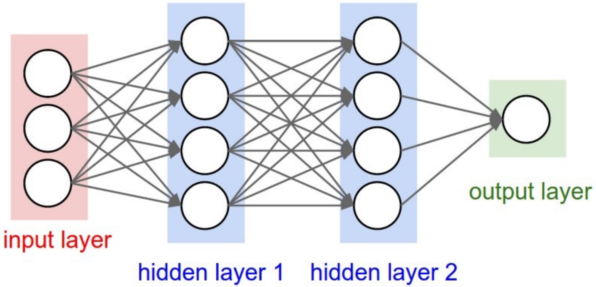
COMP 451 – Fundamentals of Machine Learning Lecture 24 — Recurrent Neural Nets

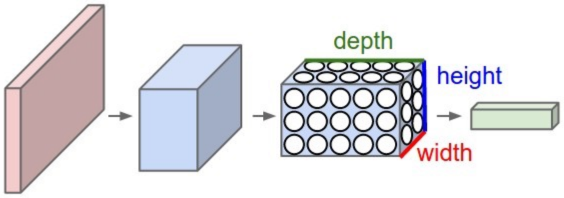
William L. Hamilton

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/*

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

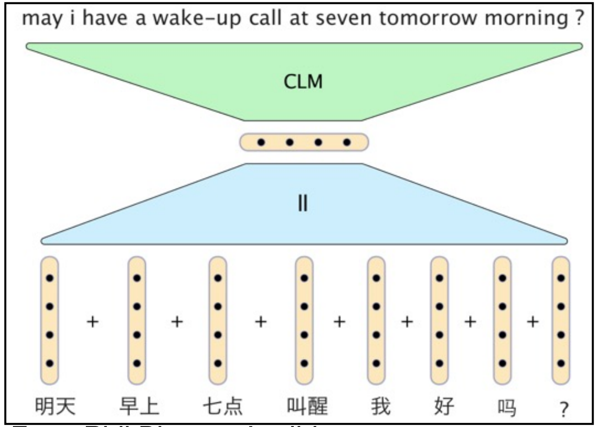
- Deep neural networks: The model should be interpreted as a computation graph.
  - Supervised training: E.g., feedforward neural networks.
  - Unsupervised training (later in the course): E.g., autoencoders.

- 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:

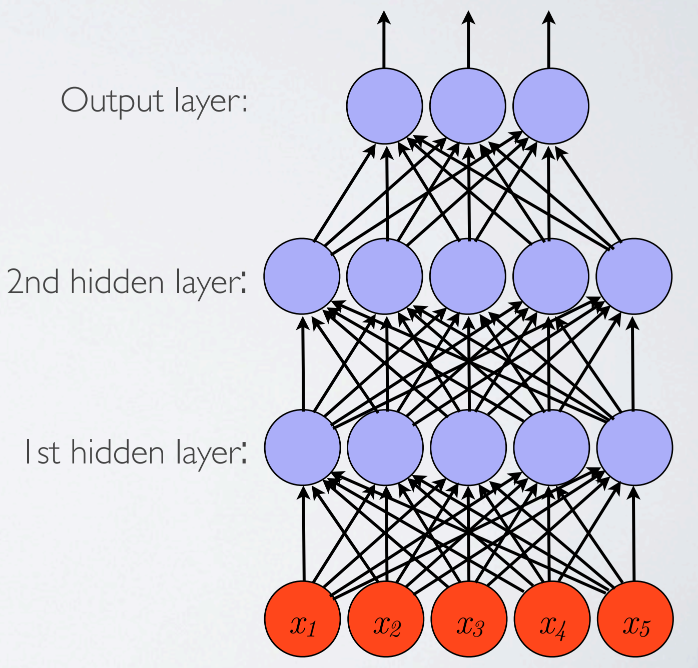
“John hit Steve on the head with a bat”

!= “Steve hit John on the bat with a head”

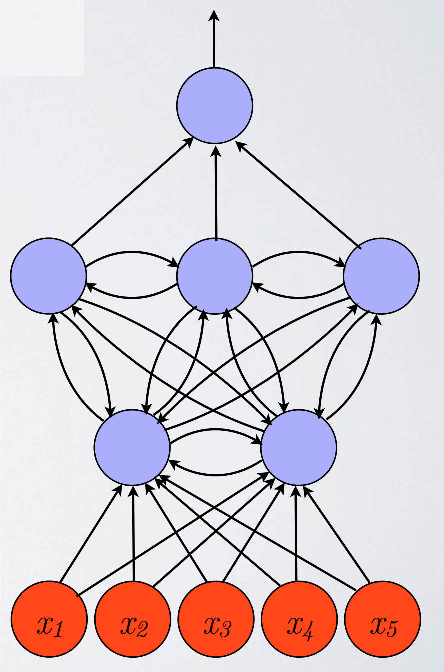
!= “Bat hit a with head on the John Steve”

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)

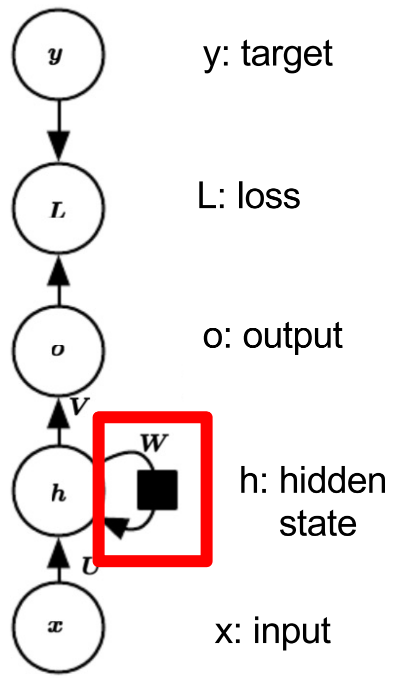


Image from deeplearningbook.org

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:

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

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

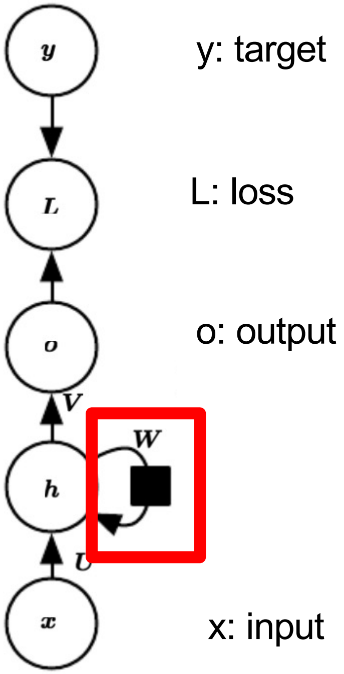
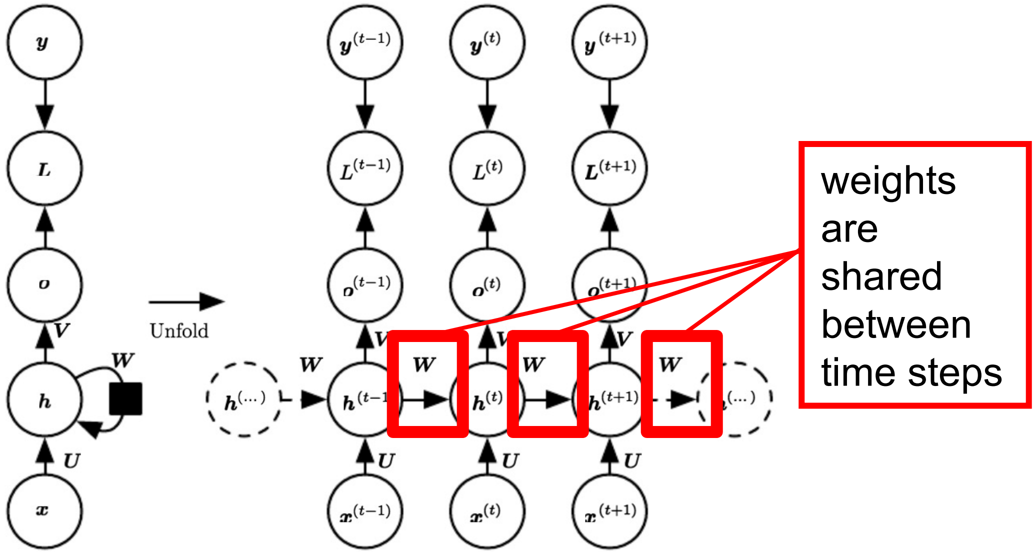


Image from deeplearningbook.org

Recurrent Neural Networks (RNNs)

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



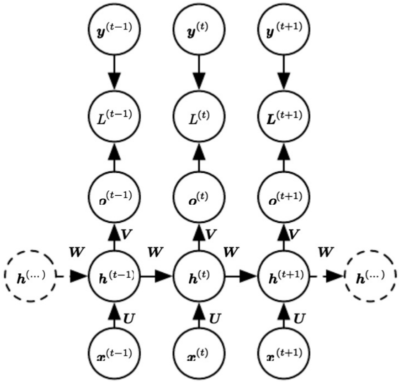
$$\text{E.g., } \mathbf{h}_3 = \sigma(\mathbf{W}(\sigma(\mathbf{W}\sigma(\mathbf{W}\mathbf{h}_0 + \mathbf{U}\mathbf{x}_1 + \mathbf{b}) + \mathbf{U}\mathbf{x}_2\mathbf{b}) + \mathbf{U}\mathbf{x}_3 + \mathbf{b})$$

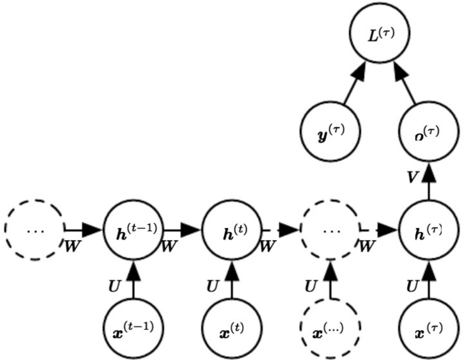
Kinds of output

- 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

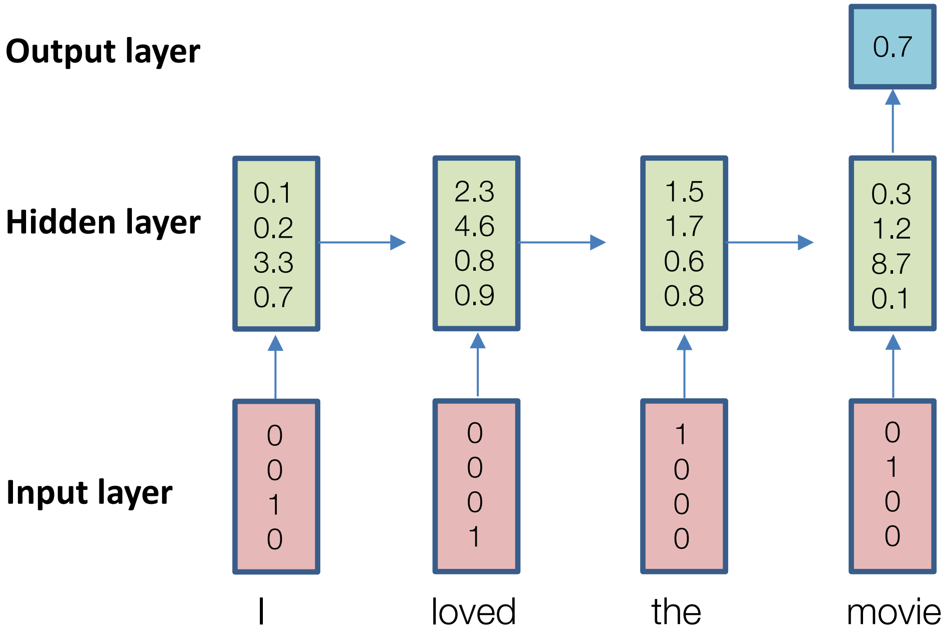
2) Can specify an target at each time step
   - Ex: generating language





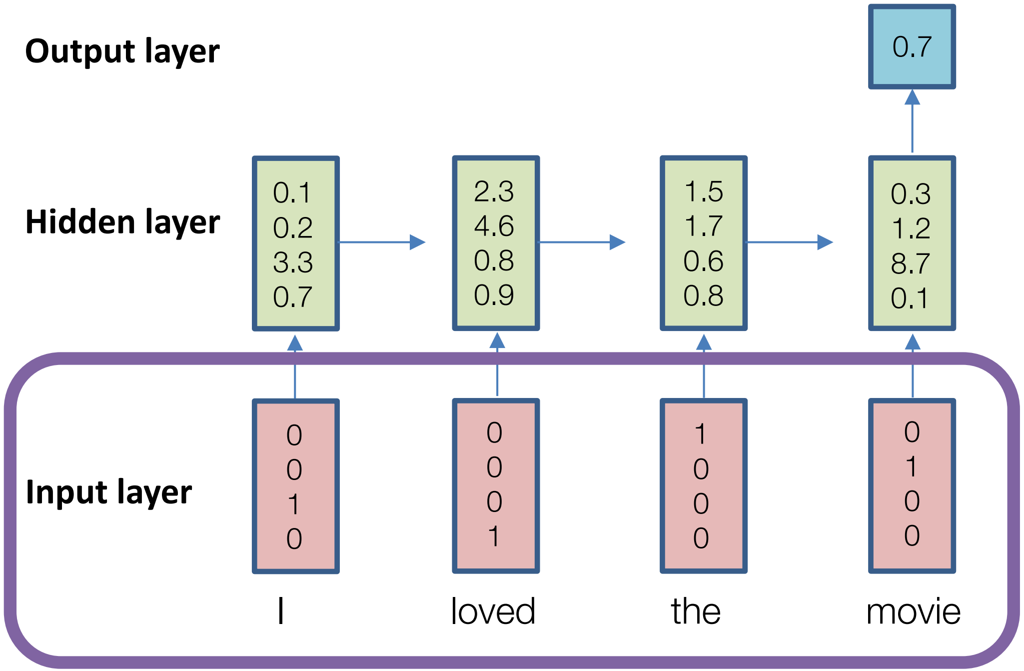
Sentiment classification

- **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



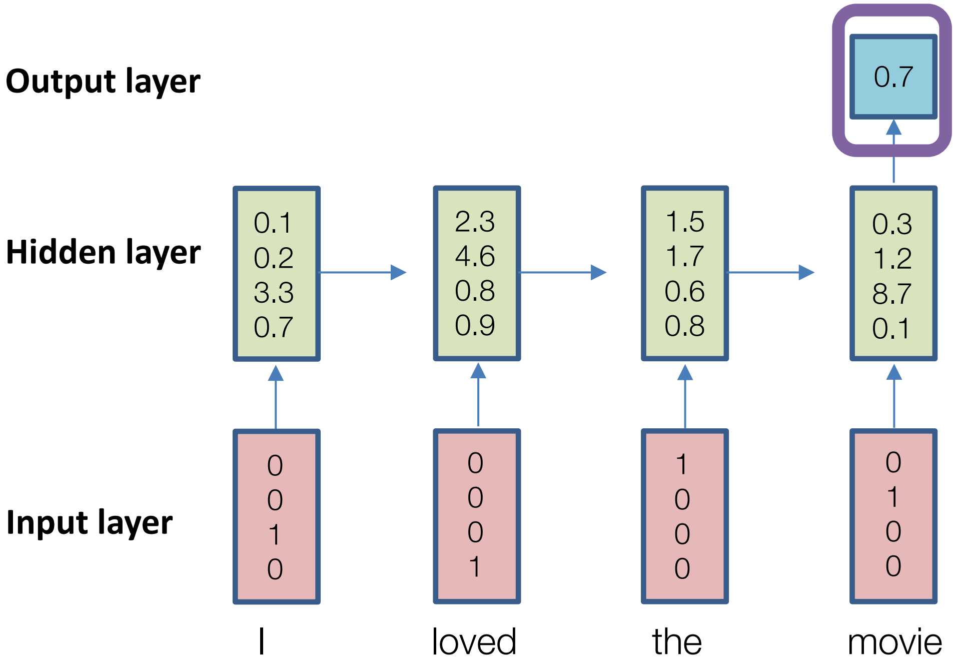
Sentiment classification

- **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.



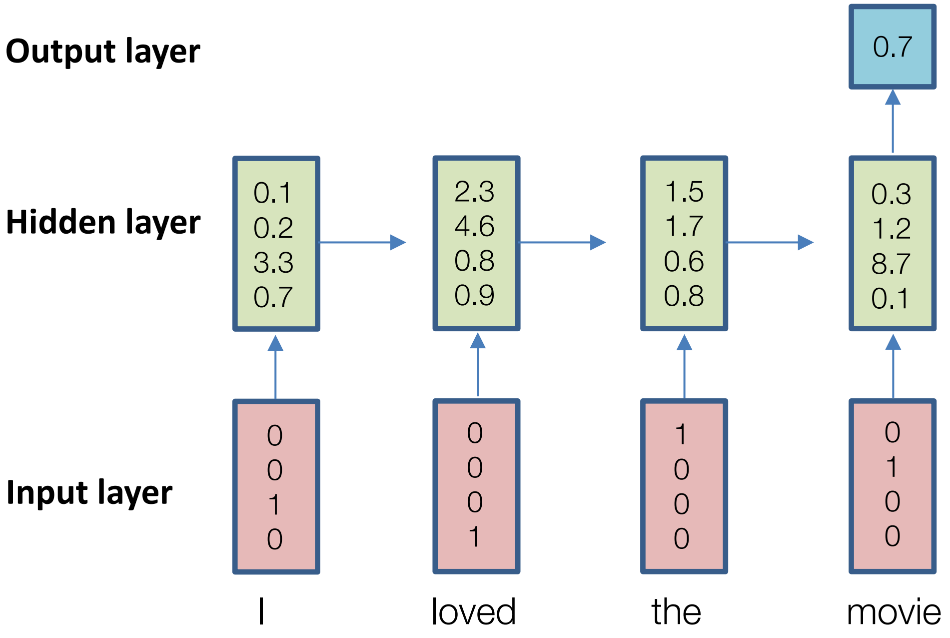
Sentiment classification

- **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



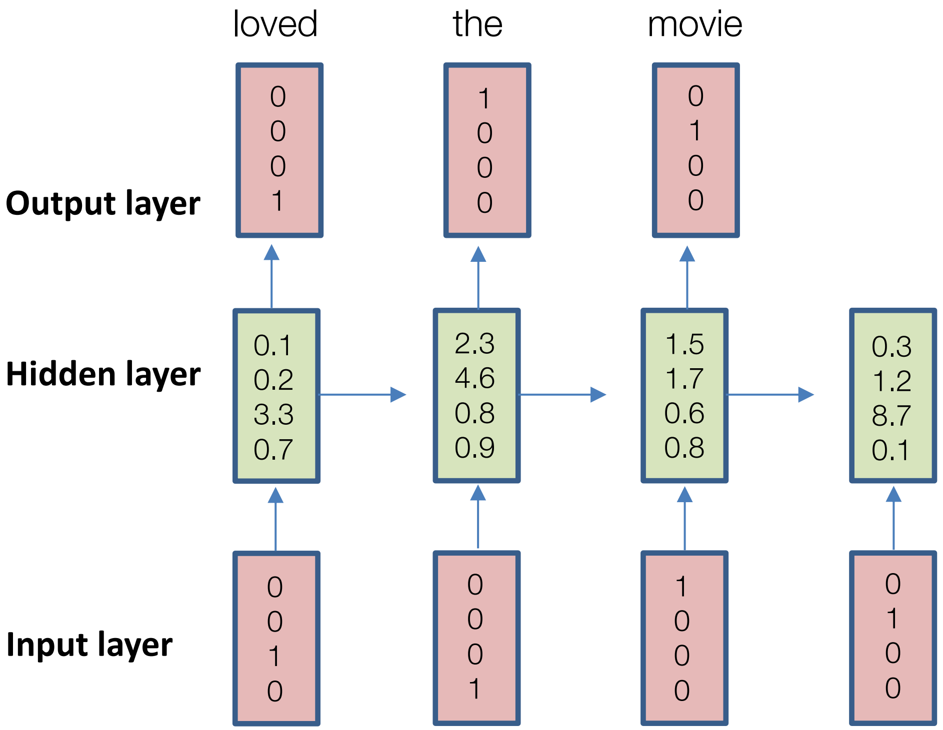
Sentiment classification

- **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.



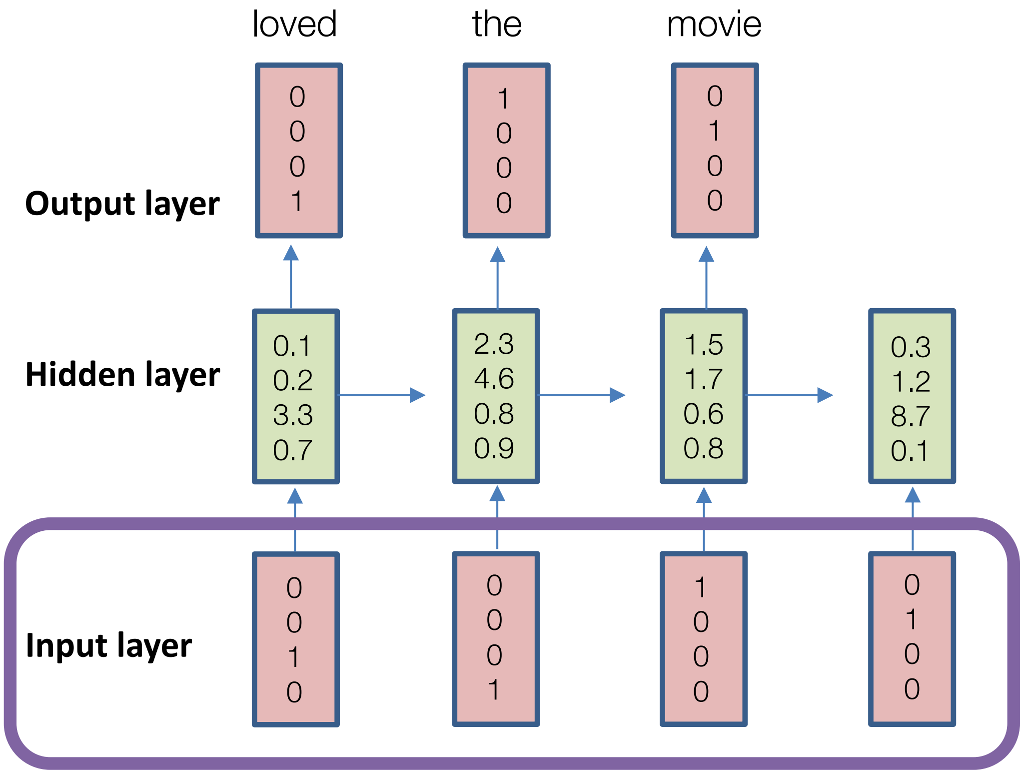
Language modeling

- **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.



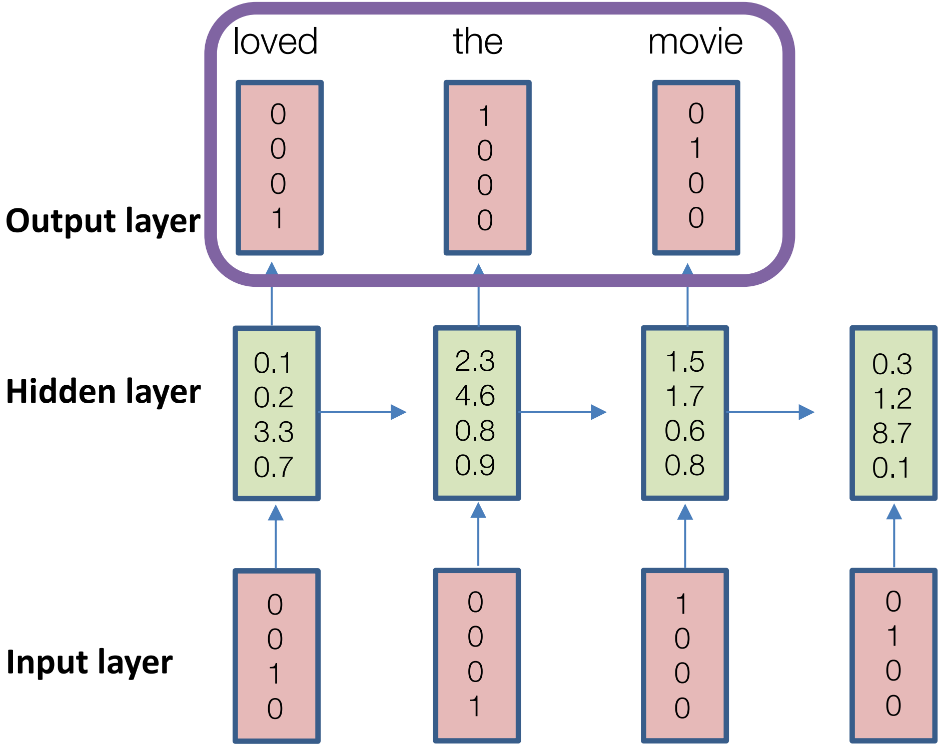
Language modeling

- **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



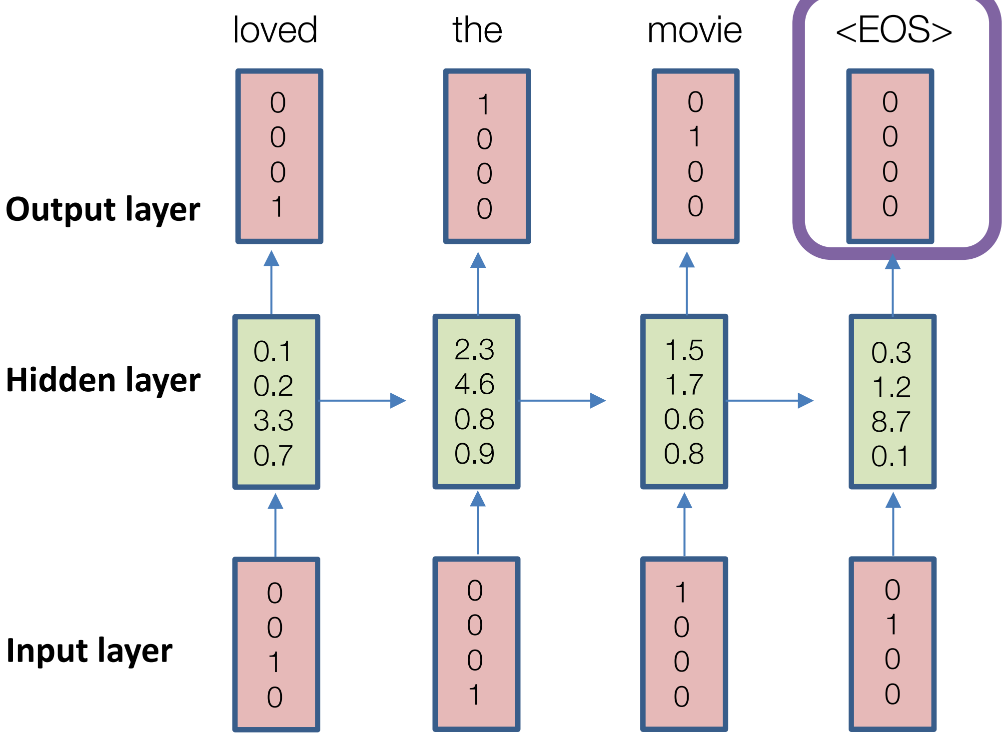
Language modeling

- **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

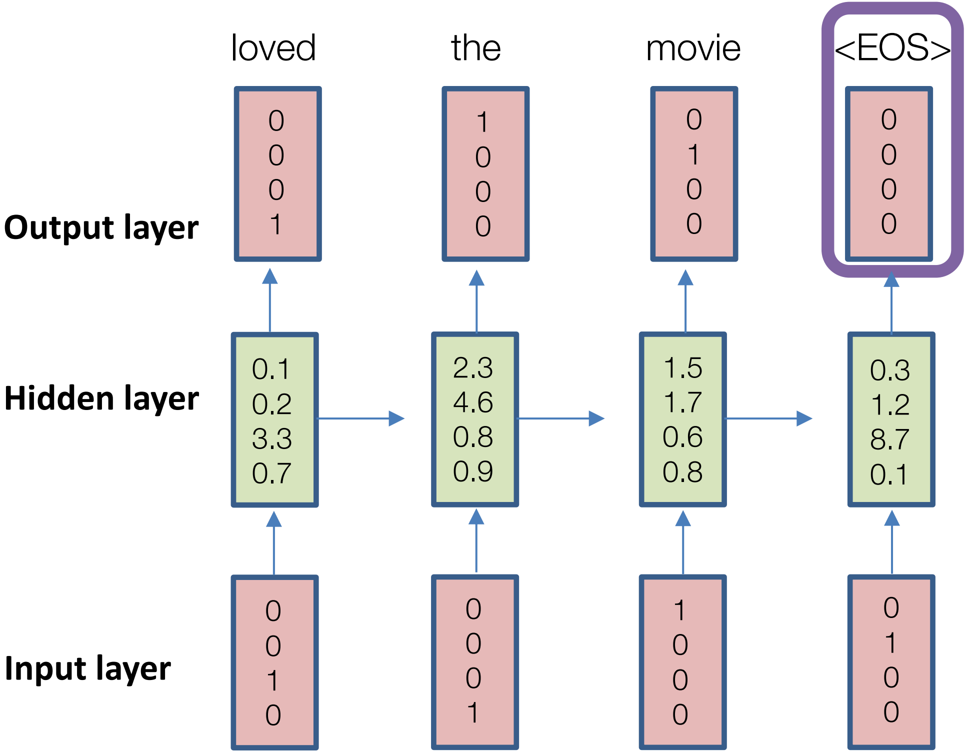
- **Input:** Sequence of words
  - E.g., review, tweet, or news article
- **Output:** Sequence of words
  - E.g., predicting the next word that will occur in a sentence.
- Usually add an "end of sentence token"



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!

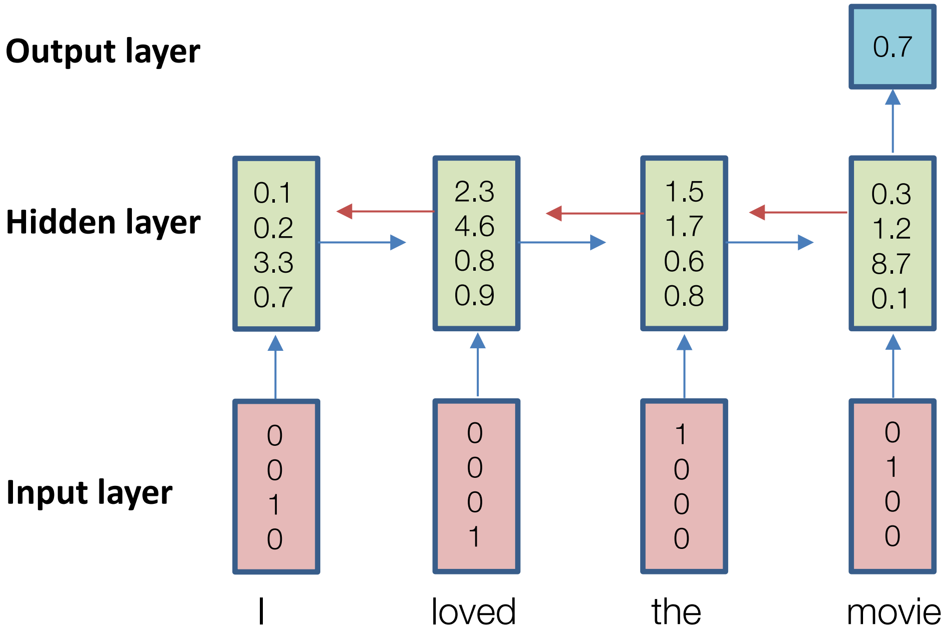
Training RNNs

- 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)

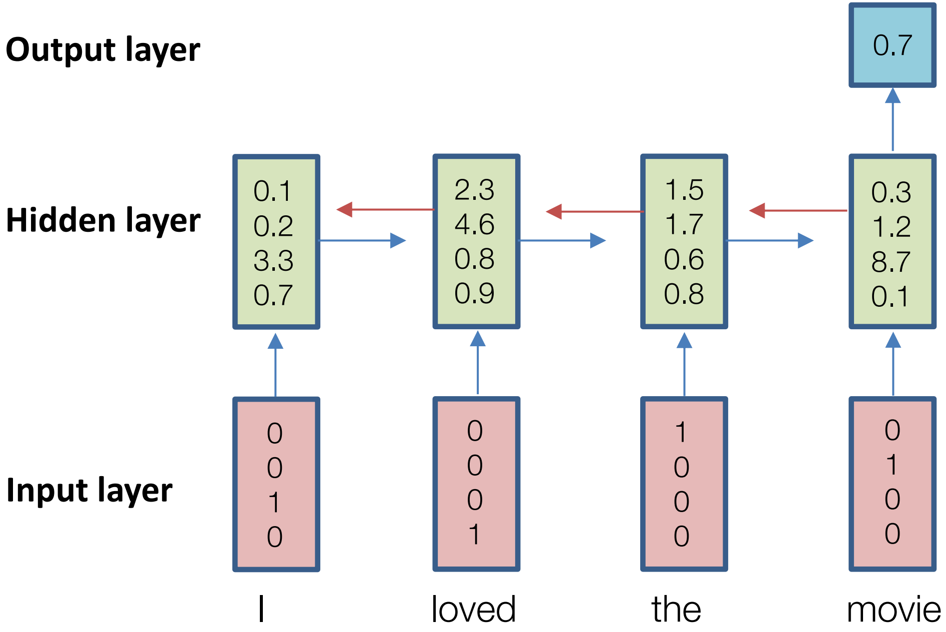
Training RNNs

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



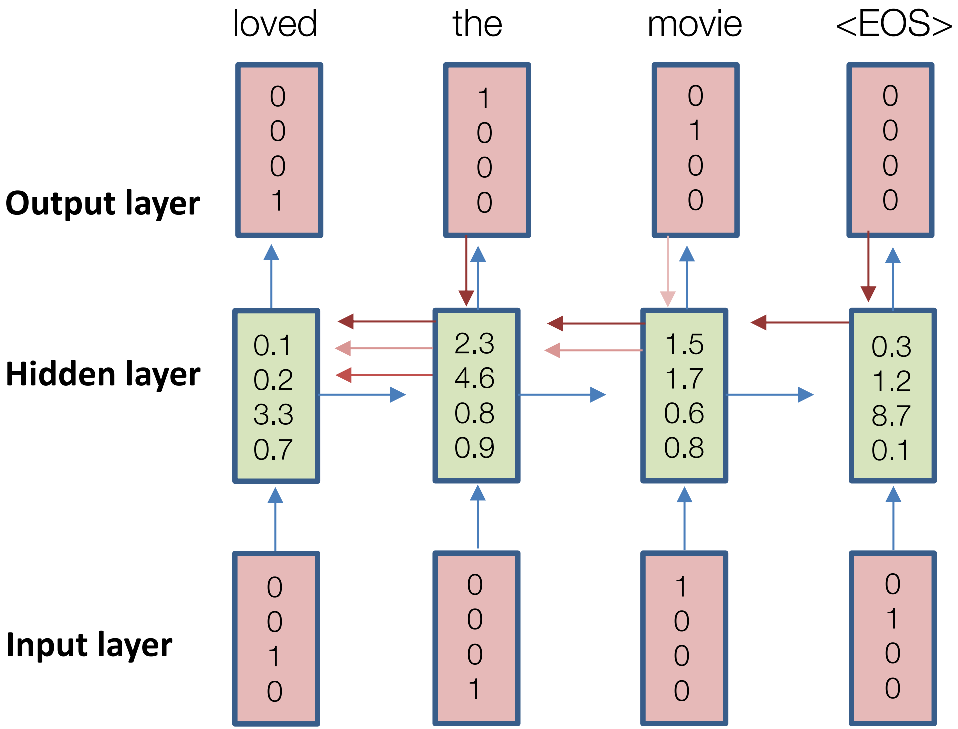
Backpropagation through time

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



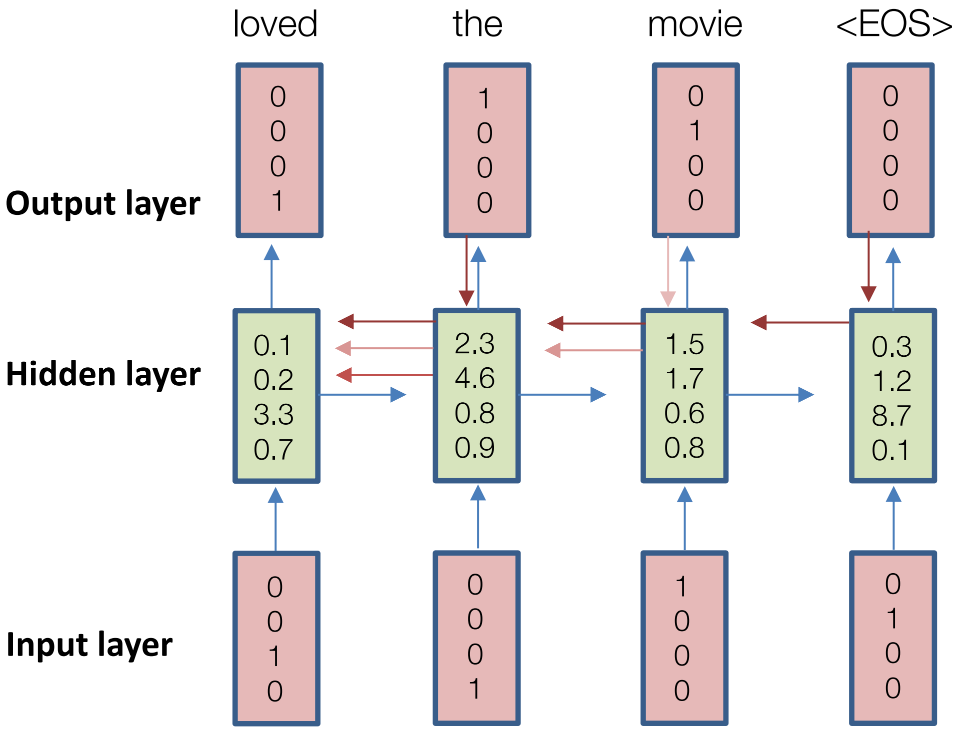
Backpropagation through time

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



Backpropagation through time

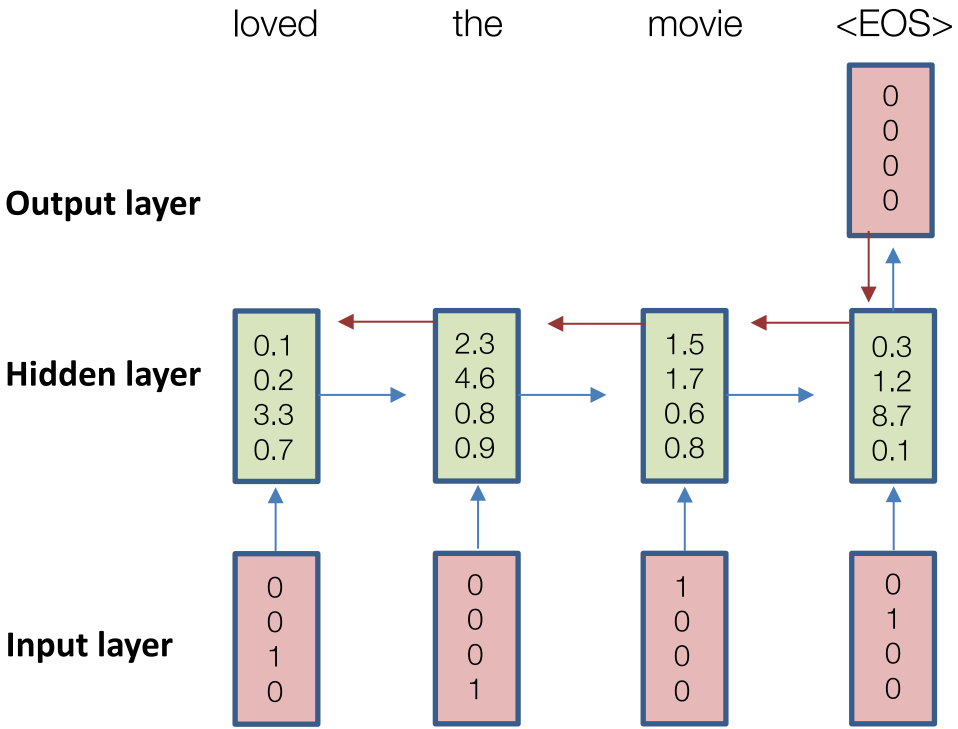
- BPTT is less straightforward for language modeling.
- Conceptually, we can think that we are jointly training on three sequence classification tasks.



Backpropagation through time

- BPTT is less straightforward for language modeling.

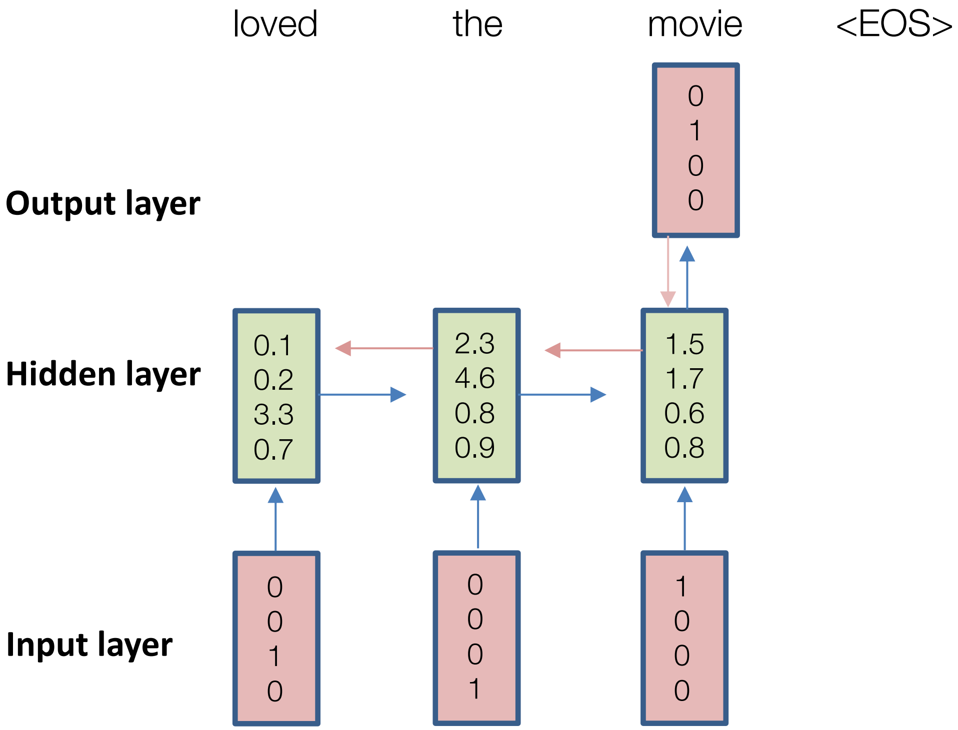
- Conceptually, we can think that we are jointly training on three sequence classification tasks.



Backpropagation through time

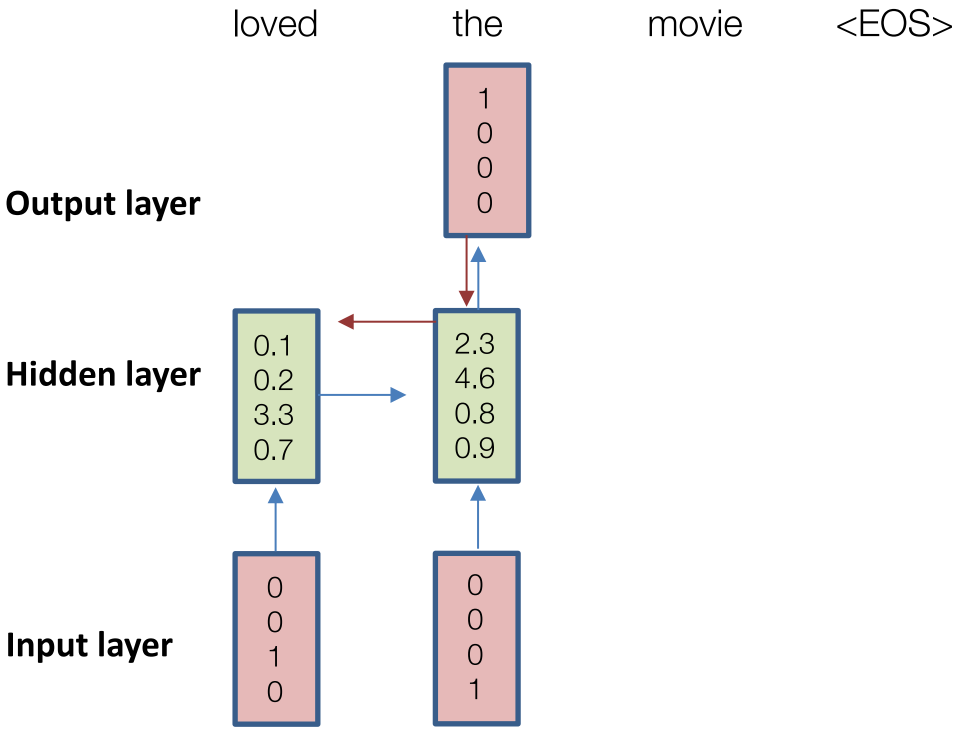
- BPTT is less straightforward for language modeling.

- Conceptually, we can think that we are jointly training on three sequence classification tasks.



Backpropagation through time

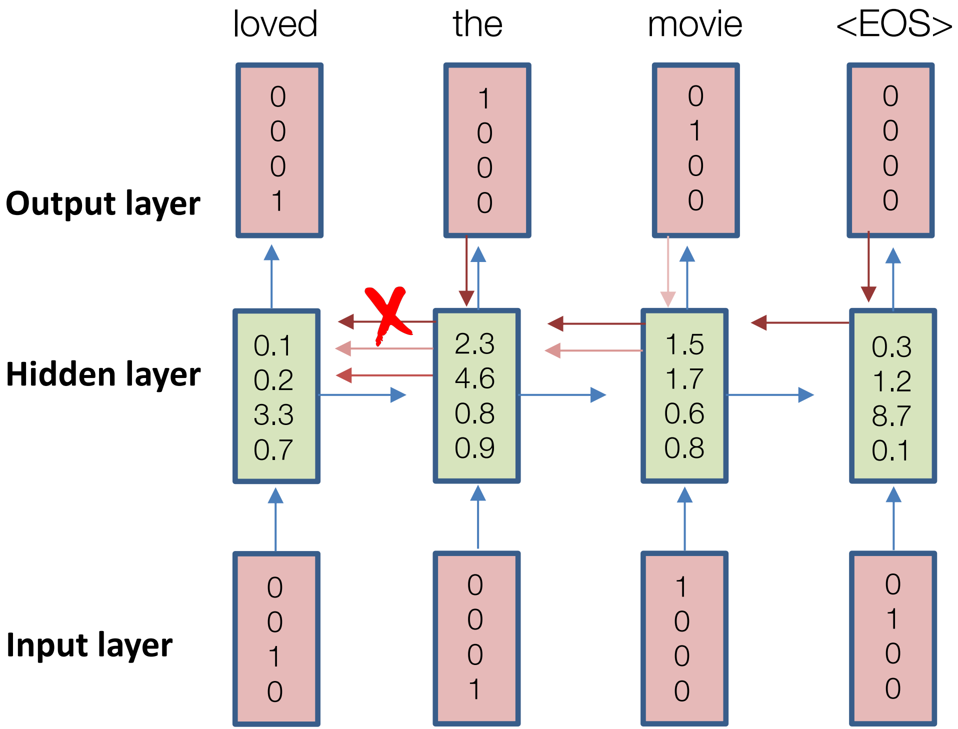
- BPTT is less straightforward for language modeling.
- Conceptually, we can think that we are jointly training on three sequence classification tasks.



Backpropagation through time

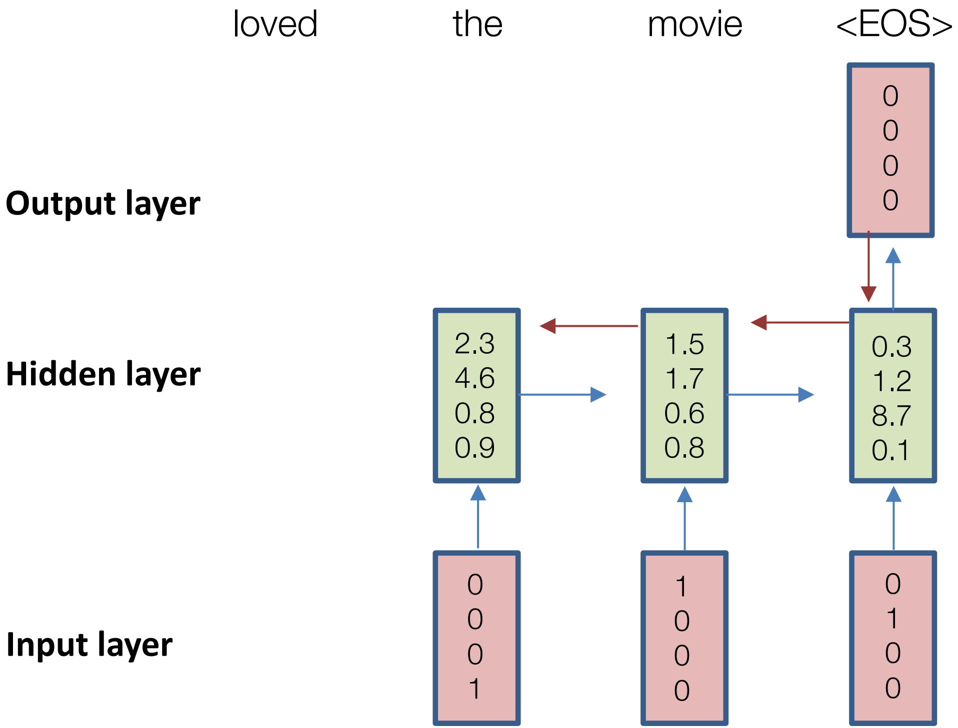
- BPTT is less straightforward for language modeling.

- 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.

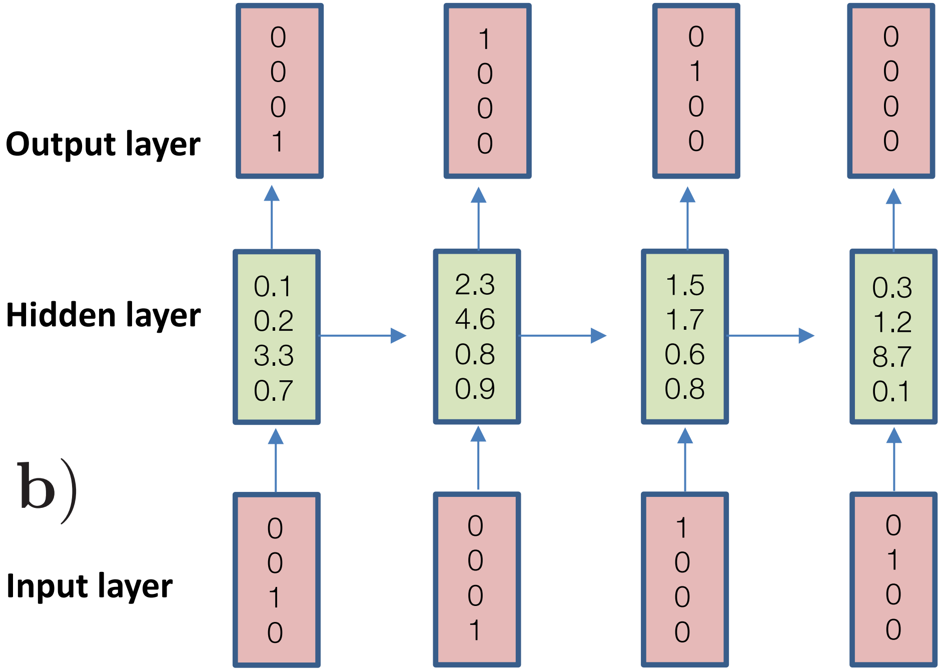


There are many ways to add recurrence

- 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.

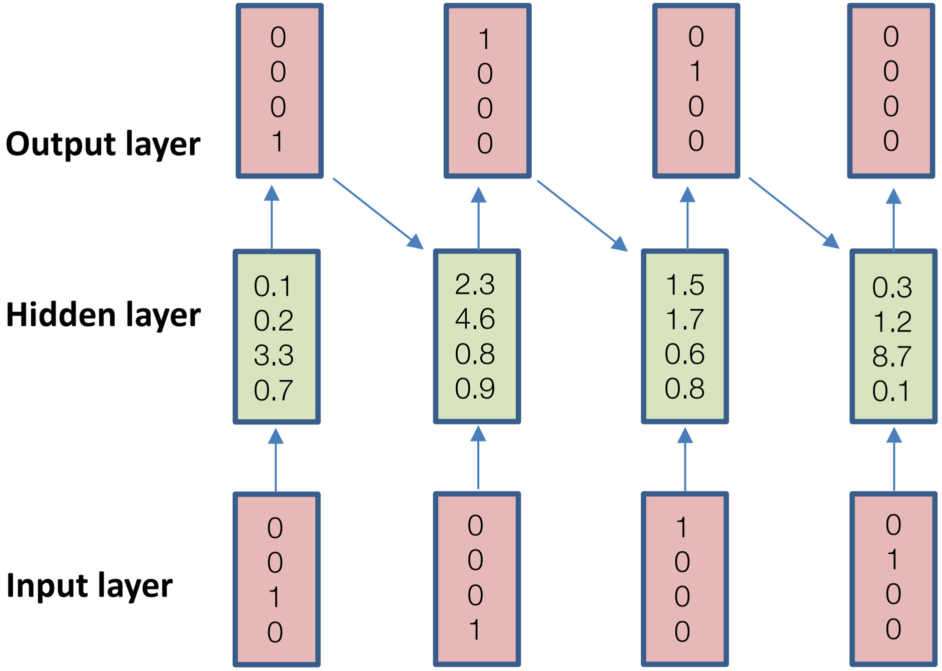


There are many ways to add recurrence

- 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.



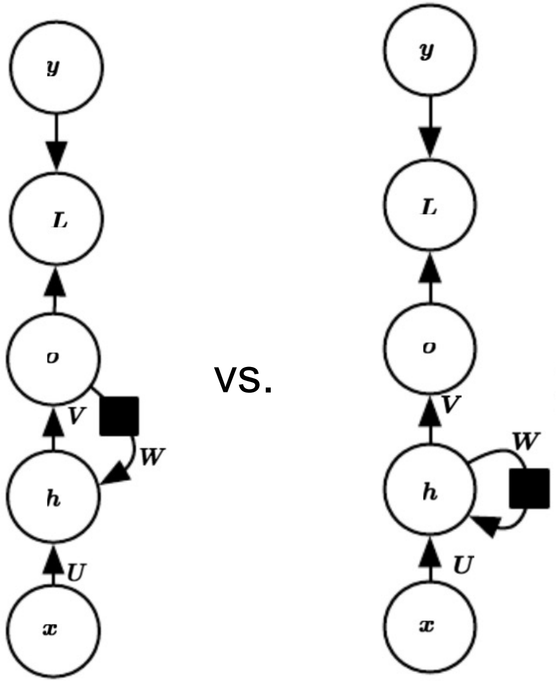
There are many ways to add recurrence

- 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:

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

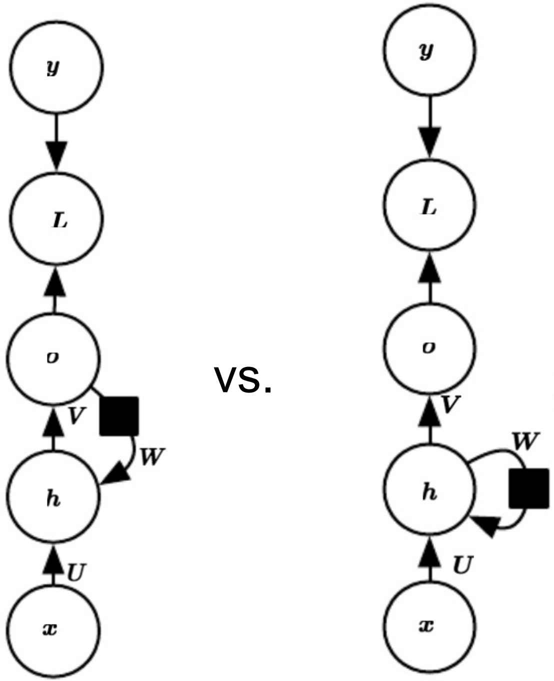


Jordan RNN Elman RNN

There are many ways to add recurrence

- Q: Which is better?

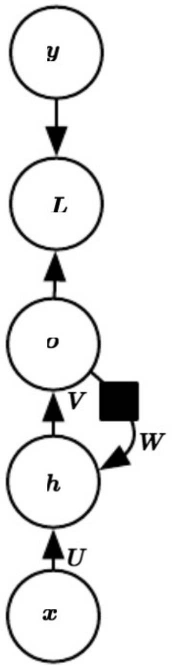
- A: Elman RNN. Usually output *o* is constrained in some way, and may be missing some important info from the past.



Jordan RNN Elman RNN

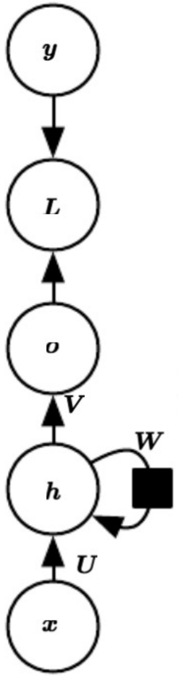
There are many ways to add recurrence

- 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.
- We can also add both types of recurrence at once!



Jordan RNN

vs.



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 $\mathbf{W}$ matrix over and over is very unstable…

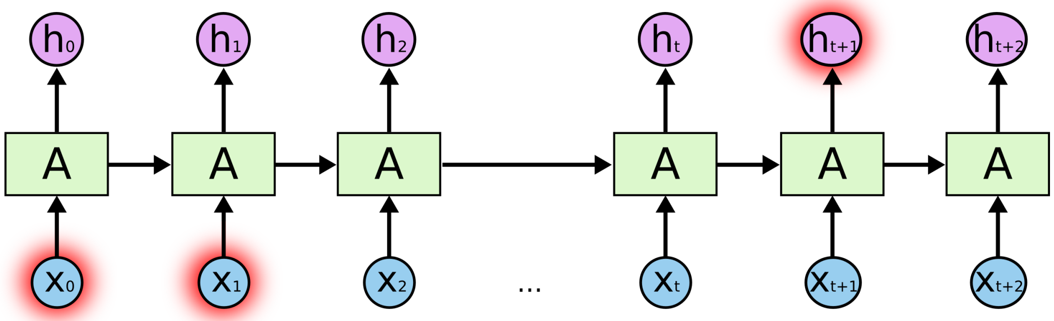
$$\mathbf{h}_t = \sigma(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b})$$

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

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

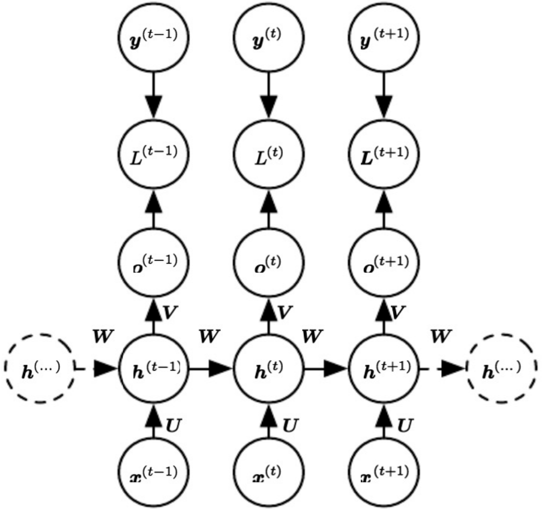
The problem of long-term dependencies

- Let's say we are doing language modelling
- Input paragraph: *"I grew up in France. I worked at […]. I speak fluent French."*
- Want to predict 'French' given words before. This can be hard!
- In practice it is very hard for RNNs to to learn dependencies lasting many time steps.
- Why could this be?



The problem of long-term dependencies

- Because the hidden-to-hidden transition matrix **W** is **the same for each time step**, this can cause the gradients to explode or vanish
- Intuition: Imagine multiplying a scalar number w by itself many times. w^{k} for $\mathrm{k} \to \infty$ will either explode (if $\mathrm{w} > 1$) or vanish (if $\mathrm{w} < 1$)
- Similar behavior occurs if **W** is a matrix



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

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

The problem of long-term dependencies

- Recall: a way to intuitively think of backpropagating gradients

- If I change my input by a small amount, what will be the result on the output?

  ⇔

  If I want my output (loss) to decrease, how do I change my input?

- If input is being multiplied by the same W many times, this could cause either a huge or tiny effect on the output.

  =>

  The gradient of loss w.r.t parameters could be huge or tiny.

The problem of long-term dependencies

- Perspective from linear algebra (eigendecomposition)
- Consider a simplified "linear" RNN with following recurrence:

$$\mathbf{h}_t = \mathbf{W}\mathbf{h}_{t-1}$$

The problem of long-term dependencies

- Perspective from linear algebra (eigendecomposition)
- Consider a simplified "linear" RNN with following recurrence:

$$\mathbf{h}_t = \mathbf{W}\mathbf{h}_{t-1}$$

- Now, we can get the eigendecomposition of $\mathbf{W}$ as:

$$\mathbf{W} = \mathbf{Q}\mathbf{D}\mathbf{Q}^\top$$

where $\mathbf{Q}$ is an orthogonal matrix of eigenvectors and $\mathbf{D}$ a matrix with eigenvalues on the diagonal.

The problem of long-term dependencies

- Perspective from linear algebra (eigendecomposition)
- Consider a simplified "linear" RNN with following recurrence:

$$\mathbf{h}_t = \mathbf{W}\mathbf{h}_{t-1}$$

- Now, we can get the eigendecomposition of $\mathbf{W}$ as:

$$\mathbf{W} = \mathbf{Q}\mathbf{D}\mathbf{Q}^\top$$

where $\mathbf{Q}$ is an orthogonal matrix of eigenvectors and $\mathbf{D}$ a matrix with eigenvalues on the diagonal.

- And, thus:

$$\begin{aligned}\mathbf{h}_t &= \mathbf{W}^t\mathbf{h}_0 \\ &= (\mathbf{Q}\mathbf{D}\mathbf{Q}^\top\mathbf{Q}\mathbf{D}\mathbf{Q}^\top\ldots)\mathbf{h}_0 \\ &= \mathbf{Q}\mathbf{D}^t\mathbf{Q}^\top\mathbf{h}_0 \qquad \text{since } \mathbf{Q}^\top\mathbf{Q} = \mathbf{I}\end{aligned}$$

- So each eigenvalue is raised to the power of $\mathbf{t}$, causing eigenvalues < 1 to vanish and eigenvalues > 1 to explode.

How to avoid vanishing/exploding gradients?

- Simple way to avoid exploding gradients: gradient clipping

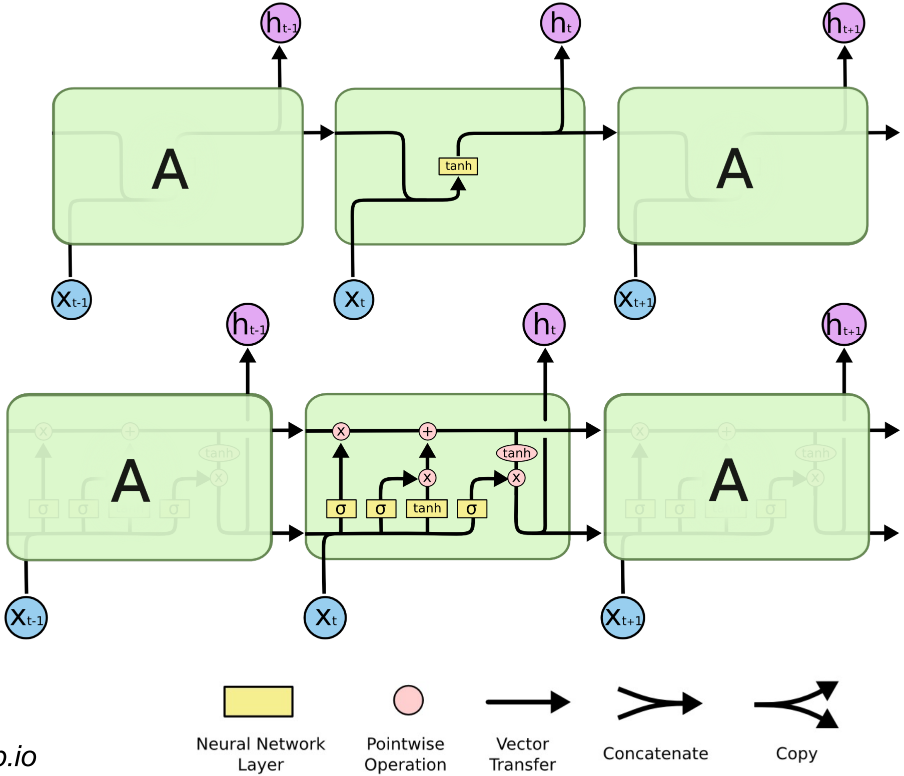
```
if |gradient| > threshold:
    gradient = threshold * sign(gradient)
```

- Another way: change the architecture of the RNN so there are some non-multiplicative interactions
  - E.g., long short-term memory (LSTM) units

Long short-term memory (LSTM) units

RNN:

LSTM:



LSTM images from: colah.github.io

Long short-term memory (LSTM) units

- Much better at dealing with long-term dependencies
- Can think of it as a special 'cell'
- Governed by a set of update equations:

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

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

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

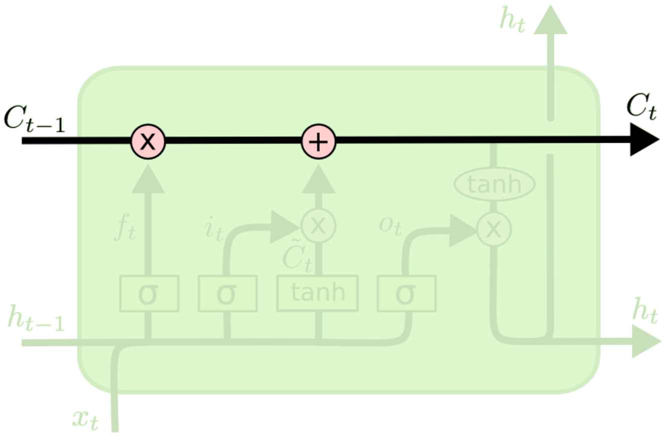
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

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

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

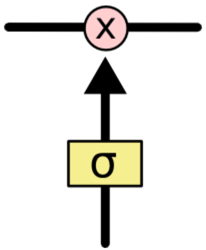
LSTMs

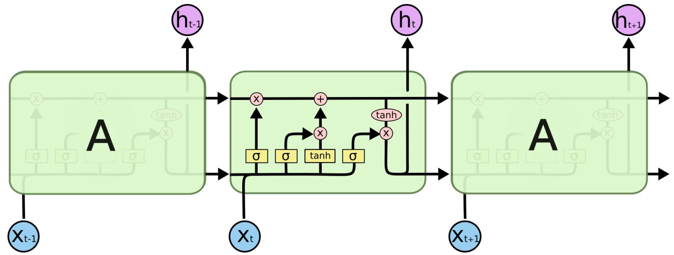
- Core idea: the **cell state is an** an 'information highway'
- Cell state is updated additively based on input, rather than multiplicatively => *less prone to exploding/ vanishing gradients*



LSTMs: Cell states, hidden states, and gating

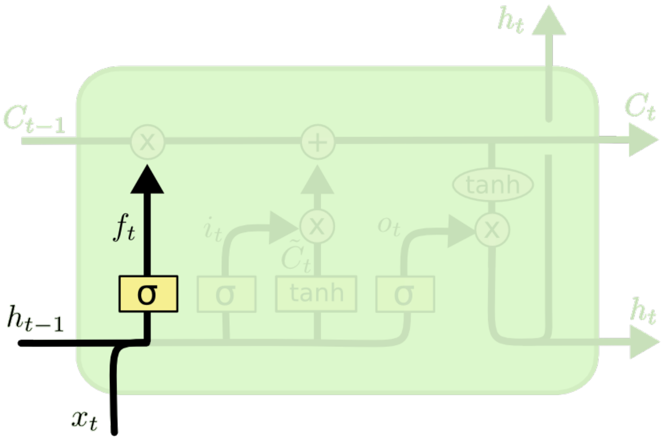
- Cell state vs hidden state (roughly)
  - Hidden state: what info from past do I need **to make my next prediction**?
  - Cell state: what info from past might I need **to make future predictions**?
- For regular RNN, hidden state plays both of these roles
- LSTM uses a set of 'gates' to control information flow
  - Gate = sigmoid layer + element-wise multiplication. Gives vector of numbers between [0,1] that determine how much of each component to let through:





LSTMs: Forget gate

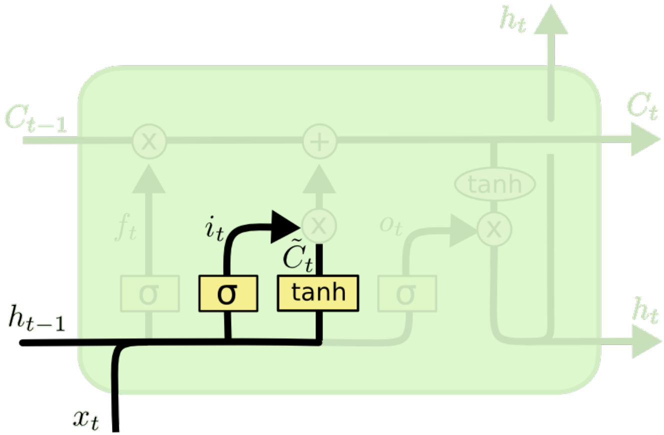
- Forget gate: how much information do we want to keep from the previous cell state?



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

LSTMs: Input gate

- Input gate: what information from the current input (and previous hidden state) do we want to transfer to the cell state?

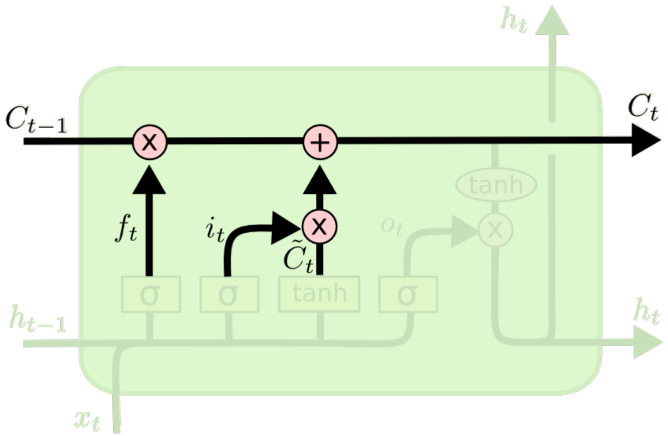


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

LSTMs: Cell update

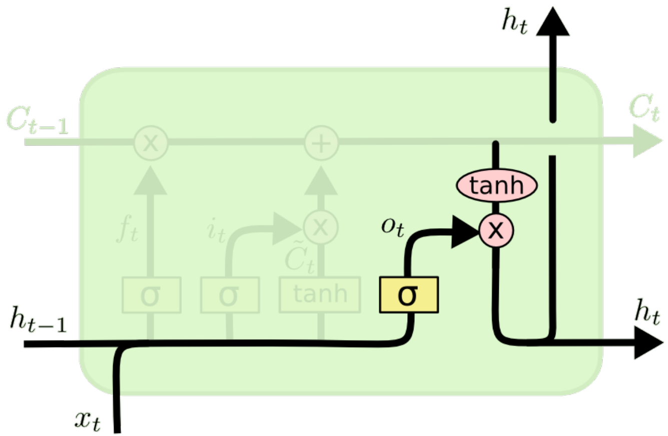
- Cell state updated as an **additive linear combination** of old cell state and processed input:



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

LSTMs: Output gate:

- Output gate: what information from the cell state do we need to make the next prediction?



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

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

LSTMs

- LSTM architecture has existed for many years (Hochreiter & Schmidhuber 1997).

- Many state-of-the-art results, e.g.,
  - Cursive handwriting recognition (Graves & Schmidhuber, 2009)
  - Speech recognition (Graves, Mohamed & Hinton, 2013)
  - Machine translation (Sutskever, Vinyals & Le, 2014)
  - Question-answer (Weston et al., 2015)
  - Unstructured dialogue response generation (Serban et al., 2016)

- Other similar models can be used (e.g. Gated Recurrent Units)