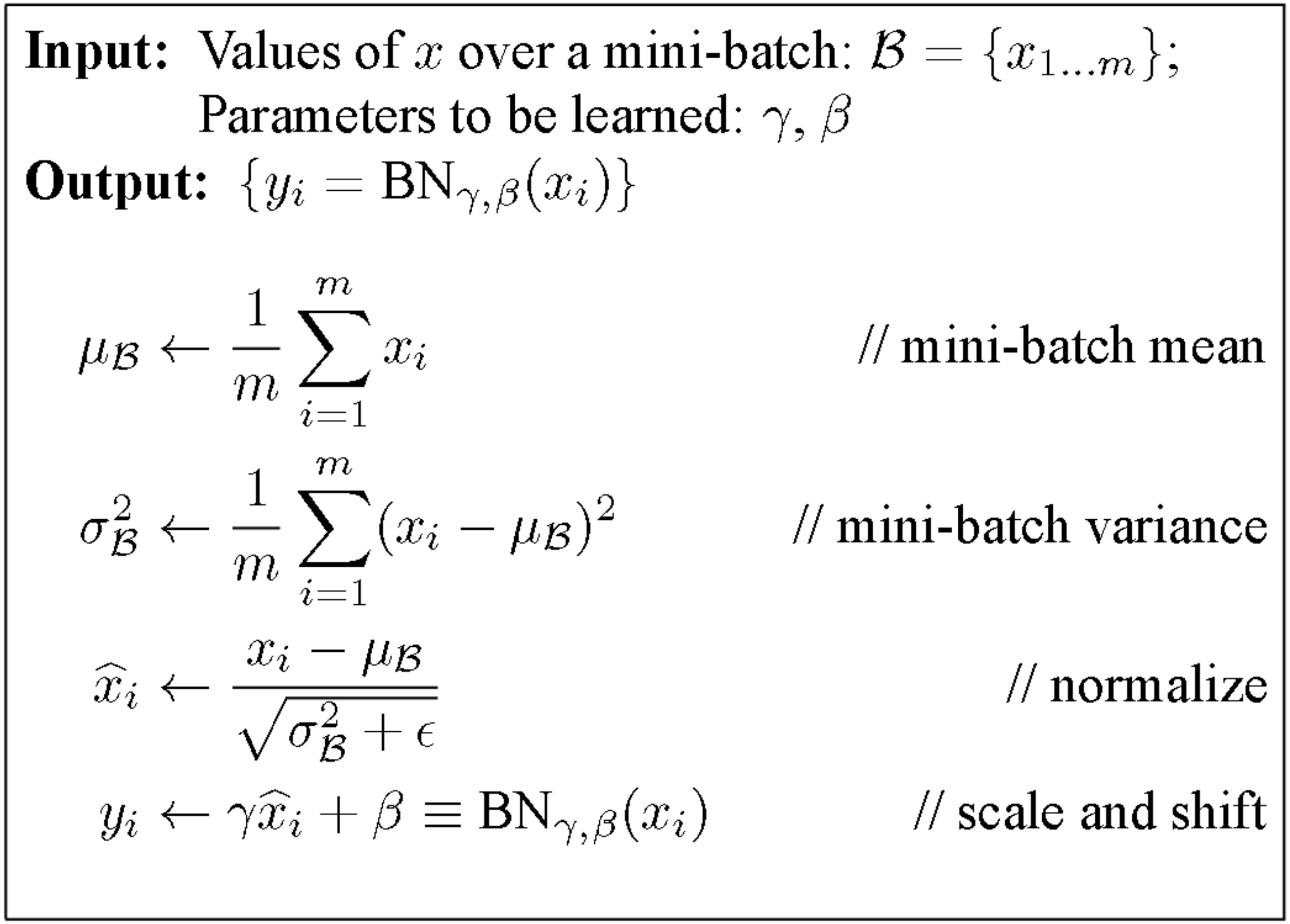
$$\frac{d\,\mathrm{ReLU}(x)}{dx} = \begin{cases} 1 & x > 0 \\ 0 & x < 0 \\ 1 \text{ or } 0 & o.w \end{cases}$$



Vanishing / exploding Gradients

- Practical solutions w.r.t. numerical operations
  - Gradient Clipping: bound gradients by a max value
  - Gradient Normalization: renormalize gradients when they are above a fixed norm
  - Careful initialization, smaller learning rates
  - Avoid saturating nonlinearities (like tanh, sigmoid)
    - ReLU or hard-tanh instead
  - Batch Normalization: add intermediate input normalization layers

Batch Normalization



```
Input: Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
       Parameters to be learned: $\gamma$, $\beta$
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

    $\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$                          // mini-batch mean
    $\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2$             // mini-batch variance
    $\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}$                      // normalize
    $y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$              // scale and shift
```

Regularization

- Regularization by objective term

$$\mathcal{L}(\theta) = \sum_{i=1}^{n} \max\{0, 1 - (\hat{y}_c - \hat{y}_{c'})\} + \lambda||\theta||^2$$

  – Modify loss with L1 or L2 norms

- Less depth, smaller hidden states, early stopping
- Dropout
  – Randomly delete parts of network during training
  – Each node (and its corresponding incoming and outgoing edges) dropped with a probability p
  – P is higher for internal nodes, lower for input nodes
  – The full network is used for testing
  – Faster training, better results
  – Vs. Bagging

Convergence of backprop

- Without non-linearity or hidden layers, learning is convex optimization
  - Gradient descent reaches **global minima**
- Multilayer neural nets (with nonlinearity) are **not convex**
  - Gradient descent gets stuck in local minima
  - Selecting number of hidden units and layers = fuzzy process
  - NNs have made a HUGE comeback in the last few years
    - Neural nets are back with a new name
      - Deep belief networks
      - Huge error reduction when trained with lots of data on GPUs

RECAP

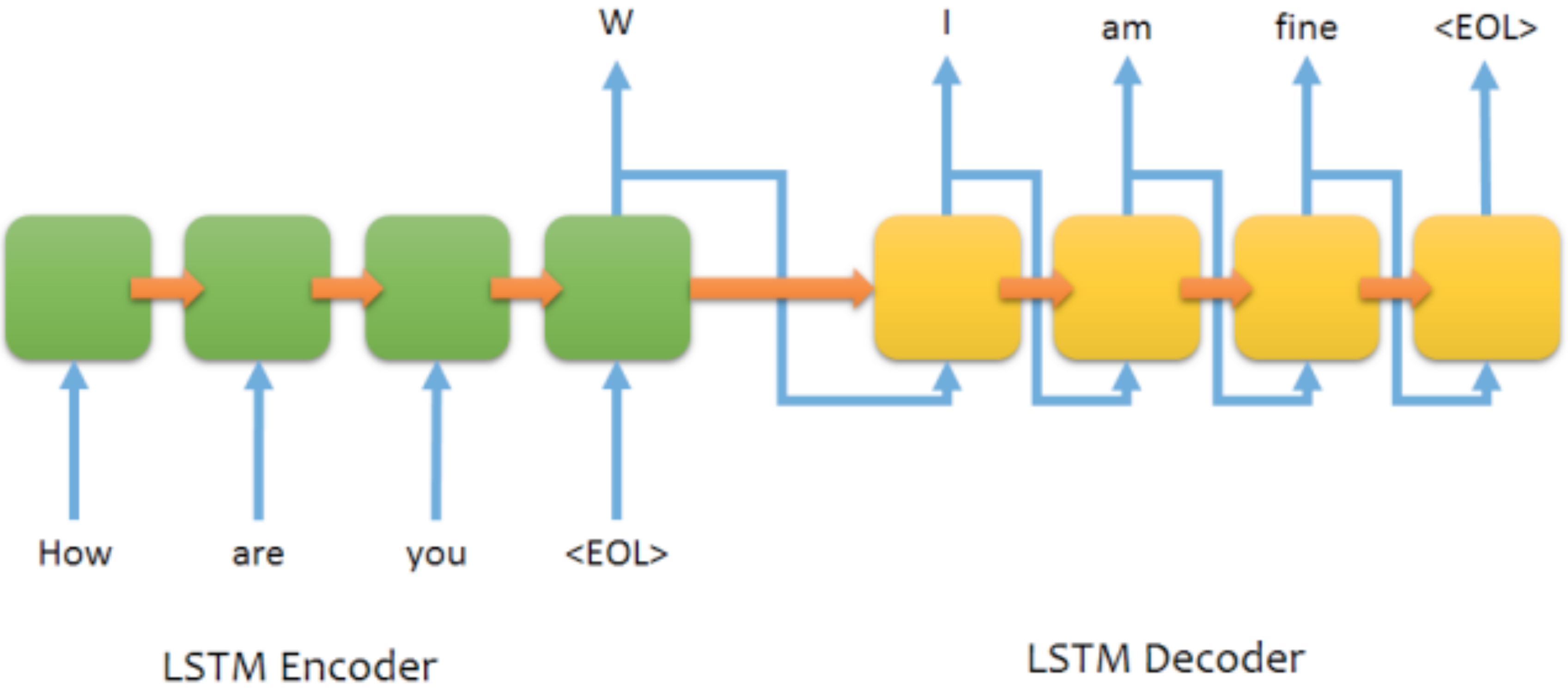
Vanishing / exploding Gradients

- Deep networks are hard to train
- Gradients go through multiple layers
- The multiplicative effect tends to lead to *exploding* or *vanishing* gradients
- Practical solutions w.r.t.
  - network architecture
  - numerical operations

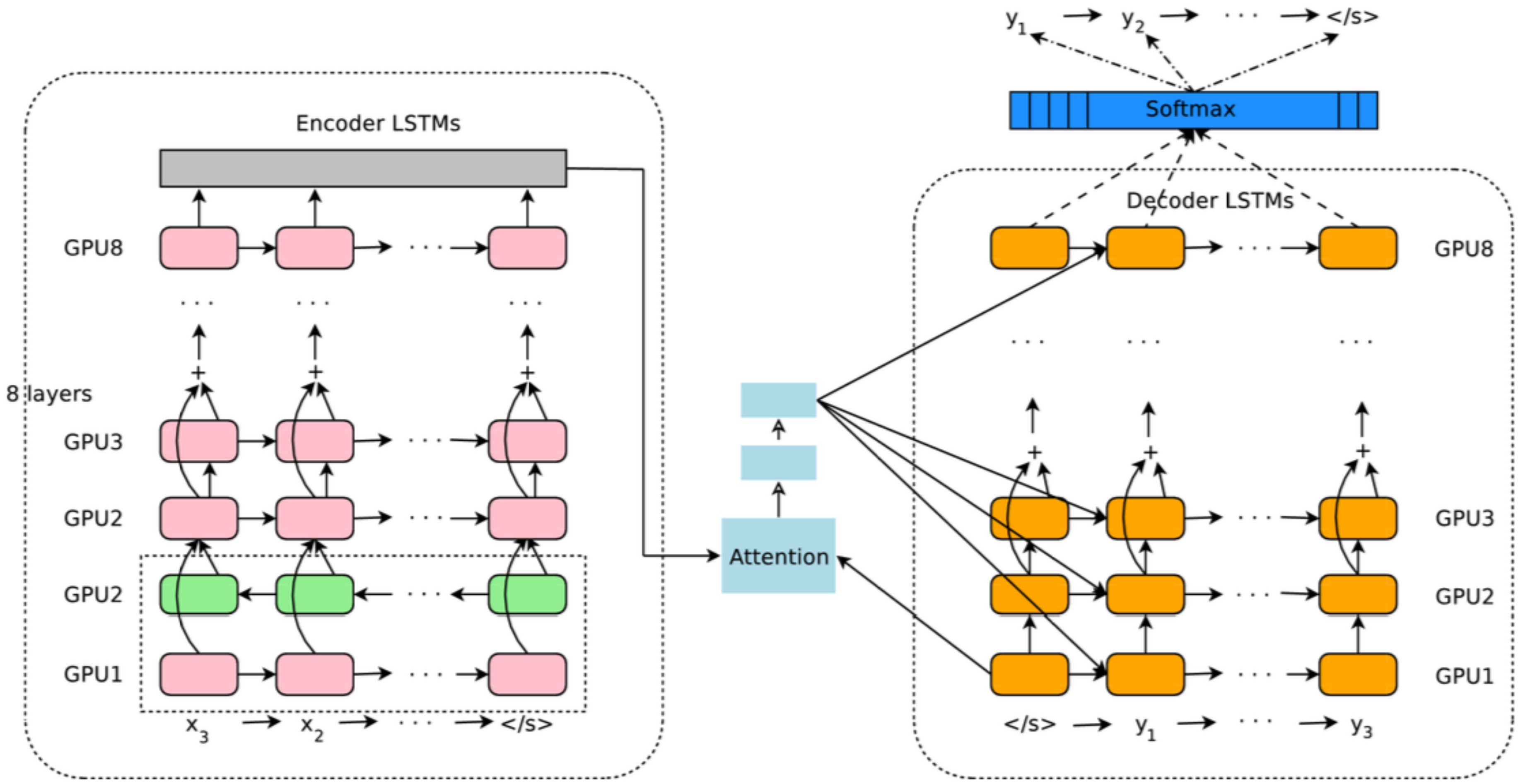
Vanishing / exploding Gradients

- Practical solutions w.r.t. network architecture
  - Add skip connections to reduce distance
    - Residual networks, highway networks, …
  - Add gates (and memory cells) to allow longer term memory
    - LSTMs, GRUs, memory networks, …

seq2seq (aka "encoder-decoder")

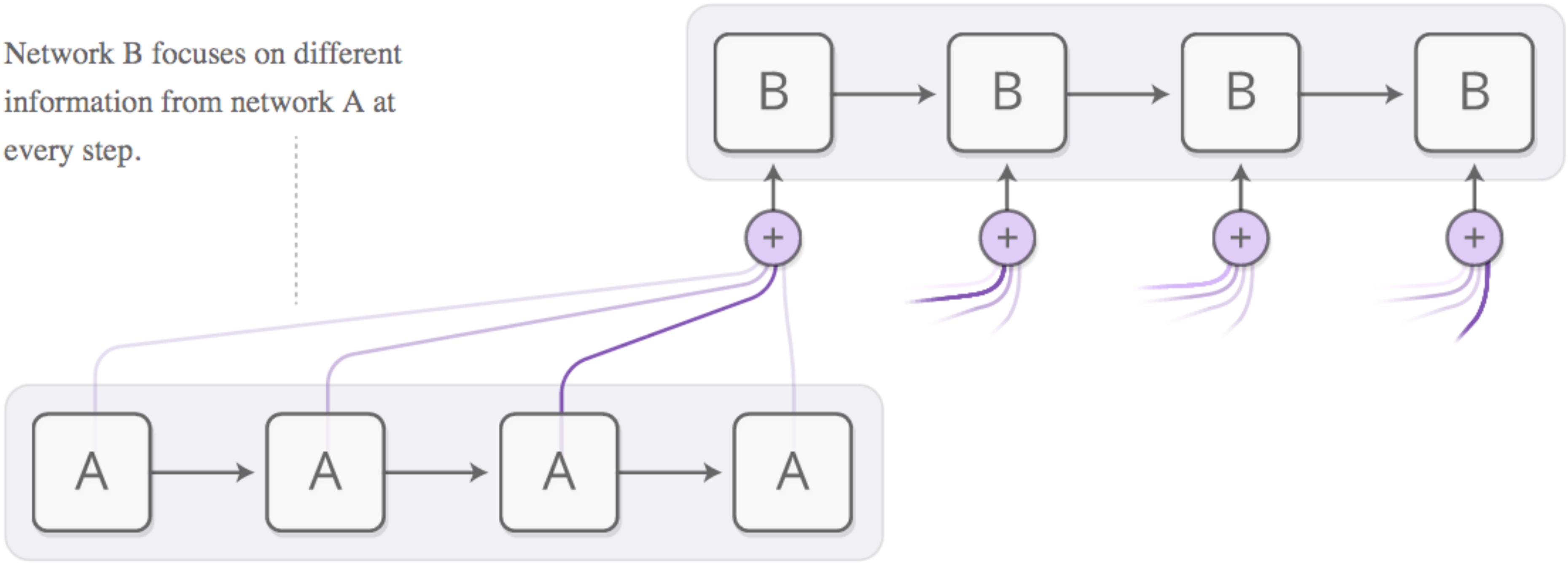


Google NMT (Oct 2016)



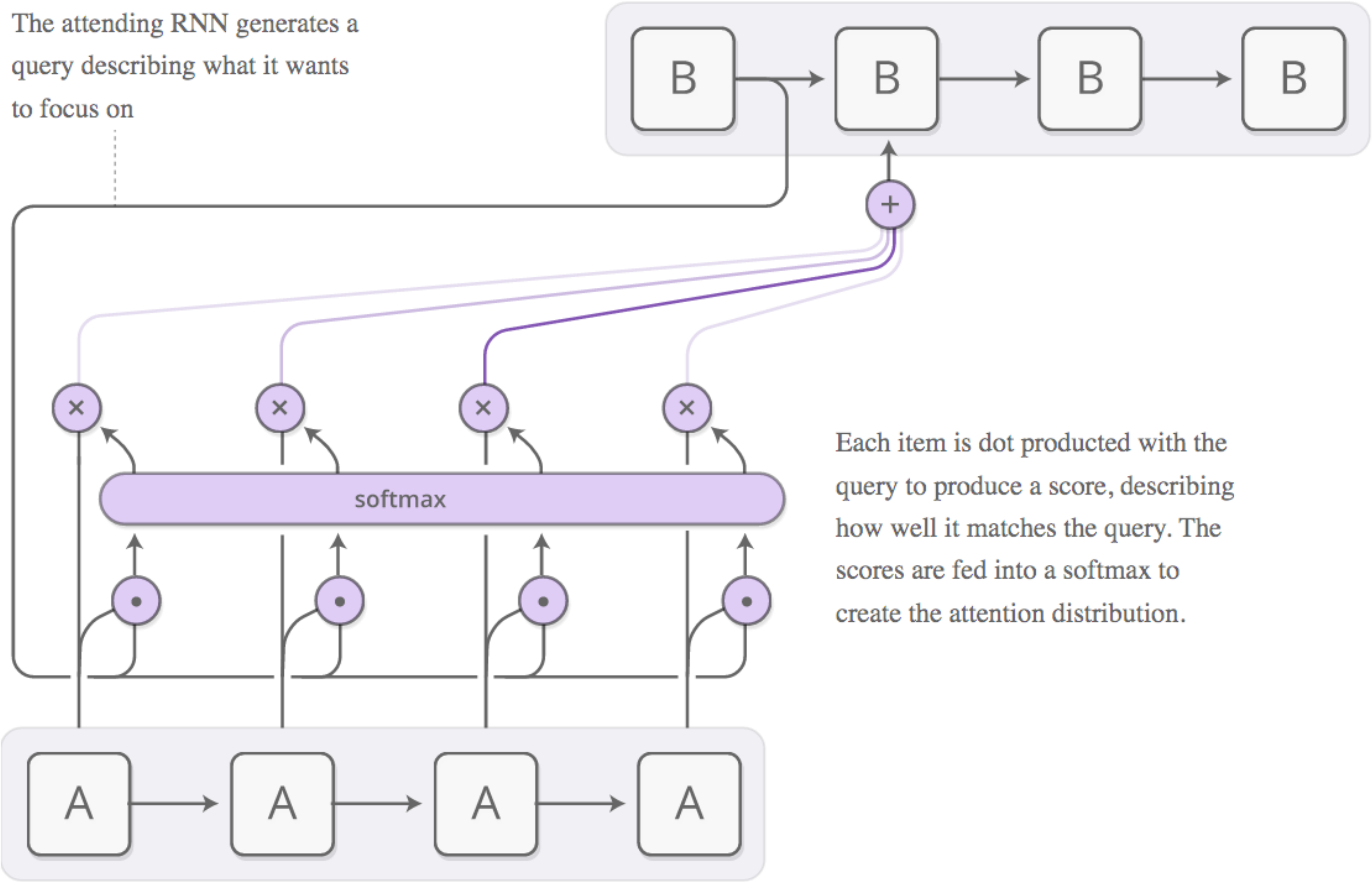
ATTENTION!

Seq-to-Seq with Attention



Network B focuses on different information from network A at every step.

Diagram from http://distill.pub/2016/augmented-rnns/



The attending RNN generates a query describing what it wants to focus on

Each item is dot producted with the query to produce a score, describing how well it matches the query. The scores are fed into a softmax to create the attention distribution.

Diagram from http://distill.pub/2016/augmented-rnns/

Trial: Hard Attention

- At each step generating the target word $\mathbf{s}_i^t$
- Compute the best alignment to the source word $\mathbf{s}_j^s$
- And incorporate the source word to generate the target word

$$w_{i+1}^t = \text{argmax}_w O(w, s_{i+1}^t, s_j^s)$$

- Contextual *hard* alignment. How?

$$z_j = \tanh([s_i^t, s_j^s]W + b)$$

$$j = \text{argmax}_j z_j$$

- Problem?

Encoder – Decoder Architecture

Sequence-to-Sequence

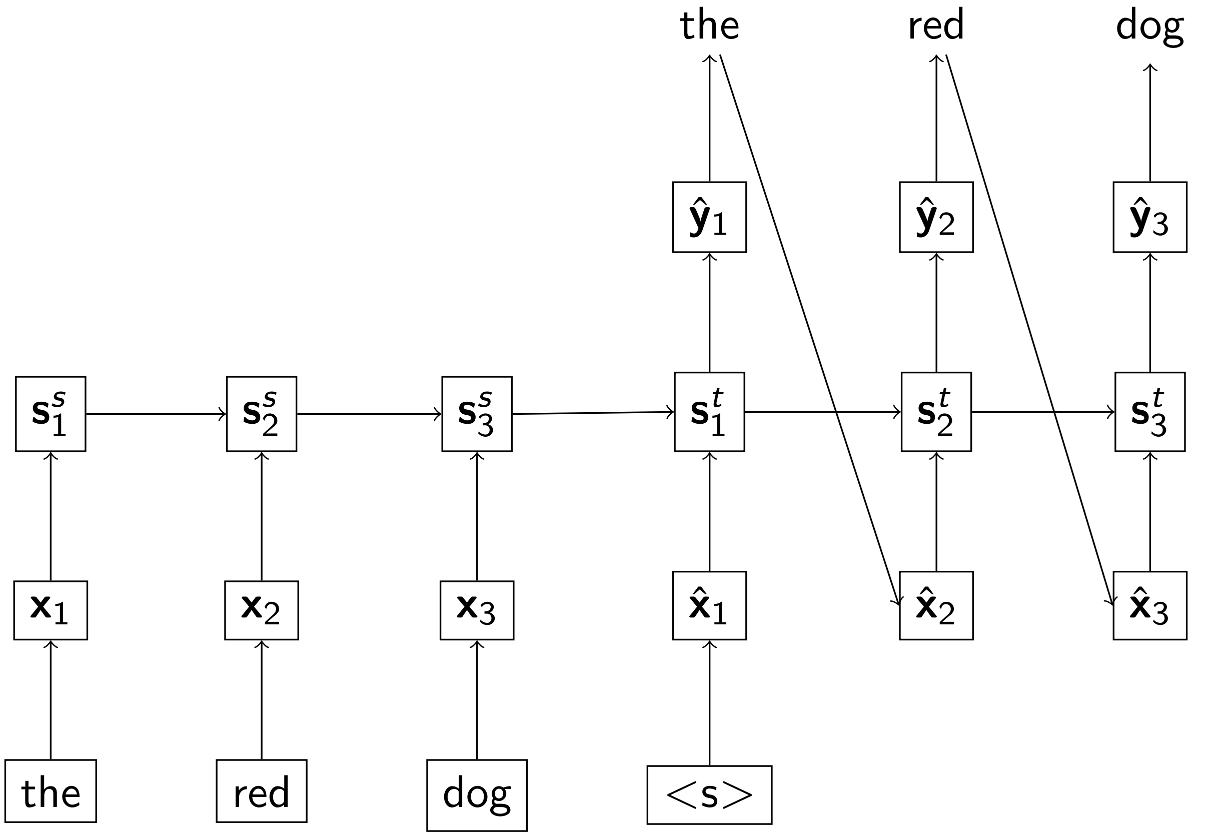


Diagram borrowed from Alex Rush

Attention: Soft Alignments

- At each step generating the target word $\mathbf{s}_i^t$
- Compute the attention $\mathbf{c}$ to the source sequence $\mathbf{s}^s$
- And incorporate the attention to generate the target word $w_{i+1}^t = \text{argmax}_w O(w, s_{i+1}^t, c)$

- Contextual *attention* as *soft* alignment. How?

$$z_j = \tanh([s_i^t, s_j^s]W + b)$$

$$\alpha = \text{softmax}(z)$$

$$c = \sum_j \alpha_j s_j^s$$

  - Step-1: compute the attention weights
  - Step-2: compute the attention vector as interpolation

Attention function parameterization

- Feedforward NNs

$$z_j = \tanh([s_i^t; s_j^s]W + b)$$

$$z_j = \tanh([s_i^t; s_j^s; s_i^t \circ s_j^s]W + b)$$

- Dot product

$$z_j = s_i^t \cdot s_j^s$$

- Cosine similarity

$$z_j = \frac{s_i^t \cdot s_j^s}{||s_i^t||||s_j^s||}$$

- Bi-linear models

$$z_j = {s_i^t}^T W s_j^s$$

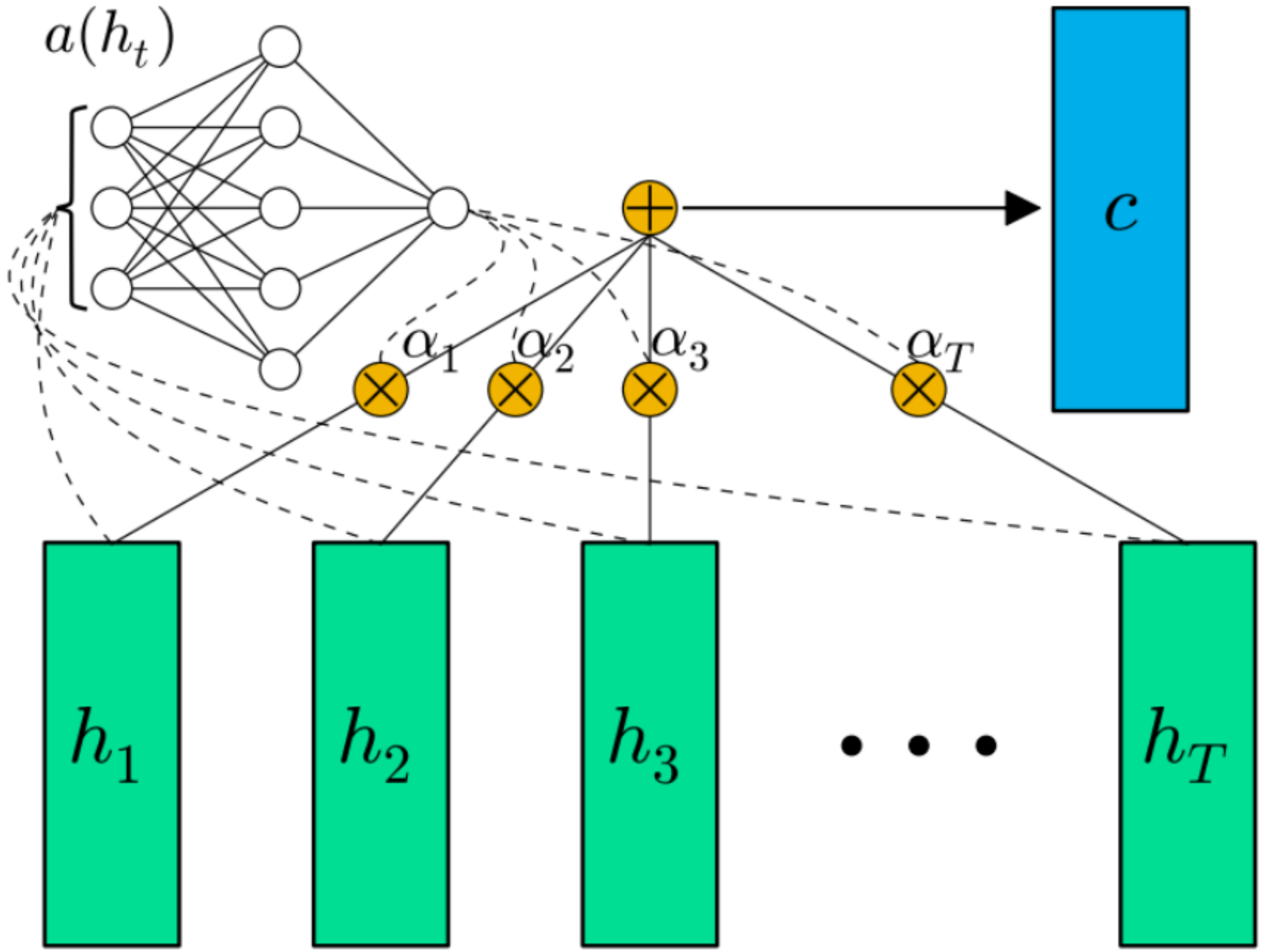


Figure 1: Schematic of our proposed "feed-forward" attention mechanism (cf. (Cho, 2015) Figure 1). Vectors in the hidden state sequence h_t are fed into the learnable function $a(h_t)$ to produce a probability vector α. The vector c is computed as a weighted average of h_t, with weighting given by α.

Learned Attention!

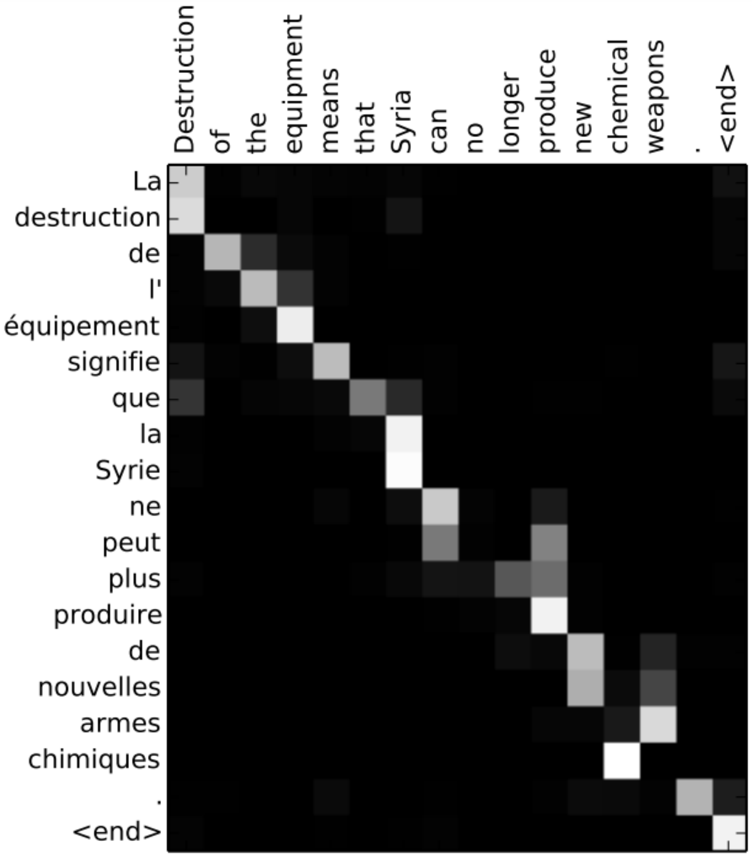


Diagram borrowed from Alex Rush

Qualitative results

Figure 2. Attention over time. As the model generates each word, its attention changes to reflect the relevant parts of the image. “soft” (top row) vs “hard” (bottom row) attention. (Note that both models generated the same captions in this example.)

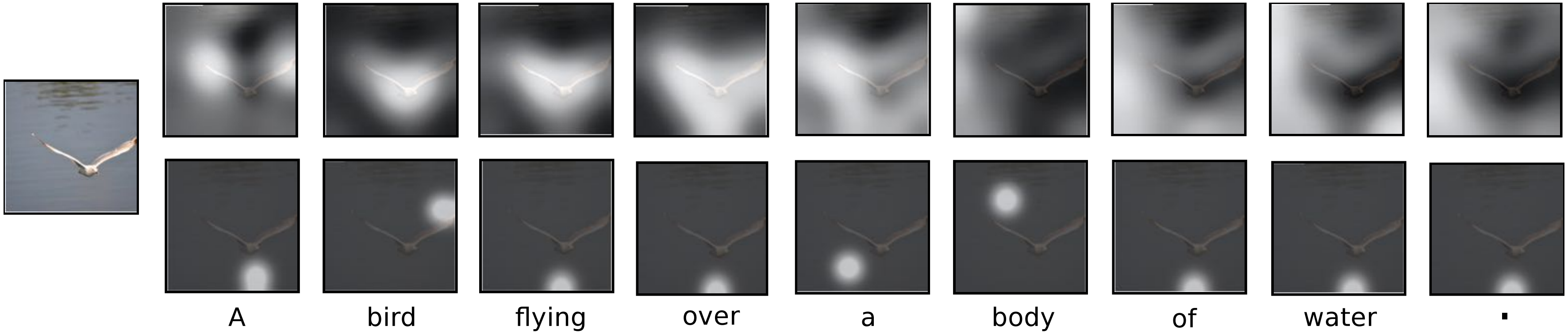
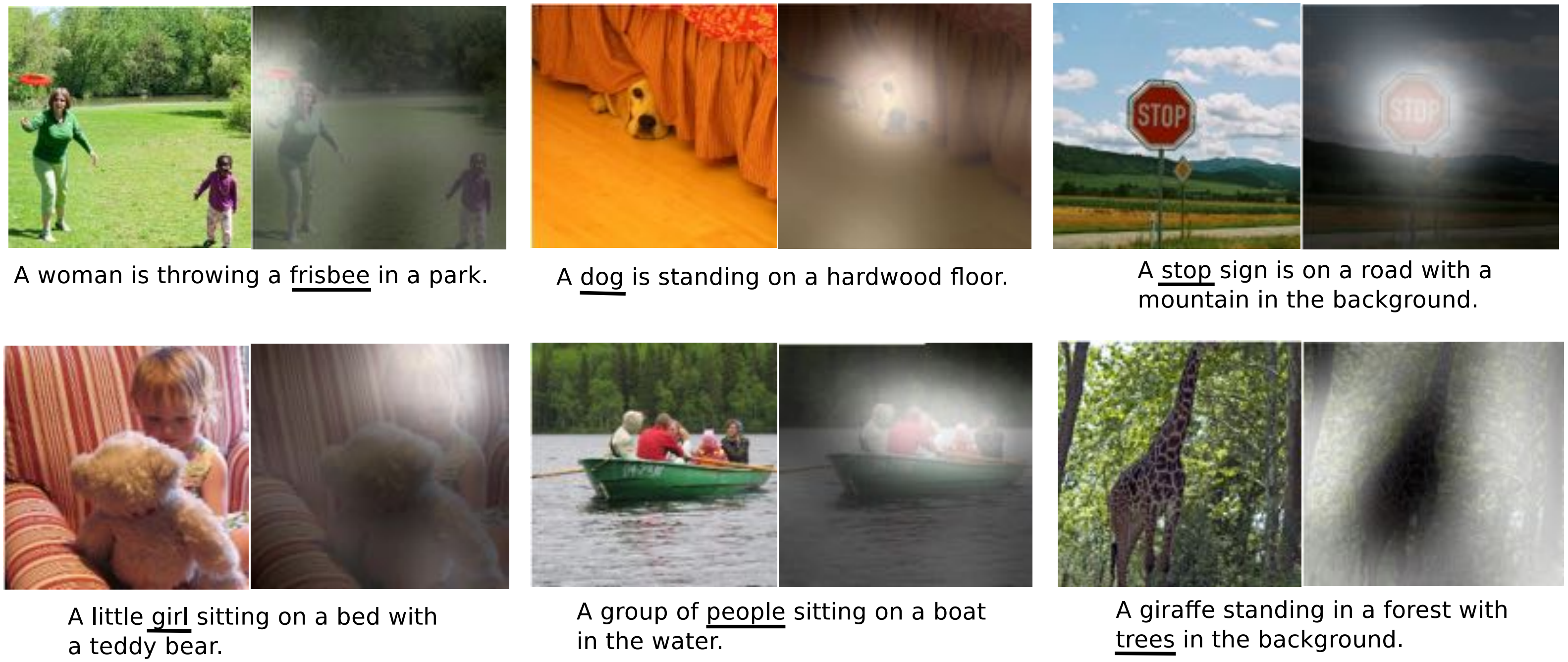


Figure 3. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)



A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

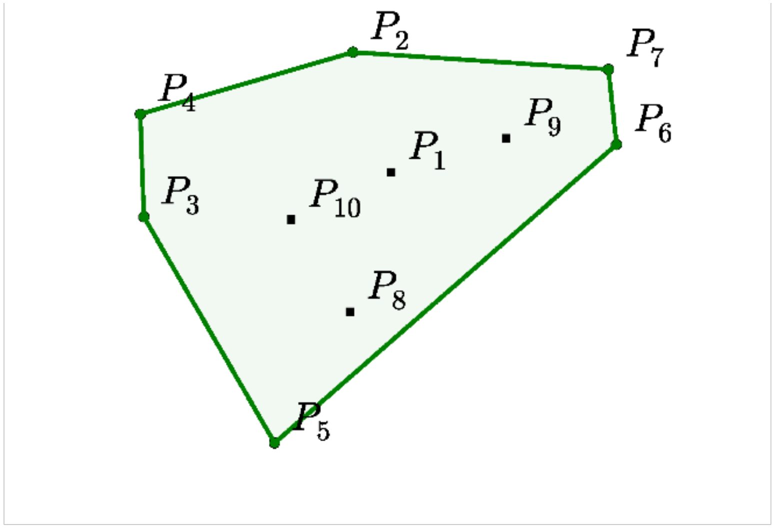
A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

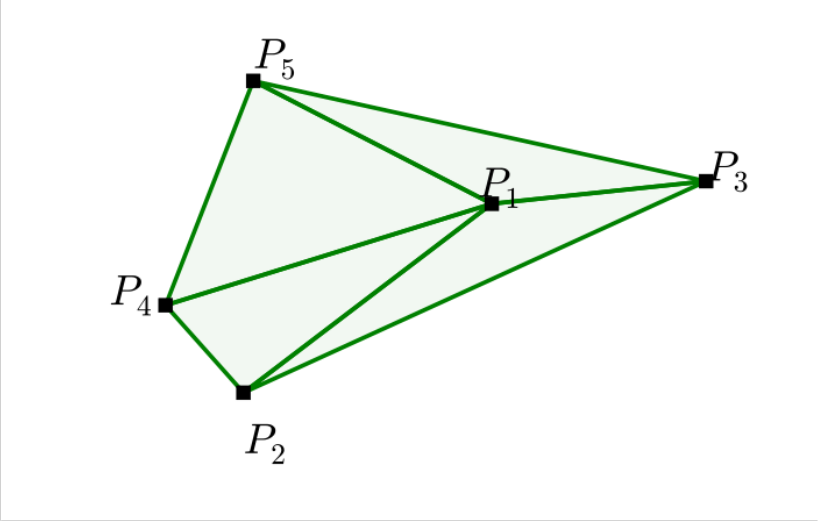
POINTER NETWORKS

Convex haul, Delaunay Triangulation, Traveling Salesman

Can we model these problems using seq-to-seq?



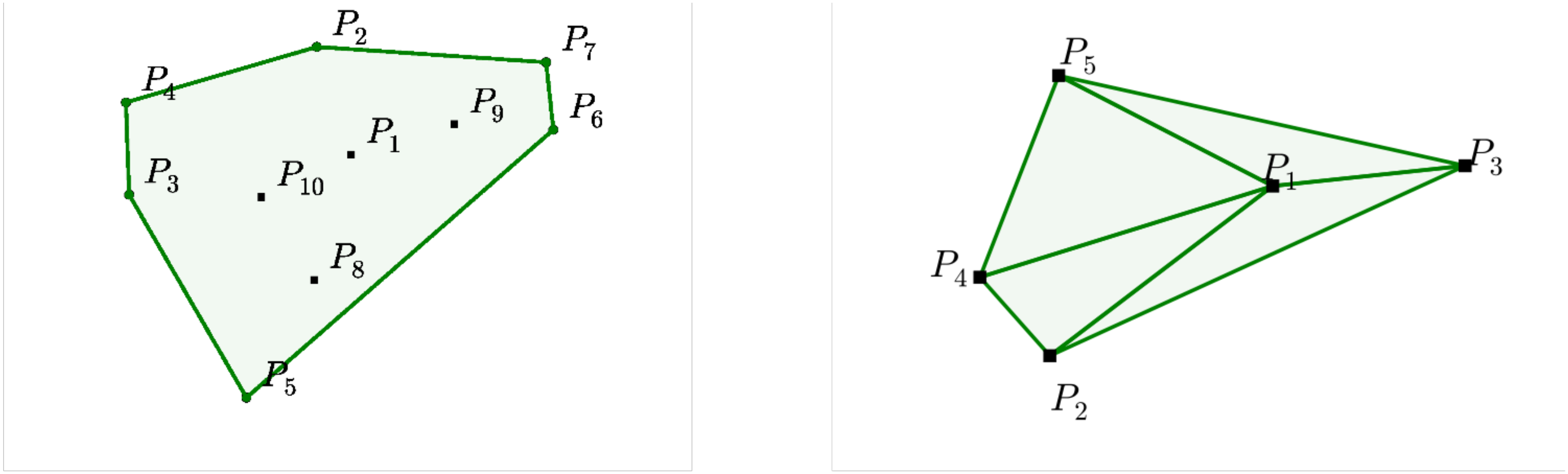
(a) Input $\mathcal{P} = \{P_1, \ldots, P_{10}\}$, and the output sequence $\mathcal{C}^{\mathcal{P}} = \{\Rightarrow, 2, 4, 3, 5, 6, 7, 2, \Leftarrow\}$ representing its convex hull.



(b) Input $\mathcal{P} = \{P_1, \ldots, P_5\}$, and the output $\mathcal{C}^{\mathcal{P}} = \{\Rightarrow, (1, 2, 4), (1, 4, 5), (1, 3, 5), (1, 2, 3), \Leftarrow\}$ representing its Delaunay Triangulation.

Pointer Networks! (Vinyals et al. 2015)

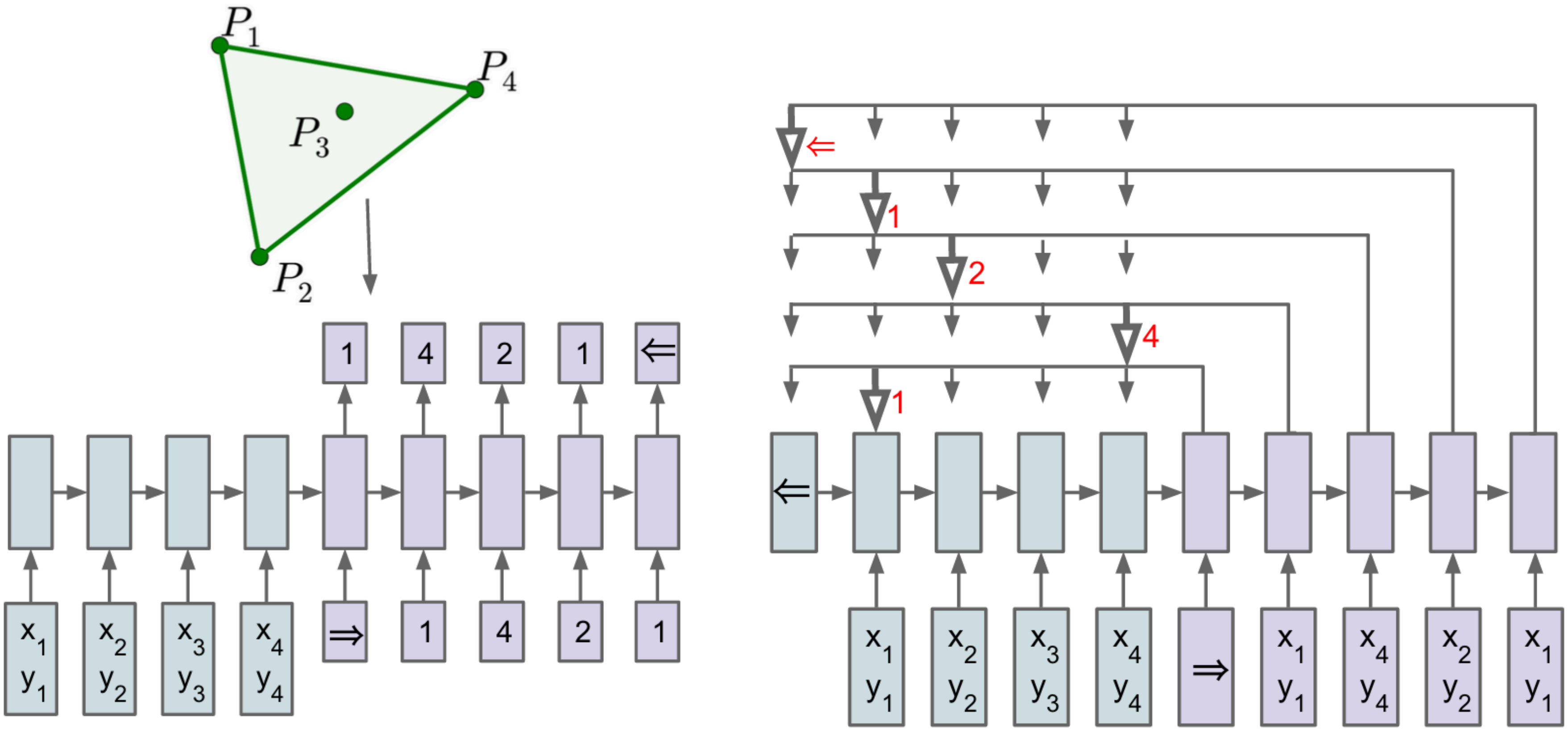
- NNs with attention: content-based attention to input
- Pointer networks: location-based attention to input



(a) Input $\mathcal{P} = \{P_1, \ldots, P_{10}\}$, and the output sequence $\mathcal{C}^{\mathcal{P}} = \{\Rightarrow, 2, 4, 3, 5, 6, 7, 2, \Leftarrow\}$ representing its convex hull.

(b) Input $\mathcal{P} = \{P_1, \ldots, P_5\}$, and the output $\mathcal{C}^{\mathcal{P}} = \{\Rightarrow, (1, 2, 4), (1, 4, 5), (1, 3, 5), (1, 2, 3), \Leftarrow\}$ representing its Delaunay Triangulation.

Pointer Networks

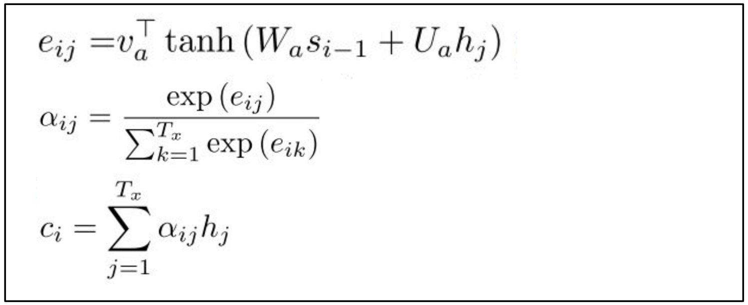


(a) Sequence-to-Sequence

(b) Ptr-Net

Pointer Networks

Attention Mechanism vs Pointer Networks

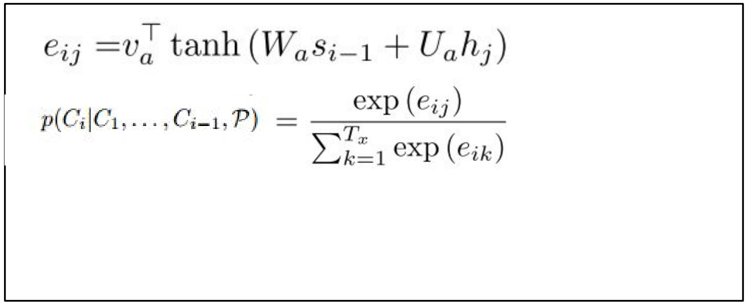


$$e_{ij} = v_a^\top \tanh\left(W_a s_{i-1} + U_a h_j\right)$$

$$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{k=1}^{T_x} \exp\left(e_{ik}\right)}$$

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

Attention mechanism



$$e_{ij} = v_a^\top \tanh\left(W_a s_{i-1} + U_a h_j\right)$$

$$p(C_i | C_1, \ldots, C_{i-1}, \mathcal{P}) = \frac{\exp\left(e_{ij}\right)}{\sum_{k=1}^{T_x} \exp\left(e_{ik}\right)}$$

Ptr-Net

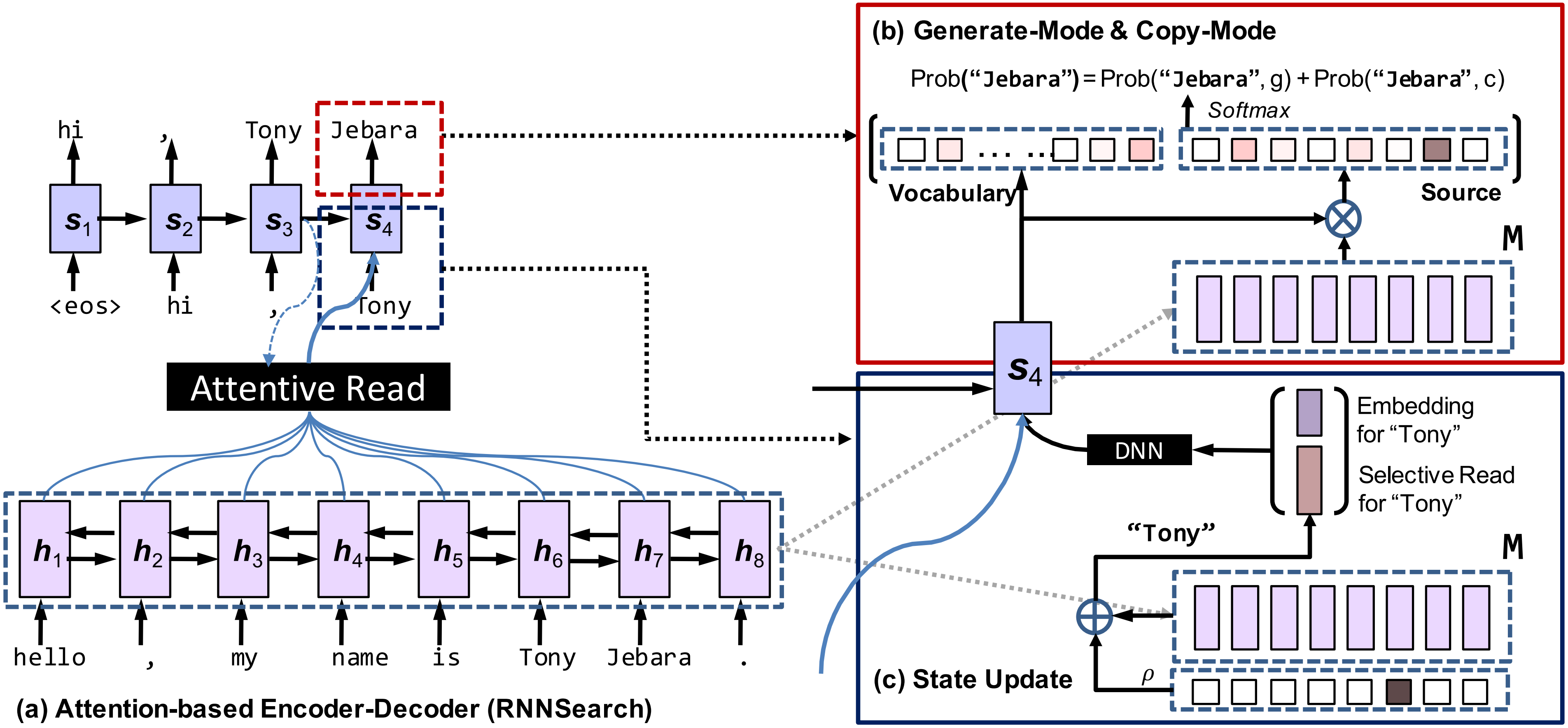
Softmax normalizes the vector e*ij* to be an output distribution over the dictionary of inputs

Diagram borrowed from Keon Kim

CopyNet (Gu et al. 2016)

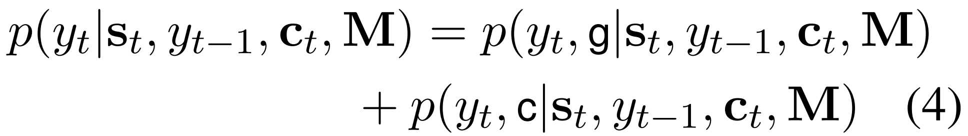
- Conversation
  - I: Hello Jack, my name is Chandralekha
  - R: Nice to meet you, Chandralekha
  - I: This new guy doesn't perform exactly as expected.
  - R: what do you mean by "doesn't perform exactly as expected?"
- Translation

CopyNet (Gu et al. 2016)

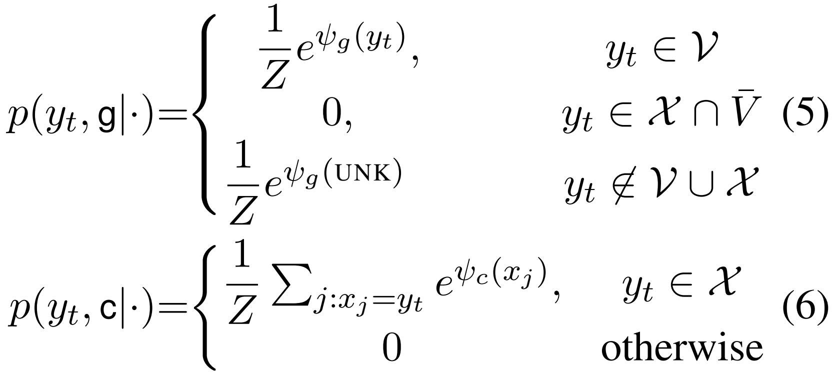


CopyNet (Gu et al. 2016)

- Key idea: interpolation between generation model & copy model

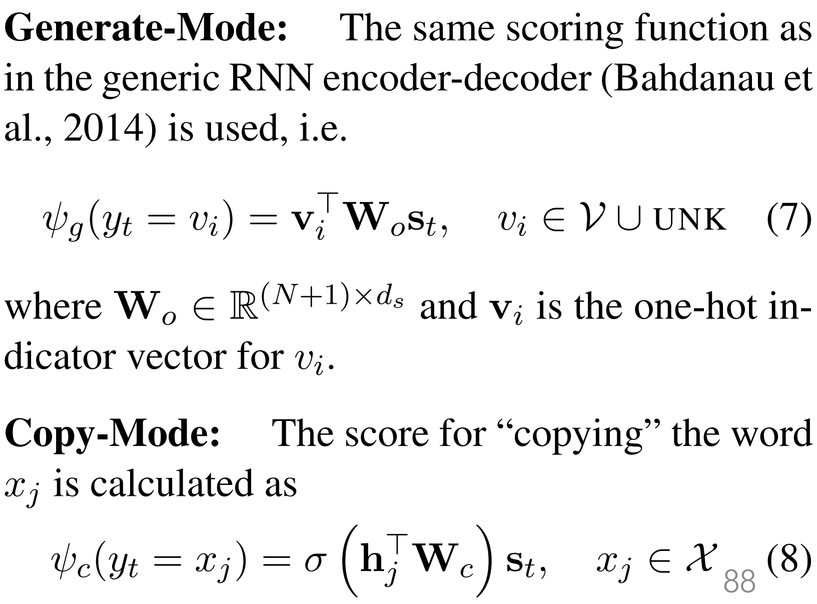


$$p(y_t|\mathbf{s}_t, y_{t-1}, \mathbf{c}_t, \mathbf{M}) = p(y_t, \mathsf{g}|\mathbf{s}_t, y_{t-1}, \mathbf{c}_t, \mathbf{M}) + p(y_t, \mathsf{c}|\mathbf{s}_t, y_{t-1}, \mathbf{c}_t, \mathbf{M}) \quad (4)$$



$$p(y_t, \mathsf{g}|\cdot)=\begin{cases} \frac{1}{Z}e^{\psi_g(y_t)}, & y_t \in \mathcal{V} \\ 0, & y_t \in \mathcal{X} \cap \bar{V} \\ \frac{1}{Z}e^{\psi_g(\text{UNK})} & y_t \notin \mathcal{V} \cup \mathcal{X} \end{cases} \quad (5)$$

$$p(y_t, \mathsf{c}|\cdot)=\begin{cases} \frac{1}{Z}\sum_{j:x_j=y_t} e^{\psi_c(x_j)}, & y_t \in \mathcal{X} \\ 0 & \text{otherwise} \end{cases} \quad (6)$$



Generate-Mode: The same scoring function as in the generic RNN encoder-decoder (Bahdanau et al., 2014) is used, i.e.

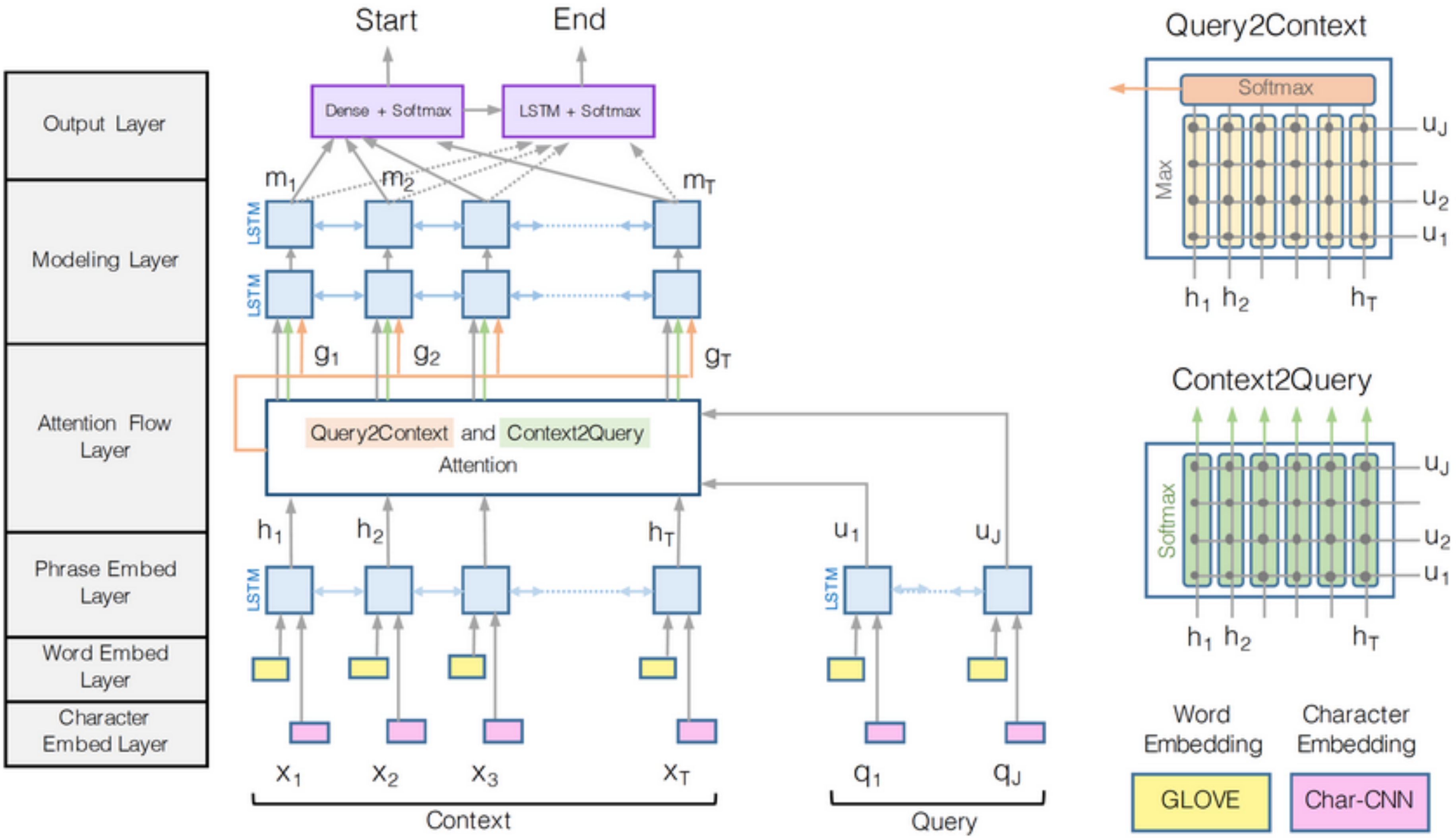
$$\psi_g(y_t = v_i) = \mathbf{v}_i^\top \mathbf{W}_o \mathbf{s}_t, \quad v_i \in \mathcal{V} \cup \text{UNK} \quad (7)$$

where $\mathbf{W}_o \in \mathbb{R}^{(N+1)\times d_s}$ and $\mathbf{v}_i$ is the one-hot indicator vector for v_i.

Copy-Mode: The score for "copying" the word x_j is calculated as

$$\psi_c(y_t = x_j) = \sigma\left(\mathbf{h}_j^\top \mathbf{W}_c\right) \mathbf{s}_t, \quad x_j \in \mathcal{X} \quad (8)$$

BiDAF


Start
End
Output Layer
Modeling Layer
Attention Flow Layer
Phrase Embed Layer
Word Embed Layer
Character Embed Layer
Dense + Softmax
LSTM + Softmax
m_1
m_2
m_T
LSTM
LSTM
g_1
g_2
g_T
Query2Context and Context2Query
Attention
h_1
h_2
h_T
u_1
u_J
LSTM
LSTM
x_1
x_2
x_3
x_T
q_1
q_J
Context
Query
Query2Context
Softmax
Max
u_J
u_2
u_1
h_1 h_2 h_T
Context2Query
Softmax
u_J
u_2
u_1
h_1 h_2 h_T
Word Embedding
Character Embedding
GLOVE
Char-CNN

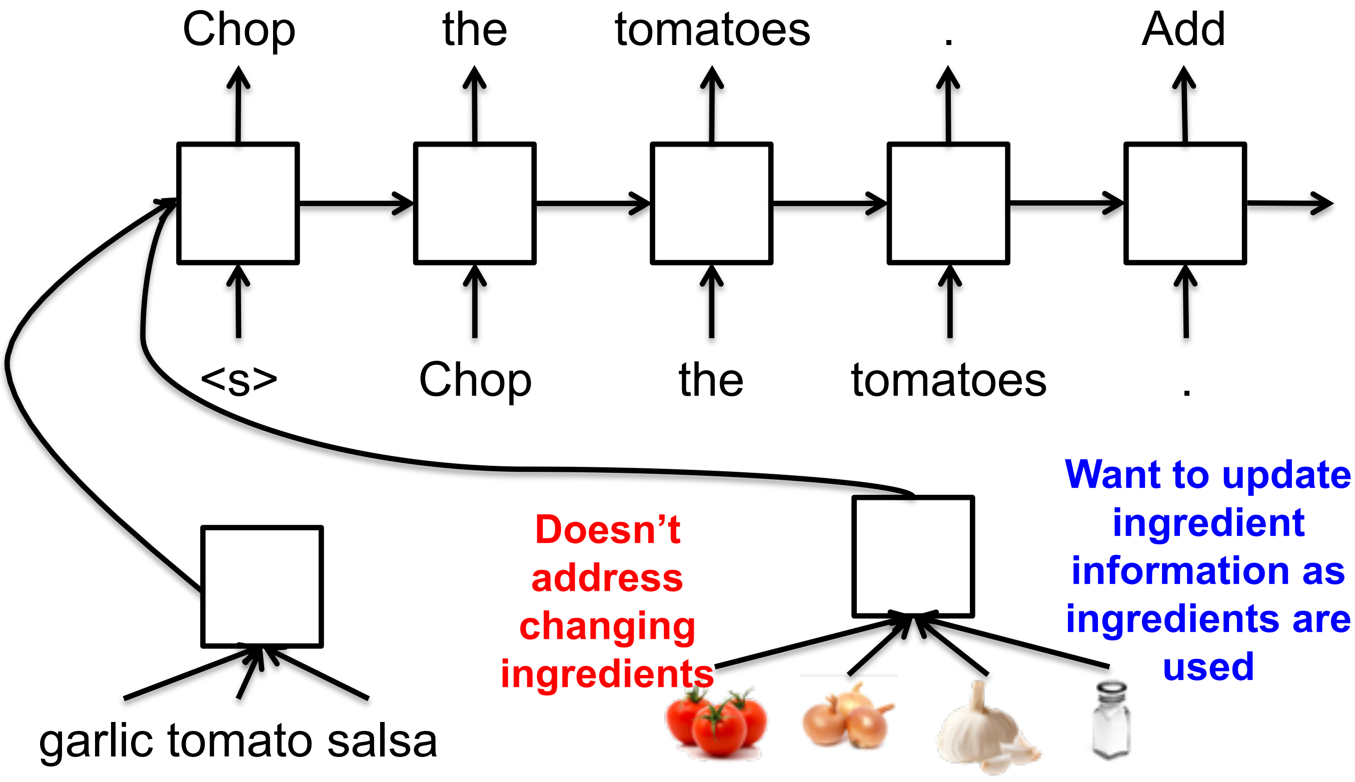
NEURAL CHECK LIST

Neural Checklist Models

(Kiddon et al., 2016)

- What can we do with gating & attention?

Encoder--Decoder Architecture



Chop the tomatoes . Add

<s> Chop the tomatoes .

garlic tomato salsa

Doesn't address changing ingredients

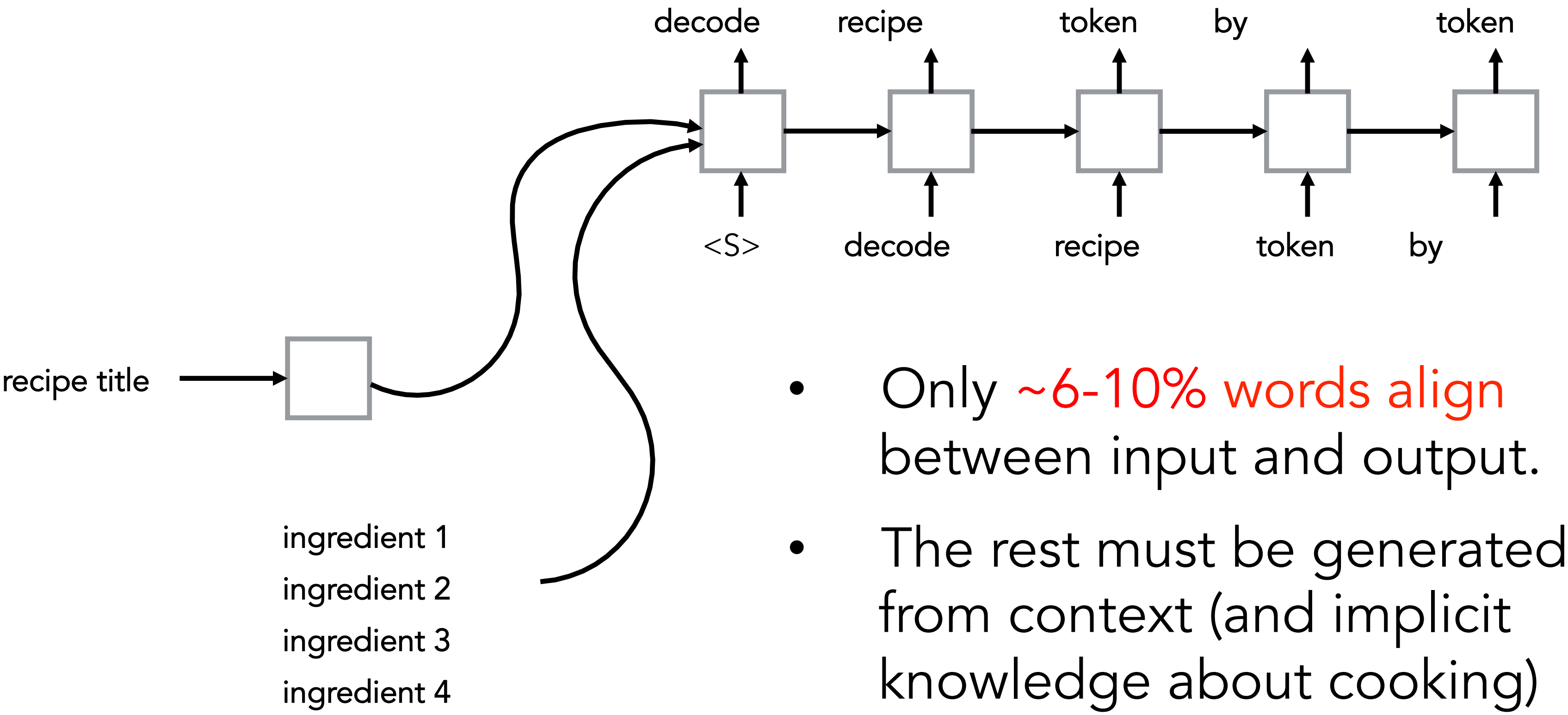
Want to update ingredient information as ingredients are used

Encode title - decode recipe

sausage sandwiches

→ Cut each sandwich in halves.
Sandwiches with sandwiches.
Sandwiches, sandwiches, Sandwiches, sandwiches, sandwiches
sandwiches, sandwiches, sandwiches, sandwiches, sandwiches, sandwiches, or sandwiches or triangles, a griddle, each sandwich.
Top each with a slice of cheese, tomato, and cheese.
Top with remaining cheese mixture.
Top with remaining cheese.
Broil until tops are bubbly and cheese is melted, about 5 minutes.

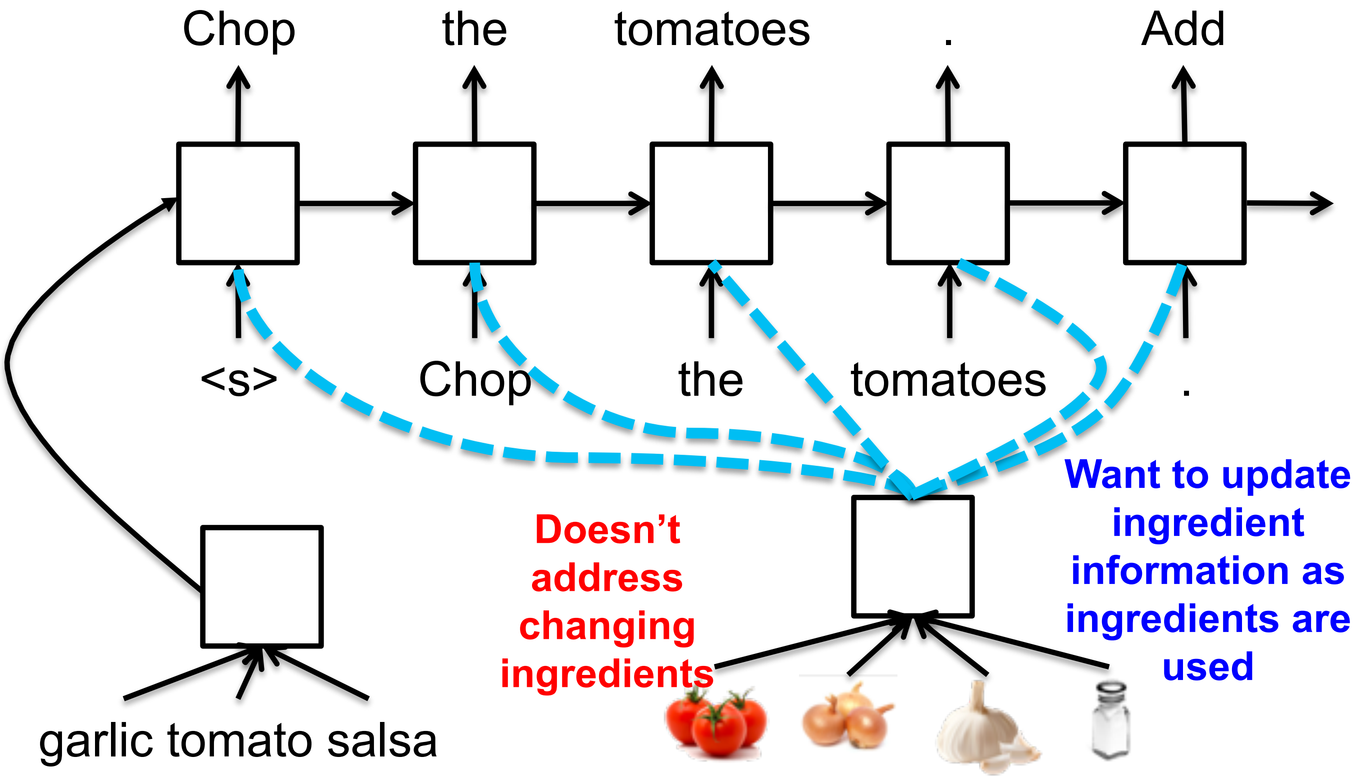
Recipe generation **vs** machine translation



- Only ~6-10% words align between input and output.
- The rest must be generated from context (and implicit knowledge about cooking)
- Contextual switch between two different input sources

Two input sources

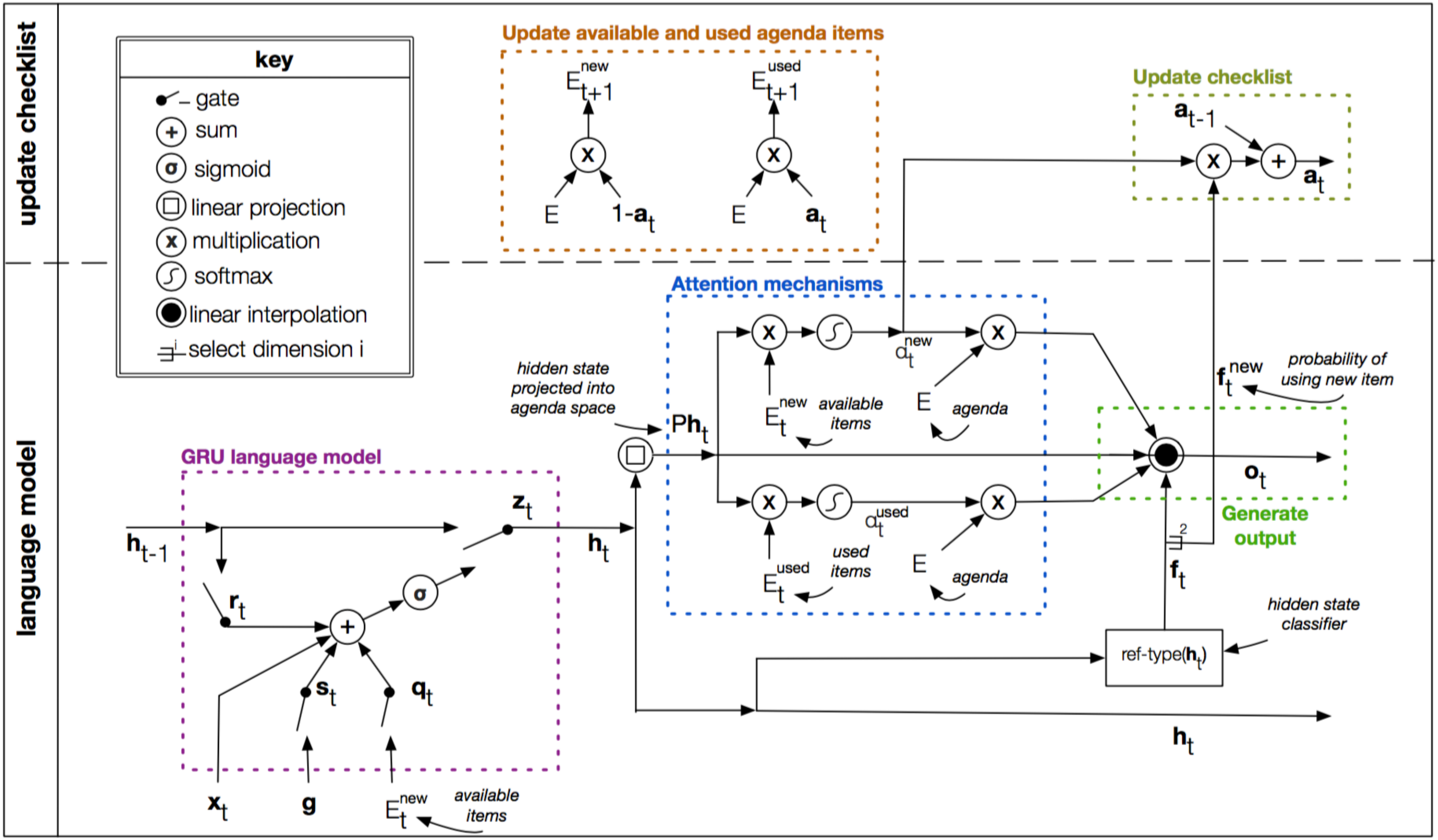
Encoder--Decoder with Attention



Doesn't address changing ingredients

Want to update ingredient information as ingredients are used

Neural checklist model

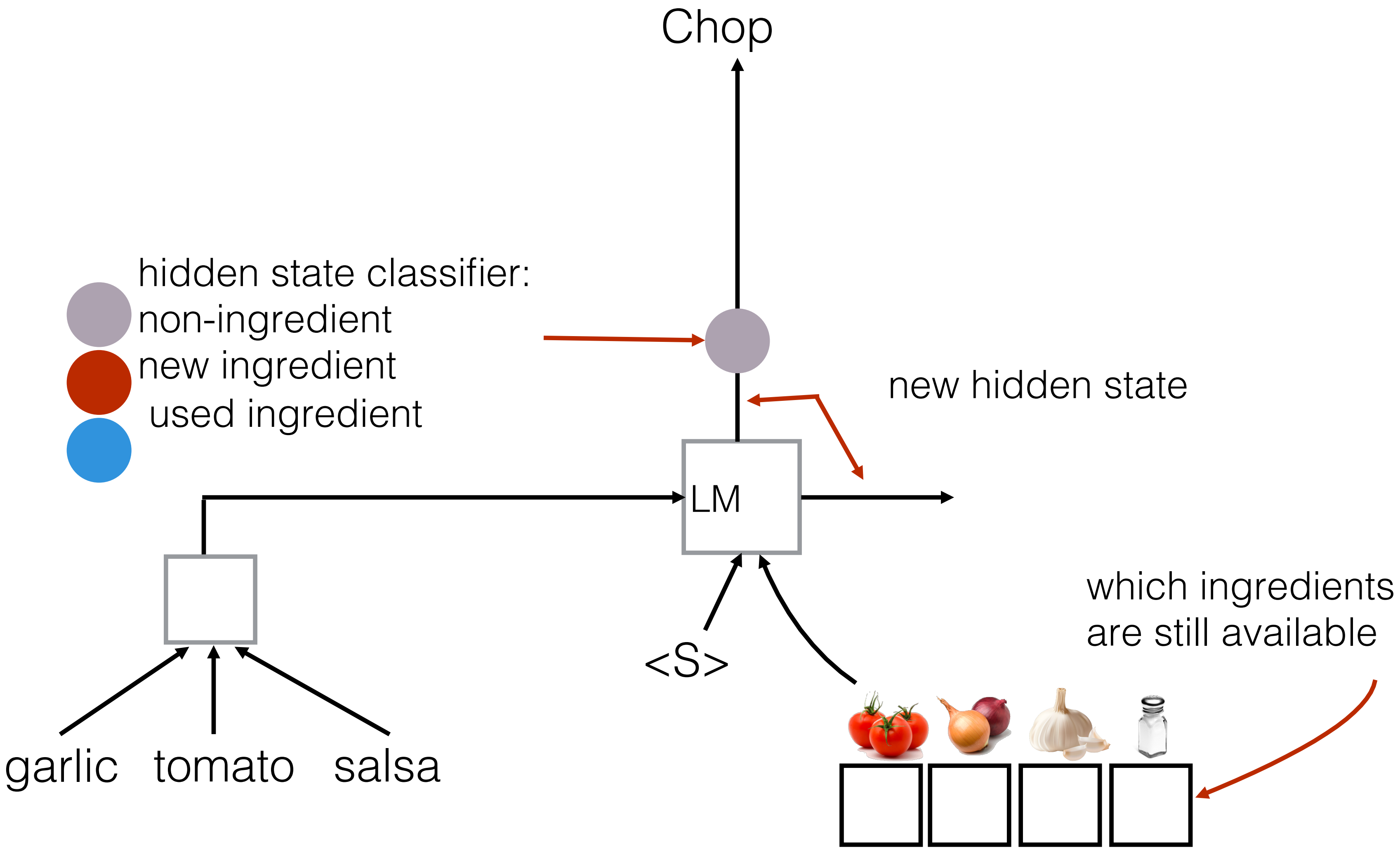


Let’s make salsa!

Garlic tomato salsa

tomatoes
onions
garlic
salt

Neural checklist model



Chop

hidden state classifier:

● non-ingredient

● new ingredient

● used ingredient

new hidden state

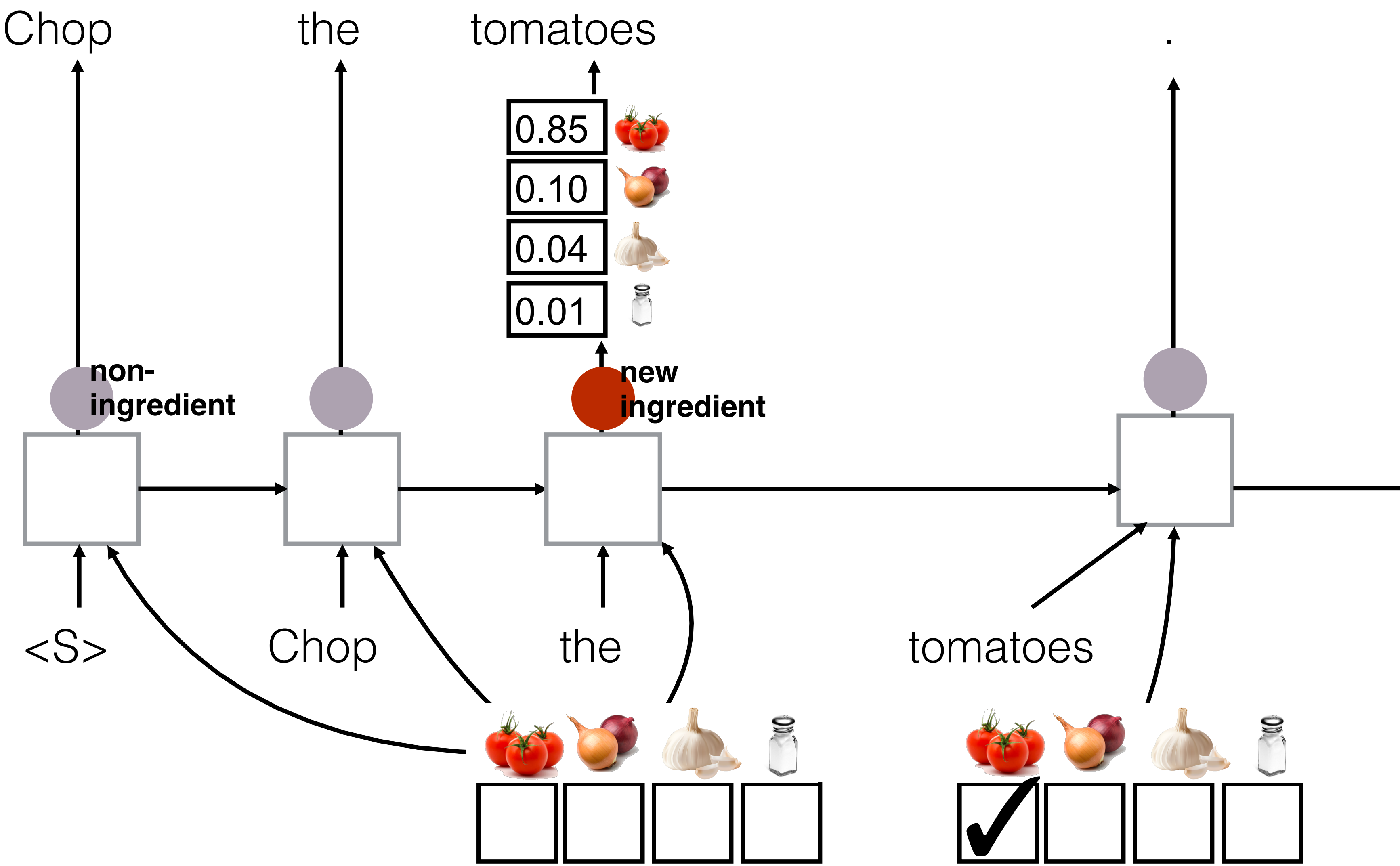
LM

<S>

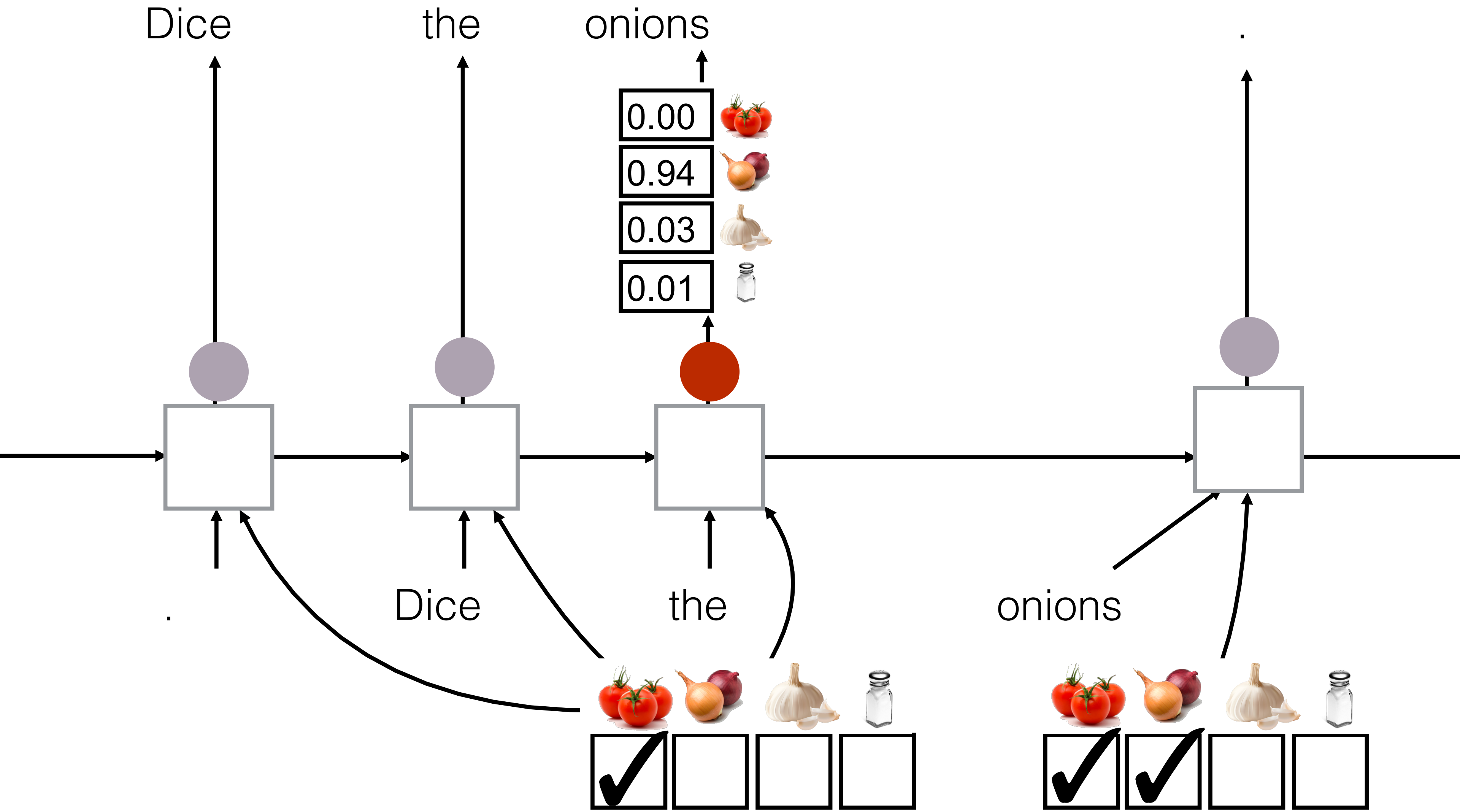
garlic tomato salsa

which ingredients are still available

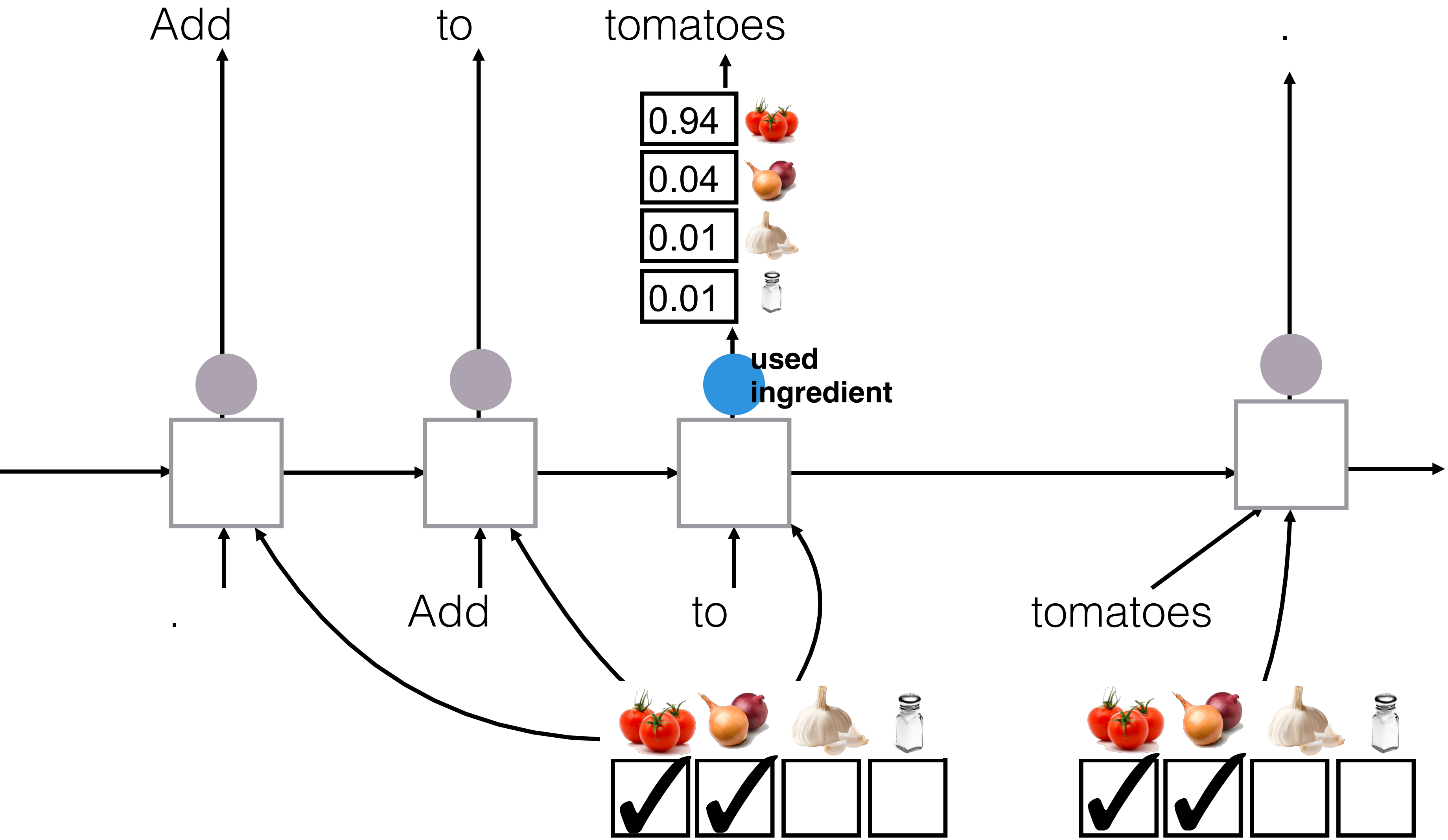
Neural checklist model



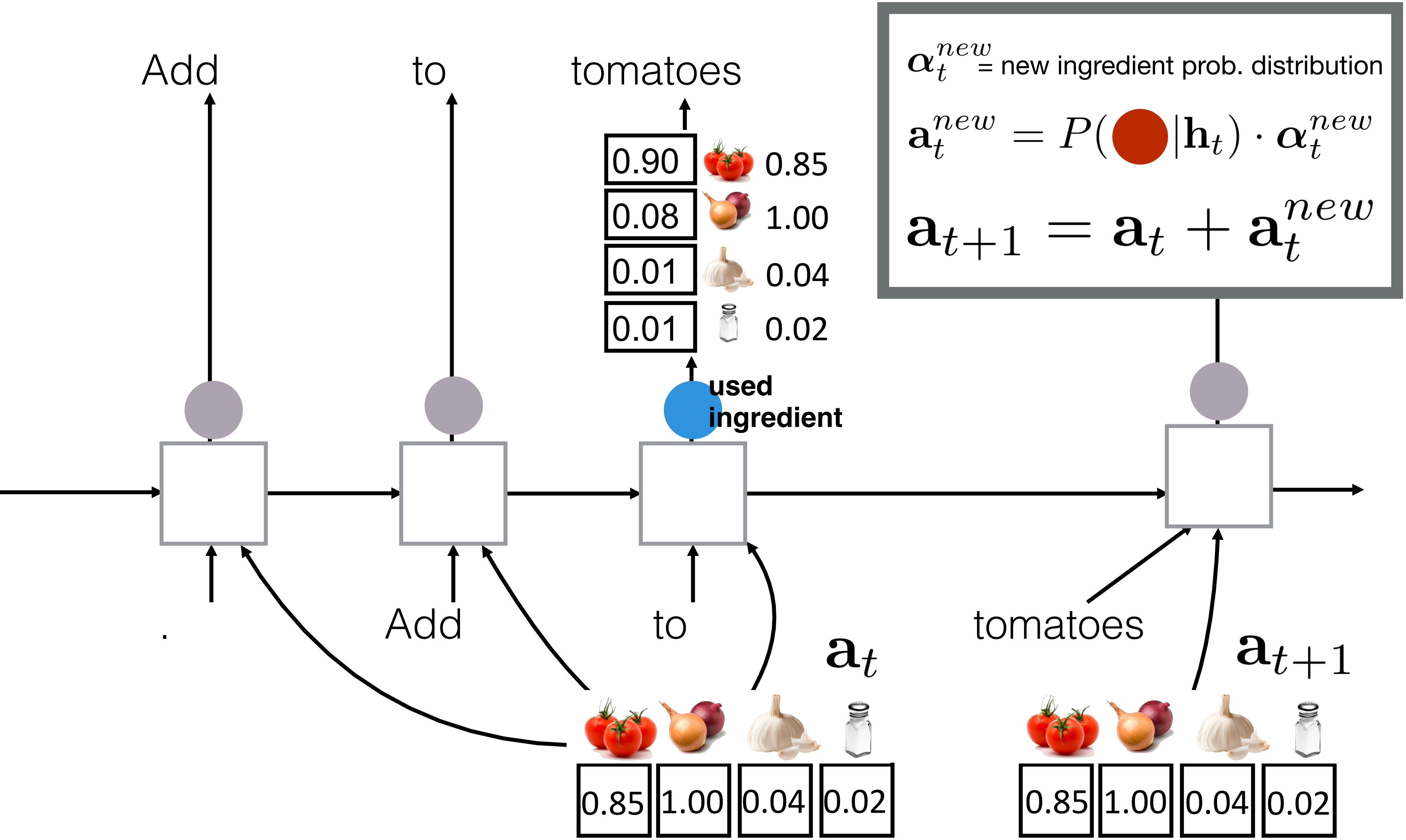
Neural checklist model



Neural checklist model



Checklist is probabilistic

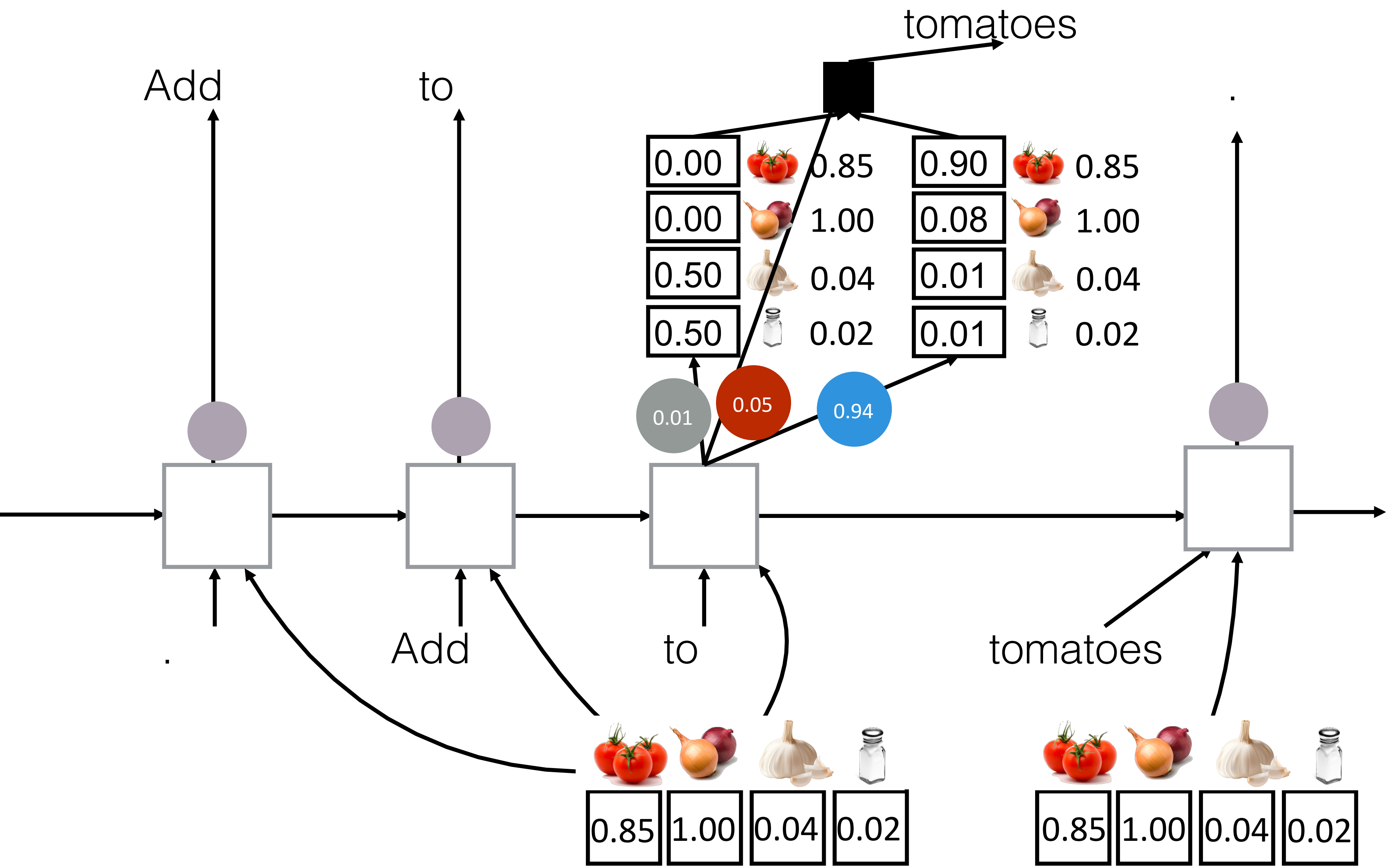


$\boldsymbol{\alpha}_t^{new}$ = new ingredient prob. distribution

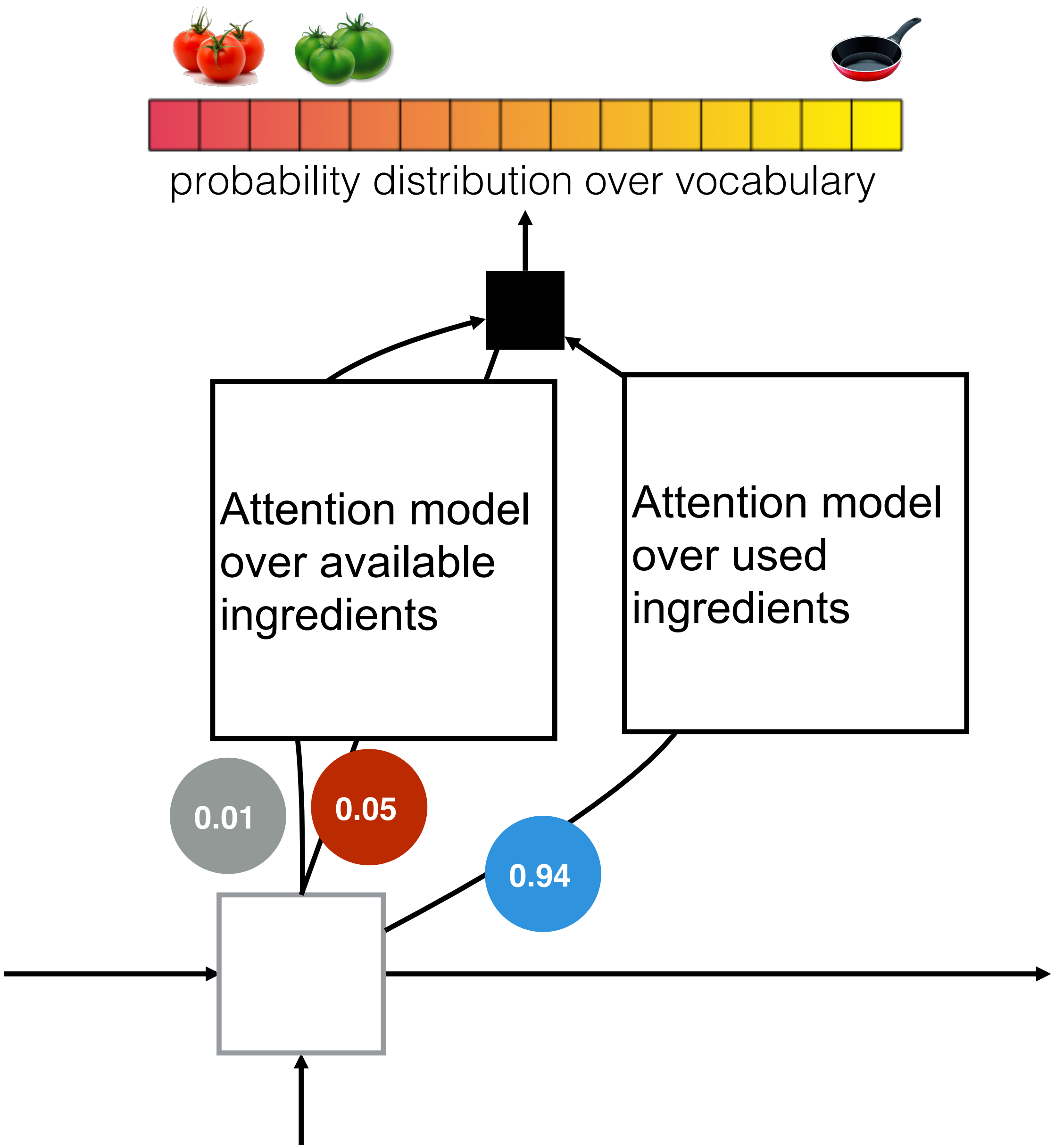
$$\mathbf{a}_t^{new} = P(●|\mathbf{h}_t) \cdot \boldsymbol{\alpha}_t^{new}$$

$$\mathbf{a}_{t+1} = \mathbf{a}_t + \mathbf{a}_t^{new}$$

Hidden state classifier is soft



Interpolation



probability distribution over vocabulary

Attention model over available ingredients

Attention model over used ingredients

$$W_o \in \mathbb{R}^{|V| \times k}$$

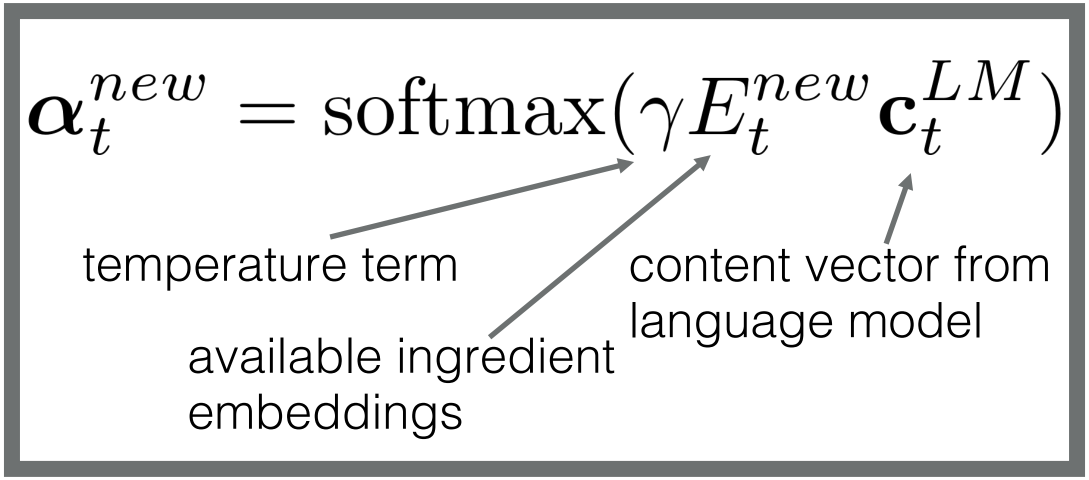
$$\mathbf{w_t} = \mathbf{softmax}(W_o \mathbf{h}_t)$$

$$\mathbf{o}_t = P({\color{gray}\bullet}|\mathbf{h}_t)\mathbf{c}_t^{LM} + P({\color{red}\bullet}|\mathbf{h}_t)\mathbf{c}_t^{new} + P({\color{blue}\bullet}|\mathbf{h}_t)\mathbf{c}_t^{used}$$

$$\mathbf{c}_t^{LM} = W_h \mathbf{h}_t$$

$$W_h \in \mathbb{R}^{k \times k}$$

Choose ingredient via attention

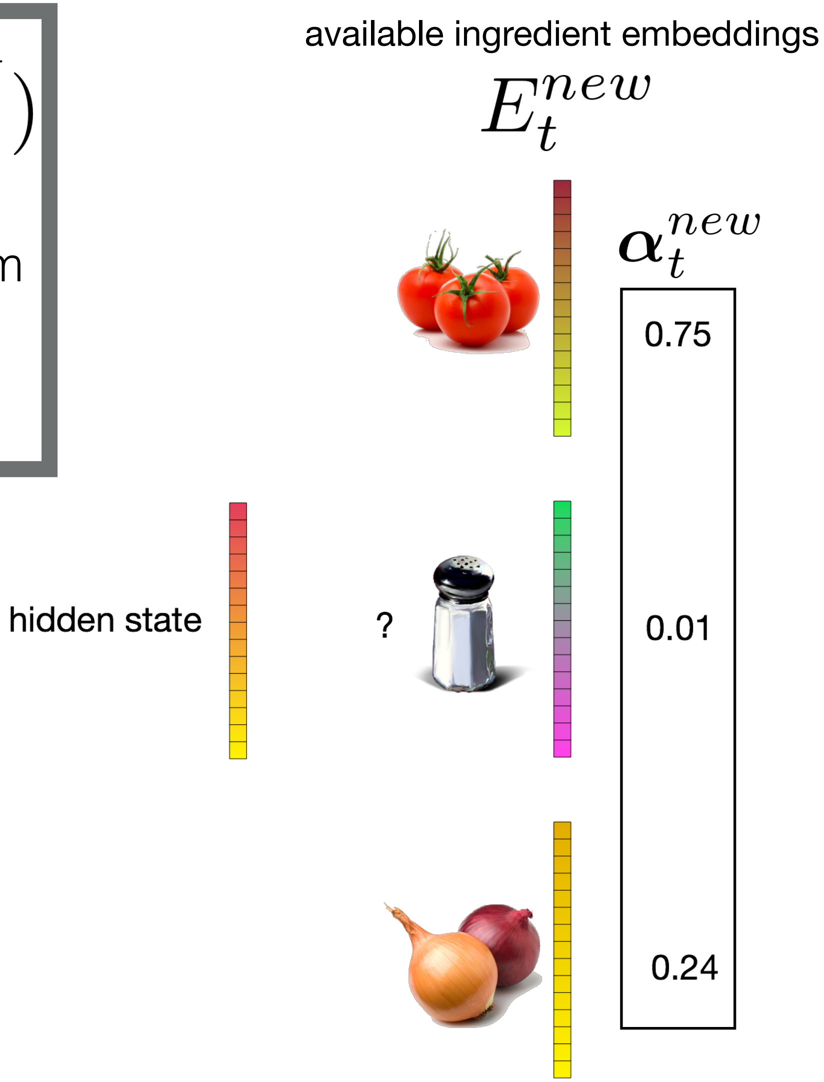


$$\boldsymbol{\alpha}_t^{new} = \mathrm{softmax}(\gamma E_t^{new} \mathbf{c}_t^{LM})$$

temperature term

available ingredient embeddings

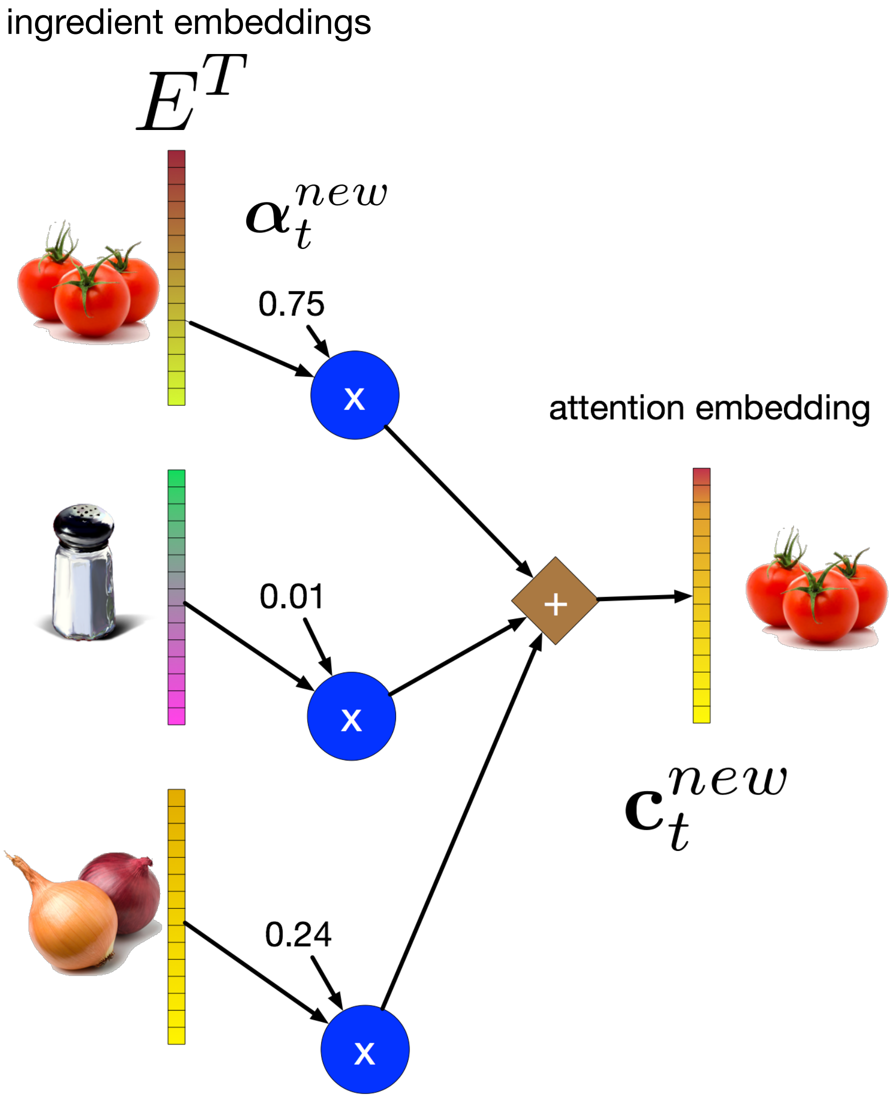
content vector from language model


available ingredient embeddings
E_t^{new}
$\boldsymbol{\alpha}_t^{new}$
0.75
hidden state
?
0.01
0.24

Attention models for other NLP tasks
MT (Balasubramanian et al. 13,
Bahdanau et al. 14)
Sentence summarization (Rush et al. 15)
Machine reading (Cheng et al. 16)
Image captioning (Xu et al. 15)

Attention-generated embeddings

Can generate an embedding from the attention probabilities



$$\mathbf{c}_t^{new} = E^T \boldsymbol{\alpha}_t^{new}$$

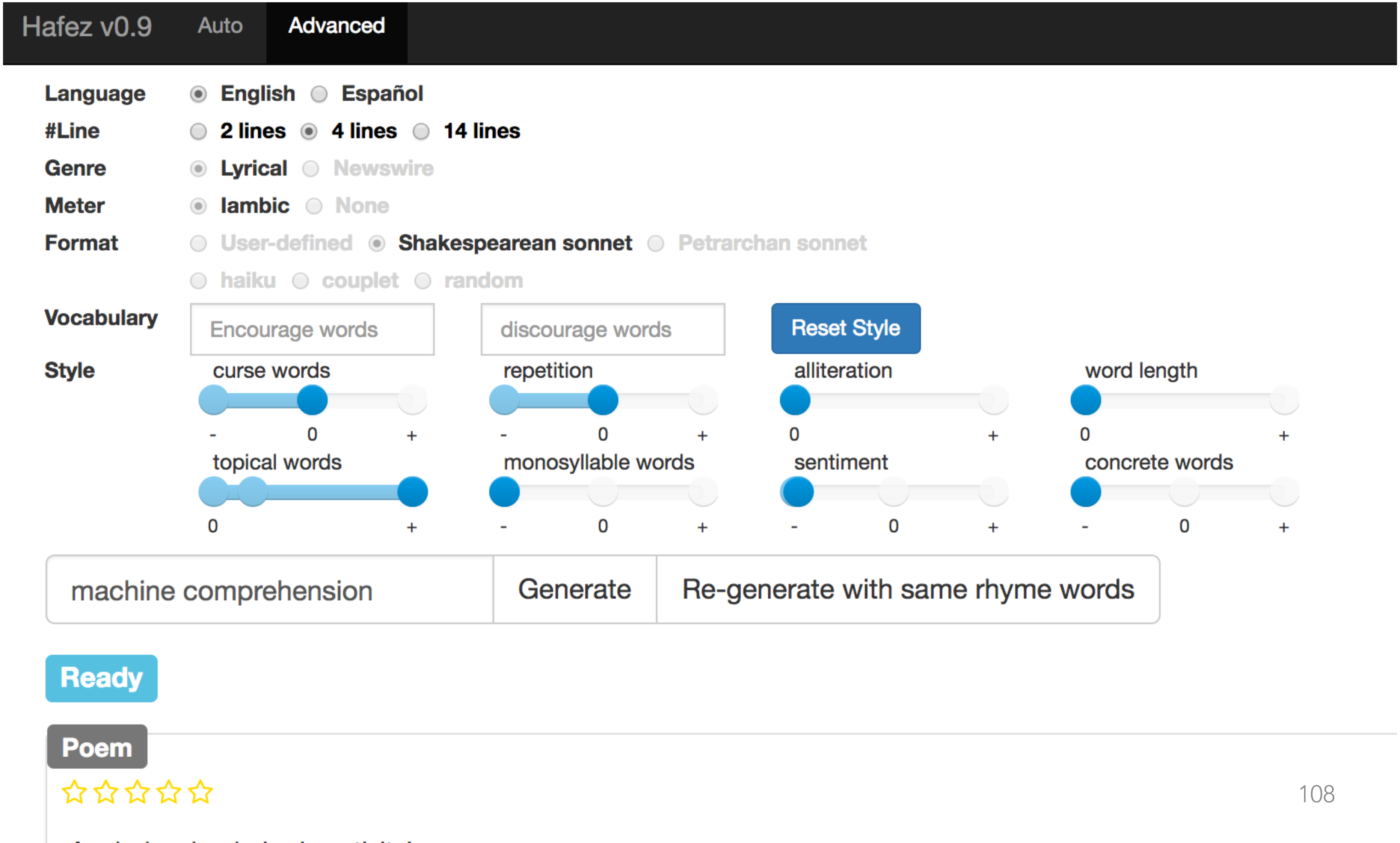
Discussion Points

- Strength and challenges of deep learning?

… what do NNs think about this?

Hafez: Neural Sonnet Writer

(Ghazvininejad et al. 2016)



Language ● English ○ Español

#Line ○ 2 lines ● 4 lines ○ 14 lines

Genre ● Lyrical ○ Newswire

Meter ● Iambic ○ None

Format ○ User-defined ● Shakespearean sonnet ○ Petrarchan sonnet ○ haiku ○ couplet ○ random

Vocabulary: Encourage words | discourage words

Style

curse words: - 0 +

repetition: - 0 +

alliteration: 0 +

word length: 0 +

topical words: 0 +

monosyllable words: - 0 +

sentiment: - 0 +

concrete words: - 0 +

machine comprehension

Neural Sonnets

Deep Convolution Network

Outrageous channels on the wrong *connections*,
An empty space without an open *layer*,
A closet full of black and blue *extensions*,
Connections by the *closure operator.*

Theory

Another way to reach the wrong *conclusion!*
A vision from a total *transformation*,
Created by the great *magnetic fusion*,
Lots of people need an *explanation.*

Discussion Points

- Strength and challenges of deep learning?

- Representation learning
  - Less efforts on feature engineering (at the cost of more hyperparameter tuning!)
  - In computer vision: NN learned representation is significantly better than human engineered features
  - In NLP: often NN induced representation is concatenated with additional human engineered features.
- Data
  - Most success from massive amount of clean (expensive) data
  - Recent surge of data creation type papers (especially AI challenge type tasks)
  - Which significantly limits the domains & applications
  - Need stronger models for unsupervised & distantly supervised approaches

Discussion Points

- Strength and challenges of deep learning?
- Architecture
    - allows for flexible, expressive, and creative modeling
- Easier entry to the field
    - Recent breakthrough from engineering advancements than theoretic advancements
    - Several NN platforms, code sharing culture

Neural Recipe Example #1

title: oven eggplant

In a small bowl , combine the cheese , eggplant , basil , oregano , tomato sauce and onion . Mix well . Shape mixture into 6 patties , each about 3/4-inch thick.
Place on baking sheet .
Bake at 350 degrees for 30 minutes or until lightly browned .
Southern living magazine , sometime in 1980 .
Typed for you by nancy coleman .

eggplant
cheese cottage
lowfat
chopped onion
bay ground leaf
basil
oregano
tomato sauce
provolone

Cook eggplant in boiling water , covered , for 10 min .
Drain and cut in half lengthwise . scoop out insides leaving 1/2 '' shell . Mash insides with cottage cheese onion , bay leaf , basil , oregano and tomato sauce .
Preheat oven to 350 ^ stuff eggplant halves , place in casserole dish and bake covered for 15 min .
Add a little water to bottom of pan to keep eggplant moist . top with provolone cheese .
Bake 5 more min uncovered 1 serving =

CONVOLUTION NEURAL NETWORK

Next several slides borrowed from Alex Rush

Models with Sliding Windows

- Classification/prediction with sliding windows
  - E.g., neural language model
- Feature representations with sliding window
  - E.g., sequence tagging with CRFs or structured perceptron

$$\left[w_1 \;\; w_2 \;\; w_3 \;\; w_4 \;\; w_5\right] \; w_6 \;\; w_7 \;\; w_8$$

$$w_1 \; \left[w_2 \;\; w_3 \;\; w_4 \;\; w_5 \;\; w_6\right] \; w_7 \;\; w_8$$

$$w_1 \;\; w_2 \; \left[w_3 \;\; w_4 \;\; w_5 \;\; w_6 \;\; w_7\right] \; w_8$$

$$\vdots$$

Sliding Windows w/ Convolution

Let our input be the embeddings of the full sentence, $\mathbf{X} \in \mathbb{R}^{n \times d^0}$

$$\mathbf{X} = [v(w_1), v(w_2), v(w_3), \dots, v(w_n)]$$

Define a window model as $NN_{window} : \mathbb{R}^{1 \times (d_{\mathrm{win}} d^0)} \mapsto \mathbb{R}^{1 \times d_{\mathrm{hid}}}$,

$$NN_{window}(\mathbf{x}_{win}) = \mathbf{x}_{win}\mathbf{W}^1 + \mathbf{b}^1$$

The convolution is defined as $NN_{conv} : \mathbb{R}^{n \times d^0} \mapsto \mathbb{R}^{(n - d_{\mathrm{win}} + 1) \times d_{\mathrm{hid}}}$,

$$NN_{conv}(\mathbf{X}) = \tanh \begin{bmatrix} NN_{window}(\mathbf{X}_{1:d_{\mathrm{win}}}) \\ NN_{window}(\mathbf{X}_{2:d_{\mathrm{win}}+1}) \\ \vdots \\ NN_{window}(\mathbf{X}_{n-d_{\mathrm{win}}:n}) \end{bmatrix}$$

Pooling Operations

- Pooling "over-time" operations $f : \mathbb{R}^{n \times m} \mapsto \mathbb{R}^{1 \times m}$
  1. $f_{max}(\mathbf{X})_{1,j} = \max_i X_{i,j}$
  2. $f_{min}(\mathbf{X})_{1,j} = \min_i X_{i,j}$
  3. $f_{mean}(\mathbf{X})_{1,j} = \sum_i X_{i,j} / n$

$$f(\mathbf{X}) = \begin{bmatrix} \Downarrow & \Downarrow & \dots \\ \Downarrow & \Downarrow & \dots \\ & \vdots & \\ \Downarrow & \Downarrow & \dots \end{bmatrix} = \begin{bmatrix} & \dots & \end{bmatrix}$$

Convolution + Pooling

$$\hat{y} = \text{softmax}(f_{max}(NN_{conv}(\mathbf{X}))\mathbf{W}^2 + \mathbf{b}^2)$$

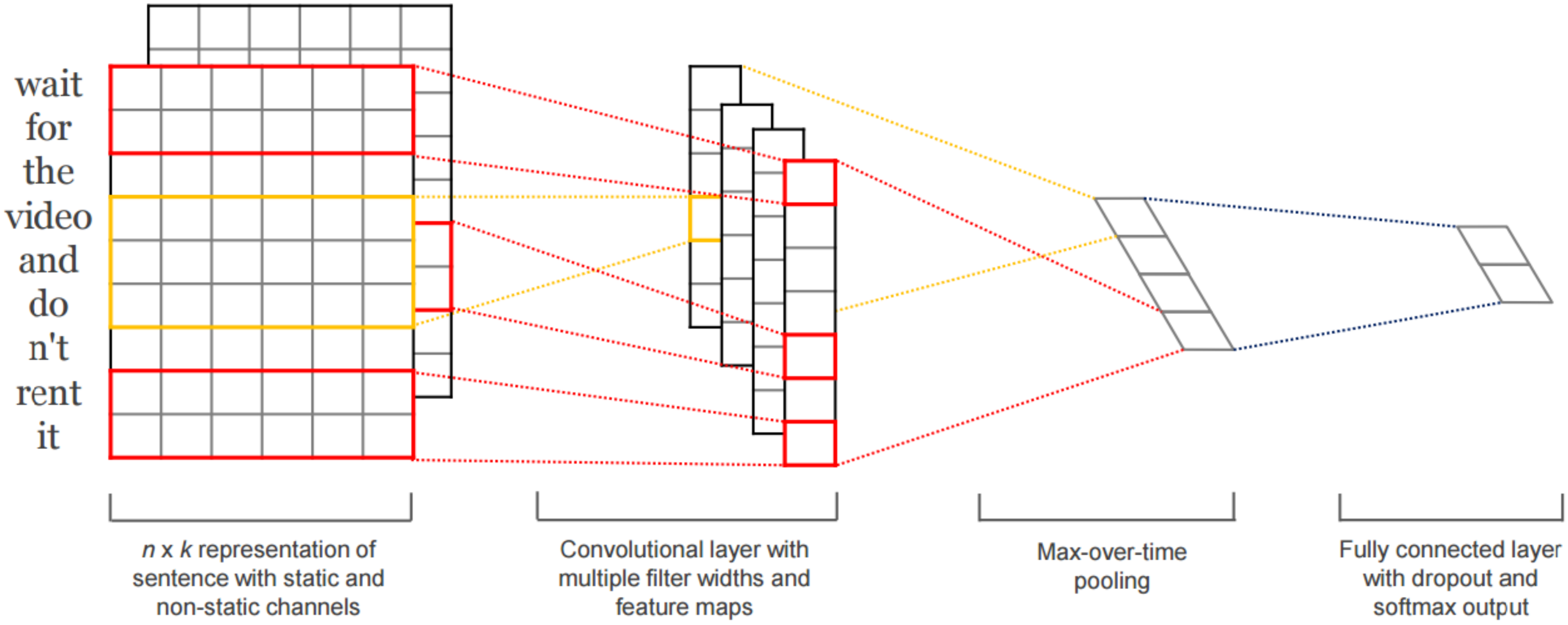
- $\mathbf{W}^2 \in \mathbb{R}^{d_{\text{hid}} \times d_{\text{out}}}$, $\mathbf{b}^2 \in \mathbb{R}^{1 \times d_{\text{out}}}$
- Final linear layer $\mathbf{W}^2$ uses learned window features

Multiple Convolutions

$$\hat{y} = \text{softmax}([f(NN^1_{conv}(\mathbf{X})), f(NN^2_{conv}(\mathbf{X})), \ldots, f(NN^f_{conv}(\mathbf{X}))]\mathbf{W}^2 + \mathbf{b}^2)$$

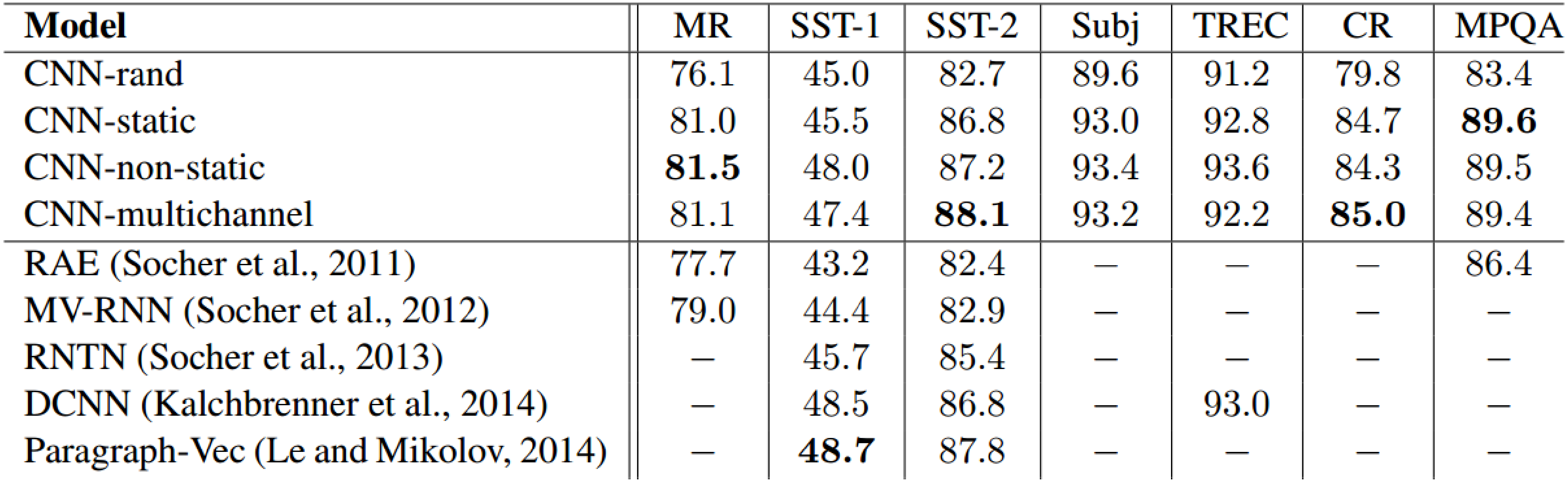
- Concat several convolutions together.
- Each NN^1, NN^2, etc uses a different d_{win}
- Allows for different window-sizes (similar to multiple n-grams)

Convolution Diagram (kim 2014)


wait
for
the
video
and
do
n't
rent
it
n x *k* representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

- $n = 9$, $d_{\text{hid}} = 4$, $d_{\text{out}} = 2$
- red- $d_{\text{win}} = 2$, blue- $d_{\text{win}} = 3$, (ignore back channel)

Text Classification (Kim 2014)



Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	**89.6**
CNN-non-static	**81.5**	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	–	–	–	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	–	–	–	–
RNTN (Socher et al., 2013)	–	45.7	85.4	–	–	–	–
DCNN (Kalchbrenner et al., 2014)	–	48.5	86.8	–	93.0	–	–
Paragraph-Vec (Le and Mikolov, 2014)	–	**48.7**	87.8	–	–	–	–

AlexNet (krizhevsky et al., 2012)

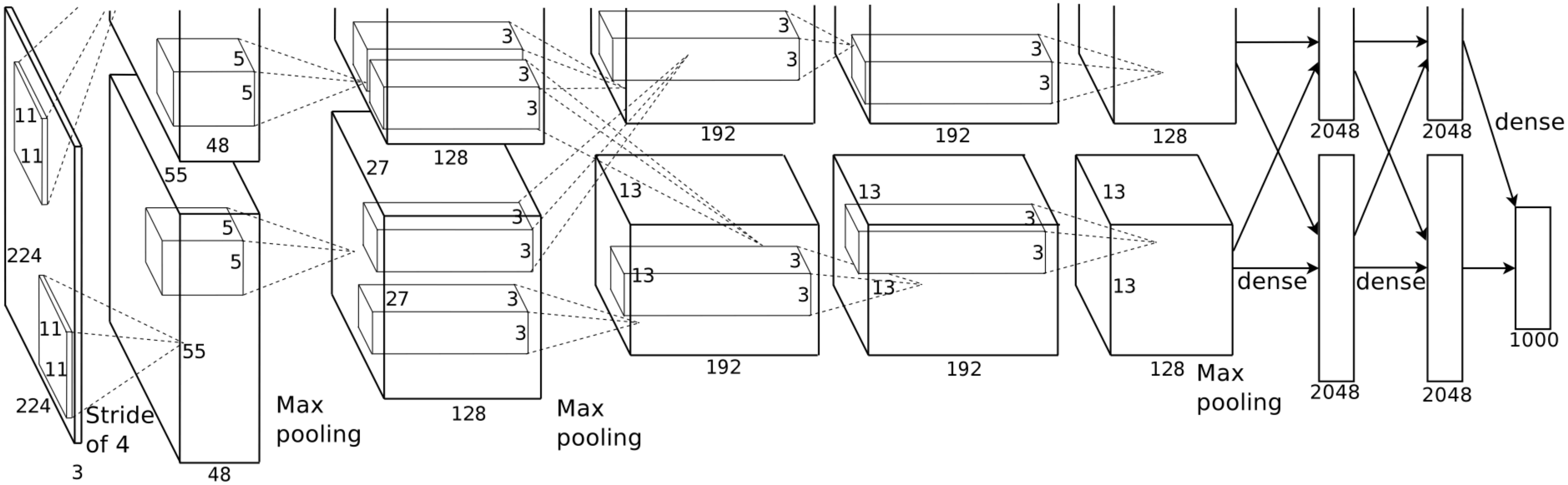


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.