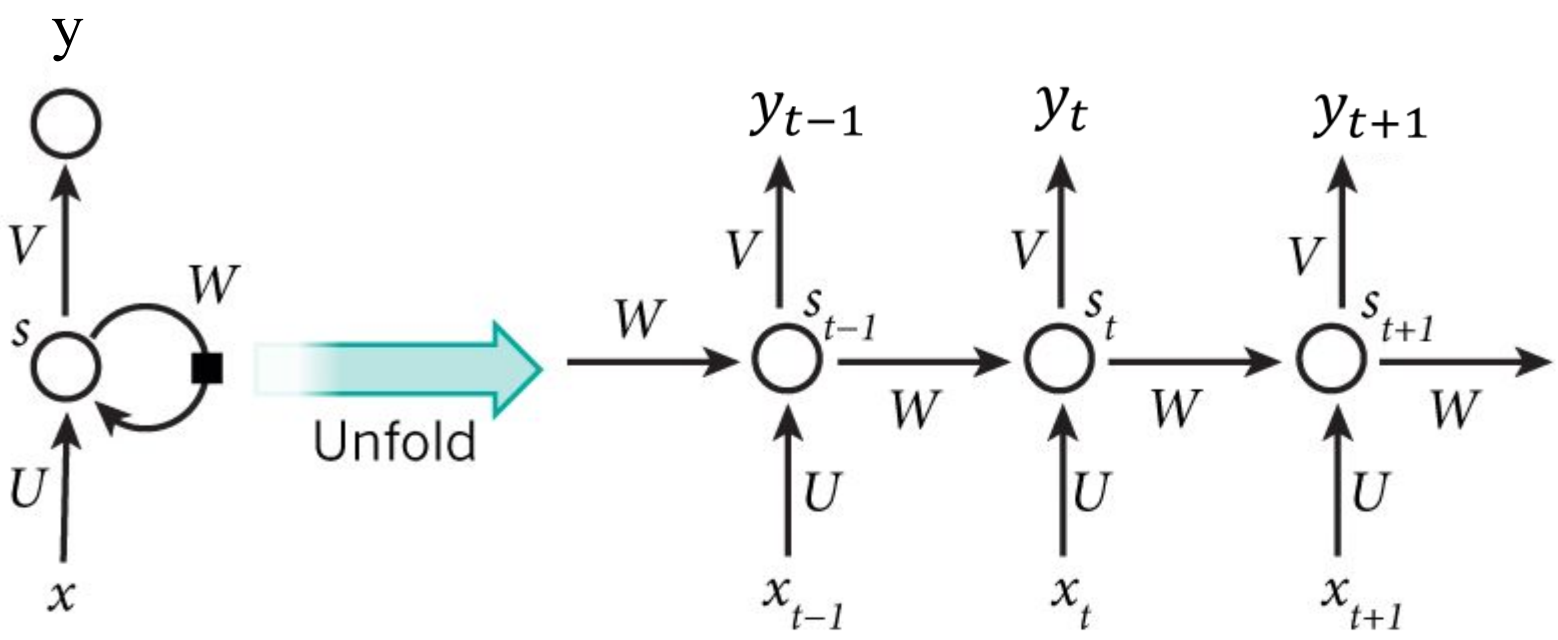
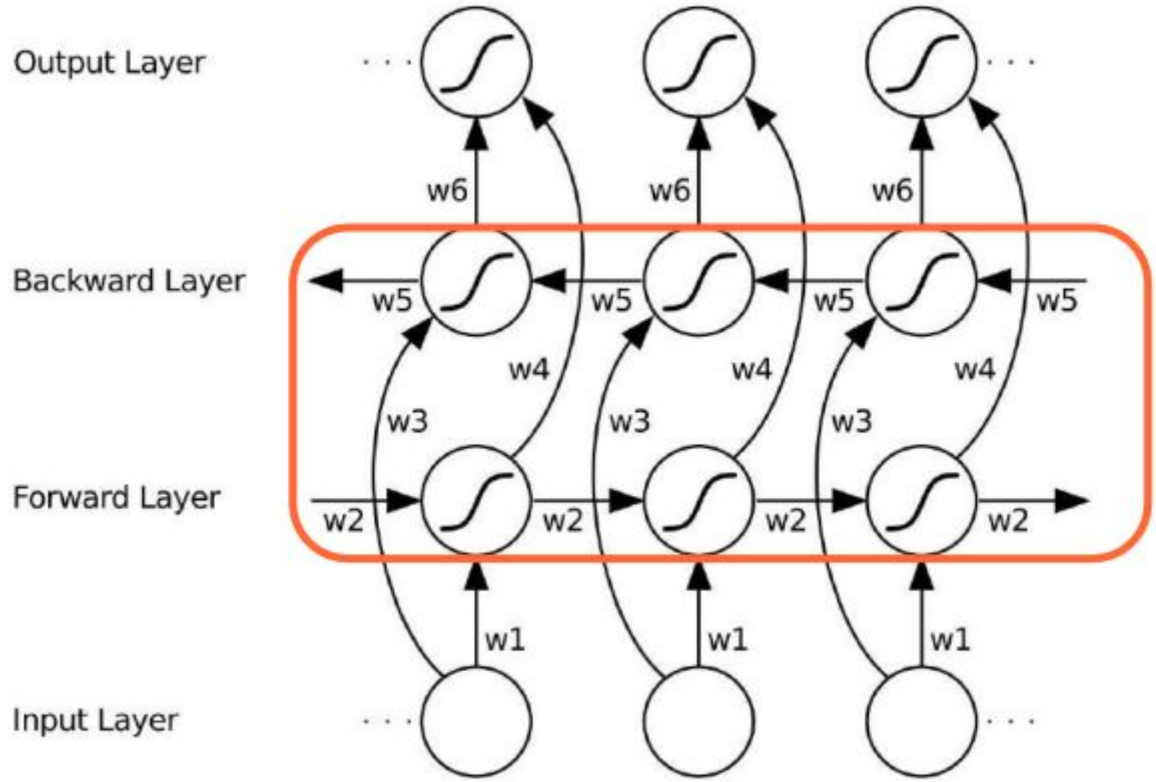
תרגול 8

רשת נוירונים נשנית



Bidirectional RNN

- Bidirectional RNNs are based on the idea that the output at time may not only depend on the previous elements in the sequence, but also future elements.



RNN

- One of the appeals of RNNs is the idea that they might be able to connect previous information to the present task.
  - previous video frames might inform the understanding of the present frame.
  - predict the next word based on the previous ones.
- Sometimes the gap between relevant information pieces is small.

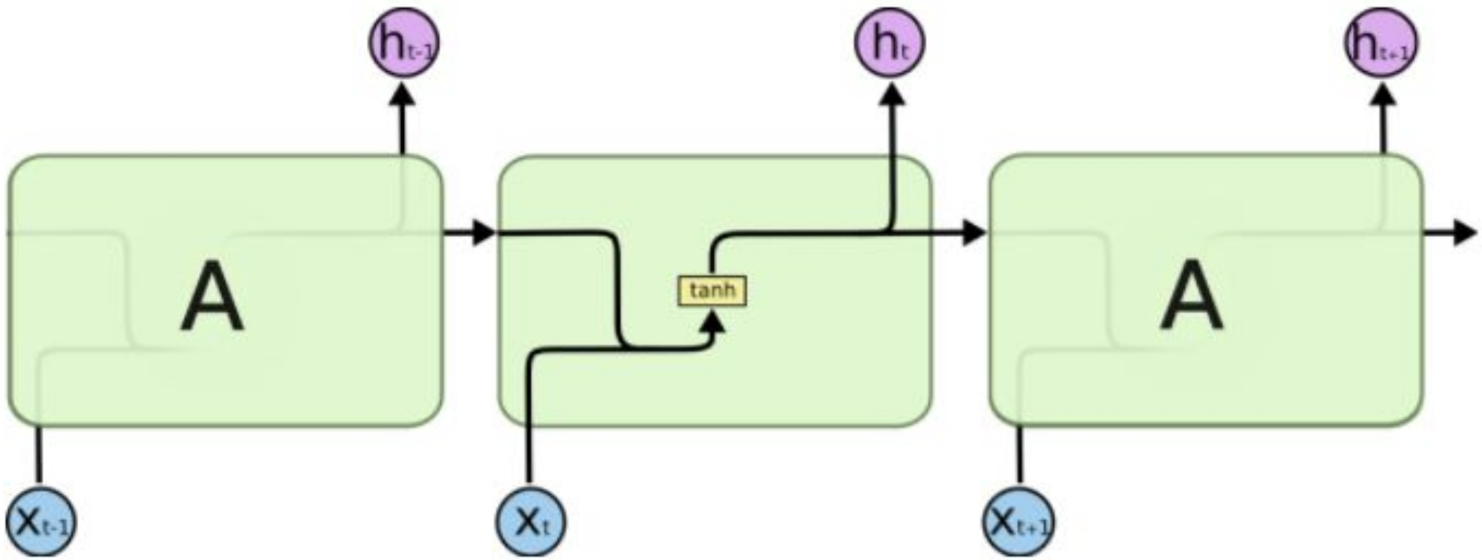
"the clouds are in the sky"

- But there are also cases where the gap is far big.

"I grew up in France… I speak fluent French."

RNN

A different view of a simple RNN with 1 tanh activation layer:

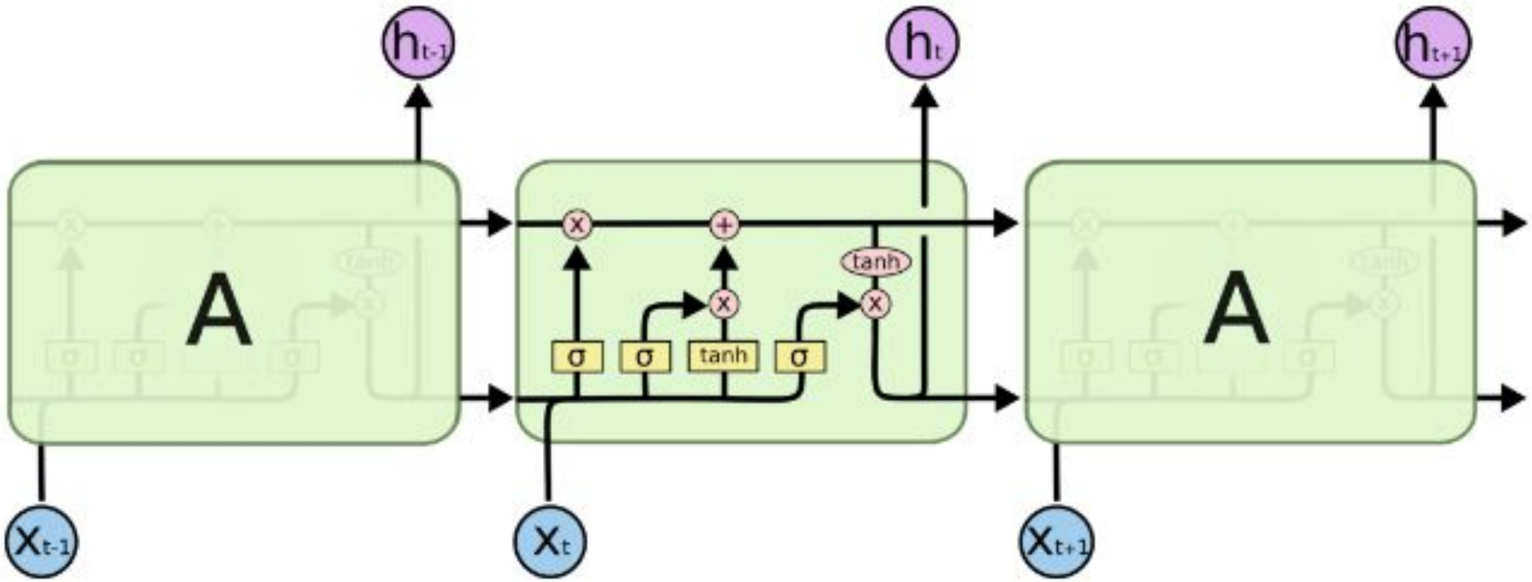


LSTM Networks

- LSTM - Long Short Term Memory
  - are a special kind of RNN, capable of learning long-term dependencies.
  - LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time.
  - The repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way

reference: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

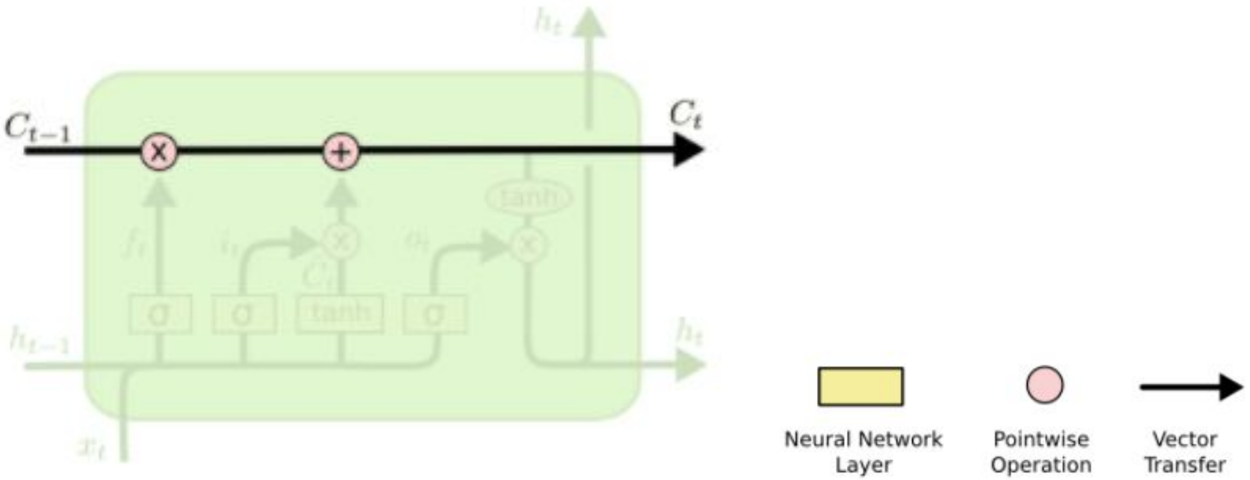
LSTM Networks



$$i_t = \text{sigmoid}(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi})$$
$$f_t = \text{sigmoid}(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf})$$
$$g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hc}h_{(t-1)} + b_{hg})$$
$$o_t = \text{sigmoid}(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho})$$
$$c_t = f_t * c_{(t-1)} + i_t * g_t$$
$$h_t = o_t * \tanh(c_t)$$

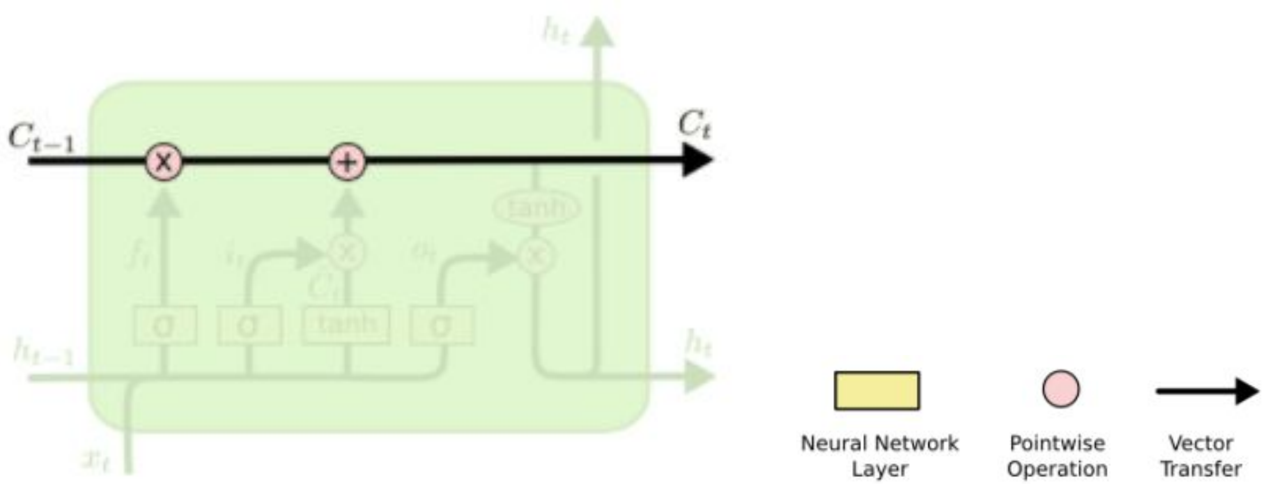
The Core Idea Behind LSTM

- The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.
- The information flows over this line from one time-step to another with only some minor linear interactions.



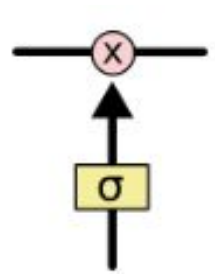
The Core Idea Behind LSTM

- You can think of the cell state as the long-term memory of the cell, that contains only the relevant information for the current and future time-steps.
- The gates have the ability to remove or add information to the cell state.



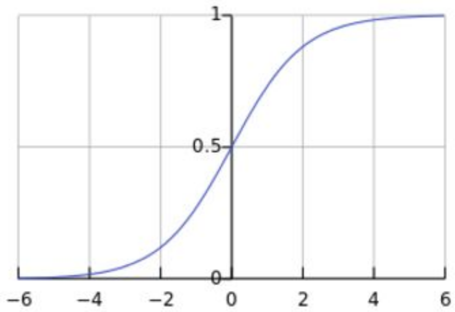
LSTM Gates

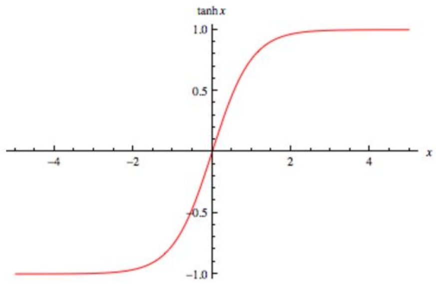
- Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



- The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through.

$$sigmoid(x) = \frac{1}{1+e^{-x}}$$



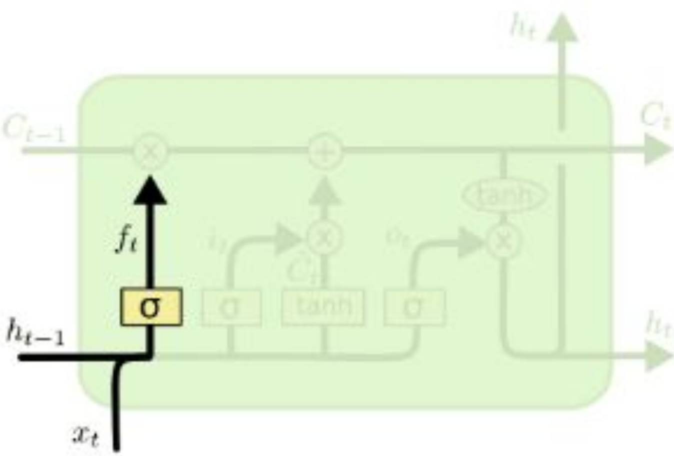


$$thnh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Forget Gate Layer

- Linear layer with a Sigmoid activation that gets as input $\boldsymbol{h_{t-1}}$ and $\boldsymbol{x_t}$, and outputs a number between 0 and 1 for each element in the cell state $\boldsymbol{C_{t-1}}$.

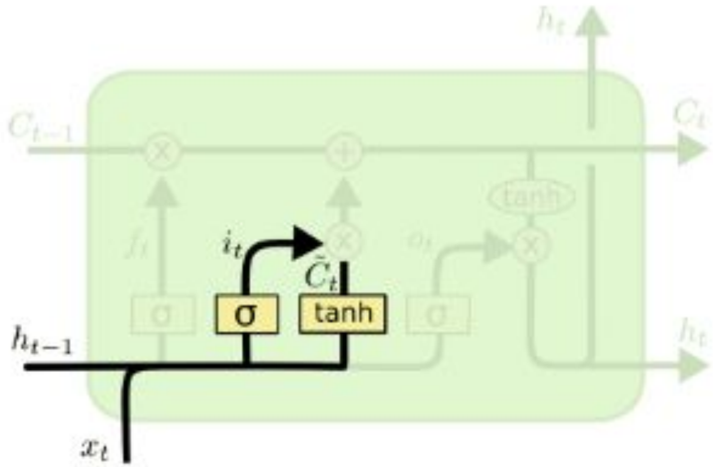
- An output of 1 represents “completely keep this” while a 0 represents “completely get rid of this.”



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

Input Gate Layer

- Decides what new information to add to the cell state.
- A Sigmoid layer decides which values to update.
- A tanh layer creates the values themselves.
- The total update is the multiplication between them.



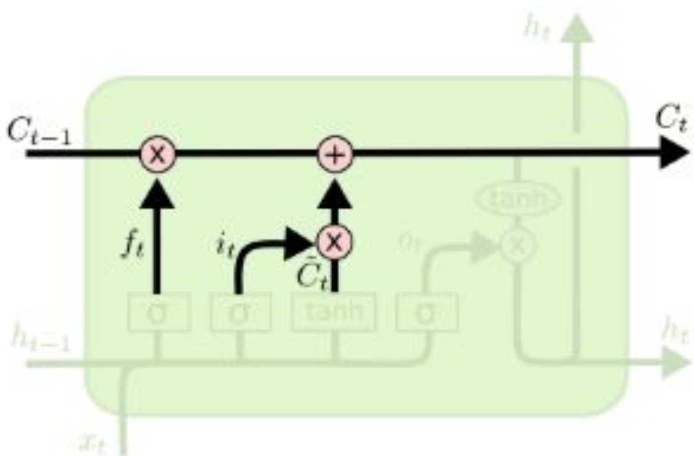
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

Update The Cell State

- multiplies the old state by f_t , forgetting the things that was decided to forget earlier.

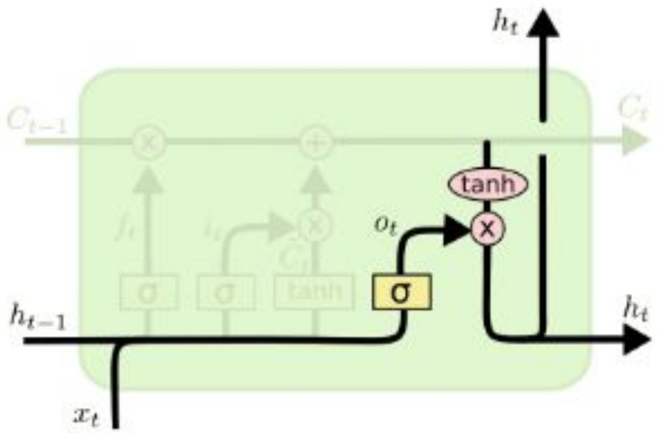
- Then we add $i_t * \tilde{C}$t. This is the new values, scaled by how much we decided to update each state value.



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

Output Gate

- First, we run a sigmoid layer which decides what parts of the cell state we're going to output.
- Then, the cell state is passed through a tanh (to push the values to be between −1 and 1) and multiplied by the output of the sigmoid gate, so the output is a filtered version of the cell state for the current time-step.

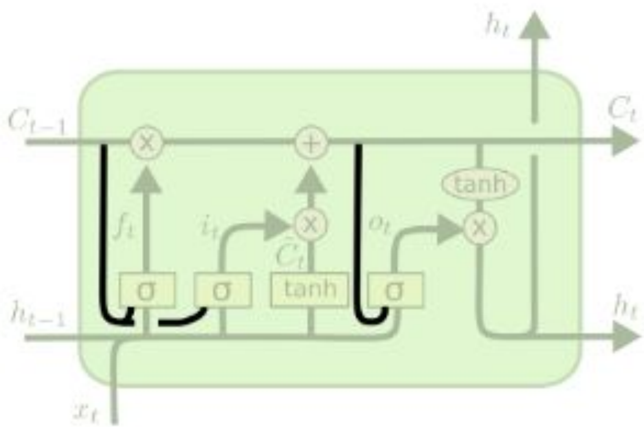


$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$

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

Variations of LSTM

- One LSTM variant, is adding "peephole connections." This means that we let the gate layers look at the cell state.

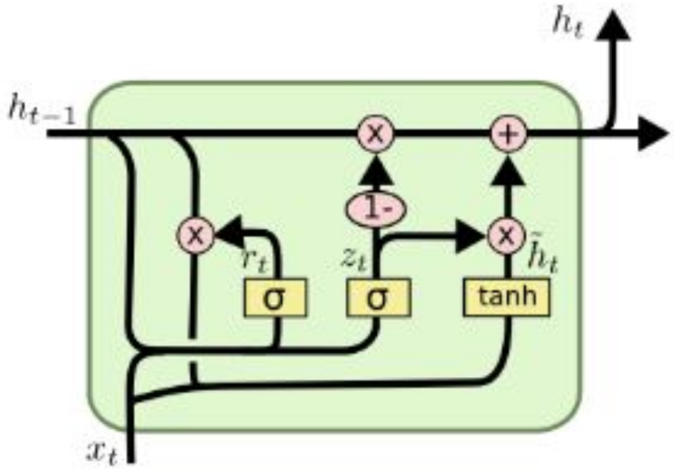


$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] + b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] + b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] + b_o\right)$$

ftp://ftp.idsia.ch/pub/juergen/TimeCount-IJCNN2000.pdf

GRU Networks

- GRU - Gated Recurrent Unit
- The main idea is the same as LSTM – the cell state (gradient of 1 between time-steps with a factor of the gates output).



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

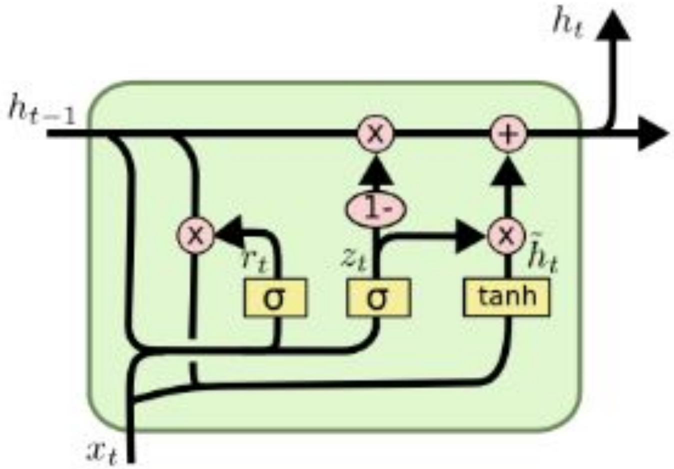
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

GRU Networks

- GRU - Gated Recurrent Unit
- More efficient – combines the input gate and output gate to one update gate (1 tanh layer).
- It ties the forget and update gates together by multiplying the update gate by the complement of the Sigmoid output.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

LSTM with Pytorch

- Pytorch provides two options:
  - Forward the input manually, step by step by *nn.LSTMCell*
  - Forward a complete sequence at once with *nn.LSTM*

class `torch.nn.LSTM`(**args*, ***kwargs*) [source]

Parameters:

- **input_size** – The number of expected features in the input x
- **hidden_size** – The number of features in the hidden state h
- **num_layers** – Number of recurrent layers.
- **bias** – If False, then the layer does not use bias weights b_ih and b_hh. Default: True
- **batch_first** – If True, then the input and output tensors are provided as (batch, seq, feature)
- **dropout** – If non-zero, introduces a dropout layer on the outputs of each RNN layer except the last layer
- **bidirectional** – If True, becomes a bidirectional RNN. Default: False

LSTM with Pytorch

Example of forwarding a sequence:

```
num_features = 10
hidden_size = 6
num_layers = 1
lstm = nn.LSTM(num_features, hidden_size, num_layers, batch_first=True)
```

```
batch_size = 1
seq_len = 7
states = (Variable(torch.zeros(num_layers, batch_size, hidden_size)),
          Variable(torch.zeros(num_layers, batch_size, hidden_size)))

input = Variable(torch.randn(batch_size,seq_len,num_features))
out,(h,c) = lstm(input,states)
print out
print h,c
```

LSTM with Pytorch

Example of forwarding a sequence:

```
Variable containing:
(0 ,.,.) =
  0.1911 -0.0405 -0.0895  0.1315  0.0557  0.0234
 -0.2337 -0.2422 -0.0903  0.0049  0.0436 -0.1097
  0.0245 -0.1949  0.2142 -0.1574  0.0955 -0.0290
  0.1165 -0.4309  0.0730 -0.1575  0.1069 -0.2643
 -0.1798 -0.2926 -0.0049 -0.0627 -0.1374 -0.1681
 -0.1291 -0.1086 -0.1107  0.1270 -0.0470 -0.2143
 -0.3929 -0.3978  0.0682 -0.0374  0.0957 -0.3295
[torch.FloatTensor of size 1x7x6]
```

7 output vectors, 1 for each time-step.

```
Variable containing:
(0 ,.,.) =
 -0.3929 -0.3978  0.0682 -0.0374  0.0957 -0.3295
[torch.FloatTensor of size 1x1x6]
```

The output of the last time step is the same as h

```
 Variable containing:
(0 ,.,.) =
 -0.5738 -0.5001  0.1389 -0.0843  0.2042 -0.4244
[torch.FloatTensor of size 1x1x6]
```

Cell state

Word Embedding

- Word embedding is:
  - Converting a sparse (discrete) vector to contiguous and dense vector
  - The new vector captures semantic information of the word

- There are two approaches of doing that:
  - Using an **unsupervised algorithm** (such as Word2Vec) as a pre-process, then using the output vectors as inputs of our model.
  - Training an **embedding layer** in our original model end-to-end (similar to a linear layer from dictionary size to hidden size)

Embedding Layer

- *torch.nn.Embedding* example:
  - Before that, we construct a word_to_index dictionary {'word' : index}

```python
# dictionary of size 10
# an Embedding module containing 10 tensors of size 3
embedding = nn.Embedding(10, 3)
# a batch of 2 samples of 4 indices each
input = Variable(torch.LongTensor([[1,2,4,5],[4,3,0,9]]))

print embedding(input)
```

```
Variable containing:
(0 ,.,.) =
  0.1867 -0.5689 -0.0966
  0.8406  0.1224 -0.3748
 -0.8595  1.6034  1.0484
  0.3727 -0.3231 -0.1000

(1 ,.,.) =
 -0.8595  1.6034  1.0484
 -1.6189  0.3070 -0.9183
  0.2407  1.2565 -0.5948
  1.5346 -2.2125  1.3315
[torch.FloatTensor of size 2x4x3]
```

- Here, the dictionary is of size 10 and therefore the maximum index number that the module can accept is 9.

Recall: Language model

$\hat{y} \in \mathbb{R}^{|V|}$ is a probability distribution over the vocabulary

Same cross entropy loss function but predicting words instead of classes

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

RNN Language model

Evaluation could just be negative of average log probability over dataset of size (number of words) T:

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

But more common: Perplexity: 2^J (or e^J)

Lower is better!

Language Model Example: The Model

```
# RNN Based Language Model
class RNNLM(nn.Module):
    def __init__(self, vocab_size, embed_size, hidden_size, num_layers):
        super(RNNLM, self).__init__()
        self.embed = nn.Embedding(vocab_size, embed_size)
        self.lstm = nn.LSTM(embed_size, hidden_size, num_layers, batch_first=True)
        self.linear = nn.Linear(hidden_size, vocab_size)
        self.init_weights()

    def init_weights(self):
        self.embed.weight.data.uniform_(-0.1, 0.1)
        self.linear.bias.data.fill_(0)
        self.linear.weight.data.uniform_(-0.1, 0.1)

    def forward(self, x, h):
        # Embed word ids to vectors
        x = self.embed(x)
        # Forward propagate RNN
        out, h = self.lstm(x, h)
        # Reshape output to (batch_size*sequence_length, hidden_size)
        out = out.contiguous().view(out.size(0)*out.size(1), out.size(2))
        # Decode hidden states of all time step
        out = self.linear(out)
        return out, h
```

Rearrange it to be in the size of a probability space over the vocabulary size

You can find the complete example here:

https://github.com/yunjey/pytorch-tutorial/tree/master/tutorials/02-intermediate/language_model

Language Model Example: Training procedure

```python
# Truncated Backpropagation
def detach(states):
    return [state.detach() for state in states]
# Training
for epoch in range(num_epochs):
    # Initial hidden and memory states
    states = (Variable(torch.zeros(num_layers, batch_size, hidden_size)),
              Variable(torch.zeros(num_layers, batch_size, hidden_size)))

    for i in range(0, ids.size(1) - seq_length, seq_length):
        # Get batch inputs and targets
        inputs = Variable(ids[:, i:i+seq_length])
        targets = Variable(ids[:, (i+1):(i+1)+seq_length].contiguous())

        # Forward + Backward + Optimize
        model.zero_grad()
        states = detach(states)
        outputs, states = model(inputs, states)
        loss = criterion(outputs, targets.view(-1))
        loss.backward()
        torch.nn.utils.clip_grad_norm(model.parameters(), 0.5)
        optimizer.step()
```

Turning the states to leaf nodes, so the gradients won't propagate from sequence to sequence

Trick to make sure the gradient does not vanish

You can find the complete example here:

https://github.com/yunjey/pytorch-tutorial/tree/master/tutorials/02-intermediate/language_model

Language Model Example: sampling procedure (inference)

```
with open(sample_path, 'w') as f:
    # Set intial hidden ane memory states
    state = (Variable(torch.zeros(num_layers, 1, hidden_size)),
             Variable(torch.zeros(num_layers, 1, hidden_size)))
    # Select one word id randomly
    prob = torch.ones(vocab_size)
    input = Variable(torch.multinomial(prob, num_samples=1).unsqueeze(1),
                     volatile=True)
    for i in range(num_samples):
        # Forward propagate rnn
        output, state = model(input, state)
        # Sample a word id
        prob = output.squeeze().data.exp()

        word_id = torch.multinomial(prob, 1)[0]
        # Feed sampled word id to next time step
        input.data.fill_(word_id)

        # File write
        word = corpus.dictionary.idx2word[word_id]
        word = '\n' if word == '<eos>' else word + ' '
        f.write(word)
```

We sample word by word (seq=1), so the states are initialized only once at the beginning, but they are passed through all sampling process.

You can find the complete example here:

https://github.com/yunjey/pytorch-tutorial/tree/master/tutorials/02-intermediate/language_model

Language Model Example: sampling - results

```
Epoch [1/25], Step[0/1549], Loss: 9.208, Perplexity: 9974.08
Epoch [1/25], Step[100/1549], Loss: 6.045, Perplexity: 422.08
Epoch [1/25], Step[200/1549], Loss: 5.965, Perplexity: 389.44
Epoch [1/25], Step[300/1549], Loss: 5.828, Perplexity: 339.51
Epoch [1/25], Step[400/1549], Loss: 5.675, Perplexity: 291.40
Epoch [1/25], Step[500/1549], Loss: 5.153, Perplexity: 172.92
Epoch [1/25], Step[600/1549], Loss: 5.229, Perplexity: 186.63
Epoch [1/25], Step[700/1549], Loss: 5.401, Perplexity: 221.59
```

⋮

```
Epoch [25/25], Step[1000/1549], Loss: 2.314, Perplexity: 10.11
Epoch [25/25], Step[1100/1549], Loss: 2.495, Perplexity: 12.13
Epoch [25/25], Step[1200/1549], Loss: 2.257, Perplexity:  9.56
Epoch [25/25], Step[1300/1549], Loss: 2.100, Perplexity:  8.17
Epoch [25/25], Step[1400/1549], Loss: 2.043, Perplexity:  7.72
```

Language Model Example: sampling - results

<unk> = Unknown word

“the offer a southern bank managed to publish it 's early.
however he said drexel remains contributing to the offer almost adequate provisions an indication priority including a federal dispute.
the white house says polish money will be <unk> by the <unk> of the machinists and court.
u.s. trade development carla hills have been losing smaller firms with others.
stockbrokers and <unk> said they are leaving up the account for close of N people.
<unk> corp. said it will close N memories today by an agreement to acquire <unk>.
the partnership 's manager in operations to be completed the transition systems will be completed and will be expansion to the underwriters.
a turner spokesman declined to heart but due N that its new natural gas would be inserted in a <unk> cells with time blood state sales and other paper operations according to a spokeswoman.“