Deep Learning for NLP
Recurrent Neural Networks

Nils Reimers

Course-Website: www.deeplearning4nlp.com

Recommended Readings

- http://karpathy.github.io/2015/05/21/rnn-effectiveness/
  - Video: https://skillsmatter.com/skillscasts/6611-visualizing-and-understanding-recurrent-networks
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- C224d Lecture 7: https://www.youtube.com/watch?v=rFVYTydGLr4

Excursion: Language Model

- Compute the probability of a sentence

- Useful in machine translation
  - Word ordering: p(the cat is small) > p(small the cat is)
  - Word choice: p(walking home after school) > p(walking house after school)

Excursion: Language Model

- unigram language model:

$$P(w_i|w_{i-1}) \qquad P(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}w_i)}{\text{count}(w_{i-1})}$$

- bigram language model:

$$P(w_i|w_{i-1}, w_{i-2}) \qquad P(w_i|w_{i-1}, w_{i-2}) = \frac{\text{count}(w_{i-2}w_{i-1}w_i)}{\text{count}(w_{i-2}w_{i-1})}$$

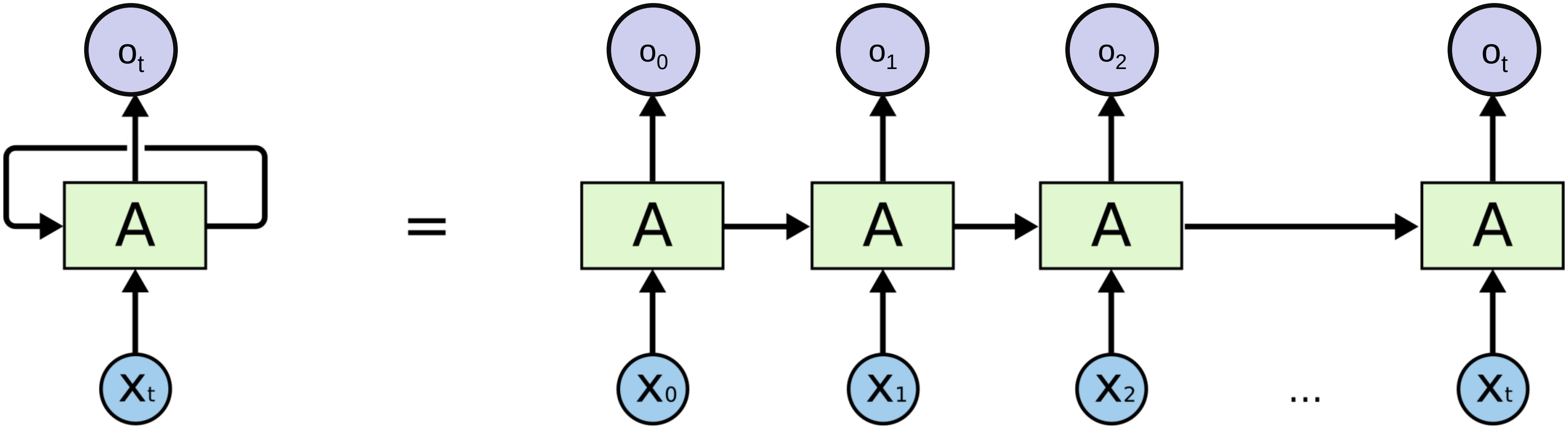
- Such models can also be used to generate new sentences
  - Sample word w_i with probability $P(w_i|w_{i-1}, w_{i-2})$
- Longer n-gram models give a better accuracy
  - Required training data &model size increases extremely
- Long term relationships impossible to capture
  - p(I grew up in *France* and lived there until I was 18. Therefore I speak fluent *French*) >
    p(I grew up in *France* and lived there until I was 18. Therefore I speak fluent *English*)

Excursion: Language Models

- Performance increases with longer n-grams
- There are a lot of n-grams
  - 500 000 German words (according to Duden)
  - 2-grams: 250 billion combinations
  - 3-grams: > 10^{17} combinations
  - 4-grams: > 10^{22} combinations

- Gigantic training corpus & RAM requirement
  - *"Using one machine with 140GB RAM for 2.8 days, we built an unpruned model on 126 billion tokens"* (Heafield et al.)

Recurrent Neural Network

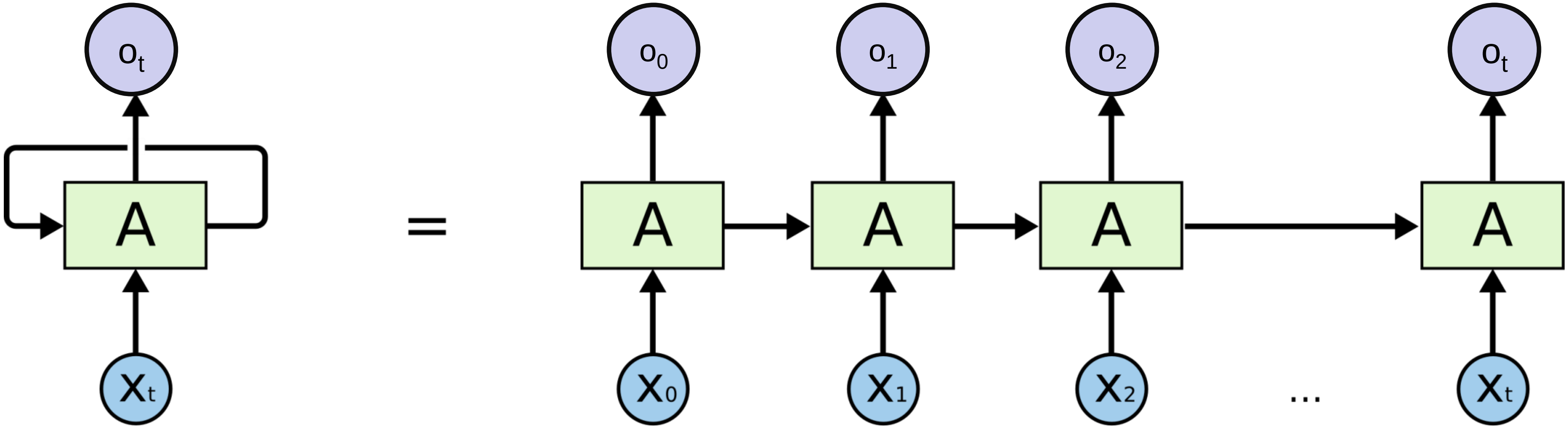


- Recurrent Neural Network have an internal state
- State is passed from input x_t to x_{t+1}

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

Language Models with RNN

- Let x_0, x_1, x_2… denote words (or characters)
- Let o_0, o_1, o_2… denote the probability of the sentence
- Memory requirement scales nicely (linear with the number of word embeddings / number of character)



RNN as Generative Language Models

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{\acute{e}tale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma **??**. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

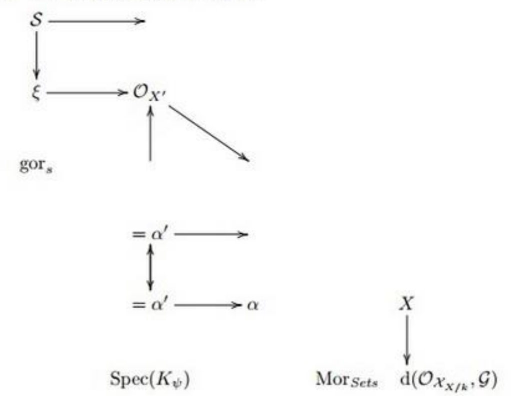
be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram



is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type f_*. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas **??**.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \;\; \text{-}1(\mathcal{O}_{X_{\acute{e}tale}}) \longrightarrow \mathcal{O}_{X_\ell}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{v}})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition **??** and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. □

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma **??**. This is a sequence of $\mathcal{F}$ is a similar morphism.

Generated LaTeX-Code from an Character-RNN

Img Source: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

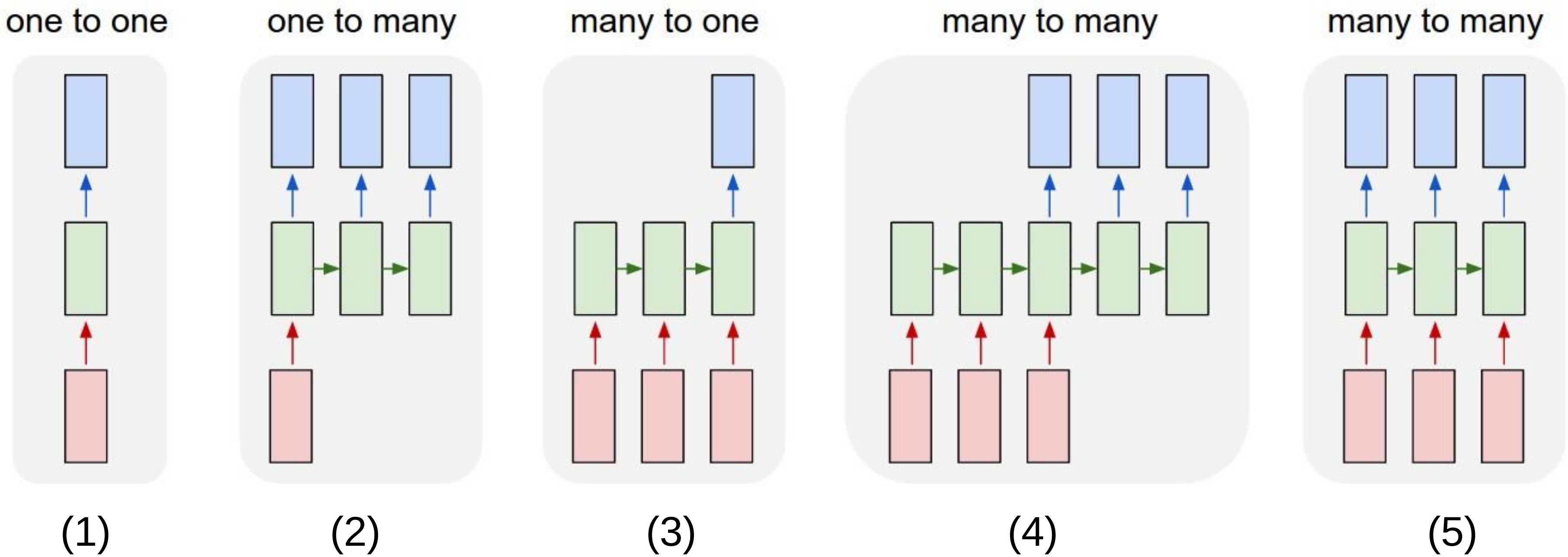
RNN as Generative Language Models

```c
/*
 * If this error is set, we will need anything right after that BSD.
 */
static void action_new_function(struct s_stat_info *wb)
{
  unsigned long flags;
  int lel_idx_bit = e->edd, *sys & ~((unsigned long) *FIRST_COMPAT);
  buf[0] = 0xFFFFFFFF & (bit << 4);
  min(inc, slist->bytes);
  printk(KERN_WARNING "Memory allocated %02x/%02x, "
    "original MLL instead\n"),
    min(min(multi_run - s->len, max) * num_data_in),
    frame_pos, sz + first_seg);
  div_u64_w(val, inb_p);
  spin_unlock(&disk->queue_lock);
  mutex_unlock(&s->sock->mutex);
  mutex_unlock(&func->mutex);
  return disassemble(info->pending_bh);
}
```

Generated C-Code from an Character-RNN

Img Source: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Topologies of Recurrent Neural Network



1) Common Neural Network (e.g. feed forward network)
2) Prediction of future states base on single observation
3) Sentiment classification
4) Machine translation
5) Simultaneous interpretation

Img Source: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

(Vanilla) RNN

```
rnn = RNN()
y = rnn.step(x) # x is an input vector, y is the RNN's output vector
```

```
class RNN:
  # ...
  def step(self, x):
    # update the hidden state
    self.h = np.tanh(np.dot(self.W_hh, self.h) + np.dot(self.W_xh, x))
    # compute the output vector
    y = np.dot(self.W_hy, self.h)
    return y
```

- Compute the hidden state: $h_{t+1} = \tanh(W_{hh}h_t + W_{xh}x_t)$
- Compute the output: $y_{t+1} = W_{hy}h_{t+1}$

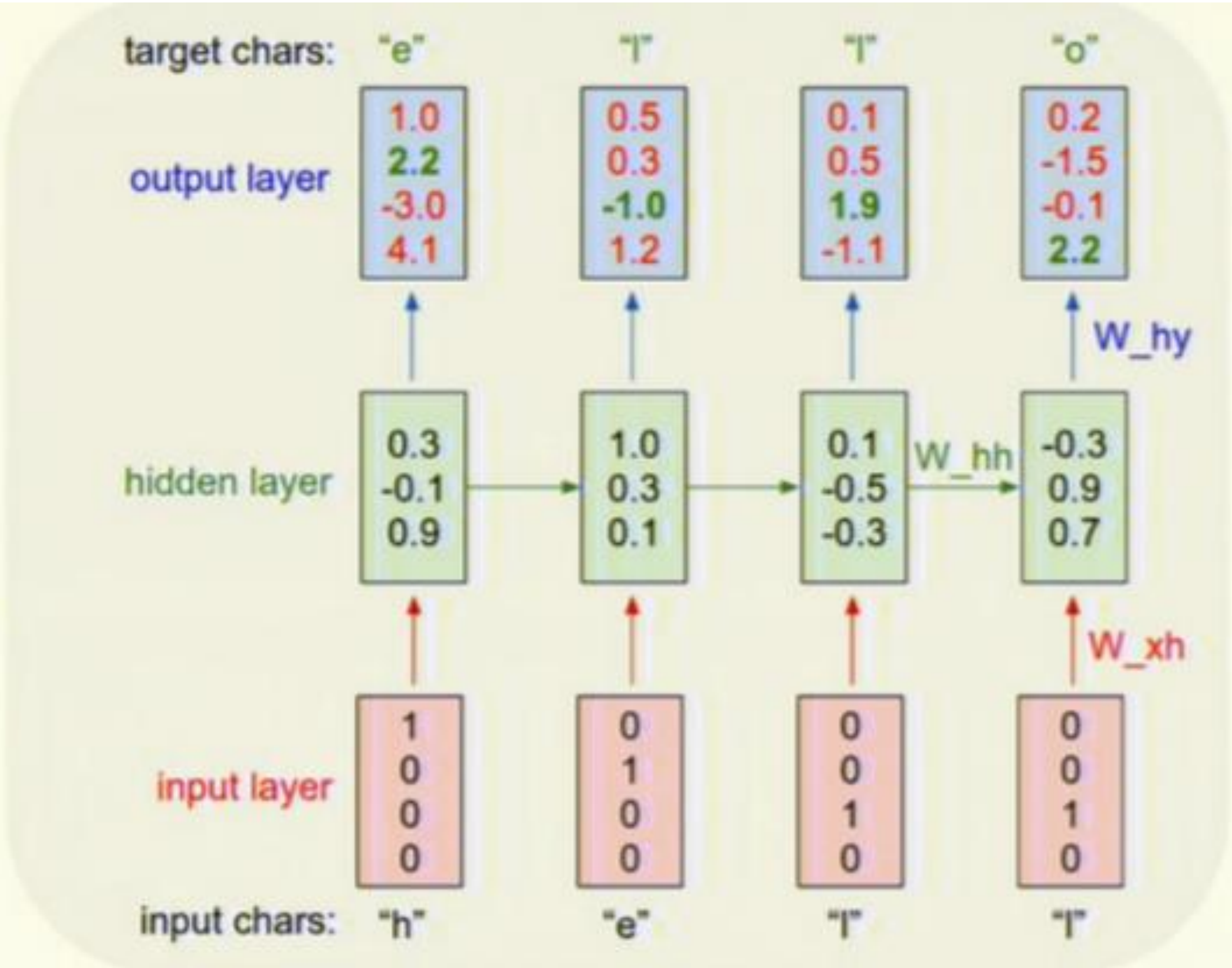
Img Source: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

(Vanilla) RNN

Character-level language model example

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”



Img Source: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

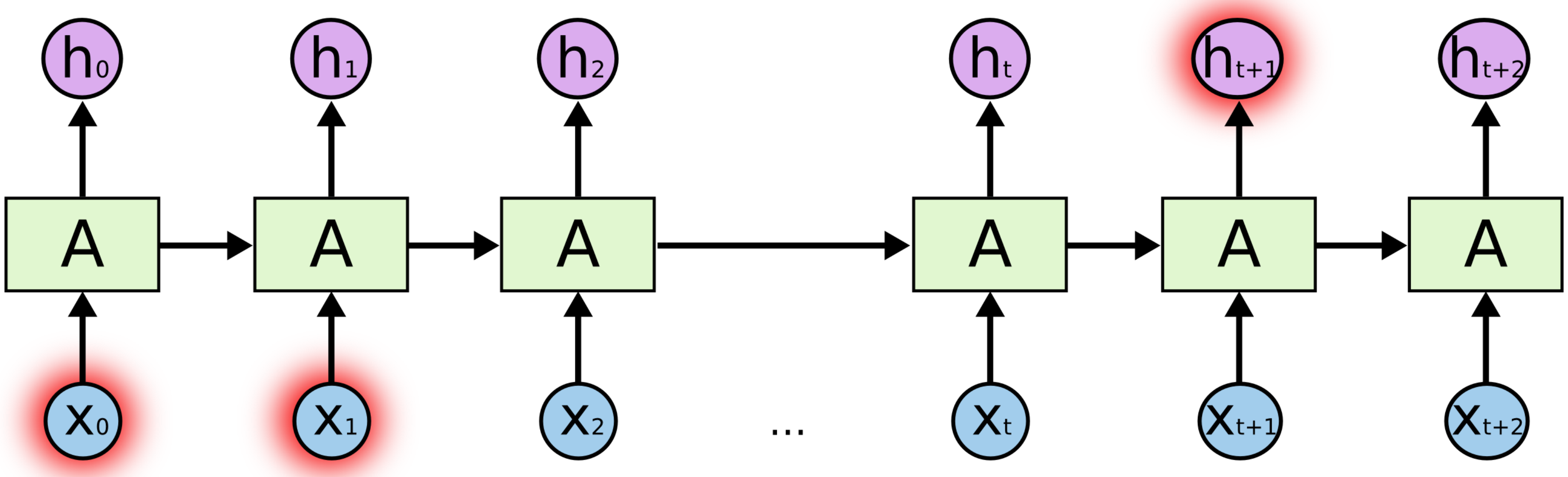
No Magic Involved (in Theory)

- You unroll your data in time
- You compute the gradients
- You use back propagation to train your network
- Karpathy presents a Python implementation for Char-RNN with 112 lines

- Training RNNs is hard:
  - Inputs from many time steps ago can modify output
  - Vanishing / Exploding Gradient Problem

- Vanishing gradients can be solved by Gated-RNNs like Long-Short-Term-Memory (LSTM) Models
  - LSTM became popular 2015 in NLP

Long-Short-Term Memory (LSTM)



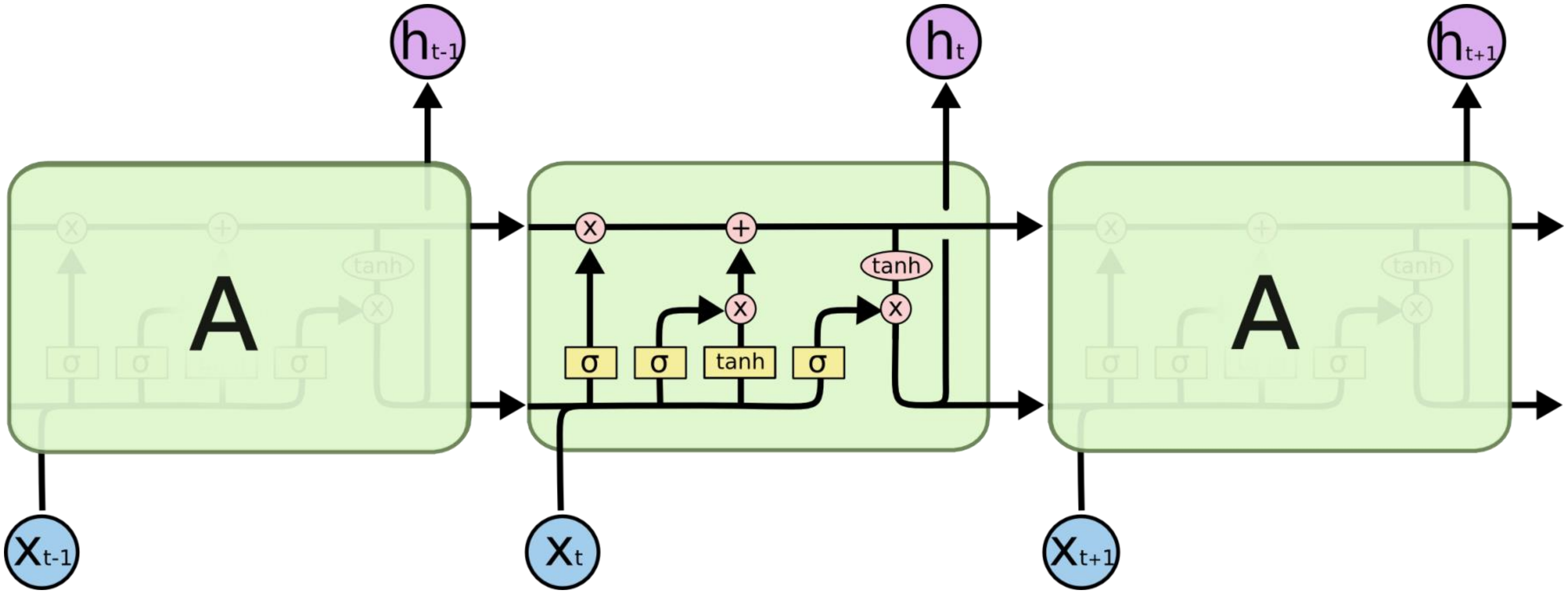
- Long-term dependencies:

  I grew up in France and lived there until I was 18. Therefore I speak fluent ???

- Presented (vanilla) RNN is unable to learn long term dependencies
  - Issue: More recent input data has higher influence on the output
- Long-Short-Term Memory (LSTM) models solves this problem

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

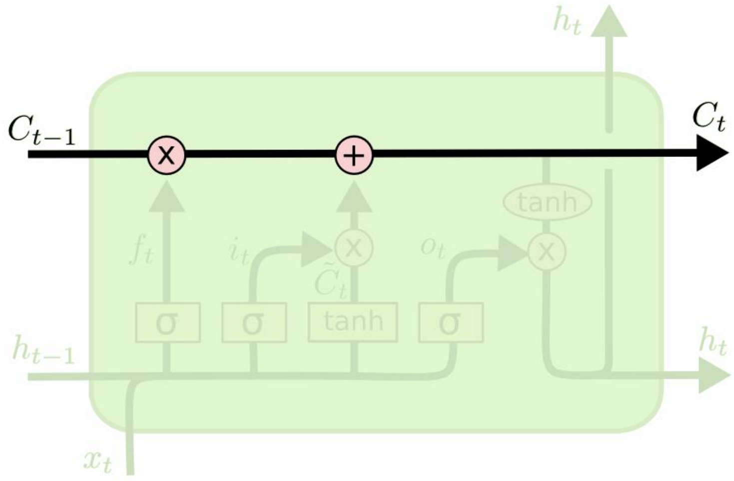
LSTM Model



- The LSTM model implements a *forget-gate* and an *add-gate*
- The models learns when to forget something and when to update internal storage

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

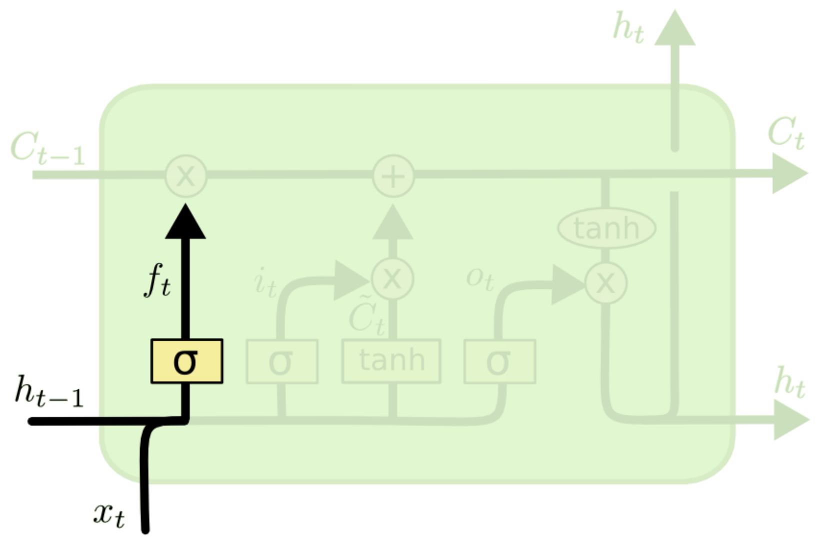
LSTM Model



- Core: Cell-state *C* (a vector of certain size)
- The model has the ability to remove or add information using Gates

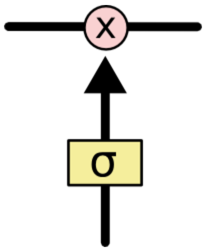
Img Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Forgot-Gate



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

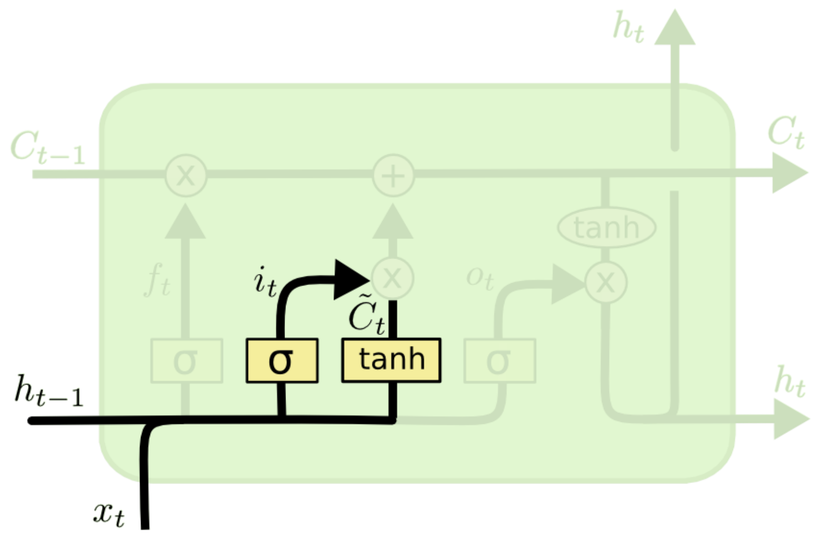
- Sigmoid function σ output a value between 0 and 1
- The output is point-wise multiplied with the cell state C_{t-1}
- Interpretation:
  - 0: *Let nothing through*
  - 1: *Let everything through*



- Example: When we see a new subject, forget gender of old subject

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

Set-Gate



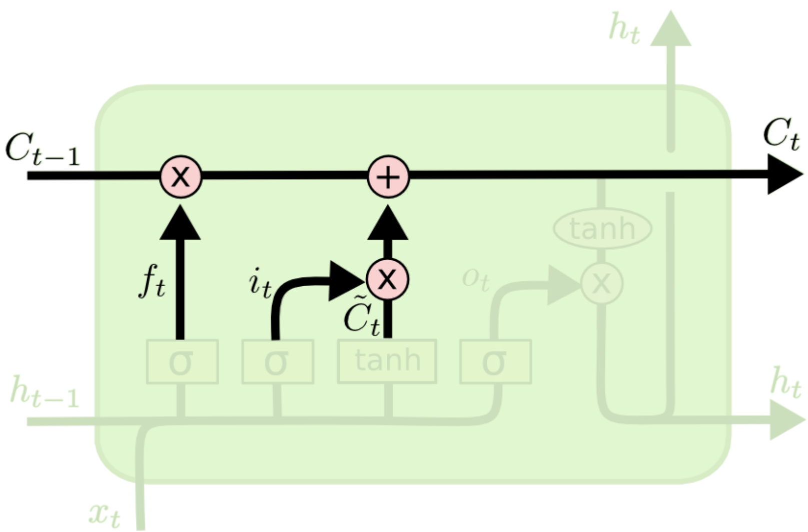
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

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

- Compute i_t which cells we want to update and to which degree (σ: 0 … 1)
- Compute the new cell value using the *tanh* function

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

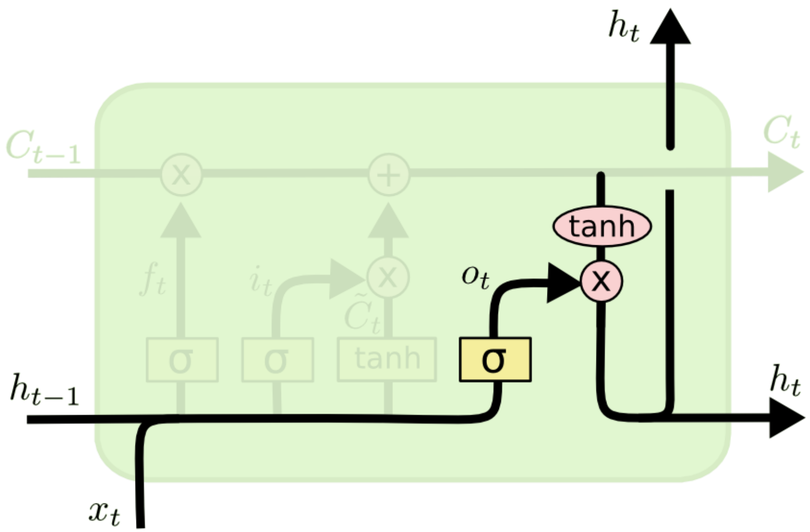
Update Internal Cell State



$$C_t = \underbrace{f_t * C_{t-1}}_{\text{Forget state cells}} + \underbrace{i_t * \tilde{C}_t}_{\text{Update state cells}}$$

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

Compute Output h_t



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

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

- We use the updated cell state C_t to compute the output
- We might not need the complete cell state as output
  - Compute o_t, defining how relevant each cell is for the output
  - Pointwise multiply o_t with *tanh*(C_t)

- Cell state C_t and output h_t is passed to the next time step

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

Training

- **RMSProp** / Adam / Adagrad (SGD can work too, but has much higher sensitivity to learning rate)
- **Clip gradients** (at 5.0 is a common value to use)
- **Initialize forget gates** with high bias (to encourage remembering at start) can help
- **L2 regularization** not very common, can even hurt sometimes
- **Dropout** always good along depth, but NOT along time (in the recurrent part). [*Zaremba et al.*]
- Typical training on good GPU: ~10mil params, 1-2 days

Img Source: https://skillsmatter.com/skillscasts/6611-visualizing-and-understanding-recurrent-networks

Searching for interpretable cells

Cell that turns on inside quotes:

"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.

Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."

Source: Karpathy et al., 2015, *Visualizing and Understanding Recurrent Networks*

Searching for interpretable cells

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Cell sensitive to position in line:

The sole importance of the crossing of the Berezina lies in the fact
that it plainly and indubitably proved the fallacy of all the plans for
cutting off the enemy's retreat and the soundness of the only possible
line of action--the one Kutuzov and the general mass of the army
demanded--namely, simply to follow the enemy up. The French crowd fled
at a continually increasing speed and all its energy was directed to
reaching its goal. It fled like a wounded animal and it was impossible
to block its path. This was shown not so much by the arrangements it
made for crossing as by what took place at the bridges. When the bridges
broke down, unarmed soldiers, people from Moscow and women with children
who were with the French transport, all--carried on by vis inertiae--
pressed forward into boats and into the ice-covered water and did not,
surrender.

Source: Karpathy et al., 2015, *Visualizing and Understanding Recurrent Networks*

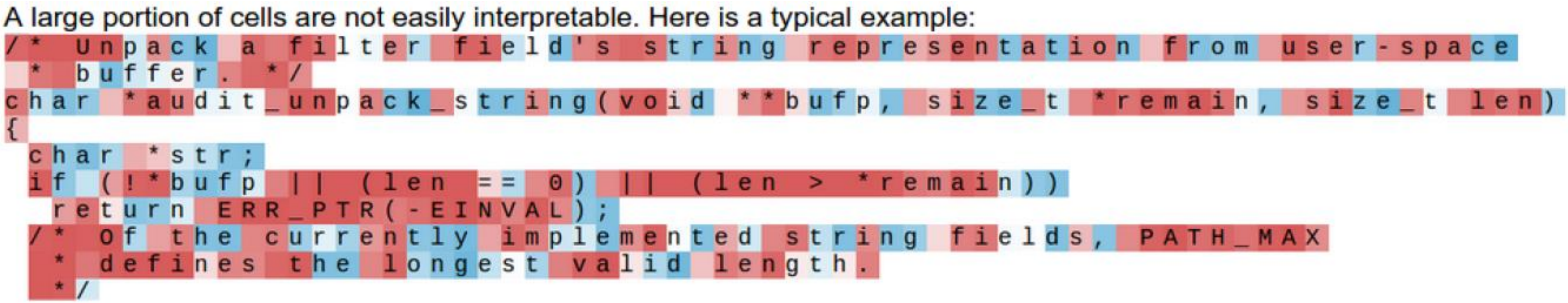
Searching for interpretable cells

Cell that is sensitive to the depth of an expression:

```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
 int i;
 if (classes[class]) {
  for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
   if (mask[i] & classes[class][i])
    return 0;
 }
 return 1;
}
```

Source: Karpathy et al., 2015, *Visualizing and Understanding Recurrent Networks*

Searching for interpretable cells



Source: Karpathy et al., 2015, *Visualizing and Understanding Recurrent Networks*

Variants of LSTM

- GRU
  - Cho et al., 2014, http://arxiv.org/pdf/1406.1078v3.pdf
- Depth Gated RNN
  - Yao et al., 2015, http://arxiv.org/pdf/1508.03790v2.pdf
- Clockwork RNN
  - Koutnik et al., 2014, http://arxiv.org/pdf/1402.3511v1.pdf

- Does the difference matter? Not really
  - Greff et al., 2015, http://arxiv.org/pdf/1503.04069.pdf
  - Jozefowicz et al., 2015, http://jmlr.org/proceedings/papers/v37/jozefowicz15.pdf

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