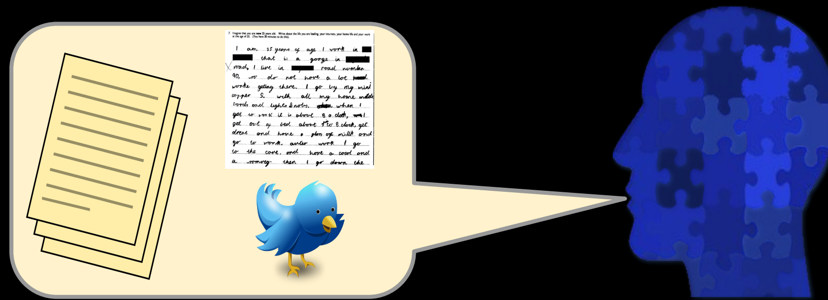
Recurrent Neural Networks for Language Modeling

CSE392 - Spring 2019
Special Topic in CS

Tasks



- Language Modeling: Generate next word, sentence ≈ capture hidden representation of sentences.

how?

- Recurrent Neural Network and Sequence Models

Language Modeling

Task: Estimate $P(w_n \mid w_1, w_2, \ldots, w_{n-1})$

:probability of a next word given history

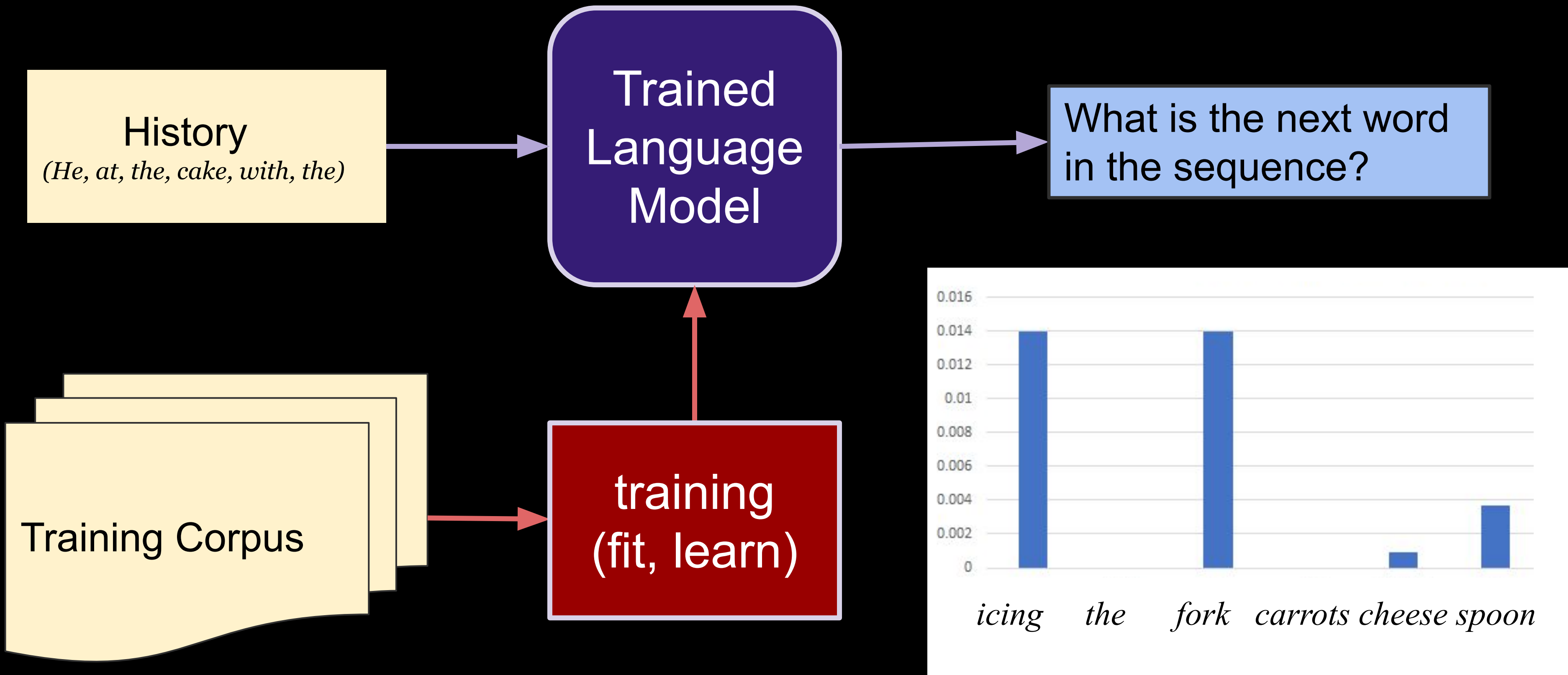
$P(fork \mid He\ ate\ the\ cake\ with\ the) = ?$

Language Modeling

Task: Estimate $P(w_n \mid w_1, w_2, \ldots, w_{n-1})$

:probability of a next word given history

P(fork | He ate the cake with the) = ?



Neural Networks: Graphs of Operations (excluding the optimization nodes)

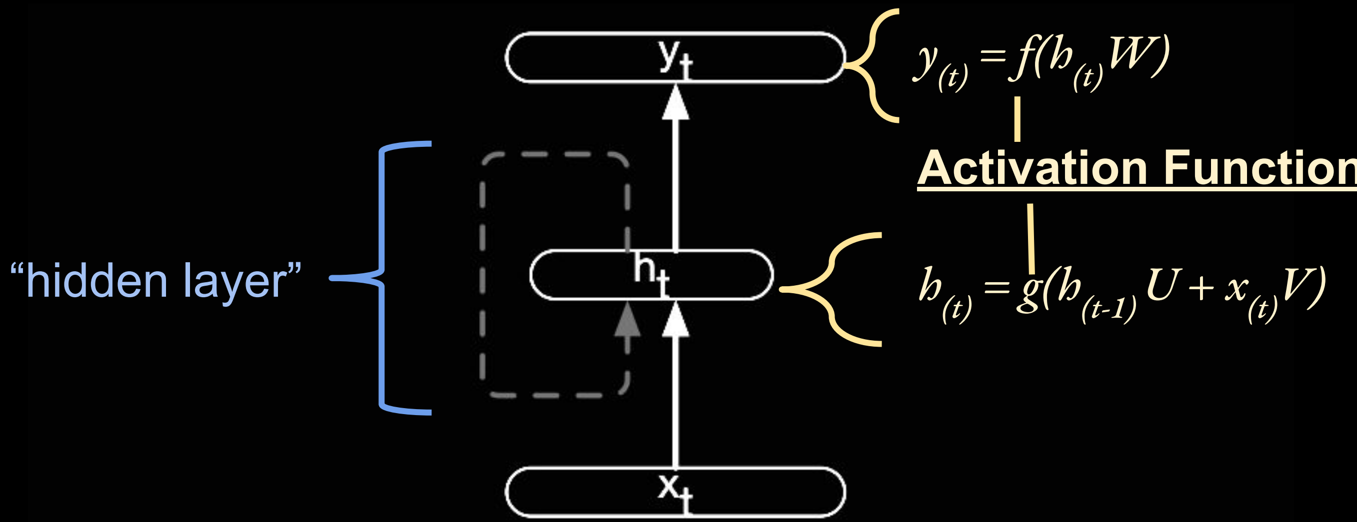


Figure 9.2 Simple recurrent neural network after Elman (Elman, 1990). The hidden layer includes a recurrent connection as part of its input. That is, the activation value of the hidden layer depends on the current input as well as the activation value of the hidden layer from the previous timestep.

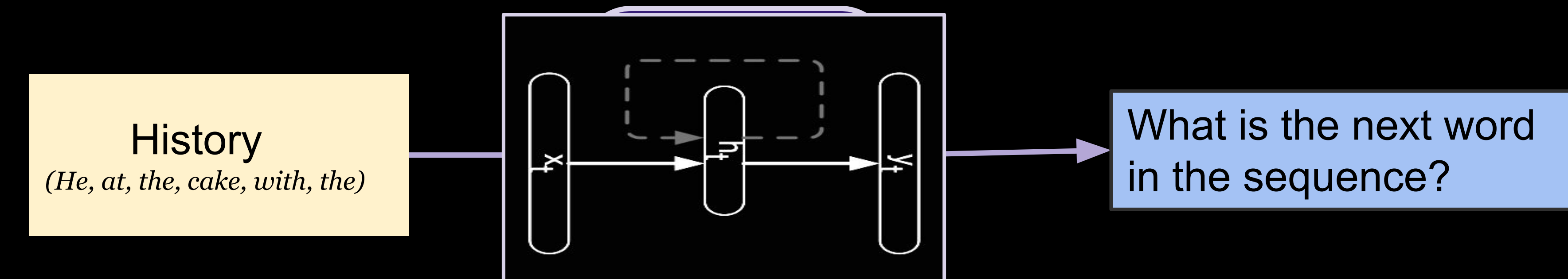
(Jurafsky, 2019)

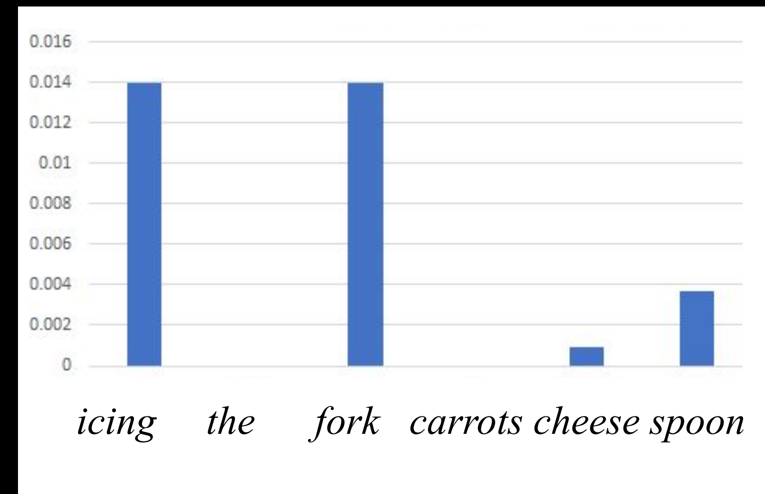
Language Modeling

Task: Estimate $P(w_n \mid w_1, w_2, \ldots, w_{n-1})$

:probability of a next word given history

P(fork | He ate the cake with the) = ?





Language Modeling

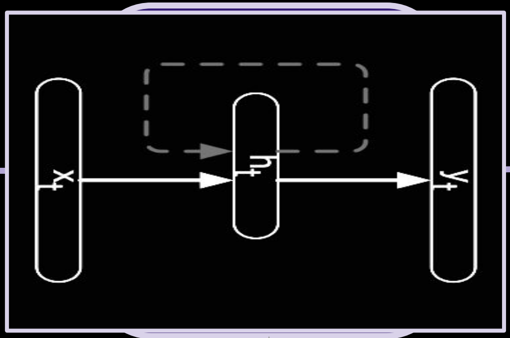
Task: Estimate $P(w_n \mid w_1, w_2, \ldots, w_{n-1})$

:probability of a next word given history

P(fork | He ate the cake with the) = ?

~~History~~Last word

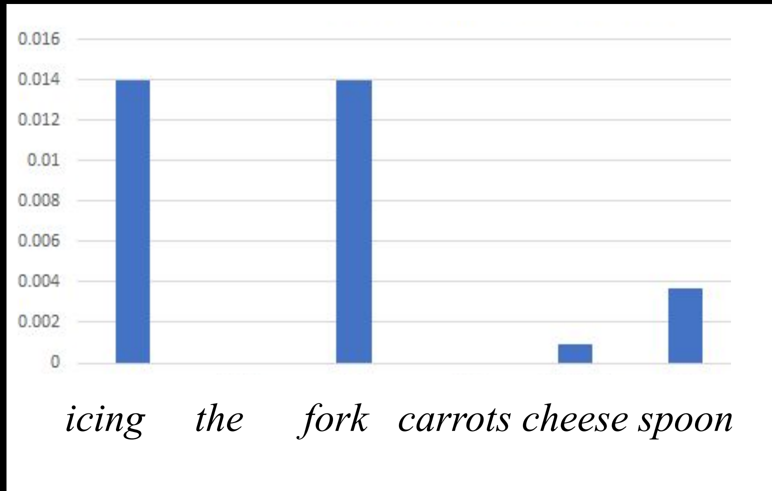
(~~He, at, the, cake, with,~~ the)



What is the next word in the sequence?

Training Corpus

training (fit, learn)



Language Modeling

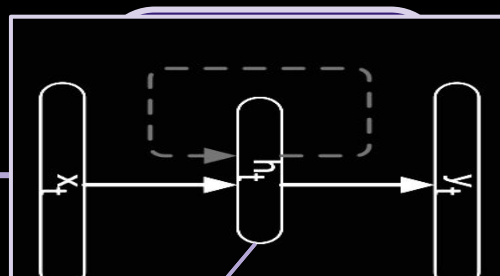
Task: Estimate $P(w_n \mid w_1, w_2, \ldots, w_{n-1})$

:probability of a next word given history

P(fork | He ate the cake with the) = ?

Last word

(the)



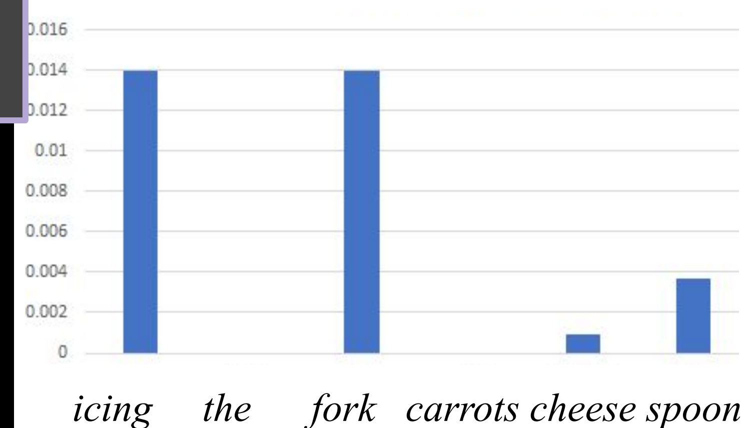
What is the next word in the sequence?

h_t: a vector that we hope "stores" relevant history from previous inputs:

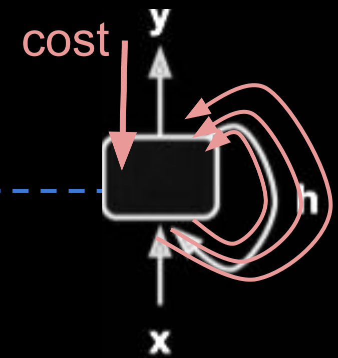
He, at, the, cake, with,

Training Corpus

training (fit, learn)



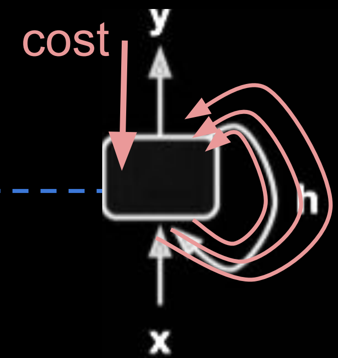
Optimization:



Backward Propagation

```
...
#define forward pass graph:
h(0) = 0
for i in range(1, len(x)):
    h(i) = tf.tanh(tf.matmul(U,h(i-1))+ tf.matmul(W,x(i))) #update hidden state
    y(i) = tf.softmax(tf.matmul(V, h(i))) #update output
...
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(y_pred))
```

Optimization:



Backward Propagation

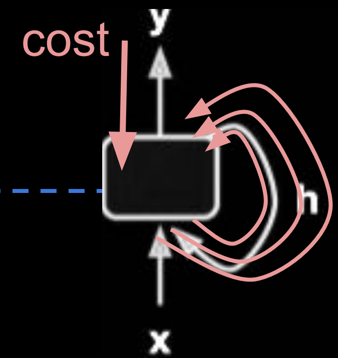
```
...
#define forward pass graph:
h(0) = 0
for i in range(1, len(x)):
    h(i) = tf.tanh(tf.matmul(U,
state
    y(i) = tf.softmax(tf.matmul
...
cost = tf.reduce_mean(-tf.redu
```

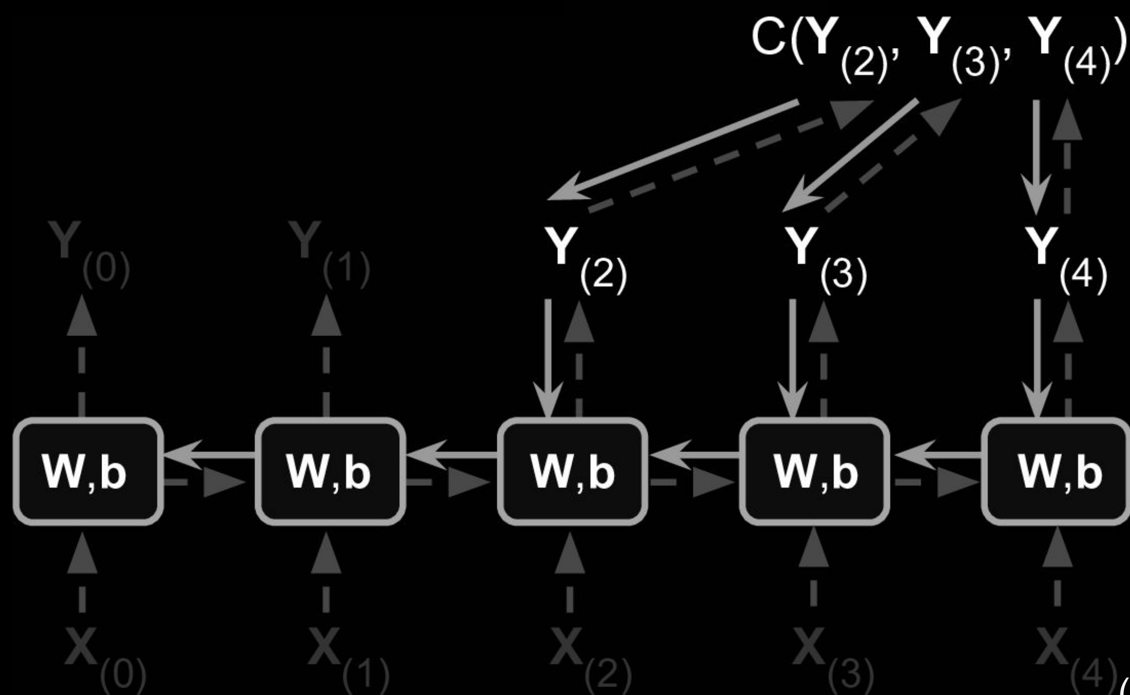
To find the gradient for the overall graph, we use **back propogation,** which *essentially* chains together the gradients for each node (function) in the graph.

With many recursions, the gradients can vanish or explode (become too large or small for floating point operations).

Optimization:

Backward Propagation





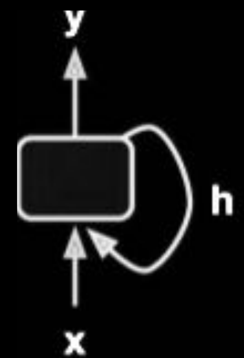
(Geron, 2017)

How to address exploding and vanishing gradients?

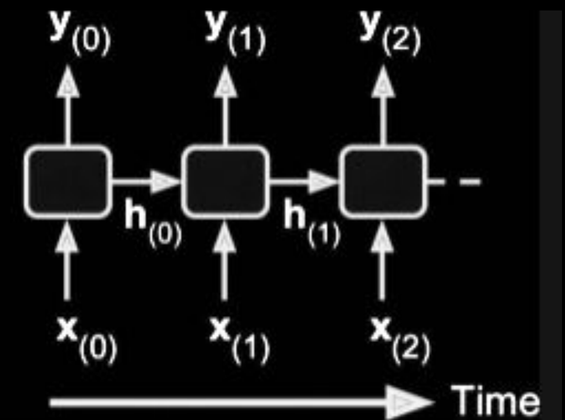
Ad Hoc approaches: e.g. stop backprop iterations very early. “clip” gradients when too high.

How to address exploding and vanishing gradients?

Dominant approach: Use Long Short Term Memory Networks (LSTM)



RNN model

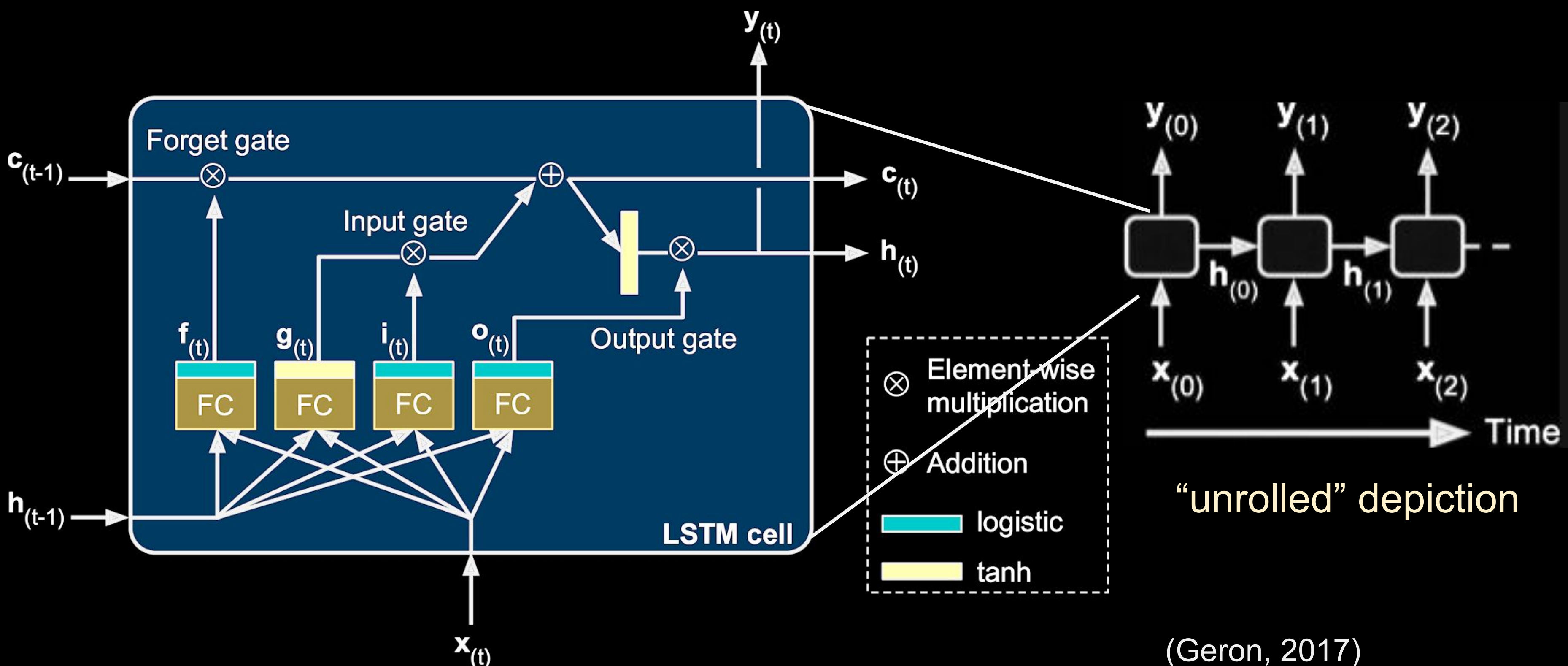


"unrolled" depiction

(Geron, 2017)

How to address exploding and vanishing gradients?

The LSTM Cell



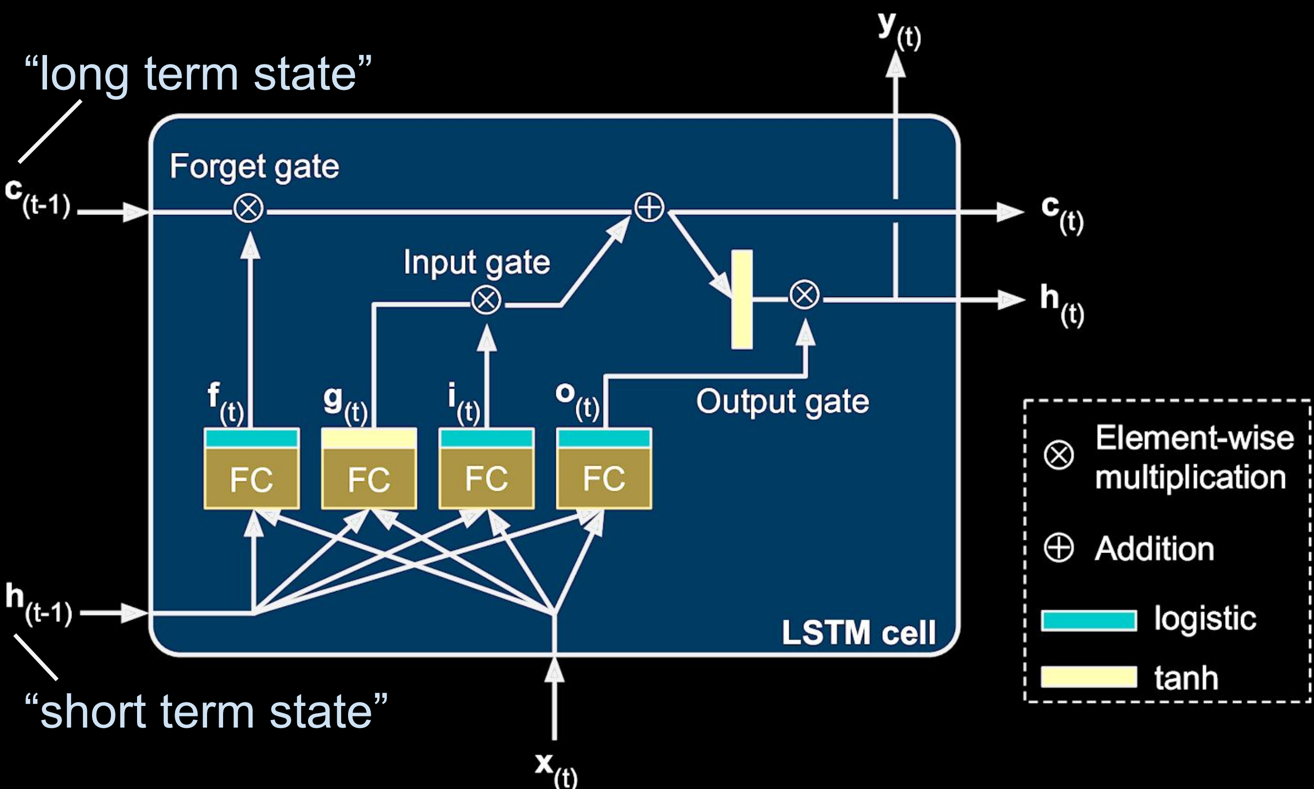
“unrolled” depiction

(Geron, 2017)

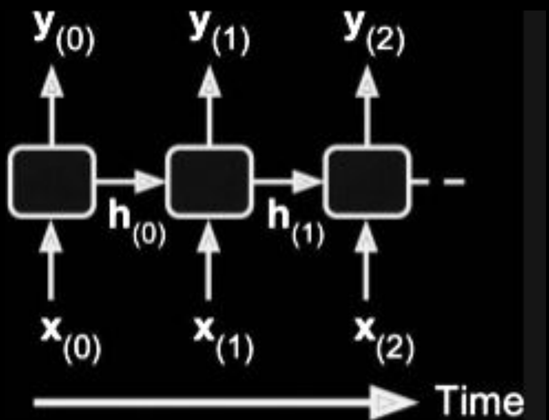
How to address exploding and vanishing gradients?

The LSTM Cell

“long term state”



“short term state”



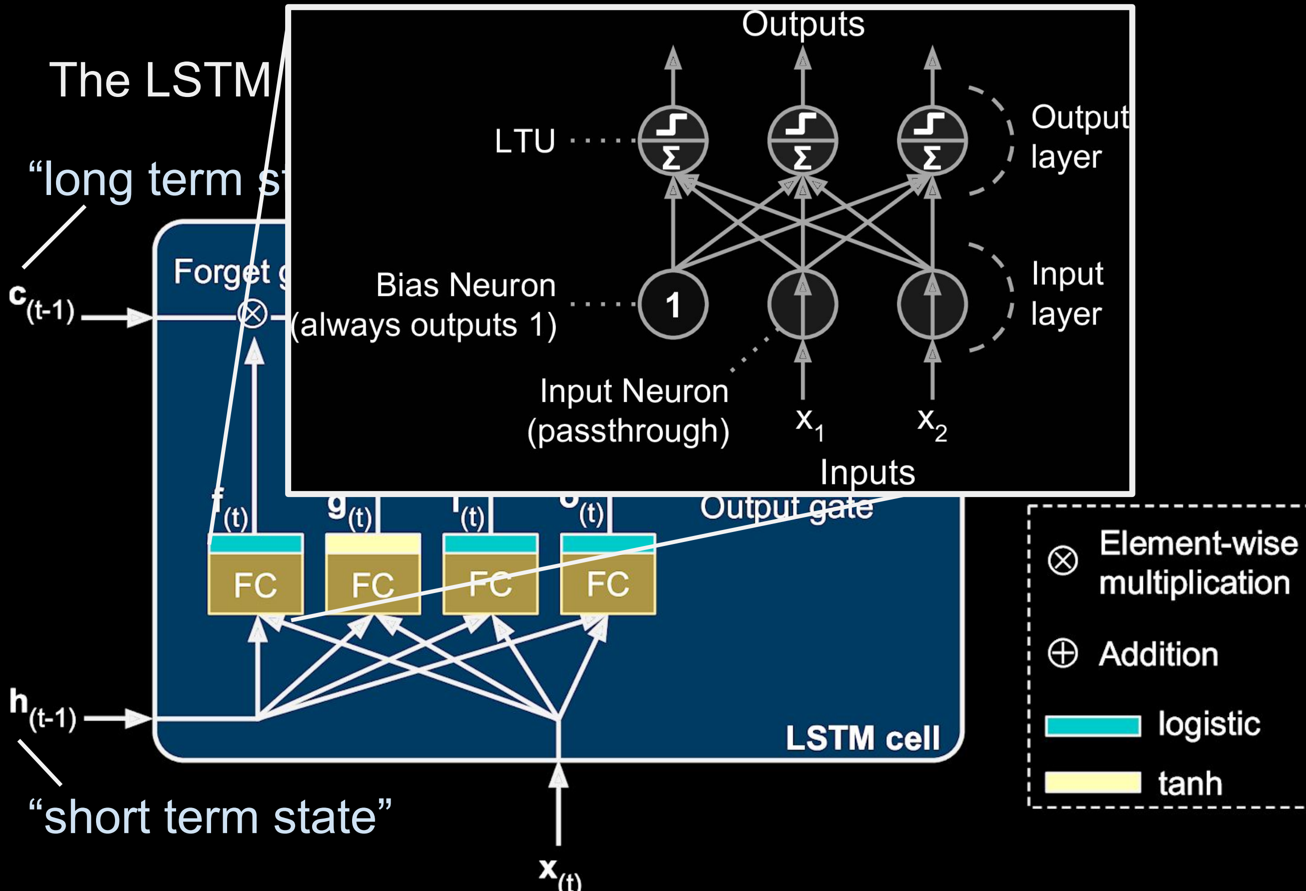
“unrolled” depiction

(Geron, 2017)

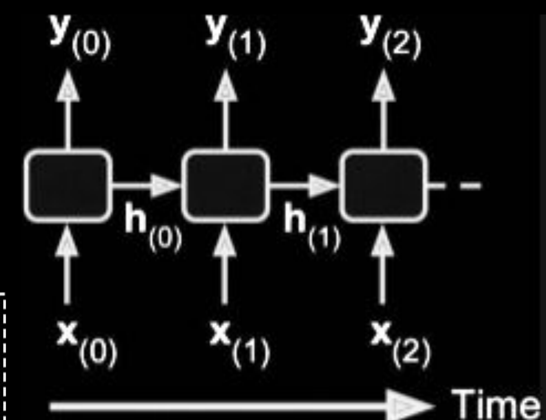
How to address exploding and vanishing gradients?

The LSTM

“long term s



“short term state”

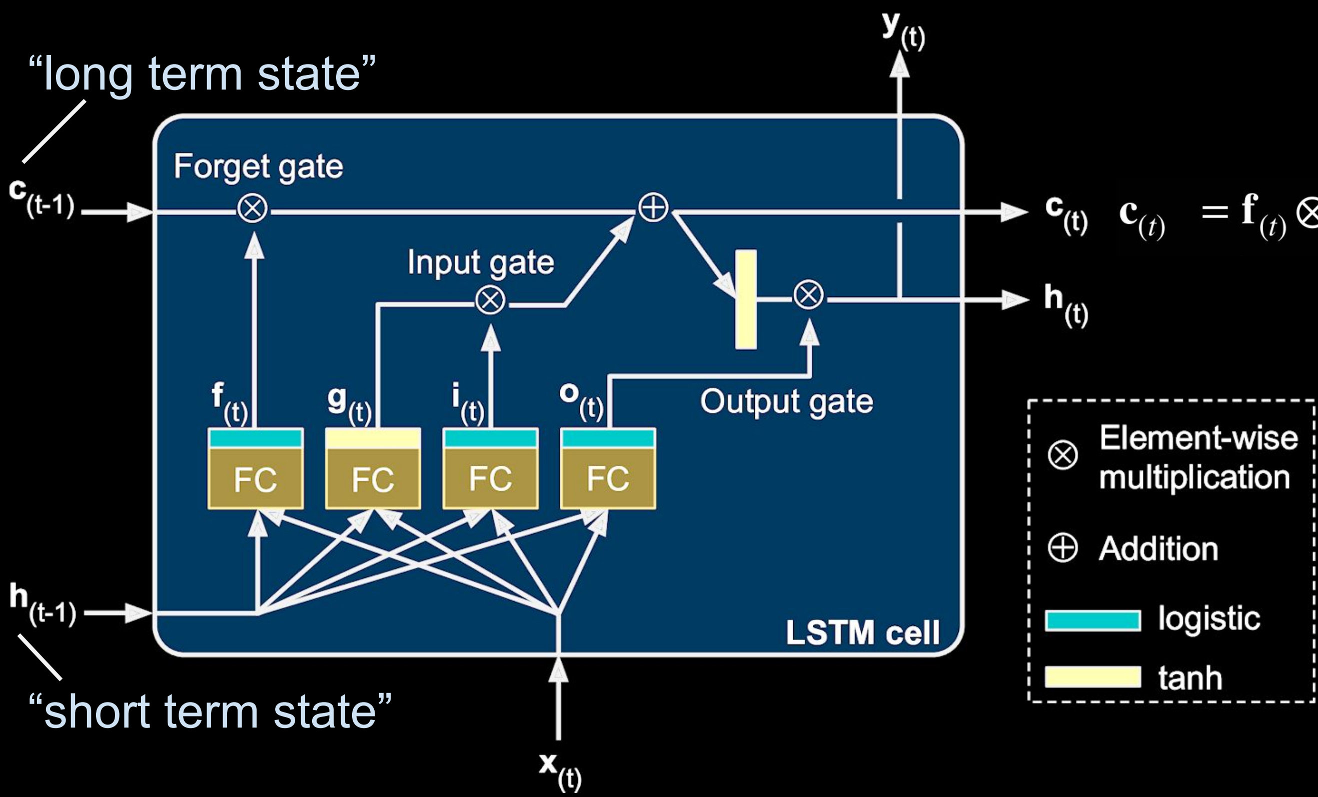


“unrolled” depiction

(Geron, 2017)

How to address exploding and vanishing gradients?

The LSTM Cell

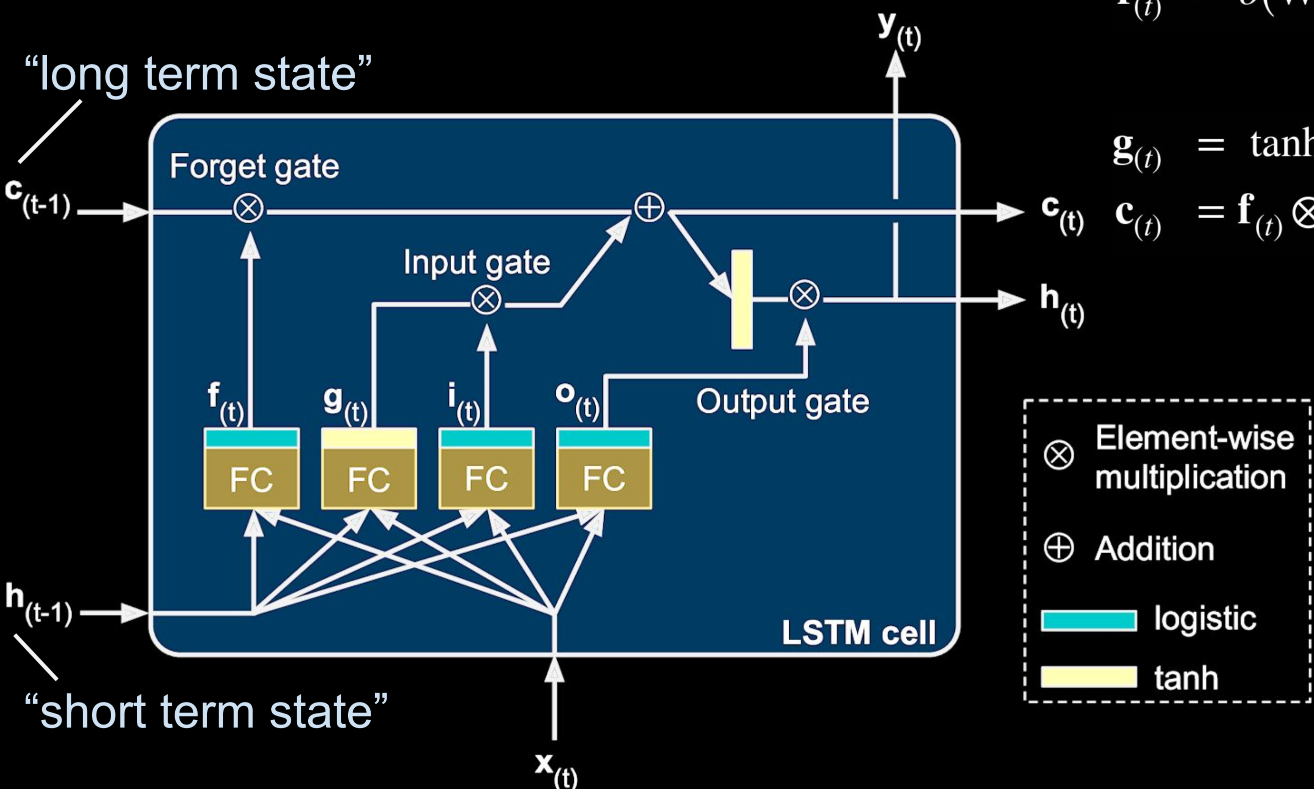


$$\mathbf{c}_{(t)} = \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)}$$

How to address exploding and vanishing gradients?

The LSTM Cell

“long term state”



“short term state”

$$\mathbf{i}_{(t)} = \sigma(\mathbf{W}_{xi}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{i})$$

$$\mathbf{f}_{(t)} = \sigma(\mathbf{W}_{xf}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{f})$$

$$\mathbf{g}_{(t)} = \tanh(\mathbf{W}_{xg}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{g})$$

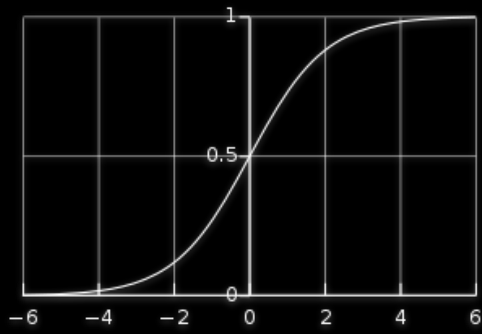
$$\mathbf{c}_{(t)} = \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)}$$

bias term

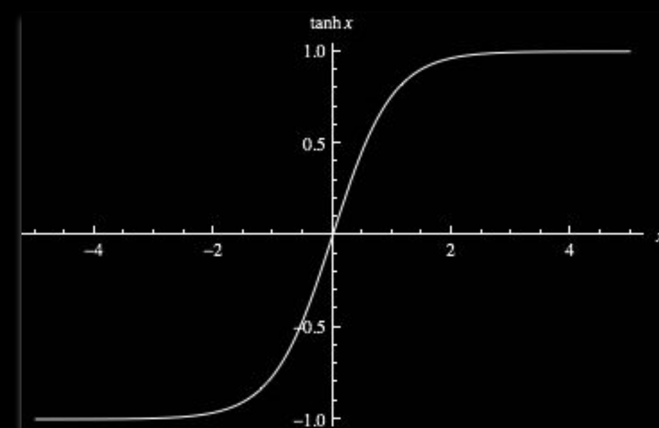
Common Activation Functions

$z = h_{(t)}W$

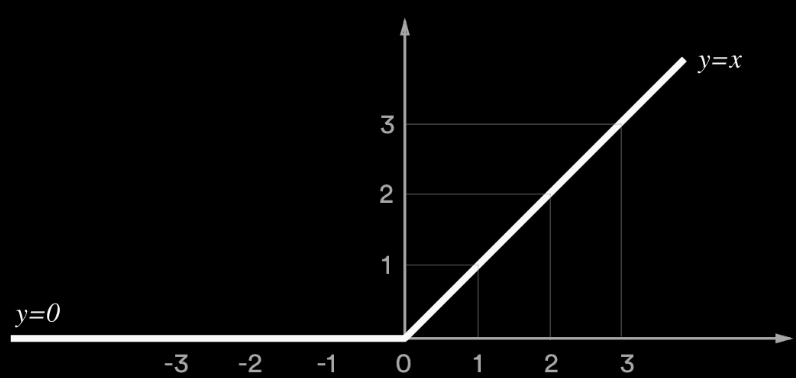
Logistic: $\sigma(z) = 1 / (1 + e^{-z})$



Hyperbolic tangent: $tanh(z) = 2\sigma(2z) - 1 = (e^{2z} - 1) / (e^{2z} + 1)$

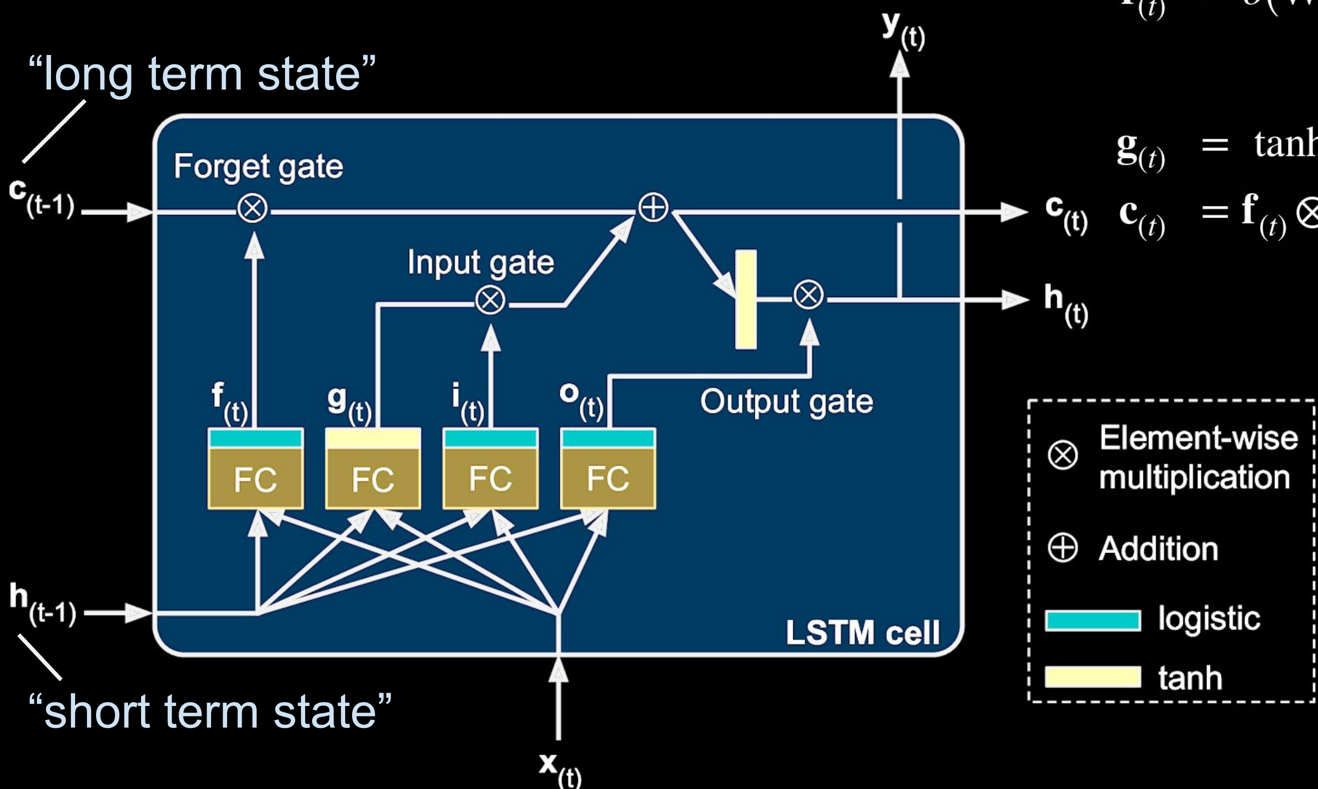


Rectified linear unit (ReLU): $ReLU(z) = \max(0, z)$



LSTM

The LSTM Cell



$$\mathbf{i}_{(t)} = \sigma(\mathbf{W}_{xi}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_i)$$

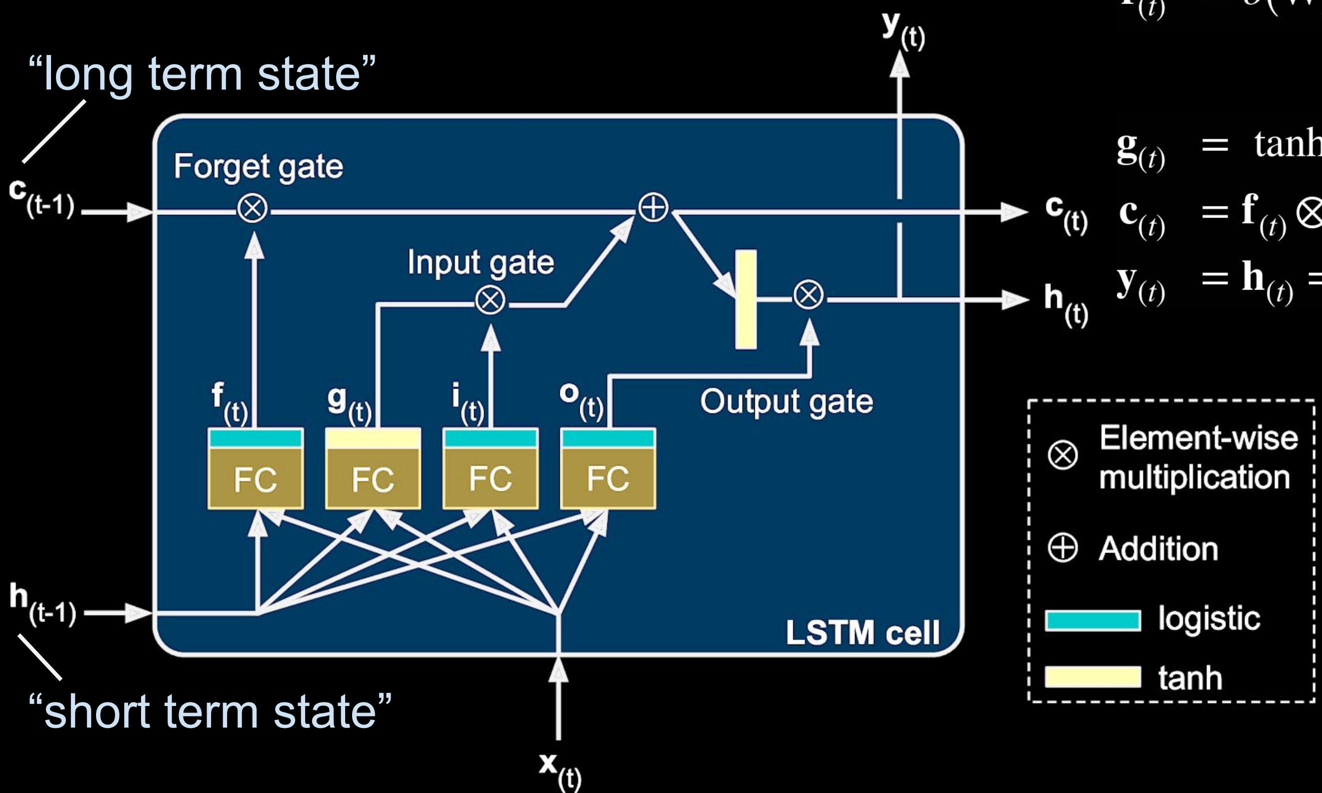
$$\mathbf{f}_{(t)} = \sigma(\mathbf{W}_{xf}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_f)$$

$$\mathbf{g}_{(t)} = \tanh(\mathbf{W}_{xg}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_g)$$

$$\mathbf{c}_{(t)} = \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)}$$

LSTM

The LSTM Cell

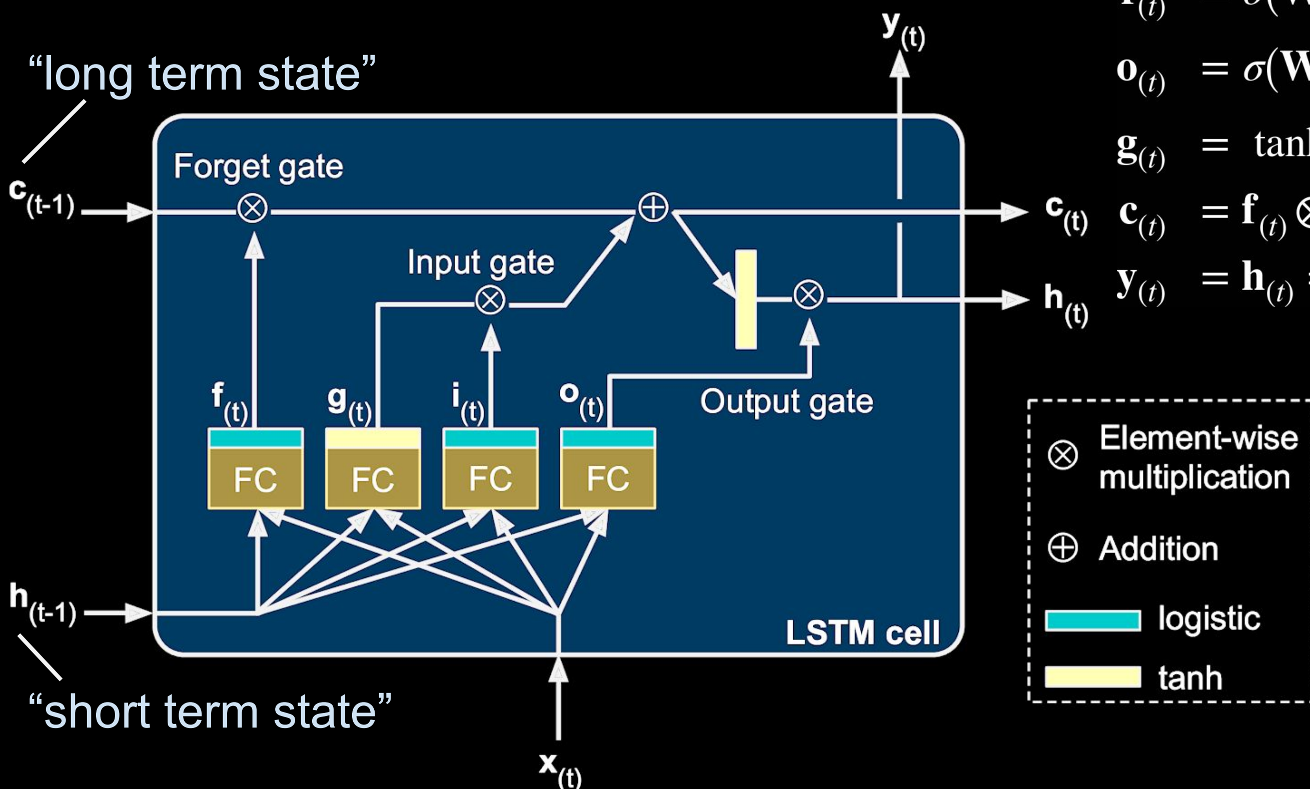


$$
\begin{aligned}
\mathbf{i}_{(t)} &= \sigma(\mathbf{W}_{xi}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_i) \\
\mathbf{f}_{(t)} &= \sigma(\mathbf{W}_{xf}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_f)
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{g}_{(t)} &= \tanh\,(\mathbf{W}_{xg}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_g) \\
\mathbf{c}_{(t)} &= \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)} \\
\mathbf{y}_{(t)} &= \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh\,(\mathbf{c}_{(t)})
\end{aligned}
$$

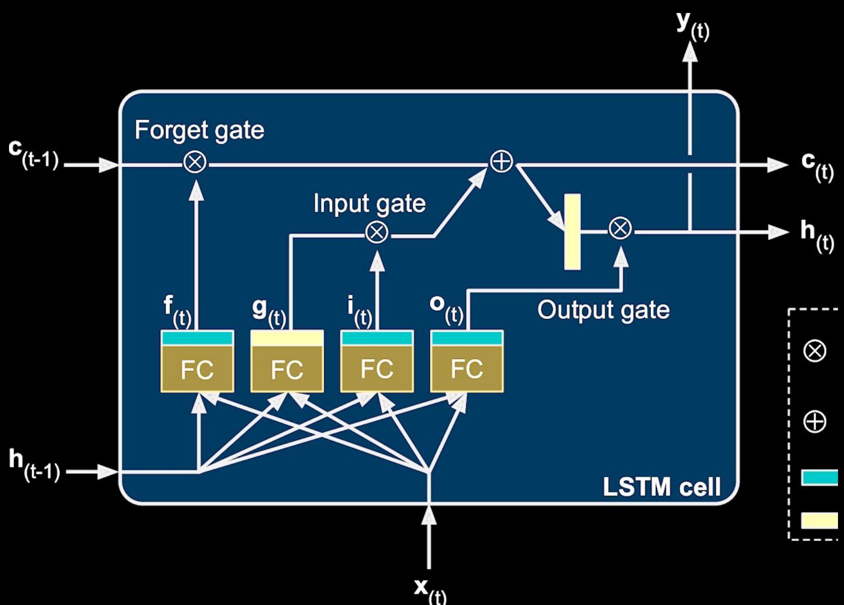
LSTM

The LSTM Cell



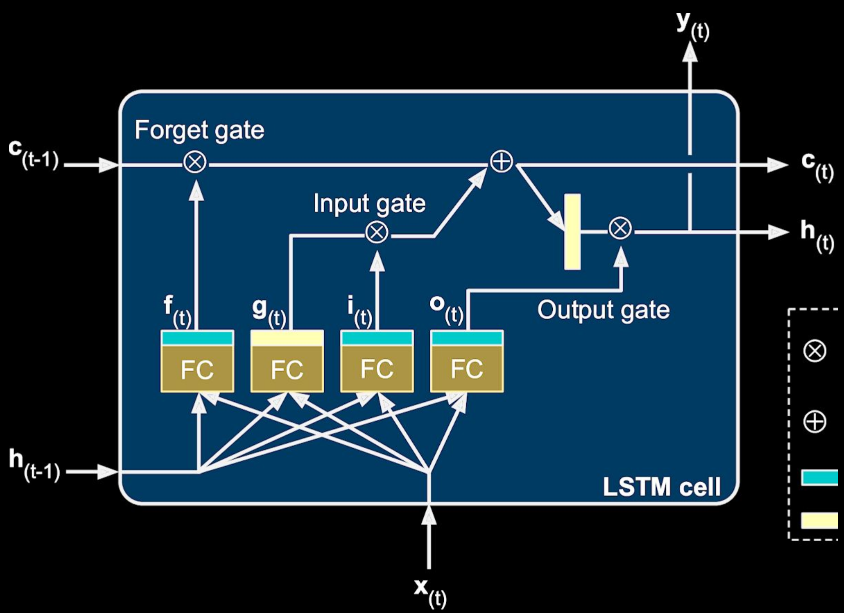
$$
\begin{aligned}
\mathbf{i}_{(t)} &= \sigma(\mathbf{W}_{xi}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{i}) \\
\mathbf{f}_{(t)} &= \sigma(\mathbf{W}_{xf}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{f}) \\
\mathbf{o}_{(t)} &= \sigma(\mathbf{W}_{xo}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{ho}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{o}) \\
\mathbf{g}_{(t)} &= \tanh\,(\mathbf{W}_{xg}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_{g}) \\
\mathbf{c}_{(t)} &= \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)} \\
\mathbf{y}_{(t)} &= \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh\,(\mathbf{c}_{(t)})
\end{aligned}
$$

Input to LSTM



?

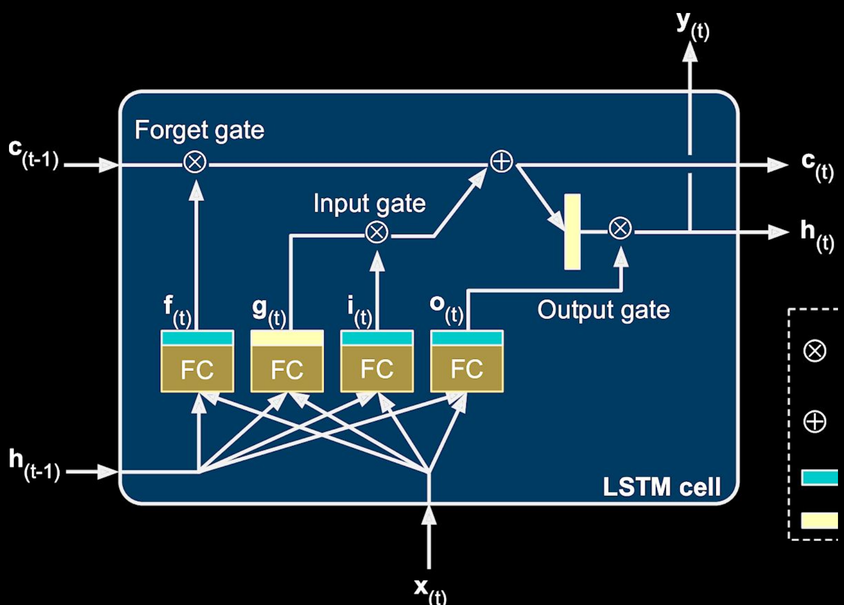
Input to LSTM



?

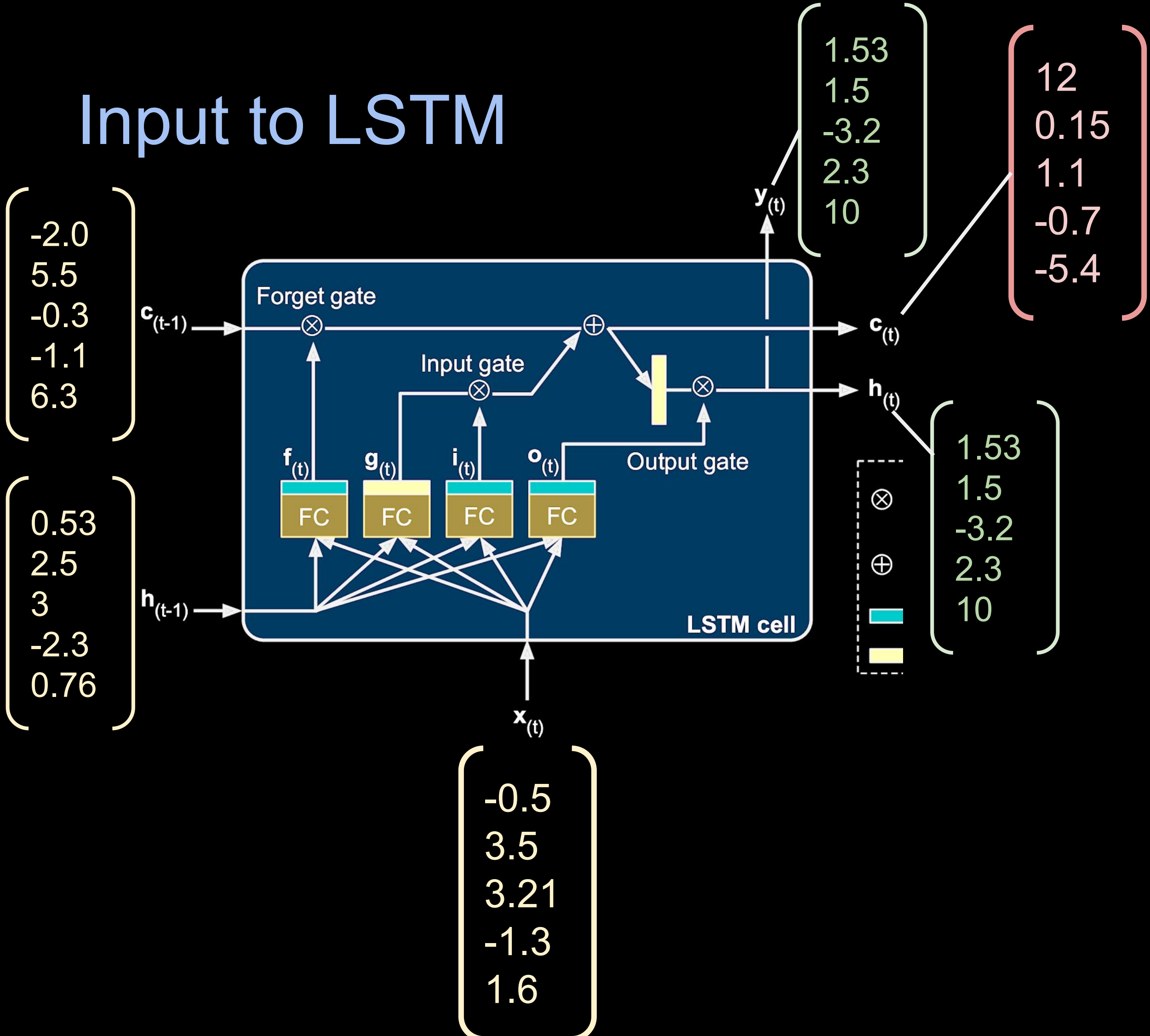
- One-hot encoding?
- Word Embedding

Input to LSTM

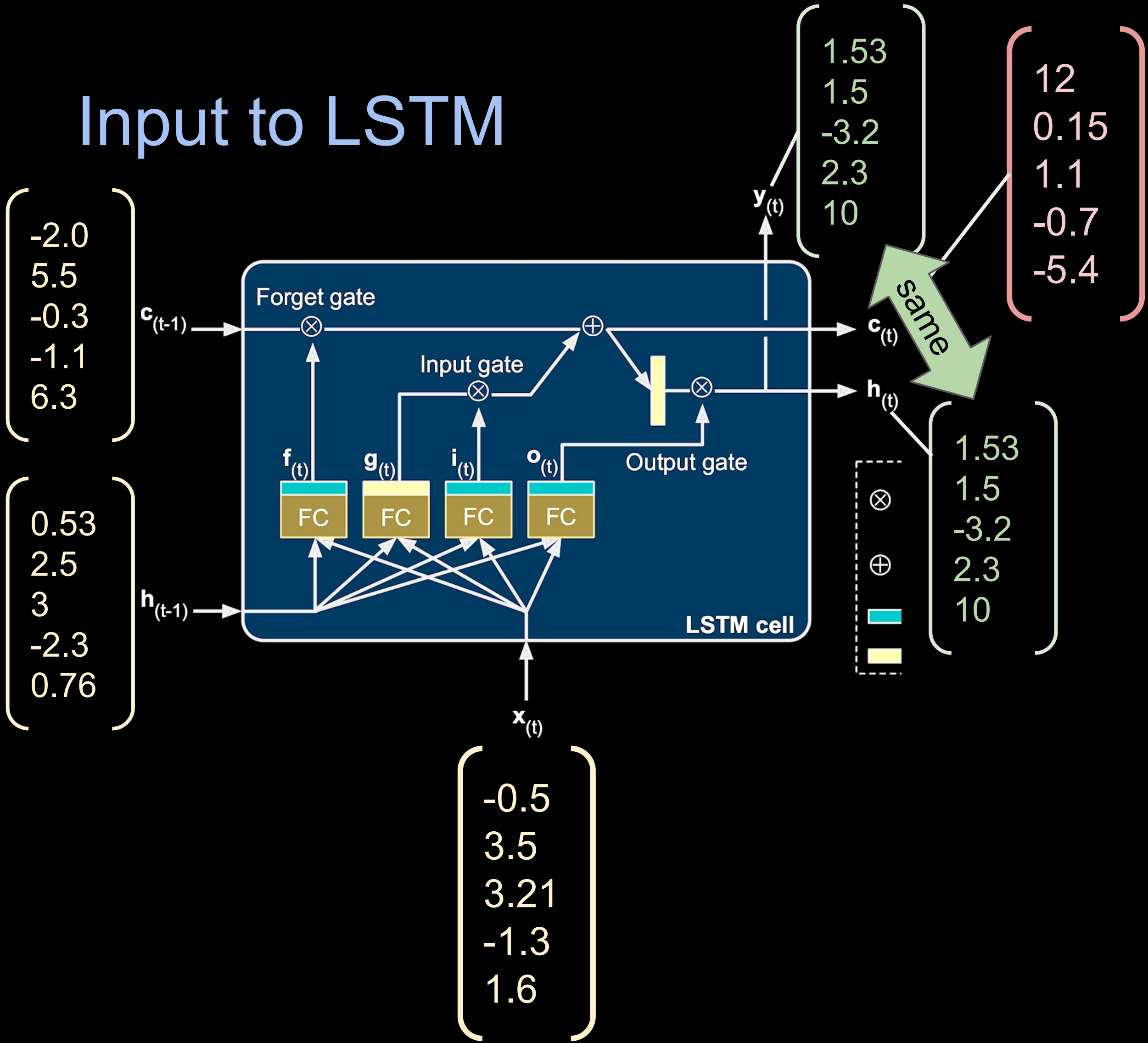


$$\begin{bmatrix} -0.5 \\ 3.5 \\ 3.21 \\ -1.3 \\ 1.6 \end{bmatrix}$$

Input to LSTM

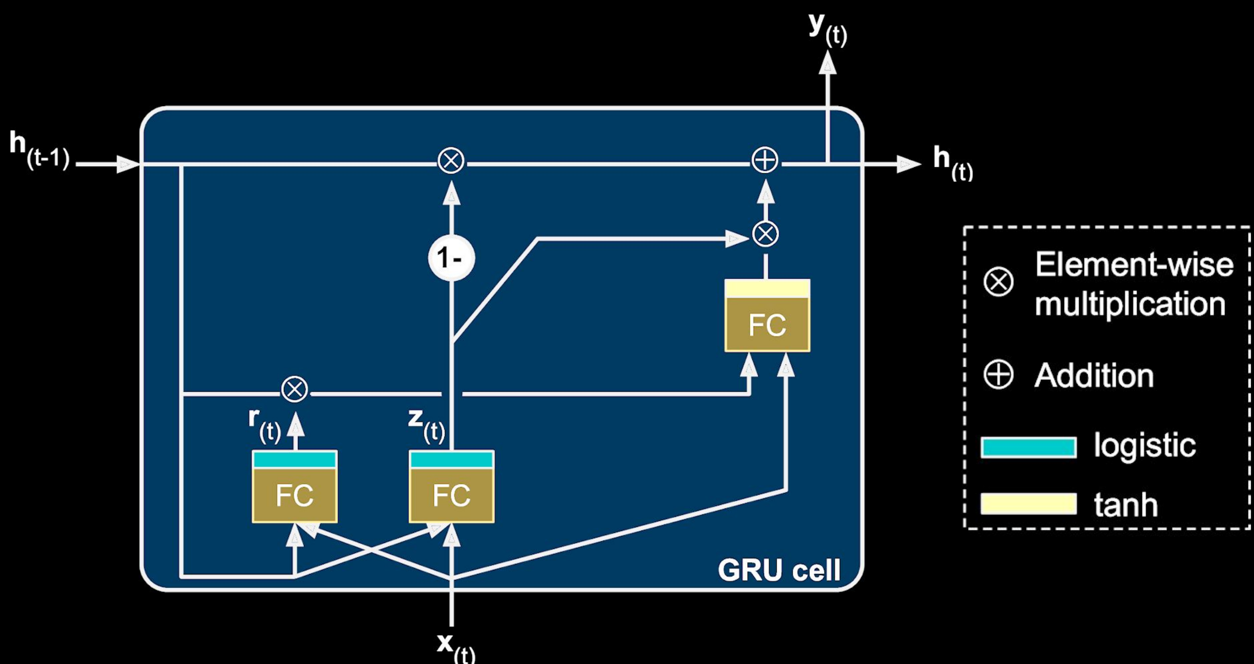


Input to LSTM



The GRU

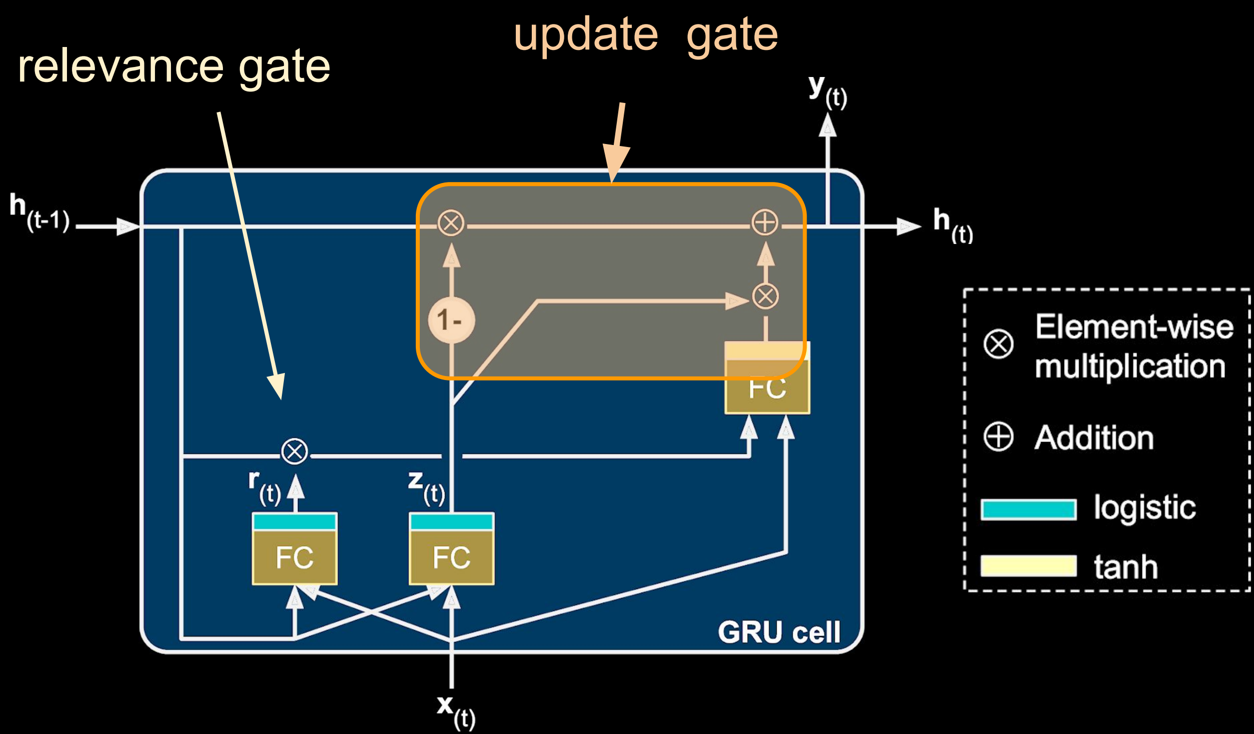
Gated Recurrent Unit



(Geron, 2017)

The GRU

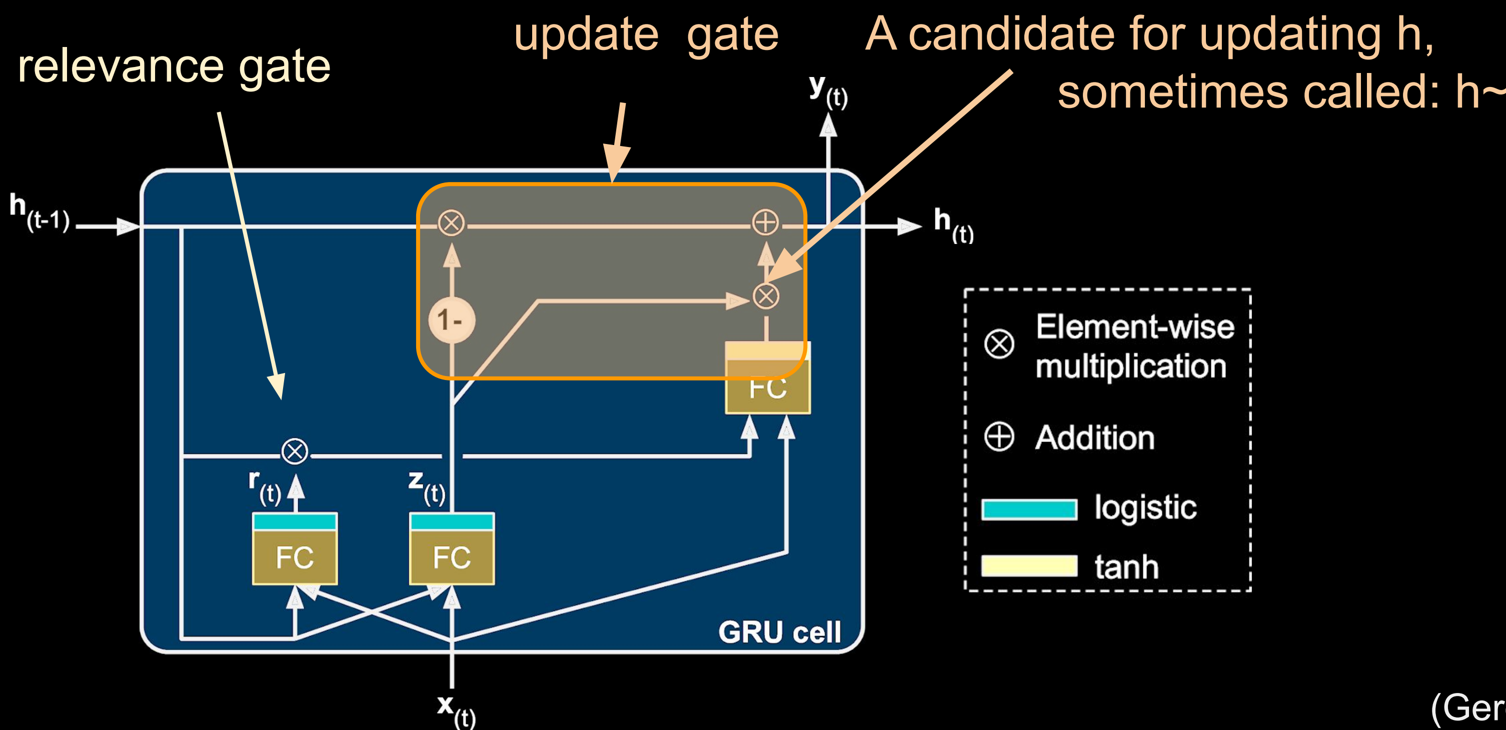
Gated Recurrent Unit



(Geron, 2017)

The GRU

Gated Recurrent Unit

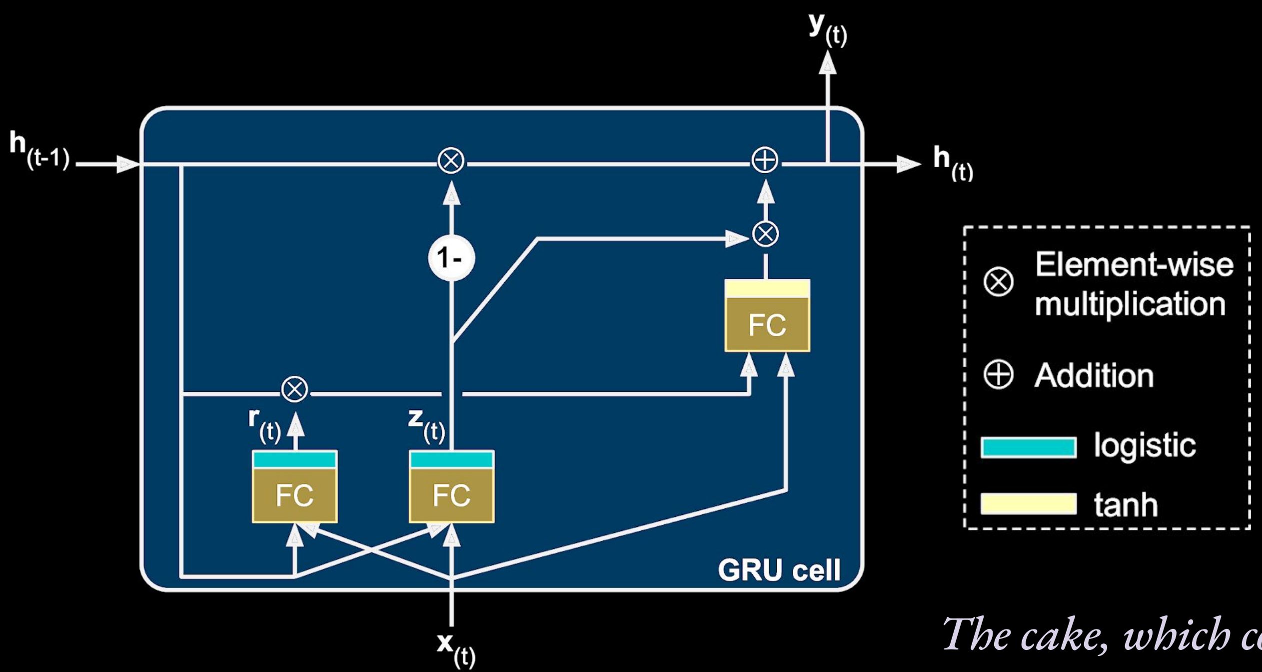


(Geron, 2017)

The GRU

Gated Recurrent Unit

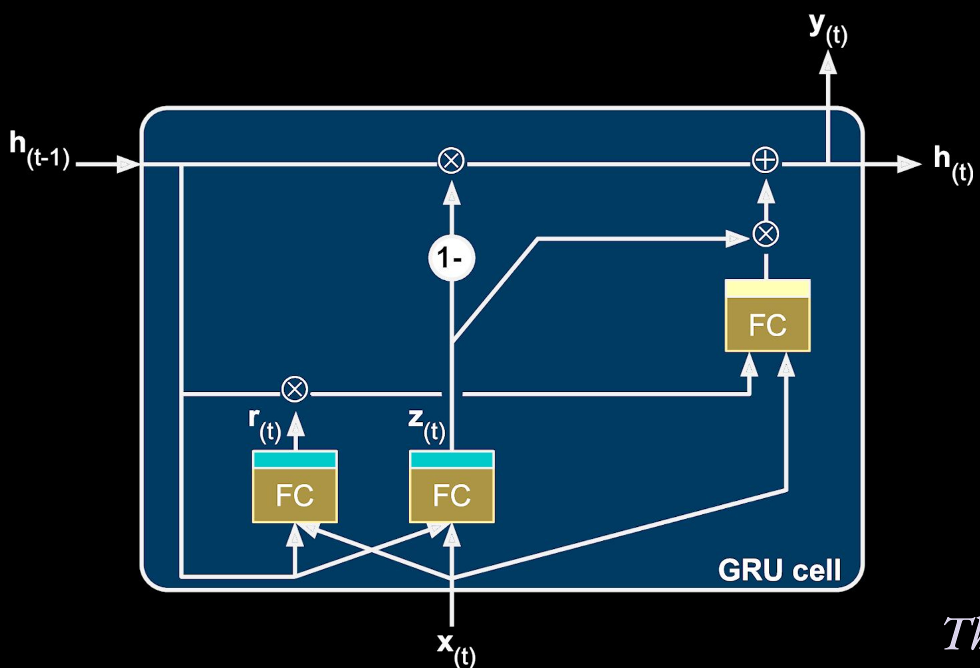
$$\mathbf{z}_{(t)} = \sigma(\mathbf{W}_{xz}{}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hz}{}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_z)$$
$$\mathbf{r}_{(t)} = \sigma(\mathbf{W}_{xr}{}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hr}{}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_r)$$
$$\mathbf{g}_{(t)} = \tanh(\mathbf{W}_{xg}{}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^T \cdot (\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)}) + \mathbf{b}_g)$$
$$\mathbf{h}_{(t)} = \mathbf{z}_{(t)} \otimes \mathbf{h}_{(t-1)} + (1 - \mathbf{z}_{(t)}) \otimes \mathbf{g}_{(t)}$$



The cake, which contained candles, was eaten.

What about the gradient?

$$\mathbf{z}_{(t)} = \sigma(\mathbf{W}_{xz}{}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hz}{}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_z)$$
$$\mathbf{r}_{(t)} = \sigma(\mathbf{W}_{xr}{}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hr}{}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_r)$$
$$\mathbf{g}_{(t)} = \tanh\left(\mathbf{W}_{xg}{}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^T \cdot (\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)}) + \mathbf{b}_g\right)$$
$$\mathbf{h}_{(t)} = \mathbf{z}_{(t)} \otimes \mathbf{h}_{(t-1)} + (1 - \mathbf{z}_{(t)}) \otimes \mathbf{g}_{(t)}$$



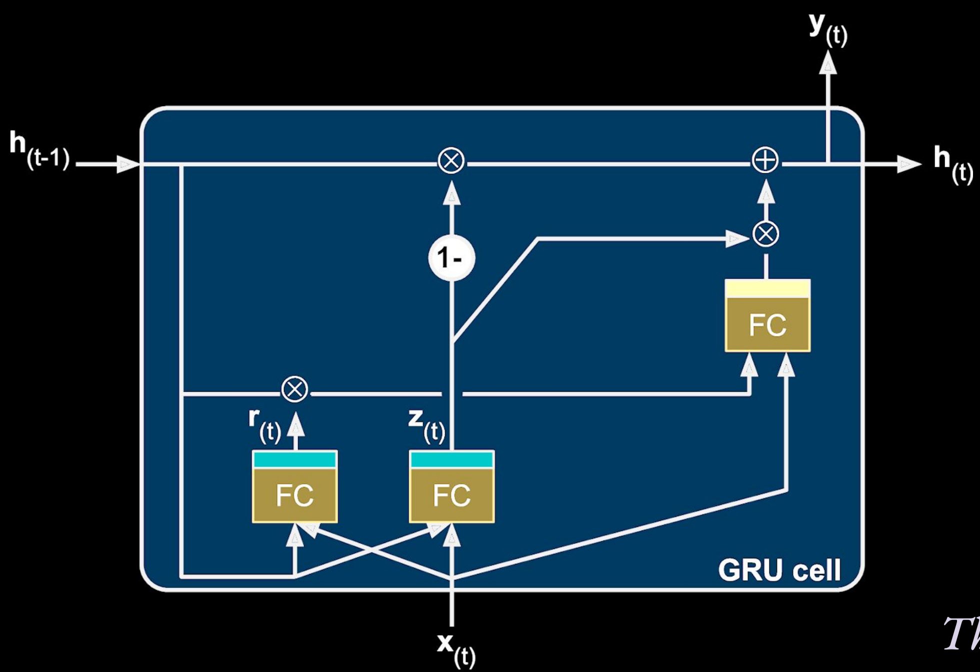
The gates (i.e. multiplications based on a logistic) often end up keeping the hidden state exactly (or nearly exactly) as it was. Thus, for most dimensions of h,

$$h_{(t)} \approx h_{(t-1)}$$

The cake, which contained candles, was eaten.

What about the gradient?

$$\mathbf{z}_{(t)} = \sigma(\mathbf{W}_{xz}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hz}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_z)$$
$$\mathbf{r}_{(t)} = \sigma(\mathbf{W}_{xr}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hr}{}^{T} \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_r)$$
$$\mathbf{g}_{(t)} = \tanh\left(\mathbf{W}_{xg}{}^{T} \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}{}^{T} \cdot (\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)}) + \mathbf{b}_g\right)$$
$$\mathbf{h}_{(t)} = \mathbf{z}_{(t)} \otimes \mathbf{h}_{(t-1)} + (1 - \mathbf{z}_{(t)}) \otimes \mathbf{g}_{(t)}$$



The gates (i.e. multiplications based on a logistic) often end up keeping the hidden state exactly (or nearly exactly) as it was. Thus, for most dimensions of h,

$$h_{(t)} \approx h_{(t-1)}$$

This tends to keep the gradient from vanishing since the same values will be present through multiple times in backpropagation through time. (The same idea applies to LSTMs but is easier to see here).

The cake, which contained candles, was eaten.

How to train an LSTM-style RNN

```
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(y_pred))
```

Cost Function: $J_{(t)} = -\sum_{j=1}^{|V|} y_{(t),j} log \ y_{(\hat{t}),j}$ -- ”cross entropy error”

How to train an LSTM-style RNN

```
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(y_pred))
```

Cost Function: $J_{(t)} = -\sum_{j=1}^{|V|} y_{(t),j} log\ y_{\hat{(t)},j}$ -- "cross entropy error"

$$J = \sum_{t}^{T} - \frac{\sum_{j=1}^{|V|} y_{(t),j} log\ \hat{y}_{(t),j}}{T}$$

How to train an LSTM-style RNN

```
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(y_pred))
```

Cost Function: $J_{(t)} = -\sum_{j=1}^{|V|} y_{(t),j} log\ y_{(\hat{t}),j}$ -- ”cross entropy error”

Stochastic Gradient Descent -- a method

RNN-Based Language Models

Take-Aways

- Simple RNNs are difficult to train: exploding and vanishing gradients
- LSTM and GRU cells solve
  - Hidden states past from one time-step to the next, allow for long-distance dependencies.
  - Gates are used to keep hidden states from changing rapidly (and thus keeps gradients under control).
  - LSTM and GRU are complex, but simply a series of functions:
    - logit (w⋅x)
    - tanh (w⋅x)
    - element-wise multiplication and addition
  - To train: mini-batch stochastic gradient descent over cross-entropy cost

$$\begin{pmatrix} 0.53 \\ 1.5 \\ 3.21 \\ -2.3 \\ .76 \end{pmatrix}$$