Machine Learning – Lecture 17

Recurrent Neural Networks

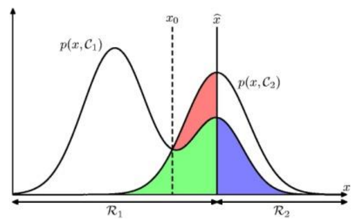
21.01.2019

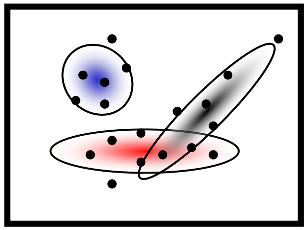
Bastian Leibe
RWTH Aachen
http://www.vision.rwth-aachen.de

leibe@vision.rwth-aachen.de

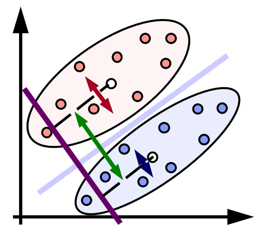
Course Outline

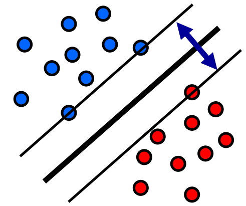
- Fundamentals
  - Bayes Decision Theory
  - Probability Density Estimation

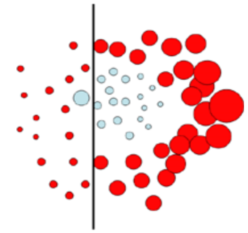


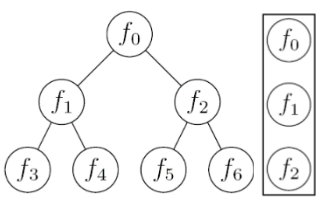


- Classification Approaches
  - Linear Discriminants
  - Support Vector Machines
  - Ensemble Methods & Boosting
  - Random Forests

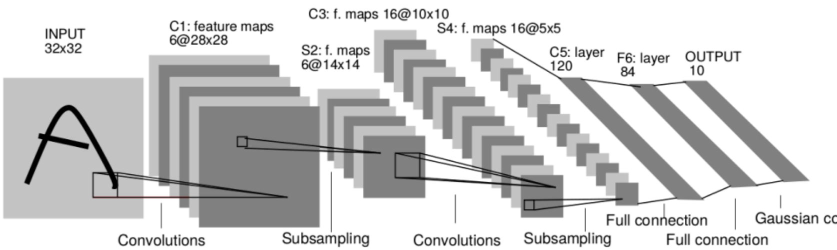




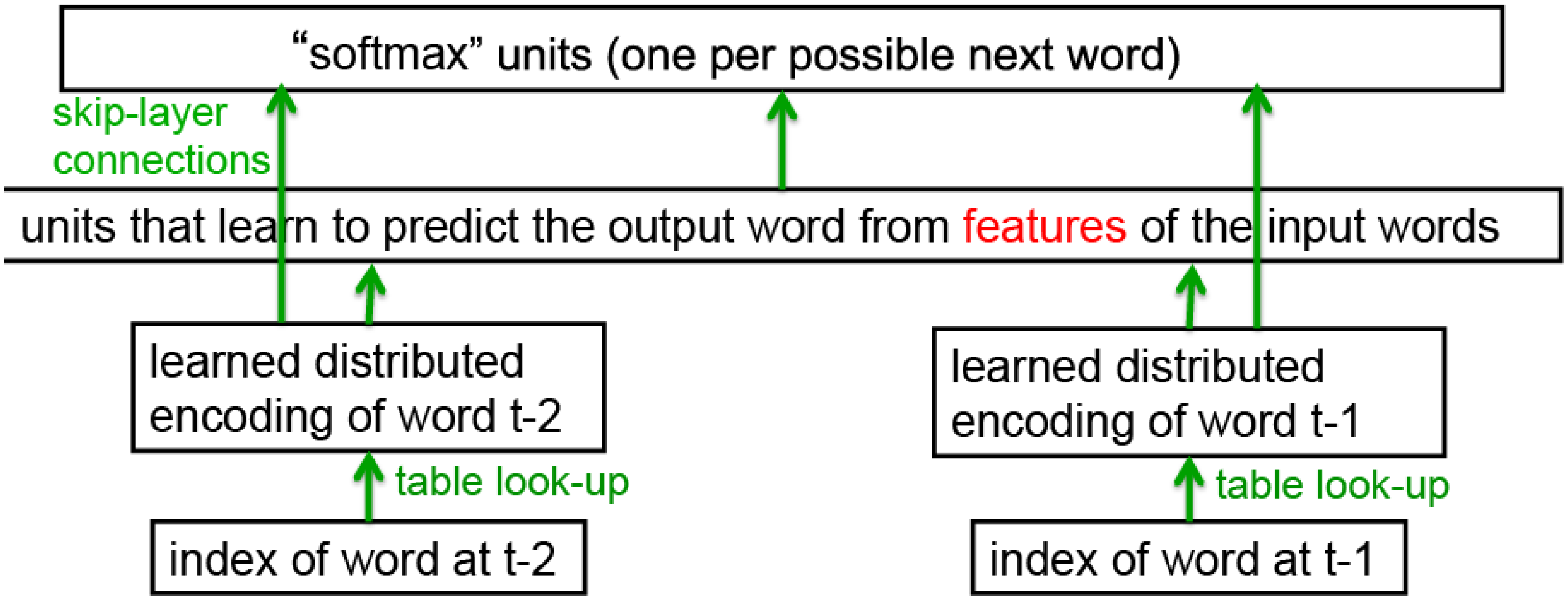




- Deep Learning
  - Foundations
  - Convolutional Neural Networks
  - Recurrent Neural Networks



Recap: Neural Probabilistic Language Model



- Core idea
  - Learn a shared distributed encoding (word embedding) for the words in the vocabulary.

Y. Bengio, R. Ducharme, P. Vincent, C. Jauvin, A Neural Probabilistic Language Model, In JMLR, Vol. 3, pp. 1137-1155, 2003.

Recap: word2vec

- Goal
  - Make it possible to learn high-quality word embeddings from huge data sets (billions of words in training set).
- Approach
  - Define two alternative learning tasks for learning the embedding:
    - “Continuous Bag of Words” (CBOW)
    - “Skip-gram”
  - Designed to require fewer parameters.

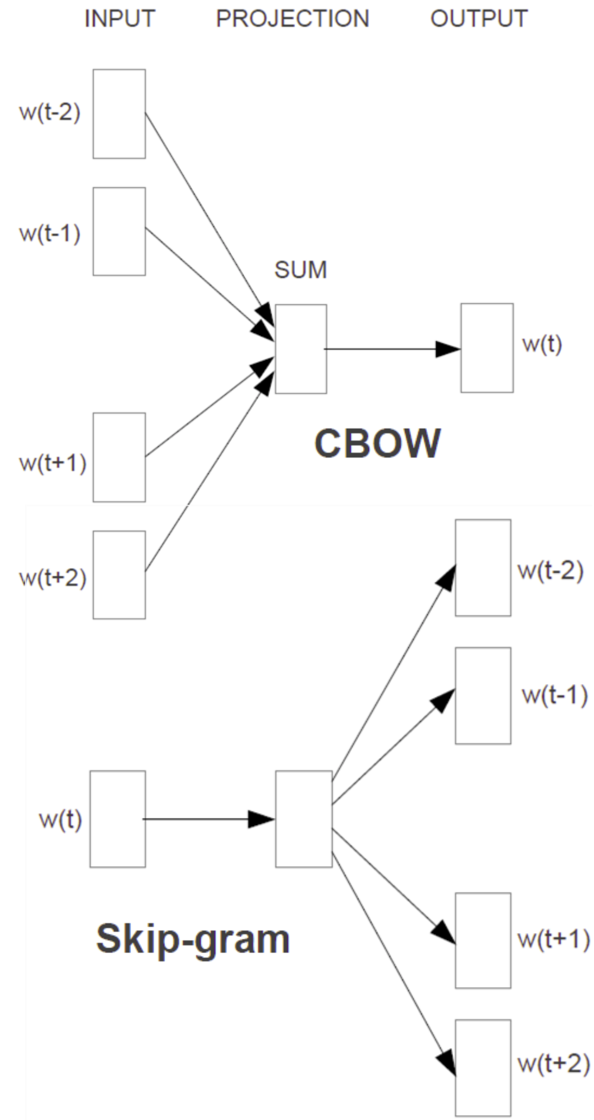


Image source: Mikolov et al., 2015

Recap: word2vec CBOW Model

- Continuous BOW Model
  - Remove the non-linearity from the hidden layer
  - Share the projection layer for all words (their vectors are averaged)

$\Rightarrow$ Bag-of-Words model (order of the words does not matter anymore)

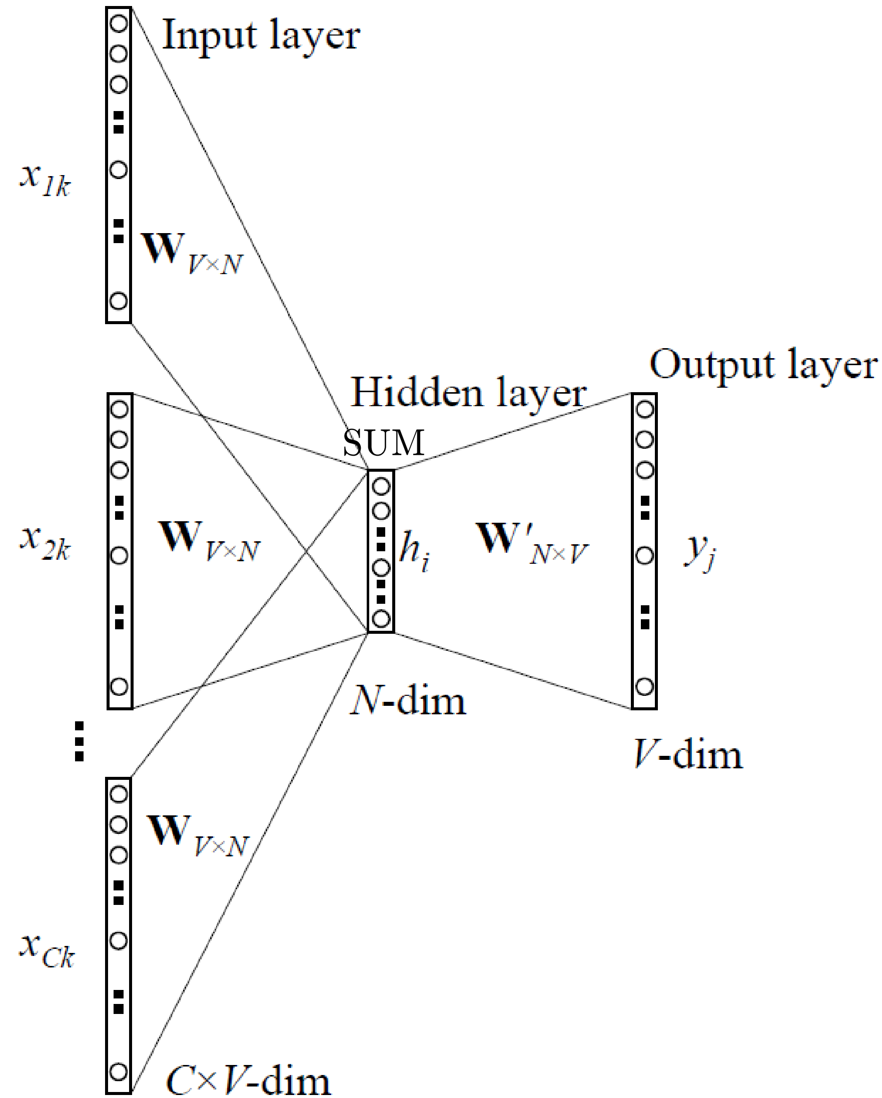


Image source: Xin Rong, 2015

Recap: word2vec Skip-Gram Model

- Continuous Skip-Gram Model
  - Similar structure to CBOW
  - Instead of predicting the current word, predict words within a certain range of the current word.
  - Give less weight to the more distant words

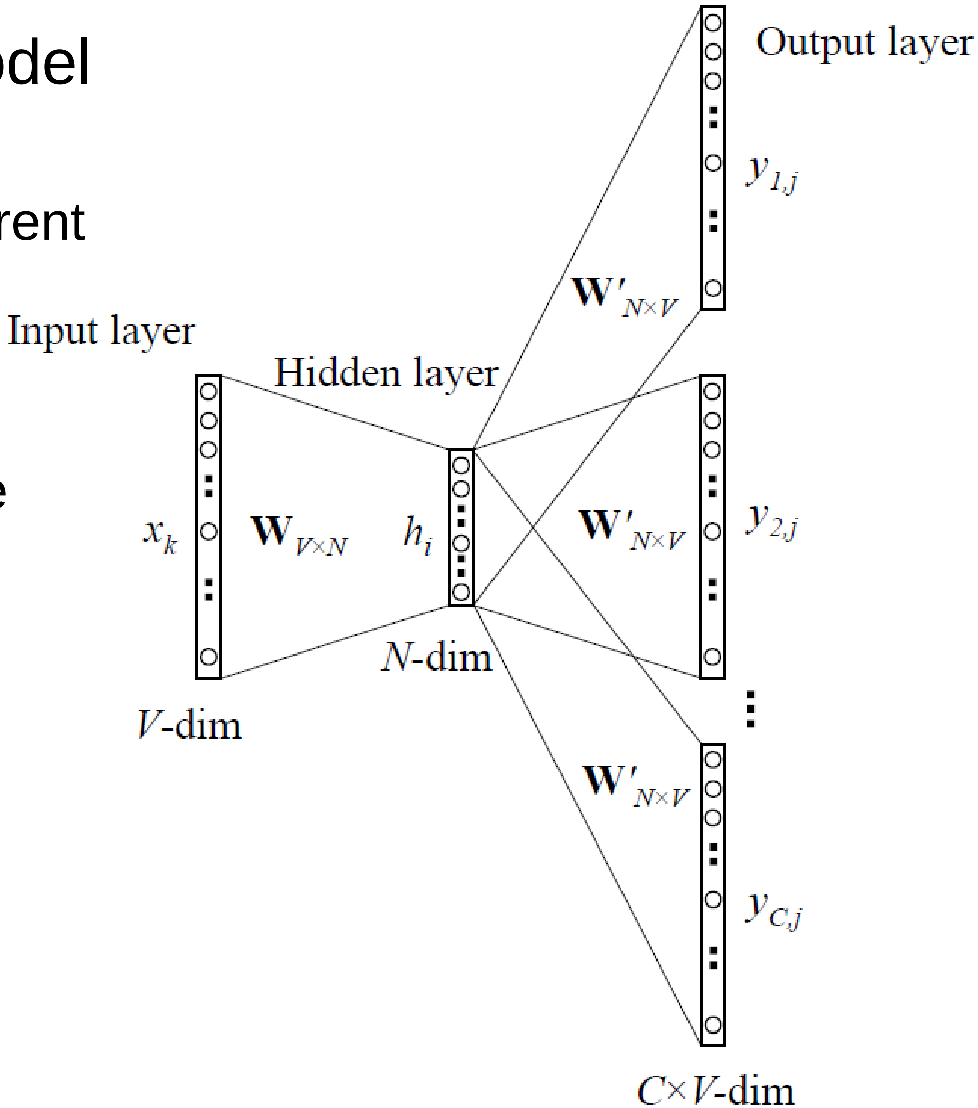


Image source: Xin Rong, 2015

Problems with 100k-1M outputs

- Weight matrix gets huge!
  - Example: CBOW model
  - One-hot encoding for inputs
  - $\Rightarrow$ Input-hidden connections are just vector lookups.
  - This is not the case for the hidden-output connections!
  - State h is not one-hot, and vocabulary size is 1M.
  - $\Rightarrow$ $\mathbf{W}'_{N\times V}$ has 300$\times$1M entries
- Softmax gets expensive!
  - Need to compute normalization over 100k-1M outputs

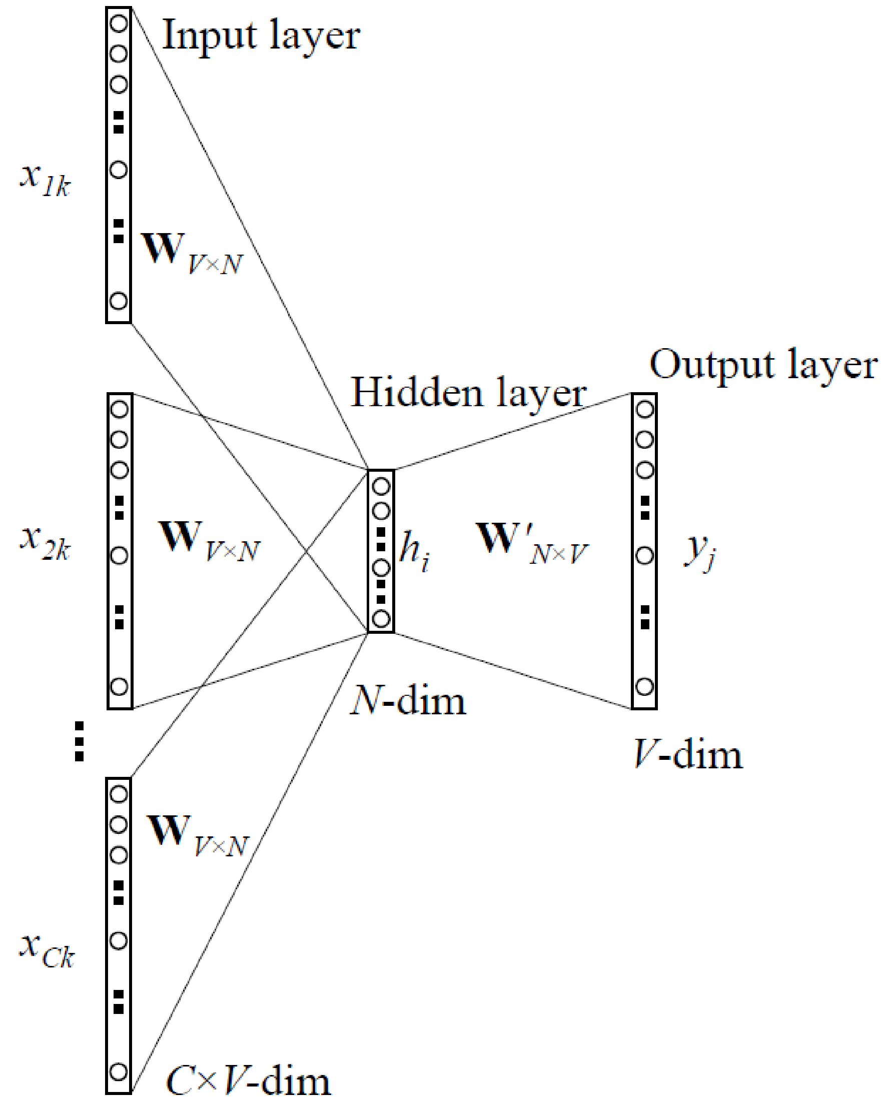
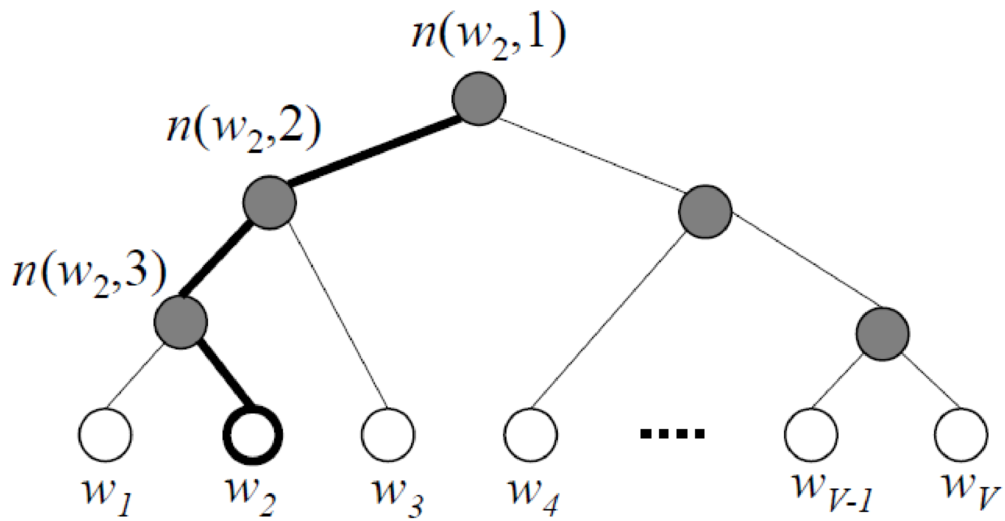


Image source: Xin Rong, 2015

Solution: Hierarchical Softmax



- Idea
  - Organize words in binary search tree, words are at leaves
  - Factorize probability of word w_0 as a product of node probabilities along the path.
  - Learn a linear decision function $y = v_{n(w,j)} \cdot h$ at each node to decide whether to proceed with left or right child node.

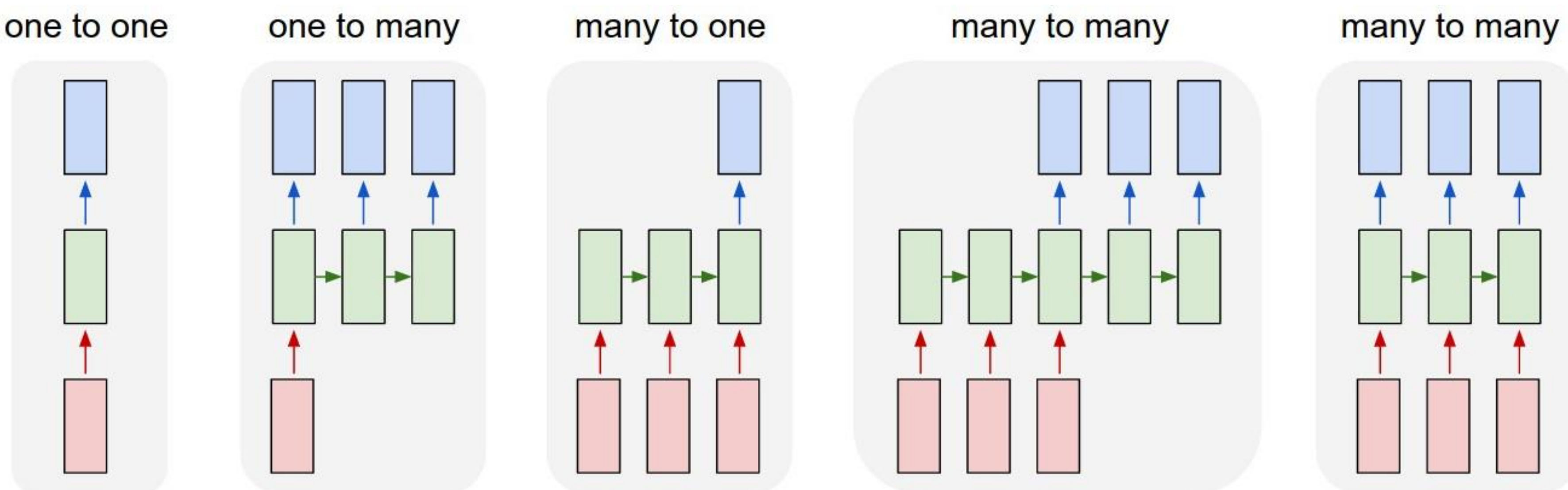
  $\Rightarrow$ Decision based on output vector of hidden units directly.

Image source: Xin Rong, 2015

Topics of This Lecture

- Recurrent Neural Networks (RNNs)
  - Motivation
  - Intuition
- Learning with RNNs
  - Formalization
  - Comparison of Feedforward and Recurrent networks
  - Backpropagation through Time (BPTT)
- Problems with RNN Training
  - Vanishing Gradients
  - Exploding Gradients
  - Gradient Clipping

Recurrent Neural Networks



- Up to now
  - Simple neural network structure: 1-to-1 mapping of inputs to outputs
- This lecture: Recurrent Neural Networks
  - Generalize this to arbitrary mappings

Image source: Andrej Karpathy

Application: Part-of-Speech Tagging

Legend: Click the legend words to toggle highlighting. Get help on this page.

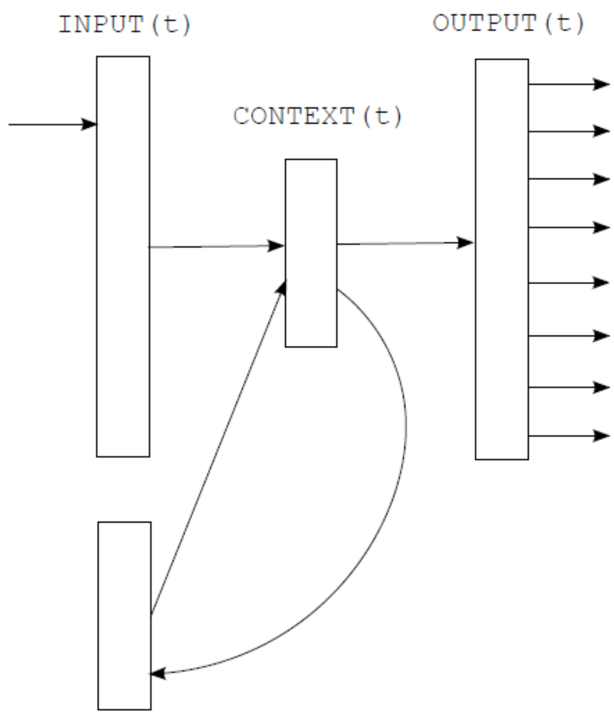
Noun Pronoun Verb Adjective Adverb Conjunction Preposition Article Interjection

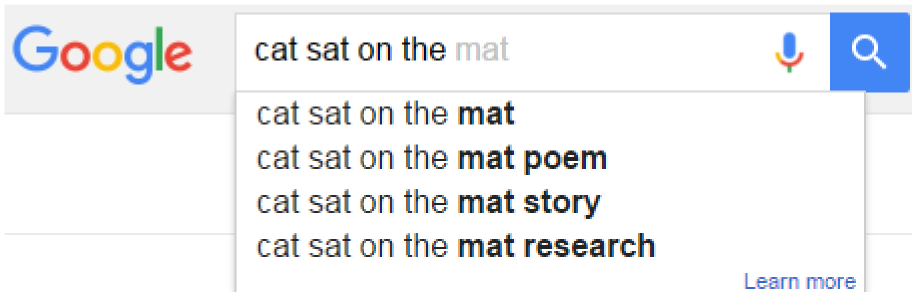
---

Andrew and Maria thought their jobs were secure after the rancorous argument with the customer , but alas ! Bad news is fast approaching them , especially after they viciously insulted the customer on social media .

Image source: http://rewordify.com

Application: Predicting the Next Word

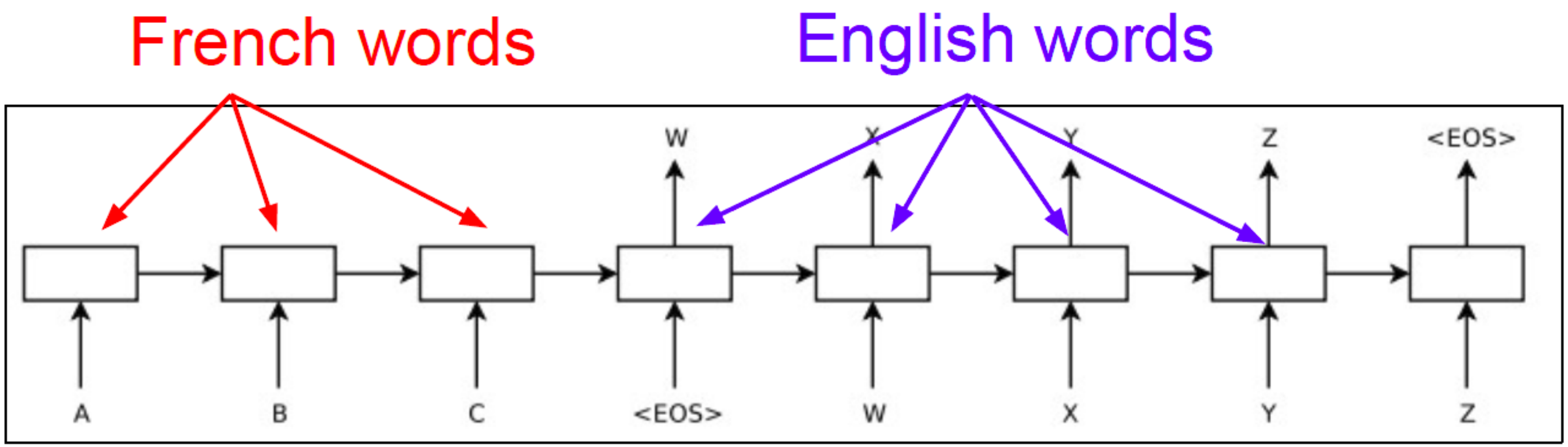




T. Mikolov, M. Karafiat, L. Burget, J. Cernocky, S. Khudanpur, Recurrent Neural Network Based Language Model, Interspeech 2010.

Image source: Mikolov et al., 2010

Application: Machine Translation



I. Sutskever, O. Vinyals, Q. Le, Sequence to Sequence Learning with Neural Networks, NIPS 2014.

RNNs: Intuition

- Example: Language modeling
  - Suppose we had the training sequence "cat sat on mat"
  - We want to train a language model

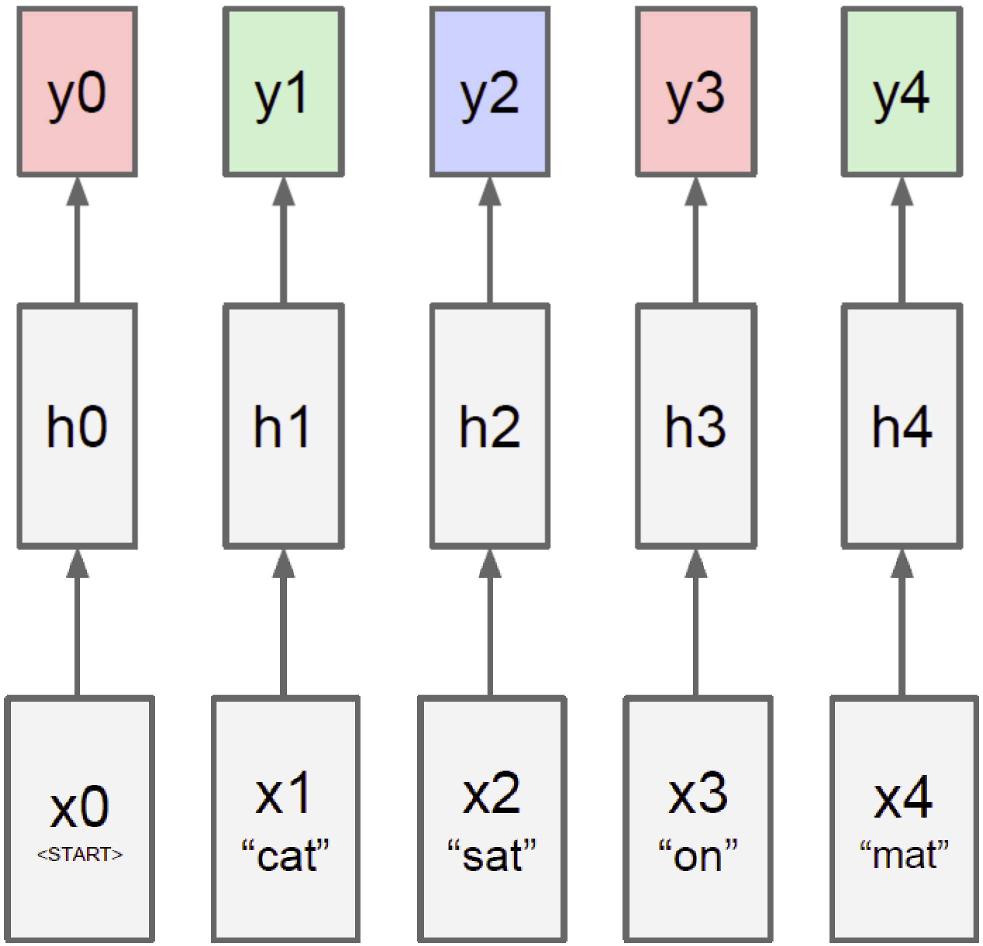
$$p(\textit{next word} \mid \textit{previous words})$$

  - First assume we only have a finite, 1-word history.
  - I.e., we want those probabilities to be high:
    - $p(cat \mid <S>)$
    - $p(sat \mid cat)$
    - $p(on \mid sat)$
    - $p(mat \mid on)$
    - $p(<E> \mid mat)$

$< S >$ and $< E >$ are start and end tokens.

RNNs: Intuition

- Vanilla 2-layer classification net



10,001D class scores (Softmax over 10k words and a special <END> token)

$$\mathbf{y}_4 = \mathbf{W}_{hy}\mathbf{h}_4$$

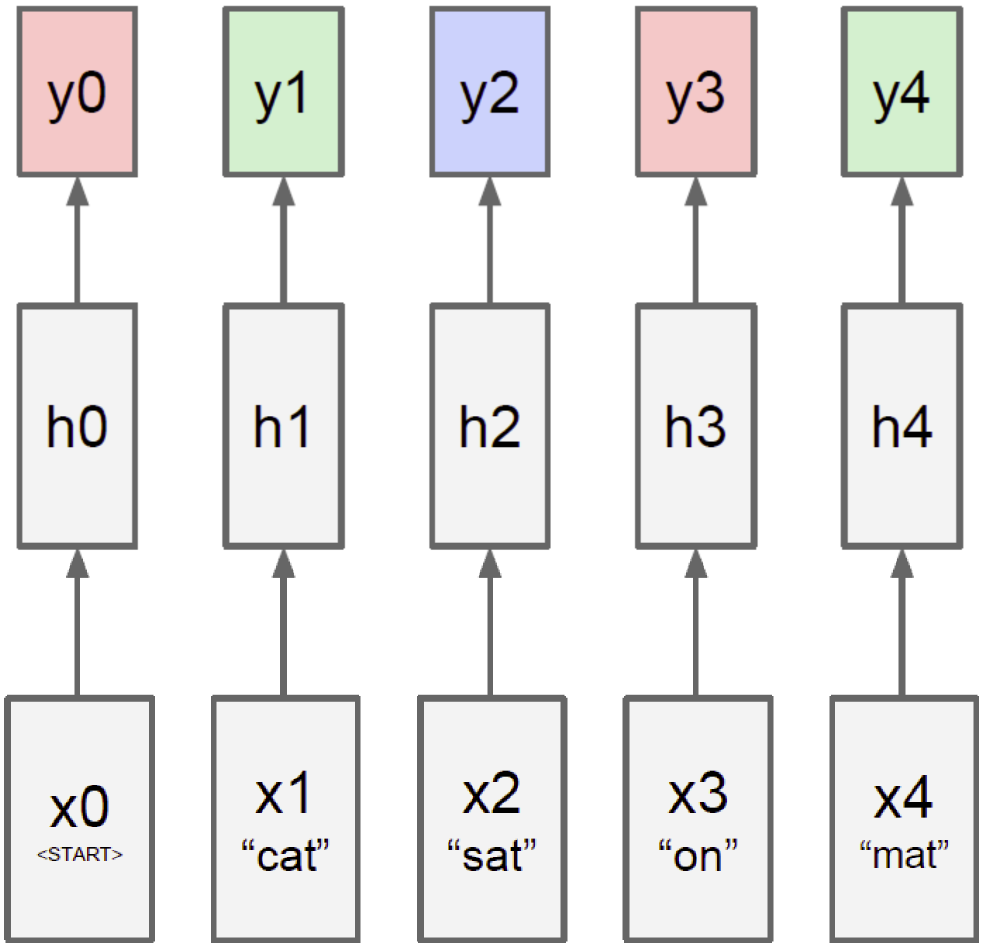
Hidden layer (e.g., 500D vectors)

$$\mathbf{h}_4 = \max\{0, \mathbf{W}_{xh}\mathbf{x}_4\}$$

Word embedding (300D vector for each word)

RNNs: Intuition

- Turning this into an RNN (wait for it...)



10,001D class scores (Softmax over 10k words and a special <END> token)

$$\mathbf{y}_4 = \mathbf{W}_{hy}\mathbf{h}_4$$

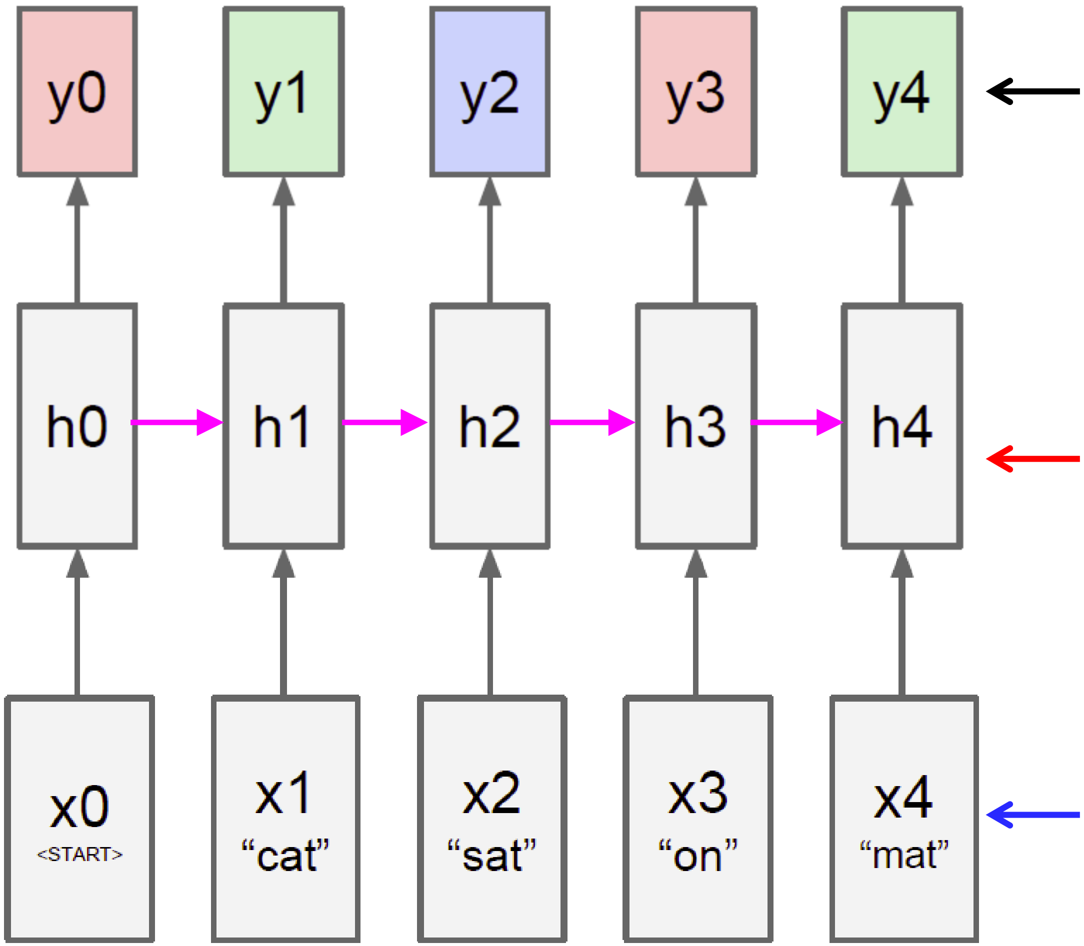
Hidden layer (e.g., 500D vectors)

$$\mathbf{h}_4 = \max\{0, \mathbf{W}_{xh}\mathbf{x}_4\}$$

Word embedding (300D vector for each word)

RNNs: Intuition

- Turning this into an RNN (done!)



10,001D class scores (Softmax over 10k words and a special <END> token)

$$\mathbf{y}_4 = \mathbf{W}_{hy}\mathbf{h}_4$$

Hidden layer (e.g., 500D vectors)

$$\mathbf{h}_4 = \max\{0, \mathbf{W}_{xh}\mathbf{x}_4 + \mathbf{W}_{hh}\mathbf{h}_3\}$$

Word embedding (300D vector for each word)

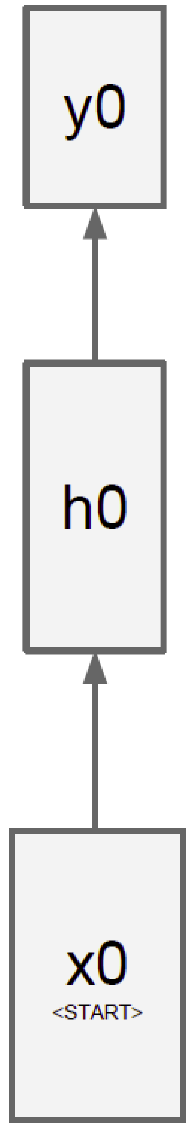
Image source: Andrej Karpathy

RNNs: Intuition

- Training this on a lot of sentences would give us a language model.

- I.e., a way to predict

$$p(\textit{next word} \mid \textit{previous words})$$

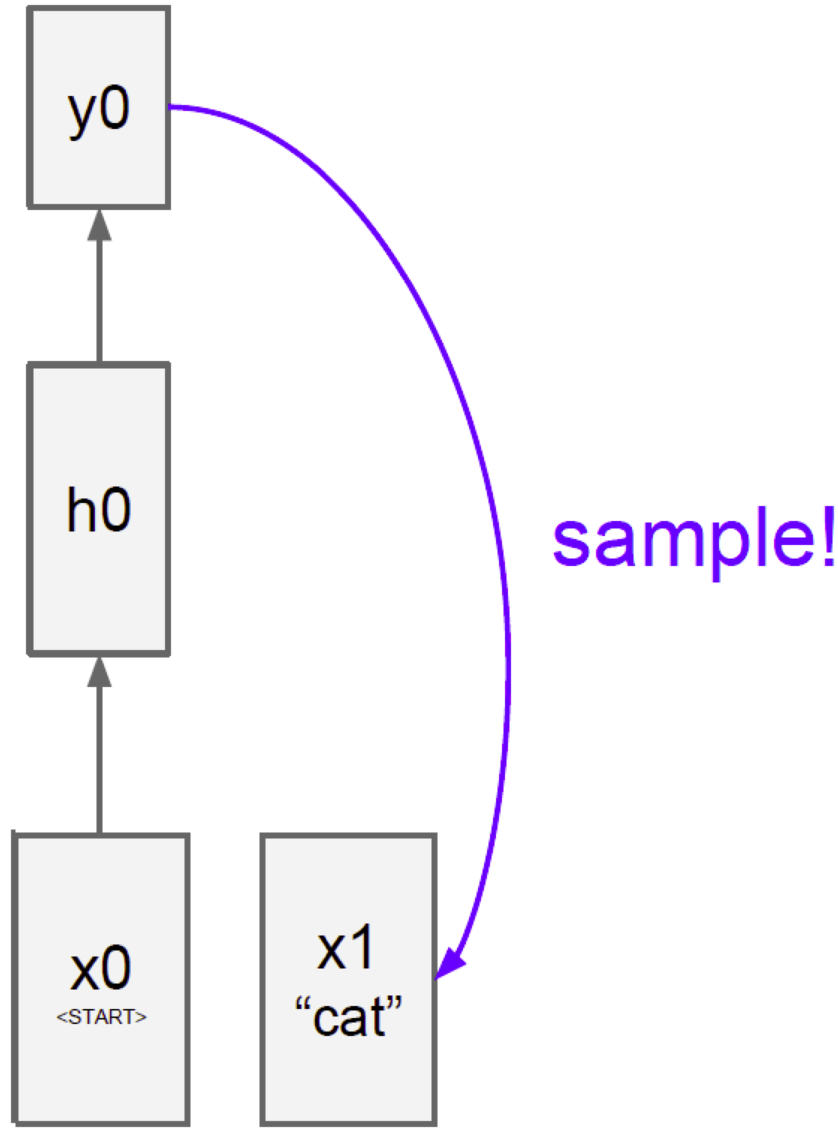


RNNs: Intuition

- Training this on a lot of sentences would give us a language model.

- I.e., a way to predict

$p(\textcolor{red}{next\ word} \mid \textcolor{blue}{previous\ words})$

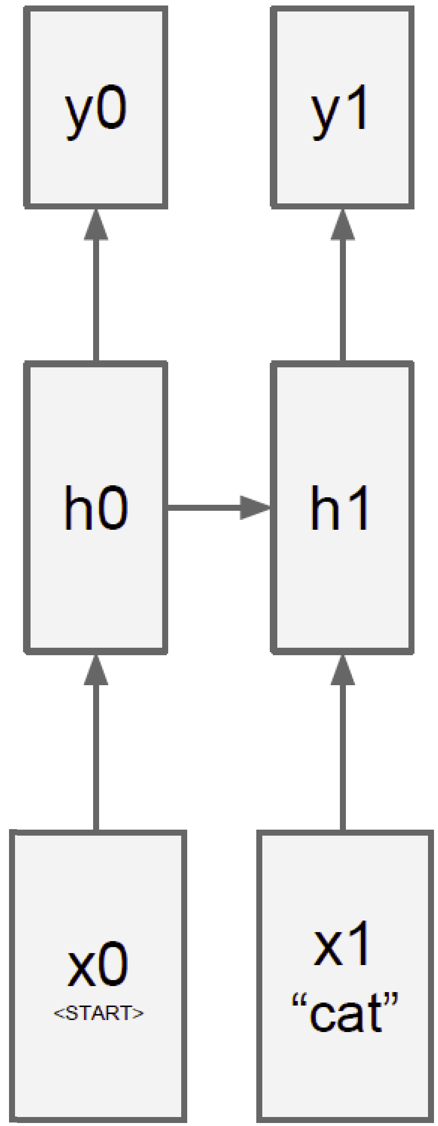


RNNs: Intuition

- Training this on a lot of sentences would give us a language model.

- I.e., a way to predict

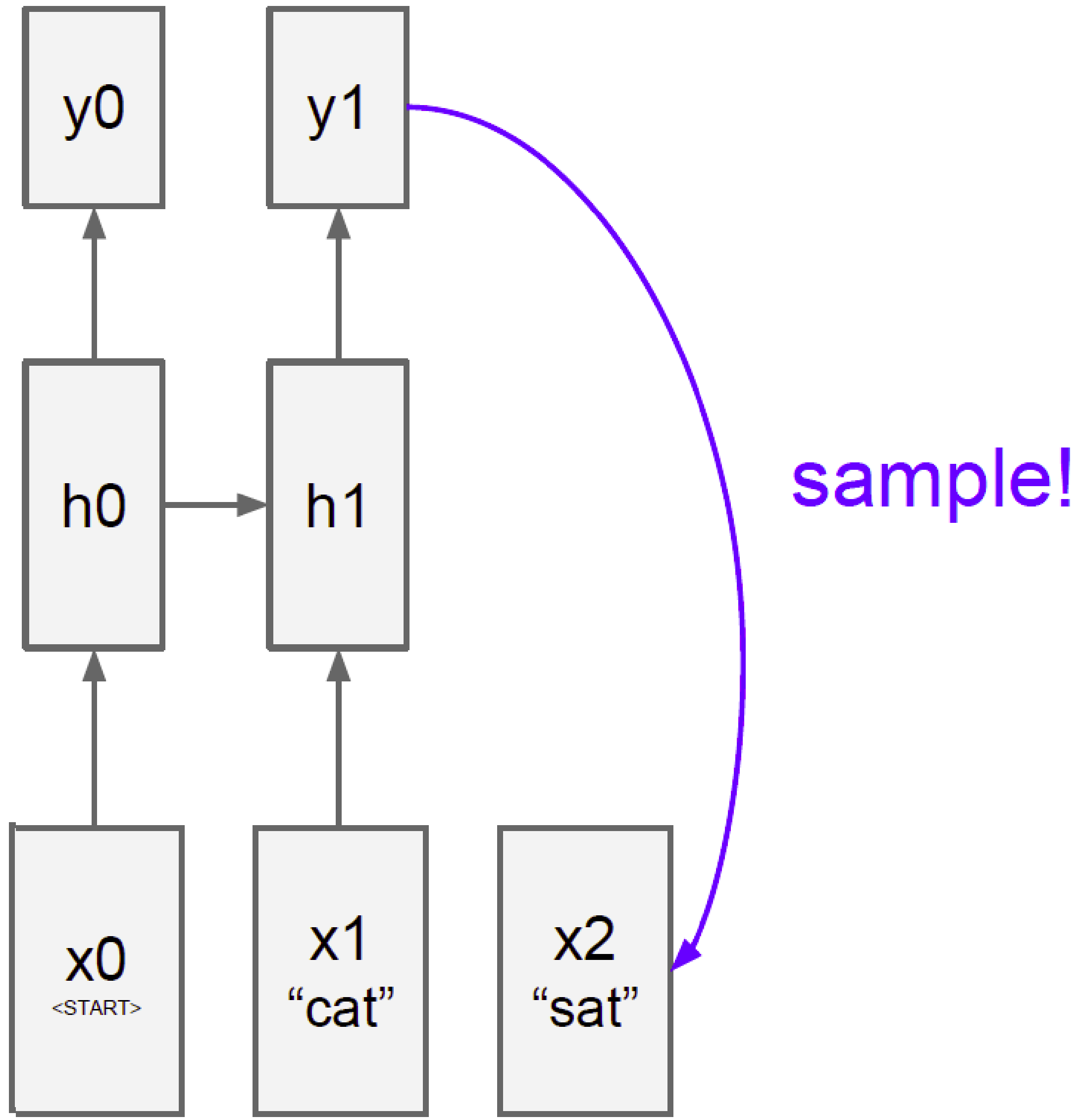
$$p(\color{red}{next\ word} \mid \color{blue}{previous\ words})$$



RNNs: Intuition

- Training this on a lot of sentences would give us a language model.
- I.e., a way to predict

$$p(\textit{next word} \mid \textit{previous words})$$



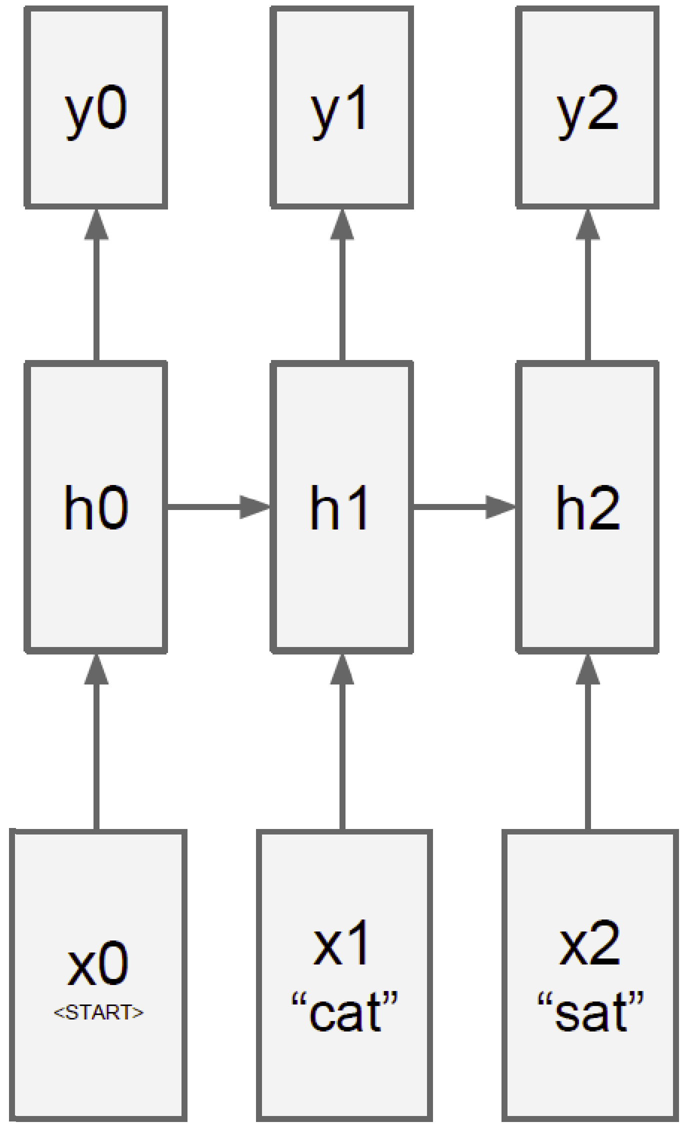
Slide credit: Andrej Karpathy, Fei-Fei Li

RNNs: Intuition

- Training this on a lot of sentences would give us a language model.

- I.e., a way to predict

  $p(\textit{next word} \mid \textit{previous words})$

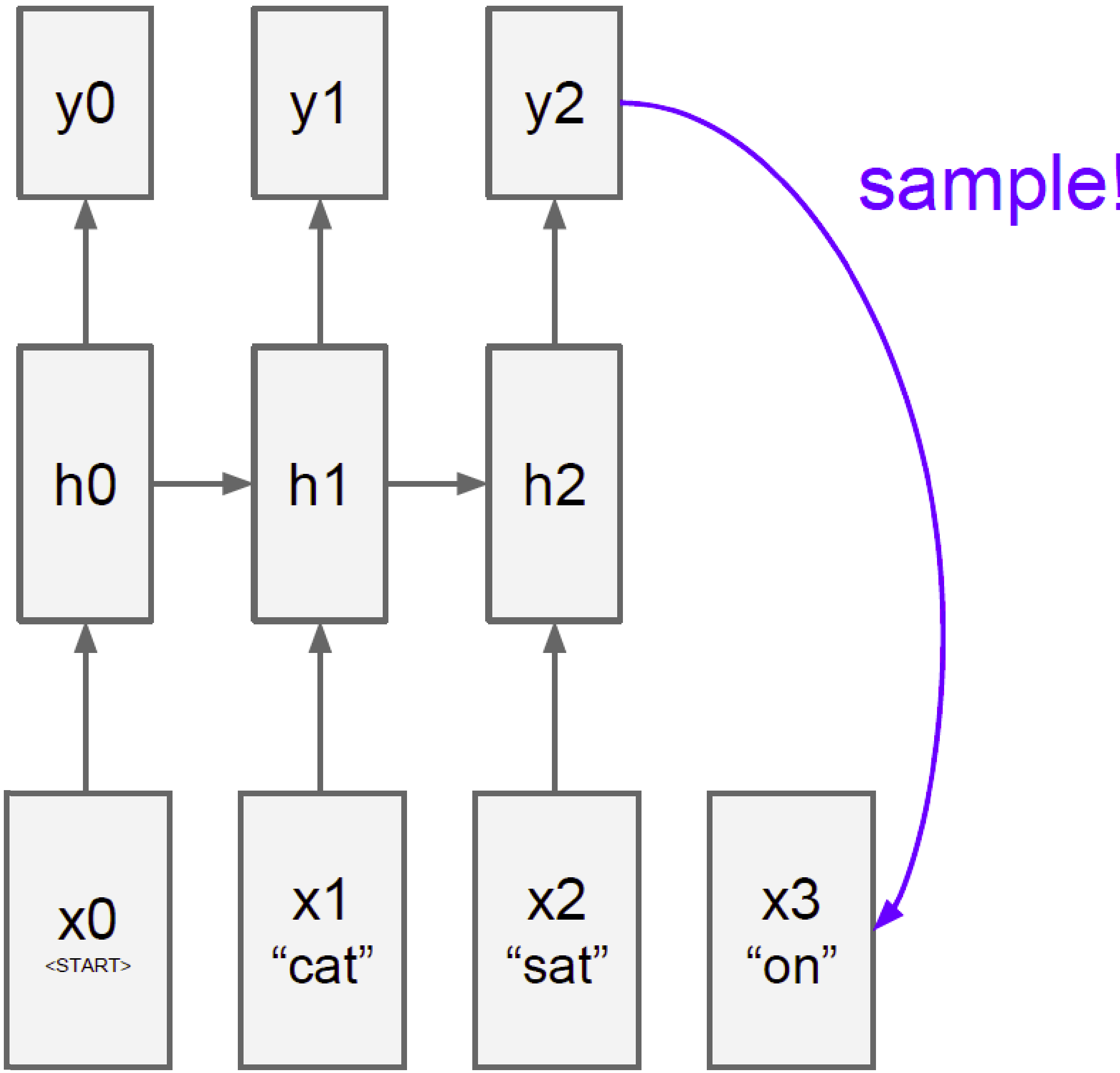


RNNs: Intuition

- Training this on a lot of sentences would give us a language model.

- I.e., a way to predict

$$p(\textit{next word} \mid \textit{previous words})$$

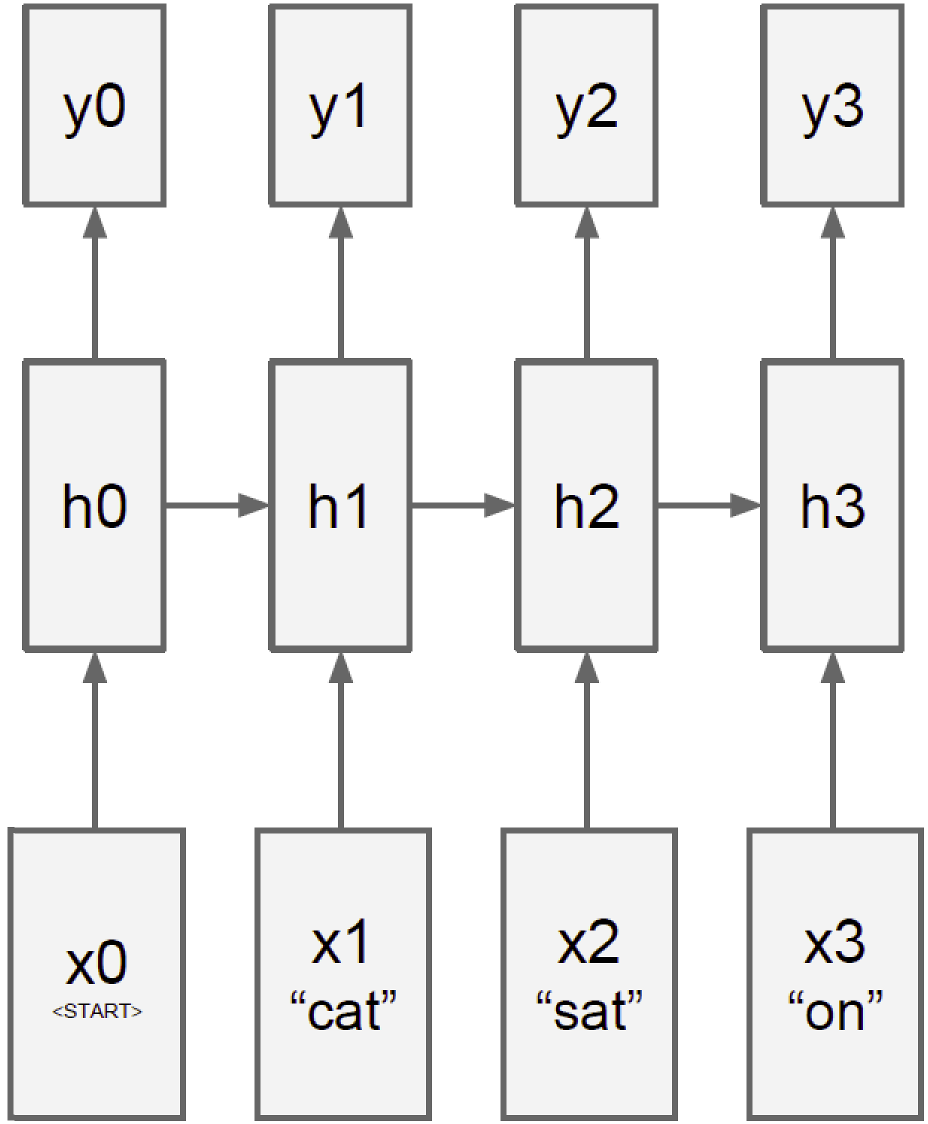


RNNs: Intuition

- Training this on a lot of sentences would give us a language model.

- I.e., a way to predict

$$p(\textit{next word} \mid \textit{previous words})$$



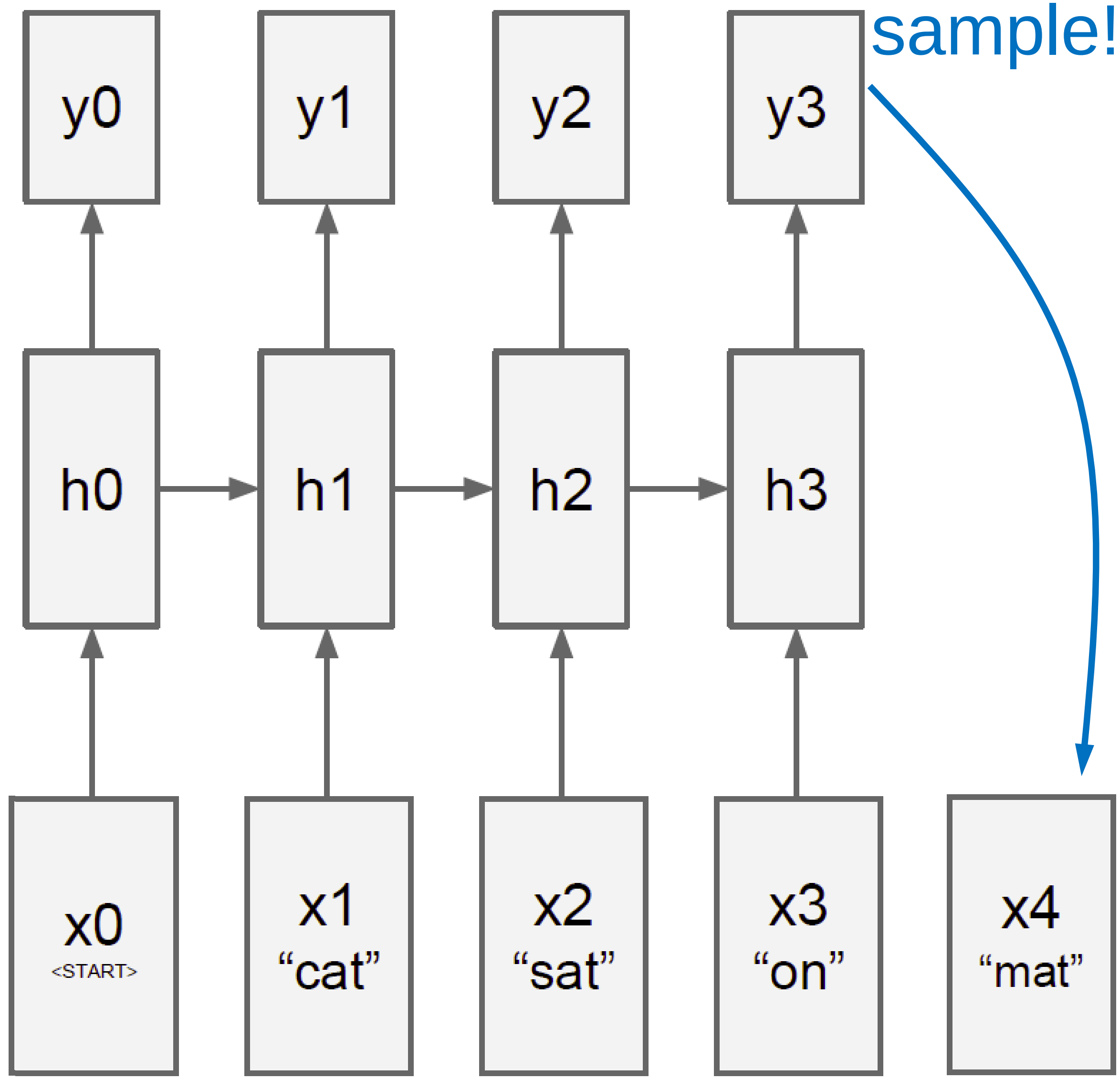
Slide credit: Andrej Karpathy, Fei-Fei Li

RNNs: Intuition

- Training this on a lot of sentences would give us a language model.

- I.e., a way to predict

  $p(\textcolor{red}{next\ word} \mid \textcolor{blue}{previous\ words})$

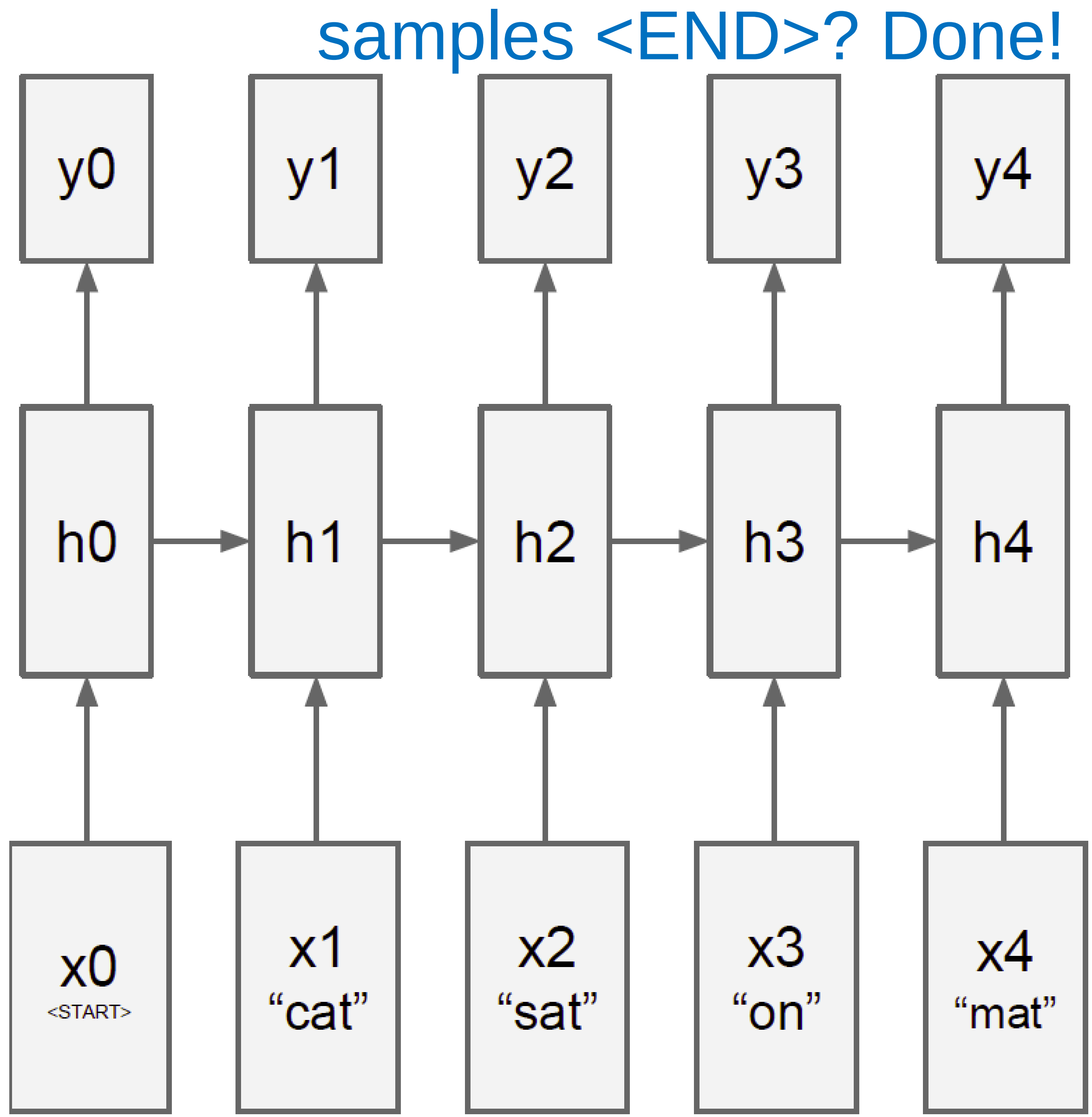


RNNs: Intuition

- Training this on a lot of sentences would give us a language model.

- I.e., a way to predict

$$p(\textit{next word} \mid \textit{previous words})$$



Slide credit: Andrej Karpathy, Fei-Fei Li

Topics of This Lecture

- Recurrent Neural Networks (RNNs)
  - Motivation
  - Intuition
- Learning with RNNs
  - Formalization
  - Comparison of Feedforward and Recurrent networks
  - Backpropagation through Time (BPTT)
- Problems with RNN Training
  - Vanishing Gradients
  - Exploding Gradients
  - Gradient Clipping

RNNs: Introduction

- RNNs are regular NNs whose hidden units have additional forward connections over time.
  - You can unroll them to create a network that extends over time.
  - When you do this, keep in mind that the weights for the hidden units are shared between temporal layers.

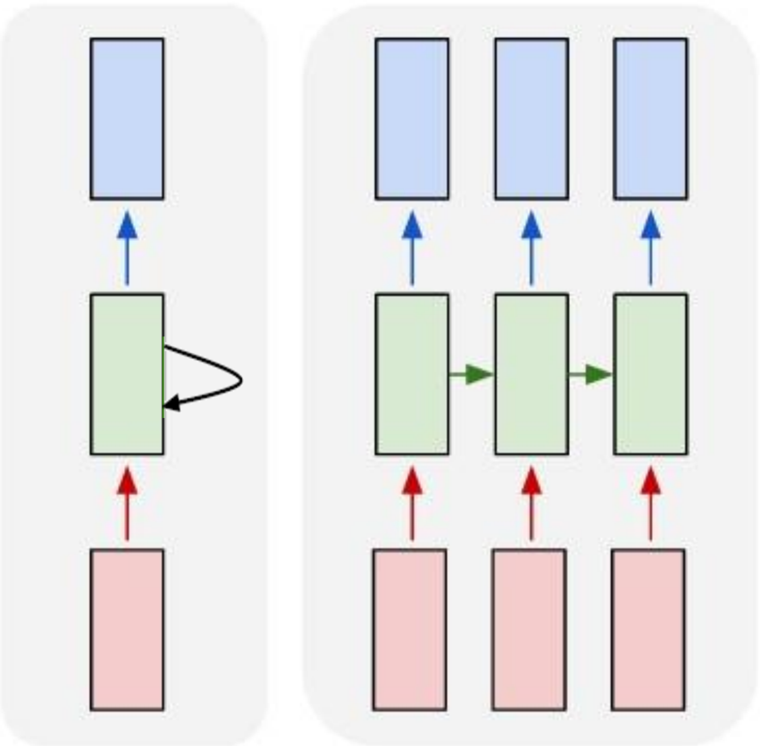
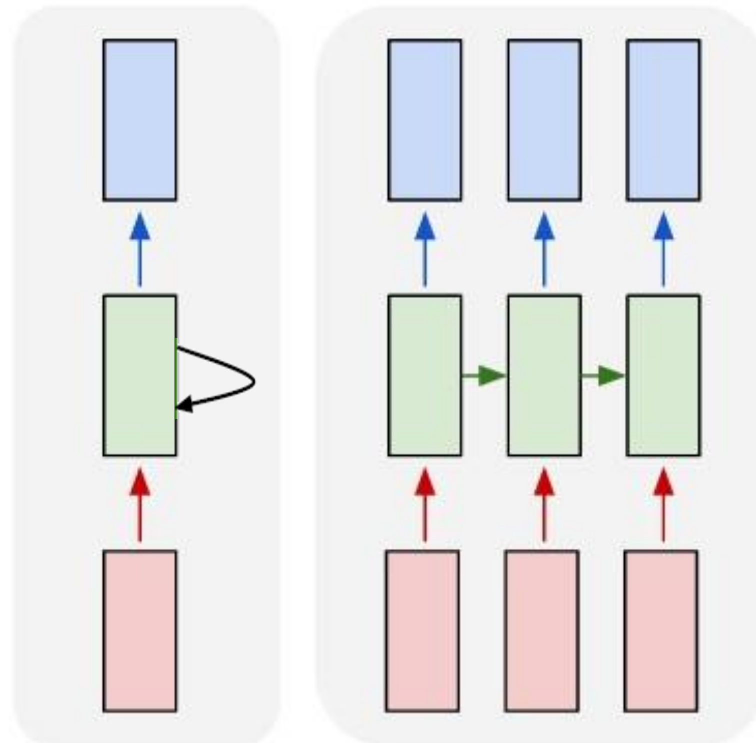


Image source: Andrej Karpathy

RNNs: Introduction

- RNNs are very powerful, because they combine two properties:
  - Distributed hidden state that allows them to store a lot of information about the past efficiently.
  - Non-linear dynamics that allows them to update their hidden state in complicated ways.

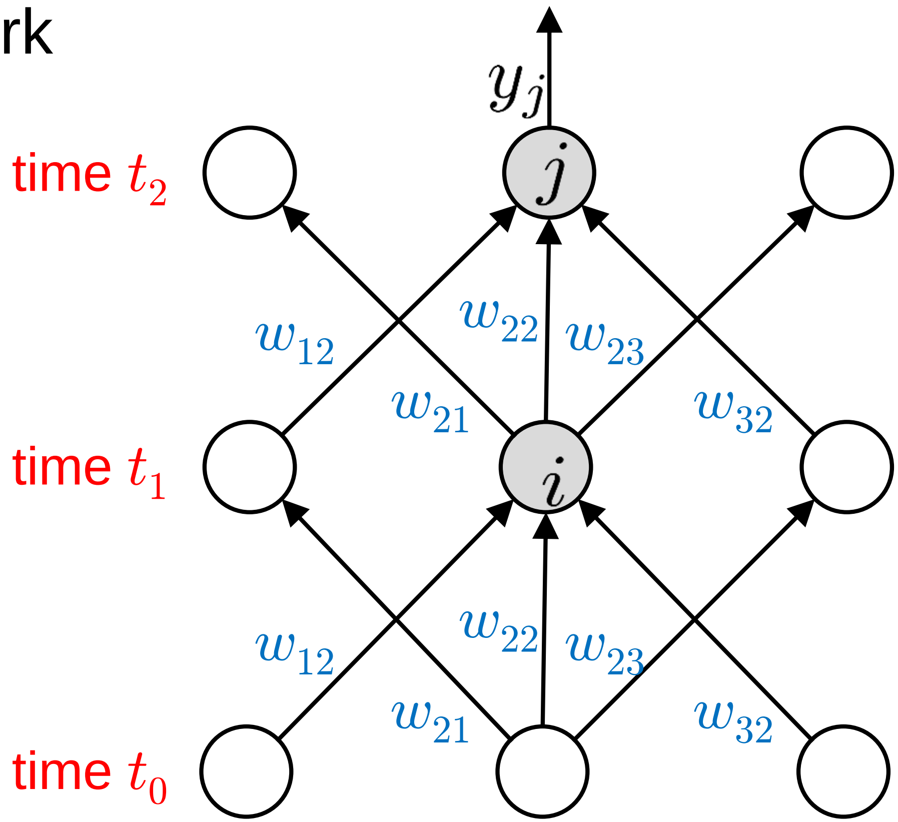


- With enough neurons and time, RNNs can compute anything that can be computed by your computer.

Feedforward Nets vs. Recurrent Nets

- Imagine a feedforward network
  - Assume there is a time delay of 1 in using each connection.

  ⇒ This is very similar to how an RNN works.
  - Only change: the layers share their weights.



⇒ The recurrent net is just a feedforward net that keeps reusing the same weights.

Backpropagation with Weight Constraints

- It is easy to modify the backprop algorithm to incorporate linear weight constraints
  - To constrain $w_1 = w_2$, we start with the same initialization and then make sure that the gradients are the same:

    $$\nabla w_1 = \nabla w_2$$

  - We compute the gradients as usual and then use

    $$\frac{\partial E}{\partial w_1} + \frac{\partial E}{\partial w_2}$$

    for both w_1 and w_2.

Backpropagation Through Time (BPTT)

- Formalization
  - Inputs $\mathbf{x}_t$
  - Outputs $\mathbf{y}_t$
  - Hidden units $\mathbf{h}_t$
  - Initial state $\mathbf{h}_0$
  - Connection matrices
    - $\mathbf{W}_{xh}$
    - $\mathbf{W}_{hy}$
    - $\mathbf{W}_{hh}$
  - Configuration

$$\mathbf{h}_t = \sigma\left(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + b\right)$$

$$\hat{\mathbf{y}}_t = \mathrm{softmax}\left(\mathbf{W}_{hy}\mathbf{h}_t\right)$$

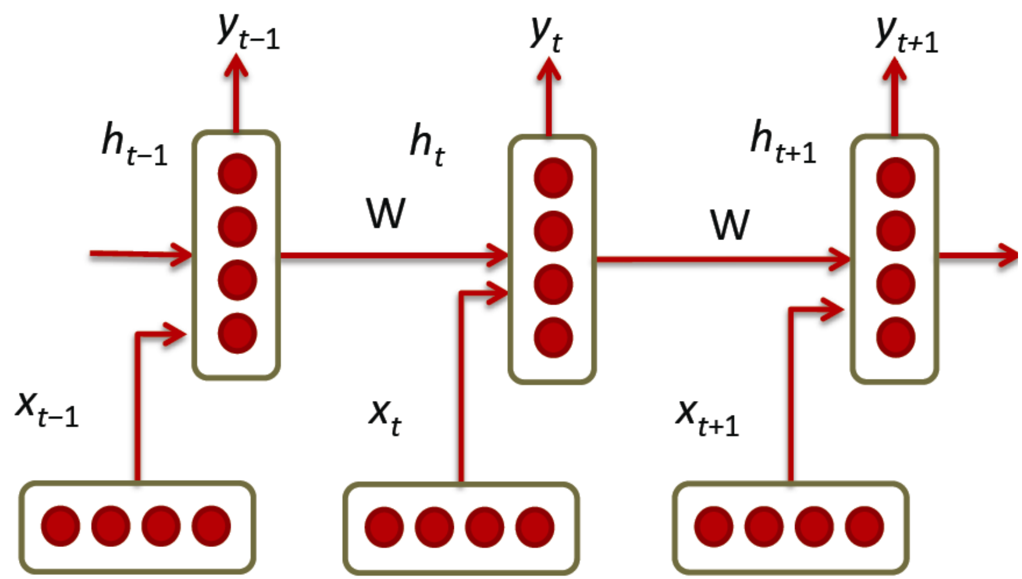
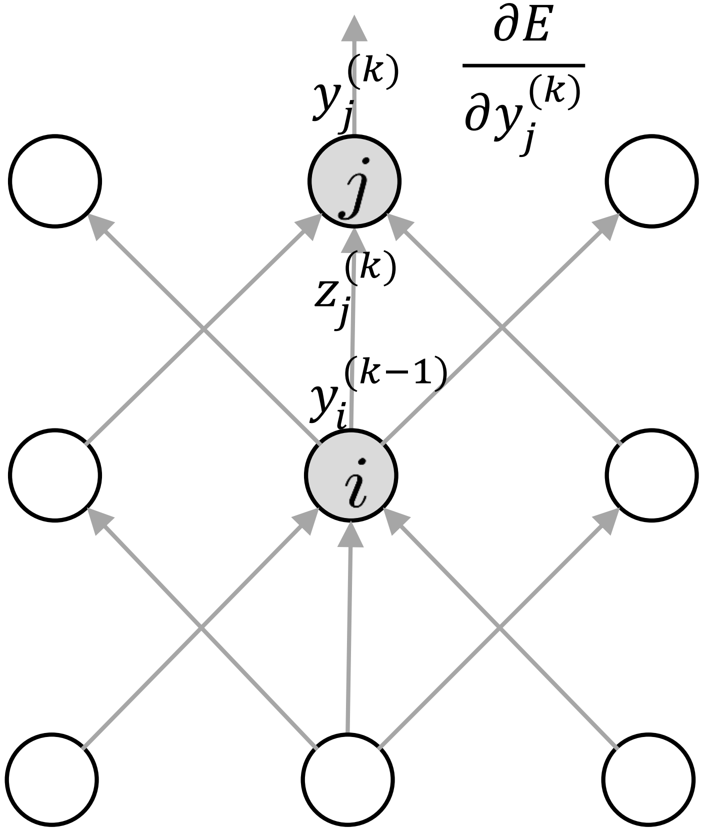


Image source: Richard Socher

Recap: Backpropagation Algorithm



$$\frac{\partial E}{\partial z_j^{(k)}} = \frac{\partial y_j^{(k)}}{\partial z_j^{(k)}} \frac{\partial E}{\partial y_j^{(k)}} = \frac{\partial g\left(z_j^{(k)}\right)}{\partial z_j^{(k)}} \frac{\partial E}{\partial y_j^{(k)}}$$

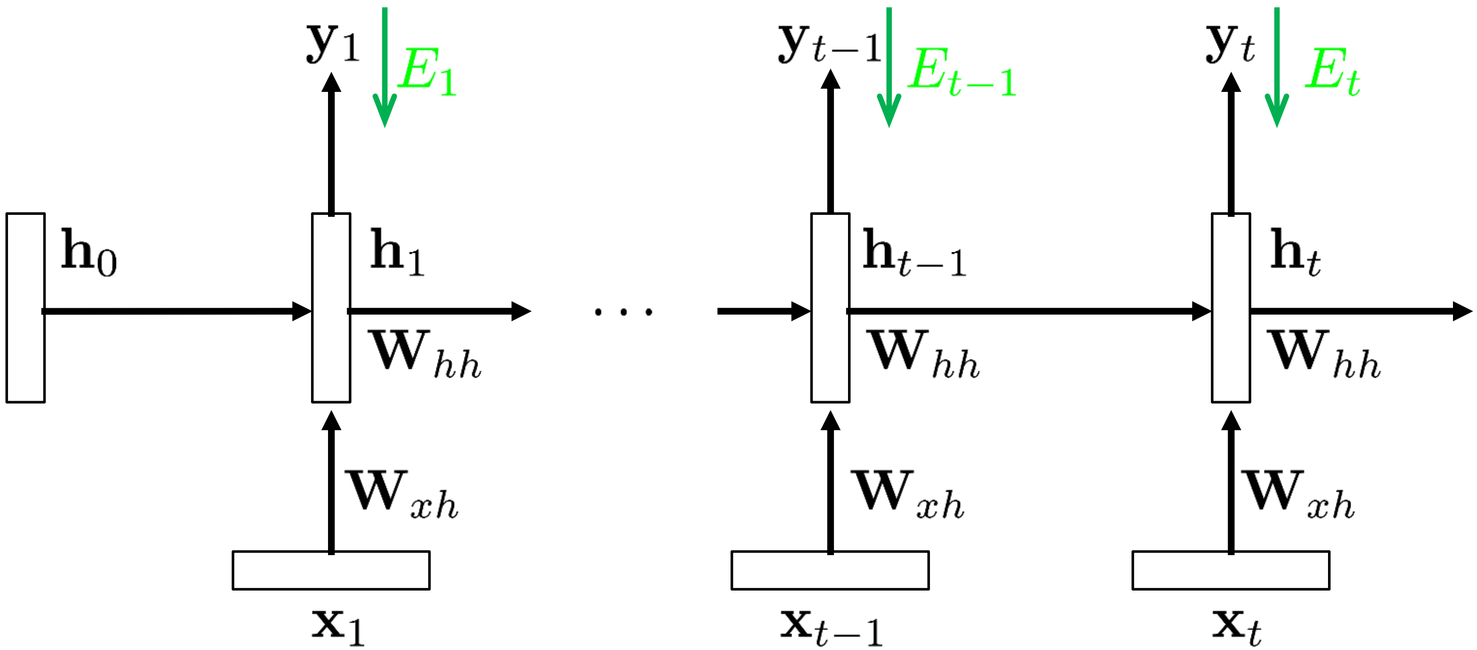
$$\frac{\partial E}{\partial y_i^{(k-1)}} = \sum_j \frac{\partial z_j^{(k)}}{\partial y_i^{(k-1)}} \frac{\partial E}{\partial z_j^{(k)}} = \sum_j w_{ji}^{(k-1)} \frac{\partial E}{\partial z_j^{(k)}}$$

$$\frac{\partial E}{\partial w_{ji}^{(k-1)}} = \frac{\partial z_j^{(k)}}{\partial w_{ji}^{(k-1)}} \frac{\partial E}{\partial z_j^{(k)}} = y_i^{(k-1)} \frac{\partial E}{\partial z_j^{(k)}}$$

- Efficient propagation scheme
  - $y_i^{(k-1)}$ is already known from forward pass! (Dynamic Programming)

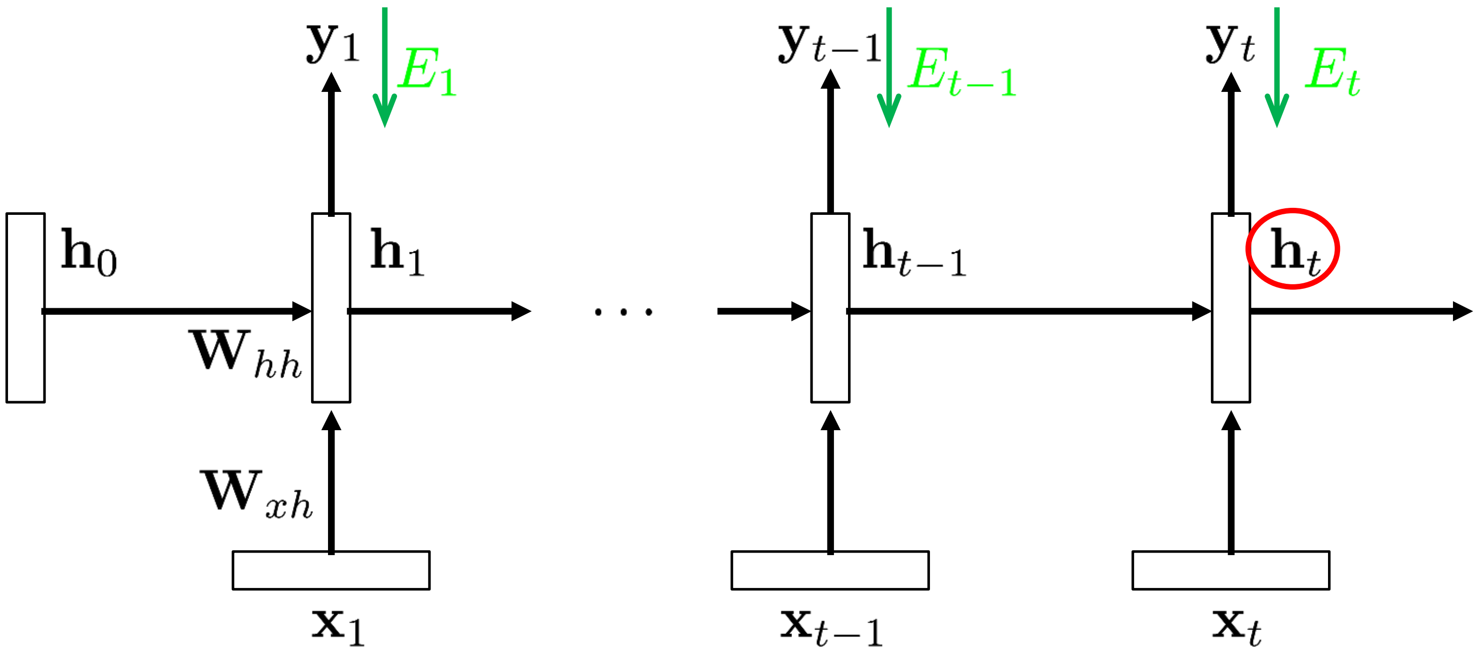
$\Rightarrow$ Propagate back the gradient from layer k and multiply with $y_i^{(k-1)}$.

Backpropagation Through Time (BPTT)



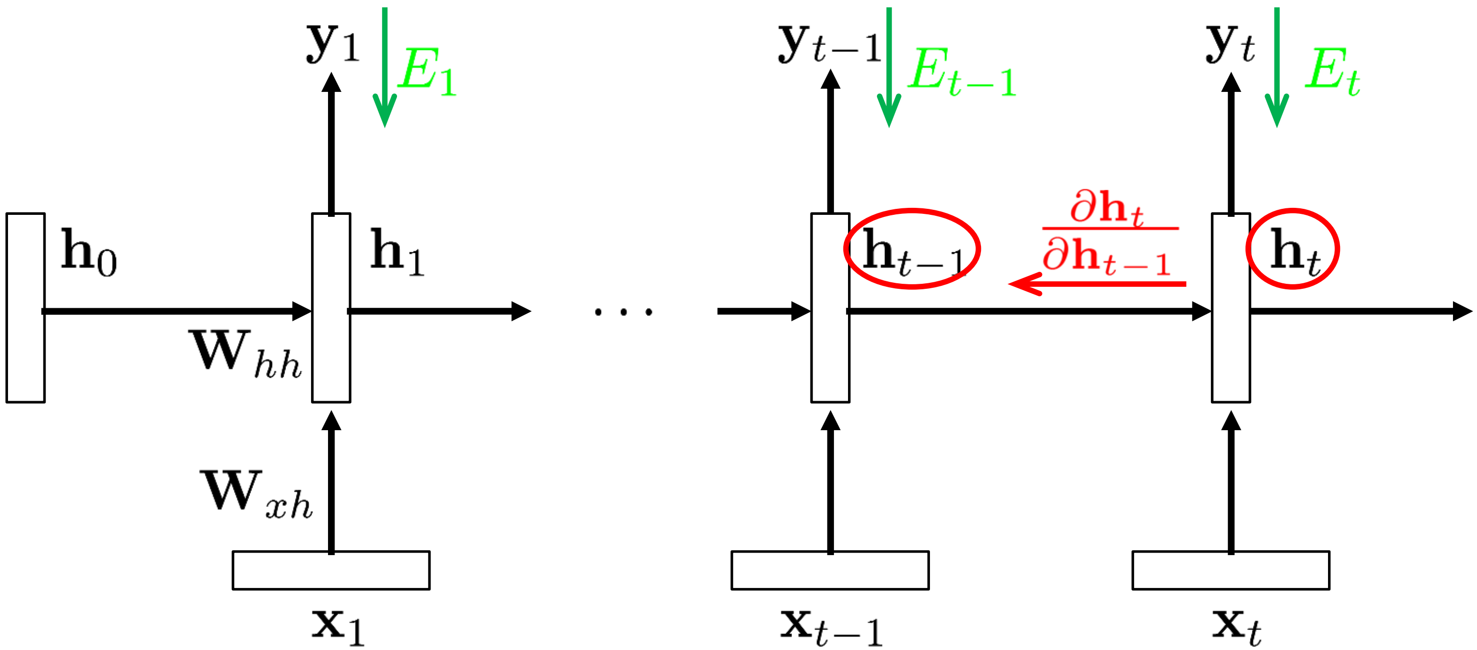
- Error function
  - Computed over all time steps: $E = \sum_{1 \leq t \leq T} E_t$

Backpropagation Through Time (BPTT)



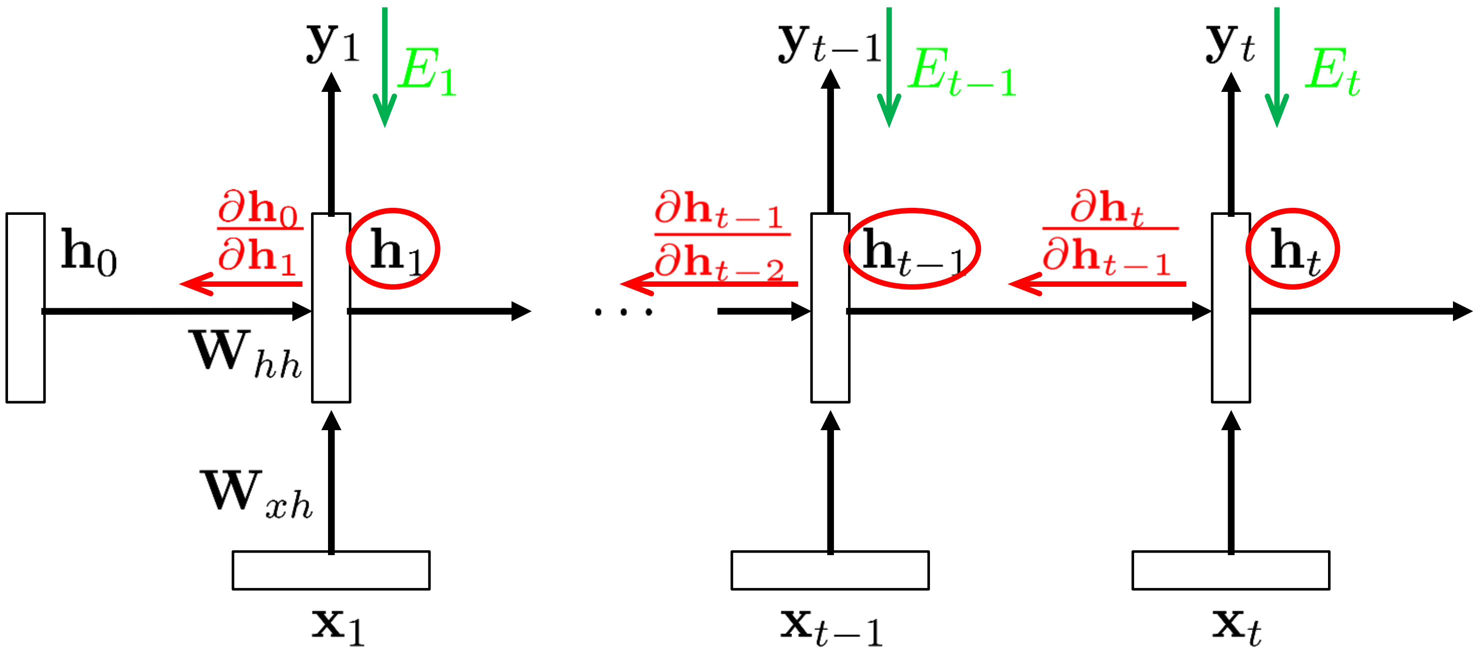
- Backpropagated gradient
  - For weight w_{ij}: $$\frac{\partial E_t}{\partial w_{ij}} = \frac{\partial E_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial w_{ij}}$$

Backpropagation Through Time (BPTT)



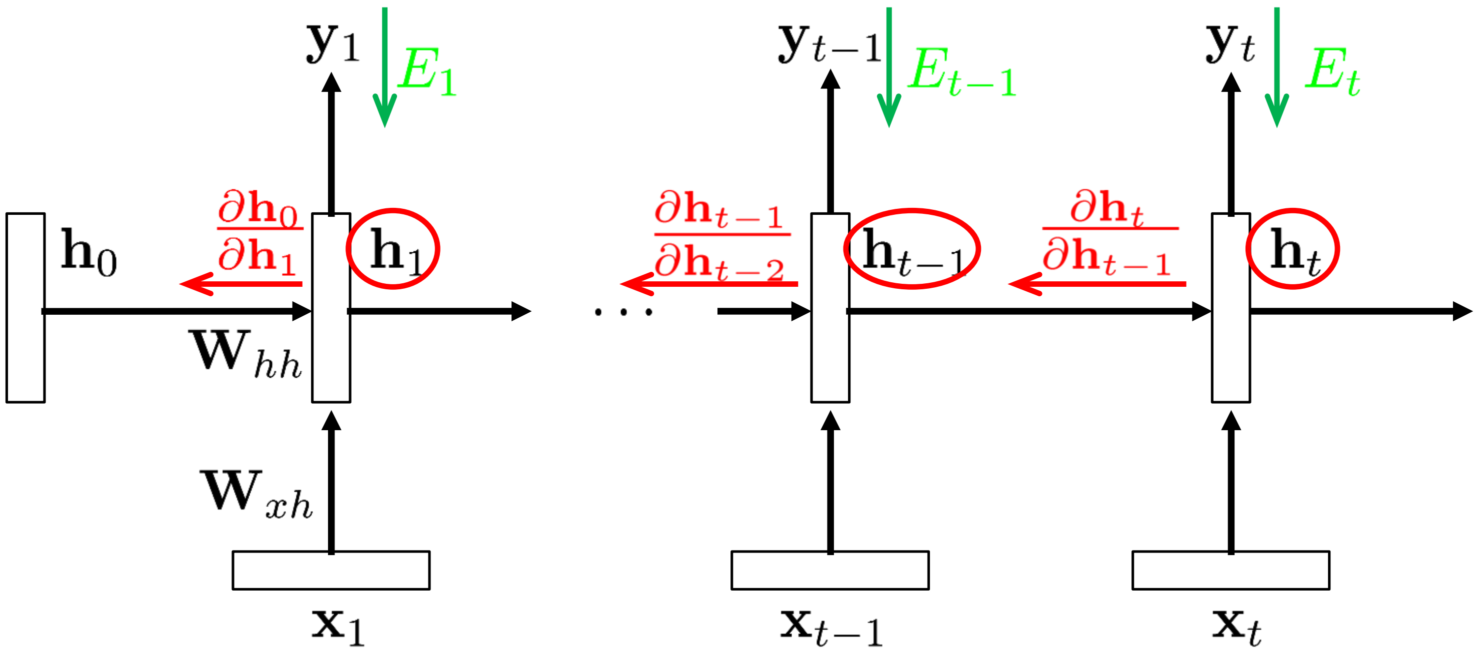
- Backpropagated gradient
  - For weight w_{ij}:
$$\frac{\partial E_t}{\partial w_{ij}} = \frac{\partial E_t}{\partial \mathbf{h}_t}\frac{\partial \mathbf{h}_t}{\partial w_{ij}} + \frac{\partial E_t}{\partial \mathbf{h}_t}\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}\frac{\partial \mathbf{h}_{t-1}}{\partial w_{ij}}$$

Backpropagation Through Time (BPTT)



- Backpropagated gradient
  - For weight w_{ij}: $$\frac{\partial E_t}{\partial w_{ij}} = \frac{\partial E_t}{\partial \mathbf{h}_t}\frac{\partial \mathbf{h}_t}{\partial w_{ij}} + \frac{\partial E_t}{\partial \mathbf{h}_t}\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}\frac{\partial \mathbf{h}_{t-1}}{\partial w_{ij}} + \cdots$$
  - In general: $$\frac{\partial E_t}{\partial w_{ij}} = \sum_{1 \le k \le t}\left(\frac{\partial E_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial^+ h_k}{\partial w_{ij}}\right)$$

Backpropagation Through Time (BPTT)

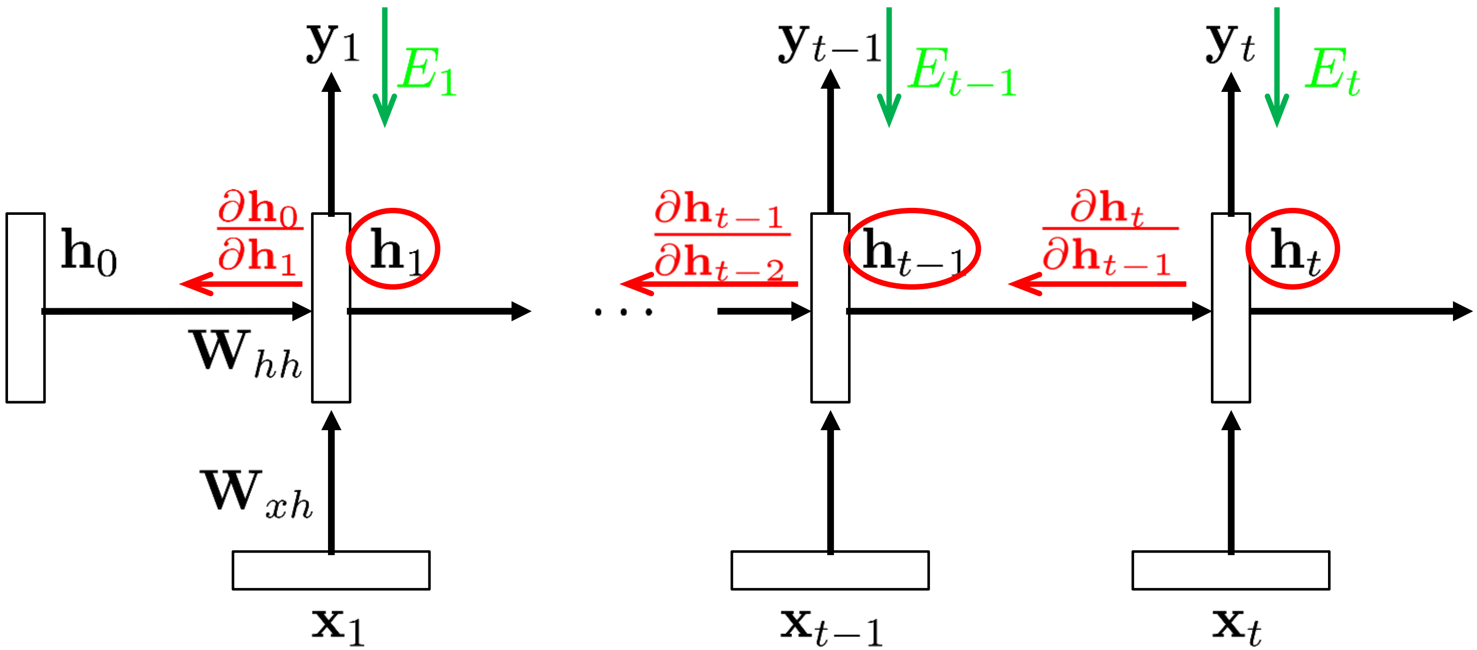


- Analyzing the terms
  - For weight w_{ij}:

$$\frac{\partial E_t}{\partial w_{ij}} = \sum_{1 \le k \le t} \left(\frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial^+ h_k}{\partial w_{ij}} \right)$$

  - This is the "immediate" partial derivative (with $\mathbf{h}_{k\text{-}1}$ as constant)

Backpropagation Through Time (BPTT)



- Analyzing the terms
  - For weight w_{ij}:
  $$\frac{\partial E_t}{\partial w_{ij}} = \sum_{1 \le k \le t} \left(\frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial^+ h_k}{\partial w_{ij}} \right)$$
  - Propagation term:
  $$\frac{\partial h_t}{\partial h_k} = \prod_{t \ge i > k} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}}$$

Backpropagation Through Time (BPTT)

- Summary
  - Backpropagation equations

$$E = \sum_{1 \leq t \leq T} E_t$$

$$\frac{\partial E_t}{\partial w_{ij}} = \sum_{1 \leq k \leq t} \left(\frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial^+ h_k}{\partial w_{ij}} \right)$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{t \geq i > k} \frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \prod_{t \geq i > k} \mathbf{W}_{hh}^\top diag\left(\sigma'(\mathbf{h}_{i-1})\right)$$

  - Remaining issue: how to set the initial state $\mathbf{h}_0$?

$\Rightarrow$ Learn this together with all the other parameters.

Topics of This Lecture

- Recurrent Neural Networks (RNNs)
  - Motivation
  - Intuition
- Learning with RNNs
  - Formalization
  - Comparison of Feedforward and Recurrent networks
  - Backpropagation through Time (BPTT)
- Problems with RNN Training
  - Vanishing Gradients
  - Exploding Gradients
  - Gradient Clipping

Problems with RNN Training

- Training RNNs is very hard
  - As we backpropagate through the layers, the magnitude of the gradient may grow or shrink exponentially

  $\Rightarrow$ Exploding or vanishing gradient problem!

  - In an RNN trained on long sequences (e.g., 100 time steps) the gradients can easily explode or vanish.
  - Even with good initial weights, it is very hard to detect that the current target output depends on an input from many time-steps ago.

Exploding / Vanishing Gradient Problem

- Consider the propagation equations:

$$\frac{\partial E_t}{\partial w_{ij}} = \sum_{1\le k\le t}\left(\frac{\partial E_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial^+ h_k}{\partial w_{ij}}\right)$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{t\ge i>k}\frac{\partial \mathbf{h}_i}{\partial \mathbf{h}_{i-1}} = \prod_{t\ge i>k}\mathbf{W}_{hh}^{\top}\,diag\left(\sigma'(\mathbf{h}_{i-1})\right)$$

$$= \left(\mathbf{W}_{hh}^{\top}\right)^l$$

  - if t goes to infinity and $l = t - k$.

$\Rightarrow$ We are effectively taking the weight matrix to a high power.

  - The result will depend on the eigenvalues of $\mathbf{W}_{hh}$.
    - Largest eigenvalue > 1 $\Rightarrow$ Gradients *may* explode.
    - Largest eigenvalue < 1 $\Rightarrow$ Gradients *will* vanish.
    - This is very bad...

Why Is This Bad?

- Vanishing gradients in language modeling
  - Words from time steps far away are not taken into consideration when training to predict the next word.

- Example:
  - „Jane walked into the room. John walked in too. It was late in the day. Jane said hi to ____"

$\Rightarrow$ The RNN will have a hard time learning such long-range dependencies.

Gradient Clipping

- Trick to handle exploding gradients
  - If the gradient is larger than a threshold, clip it to that threshold.

Algorithm 1 Pseudo-code for norm clipping the gradients whenever they explode

---

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$

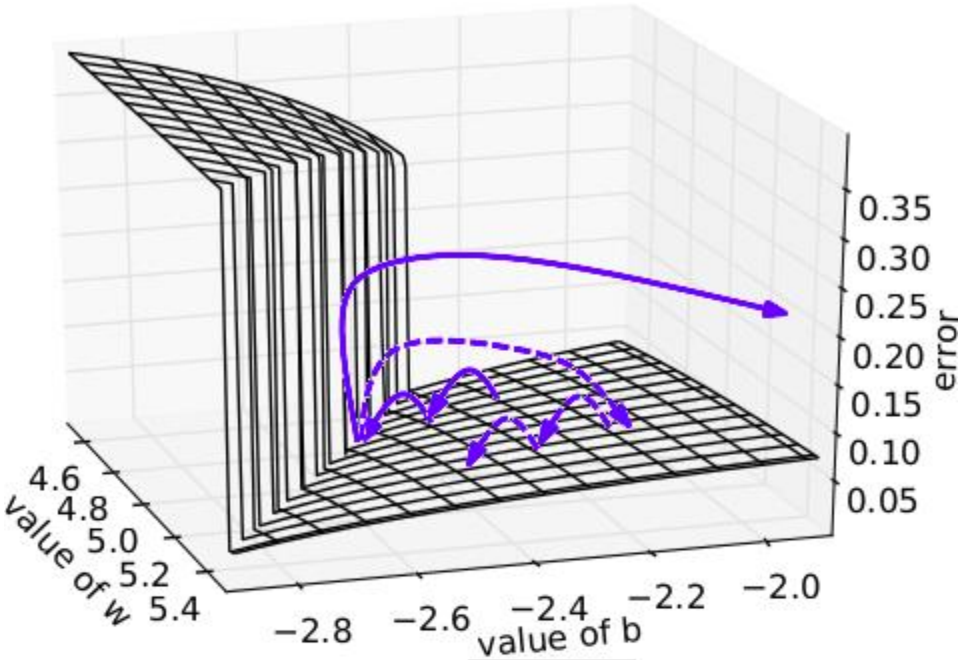
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**

$\quad \hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$

end if

  - This makes a big difference in RNNs

Gradient Clipping Intuition



[redacted]

- Example
  - Error surface of a single RNN neuron
  - High curvature walls
  - Solid lines: standard gradient descent trajectories
  - Dashed lines: gradients rescaled to fixed size

Image source: Pascanu et al., 2013

Handling Vanishing Gradients

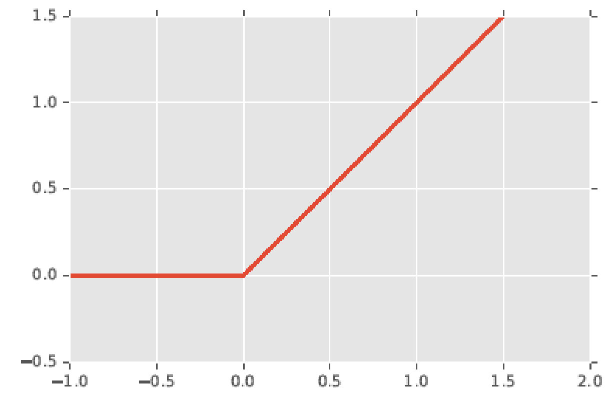
- Vanishing Gradients are a harder problem
  - They severely restrict the dependencies the RNN can learn.
  - The problem gets more severe the deeper the network is.
  - It can be very hard to diagnose that Vanishing Gradients occur (you just see that learning gets stuck).

- Ways around the problem
  - Glorot/He initialization (more on that in Lecture 21)
  - ReLU
  - More complex hidden units (LSTM, GRU)

ReLU to the Rescue

- Idea
  - Initialize $\mathbf{W}_{hh}$ to identity matrix
  - Use Rectified Linear Units (ReLU)

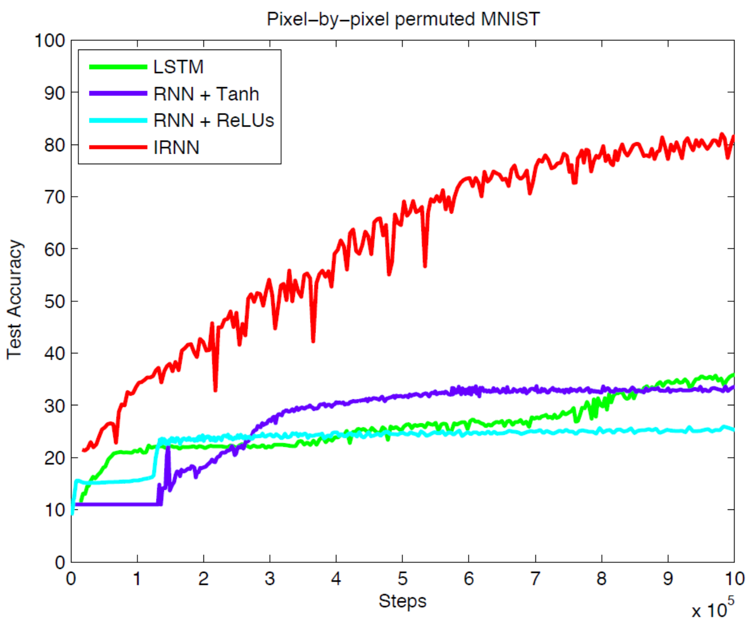
$$g(a) = \max\{0, a\}$$



- Effect
  - The gradient is propagated with a constant factor

$$\frac{\partial g(a)}{\partial a} = \begin{cases} 1, & a > 0 \\ 0, & \text{else} \end{cases}$$

$\Rightarrow$ Huge difference in practice!



References and Further Reading

- RNNs
  - R. Pascanu, T. Mikolov, Y. Bengio, On the difficulty of training recurrent neural networks, JMLR, Vol. 28, 2013.
  - A. Karpathy, The Unreasonable Effectiveness of Recurrent Neural Networks, blog post, May 2015.