Deep Learning for Sequences

Lecture 3
Architectures: GRU's, LSTM's, Bidirectional,Multilayer

David Kriegman
Computer Science & Engineering
UCSD

Announcements

- Initial paper presentation assignments. See syllabus.
  - This include students who were both registered & submitted forms.
  - Google form has been reopened for remaining students. Will close Tuesday at 7:00PM
  -

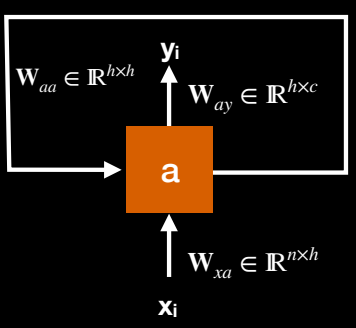
Last Lecture

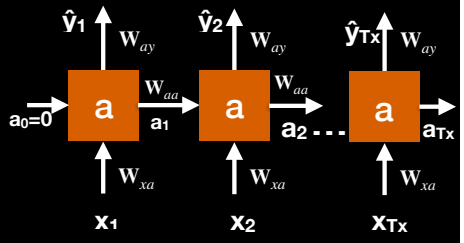
- One-hot representations of words and characters
- Problems with fully connected feedforward networks
- "Vanilla Recurrent networks
- Unrolling an RNN and Back Propogation through time
- Statistical language model and RNN character model

Vanilla Recurrent Neural Nets

Elman Network

$a_i = g_1(W_{xa}x_i + W_{aa}a_{i-1} + b_a)$
$y_i = g_2(W_{ay}a_i + b_y)$

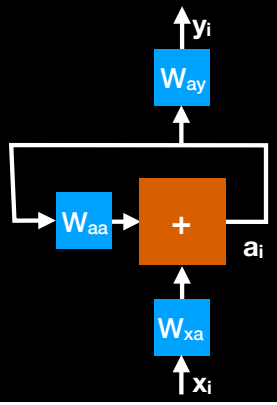




Two Simple RNN's

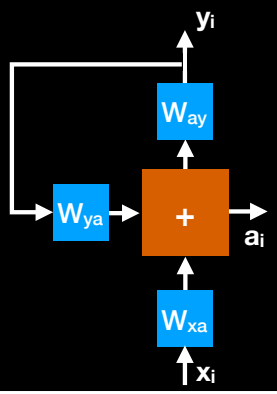
Elman Network, 1990

$a_i = g_1(W_{xa}x_i + W_{aa}a_{i-1} + b_a)$
$y_i = g_2(W_{ay}a_i + b_y)$



Jordan Network, 1997

$a_i = g_1(W_{xa}x_i + W_{yay_{i-1}} + b_a)$
$y_i = g_2(W_{ay}a_i + b_y)$



Examples from Karpathy's Blog

"The Unreasonable Effectiveness of Recurrent Neural Networks", Andrej Karpathy, 2015

- Shakespeare
  - Trained on all works of William Shakespeare
  - 512 Hidden units
  - 1000's of epochs
  - Characters only!!!
- Results as Function of number of epochs (Tolstoy data)
- Latex (produces mathy looking latex)
  - 16MB of latex.
- C code
  - Trained on 474MB of code from Linux Github Repo

I didn't tell the whole story 🙈

Problems with Vanilla RNN

- Exploding gradients

  A solution: Gradient clipping

Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if

- Vanishing gradients
  - A solution: Regularization

$$\Omega = \sum_k \Omega_k = \sum_k \left(\frac{\left\| \frac{\partial \mathcal{E}}{\partial \mathbf{x}_{k+1}} \frac{\partial \mathbf{x}_{k+1}}{\partial \mathbf{x}_k} \right\|}{\left\| \frac{\partial \mathcal{E}}{\partial \mathbf{x}_{k+1}} \right\|} - 1 \right)^2 \quad (9)$$

- Information doesn't propagate far back in time and earlier elements of input sequences don't affect output.
  - A solution: Tastier networks such as LSTM and GRU
  - Also less prone to vanishing gradient problems

[Pascanu, Mikolov, Bengio, "On the difficulty of training recurrent neural networks", 2012]

Gated Recurrent Unit (GRU)

[Cho, Van Merriënboer, Gulcehre, Bahdanau, Bougares, Schwenk, & Bengio. "Learning phrase representations using RNN encoder-decoder for statistical machine translation", 2014]

Gates in Recurrent Nets

- Gates are used in GRU's (2 gates and LSTM's
- These aren't digital gates
- Gate signals: Γ ranging from 0 to 1
- Usually Γ is produced by sigmoid function σ(x) whose output ranges from 0 to 1
- Used to mix two signals **a** and **b** as convex combination:

$$\Gamma\ \mathbf{a} + (1 - \Gamma)\mathbf{b}$$

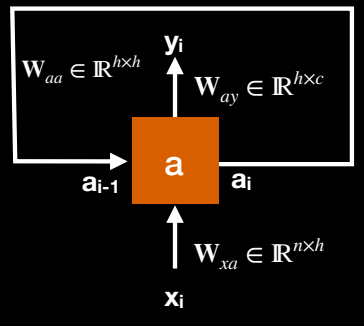
- or scale a signal (considering **b** to be 0)

$$\Gamma\ \mathbf{a}$$

Vanilla Recurrent Neural Nets

Add in recurrent loop

Input $\mathbf{x} \in \mathbb{R}^n$ n-D vector representing sequence element
Output $\mathbf{y} \in \mathbb{R}^c$: C classes
Hidden layer $\mathbf{a} \in \mathbb{R}^h$: h-nodes

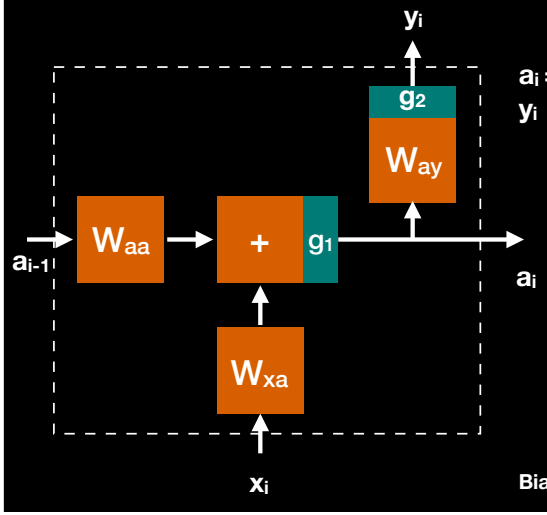


$a_i = g_1(W_{xa}x_i + W_{aa}a_{i-1} + b_a)$
$y_i = g_2(W_{ay}a_i + b_y)$

where
g_1 is usually tan
g_2 is usually softmax

Note:
a_0: State must be initialized, usually vector zeros

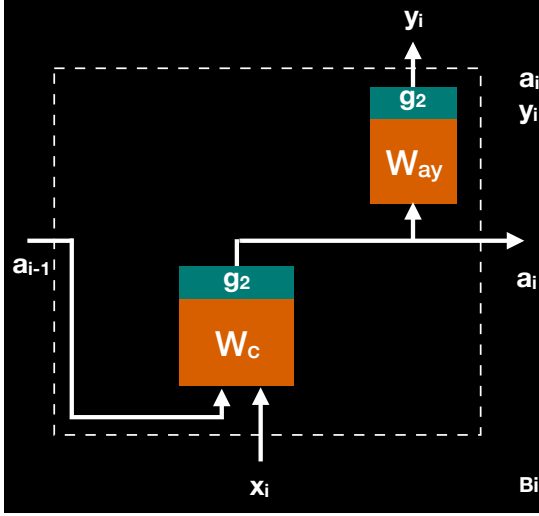
Unpacking the Elman RNN



$a_i = g_1(W_{xa}x_i + W_{aa}a_{i-1} + b_a)$
$y_i = g_2(W_{ay}a_i + b_y)$

Bias terms not shown in drawing

Slightly modifying the Elman RNN



$a_i = g_1(W_c\,[x_i, a_{i-1}] + b_a)$

$y_i = g_2(W_{ay} a_i + b_y)$

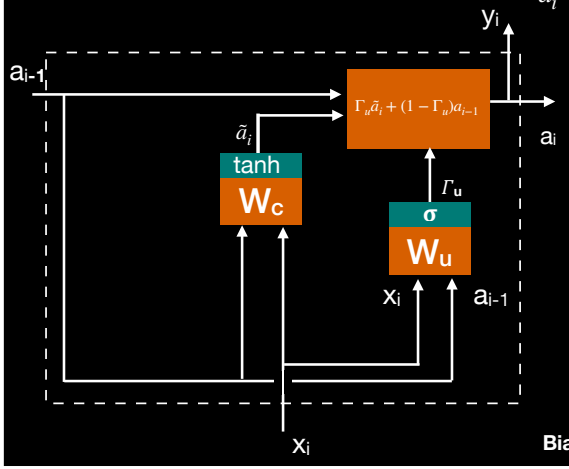
Bias terms not shown in drawing

Simple GRU

$$\Gamma_u = \sigma(W_u[a_{i-1}, x_i] + b_u)$$

$$\tilde{a}_i = \tanh(W_c[a_{i-1}, x_i] + b_c)$$

$$a_i = \Gamma_u * \tilde{a}_i + (1 - \Gamma_u) * a_{i-1}$$



Gates

Γ_u: Update Gate

Parameters

W_c, W_u, b_c, b_u

Output

$y_i = a_i$

Bias terms not shown in drawing

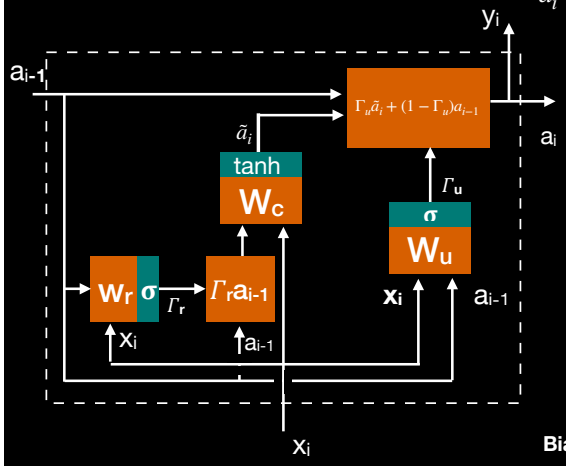
Full GRU

$$\Gamma_u = \sigma(W_u[a_{i-1}, x_i] + b_u)$$

$$\Gamma_r = \sigma(W_r[a_{t-1}, x_i] + br)$$

$$\tilde{a}_i = \tanh(W_c[\Gamma_r * a_{i-1}, x_i] + b_c)$$

$$a_i = \Gamma_u * \tilde{a}_i + (1 - \Gamma_u) * a_{i-1}$$



Gates

Γ_u: Update Gate

Γ_r: Reset Gate

Parameters

W_c, W_u, W_r, b_c, b_u, b_r

Output

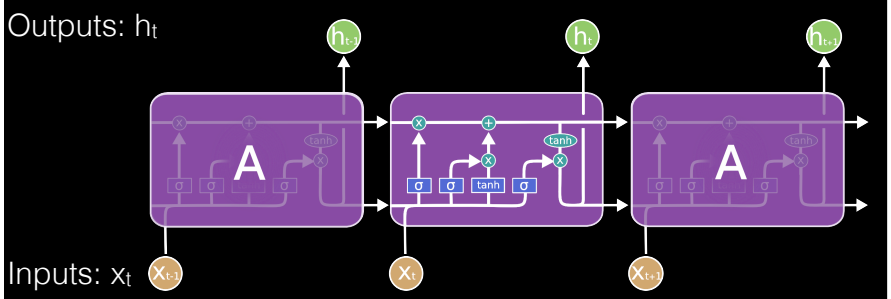
$y_i = a_i$

Bias terms not shown in drawing

LSTM Networks

- Long Short Term Memory
- Feedback of both state and output
- Three gates instead for LSTM vs. Two for GRU

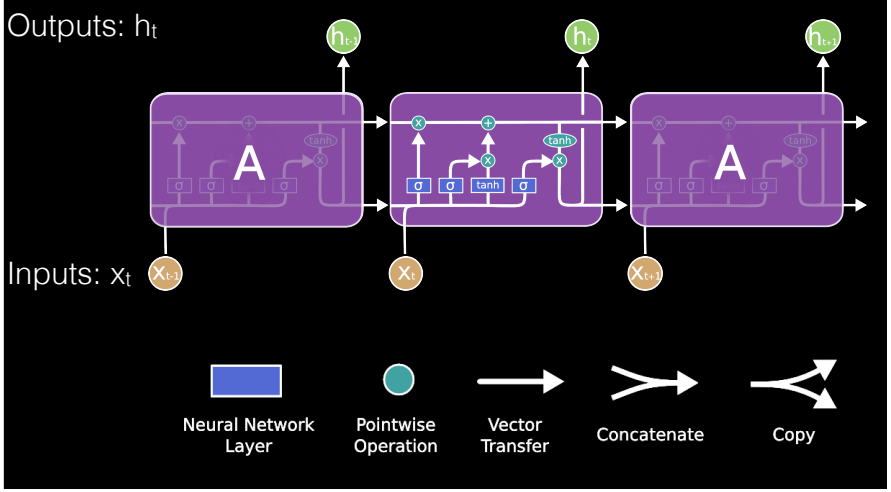
LSTM Networks [Hochreiter and Schmidhuber, 1997]



Four layers gate what to pass forward to next time instance in LSTM networks.

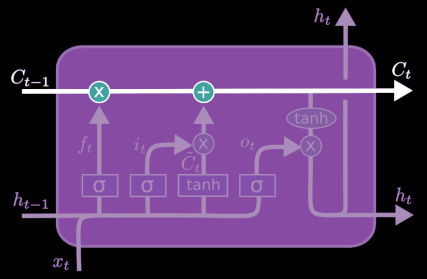
Images from Colah's Blog: "Understanding LSTM Networks"

LSTM Networks



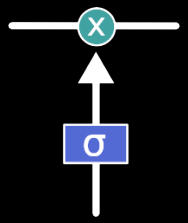
Let's look at the different components that make up an LSTM.

Cell State: C_t



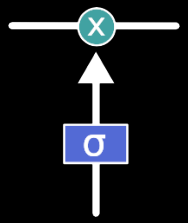
- The cell state C_t is directly carried forward from one time instance to the next.
- Note that state is a vector, not scalar
- C_t is modified along the way

LSTM Gates



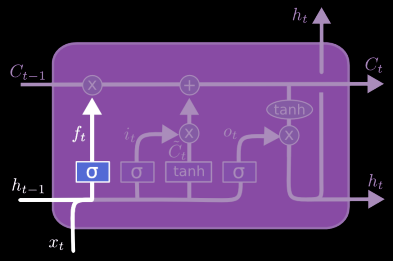
This is an LSTM “gate” which makes an element-wise decision on what to allow through. It is a fully connected layer followed by a sigmoid with outputs between 0 and 1.

LSTM Gates



Together with the “x” operation, this layer is a gate-keeper for the channel.

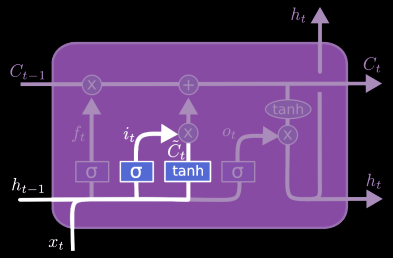
Forget Gate



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

The forget gate decides what elements of the previous cell state should be forgotten.

Input Gate

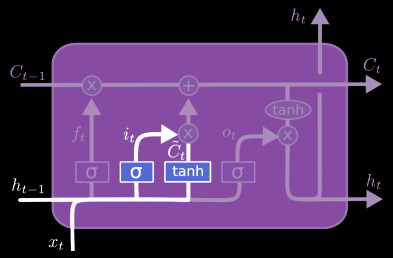


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

The input gate decides what new information should be added to the cell state.

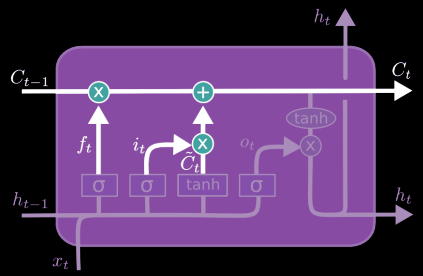
Input Gate



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

This new information is linearly transformed and then squashed by the ***tanh*** function which forces the output to lie between -1 and 1.

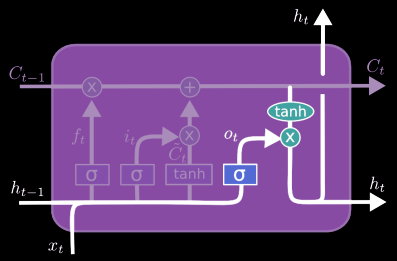
Cell State is updated



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

Outputs from the forget gate and the input gate combine to update the current cell state.
Note: GRU took convex combination of C and $\tilde{C}$

Output Gate



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

The output gate decides what information from the cell state should be output by the LSTM.

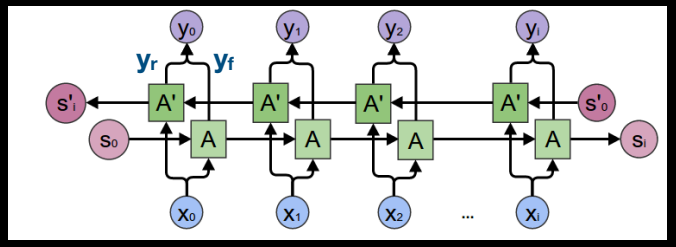
LSTM vs GRU

LSTM	GRU
• 1997	• 2014
• 3 Gates	• 2 Gates
• More parameters	• Fewer parameters
• More proven	• Better training on small datasets

Both are implemented in major frameworks and easy to try

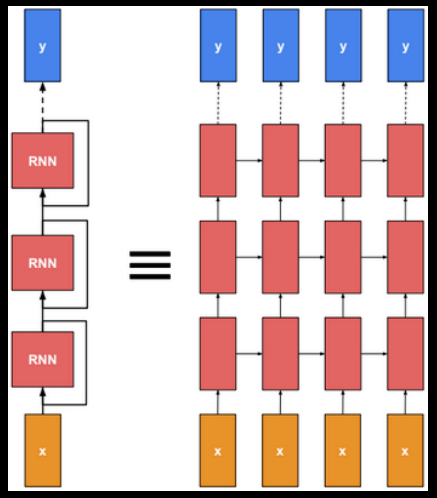
```
model = Sequential()
model.add(LSTM(128, input_shape=(maxlen, len(chars))))
model.add(Dense(len(chars)))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer=RMSprop(lr=0.01))
```

Bidirectional RNN



- Two independent RNN's (Can be any type)
  - Forward direction as usual
  - Reverse starts at end of sequence and goes backward to start
- Once forward and reverse RNN's are complete, outputs for each time step are concatenated.
- Concatenated outputs may be passed through fully connected layer $\hat{y}_i = g(W_y[y_f, y_r] + b_y)$

Multilayer RNN



- Can be any RNN type (LSTM, GRU)
- Can be directional
- Can have any number layers
  Usually 2 or 3
- Sometimes called a "Deep RNN"
- Backprop depth is really governed by unrolled sequence length

RNN Application Architectures

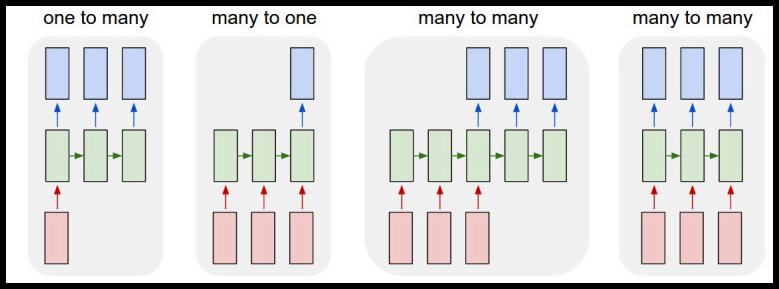
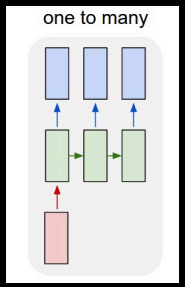


Diagram from Blog: Karpathy, "The Unreasonable Effectiveness of Recurrent Neural Networks"

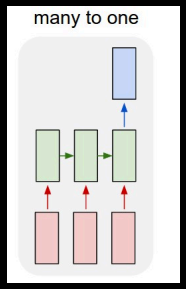
One to Many



- Sequence generation
  - Audio or speech generation
  - Image caption generation
- Loss function defined over all outputs

Diagram from Blog: Karpathy, "The Unreasonable Effectiveness of Recurrent Neural Networks"

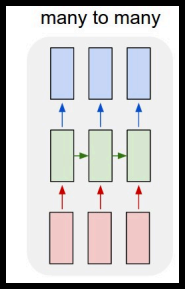
Many to One



- Sequence classification
  - Sentiment analysis
  - Video classification
  - Stock Market "Buy" decision
- Loss function defined over single output. Usually cross entropy

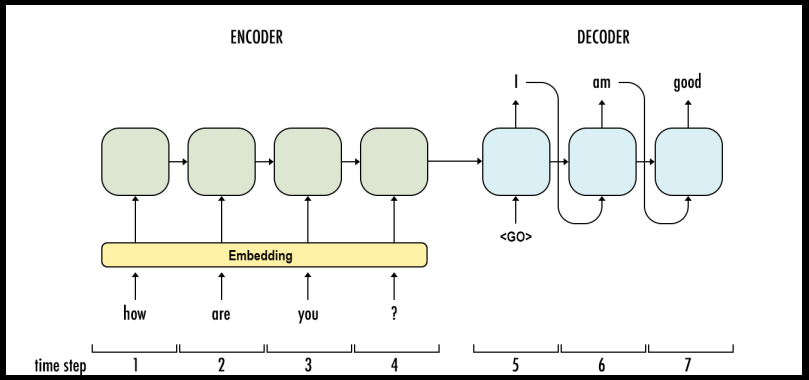
Diagram from Blog: Karpathy, "The Unreasonable Effectiveness of Recurrent Neural Networks"

Many to Many — same length



- Usually classifying elements of a sequence
  - Named entity classification
  - Video segmentation/activity labelling
  - Statistical language model (char RNN)
  - Some approaches to speech recognition (using CTC)
- Loss function defined over all outputs. Usually sum of losses for each output

Many to Many — different length



- Encoder-Decoder Architecture with two RNN's
- Input sequence encoded into a vector; output sequence generated from v
- Language translation
- Video captioning
- Question-Answer
-

Image from Blog: Chablani "Sequence to sequence model: Introduction and concepts"