COMP 551 - Applied Machine Learning Lecture 16 — Recurrent neural networks

William L. Hamilton

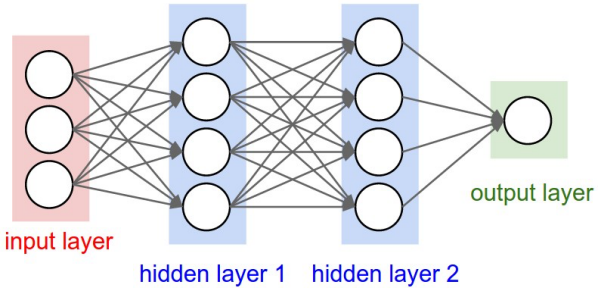
(with slides and content from Joelle Pineau and Ryan Lowe)

Midterm

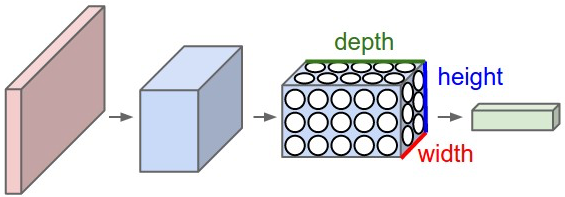
- I released practice questions. (See the announcement on MyCourses).
- The midterm is 6-8pm on Nov. 18th.
  - We will send room assignments soon!
- The midterm itself will be 100 minutes.

Last time: Convolutional Neural Networks

Feedforward network



Convolutional neural network (CNN)



- CNN characteristics:
  - Input is usually a 3D tensor: 2D image x 3 colours
  - Each layer transforms an input 3D tensor to an output 3D tensor using a differentiable function.

From: *http://cs231n.github.io/convolutional-networks/*

Some practical issues

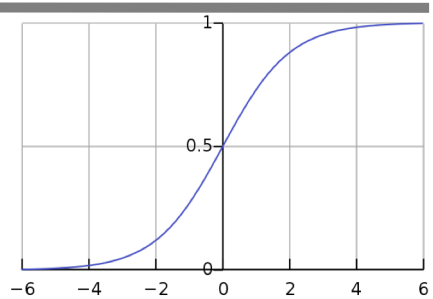
- Issue 1: Softmax vs. sigmoid

- Issue 2: Optimization concerns

Practical issue #1: softmax vs. sigmoid

- We have discussed the sigmoid function for binary classification

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



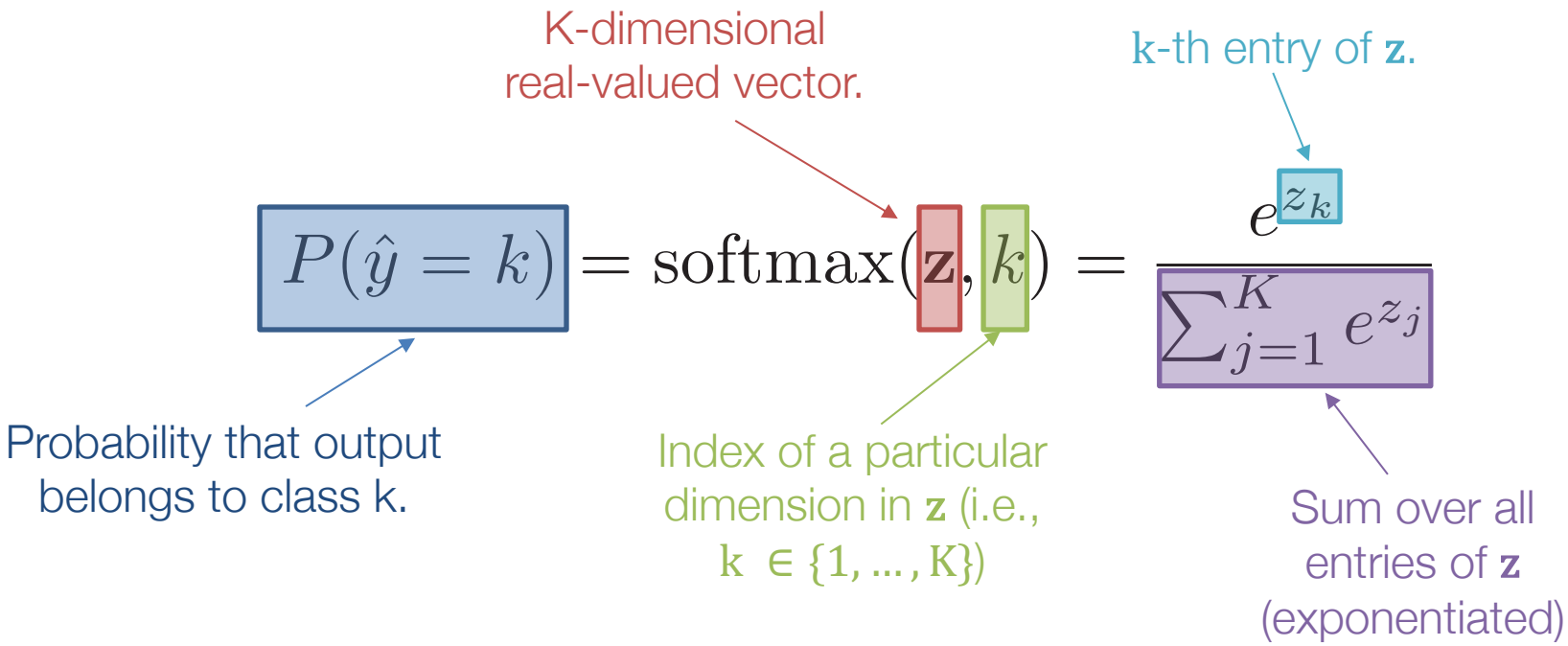
  - Maps a real-valued scalar to probability.
  - Combine with cross-entropy loss to train binary classification models.

- It can be extended to the softmax function for multiclass classification

$$\text{softmax}(\mathbf{z}, k) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

  - Maps a real-valued K-dimensional vector to a K-way categorical distribution.
  - Combine with cross-entropy loss to train K-way classification models

Practical issue #1: softmax vs. sigmoid



Practical issue #1: softmax vs. sigmoid

- Sigmoid is essentially a special case of the softmax with only two classes:

$$\sigma(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{1+e^z} = \frac{e^z}{e^z+e^0}$$

- Cross entropy can be generalized to the multiclass/softmax setting:

$$\begin{aligned}\text{softmax-cross-entropy}(y, \mathbf{z}) &= -\log(P(\hat{y} = y)) \\ &= -\log(\text{softmax}(\mathbf{z}, y))\end{aligned}$$

- Multi-class (K-dimensional softmax) is not multi-output (K-independent sigmoids).
  - Use softmax when every point belongs to one of K-classes.
  - Use K independent sigmoids

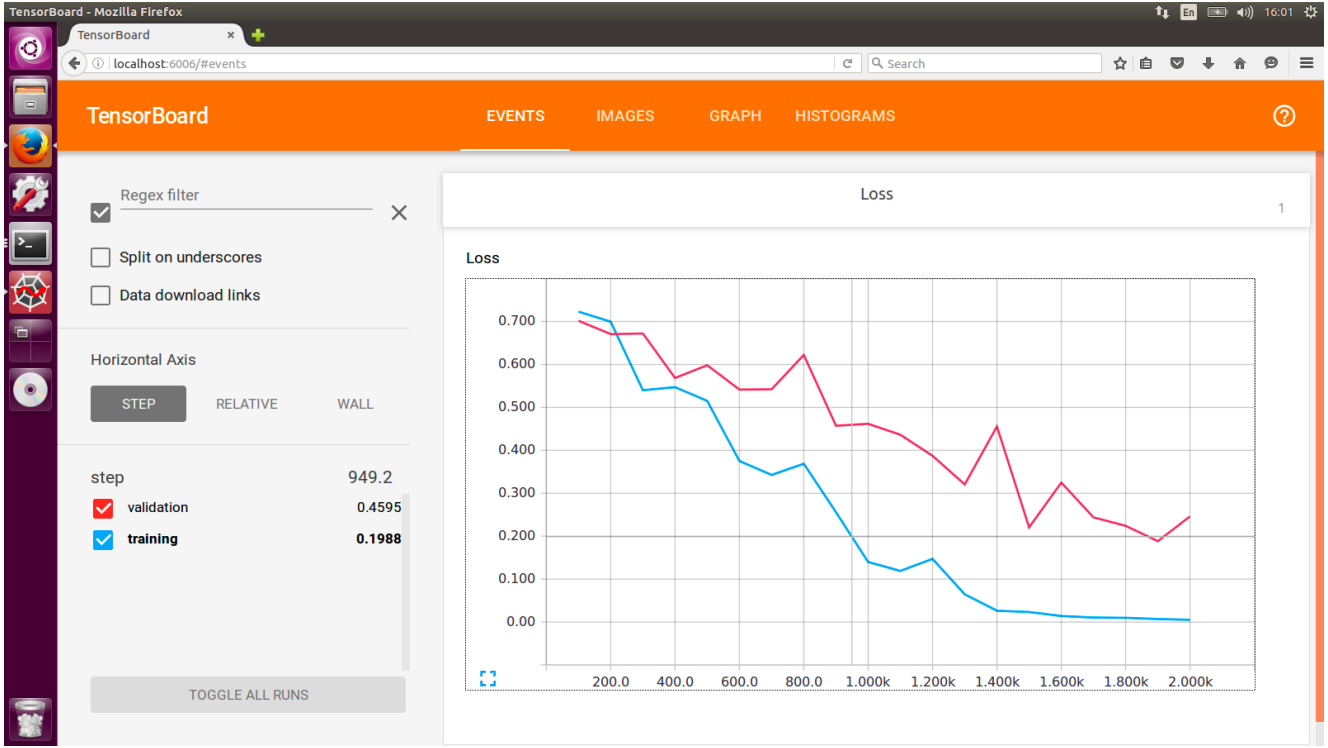
Practical issue #1: softmax vs. sigmoid

- Multi-class classification (K-dimensional softmax) is not the same as multi-output classification (K-independent sigmoid functions).

- Use softmax/multiclass when every point belongs to one of K-classes.
  - E.g., standard MNIST digit classification
  - More common scenario.

- Use K independent sigmoids when there are multiple labels for each point.
  - E.g., suppose we are detecting shapes in an images that contain multiple shapes

Practical issue #2: Optimization concerns



Practical issue # 2: Choosing the learning rate

- Backprop is very sensitive to the choice of learning rate.
  - Too large ⇒ divergence.
  - Too small ⇒ VERY slow learning.
  - The learning rate also influences the ability to escape local optima.

- The learning rate is a critical hyperparameter.

Practical issue # 2: Adaptive optimization

- It is now standard to use "adaptive" optimization algorithms.

- These approaches modify the learning rate adaptively depending on how the training is progressing.

- Adam, RMSProp, and AdaGrad are popular approaches, with Adam being the de facto standard in deep learning.
  - All of these approaches scale the learning rate for each parameter based on statistics of the history of gradient updates for that parameter.
  - **Intuition:** increase the update strength for parameters that have had smaller updates in the past.

Practical issue # 2: Adding momentum

- At each iteration of gradient descent, we are computing an update based based on the derivative of the current (mini)batch of training examples:

Update for weight vector

$$\Delta_i \mathbf{w} = \alpha \frac{\partial J}{\partial \mathbf{w}}$$

Derivative of error w.r.t. weights for current minibatch

$$\mathbf{w} \leftarrow \Delta_i \mathbf{w}$$

Practical issue # 2: Adding momentum

- On i'th gradient descent update, instead of:

$$\Delta_i \mathbf{w} = \alpha \frac{\partial J}{\partial \mathbf{w}}$$

We do:

$$\Delta_i \mathbf{w} = \alpha \frac{\partial J}{\partial \mathbf{w}} + \beta \Delta_{i-1} \mathbf{w}$$

The second term is called momentum

Practical issue # 2: Adding momentum

- On i'th gradient descent update, instead of:

$$\Delta_i \mathbf{w} = \alpha \frac{\partial J}{\partial \mathbf{w}}$$

We do:

$$\Delta_i \mathbf{w} = \alpha \frac{\partial J}{\partial \mathbf{w}} + \beta \Delta_{i-1} \mathbf{w}$$

The second term is called momentum

Advantages:

- Easy to pass small local minima.
- Keeps the weights moving in areas where the error is flat.

Disadvantages:

- With too much momentum, it can get out of a global maximum!
- One more parameter to tune, and more chances of divergence

Major paradigms for deep learning

- Deep neural networks: The model should be interpreted as a computation graph.
  - Supervised training: E.g., feedforward neural networks.
  - Unsupervised training (later in the course): E.g., autoencoders.

- Special architectures for different problem domains.
  - Computer vision => Convolutional neural nets.
  - Text and speech => Recurrent neural nets.

Major paradigms for deep learning

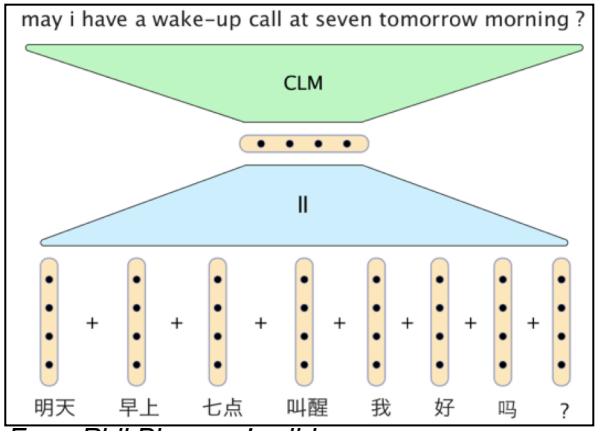
- Deep neural networks: The model should be interpreted as a computation graph.
  - Supervised training: E.g., feedforward neural networks.
  - Unsupervised training (later in the course): E.g., autoencoders.

- Special architectures for different problem domains.
  - Computer vision => Convolutional neural nets.
  - Text and speech => Recurrent neural nets.

Neural models for sequences

- Several datasets contain sequences of data (e.g. time-series, text)
- How could we process sequences with a **feed-forward neural network**?
  1. Take vectors representing the last N timesteps and **concatenate** (join) them
  2. Take vectors representing the last N timesteps and **average** them

Example: machine translation
- Input: sequence of words
- Output: sequence of words



From Phil Blumson's slides

Neural models for sequences

- Problem: these approaches don't exploit the sequential nature of the data!!
- Also, they can only consider information from a fixed-size context window
- Temporal information is very important in sequences!!
- E.g. machine translation:

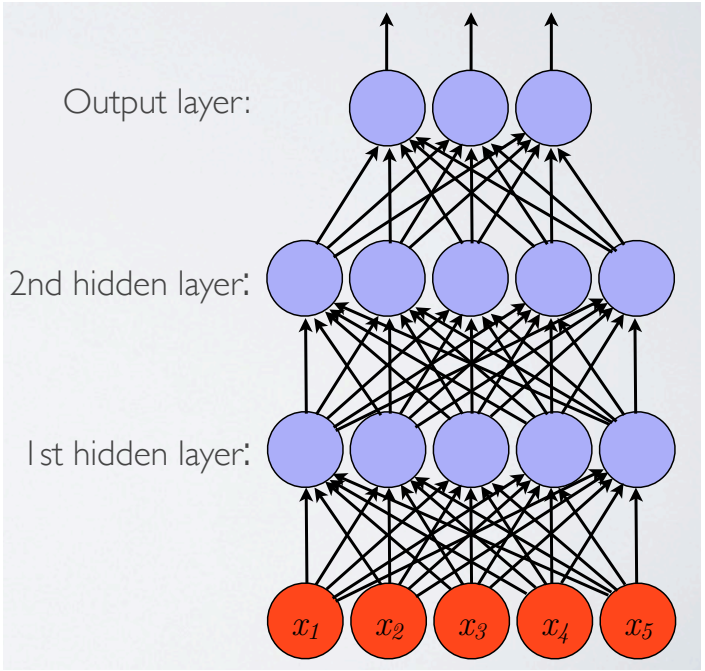
"John hit Steve on the head with a bat"

!= "Steve hit John on the bat with a head"

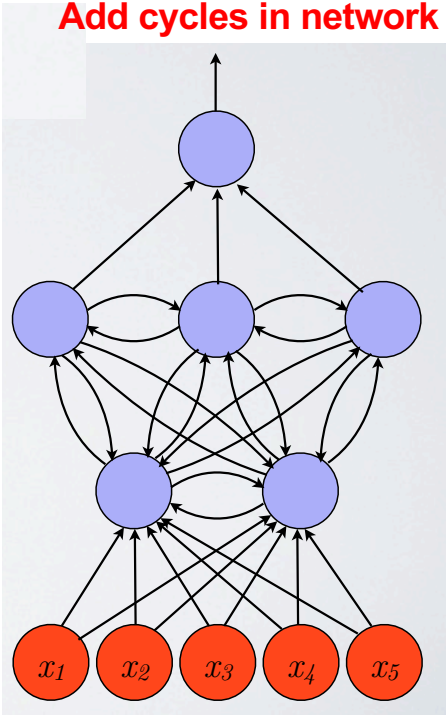
!= "Bat hit a with head on the John Steve"

Recurrent Neural Networks (RNNs)

Feed-forward neural net



Add cycles in network



Recurrent Neural Networks (RNNs)

- What kind of cycles?
- Cycles with a time delay
- Box means that the information is sent at the next time step (no infinite loops)

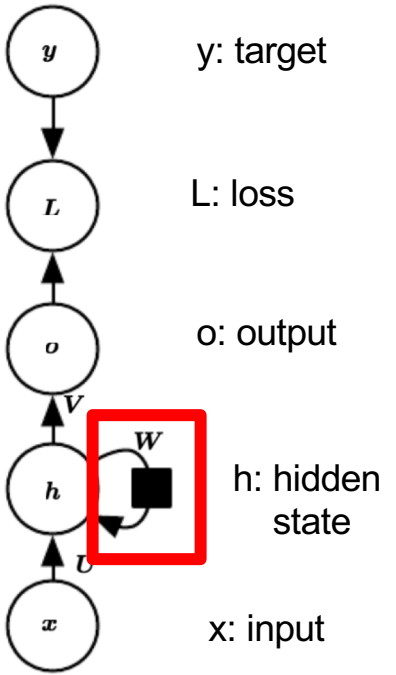


Image from deeplearningbook.org

Recurrent Neural Networks (RNNs)

- What does this allow us to do?
- Can view RNN as having a hidden state $\mathbf{h}_t$ that changes over time.
- $\mathbf{h}_t$ represents "useful information" from past inputs.
- A standard/simple RNN:

$$\mathbf{h}_t = \sigma(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b})$$

$$\mathbf{o}_t = \phi(\mathbf{V}\mathbf{h}_t + \mathbf{c})$$

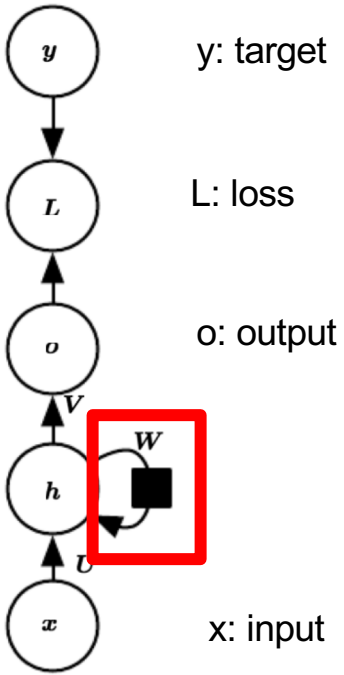
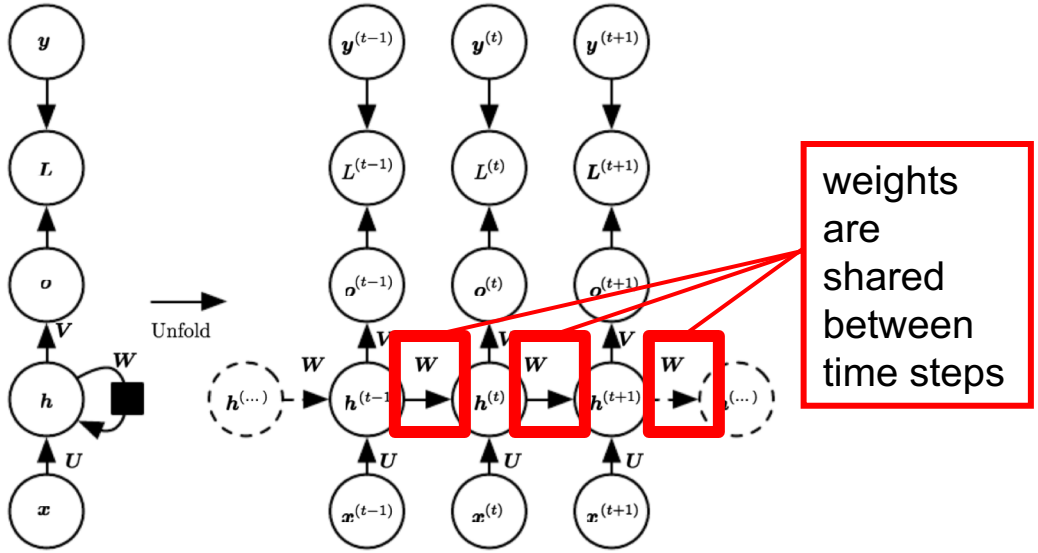


Image from deeplearningbook.org

Recurrent Neural Networks (RNNs)

- Can unroll the RNN over time to form an acyclic graph.
- RNN = special kind of feed-forward network



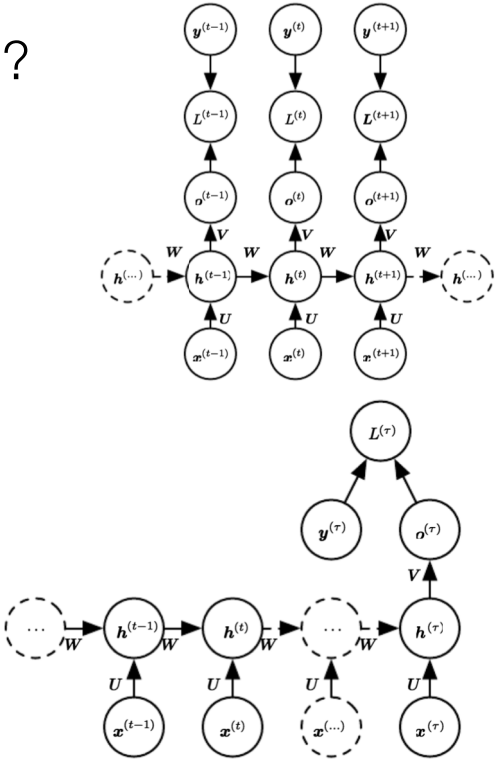
$$\text{E.g., } \mathbf{h}_3 = \sigma(\mathbf{W}(\sigma(\mathbf{W}\sigma(\mathbf{W}\mathbf{h}_0 + \mathbf{U}\mathbf{x}_1 + \mathbf{b}) + \mathbf{U}\mathbf{x}_2\mathbf{b}) + \mathbf{U}\mathbf{x}_3 + \mathbf{b})$$

Kinds of output

- How do we specify the target output of an RNN?
- Many ways! Two main ones:

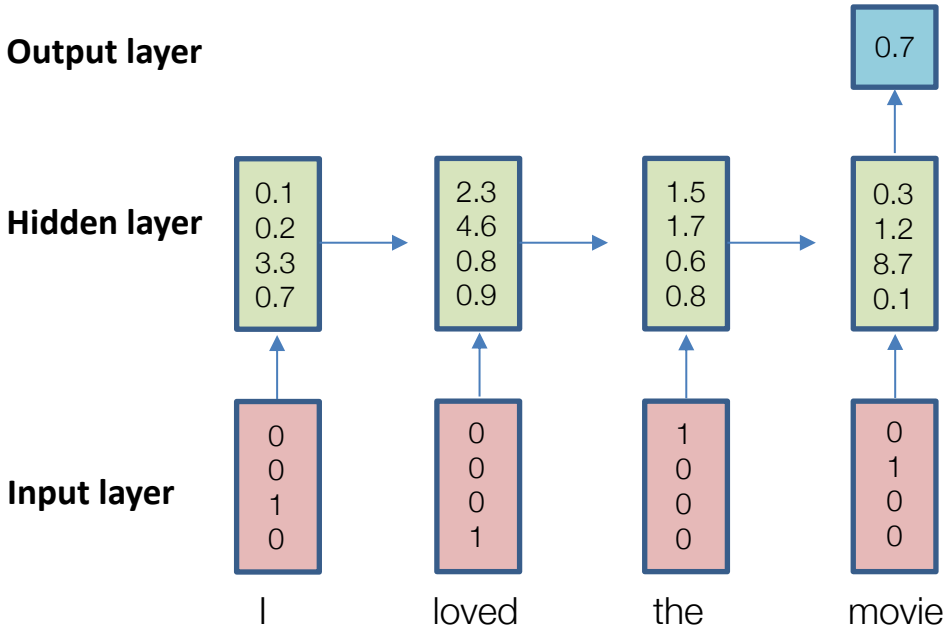
1) Can specify one target at the end of the sequence
   - Ex: sentiment classification

2) Can specify an target at each time step
   - Ex: generating language



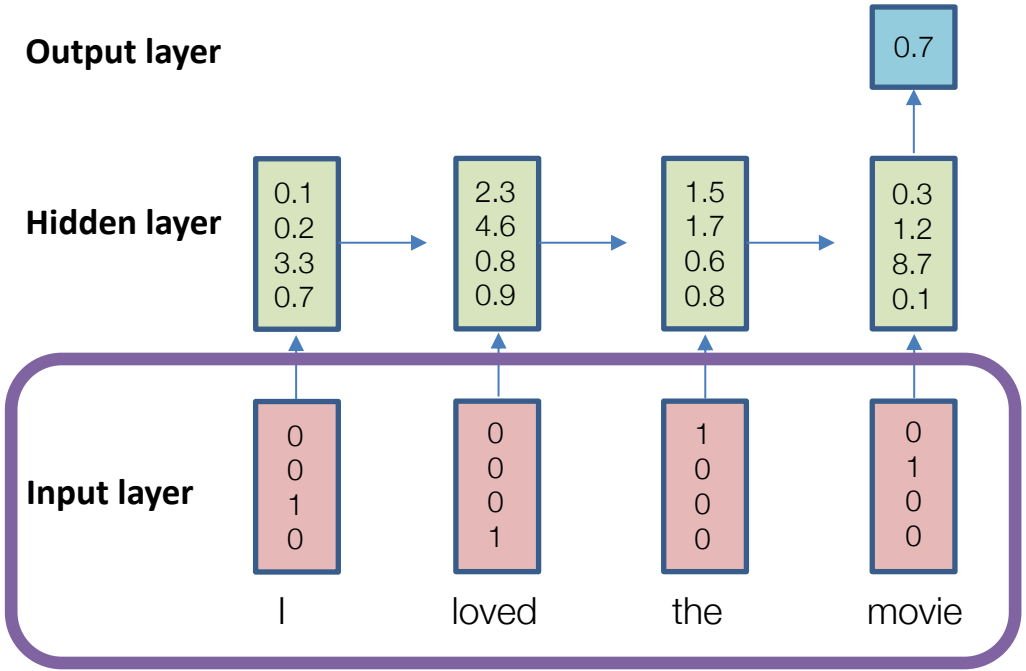
Sentiment classification

- **Input:** Sequence of words
  - E.g., a review, a tweet, a news article
- **Output:** A single value
  - E.g., indicating the probability that the text has a positive sentiment



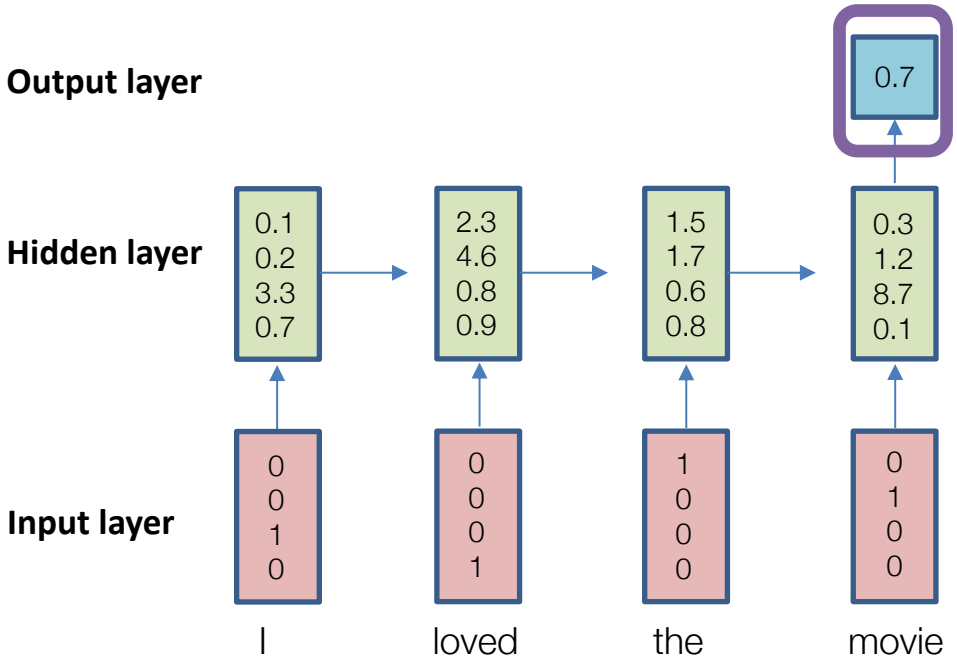
Sentiment classification

- **Input:** Sequence of words
  - E.g., review, tweet, or news article
- **Output:** A single value
  - E.g., indicating the probability that the text has a positive sentiment
- Words can be encoded as "one-hot" vectors.



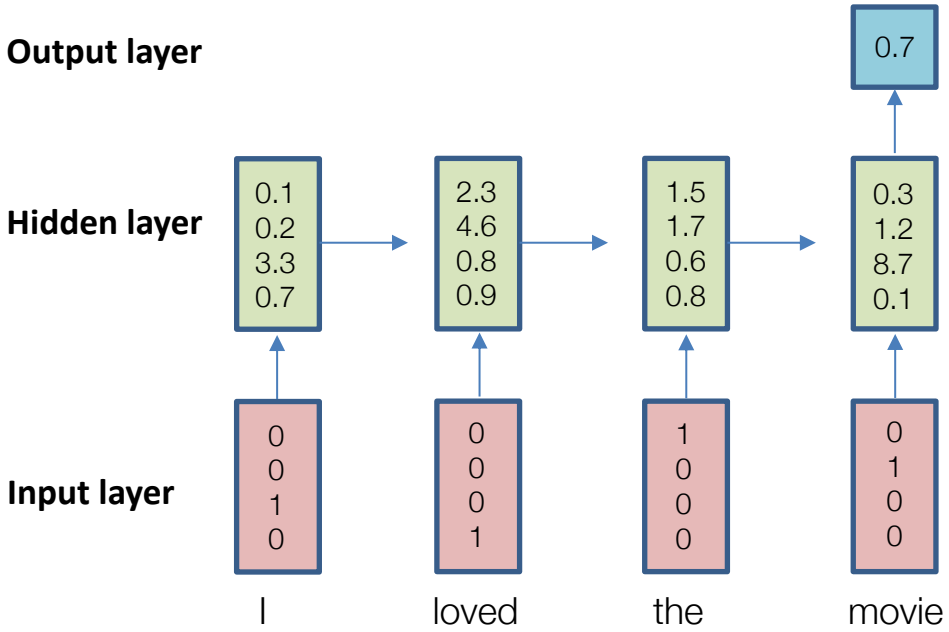
Sentiment classification

- **Input:** Sequence of words
  - E.g., review, tweet, or news article
- **Output:** A single value
  - E.g., indicating the probability that the text has a positive sentiment
- There is only output at the end of the sequence



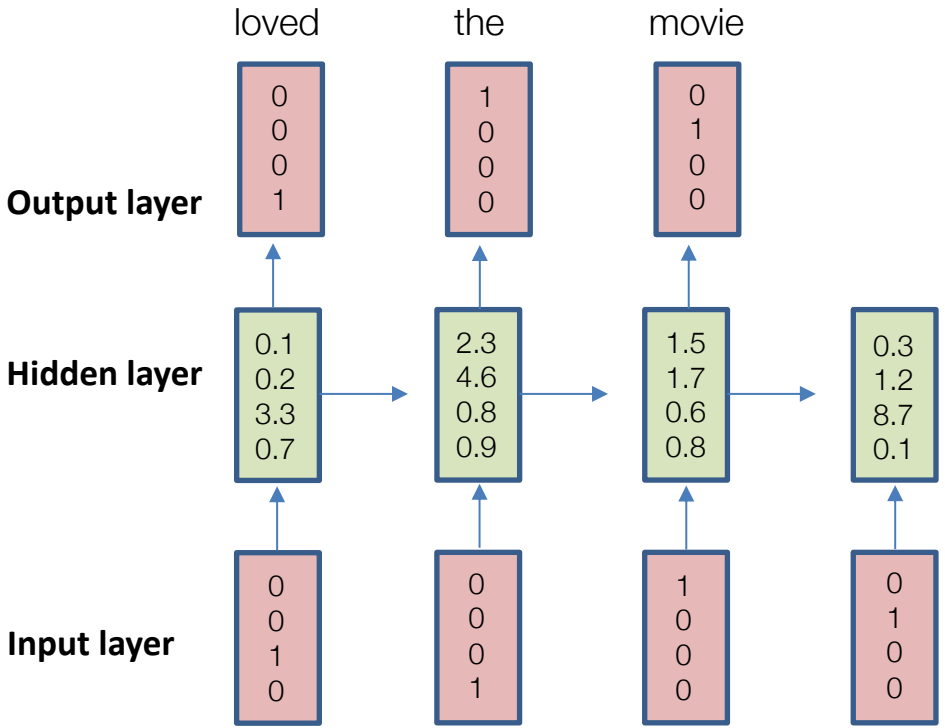
Sentiment classification

- **Input:** Sequence of words
  - E.g., review, tweet, or news article
- **Output:** A single value
  - E.g., indicating the probability that the text has a positive sentiment
- Classic example of a "sequence classification" task.



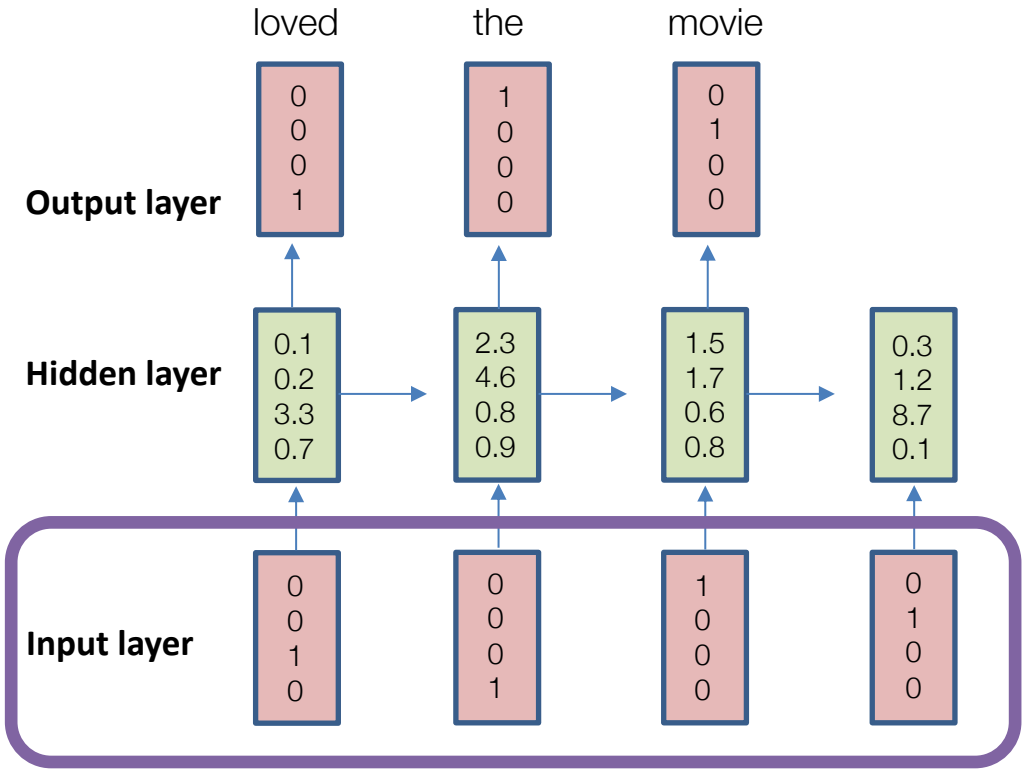
Language modeling

- **Input:** Sequence of words
  - E.g., review, tweet, or news article
- **Output:** Sequence of words
  - E.g., predicting the next word that will occur in a sentence.



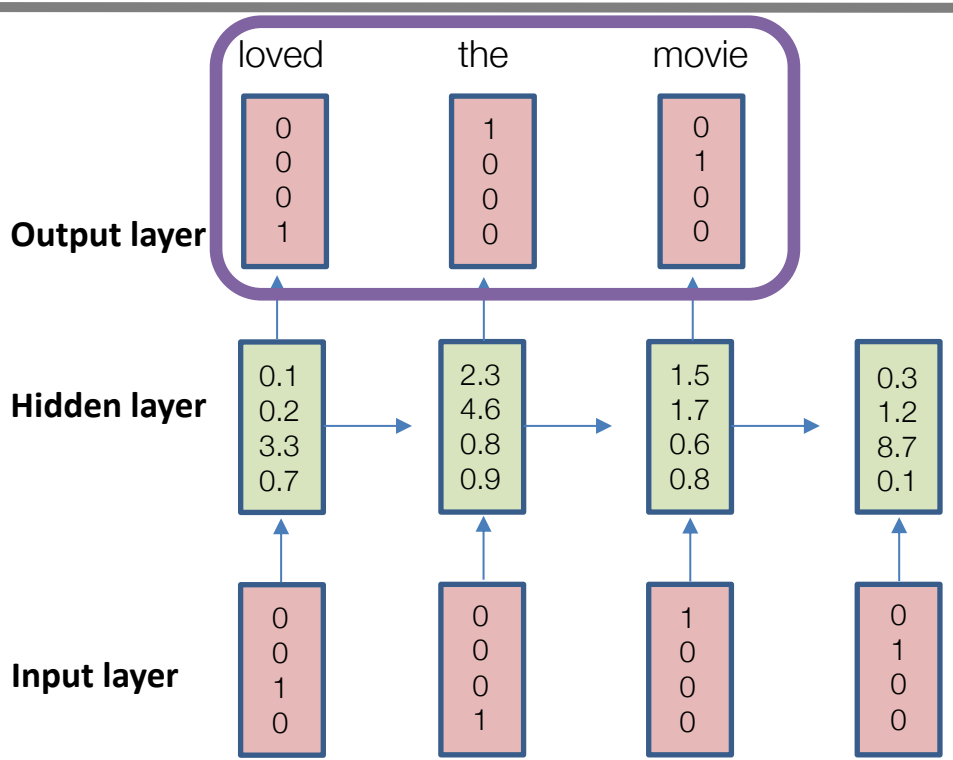
Language modeling

- **Input:** Sequence of words
  - E.g., review, tweet, or news article
- **Output:** Sequence of words
  - E.g., predicting the next word that will occur in a sentence.
- Same input representation as sentiment classification



Language modeling

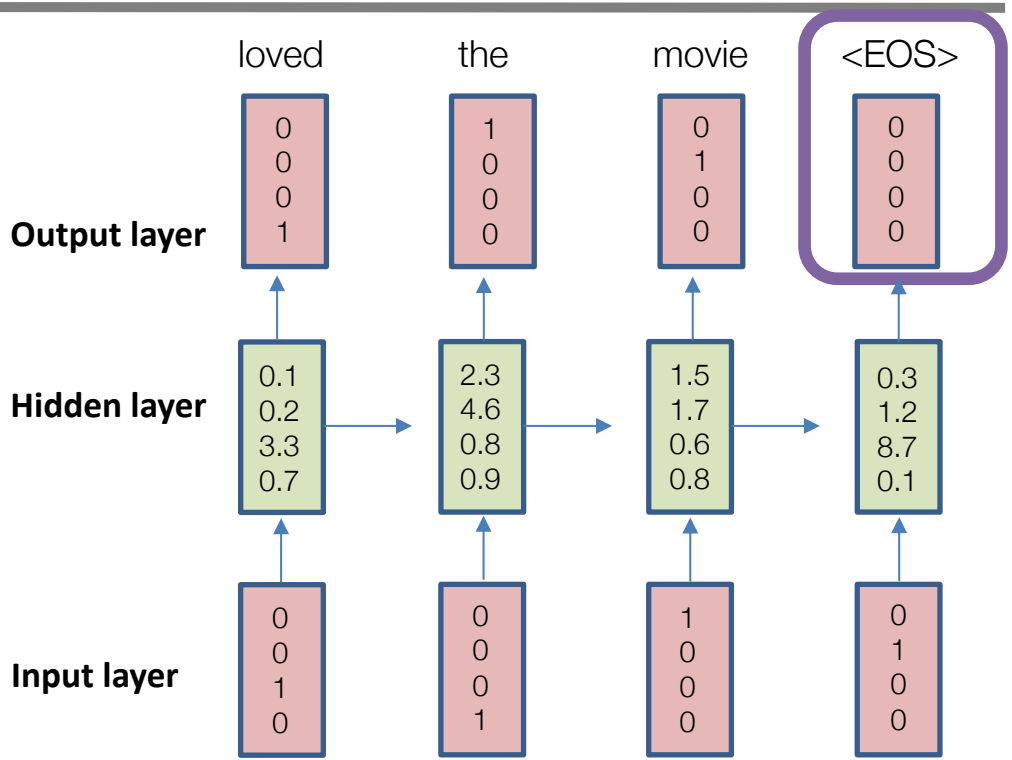
- **Input:** Sequence of words
  - E.g., review, tweet, or news article
- **Output:** Sequence of words
  - E.g., predicting the next word that will occur in a sentence.
- But now we have an output at each time-step!
- Predicting the next word is a multiclass prediction problem (each word is a class), so use a softmax!



Language modeling

- **Input:** Sequence of words
  - E.g., review, tweet, or news article
- **Output:** Sequence of words
  - E.g., predicting the next word that will occur in a sentence.

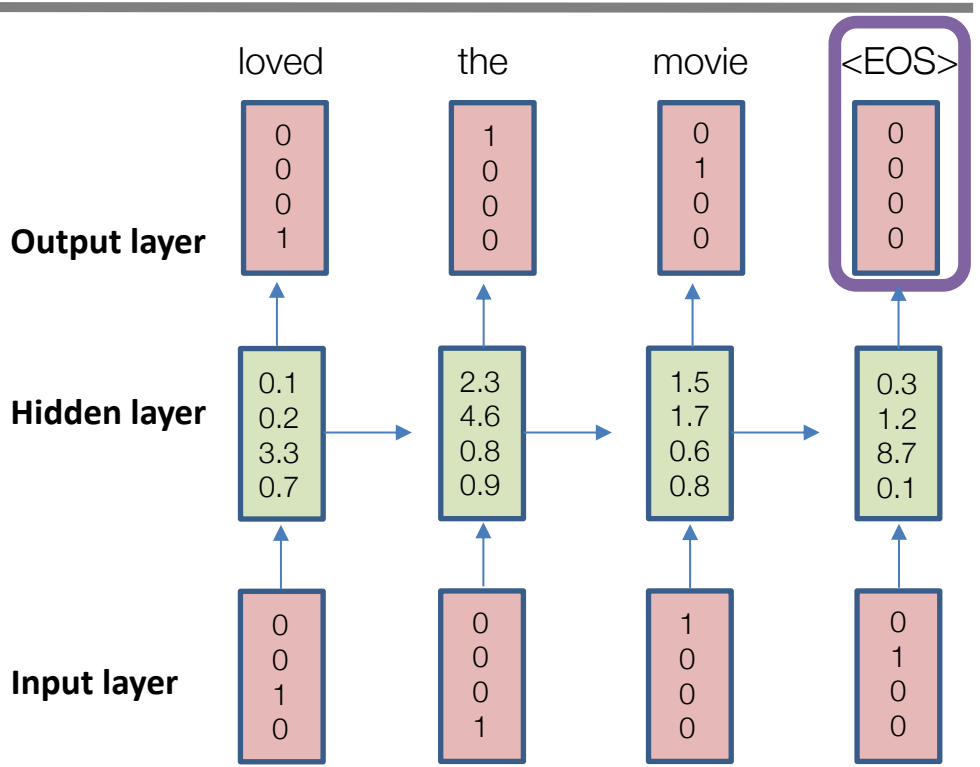
- Usually add an "end of sentence token"



Language modeling

- **Input:** Sequence of words
  - E.g., review, tweet, or news article
- **Output:** Sequence of words
  - E.g., predicting the next word that will occur in a sentence.

- Classic "sequence modeling" task.
- Language modelling is the "backbone" of NLP.
  - Useful for machine translation, dialogue systems, automated captioning, etc.



Training RNNs

- How can we train RNNs?
- Same as feed-forward networks: train with backpropagation on unrolled computation graph!

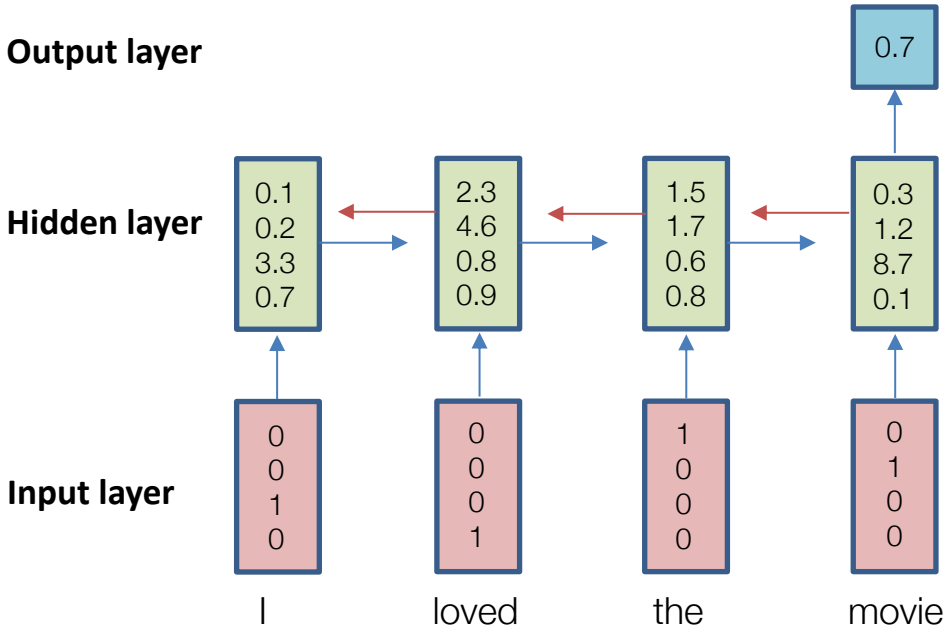
Training RNNs

- How can we train RNNs?
- Same as feed-forward networks: train with backpropagation on unrolled computation graph!

- This is called backpropagation through time (BPTT)
- Same derivation as regular backprop (use chain rule)

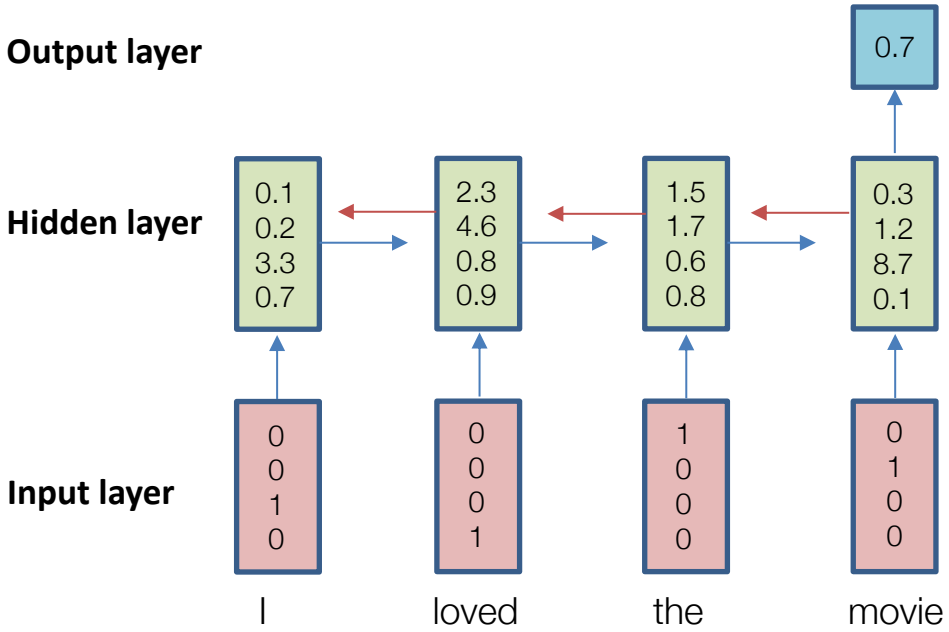
Training RNNs

- BPTT is straightforward for sequence classification.
- Gradient flows from the final prediction back through all the layers.



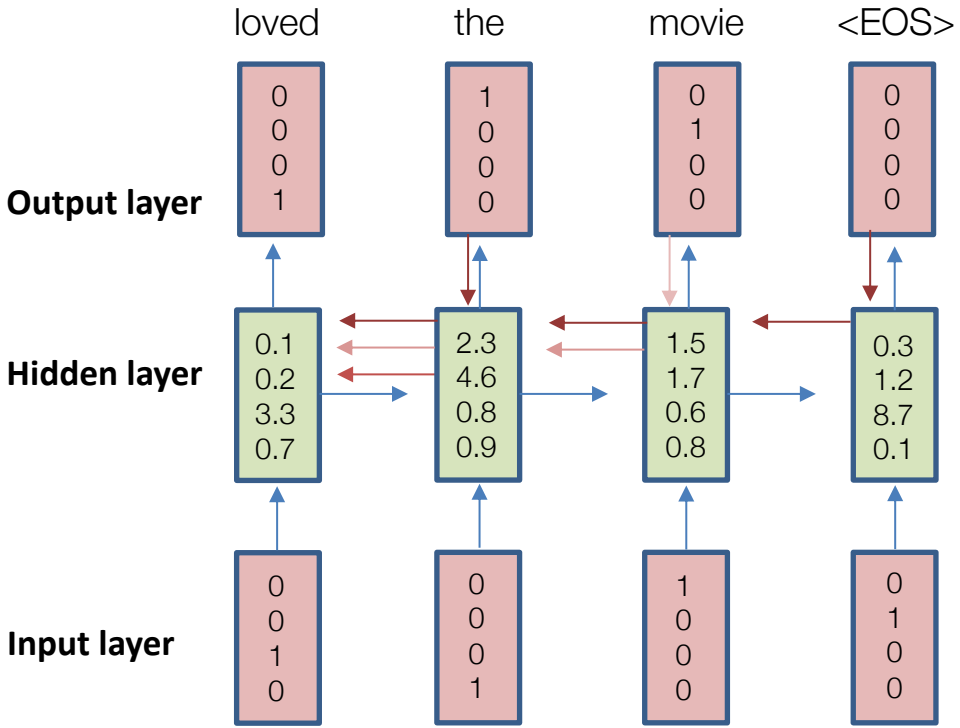
Backpropagation through time

- BPTT is straightforward for sequence classification.
- Gradient flows from the final prediction back through all the layers.



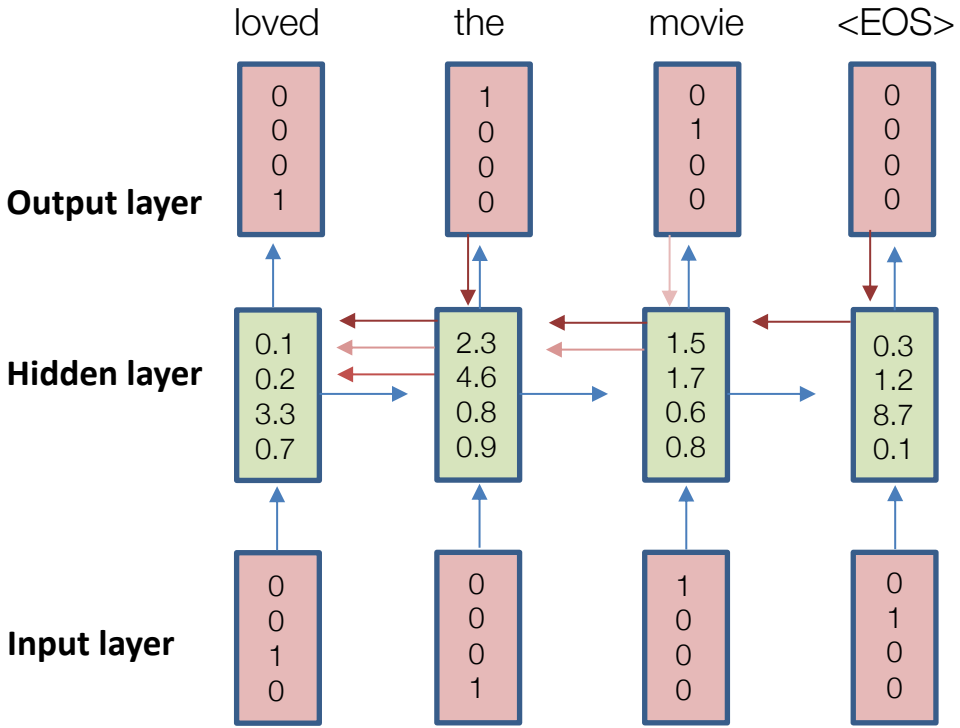
Backpropagation through time

- BPTT is less straightforward for language modeling.
- Gradient flows from the prediction at each time-step to the preceding time-steps.



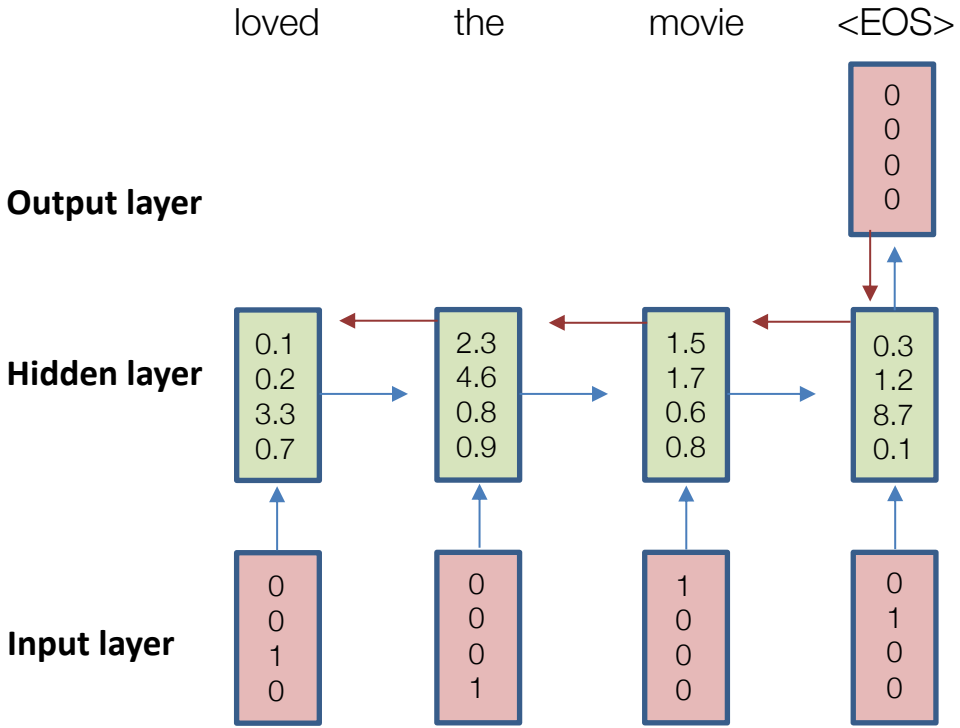
Backpropagation through time

- BPTT is less straightforward for language modeling.
- Conceptually, we can think that we are jointly training on three sequence classification tasks.



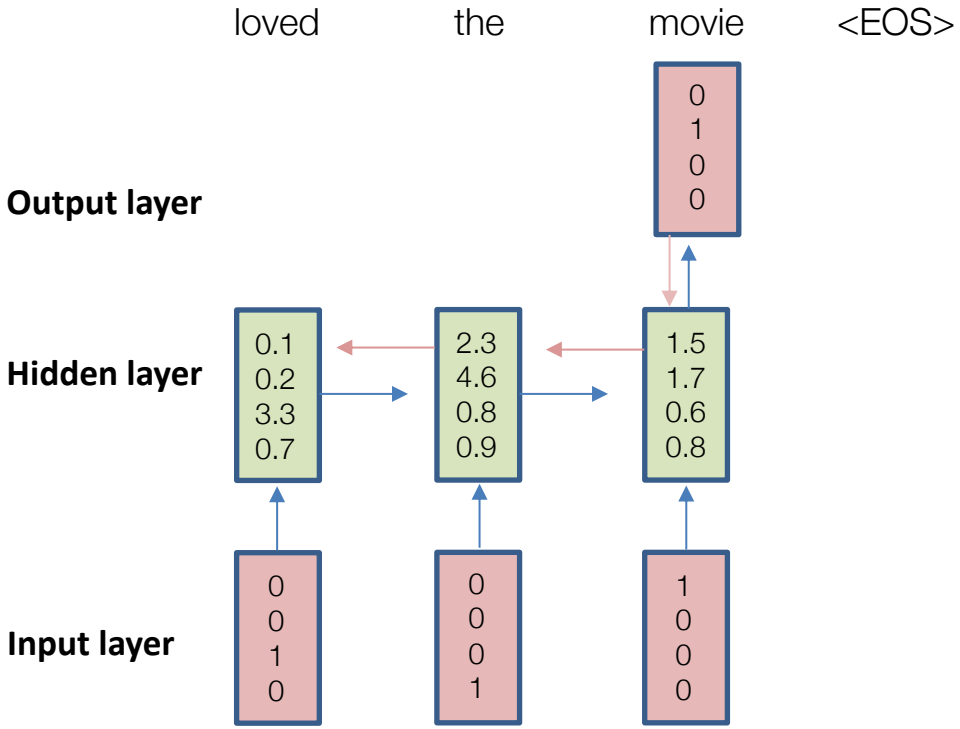
Backpropagation through time

- BPTT is less straightforward for language modeling.
- Conceptually, we can think that we are jointly training on three sequence classification tasks.



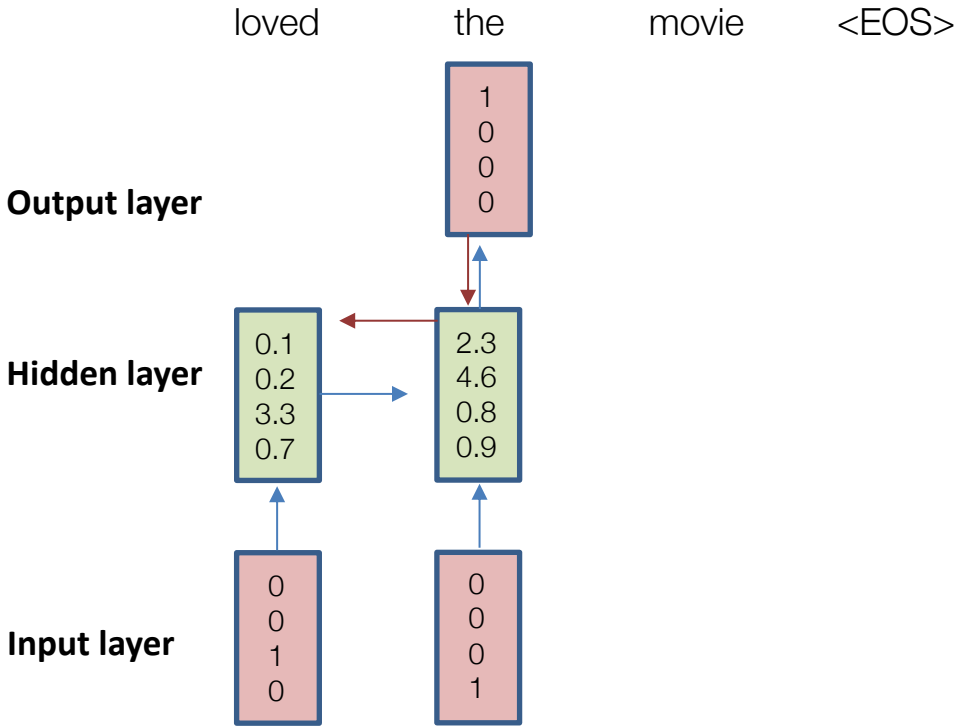
Backpropagation through time

- BPTT is less straightforward for language modeling.
- Conceptually, we can think that we are jointly training on three sequence classification tasks.



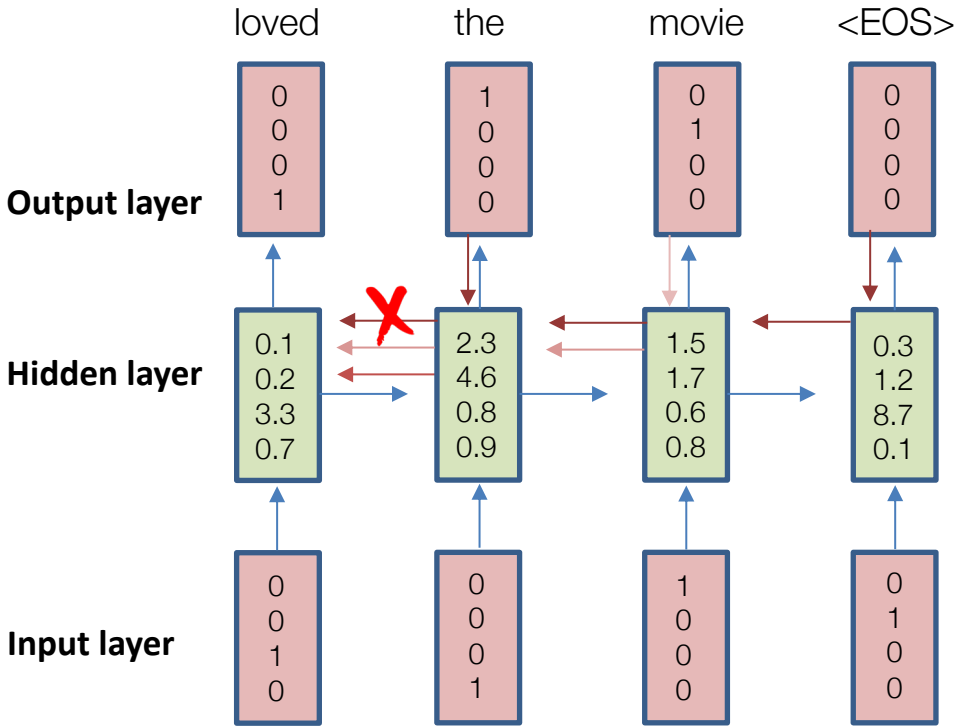
Backpropagation through time

- BPTT is less straightforward for language modeling.
- Conceptually, we can think that we are jointly training on three sequence classification tasks.



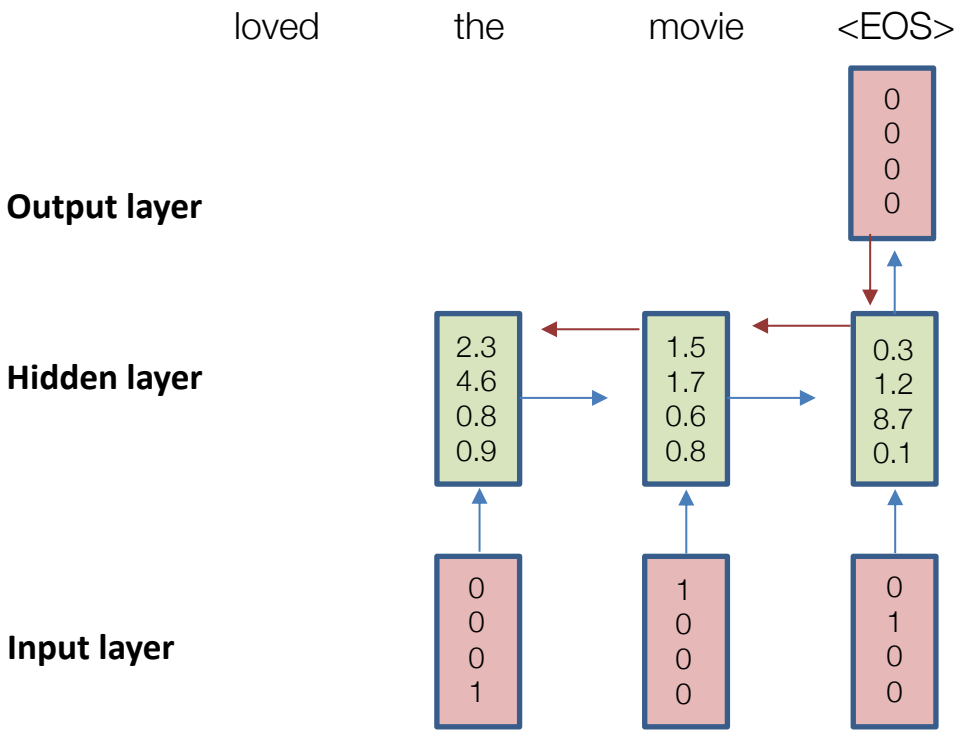
Backpropagation through time

- BPTT is less straightforward for language modeling.
- For very long sequence/language modeling tasks, sometimes we "truncate" the gradient flow.



Backpropagation through time

- BPTT is less straightforward for language modeling.
- For very long sequence/language modeling tasks, sometimes we "truncate" the gradient flow.
  - I.e., only the last K time-steps are used to train the prediction.

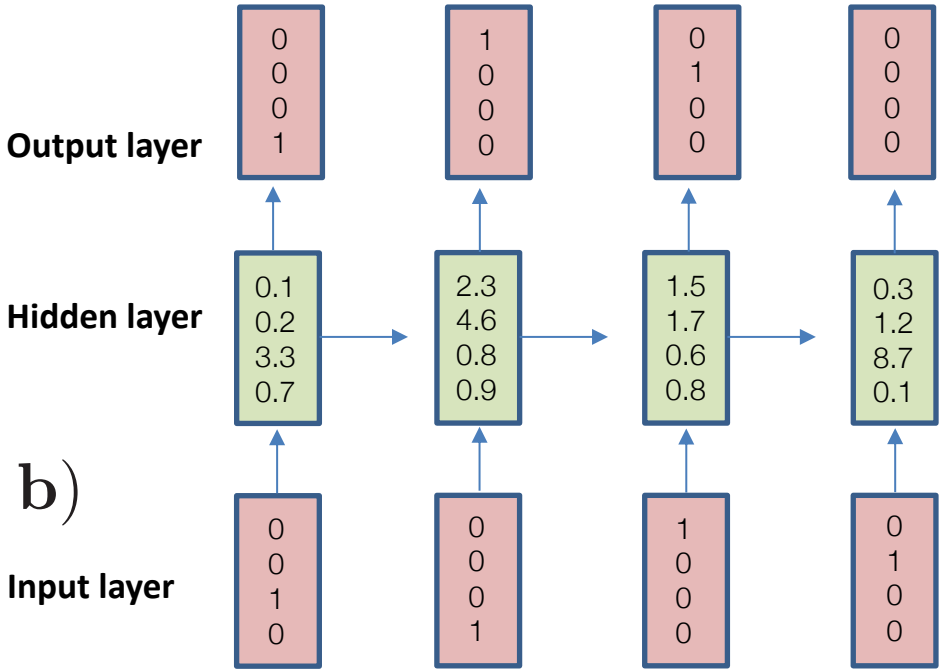


There are many ways to add recurrence

- So far, we have been considering a standard/simple RNN.
- Recurrence is between the hidden states:

$$\mathbf{h}_t = \sigma(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b})$$

- This is called the Elman RNN.

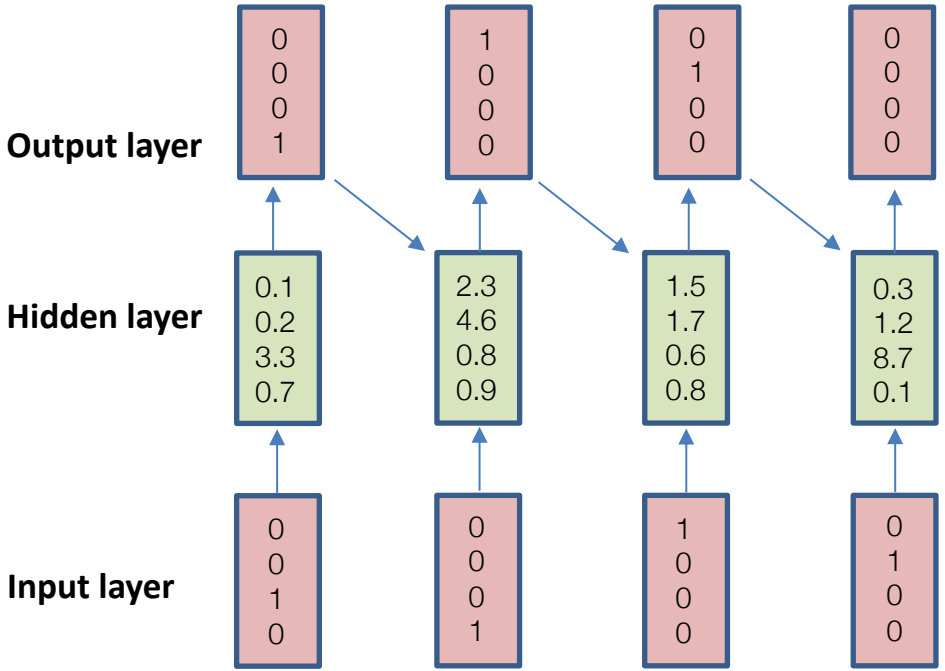


There are many ways to add recurrence

- But there are other options!
- E.g., recurrence based on the output:

$$\mathbf{h}_t = \sigma(\mathbf{W}\mathbf{o}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b})$$

- This is called the Jordan RNN.



There are many ways to add recurrence

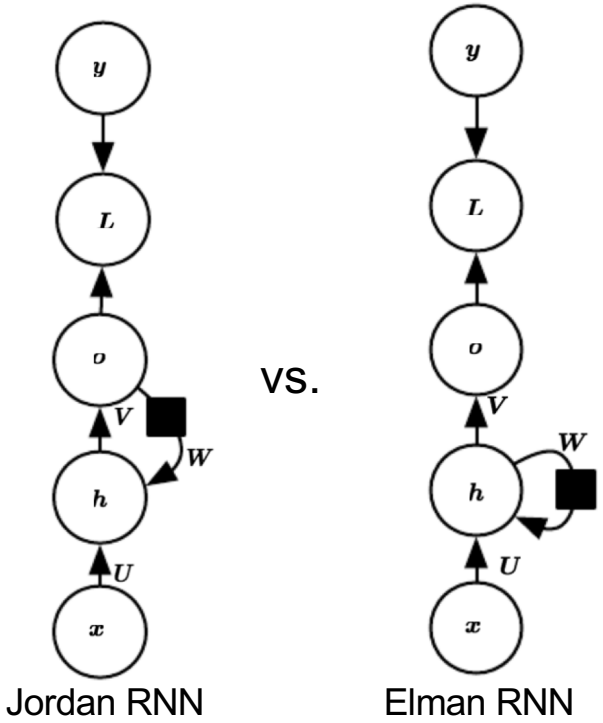
- **Q:** Which is better?

- Elman RNN:

$$\mathbf{h}_t = \sigma(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b})$$
$$\mathbf{o}_t = \phi(\mathbf{V}\mathbf{h}_t + \mathbf{c})$$

- Jordan RNN:

$$\mathbf{h}_t = \sigma(\mathbf{W}\mathbf{o}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b})$$
$$\mathbf{o}_t = \phi(\mathbf{V}\mathbf{h}_t + \mathbf{c})$$

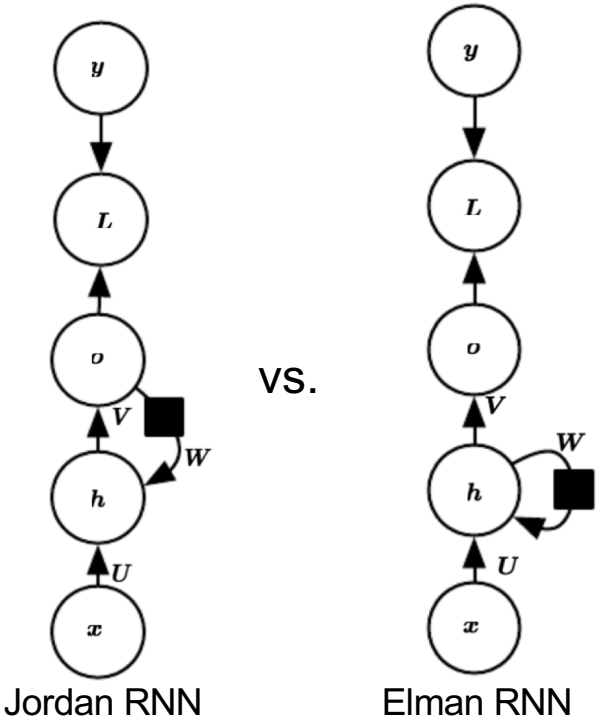


Jordan RNN Elman RNN

There are many ways to add recurrence

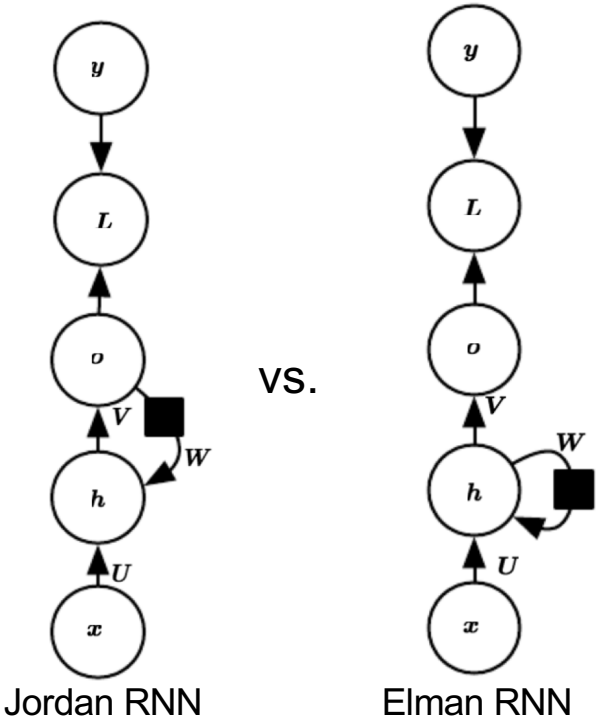
- **Q:** Which is better?

- A: Elman RNN. Usually output *o* is constrained in some way, and may be missing some important info from the past.



There are many ways to add recurrence

- **Q:** Which is better?
- A: Elman RNN. Usually output *o* is constrained in some way, and may be missing some important info from the past.
- We can also add both types of recurrence at once!



Jordan RNN Elman RNN

Beyond Elman and Jordan RNNs

- Elman and Jordan RNNs are relatively straightforward.
- But in practice they are very hard to train!
- Issue: Multiplying by the same **W** matrix over and over is very unstable…

$$\mathbf{h}_t = \sigma(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b})$$

$$\text{E.g., } \mathbf{h}_3 = \sigma(\mathbf{W}(\sigma(\mathbf{W}\sigma(\mathbf{W}\mathbf{h}_0 + \mathbf{U}\mathbf{x}_1 + \mathbf{b}) + \mathbf{U}\mathbf{x}_2\mathbf{b}) + \mathbf{U}\mathbf{x}_3 + \mathbf{b})$$

- There are recurrent architectures that fix this! (Next lecture).