Lecture 6: Recurrent Neural Networks

André Martins & Vlad Niculae

Deep Structured Learning Course, Fall 2019

Announcements

- Solution to Homework 1 is out.
- Homeworks and projects will be graded today!
- The deadline for turning in Homework 2 is in one week (November 1).

Today's Roadmap

Before we talked about linear sequential models. Today we'll cover neural sequential models:

- Recurrent neural networks.
- Backpropagation through time.
- Neural language models.
- The vanishing gradient problem.
- Gated units: LSTMs and GRUs.
- Bidirectional LSTMs.
- Example: ELMO representations.
- From sequences to trees: recursive neural networks.
- Other deep auto-regressive models: PixelRNNs.

Outline

1. **Recurrent Neural Networks**
   - Sequence Generation
   - Sequence Tagging
   - Pooled Classification
2. The Vanishing Gradient Problem: GRUs and LSTMs
3. Beyond Sequences
   - Recursive Neural Networks
   - Pixel RNNs
4. Implementation Tricks
5. Conclusions

Recurrent Neural Networks

Lots of interesting data is sequential in nature: words in sentences, DNA, stock market returns

How do we represent an arbitrarily long history?

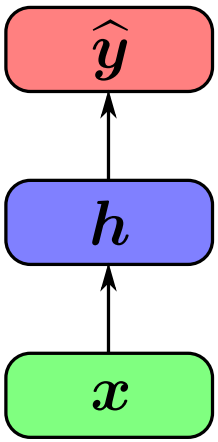
Feed-forward vs Recurrent Networks

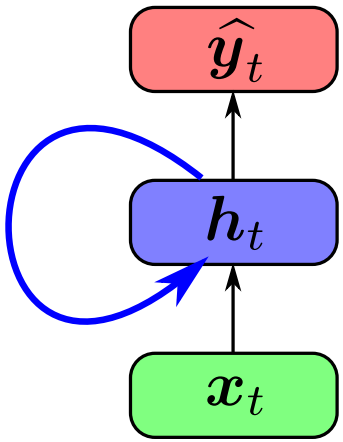
- Feed-forward neural networks:

$$\begin{aligned} \boldsymbol{h} &= \boldsymbol{g}(\mathbf{V}\boldsymbol{x} + \boldsymbol{c}) \\ \widehat{\boldsymbol{y}} &= \mathbf{W}\boldsymbol{h} + \boldsymbol{b} \end{aligned}$$

- Recurrent neural networks (Elman, 1990):

$$\begin{aligned} \boldsymbol{h}_t &= \boldsymbol{g}(\mathbf{V}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{h}_{t-1} + \boldsymbol{c}) \\ \widehat{\boldsymbol{y}}_t &= \mathbf{W}\boldsymbol{h}_t + \boldsymbol{b} \end{aligned}$$

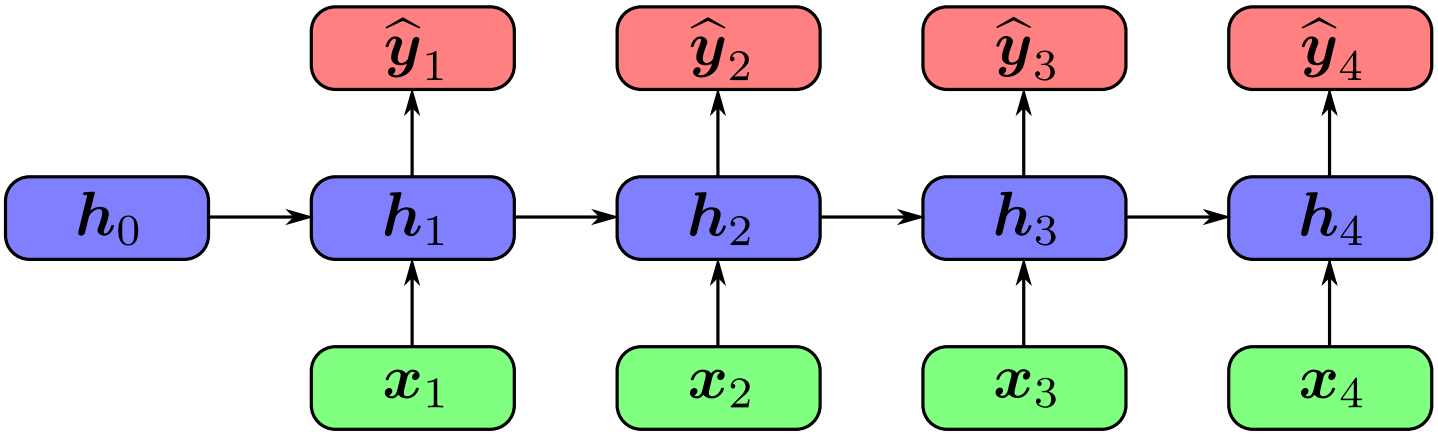




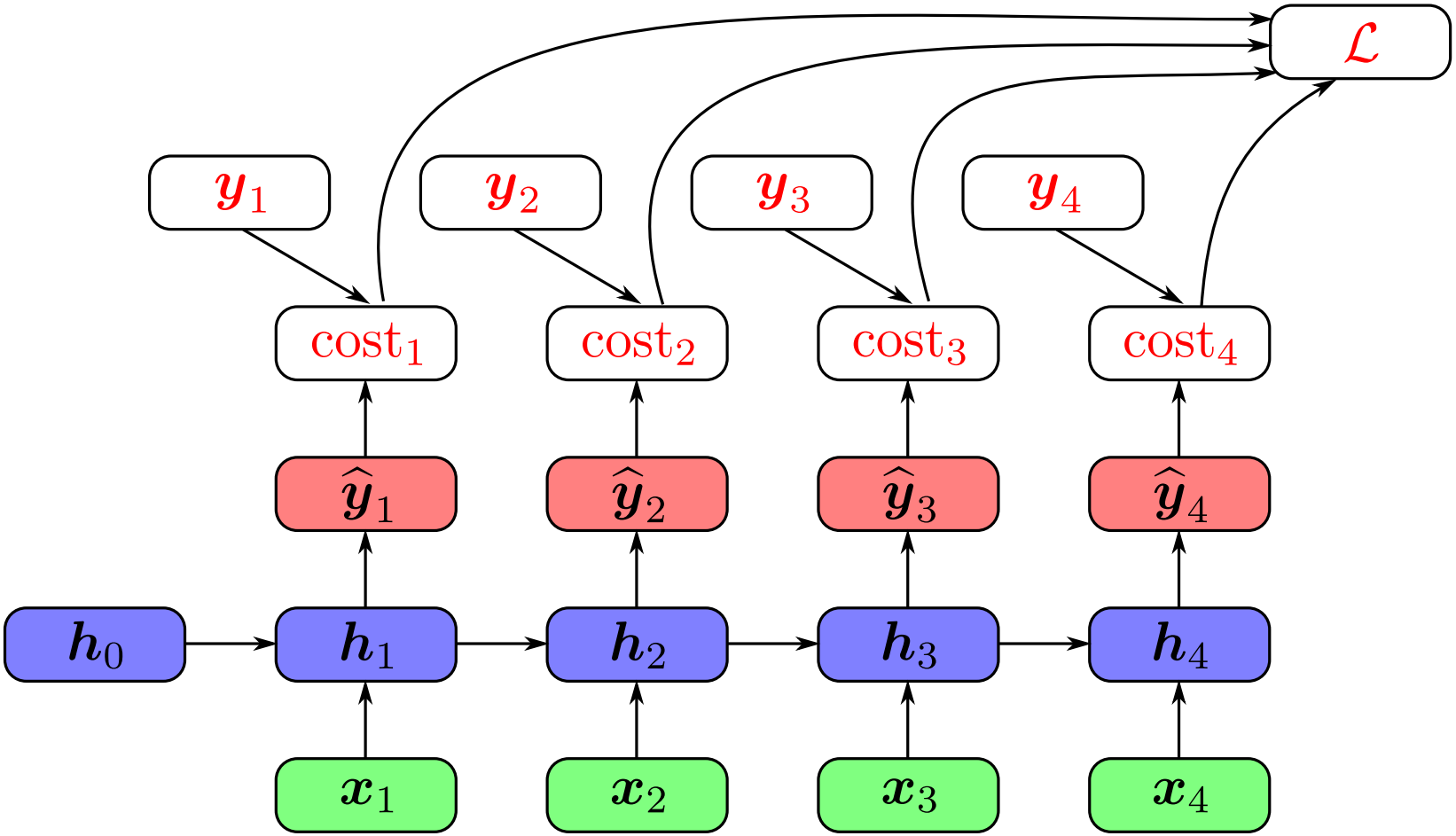
Unrolling the Graph

What happens if we unroll this graph?

Unrolling the Graph



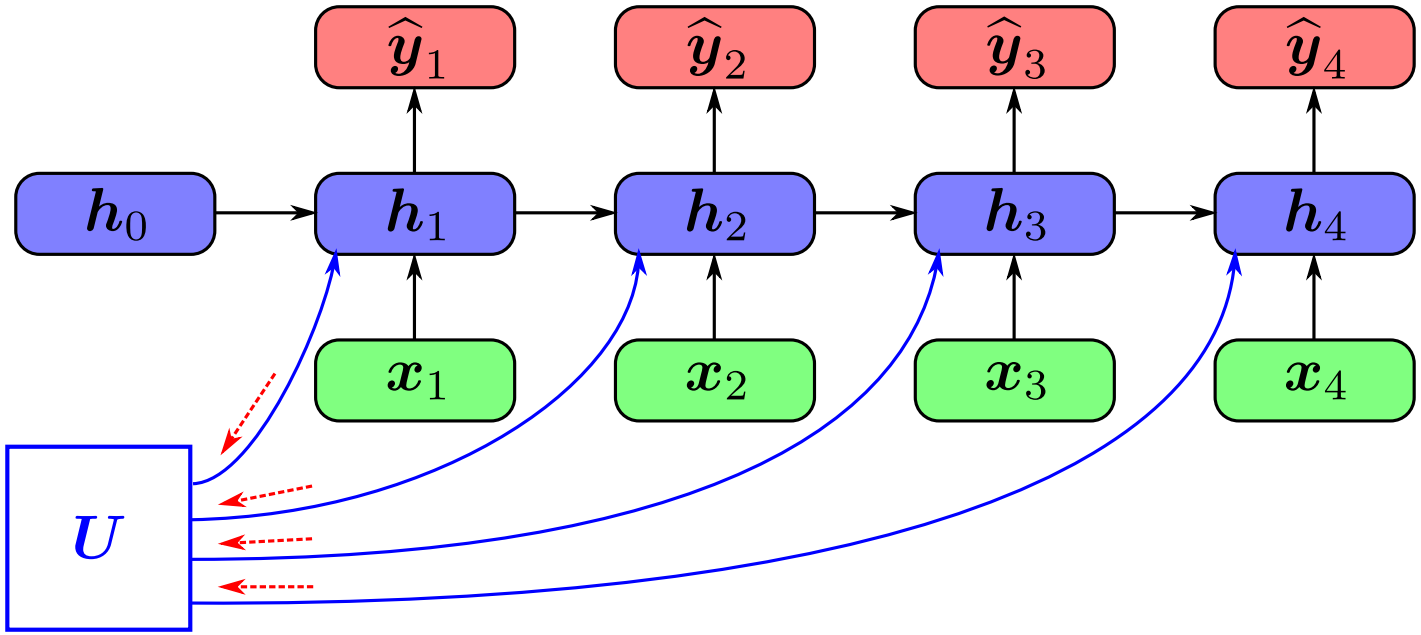
Unrolling the Graph



How do We Train the RNN Parameters?

- The unrolled graph is a well-formed (DAG) computation graph—we can run the gradient backpropagation algorithm as usual
- Parameters are tied (shared) accross "time"
- Derivatives are aggregated across time steps
- This instantiation is called backpropagation through time (BPTT).

Parameter Tying



$$\frac{\partial \mathcal{L}}{\partial \mathbf{U}} = \sum_{t=1}^{4} \frac{\partial \boldsymbol{h}_t}{\partial \mathbf{U}} \frac{\partial \mathcal{L}}{\partial \boldsymbol{h}_t}$$

- Same idea as when learning the filters in convolutional neural networks

Where Can We Use RNNs?

We'll see three usages of RNNs:

1. **Sequence generation:** generates symbols sequentially with an auto-regressive model (e.g. language modeling)
2. **Sequence tagging:** takes a sequence as input, and returns a label for every element in the sequence (e.g. POS tagging)
3. **Pooled classification:** takes a sequence as input, and returns a single label by pooling the RNN states.

Outline

1. **Recurrent Neural Networks**
   - Sequence Generation
   - Sequence Tagging
   - Pooled Classification
2. **The Vanishing Gradient Problem: GRUs and LSTMs**
3. **Beyond Sequences**
   - Recursive Neural Networks
   - Pixel RNNs
4. **Implementation Tricks**
5. **Conclusions**

Example: Language Modeling

One of the possible usages of RNNs is in language modeling.

Recap: Full History Model

$$\mathbb{P}(\text{START}, \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_L, \text{STOP}) = \prod_{i=1}^{L+1} \mathbb{P}(\boldsymbol{y}_i | \boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$$

- Assumes the generation of each word depends on the entire history (*all* the previous words)
- Huge expressive power!
- But: too many parameters to estimate! (How many?)
- Cannot generalize well

Can We Have Unlimited Memory?

In the linear sequential model class, we avoided the full history by considering limited memory models (i.e. Markov models)

Alternative: consider *all* the history, but compress it into a vector!

RNNs can do this!

Auto-Regressive Models

Key ideas:

- Feed back the output in the previous time step as input in the current time step.

$$\boldsymbol{x}_i = \boldsymbol{y}_{i-1}$$

- Maintain a state vector $\boldsymbol{h}_i$ which is a function of the previous state vector and the current input: this state will compress all the history!

$$\boldsymbol{h}_i = \boldsymbol{g}(\mathbf{V}\boldsymbol{x}_i + \mathbf{U}\boldsymbol{h}_{i-1} + \boldsymbol{c})$$

- Compute next output probability:

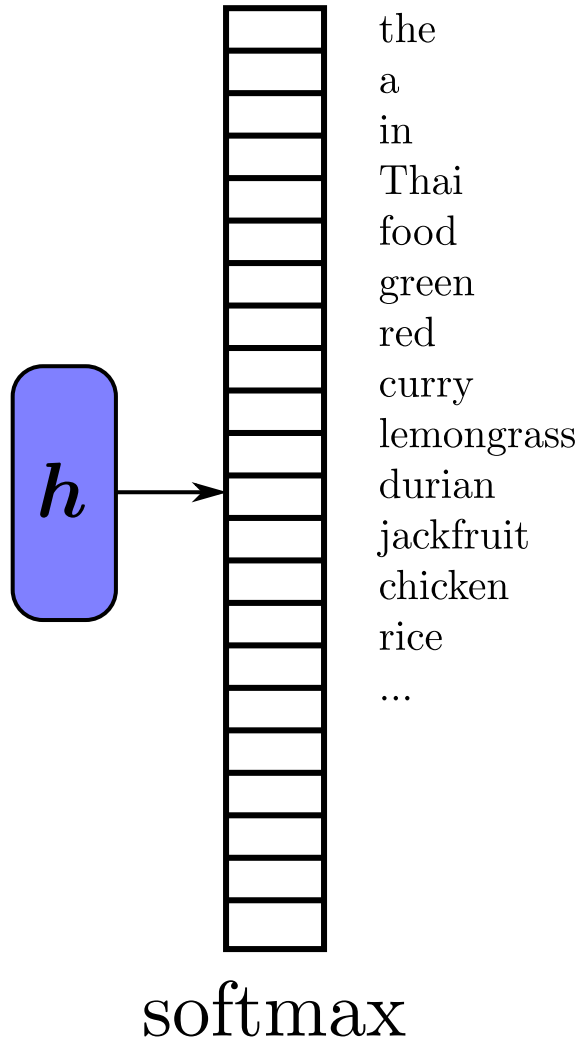
$$\mathbb{P}(\boldsymbol{y}_i | \boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1}) = \mathbf{softmax}(\mathbf{W}\boldsymbol{h}_i + \boldsymbol{b})$$

Let's see each of these steps in detail.

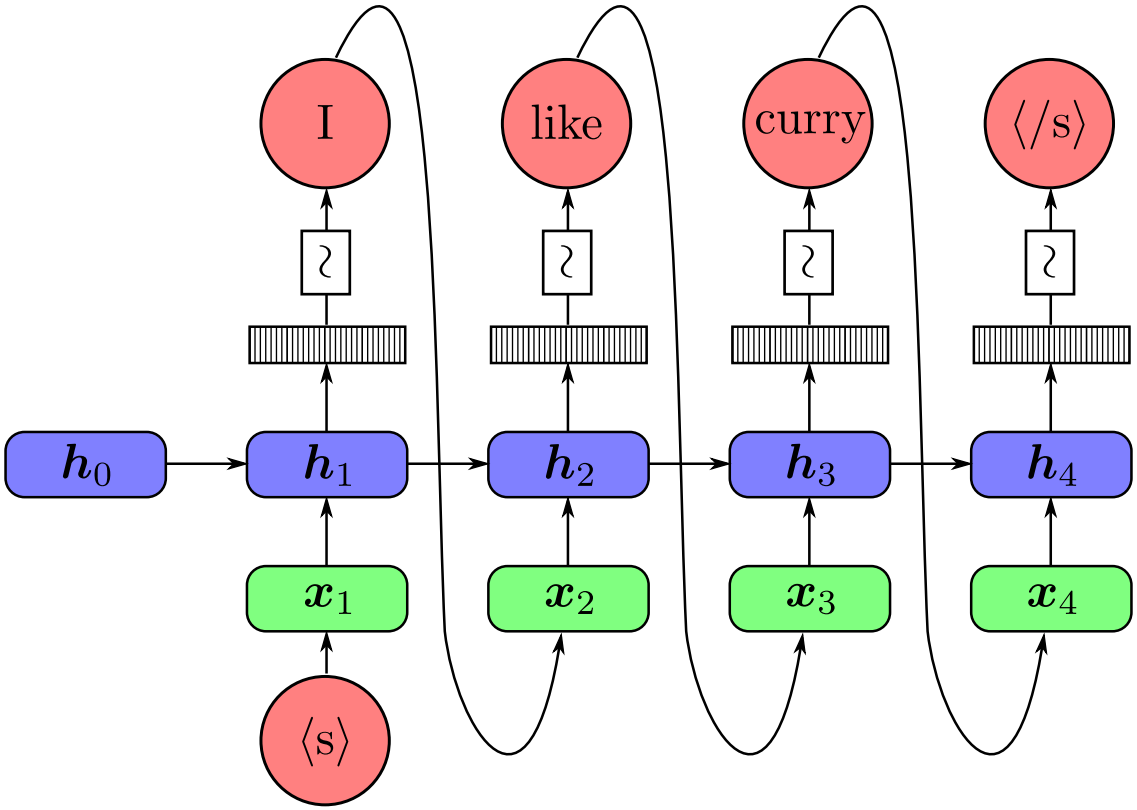
Language Modeling: Large Softmax

- Assume we want to generate text, and y_t is a word in the vocabulary
- Typically, large vocabulary size: $|V| = 100,000$

$$
\begin{aligned}
\boldsymbol{z} &= \mathbf{W}\boldsymbol{h} + \boldsymbol{b} \\
p(y_t = i) &= \frac{\exp(z_i)}{\sum_j \exp(z_j)} \\
&= \mathbf{softmax}_i(\boldsymbol{z})
\end{aligned}
$$



Language Modeling: Auto-Regression



$$\begin{aligned}
\mathbb{P}(y_1, \ldots, y_L) &= \mathbb{P}(y_1) \times \mathbb{P}(y_2 \mid y_1) \times \ldots \times \mathbb{P}(y_L \mid y_1, \ldots, y_{L-1}) \\
&= \textbf{softmax}(\mathbf{W}h_1 + b) \times \textbf{softmax}(\mathbf{W}h_2 + b) \times \ldots \\
&\quad \times \textbf{softmax}(\mathbf{W}h_L + b)
\end{aligned}$$

Three Problems for Sequence Generating RNNs

Algorithms:

- Sample a sequence from the probability distribution defined by the RNN
- Computing the most probable sequence
- Train the RNN.

Sampling a Sequence

This is easy!

- Compute $\boldsymbol{h}_1$ from $\boldsymbol{x}_1 = \text{START}$
- Sample $\boldsymbol{y}_1 \sim \textbf{softmax}(\mathbf{W}\boldsymbol{h}_1 + \boldsymbol{b})$
- Compute $\boldsymbol{h}_2$ from $\boldsymbol{h}_1$ and $\boldsymbol{x}_2 = \boldsymbol{y}_1$
- Sample $\boldsymbol{y}_2 \sim \textbf{softmax}(\mathbf{W}\boldsymbol{h}_2 + \boldsymbol{b})$
- and so on...

What is the Most Probable Sequence?

Unfortunately, this is hard!

- It would require obtaining the $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots$ that jointly maximize the product $\textbf{softmax}(\mathbf{W}\boldsymbol{h}_1 + \boldsymbol{b}) \times \textbf{softmax}(\mathbf{W}\boldsymbol{h}_2 + \boldsymbol{b}) \times \ldots$
- Note that picking the best $\boldsymbol{y}_i$ greedily at each time step doesn't guarantee the best sequence
- We can get better approximations by doing beam search.

This is rarely needed in language modeling. But it is important in conditional language modeling.

We will talk about this later when discussing sequence-to-sequence models.

Train the RNN

Sequence generating RNNs are typically trained with maximum likelihood estimation

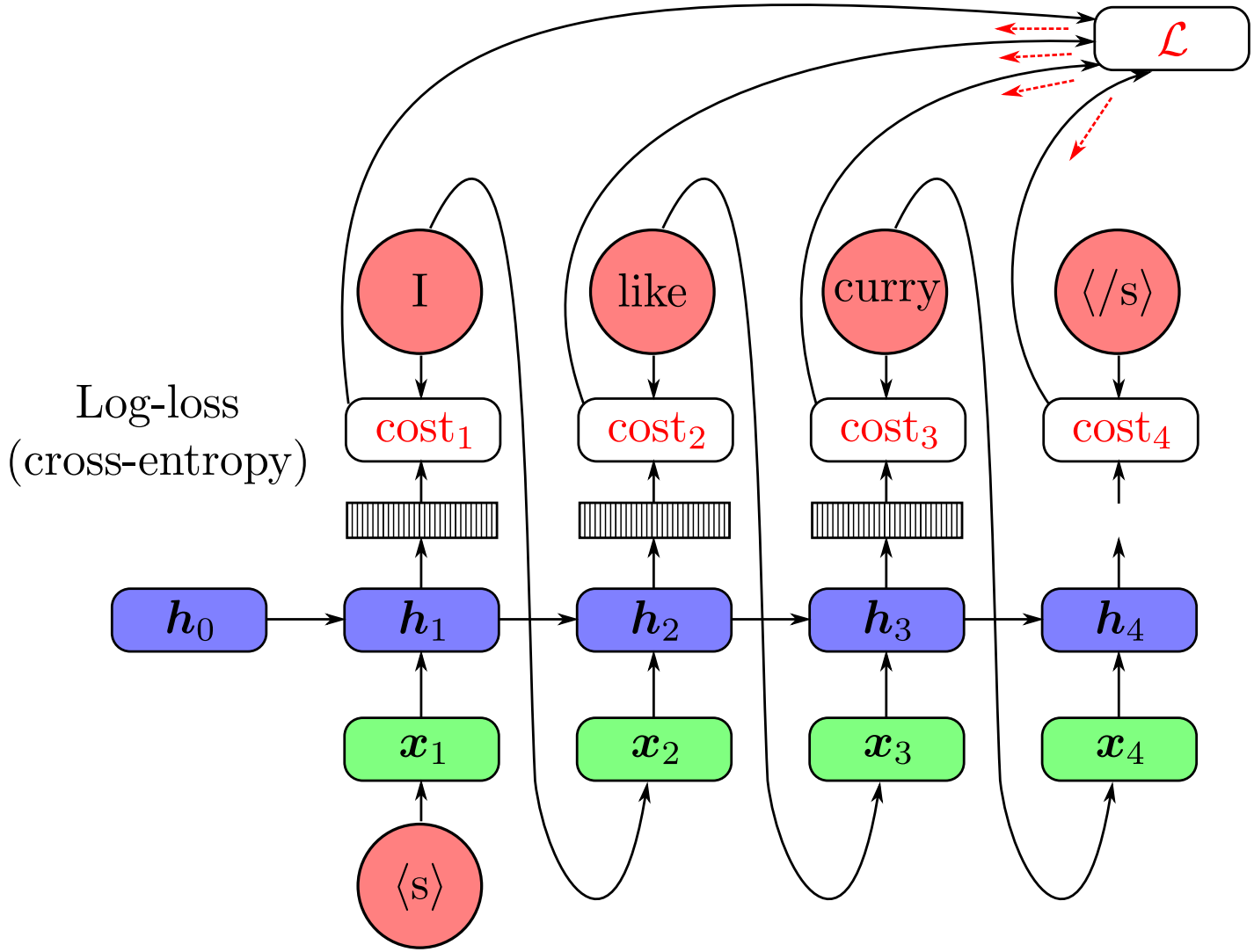
In other words, they are trained to minimize the log-loss (cross-entropy):

$$\mathcal{L}(\Theta, \boldsymbol{y}_{1:L}) = -\frac{1}{L+1} \sum_{i=1}^{L+1} \log \mathbb{P}_\Theta(\boldsymbol{y}_i \mid \boldsymbol{y}_0, \ldots, \boldsymbol{y}_{i-1})$$

This is equivalent to minimize perplexity $2^{\mathcal{L}(\Theta, \boldsymbol{y}_{1:L})}$

Intuition: how "perplex" is the model when the ith word is revealed?

Train the RNN



Train the RNN

Unlike Markov (n-gram) models, RNNs never forget!

- However we will see they might have trouble learning to use their memories (more soon...)

Teacher Forcing and Exposure Bias

Note that we always condition on the **true history** and not on the model's predictions! This is known as teacher forcing.

Teacher forcing cause exposure bias at run time: the model will have trouble recovering from mistakes early on, since it generates histories that it has never observed before.

How to improve this is a current area of research!

Character-Level Language Models

We can also have an RNN over characters instead of words!

Advantage: can generate any combination of characters, not just words in a closed vocabulary.

Disadvantage: need to remember further away in history!

A Char-Level RNN Generating Fake Shakespeare

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.

(Credits: Andrej Karpathy)

A Char-Level RNN Generating a Math Paper

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

(Credits: Andrej Karpathy)

A Char-Level RNN Generating C++ Code

```c
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
  return segtable;
}
```

(Credits: Andrej Karpathy)

Note: these examples are from 3 years ago, now we have even more impressive language generators (e.g. GPT-2)

Instead of RNNs, the most recent language generators use a Transformer architecture

We'll cover this in the last class!

Where Can We Use RNNs?

We'll see three usages of RNNs:

1. **Sequence generation:** generates symbols sequentially with an auto-regressive model (e.g. language modeling) ✓
2. **Sequence tagging:** takes a sequence as input, and returns a label for every element in the sequence (e.g. POS tagging)
3. **Pooled classification:** takes a sequence as input, and returns a single label by pooling the RNN states.

Outline

1. **Recurrent Neural Networks**
   - Sequence Generation
   - Sequence Tagging
   - Pooled Classification
2. The Vanishing Gradient Problem: GRUs and LSTMs
3. Beyond Sequences
   - Recursive Neural Networks
   - Pixel RNNs
4. Implementation Tricks
5. Conclusions

Sequence Tagging with RNNs

In **sequence tagging**, we are given an input sequence $x_1, \ldots, x_L$

The goal is to assign a tag to each element of the sequence, yielding an output sequence $y_1, \ldots, y_L$

Examples: POS tagging, named entity recognition

Differences with respect to sequence generation:

- The input and output are distinct (no need for an auto-regressive model)
- The length of the output is known (same size as the input)

Example: POS Tagging

Map **sentences** to sequences of **part-of-speech tags.**

Time	flies	like	an	arrow	.
noun	verb	prep	det	noun	.

- Need to predict a morphological tag for each word of the sentence
- High correlation between adjacent words!

(Ratnaparkhi, 1999; Brants, 2000; Toutanova et al., 2003)

An RNN-Based POS Tagger

- The inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_L \in \mathbb{R}^{E \times L}$ are word embeddings (obtained by looking up rows in an V-by-E embedding matrix, eventually pre-trained)
- As before, maintain a state vector $\boldsymbol{h}_i$ which is a function of the previous state vector and the current input: this state will compress all the input history!

$$\boldsymbol{h}_i = \boldsymbol{g}(\mathbf{V}\boldsymbol{x}_i + \mathbf{U}\boldsymbol{h}_{i-1} + \boldsymbol{c})$$

- A softmax output layer computes the probability of the current tag given the current and previous words:

$$\mathbb{P}(\boldsymbol{y}_i | \boldsymbol{x}_1, \ldots, \boldsymbol{x}_i) = \textbf{softmax}(\mathbf{W}\boldsymbol{h}_i + \boldsymbol{b})$$

An RNN-Based POS Tagger

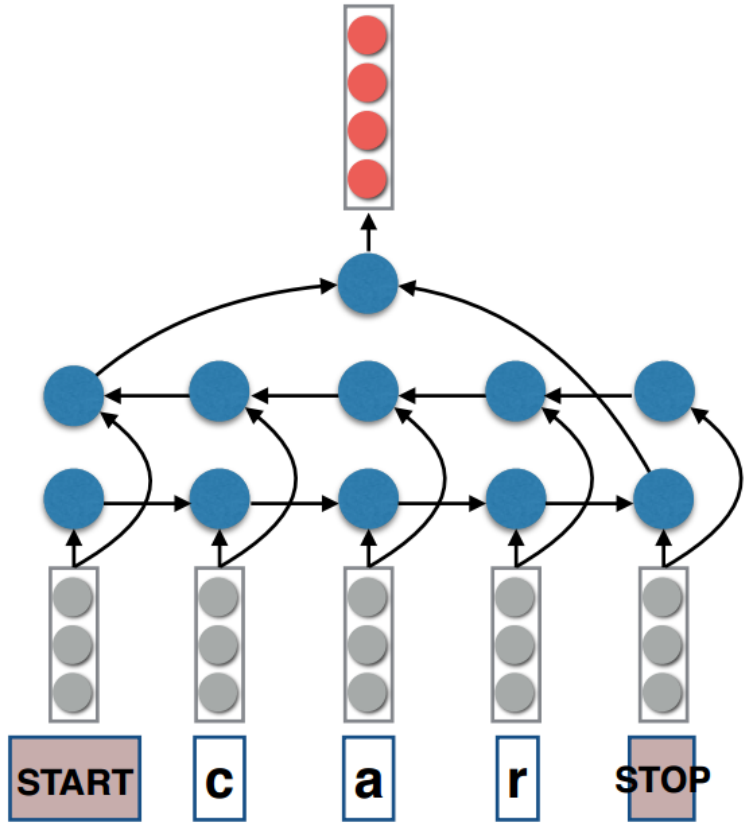
This model can be improved:

- Use a bidirectional RNN to condition also on the following words (combinining a left-to-right and a right-to-left RNN)—more later!
- Use a nested character-level CNN or RNN to obtain embeddings for unseen words.

This model achieved SOTA performance on the Penn Treebank and several other benchmarks (Ling et al., 2015; Wang et al., 2015)!

Bidirectional RNNs

- We can read a sequence from left to right to obtain a representation
- Or we can read it from right to left
- Or we can read it from both and combine the representations
- More later...



(Slide credit: Chris Dyer)

Example: Named Entity Recognition

From **sentences** extract **named entities.**

Louis	Elsevier	was	born	in	Leuven	.
B-PER	I-PER	O	O	O	B-LOC	.

- Identify word segments that refer to entities (person, organization, location)
- Typically done with sequence models and B-I-O tagging

(Zhang and Johnson, 2003; Ratinov and Roth, 2009)

RNN-Based NER

- The model we described for POS tagging works just as well for NER
- However, NER has constraints about tag transitions: e.g., we cannot have I-PER after B-LOC
- The RNN tagger model we described exploits input structure (via the states encoded in the recurrent layer) but lacks output structure...

Combining RNNs and CRFs

In previous lectures, we saw models that exploit output sequential structure: conditional random fields!

However, we described CRFs as a linear model (the scores for emissions and transitions came from a feature-based linear model)

We can easily combine the strengths of RNNs and CRFs to obtain a **non-linear CRF**!

Combining RNNs and CRFs

- Just use the RNN to compute scores for the emissions (instead of the feature-based linear model)
- For the transitions, we can either compute scores from the same recurrent layer, or just have an indicator features that looks at the tag bigrams
- Then replace the softmax output layer by a CRF output layer!
- At training time, plug the forward-backward algorithm in the gradient back-propagation
- At run time, use Viterbi to predict the most likely sequence of tags.

A variant of this model (with a BILSTM instead of a RNN) achieved the SOTA in various NER benchmarks (Lample et al., 2016).

Where Can We Use RNNs?

We'll see three usages of RNNs:

1. **Sequence generation:** generates symbols sequentially with an auto-regressive model (e.g. language modeling) ✓
2. **Sequence tagging:** takes a sequence as input, and returns a label for every element in the sequence (e.g. POS tagging) ✓
3. **Pooled classification:** takes a sequence as input, and returns a single label by pooling the RNN states.

Outline

1. **Recurrent Neural Networks**
   - Sequence Generation
   - Sequence Tagging
   - Pooled Classification
2. The Vanishing Gradient Problem: GRUs and LSTMs
3. Beyond Sequences
   - Recursive Neural Networks
   - Pixel RNNs
4. Implementation Tricks
5. Conclusions

Pooled Classification

What we talked about so far assumes we want to output a sequence of labels (either to generate or tag a full sequence).

What if we just want to predict a **single label** for the whole sequence?

We can still use an RNN to capture the input sequential structure!

We just need to pool the RNNs states, i.e., map them to a single vector

Then add a single softmax to output the final label.

Pooling Strategies

The simplest pooling strategy is just to pick the last RNN state

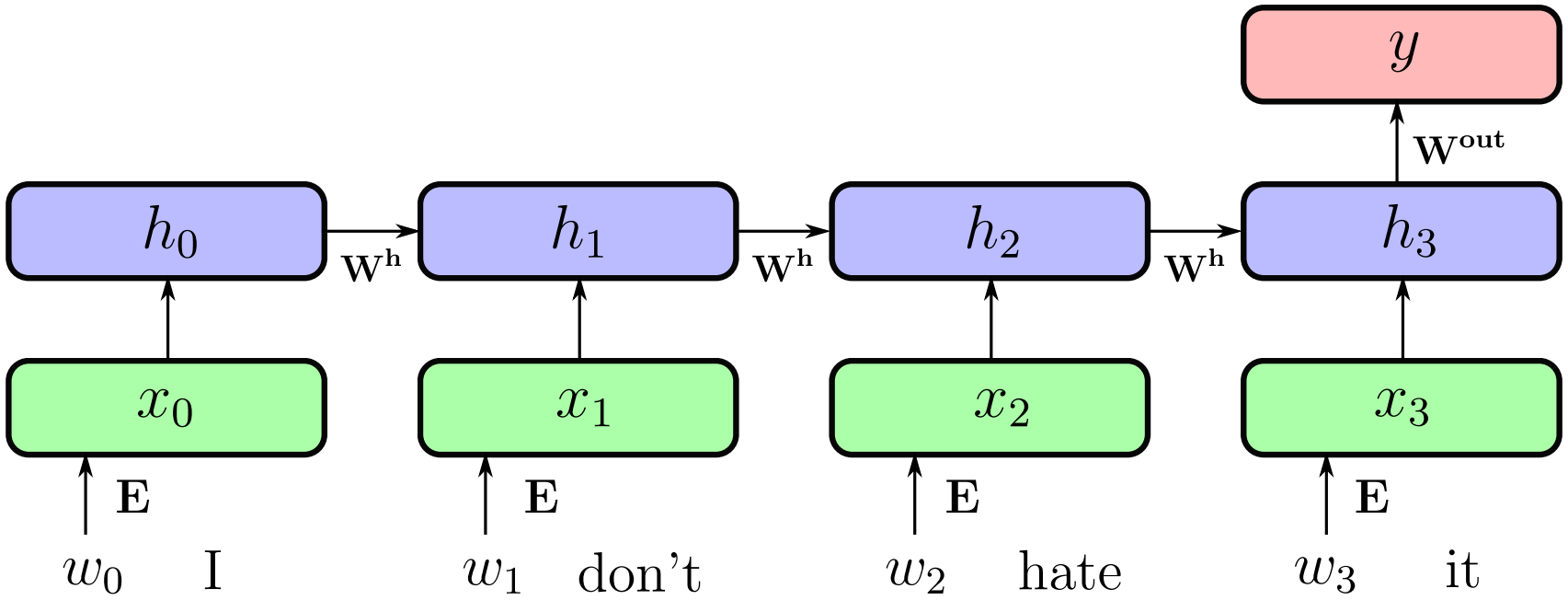
This state results from traversing the full sequence left-to-right, hence it has information about the full sequence!

Disadvantage: for long sequences, memory about the earliest words starts vanishing

Other pooling strategies:

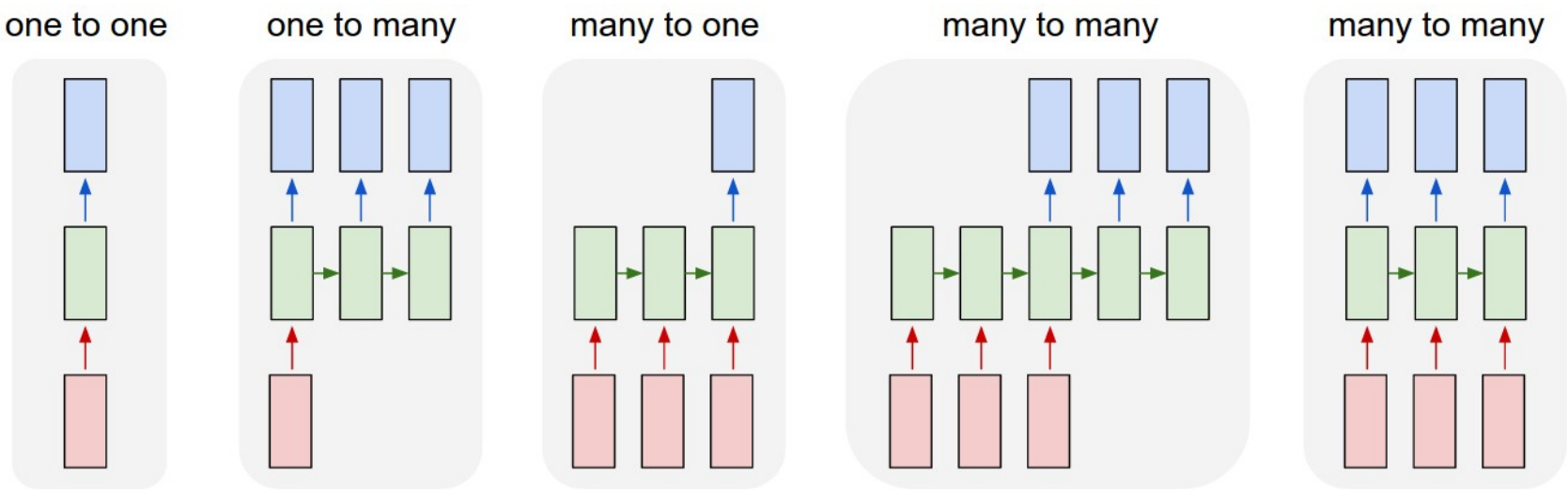
- use a bidirectional RNN and combine both last states of the left-to-right and right-to-left RNN
- average pooling
- ...

Example: Sentiment Analysis



(Slide credit: Ollion & Grisel)

Recurrent Neural Networks are Very Versatile



Check out Andrej Karpathy's blog post "The Unreasonable Effectiveness of Recurrent Neural Networks" (`http://karpathy.github.io/2015/05/21/rnn-effectiveness/`).

Outline

1. Recurrent Neural Networks
   - Sequence Generation
   - Sequence Tagging
   - Pooled Classification
2. **The Vanishing Gradient Problem: GRUs and LSTMs**
3. Beyond Sequences
   - Recursive Neural Networks
   - Pixel RNNs
4. Implementation Tricks
5. Conclusions

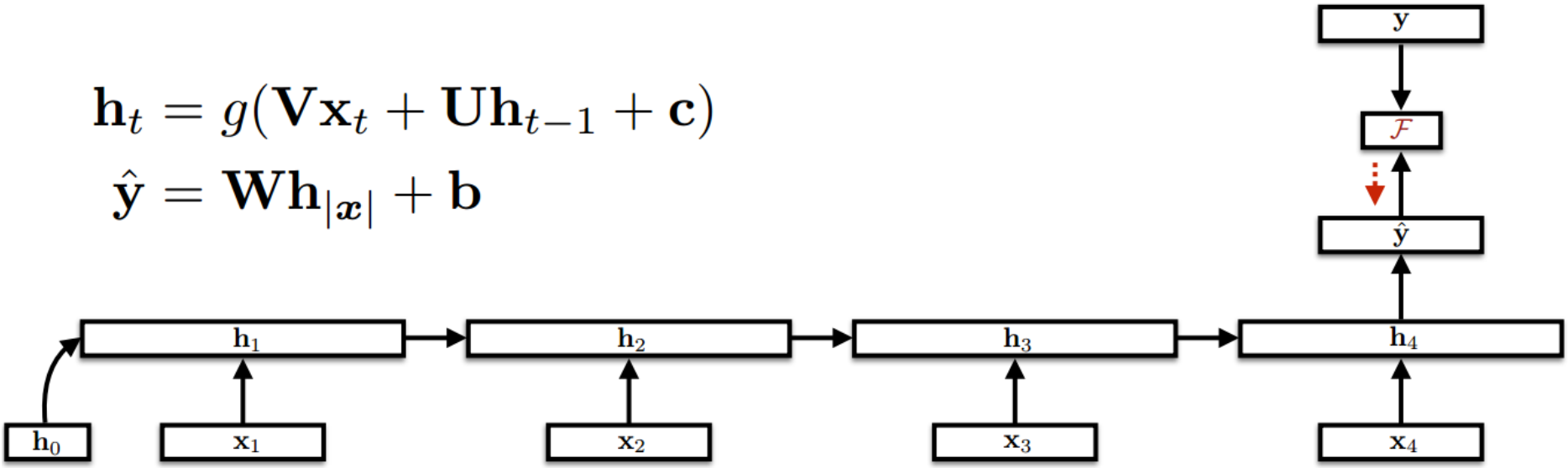
Training the RNN

This is done by backpropagation through time.

Backpropagation Through Time

What happens to the gradients as we go back in time?

$$\mathbf{h}_t = g(\mathbf{V}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{c})$$
$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{h}_{|\boldsymbol{x}|} + \mathbf{b}$$



(Slide credit: Chris Dyer)

Backpropagation Through Time

What happens to the gradients as we go back in time?

$$\frac{\partial \mathcal{F}}{\partial \boldsymbol{h}_1} = \underbrace{\frac{\partial \boldsymbol{h}_2}{\partial \boldsymbol{h}_1}\frac{\partial \boldsymbol{h}_3}{\partial \boldsymbol{h}_2}\frac{\partial \boldsymbol{h}_4}{\partial \boldsymbol{h}_3}}_{\prod_{t=2}^{4}\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}}}\frac{\partial \widehat{\boldsymbol{y}}}{\partial \boldsymbol{h}_4}\frac{\partial \mathcal{F}}{\partial \widehat{\boldsymbol{y}}}$$

where

$$\prod_t \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} = \prod_t \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{z}_t}\frac{\partial \boldsymbol{z}_t}{\partial \boldsymbol{h}_{t-1}} = \prod_t \text{Diag}(\boldsymbol{g}'(\boldsymbol{z}_t))\mathbf{U}$$

Three cases:

- largest eigenvalue of $\mathbf{U}$ exactly 1: gradient propagation is stable
- largest eigenvalue of $\mathbf{U} < 1$: gradient vanishes (exponential decay)
- largest eigenvalue of $\mathbf{U} > 1$: gradient explodes (exponential growth)

Vanishing and Exploding Gradients

Exploding gradients can be dealt with by gradient clipping (truncating the gradient if it exceeds some magnitude)

Vanishing gradients are more frequent and harder to deal with

- In practice: long-range dependencies are difficult to learn

Solutions:

- Better optimizers (second order methods)
- Normalization to keep the gradient norms stable across time
- Clever initialization so that you at least start with good spectra (e.g., start with random orthonormal matrices)
- Alternative parameterizations: LSTMs and GRUs

Gradient Clipping

- **Norm clipping:**

$$\tilde{\nabla} \leftarrow \begin{cases} \frac{c}{\|\nabla\|}\nabla & \text{if } \|\nabla\| \geq c \\ \nabla & \text{otherwise.} \end{cases}$$

- **Elementwise clipping:**

$$\tilde{\nabla}_i \leftarrow \min\{c, |\nabla_i|\} \times \textbf{sign}(\nabla_i), \ \forall i$$

Alternative RNNs

I'll next describe:

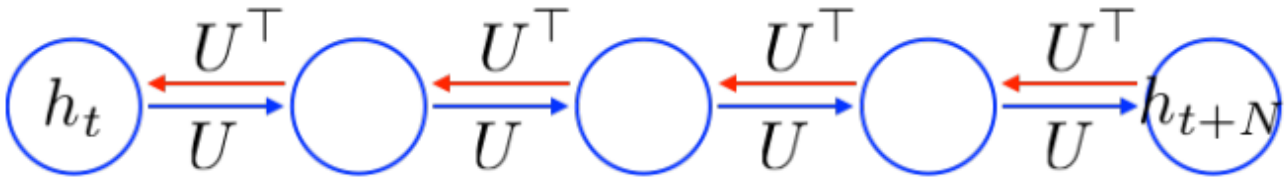
- Gated recurrent units (GRUs; Cho et al. (2014))
- Long short-term memories (LSTMs; Hochreiter and Schmidhuber (1997))

Intuition: instead of multiplying across time (which leads to exponential growth), we want the error to be approximately constant

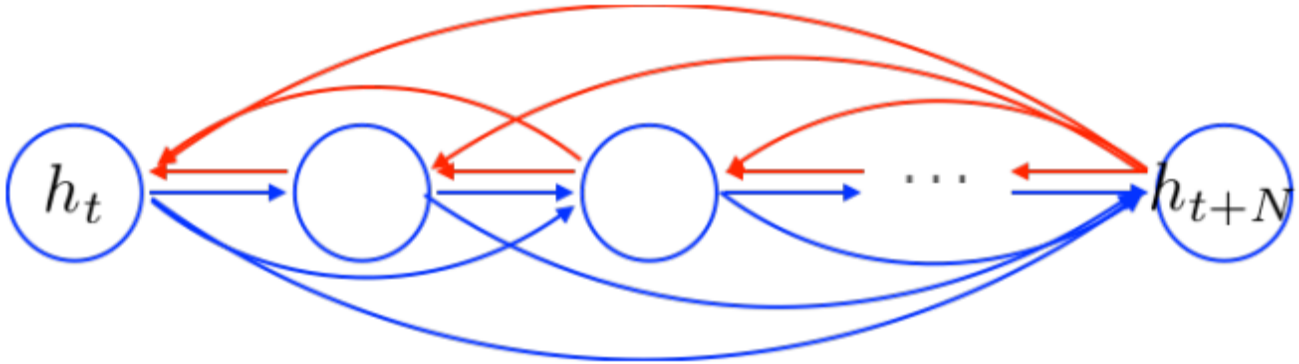
They solve the vanishing gradient problem, but still have exploding gradients (still need gradient clipping)

Gated Recurrent Units (Cho et al., 2014)

- Recall the problem: the error must backpropagate through all the intermediate nodes:



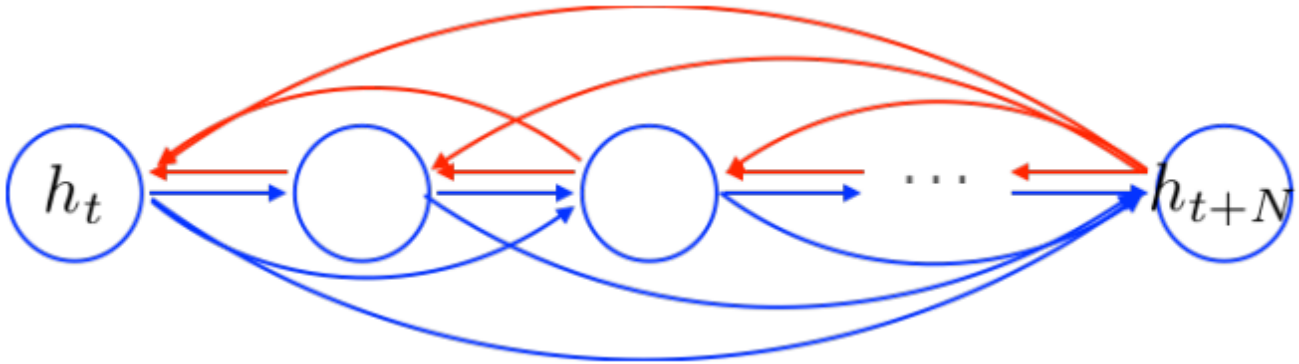
- **Idea:** Maybe we can create some kind of shortcut connections:



(Image credit: Thang Luong, Kyunghyun Cho, Chris Manning)

- Create adaptive shortcuts controlled by special gates

Gated Recurrent Units (Cho et al., 2014)



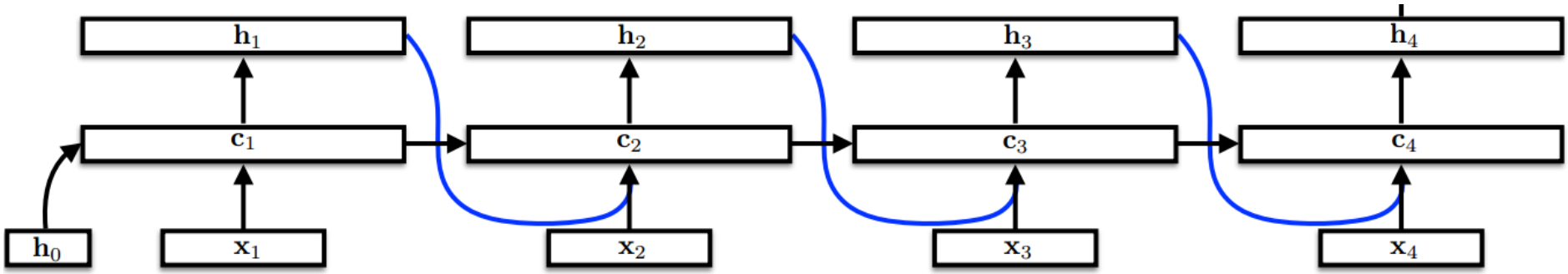
(Image credit: Thang Luong, Kyunghyun Cho, Chris Manning)

$$\boxed{\boldsymbol{h}_t = \boldsymbol{u}_t \odot \tilde{\boldsymbol{h}}_t + (1 - \boldsymbol{u}_t) \odot \boldsymbol{h}_{t-1}}$$

- **Candidate update:** $\tilde{\boldsymbol{h}}_t = \boldsymbol{g}(\mathbf{V}\boldsymbol{x}_t + \mathbf{U}(\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}) + \boldsymbol{b})$
- **Reset gate:** $\boldsymbol{r}_t = \sigma(\mathbf{V}_r\boldsymbol{x}_t + \mathbf{U}_r\boldsymbol{h}_{t-1} + \boldsymbol{b}_r)$
- **Update gate:** $\boldsymbol{u}_t = \sigma(\mathbf{V}_r\boldsymbol{x}_t + \mathbf{U}_u\boldsymbol{h}_{t-1} + \boldsymbol{b}_u)$

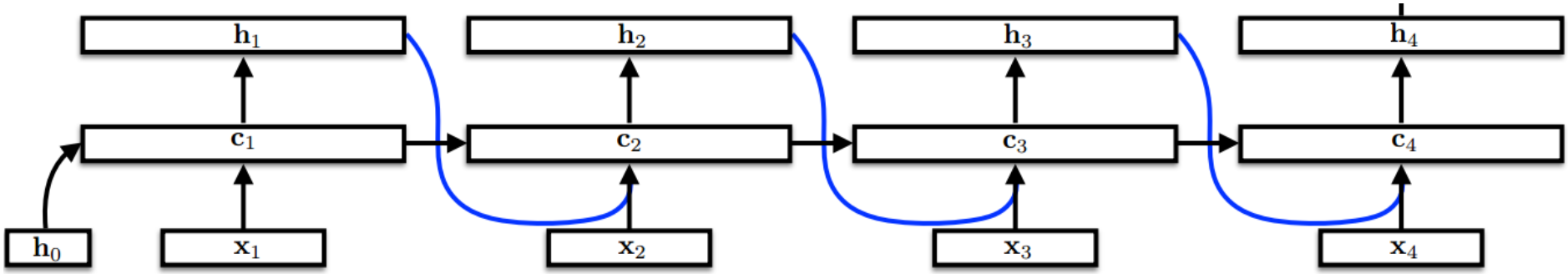
Long Short-Term Memories (Hochreiter and Schmidhuber, 1997)

- **Key idea:** use memory cells c_t
- To avoid the multiplicative effect, flow information *additively* through these cells
- Control the flow with special input, forget, and output gates



(Image credit: Chris Dyer)

Long Short-Term Memories

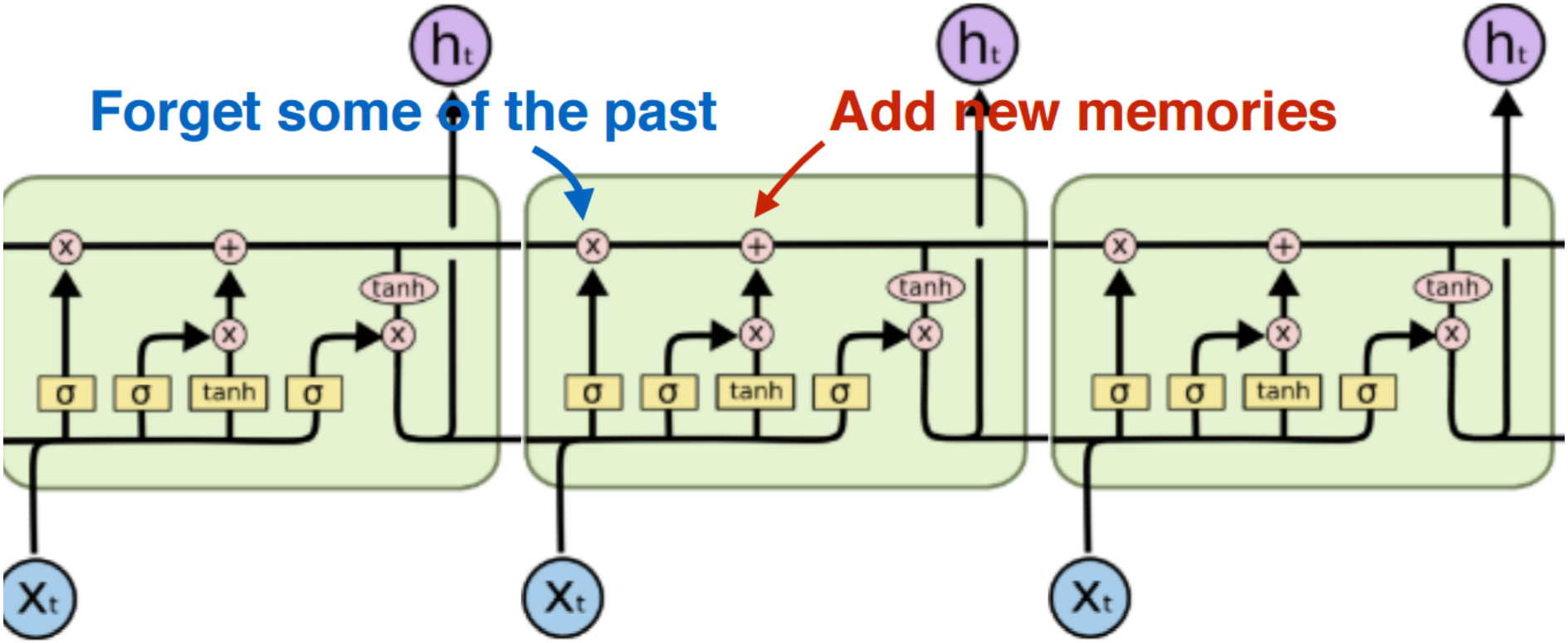


(Image credit: Chris Dyer)

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{g}(\mathbf{V}\boldsymbol{x}_t + \mathbf{U}\boldsymbol{h}_{t-1} + \boldsymbol{b}), \qquad \boldsymbol{h}_t = \boldsymbol{o}_t \odot \boldsymbol{g}(\boldsymbol{c}_t)$$

- **Forget gate:** $\boldsymbol{f}_t = \sigma(\mathbf{V}_f \boldsymbol{x}_t + \mathbf{U}_f \boldsymbol{h}_{t-1} + \boldsymbol{b}_f)$
- **Input gate:** $\boldsymbol{i}_t = \sigma(\mathbf{V}_i \boldsymbol{x}_t + \mathbf{U}_i \boldsymbol{h}_{t-1} + \boldsymbol{b}_i)$
- **Output gate:** $\boldsymbol{o}_t = \sigma(\mathbf{V}_o \boldsymbol{x}_t + \mathbf{U}_o \boldsymbol{h}_{t-1} + \boldsymbol{b}_o)$

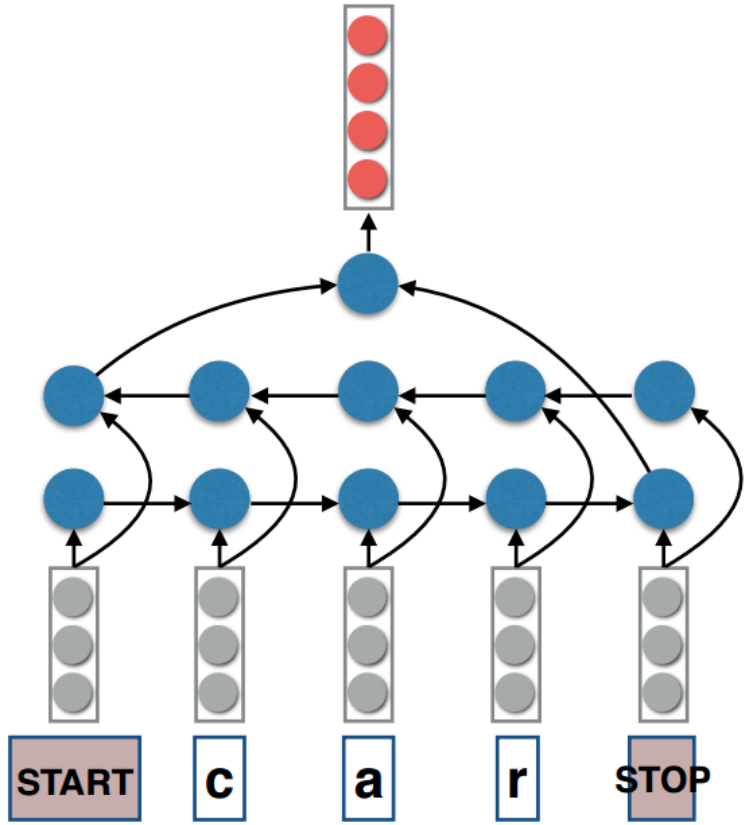
Long Short-Term Memories



(Slide credit: Christopher Olah)

Bidirectional LSTMs

- Same thing as a Bidirectional RNN, but using LSTM units instead of vanilla RNN units.



(Slide credit: Chris Dyer)

LSTMs and BILSTMs: Some Success Stories

- Time series prediction (Schmidhuber et al., 2005)
- Speech recognition (Graves et al., 2013)
- Named entity recognition (Lample et al., 2016)
- Machine translation (Sutskever et al., 2014)
- ELMo (deep contextual) word representations (Peters et al., 2018)
- ... and many others.

Summary

- Better gradient propagation is possible if we use additive rather than multiplicative/highly non-linear recurrent dynamics
- Recurrent architectures are an active area of research (but LSTMs are hard to beat)
- Other variants of LSTMs exist which tie/simplify some of the gates
- Extensions exist for *non-sequential* structured inputs/outputs (e.g. trees): recursive neural networks (Socher et al., 2011), PixelRNN (Oord et al., 2016)

Outline

1. Recurrent Neural Networks
   - Sequence Generation
   - Sequence Tagging
   - Pooled Classification
2. The Vanishing Gradient Problem: GRUs and LSTMs
3. **Beyond Sequences**
   - Recursive Neural Networks
   - Pixel RNNs
4. Implementation Tricks
5. Conclusions

Outline

1. Recurrent Neural Networks
   - Sequence Generation
   - Sequence Tagging
   - Pooled Classification
2. The Vanishing Gradient Problem: GRUs and LSTMs
3. **Beyond Sequences**
   - Recursive Neural Networks
   - Pixel RNNs
4. Implementation Tricks
5. Conclusions

From Sequences to Trees

So far we've talked about recurrent neural networks, which are designed to capture sequential structure

What about other kinds of structure? For example, trees?

It is also possible to tackle these structures with recursive computation, via recursive neural networks.

Recursive Neural Networks

Proposed by Socher et al. (2011) for parsing images and text

Assume a binary tree (each node except the leaves has two children)

Propagate states bottom-up in the tree, computing the parent state $\boldsymbol{p}$ from the children states $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$:

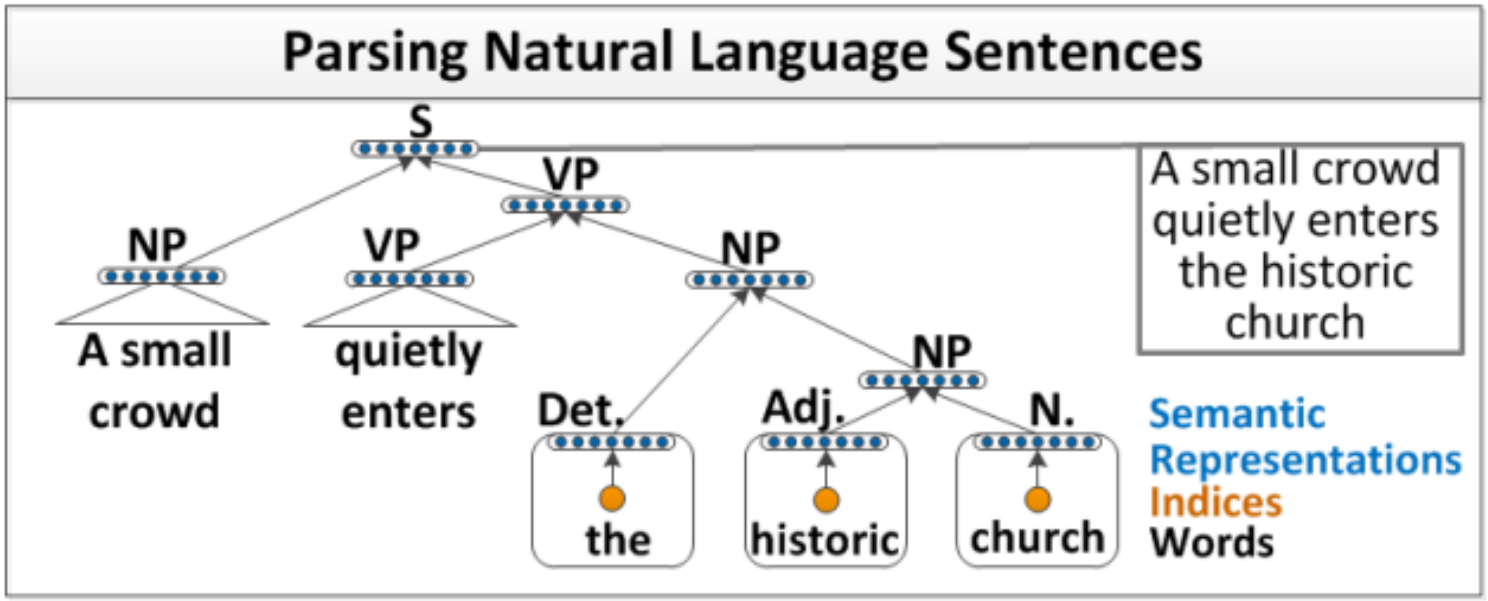
$$\boldsymbol{p} = \tanh\left(\mathbf{W}\begin{bmatrix}\boldsymbol{c}_1\\ \boldsymbol{c}_2\end{bmatrix} + \boldsymbol{b}\right)$$

Use the same parameters $\mathbf{W}$ and $\boldsymbol{b}$ at all nodes

Can compute scores at the root or at each node by appending a softmax output layer at these nodes.

Compositionality in Text

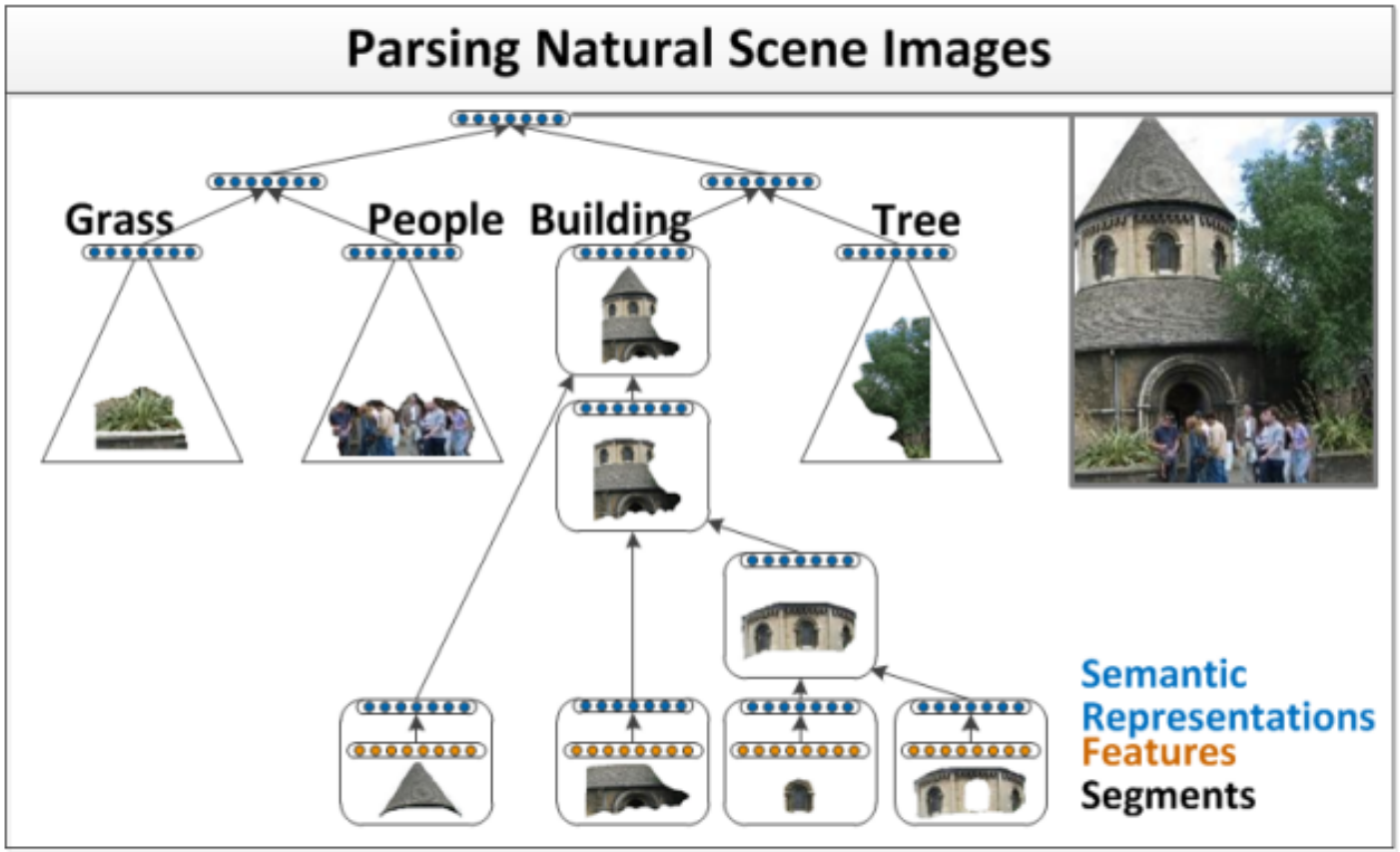
Uses a recurrent net to build a bottom-up parse tree for a sentence.



(Credits: Socher et al. (2011))

Compositionality in Images

Same idea for images.



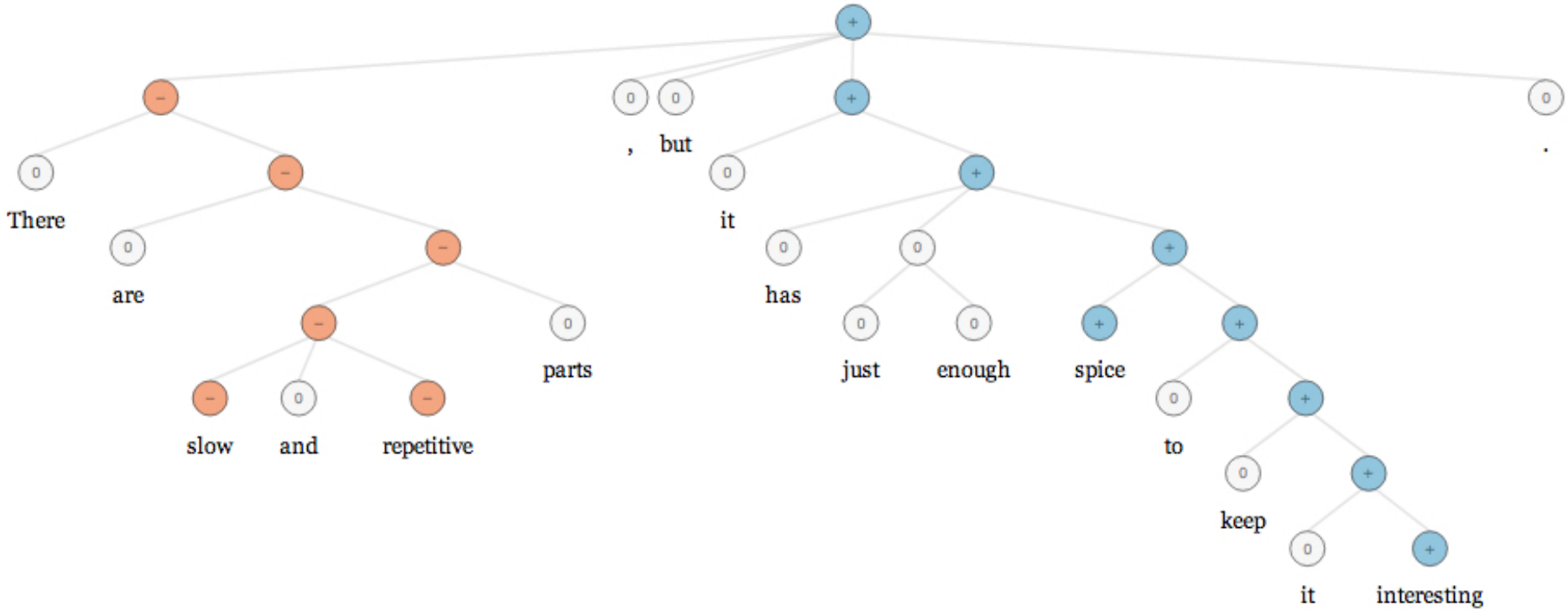
(Credits: Socher et al. (2011))

Tree-LSTMs

Extend recursive neural networks the same way LSTMs extend RNNs, with a few more gates to account for the left and right child.

Extensions exist for non-binary trees.

Fine-Grained Sentiment Analysis



(Taken from Stanford Sentiment Treebank.)

Outline

1. Recurrent Neural Networks
   - Sequence Generation
   - Sequence Tagging
   - Pooled Classification
2. The Vanishing Gradient Problem: GRUs and LSTMs
3. **Beyond Sequences**
   - Recursive Neural Networks
   - Pixel RNNs
4. Implementation Tricks
5. Conclusions

What about Images?

While sequences are 1D, images are 2D.

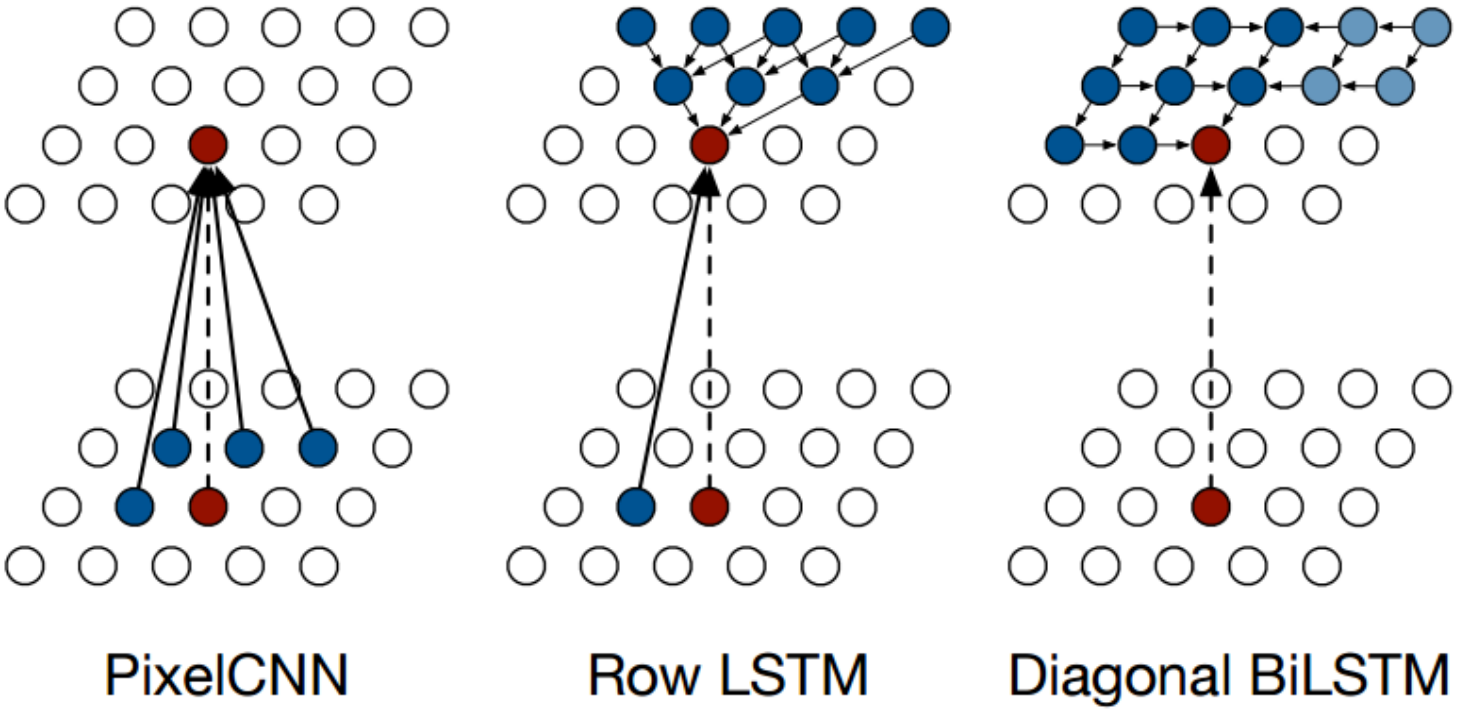
PixelRNNs are 2D extensions of Recurrent Neural Networks.

They can be used as auto-regressive models to generate images, by generating pixels in a particular order, conditioning on neighboring pixels.

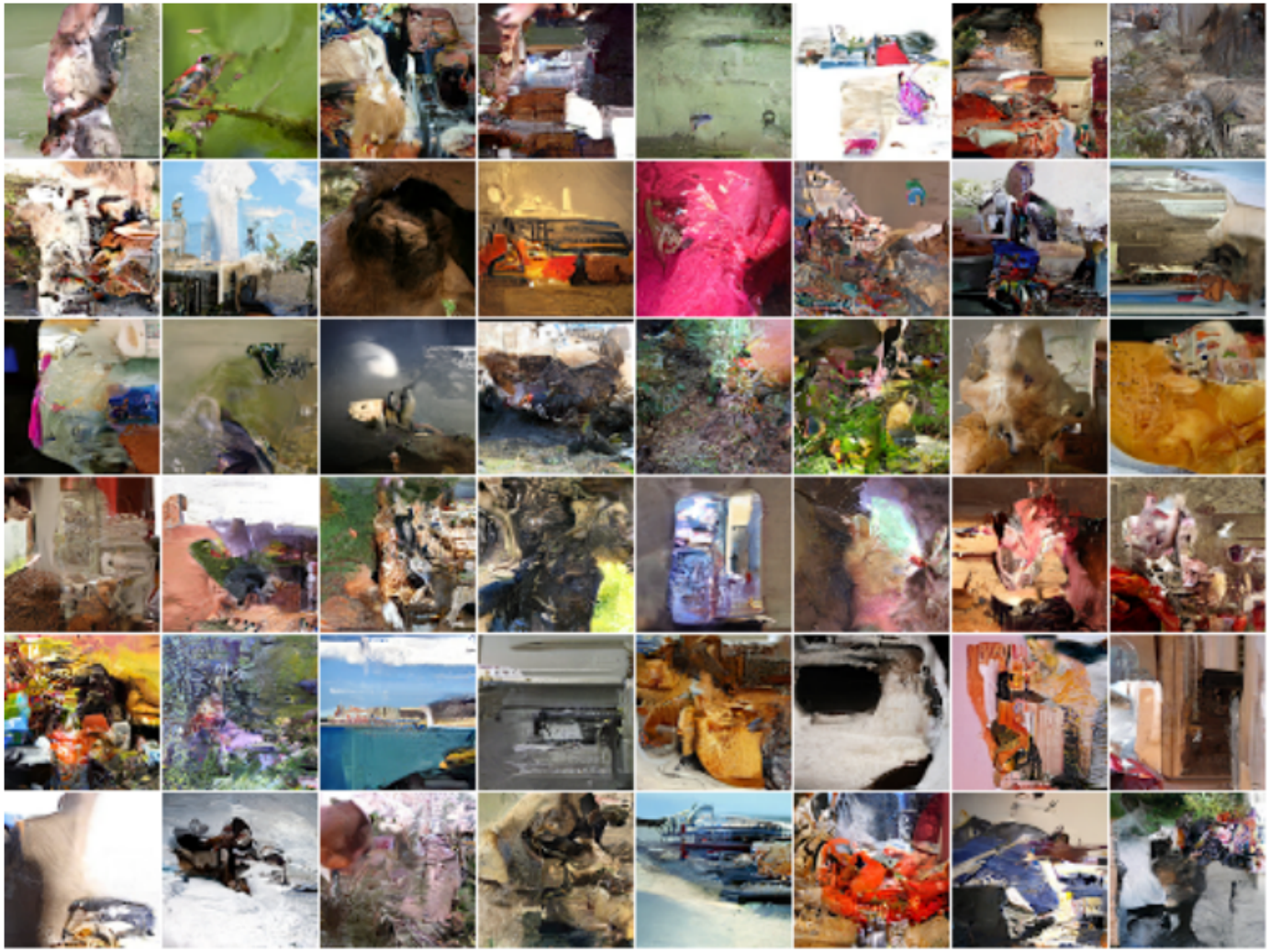
Several variants...

RNNs for Generating Images

- Input-to-state and state-to-state mappings for PixelCNN and two PixelRNN models (Oord et al., 2016):



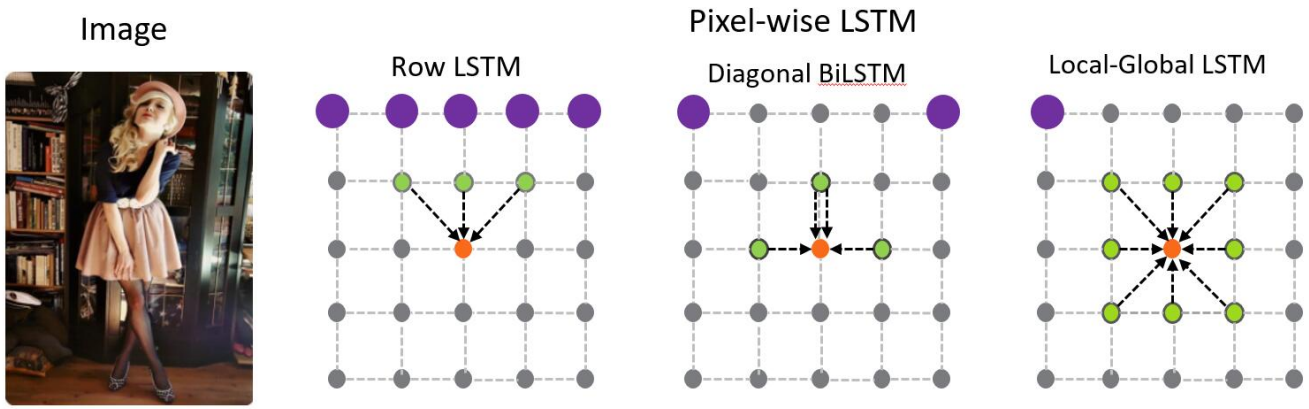
RNNs for Generating Images



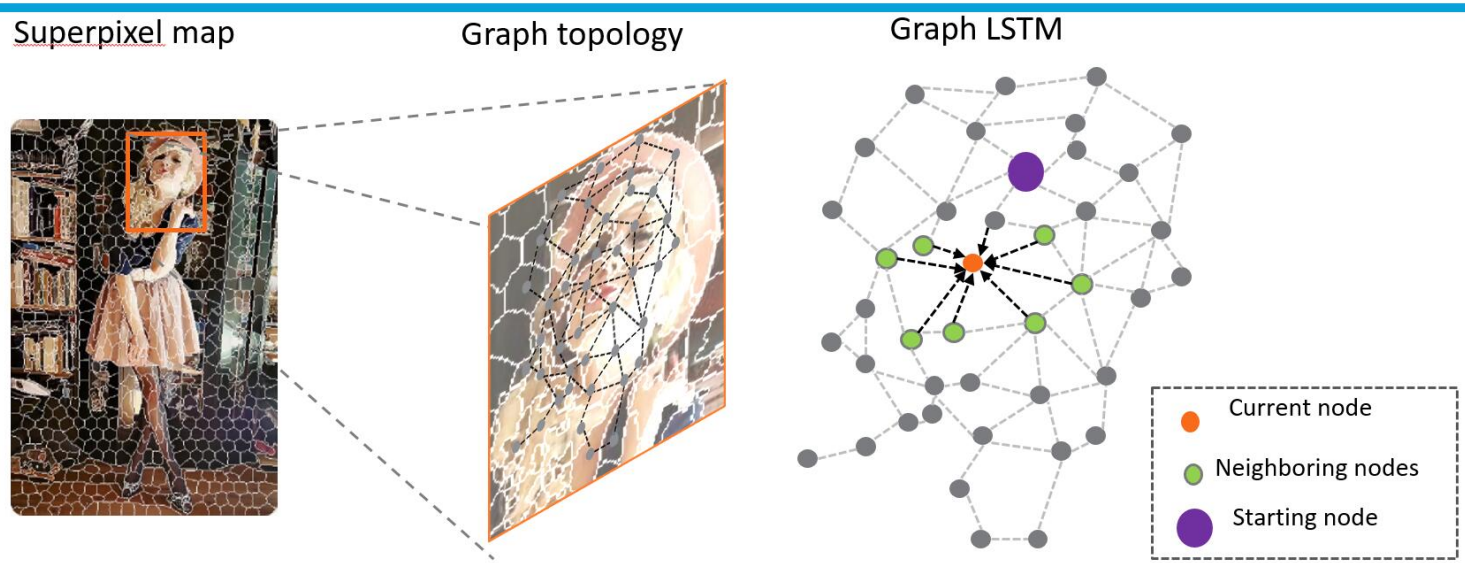
(Oord et al., 2016)

Even More General: Graph LSTMs

Traditional multi-dimensional LSTM:



Graph LSTM:



(Credits: Xiaodan Liang)

Outline

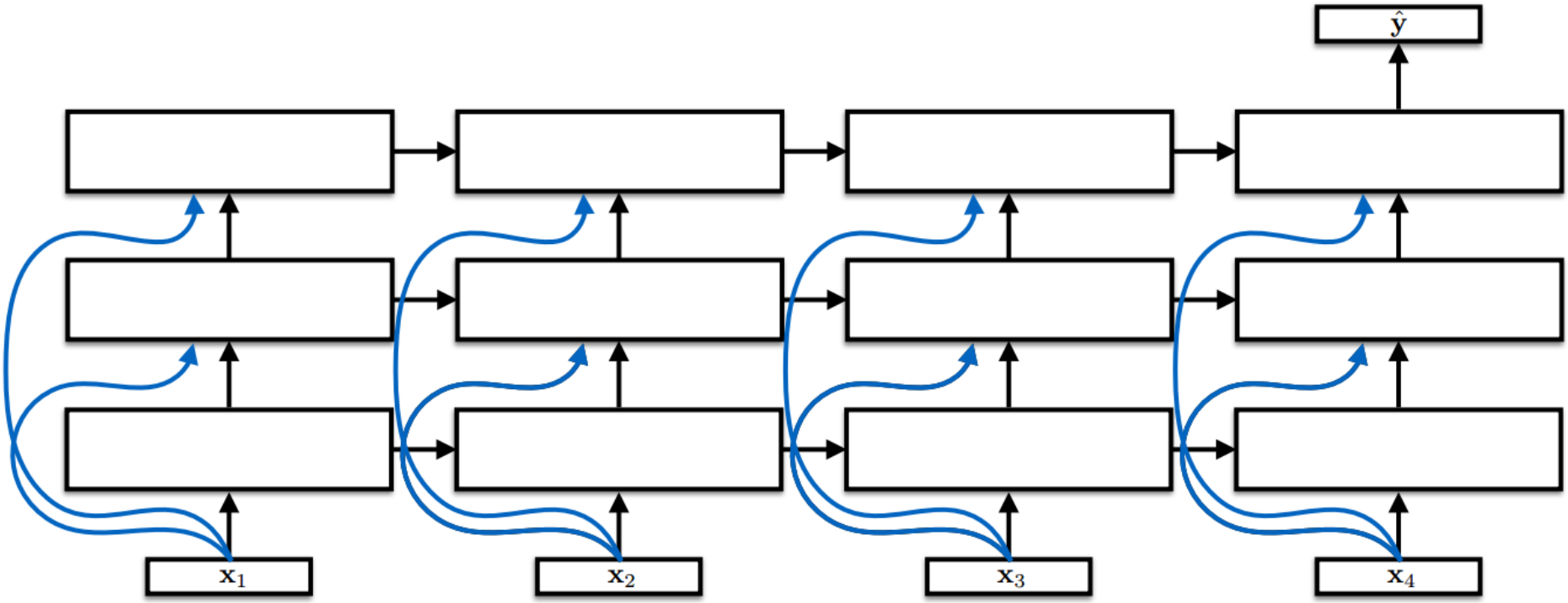
1. Recurrent Neural Networks
   - Sequence Generation
   - Sequence Tagging
   - Pooled Classification
2. The Vanishing Gradient Problem: GRUs and LSTMs
3. Beyond Sequences
   - Recursive Neural Networks
   - Pixel RNNs
4. **Implementation Tricks**
5. Conclusions

More Tricks of the Trade

- Depth
- Dropout
- Implementation Tricks
- Mini-batching

Deep RNNs/LSTMs/GRUs

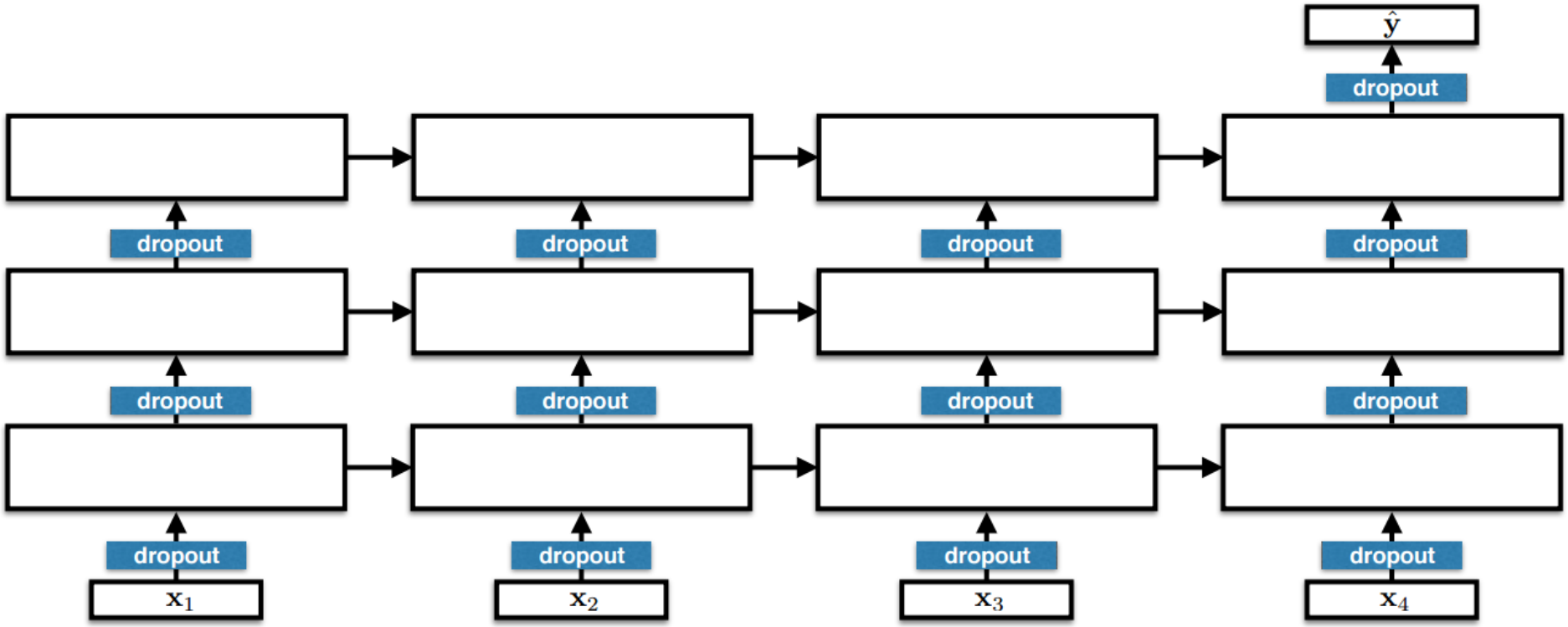
- Depth in recurrent layers helps in practice (2–8 layers seem to be standard)
- Input connections may or may not be used



(Slide credit: Chris Dyer)

Dropout in Deep RNNs/LSTMs/GRUs

- Apply dropout between layers, but not on the recurrent connections
- ... Or use the same mask for all recurrent connections (Gal and Ghahramani, 2015)



(Slide credit: Chris Dyer)

Implementation Tricks

For speed:

- Use diagonal matrices instead of full matrices (esp. for gates)
- Concatenate parameter matrices for all gates and do a single matrix-vector multiplication
- Use optimized implementations (from NVIDIA)
- Use GRUs or reduced-gate variant of LSTMs

For learning speed and performance:

- Initialize so that the bias on the forget gate is large (intuitively: at the beginning of training, the signal from the past is unreliable)
- Use random orthogonal matrices to initialize the square matrices

Mini-Batching

- RNNs, LSTMs, GRUs all consist of lots of elementwise operations (addition, multiplication, nonlinearities), and lots of matrix-vector products
- Mini-batching: convert many matrix-vector products into a single matrix-matrix multiplication
- Batch across instances, not across time
- The challenge with working with mini batches of sequences is... sequences are of different lengths (we've seen this when talking about convolutional nets)
- This usually means you bucket training instances based on similar lengths, and pad with zeros
- Be careful when padding not to back propagate a non-zero value!

Outline

1. Recurrent Neural Networks
   - Sequence Generation
   - Sequence Tagging
   - Pooled Classification
2. The Vanishing Gradient Problem: GRUs and LSTMs
3. Beyond Sequences
   - Recursive Neural Networks
   - Pixel RNNs
4. Implementation Tricks
5. Conclusions

Conclusions

Recurrent neural networks allow to take advantage of sequential input structure

They can be used to generate, tag, and classify sequences, and are trained with backpropagation through time

Vanilla RNNs suffer from vanishing and exploding gradients

LSTMs and other gated units are more complex variants of RNNs that avoid vanishing gradients

They can be extended to other structures like trees, images, and graphs.

Thank you!

Questions?

References I

Brants, T. (2000). Tnt: a statistical part-of-speech tagger. In *Proceedings of the sixth conference on Applied natural language processing*, pages 224–231. Association for Computational Linguistics.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation. In *Proc. of Empirical Methods in Natural Language Processing*.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

Gal, Y. and Ghahramani, Z. (2015). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*.

Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *ICASSP*, pages 6645–6649. IEEE.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. (2016). Neural architectures for named entity recognition. In *Proc. of the Annual Meeting of the North-American Chapter of the Association for Computational Linguistics*.

Ling, W., Luís, T., Marujo, L., Astudillo, R. F., Amir, S., Dyer, C., Black, A. W., and Trancoso, I. (2015). Finding function in form: Compositional character models for open vocabulary word representation. In *Proc. of Empirical Methods in Natural Language Processing*.

Oord, A. v. d., Kalchbrenner, N., and Kavukcuoglu, K. (2016). Pixel Recurrent Neural Networks. In *Proc. of the International Conference on Machine Learning*.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*.

Ratinov, L. and Roth, D. (2009). Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, pages 147–155. Association for Computational Linguistics.

Ratnaparkhi, A. (1999). Learning to parse natural language with maximum entropy models. *Machine Learning*, 34(1):151–175.

References II

Schmidhuber, J., Wierstra, D., and Gomez, F. J. (2005). Evolino: Hybrid neuroevolution/optimal linear search for sequence prediction. In *Proceedings of the 19th International Joint Conferenceon Artificial Intelligence (IJCAI)*.

Socher, R., Lin, C. C., Manning, C., and Ng, A. Y. (2011). Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112.

Toutanova, K., Klein, D., Manning, C. D., and Singer, Y. (2003). Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proc. of the North American Chapter of the Association for Computational Linguistics*, pages 173–180.

Wang, P., Qian, Y., Soong, F. K., He, L., and Zhao, H. (2015). Part-of-speech tagging with bidirectional long short-term memory recurrent neural network. *arXiv preprint arXiv:1510.06168*.

Zhang, T. and Johnson, D. (2003). A robust risk minimization based named entity recognition system. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003-Volume 4*, pages 204–207. Association for Computational Linguistics.