Lecture 8: RNNs, LSTMs

Administrative: Assignment 2

Due 4/27 11:59pm

- Multi-layer Neural Networks,
- Image Features,
- Optimizers

Administrative: Project proposal

Due 4/30 11:59pm

Goals:

- Develop a well-scoped research question
- What experiments will you design to answer this question
- How will you implement these experiments independently (no TA helper code)
- How will you explain your findings

Administrative: Midterm

On 4/28

- In class

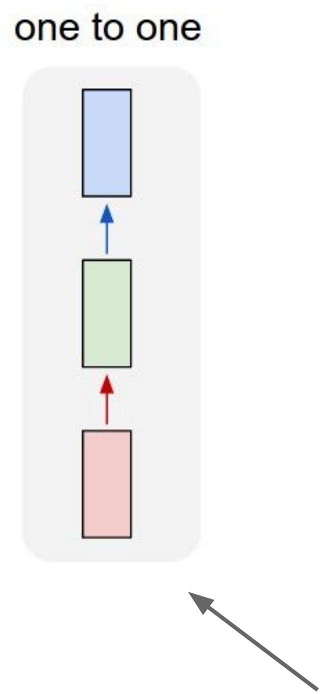
Refer to the exam archive for practice materials

Administrative: Fridays

This Friday

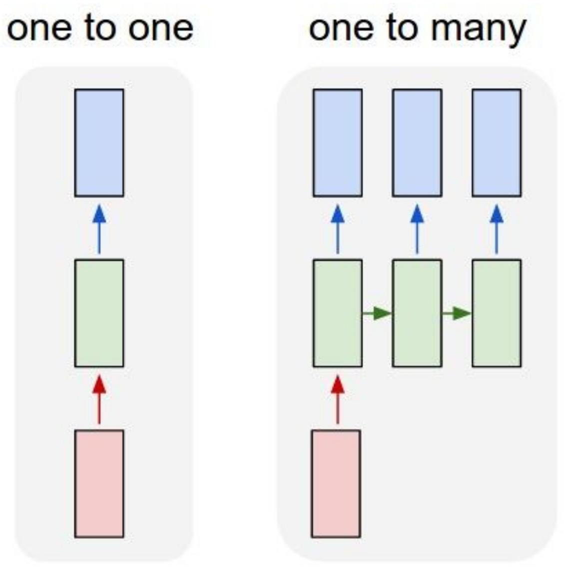
Exam review

“Vanilla” Neural Network



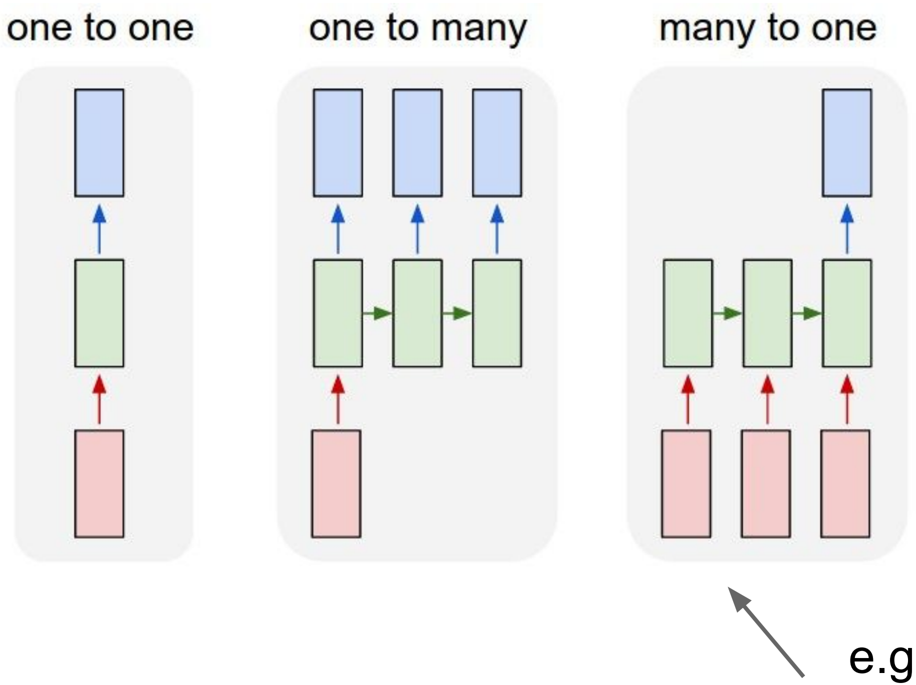
Vanilla Neural Networks

Recurrent Neural Networks: Process Sequences



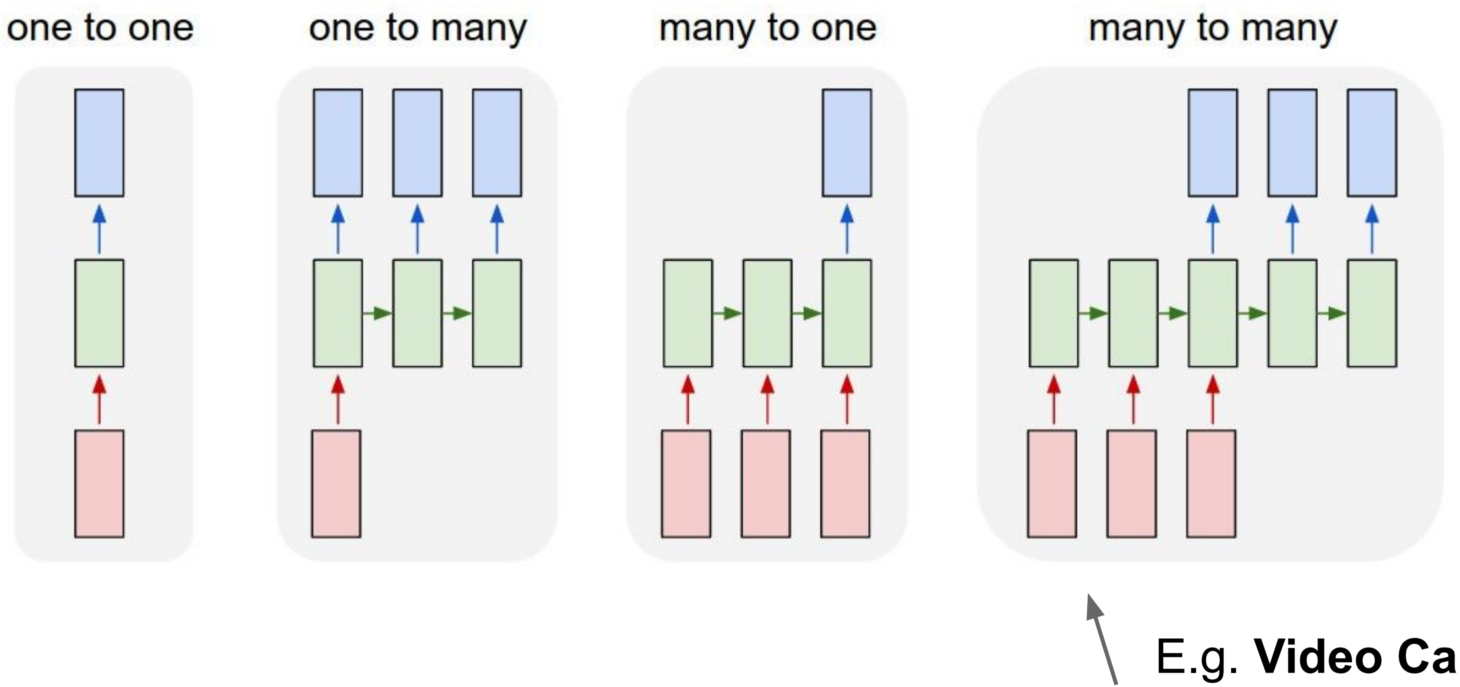
e.g. **Image Captioning**
image -> sequence of words

Recurrent Neural Networks: Process Sequences



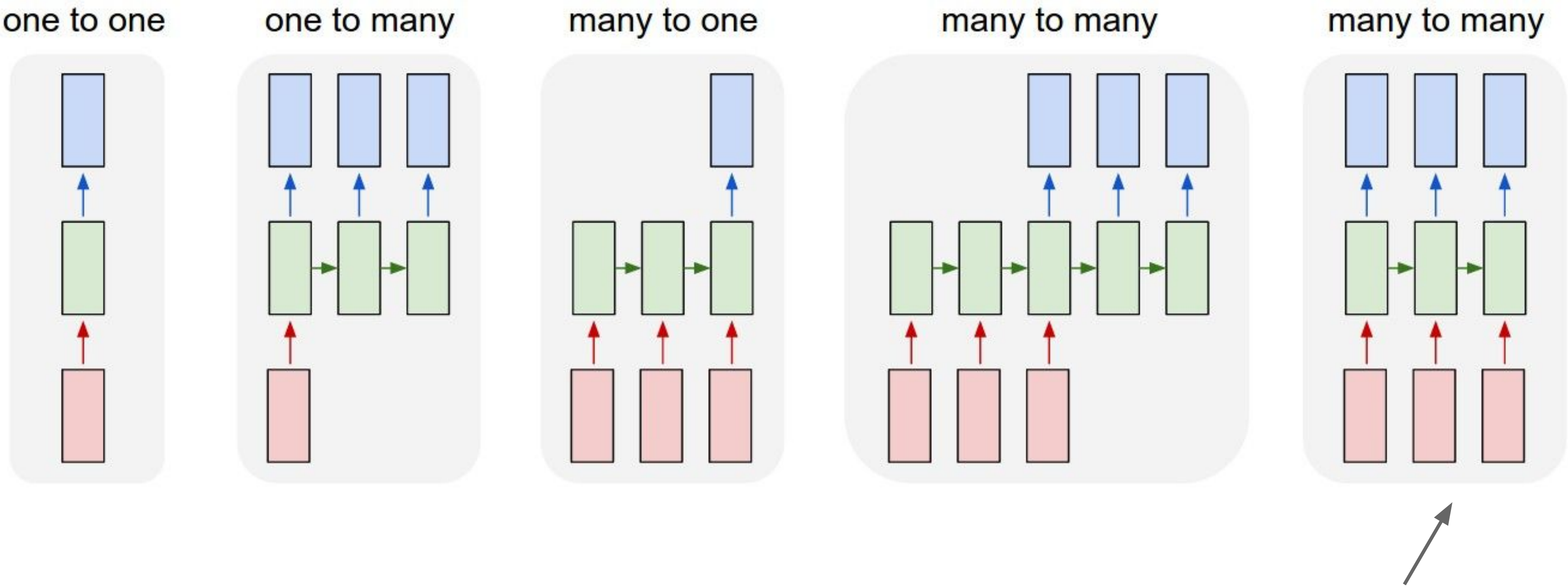
e.g. **action prediction, sentiment classification**
sequence of video frames -> action class

Recurrent Neural Networks: Process Sequences



E.g. **Video Captioning**
Sequence of video frames -> caption

Recurrent Neural Networks: Process Sequences



e.g. **Video classification on frame level**

So far: Representing words as discrete symbols

In traditional NLP, we regard words as discrete symbols:

hotel, conference, motel – each has its own symbol.

This is a localist representation

Such symbols for words can be represented by one-hot vectors:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Vector dimension = number of words in vocabulary (e.g., 500,000+)

So far: Representing words as dense vectors

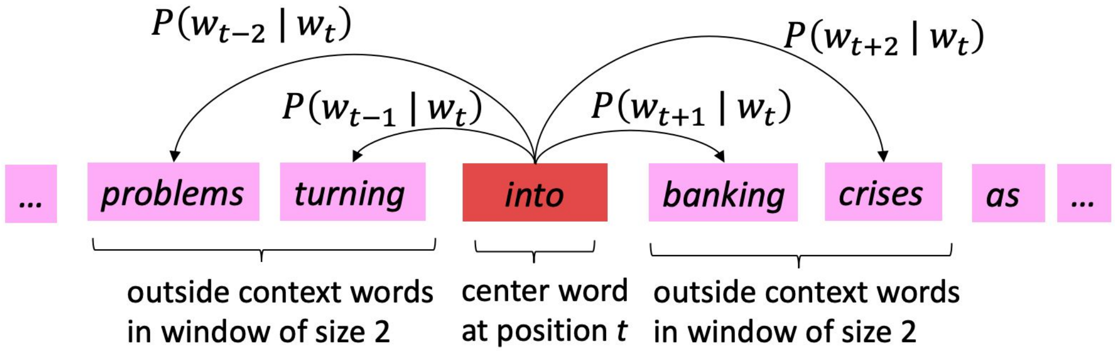
We will build a dense vector for each word,

- chosen so that it is similar to vectors of words that appear in similar contexts: e.g. jacket / coat / sweater.
- measuring similarity as the vector dot (scalar) product.
- Word vectors are also called (word) embeddings or (neural) word representations

$$\text{Jacket} = \begin{bmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{bmatrix}$$

Using a context size of 2!

Example windows and process for computing $P(w_{t+j} \mid w_t)$



The learning objective is to maximize the probabilities

For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_t!. Data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$$

θ is all variables to be optimized

The loss function

For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_t!. Data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} P(w_{t+j} \mid w_t; \theta)$$

θ is all variables to be optimized

The loss function $L(\theta)$ is the (average) negative log likelihood:

$$L(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \ne 0}} \log P(w_{t+j} \mid w_t; \theta)$$

Minimizing loss function ⇔ Maximizing predictive accuracy

How do we calculate this probability?

We want to minimize the objective function:

$$L(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m \\ j\neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

Question: How to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$

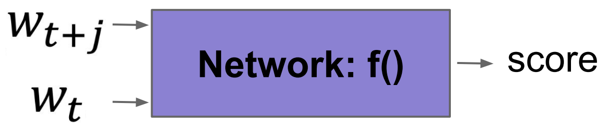
How do we calculate this probability?

We want to minimize the objective function:

$$L(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m \\ j\neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

Question: How to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$

The best idea #1: Use a neural network whenever you need to model a function



How did Word2Vec design $f_j()$?

We will use two vectors per word *w*:

- v_w when *w* is a center word
- u_w when *w* is a context word

Let $f_1(w_t, w_{t+1}) = w_t^T w_{t+1}$ be a simple dot product

- so now learnable parameters except the word vectors.

Then for a center word *c* and a context word *o*:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Remember the **Softmax (cross-entropy) Classifier**

Want to interpret raw classifier scores as **probabilities**

$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ Softmax Function

Probabilities must be >= 0

Probabilities must sum to 1

$$L_i = -\log P(Y = y_i|X = x_i)$$

	Unnormalized log-probabilities / logits		unnormalized probabilities		probabilities		Correct probs
cat	**3.2**	exp →	**24.5**	normalize →	**0.13**	→ compare ←	**1.00**
car	5.1		164.0		0.87		0.00
frog	-1.7		0.18		0.00		0.00

Cross Entropy

$$H(P, Q) = H(p) + D_{KL}(P\|Q)$$

Understanding the calculation

② Exponentiation makes anything positive

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

① Dot product compares similarity of *o* and *c*.

$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$

Larger dot product = larger probability

③ Normalize over entire vocabulary to give probability distribution

Understanding the calculation

② Exponentiation makes anything positive

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

① Dot product compares similarity of *o* and *c*.

$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$

Larger dot product = larger probability

③ Normalize over entire vocabulary to give probability distribution

$$L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

To train the model: Optimize word vectors of all words to minimize loss using backprop

Recall: θ represents all the model parameters, in one long vector

- d-dimensional vectors
- V - many words,
- every word has two vectors

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

Use stochastic gradient descent with batches of N center words

Why did this algorithm use two vectors?

We will use two vectors per word *w*:

- v_w when *w* is a center word
- u_w when *w* is a context word

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Hint: this paper came out in 2013, 1 year after AlexNet.

Why did this algorithm use two vectors?

We will use two vectors per word *w*:

- v_w when *w* is a center word
- u_w when *w* is a context word

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Hint: this paper came out in 2013, 1 year after AlexNet.

Do you think the results would be better if they used 1 instead of 2 vectors?

What are the final word vectors?

It is the average of the two vectors:

$\frac{1}{2}(v_w + u_w)$

Variants of algorithm: The skip-gram model versus CBOW

Skip-gram predicts context words given center word.

CBOW (continuous bag of words) predicts center word given context words.

What's another issue with this calculation?

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Solution: negative sampling

$$\log P(o|c) = -\log \sigma(\boldsymbol{u}_o^T \boldsymbol{v}_c) - \sum_{k \in \{K\ sampled\ indices\}} \log \sigma(-\boldsymbol{u}_k^T \boldsymbol{v}_c)$$

Replace denominator with randomly sampled k vocabulary instances.
New hyperparameter: k

How should we sample?

- Rare words (aardvark) are unlikely to be helpful.
- You can sample a word *w* based on its probability of occurrence:
    - p(w) = U(w) / Z
    - where Z is total number of words
    - U(w) is the unigram frequency of word w (i.e., number of times it appears in the dataset).

Solution: negative sampling

$$\log P(o|c) = -\log \sigma(\boldsymbol{u}_o^T \boldsymbol{v}_c) - \sum_{k \in \{K \text{ sampled indices}\}} \log \sigma(-\boldsymbol{u}_k^T \boldsymbol{v}_c)$$

Replace denominator with randomly sampled k vocabulary instances.
New hyperparameter: k

How should we sample?

- Rare words (aardvark) are unlikely to be helpful.
- You can sample a word *w* based on its probability of occurrence:
    - p(w) = U(w) / Z
    - where Z is total number of words
    - U(w) is the unigram frequency of word w (i.e., number of times it appears in the dataset).

In practice, $p(w) = U(w)^{3/4} / Z$

Problem of sparse gradients

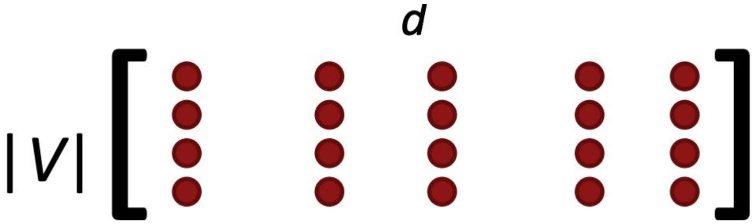
We iteratively take gradients at each *m* window for SGD

- In each window, we only have at most *2m + 1* words
- plus *2km* negative words with negative sampling
- so the gradient for each update is sparse for V of size 500K

$$\nabla_\theta J_t(\theta) = \begin{bmatrix} 0 \\ \vdots \\ \nabla_{v_{like}} \\ \vdots \\ 0 \\ \nabla_{u_I} \\ \vdots \\ \nabla_{u_{learning}} \\ \vdots \end{bmatrix} \in \mathbb{R}^{2dV}$$

Implementation detail

Most DL packages represent word vectors using a special embedding layer



Rows represent words

- even though we usually talk about words as column vectors
- In implementations, they are row vectors.
- It is a hash table with look up and write functions to avoid writing to the entire V x d matrix.

Indirectly, skim-gram is trying to calculate co-occurance of words

It does this using backprop and by iterating through the entire corpus of text data multiple times.

Can we do better?

Can we build a co-occurrence matrix directly?

- Calculate co-occurrence of words within a window.
- captures some syntactic and semantic information ("word space")
- If window is too large (size of entire articles or documents):
    - Co-occurrence matrix will represent general topics
    - Example, all sports words will have similar entries

Example co-occurrence matrix

- Let’s try an example with window length 1 (it is more common to use 5–10)
- Symmetric (irrelevant whether left or right context)

Example corpus:

- I like deep learning
- I like UW
- I enjoy class

counts	I	like	enjoy	deep	learning	UW	class	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
UW	0	1	0	0	0	0	0	1
class	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Co-occurrence vectors

Problem with simple count co-occurrence vectors

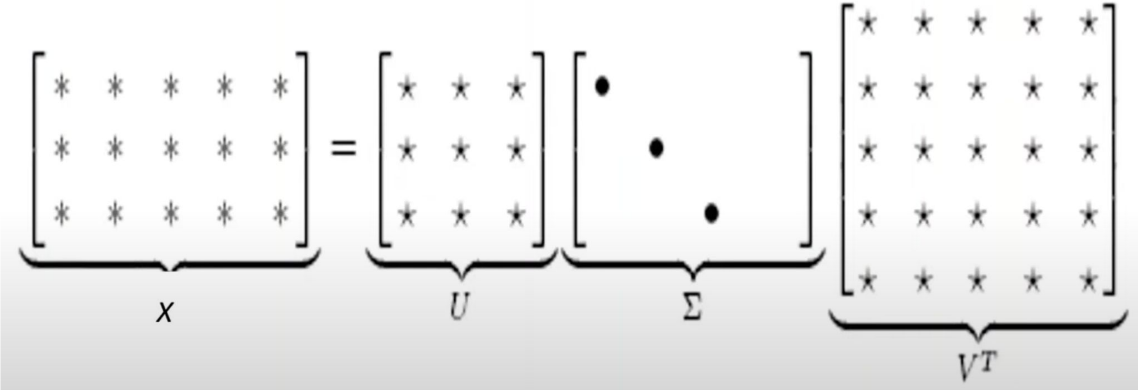
- Vectors increase in size with vocabulary
- Very high dimensional: require a lot of storage (though sparse)

Idea: Low-dimensional vectors

- Idea: store “most” of the important information in a fixed, small number of dimensions: a dense vector
- Usually 25–1000 dimensions, similar to word2vec
- **But how should we reduce the dimensionality from 500K to <1000?**

Classic Method: Dimensionality Reduction

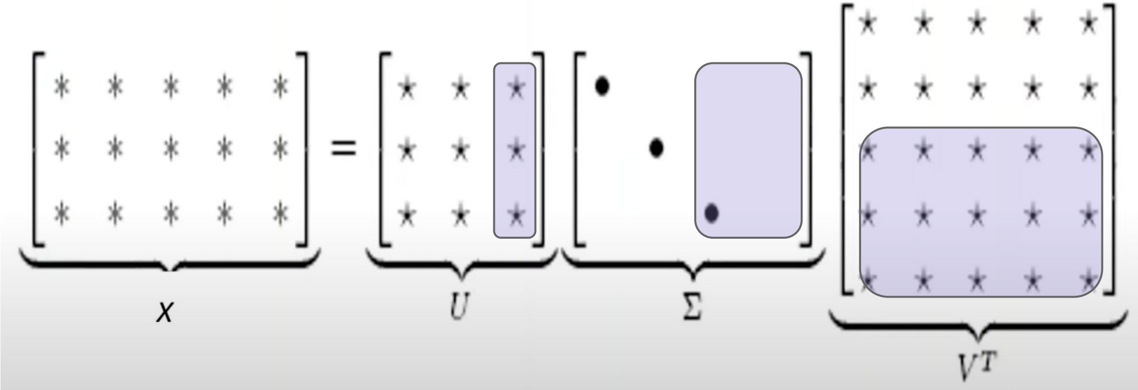
From linear algebra: Singular Value Decomposition of co-occurrence matrix X



U and V are orthogonal matrices
Σ is a diagonal matrix of singular values.

Classic Method: Dimensionality Reduction

From linear algebra: Singular Value Decomposition of co-occurrence matrix X



We can discard all except the largest d singular values and their corresponding multiplicative values in U and V

New d-dimensional word vector representations are: top-d(U) * top-d(Σ)

Making co-occurrence counts work

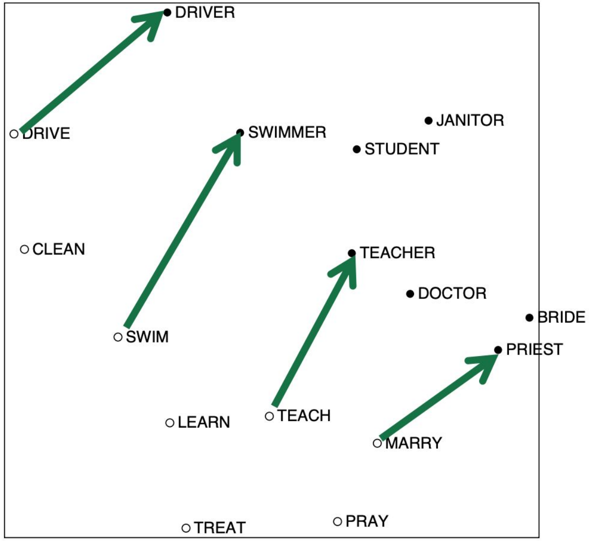
Running an SVD on raw counts doesn't work well.

Problem: function words (the, he, has) are too frequent à syntax has too much impact.

Some fixes:

- Use log the frequencies instead
- Limit the maximum values: min(X, t), with t ≈ 100
- Ignore the function words
- Ramped windows that count closer words more than further away words

Interesting semantic patterns emerge



COALS model from Rohde et al. ms., 2005. An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence

Comparing the two methods:

Co-occurrence vectors

- Fast training
  - Single iteration over data
  - SVD is fast as long as vocabulary is reasonable.
- Good for capturing word similarities
- Needs hacks to work
- Not good for anything beyond similarities
- SVD is very slow for large vocabularies

Skip-gram algorithm

- Scales well with increasing vocabulary size
- Good for many other tasks
- Not as good for word similarities
- Needs multiple iterations across the dataset as backprop is slow

Can we combine the strengths of both methods?

Log bilinear model:

- Let every word be a d-dimensional vector
- Remember from skip-gram that dot product is the probability of one word given its context

$$w_i \cdot w_j = \log P(i|j)$$

GloVe [Pennington, Socher, and Manning, EMNLP 2014]: Encoding meaning components in vector differences

Can we combine the strengths of both methods?

Log bilinear model:

- Let every word be a d-dimensional vector
- Remember from skip-gram that dot product is the probability of one word given its context

$$w_i \cdot w_j = \log P(i|j)$$

Main idea: Similarity between two words should be proportional to their co-occurrence count.

- Log of count used as a hack

$$L(\theta) = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

GloVe [Pennington, Socher, and Manning, EMNLP 2014]: Encoding meaning components in vector differences

Can we combine the strengths of both methods?

Log bilinear model:
- Let every word be a d-dimensional vector
- Remember from skip-gram that dot product is the probability of one word given its context $w_i \cdot w_j = \log P(i|j)$

Main idea: Similarity between two words should be proportional to their co-occurrence count.
- Log of count used as a hack

$$L(\theta) = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

GloVe [Pennington, Socher, and Manning, EMNLP 2014]: Encoding meaning components in vector differences

Can we combine the strengths of both methods?

Log bilinear model:

- Let every word be a d-dimensional vector
- Remember from skip-gram that dot product is the probability of one word given its context

$$w_i \cdot w_j = \log P(i|j)$$

Main idea: Similarity between two words should be proportional to their co-occurrence count.

- Log of count used as a hack

$$L(\theta) = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

GloVe [Pennington, Socher, and Manning, EMNLP 2014]: Encoding meaning components in vector differences

Can we combine the strengths of both methods?

Log bilinear model:

- Let every word be a d-dimensional vector
- Remember from skip-gram that dot product is the probability of one word given its context

$$w_i \cdot w_j = \log P(i|j)$$

Main idea: Similarity between two words should be proportional to their co-occurrence count.

- Log of count used as a hack
- f() is a threshold for large values

$$L(\theta) = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2$$

GloVe [Pennington, Socher, and Manning, EMNLP 2014]: Encoding meaning components in vector differences

Word vectors are very good at analogies

a:b :: c:?

man:woman :: king:?

Word vectors are very good at analogies

a:b :: c:?

man:woman :: king:?

$$d = \arg\max_i \frac{(x_b - x_a + x_c)^T x_i}{||x_b - x_a + x_c||}$$

(woman - man + king)* w_i

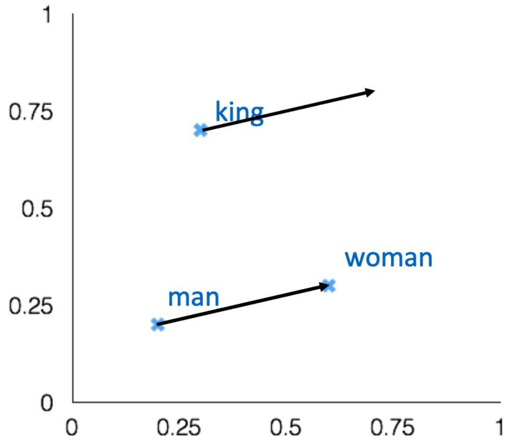
Word vectors are very good at analogies

a:b :: c:?

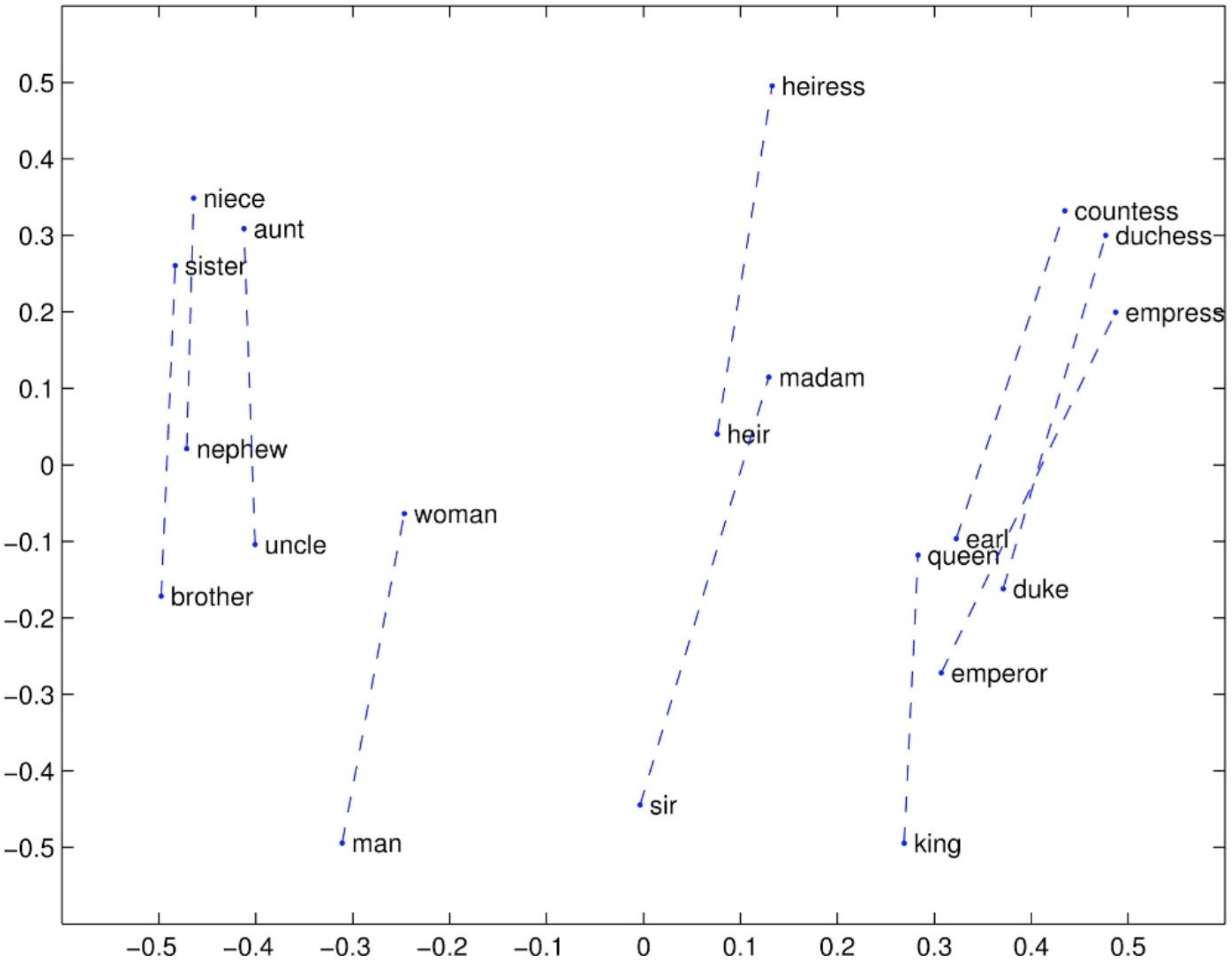
man:woman :: king:?

$$d = \arg\max_i \frac{(x_b - x_a + x_c)^T x_i}{||x_b - x_a + x_c||}$$

(woman - man + king)* w_i



More visualizations



Word vectors correlate with human judgement

Linguists have created a dataset of word similarity judgements

- Word vector distances and their correlation with human judgments
- Example dataset: [WordSim353]

Word 1	Word 2	Human (mean)
tiger	cat	7.35
tiger	tiger	10
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

Problem of **polysemy**

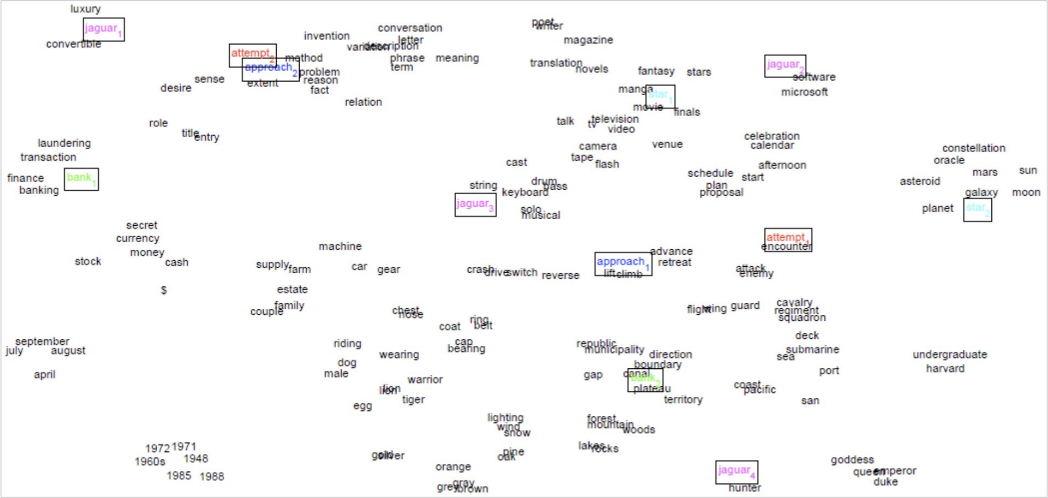
Word senses and word sense ambiguity

Cap

- Using to only be similar to *Hat*
- But now, thanks to gen z, should also be closer to *Lying*

Can one vector capture all these meanings? Probably not!

Researchers have tried to segregate words into multiple vectors, each with its own meaning



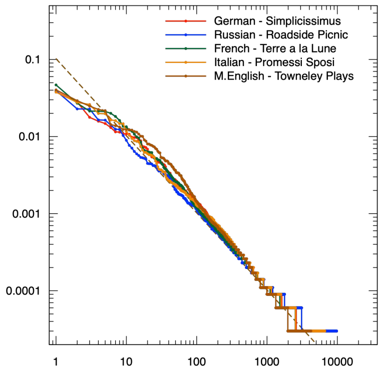
But it doesn’t work well. A word’s usage in a sentence defines its meaning.

Words should be a function of not just its context but its position in the sentence

-> **Future lecture**

So far: vectors are associated with words

Our vocabulary was comprised of all of the **words in a language**
Problems:
- 500,000 words Webster's English Dictionary (3rd edition)
- Language is changing all of the time
  - 690 words were added to Merriam Webster's in September 2023 ("rizz", "goated", "mid")
- Long tail of infrequent words.
  - Zipf's law: word frequency is inversely proportional to word rank
- Some words may not appear in a training set of documents
- No modeled relationship between words - e.g., "run", "ran", "runs", "runner" are all separate entries despite being linked in meaning



Zipf's Law: Word Rank vs. Word Frequency for Several Languages

Character level vectors instead?

What about assigning a vector to every character instead?

(Maybe add capital letters, punctuation, spaces, …)
Pros:
- Small vocabulary size (for English)
- Complete coverage (unseen words are represented by letters)

Cons:
- Encoding a single sentence becomes very long!
  - # chars instead of # words
- Characters mean very different things in different words!
  - Even worse for representing multiple meanings

Subword tokenization!

How can we combine
1. the **high coverage** of character-level representations
2. with the efficiency of word-level representation?

Subword tokenization! (e.g., **Byte-Pair Encoding**)
- Start with character-level representations
- Build up representations from there

Original BPE Paper (Sennrich et al., 2016)

Example of how Byte-pair encoding works

Let's say our entire dataset contains only these 3 sentences:

$$\mathcal{D} = \{\text{“i hug pugs”},\ \text{“hugging pugs is fun”},\ \text{“i make puns”}\}$$

Example of how Byte-pair encoding works

Let's say our entire dataset contains only these 3 sentences:

$$\mathcal{D} = \{\text{“i hug pugs”},\ \text{“hugging pugs is fun”},\ \text{“i make puns”}\}$$

Initialize the vocabulary as all the individual characters. Current Vocab:

$$\mathcal{V} = \{\text{‘ ’}, \text{‘a’}, \text{‘e’}, \text{‘f’}, \text{‘g’}, \text{‘h’}, \text{‘i’}, \text{‘k’}, \text{‘m’}, \text{‘n’}, \text{‘p’}, \text{‘s’}, \text{‘u’}\}, |\mathcal{V}| = 13$$

Example of how Byte-pair encoding works

Let's say our entire dataset contains only these 3 sentences:

$$\mathcal{D} = \{\text{“i hug pugs”},\ \text{“hugging pugs is fun”},\ \text{“i make puns”}\}$$

Let's split it up into words by splitting right **before the whitespace:**

$$\mathcal{D} = \{\text{“i”},\ \text{“ hug”},\ \text{“ pugs”},\ \text{“hugging”},\ \text{“ pugs”},\ \text{“ is”},\ \text{“ fun”},\ \text{“i”},\ \text{“ make”},\ \text{“ puns”}\}$$

Example of how Byte-pair encoding works

The vocabulary for reference:

$$\mathcal{V} = \{\text{` '}, \text{`a'}, \text{`e'}, \text{`f'}, \text{`g'}, \text{`h'}, \text{`i'}, \text{`k'}, \text{`m'}, \text{`n'}, \text{`p'}, \text{`s'}, \text{`u'}\}, |\mathcal{V}| = 13$$

Let's split it up into words by splitting right **before the whitespace:**

$$\mathcal{D} = \{\text{``i''}, \text{`` hug''}, \text{`` pugs''}, \text{``hugging''}, \text{`` pugs''}, \text{`` is''}, \text{`` fun''}, \text{``i''}, \text{`` make''}, \text{`` puns''}\}$$

Example of how Byte-pair encoding works

The vocabulary for reference:

$$\mathcal{V} = \{\text{` '}, \text{`a'}, \text{`e'}, \text{`f'}, \text{`g'}, \text{`h'}, \text{`i'}, \text{`k'}, \text{`m'}, \text{`n'}, \text{`p'}, \text{`s'}, \text{`u'}\}, |\mathcal{V}| = 13$$

Let's represent the dataset with only vocabulary elements:

$$\mathcal{D} = \{ [\text{`i'}], [\text{` '}, \text{`h'}, \text{`u'}, \text{`g'}], [\text{` '}, \text{`p'}, \text{`u'}, \text{`g'}, \text{`s'}], [\text{`h'}, \text{`u'}, \text{`g'}, \text{`g'}, \text{`i'}, \text{`n'}, \text{`g'}], [\text{` '}, \text{`p'}, \text{`u'}, \text{`g'}, \text{`s'}], [\text{` '}, \text{`i'}, \text{`s'}], [\text{` '}, \text{`f'}, \text{`u'}, \text{`n'}], [\text{`i'}], [\text{` '}, \text{`m'}, \text{`a'}, \text{`k'}, \text{`e'}], [\text{` '}, \text{`p'}, \text{`u'}, \text{`n'}, \text{`s'}]\}$$

Example of how Byte-pair encoding works

The vocabulary for reference:

$$\mathcal{V} = \{\text{` '}, \text{`a'}, \text{`e'}, \text{`f'}, \text{`g'}, \text{`h'}, \text{`i'}, \text{`k'}, \text{`m'}, \text{`n'}, \text{`p'}, \text{`s'}, \text{`u'}\}, |\mathcal{V}| = 13$$

Now, let's find the most common bi-gram

$$\mathcal{D} = \{ [\text{`i'}], [\text{` '}, \text{`h'}, \text{`u'}, \text{`g'}], [\text{` '}, \text{`p'}, \text{`u'}, \text{`g'}, \text{`s'}], [\text{`h'}, \text{`u'}, \text{`g'}, \text{`g'}, \text{`i'}, \text{`n'}, \text{`g'}], [\text{` '}, \text{`p'}, \text{`u'}, \text{`g'}, \text{`s'}], [\text{` '}, \text{`i'}, \text{`s'}], [\text{` '}, \text{`f'}, \text{`u'}, \text{`n'}], [\text{`i'}], [\text{` '}, \text{`m'}, \text{`a'}, \text{`k'}, \text{`e'}], [\text{` '}, \text{`p'}, \text{`u'}, \text{`n'}, \text{`s'}]\}$$

Example of how Byte-pair encoding works

The vocabulary for reference:

$$\mathcal{V} = \{\text{` '}, \text{`a'}, \text{`e'}, \text{`f'}, \text{`g'}, \text{`h'}, \text{`i'}, \text{`k'}, \text{`m'}, \text{`n'}, \text{`p'}, \text{`s'}, \text{`u'}\}, |\mathcal{V}| = 13$$

Now, let's find the most common bi-gram

$$\mathcal{D} = \{ [\text{`i'}], [\text{` '}, \text{`h'}, \text{`u'}, \text{`g'}], [\text{` '}, \text{`p'}, \text{`u'}, \text{`g'}, \text{`s'}], [\text{`h'}, \text{`u'}, \text{`g'}, \text{`g'}, \text{`i'}, \text{`n'}, \text{`g'}], [\text{` '}, \text{`p'}, \text{`u'}, \text{`g'}, \text{`s'}], [\text{` '}, \text{`i'}, \text{`s'}], [\text{` '}, \text{`f'}, \text{`u'}, \text{`n'}], [\text{`i'}], [\text{` '}, \text{`m'}, \text{`a'}, \text{`k'}, \text{`e'}], [\text{` '}, \text{`p'}, \text{`u'}, \text{`n'}, \text{`s'}]\}$$

Create new vocab:

$$v_{14} := \text{concat}(\text{`u'}, \text{`g'}) = \text{`ug'}$$

Example of how Byte-pair encoding works

Update vocabulary with new vocab v_{14}:

$$\mathcal{V} = \{\text{` '}, \text{`a'}, \text{`e'}, \text{`f'}, \text{`g'}, \text{`h'}, \text{`i'}, \text{`k'}, \text{`m'}, \text{`n'}, \text{`p'}, \text{`s'}, \text{`u'}, \text{`ug'}\}, |\mathcal{V}| = 14$$

Update dataset by replace bigram with new vocab v_{14}:

Create new vocab:

$$v_{14} := \text{concat}(\text{`u'}, \text{`g'}) = \text{`ug'}$$

$$\mathcal{D} = \{ [\text{`i'}], [\text{` '}, \text{`h'}, \text{`ug'}], [\text{` '}, \text{`p'}, \text{`ug'}, \text{`s'}], [\text{`h'}, \text{`ug'}, \text{`g'}, \text{`i'}, \text{`n'}, \text{`g'}], [\text{` '}, \text{`p'}, \text{`ug'}, \text{`s'}], [\text{` '}, \text{`i'}, \text{`s'}], [\text{` '}, \text{`f'}, \text{`u'}, \text{`n'}], [\text{`i'}], [\text{` '}, \text{`m'}, \text{`a'}, \text{`k'}, \text{`e'}], [\text{` '}, \text{`p'}, \text{`u'}, \text{`n'}, \text{`s'}] \}$$

Example of how Byte-pair encoding works

Current vocabulary:

$$\mathcal{V} = \{\text{` '}, \text{`a'}, \text{`e'}, \text{`f'}, \text{`g'}, \text{`h'}, \text{`i'}, \text{`k'}, \text{`m'}, \text{`n'}, \text{`p'}, \text{`s'}, \text{`u'}, \text{`ug'}\}, |\mathcal{V}| = 14$$

Find the next common bigram:

$$\mathcal{D} = \{ [\text{`i'}], [\text{` '}, \text{`h'}, \text{`ug'}], [\text{` '}, \text{`p'}, \text{`ug'}, \text{`s'}], [\text{`h'}, \text{`ug'}, \text{`g'}, \text{`i'}, \text{`n'}, \text{`g'}], [\text{` '}, \text{`p'}, \text{`ug'}, \text{`s'}], [\text{` '}, \text{`i'}, \text{`s'}], [\text{` '}, \text{`f'}, \text{`u'}, \text{`n'}], [\text{`i'}], [\text{` '}, \text{`m'}, \text{`a'}, \text{`k'}, \text{`e'}], [\text{` '}, \text{`p'}, \text{`u'}, \text{`n'}, \text{`s'}]\}$$

Example of how Byte-pair encoding works

Current vocabulary:

$$\mathcal{V} = \{\text{` '}, \text{`a'}, \text{`e'}, \text{`f'}, \text{`g'}, \text{`h'}, \text{`i'}, \text{`k'}, \text{`m'}, \text{`n'}, \text{`p'}, \text{`s'}, \text{`u'}, \text{`ug'}\}, |\mathcal{V}| = 14$$

Find the next common bigram:

$$\mathcal{D} = \{ [\text{`i'}], [\text{` '}, \text{`h'}, \text{`ug'}], [\text{` '}, \text{`p'}, \text{`ug'}, \text{`s'}], [\text{`h'}, \text{`ug'}, \text{`g'}, \text{`i'}, \text{`n'}, \text{`g'}], [\text{` '}, \text{`p'}, \text{`ug'}, \text{`s'}], [\text{` '}, \text{`i'}, \text{`s'}], [\text{` '}, \text{`f'}, \text{`u'}, \text{`n'}], [\text{`i'}], [\text{` '}, \text{`m'}, \text{`a'}, \text{`k'}, \text{`e'}], [\text{` '}, \text{`p'}, \text{`u'}, \text{`n'}, \text{`s'}]\}$$

Create new vocab:

$$v_{15} := \text{concat}(\text{` '}, \text{`p'}) = \text{` p'}$$

Example of how Byte-pair encoding works

Update vocabulary with new vocab v_{15}:

$$\mathcal{V} = \{\text{` '}, \text{`a'}, \text{`e'}, \text{`f'}, \text{`g'}, \text{`h'}, \text{`i'}, \text{`k'}, \text{`m'}, \text{`n'}, \text{`p'}, \text{`s'}, \text{`u'}, \text{`ug'}, \text{` p'}\}, |\mathcal{V}| = 15$$

Update dataset by replace bigram with new vocab v_{15}:

Create new vocab:

$$v_{15} := \text{concat}(\text{` '}, \text{`p'}) = \text{` p'}$$

$$\mathcal{D} = \{ [\text{`i'}], [\text{` '}, \text{`h'}, \text{`ug'}], [\text{` p'}, \text{`ug'}, \text{`s'}], [\text{`h'}, \text{`ug'}, \text{`g'}, \text{`i'}, \text{`n'}, \text{`g'}], [\text{` p'}, \text{`ug'}, \text{`s'}], [\text{` '}, \text{`i'}, \text{`s'}], [\text{` '}, \text{`f'}, \text{`u'}, \text{`n'}], [\text{`i'}], [\text{` '}, \text{`m'}, \text{`a'}, \text{`k'}, \text{`e'}], [\text{` p'}, \text{`u'}, \text{`n'}, \text{`s'}]\}$$

Repeat until vocab size reaches the amount you want (20 for example)

Final vocabulary:

$$\mathcal{V} = \{\text{` '}, \text{`a'}, \text{`e'}, \text{`f'}, \text{`g'}, \text{`h'}, \text{`i'}, \text{`k'}, \text{`m'}, \text{`n'}, \text{`p'}, \text{`s'}, \text{`u'}, \text{`ug'}, \text{` p'}, \text{`hug'}, \text{` pug'}, \text{` pugs'}, \text{`un'}, \text{` hug'}\},$$

Final dataset:

$$\mathcal{D} = \{ [\text{`i'}], [\text{` hug'}], [\text{` pugs'}], [\text{`hug'}, \text{`g'}, \text{`i'}, \text{`n'}, \text{`g'}], [\text{` pugs'}], [\text{` '}, \text{`i'}, \text{`s'}], [\text{` '}, \text{`f'}, \text{`un'}], [\text{`i'}], [\text{` '}, \text{`m'}, \text{`a'}, \text{`k'}, \text{`e'}], [\text{` p'}, \text{`un'}, \text{`s'}]\}$$

With this vocabulary, can you represent (or, tokenize/encode):

Q: Can you encode “apple”?

$$\mathcal{V} = \{1: \text{‘ ’}, 2: \text{‘a’}, 3: \text{‘e’}, 4: \text{‘f’}, 5: \text{‘g’}, 6: \text{‘h’}, 7: \text{‘i’}, 8: \text{‘k’}, 9: \text{‘m’}, 10: \text{‘n’}, 11: \text{‘p’}, 12: \text{‘s’}, 13: \text{‘u’}, 14: \text{‘ug’}, 15: \text{‘ p’}, 16: \text{‘hug’}, 17: \text{‘ pug’}, 18: \text{‘ pugs’}, 19: \text{‘un’}, 20: \text{‘ hug’}\}$$

With this vocabulary, can you represent (or, tokenize/encode):

Q: Can you encode “apple”?

- No, there is no ‘l’ in the vocabulary

$$\mathcal{V} = \{1: \text{` '}, 2: \text{`a'}, 3: \text{`e'}, 4: \text{`f'}, 5: \text{`g'}, 6: \text{`h'}, 7: \text{`i'}, 8: \text{`k'}, 9: \text{`m'}, 10: \text{`n'}, 11: \text{`p'}, 12: \text{`s'}, 13: \text{`u'}, 14: \text{`ug'}, 15: \text{` p'}, 16: \text{`hug'}, 17: \text{` pug'}, 18: \text{` pugs'}, 19: \text{`un'}, 20: \text{` hug'}\}$$

With this vocabulary, can you represent (or, tokenize/encode):

Q: Can you encode “apple”?

- No, there is no ‘l’ in the vocabulary

Q: “map”?

$$\mathcal{V} = \{1: \text{` '}, 2: \text{`a'}, 3: \text{`e'}, 4: \text{`f'}, 5: \text{`g'}, 6: \text{`h'}, 7: \text{`i'}, \\ 8: \text{`k'}, 9: \text{`m'}, 10: \text{`n'}, 11: \text{`p'}, 12: \text{`s'}, 13: \text{`u'}, \\ 14: \text{`ug'}, 15: \text{` p'}, 16: \text{`hug'}, 17: \text{` pug'}, 18: \text{` pugs'}, \\ 19: \text{`un'}, 20: \text{` hug'}\}$$

With this vocabulary, can you represent (or, tokenize/encode):

Q: Can you encode “apple”?
- No, there is no ‘l’ in the vocabulary

Q: “map”?
Yes - [9,2,11]

$$\mathcal{V} = \{1: \text{` '}, 2: \text{`a'}, 3: \text{`e'}, 4: \text{`f'}, 5: \text{`g'}, 6: \text{`h'}, 7: \text{`i'}, 8: \text{`k'}, 9: \text{`m'}, 10: \text{`n'}, 11: \text{`p'}, 12: \text{`s'}, 13: \text{`u'}, 14: \text{`ug'}, 15: \text{` p'}, 16: \text{`hug'}, 17: \text{` pug'}, 18: \text{` pugs'}, 19: \text{`un'}, 20: \text{` hug'}\}$$

With this vocabulary, can you represent (or, tokenize/encode):

Q: Can you encode “apple”?

- No, there is no ‘l’ in the vocabulary

Q: “map”?
Yes - [9,2,11]

Q: “huge”?

$$\mathcal{V} = \{1: \text{' '}, 2: \text{'a'}, 3: \text{'e'}, 4: \text{'f'}, 5: \text{'g'}, 6: \text{'h'}, 7: \text{'i'}, 8: \text{'k'}, 9: \text{'m'}, 10: \text{'n'}, 11: \text{'p'}, 12: \text{'s'}, 13: \text{'u'}, 14: \text{'ug'}, 15: \text{' p'}, 16: \text{'hug'}, 17: \text{' pug'}, 18: \text{' pugs'}, 19: \text{'un'}, 20: \text{' hug'}\}$$

With this vocabulary, can you represent (or, tokenize/encode):

Q: Can you encode “apple”?

- No, there is no ‘l’ in the vocabulary

$$\mathcal{V} = \{1 : \text{‘ ’}, 2 : \text{‘a’}, 3 : \text{‘e’}, 4 : \text{‘f’}, 5 : \text{‘g’}, 6 : \text{‘h’}, 7 : \text{‘i’}, 8 : \text{‘k’}, 9 : \text{‘m’}, 10 : \text{‘n’}, 11 : \text{‘p’}, 12 : \text{‘s’}, 13 : \text{‘u’}, 14 : \text{‘ug’}, 15 : \text{‘ p’}, 16 : \text{‘hug’}, 17 : \text{‘ pug’}, 18 : \text{‘ pugs’}, 19 : \text{‘un’}, 20 : \text{‘ hug’}\}$$

Q: “map”?
Yes - [9,2,11]

Q: “huge”?
Yes - [16, 3] or [6,14,3] or [6,13,5,3]

With this vocabulary, can you represent (or, tokenize/encode):

Q: Can you encode “apple”?

- No, there is no ‘l’ in the vocabulary

$$\mathcal{V} = \{1 : \text{‘ ’}, 2 : \text{‘a’}, 3 : \text{‘e’}, 4 : \text{‘f’}, 5 : \text{‘g’}, 6 : \text{‘h’}, 7 : \text{‘i’}, 8 : \text{‘k’}, 9 : \text{‘m’}, 10 : \text{‘n’}, 11 : \text{‘p’}, 12 : \text{‘s’}, 13 : \text{‘u’}, 14 : \text{‘ug’}, 15 : \text{‘ p’}, 16 : \text{‘hug’}, 17 : \text{‘ pug’}, 18 : \text{‘ pugs’}, 19 : \text{‘un’}, 20 : \text{‘ hug’}\}$$

Q: “map”?
Yes - [9,2,11]

Q: “huge”?
Yes - [16, 3] or [6,14,3] or [6,13,5,3]

Q: “ huge” with a space in the front?

With this vocabulary, can you represent (or, tokenize/encode):

Q: Can you encode “apple”?

- No, there is no ‘l’ in the vocabulary

$$\mathcal{V} = \{1: \text{` '}, 2: \text{`a'}, 3: \text{`e'}, 4: \text{`f'}, 5: \text{`g'}, 6: \text{`h'}, 7: \text{`i'}, 8: \text{`k'}, 9: \text{`m'}, 10: \text{`n'}, 11: \text{`p'}, 12: \text{`s'}, 13: \text{`u'}, 14: \text{`ug'}, 15: \text{` p'}, 16: \text{`hug'}, 17: \text{` pug'}, 18: \text{` pugs'}, 19: \text{`un'}, 20: \text{` hug'}\}$$

Q: “map”?
Yes - [9,2,11]

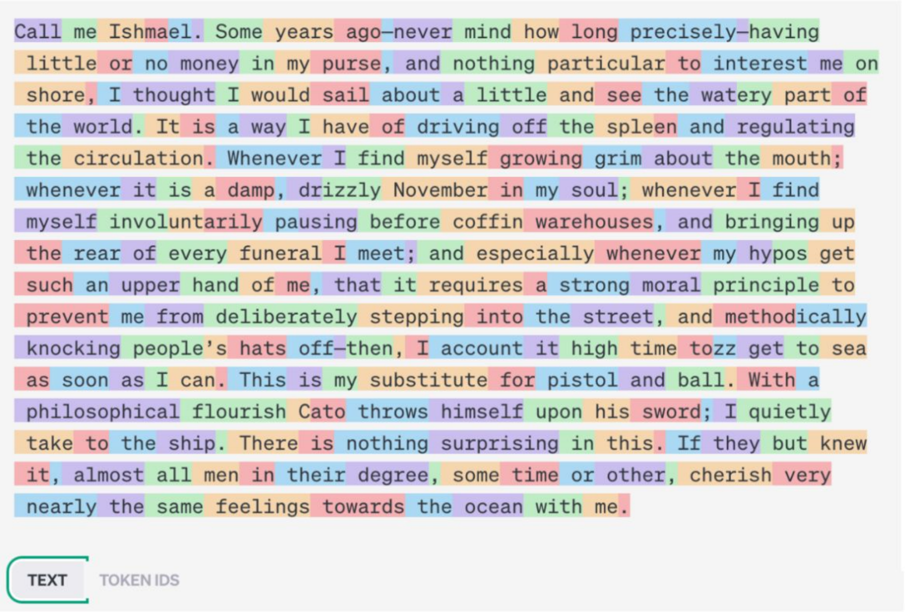
Q: “huge”?
Yes - [16, 3] or [6,14,3] or [6,13,5,3]

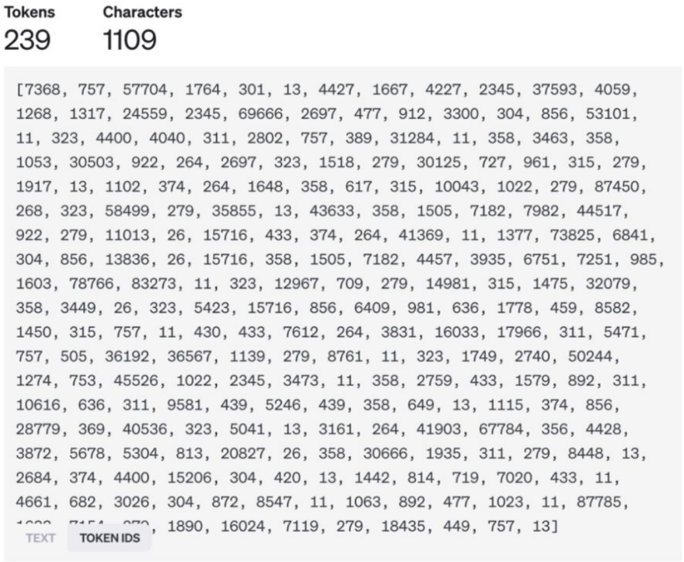
Q: “ huge” with a space in the front?
Yes - [20, 3]

Benefits of Byte-pair encoding

1. Efficient to run (greedy vs. global optimization)
2. Lossless compression
3. Potentially some shared representations
    a. e.g., the token “hug” could be used both in “hug” and “hugging”

Byte-pair encoding - ChatGPT Example

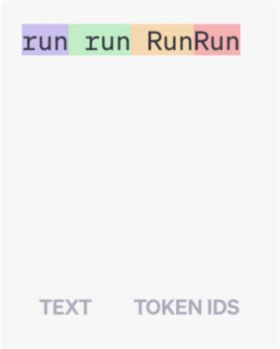


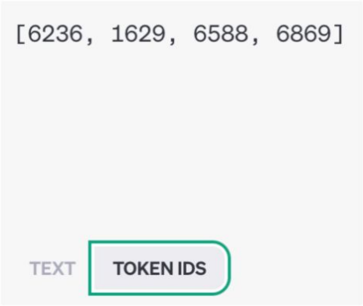


Weird properties of tokenizers

Token != word

- Spaces are part of token
- "run" is a different token than " run"
- Not invariant to case changes
- "Run" is a different token than "run"





Weird properties of tokenizers

Token != word

- Spaces are part of token
- “run” is a different token than “ run”
- Not invariant to case changes
- “Run” is a different token than “run”
- Tokenization fits statistics of your data
  - e.g., while these words are multiple tokens…

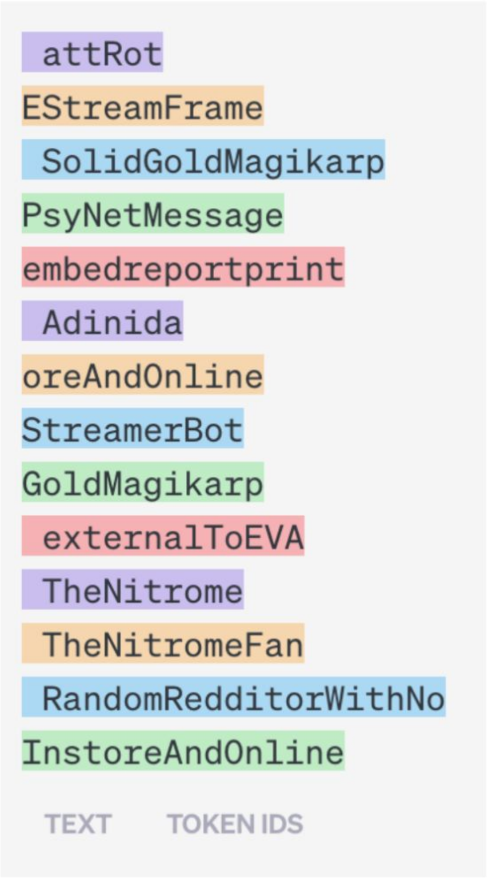
Weird properties of tokenizers

Token != word

- Spaces are part of token
- “run” is a different token than “ run”
- Not invariant to case changes
- “Run” is a different token than “run”
- Tokenization fits statistics of your data
  - e.g., while these words are multiple tokens…

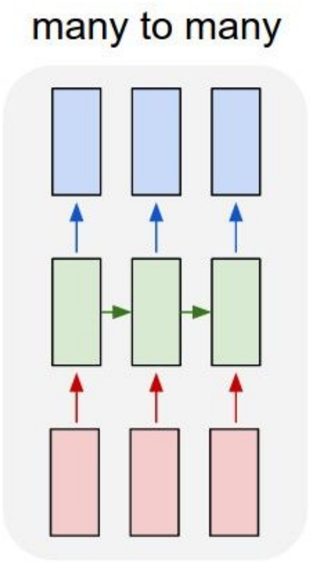
These words are all 1 token in GPT-3’s tokenizer!

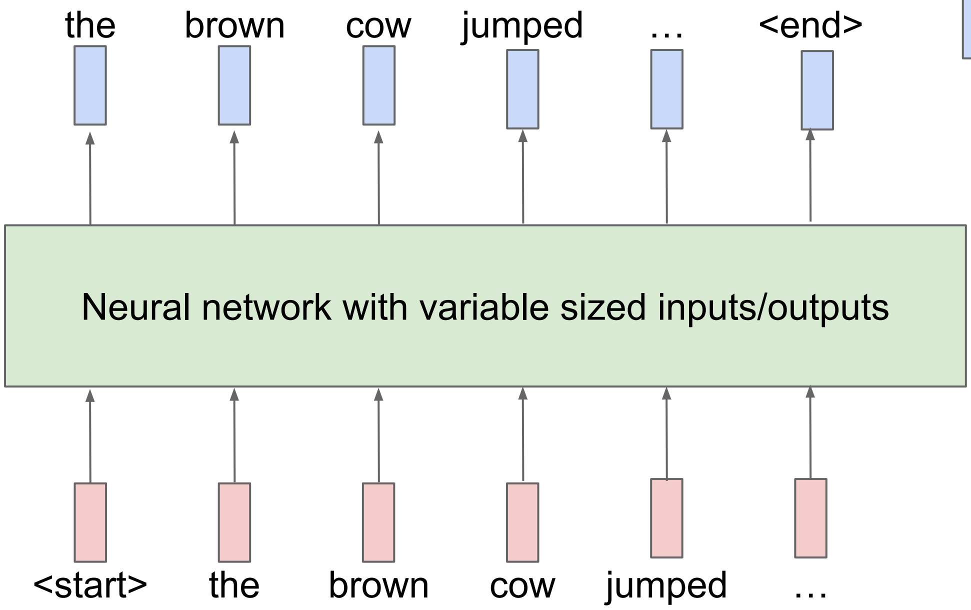
Does anyone know why?



RNNs & LSTMs

Now let's talk about how we can model language

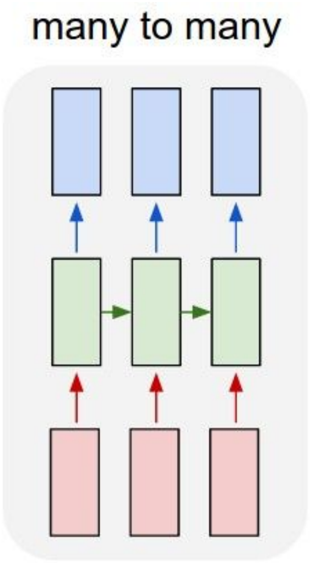




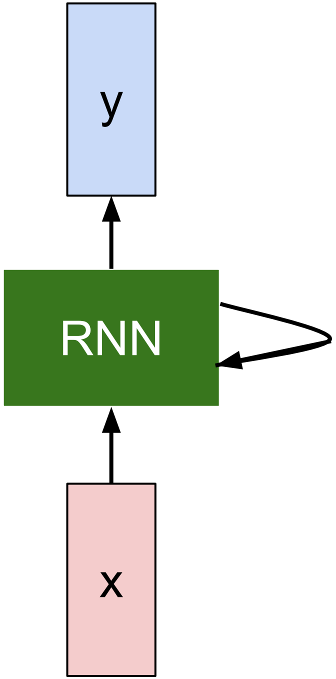
Word representations can be V-dimensional **one-hot** vectors
Or d-dimensional **dense** vectors

Outputs are classification (softmax) over V-dimensions

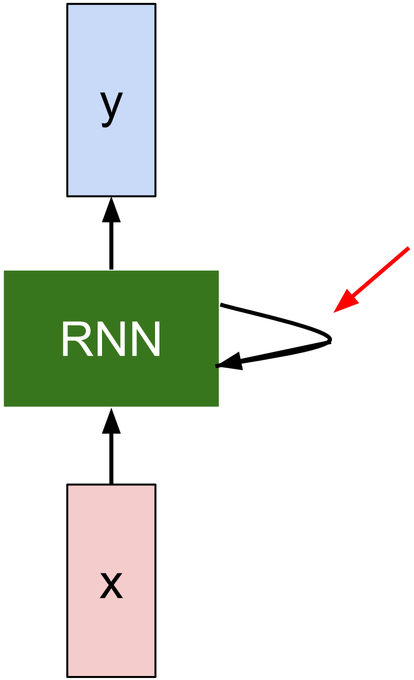
Now let's talk about how we can model language



Recurrent Neural Network

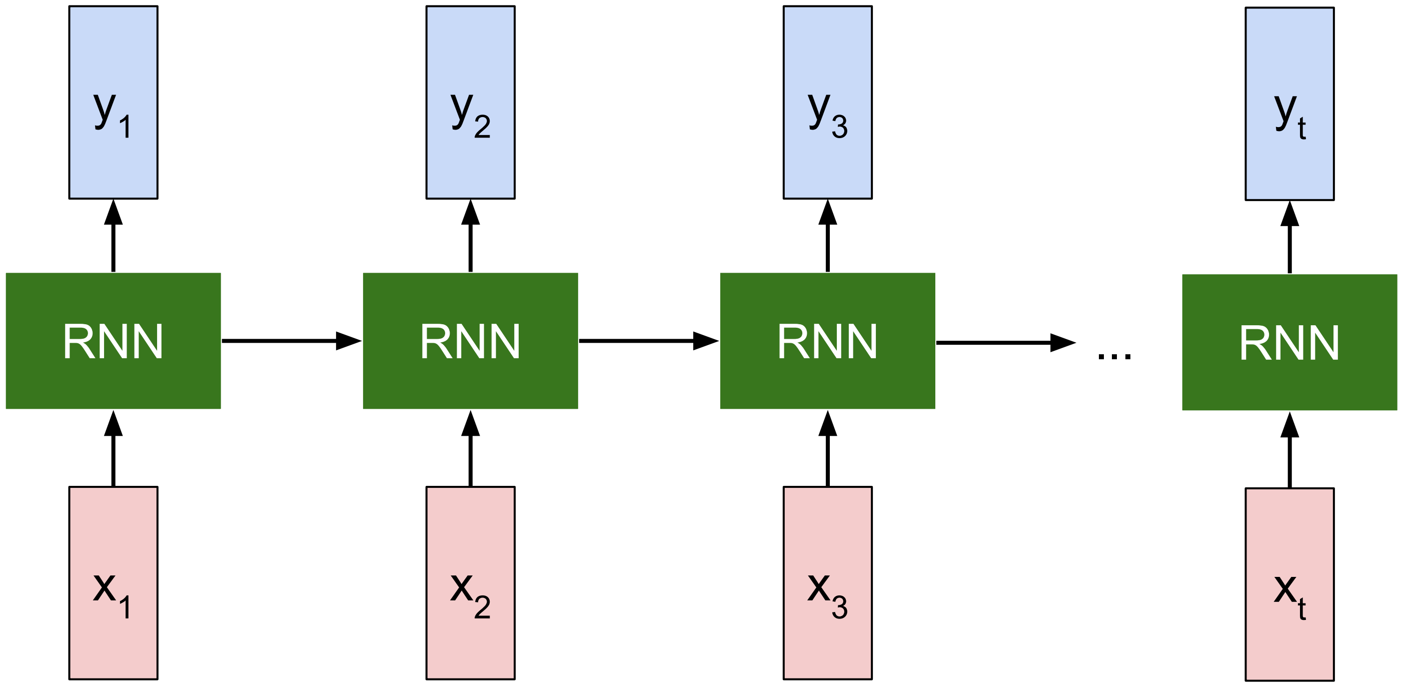


Recurrent Neural Network



Key idea: RNNs have an "internal state" that is updated as a sequence is processed

Unrolled RNN



RNN hidden state update

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

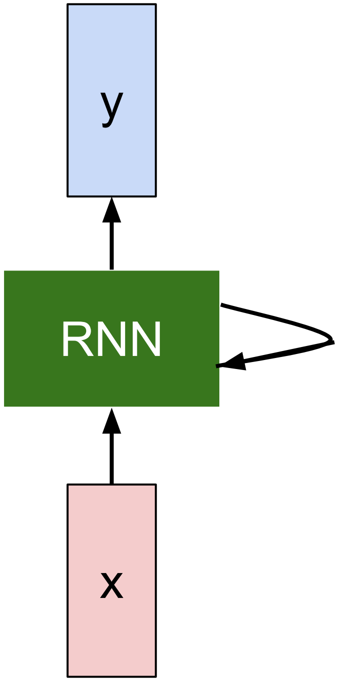
$$h_t = f_W(h_{t-1}, x_t)$$

new state

some function with parameters W

old state

input vector at some time step



RNN output generation

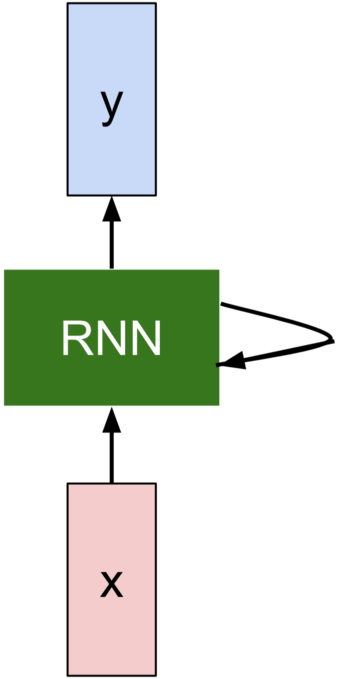
We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

$$y_t = f_{W_{hy}}(h_t)$$

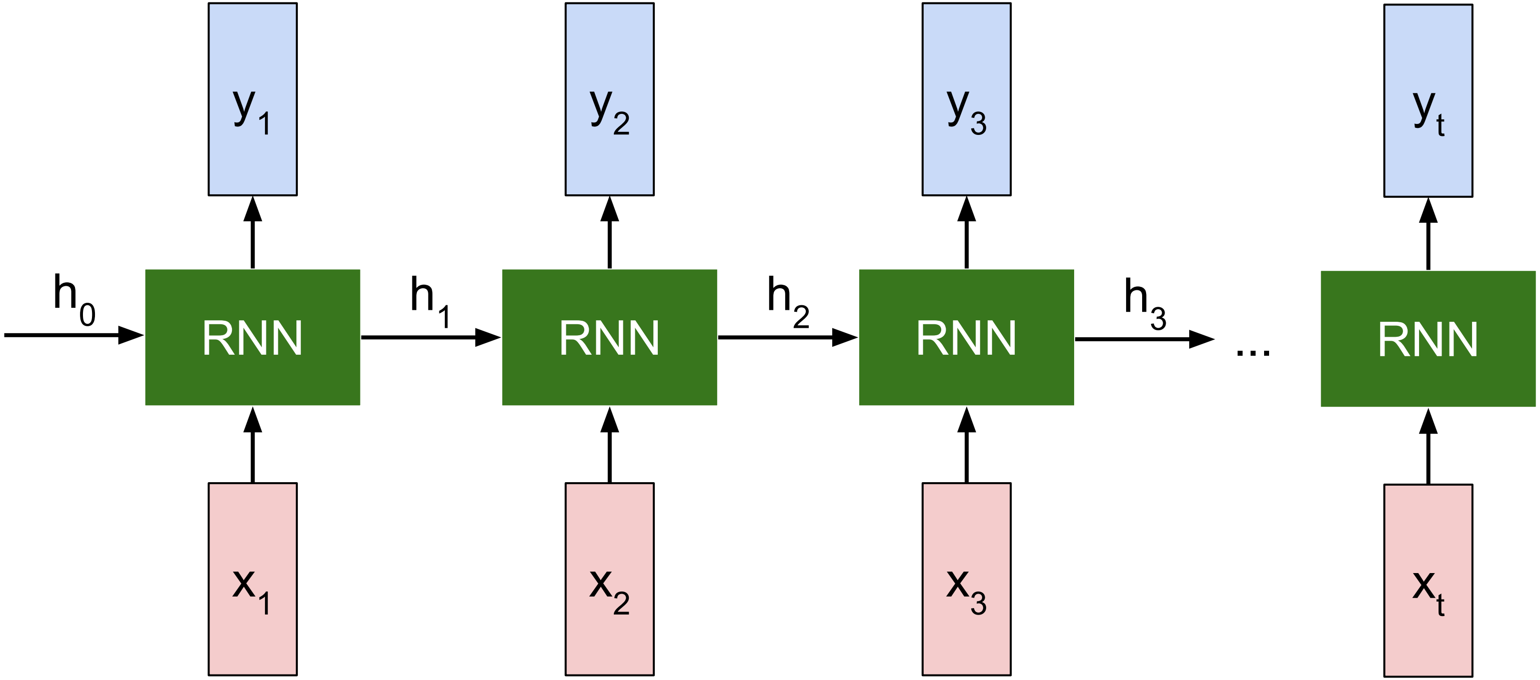
output

new state

another function with parameters W_o



Recurrent Neural Network

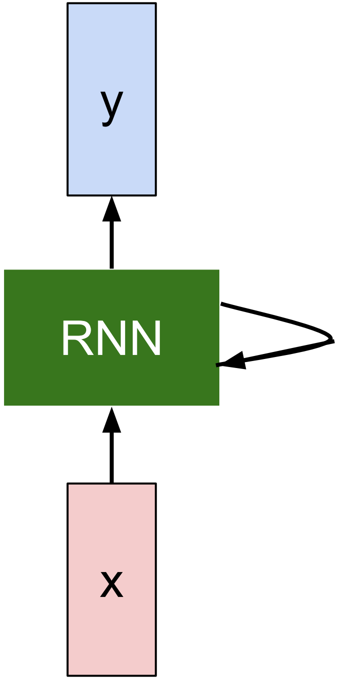


Recurrent Neural Network

We can process a sequence of vectors **x** by applying a **recurrence formula** at every time step:

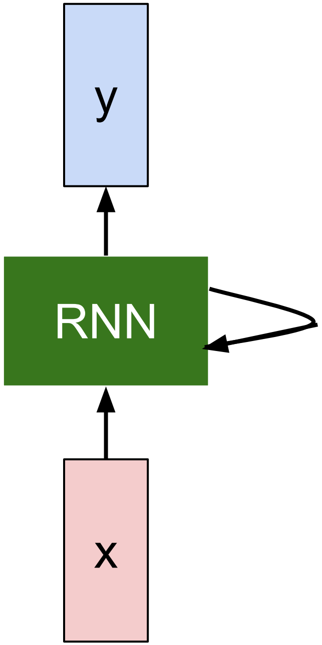
$$h_t = f_W(h_{t-1}, x_t)$$

Notice: the same function and the same set of parameters are used at every time step.



(Simple) Recurrent Neural Network

The state consists of a single *"hidden"* vector **h**:



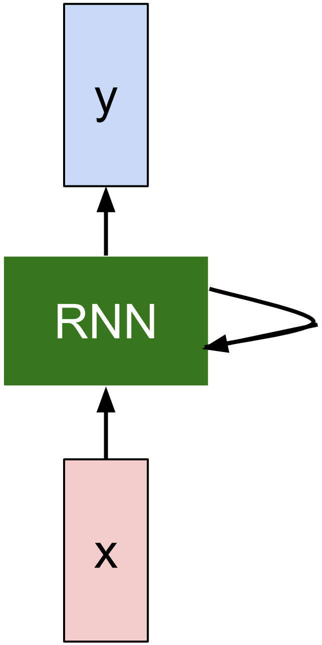
$$h_t = f_W(h_{t-1}, x_t)$$

$$y_t = f_{W_{hy}}(h_t)$$

Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

(Simple) Recurrent Neural Network

The state consists of a single *"hidden"* vector **h**:



$$h_t = f_W(h_{t-1}, x_t)$$

$$y_t = f_{W_{hy}}(h_t)$$

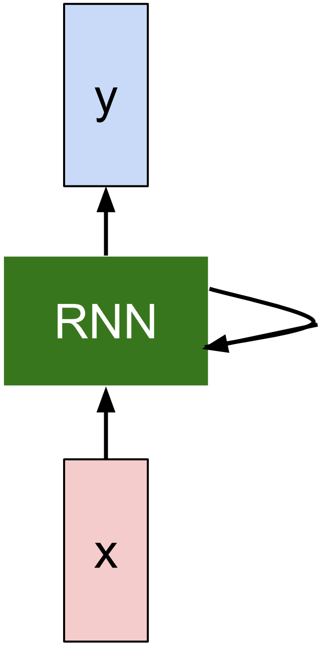
↓

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

(Simple) Recurrent Neural Network

The state consists of a single *"hidden"* vector **h**:



$$h_t = f_W(h_{t-1}, x_t)$$

$$y_t = f_{W_{hy}}(h_t)$$

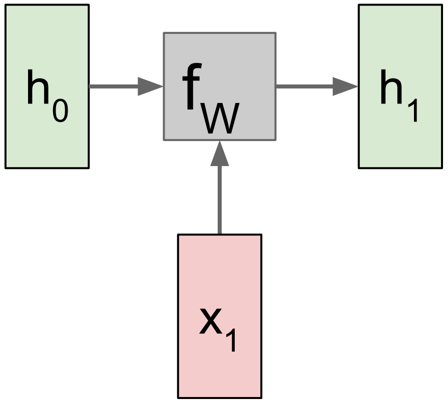
↓

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

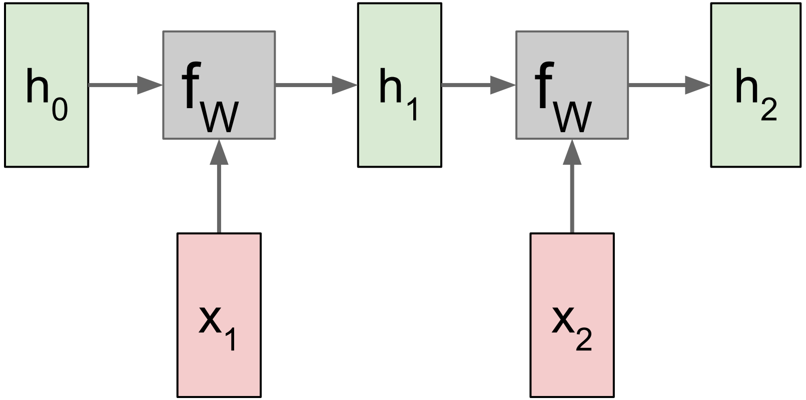
$$y_t = W_{hy}h_t$$

Sometimes called a "Vanilla RNN" or an "Elman RNN" after Prof. Jeffrey Elman

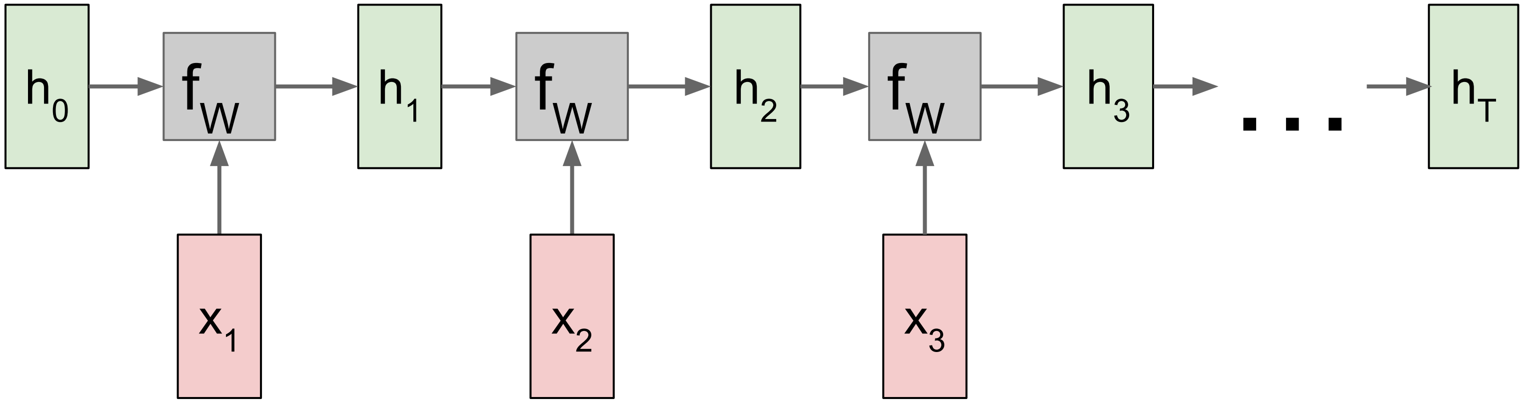
RNN: Computational Graph



RNN: Computational Graph

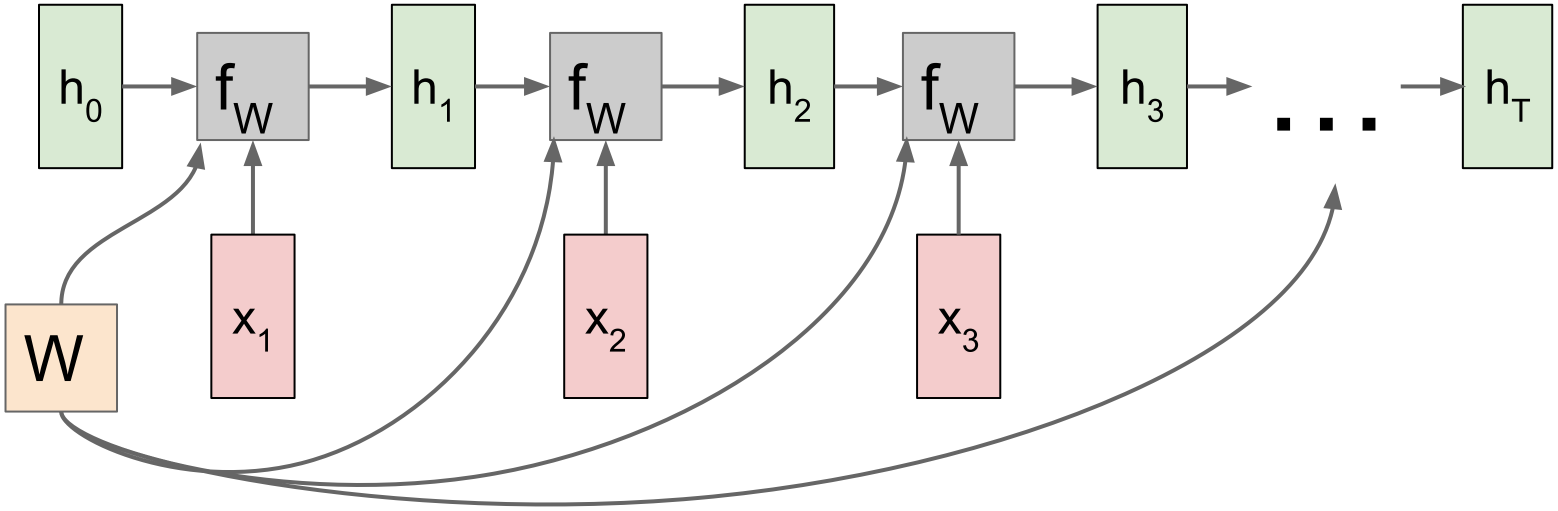


RNN: Computational Graph

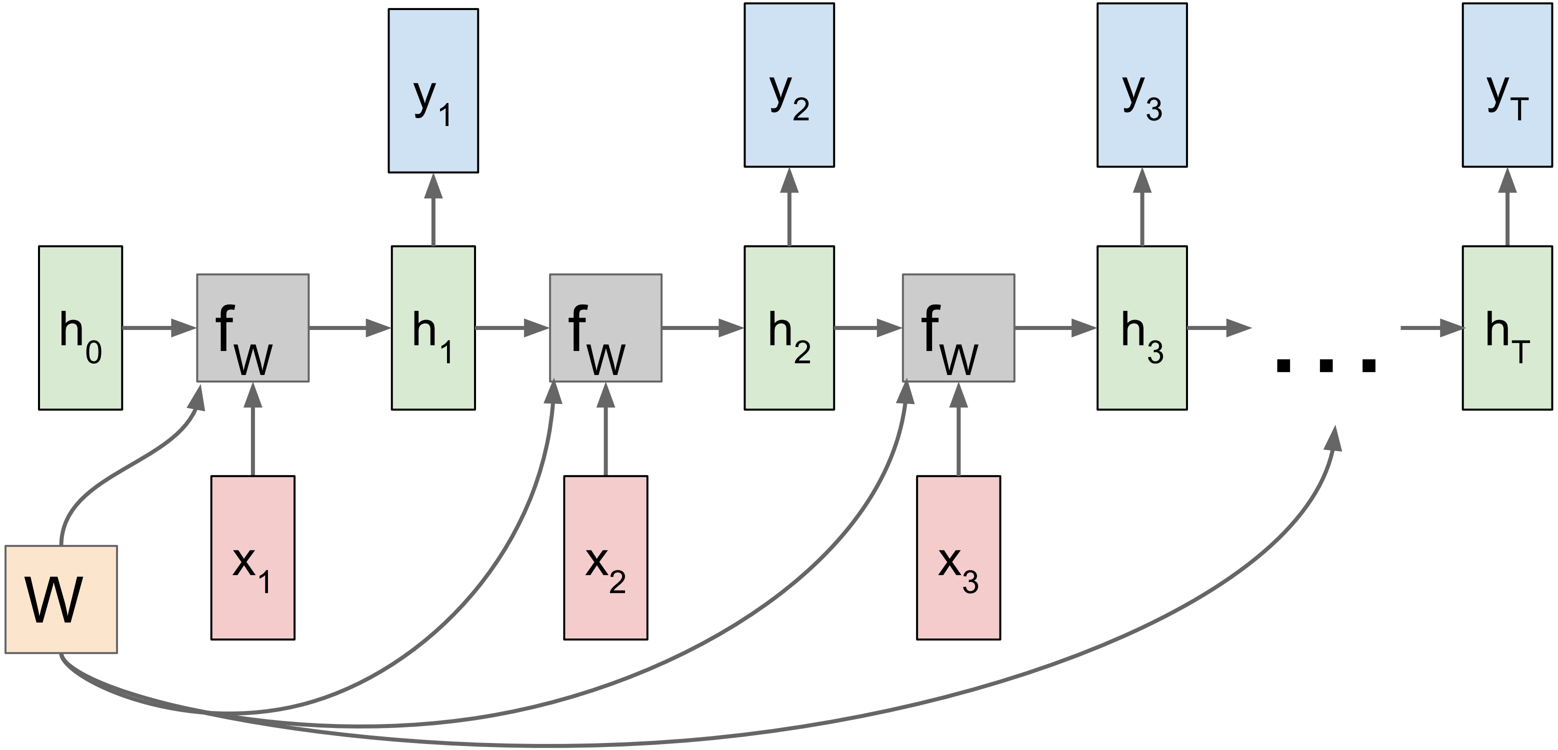


RNN: Computational Graph

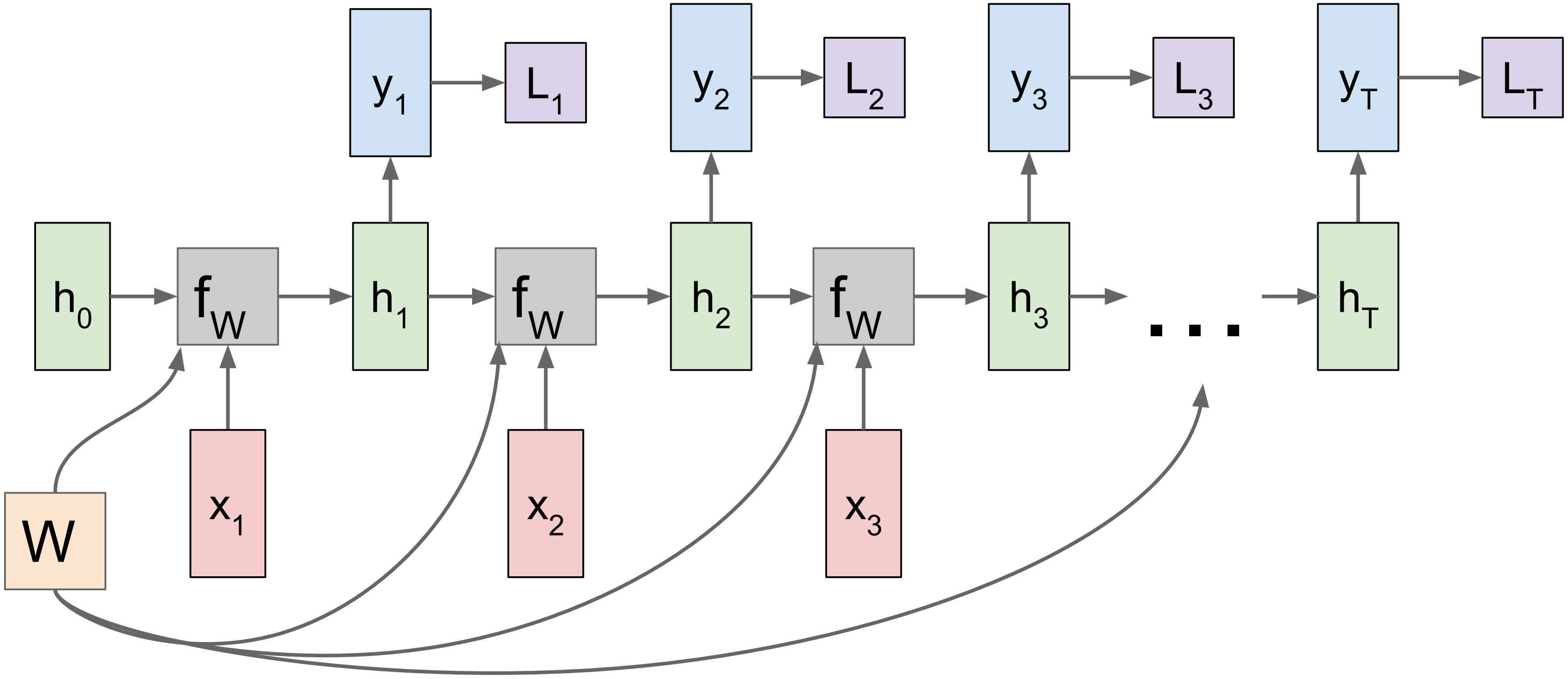
Re-use the same weight matrix at every time-step



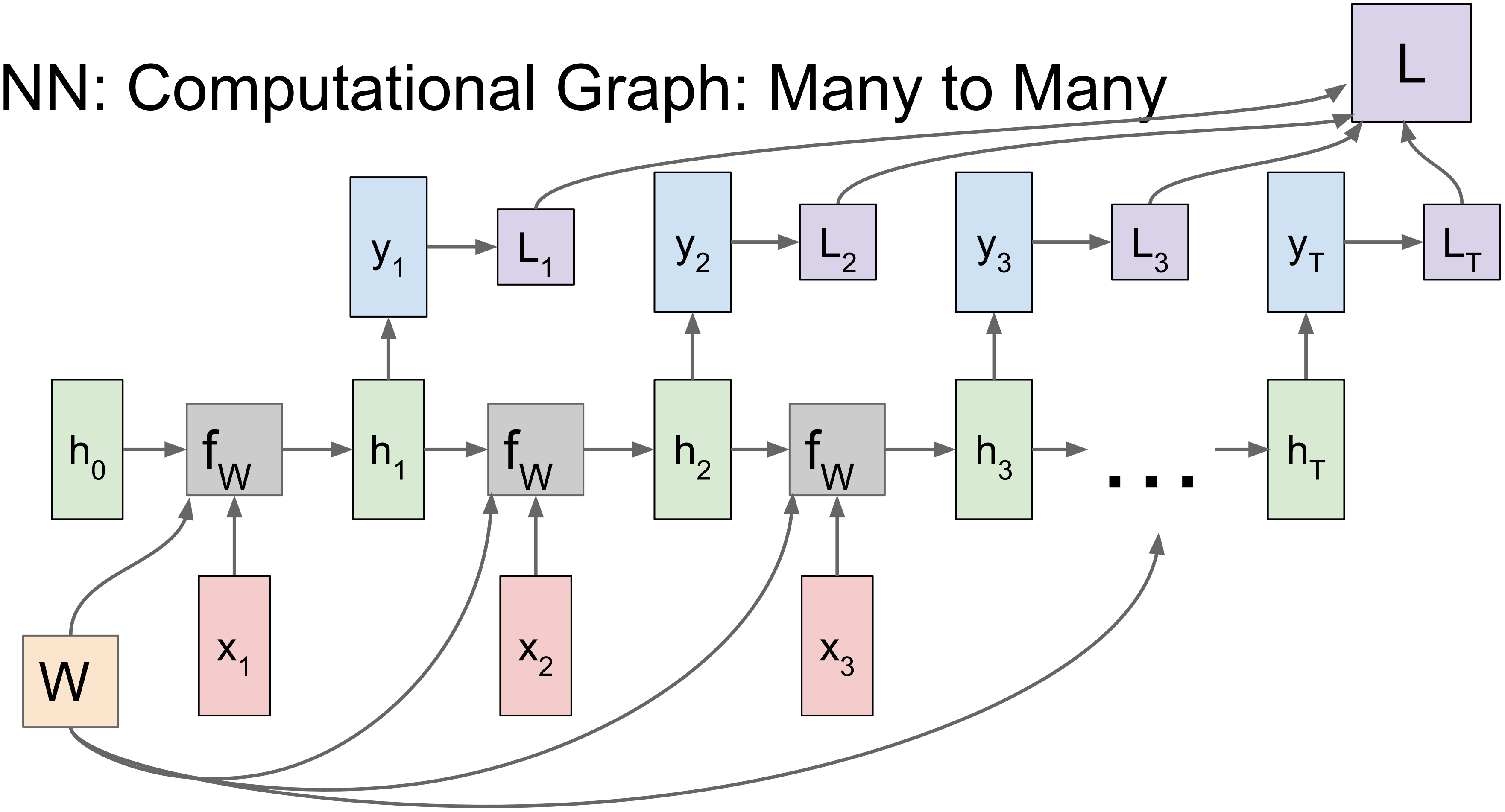
RNN: Computational Graph: Many to Many



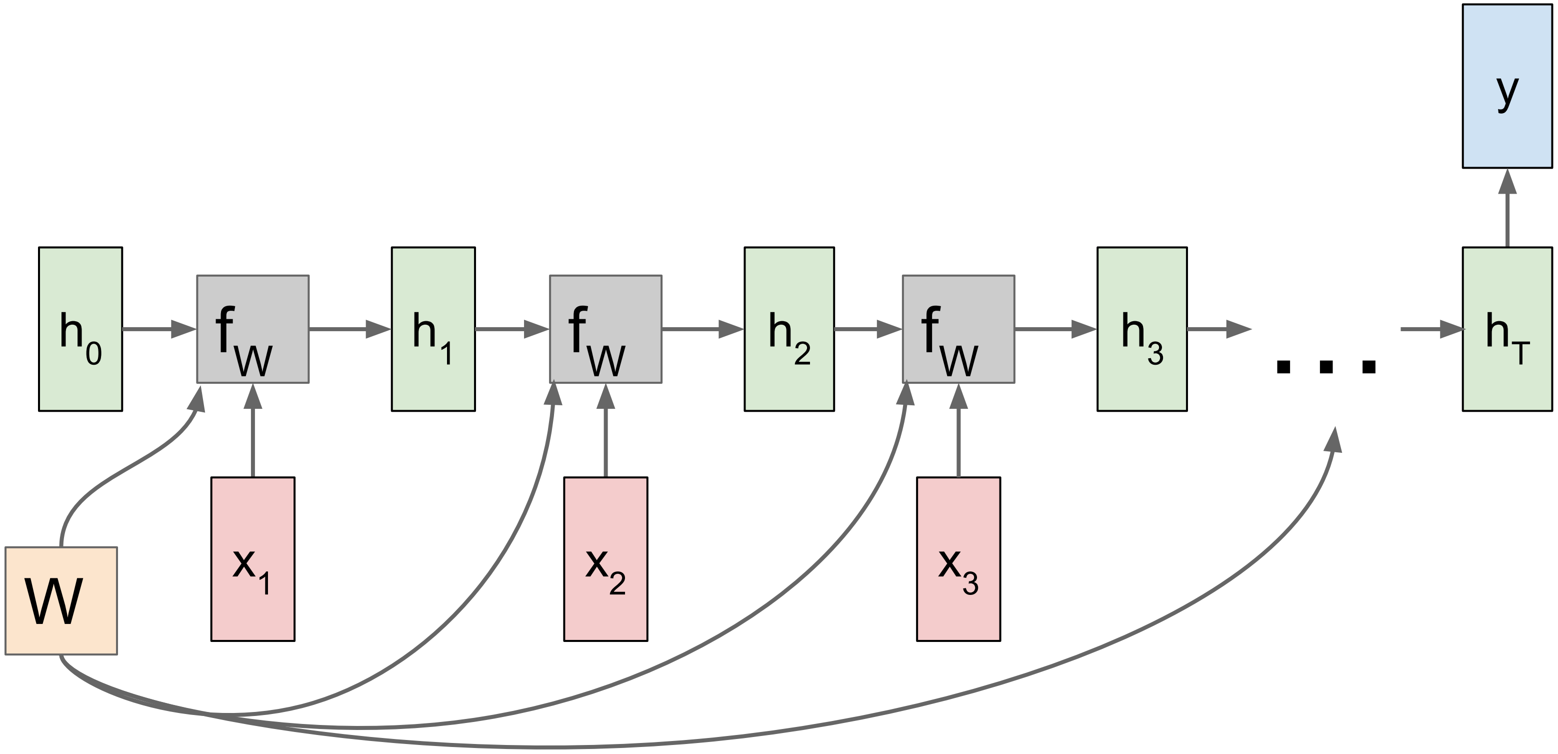
RNN: Computational Graph: Many to Many



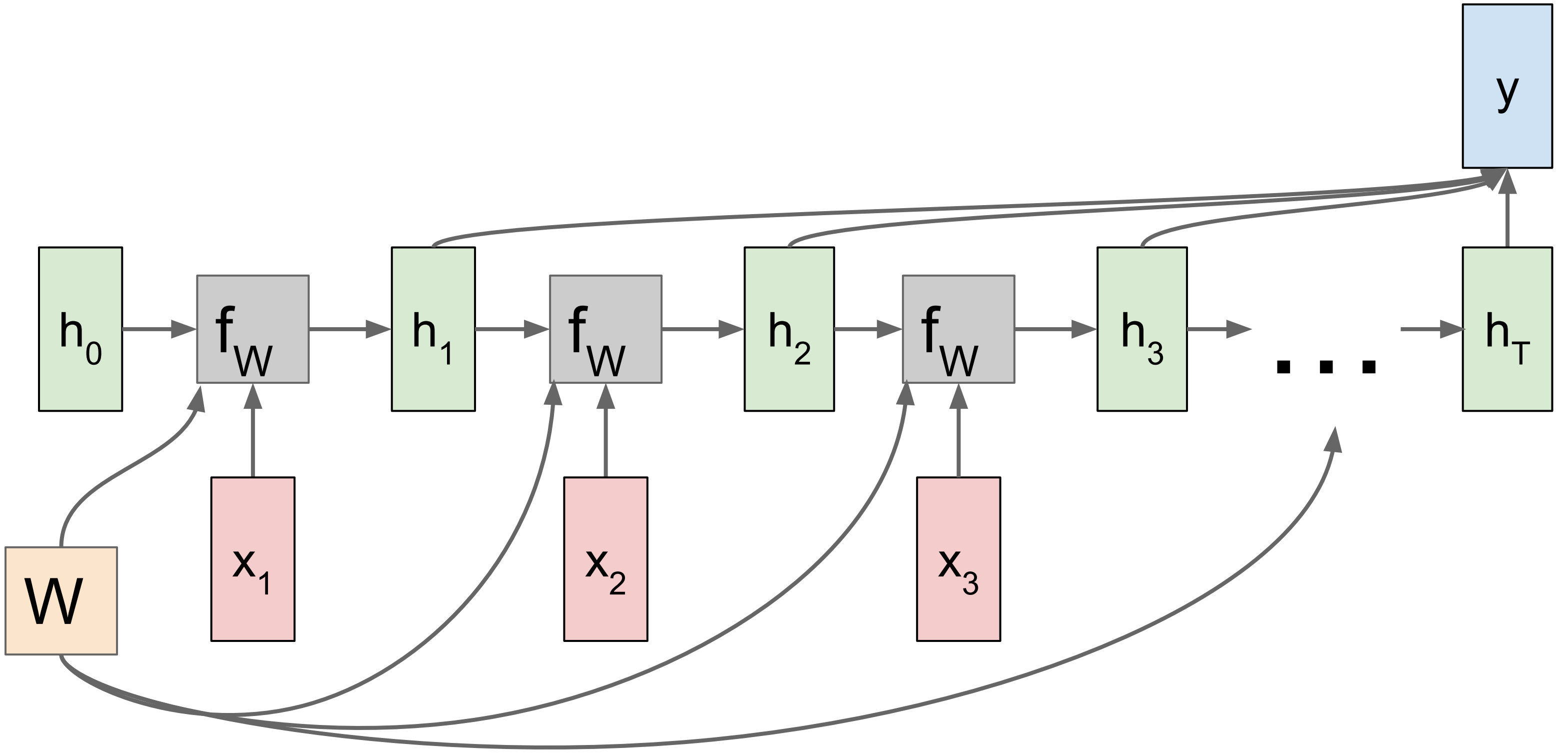
RNN: Computational Graph: Many to Many



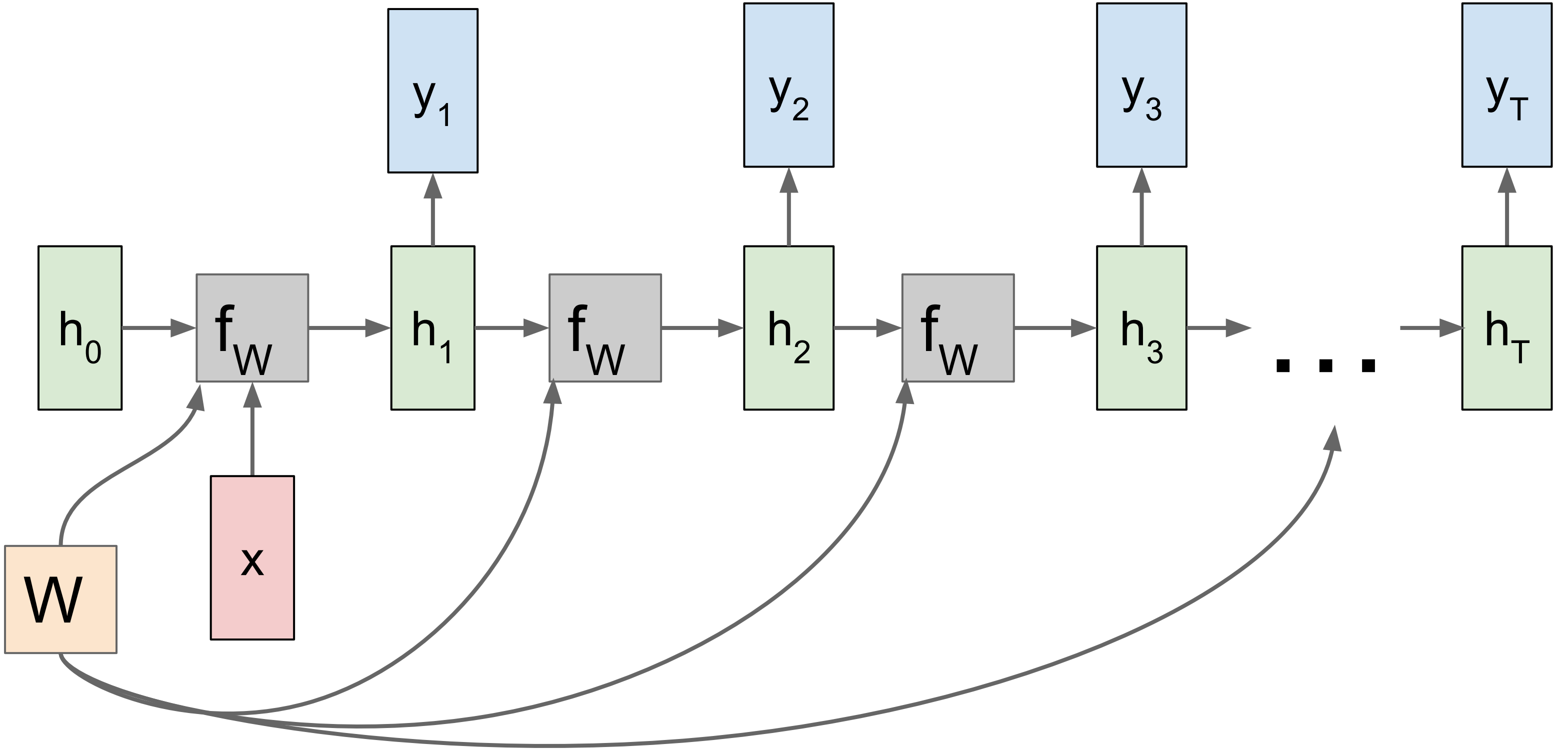
RNN: Computational Graph: Many to One



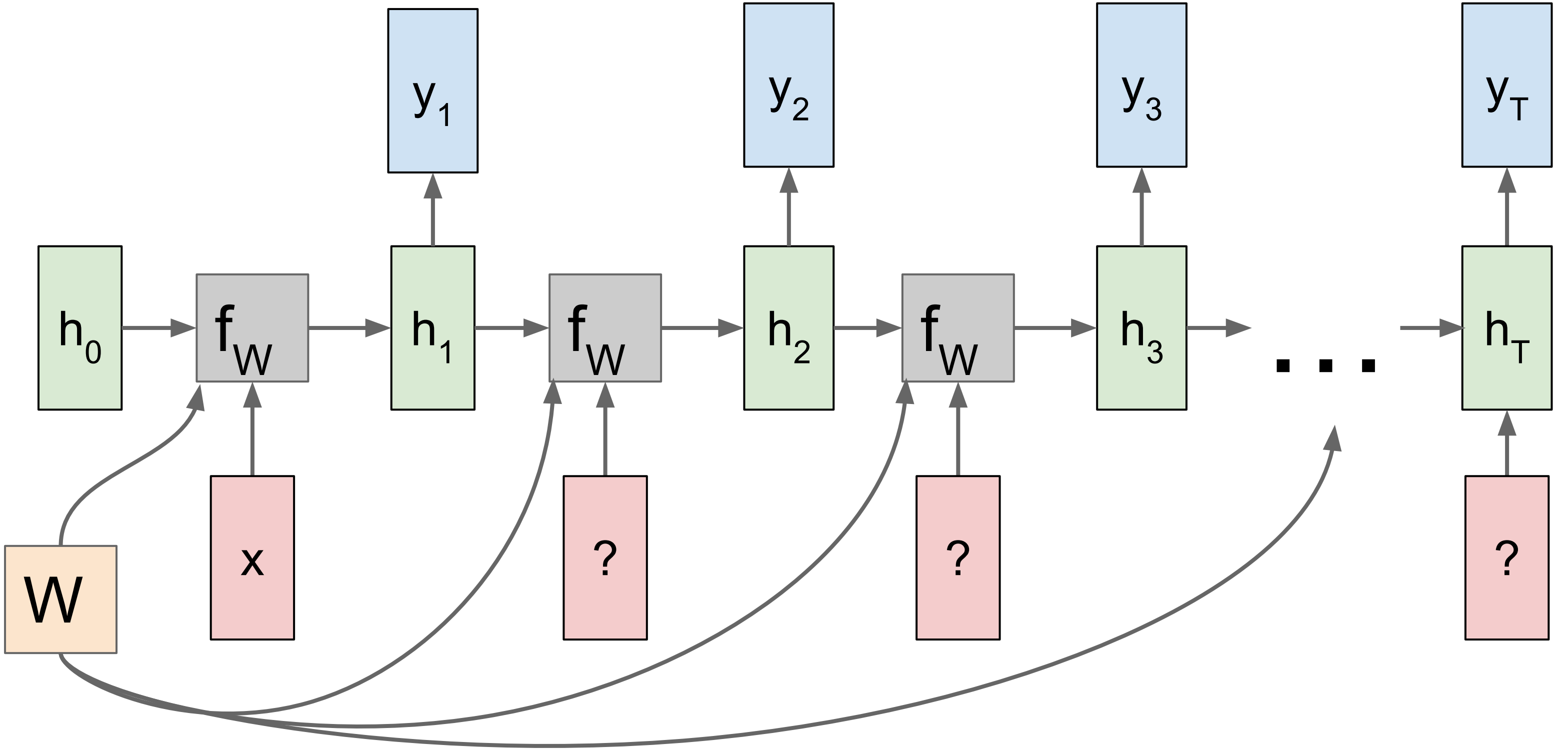
RNN: Computational Graph: Many to One



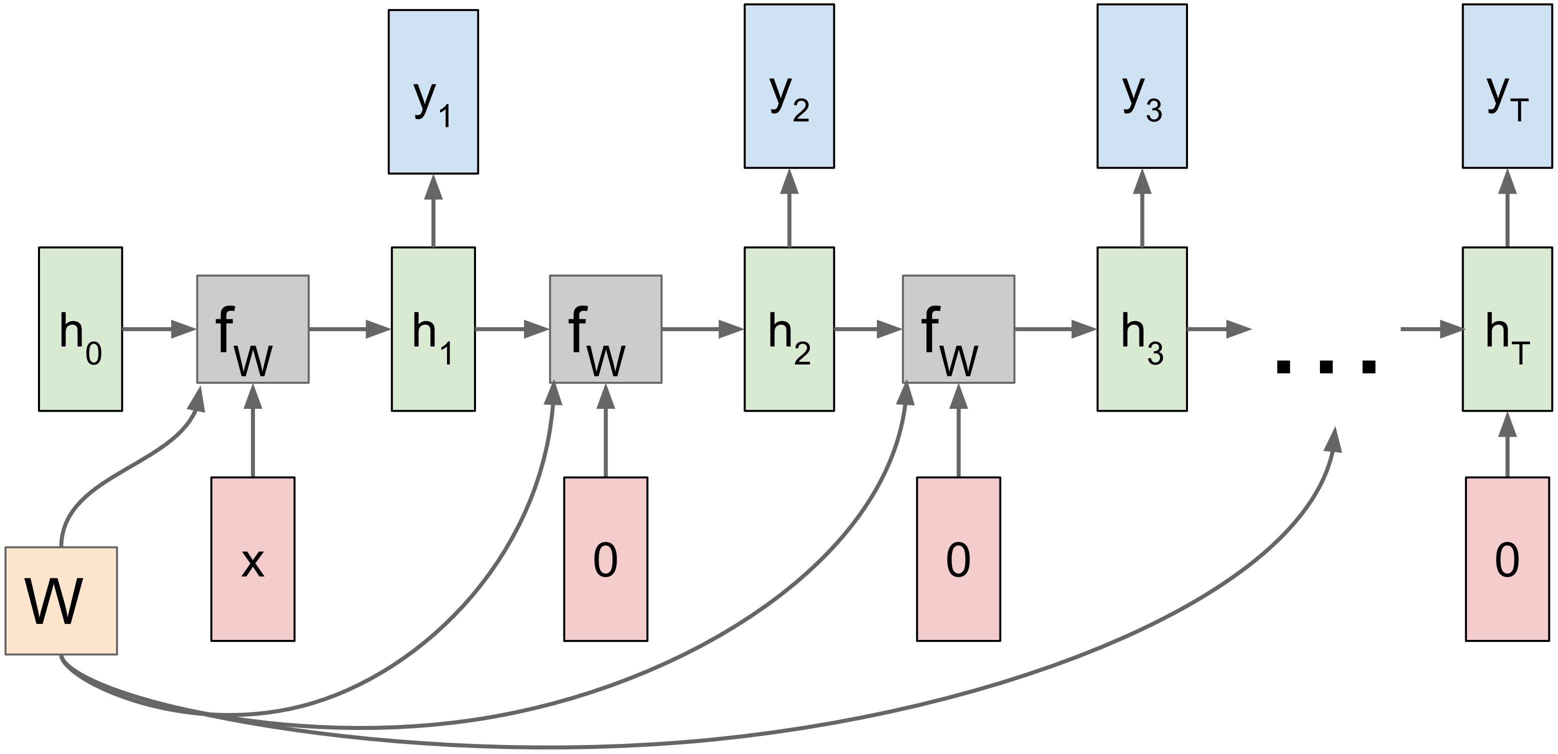
RNN: Computational Graph: One to Many



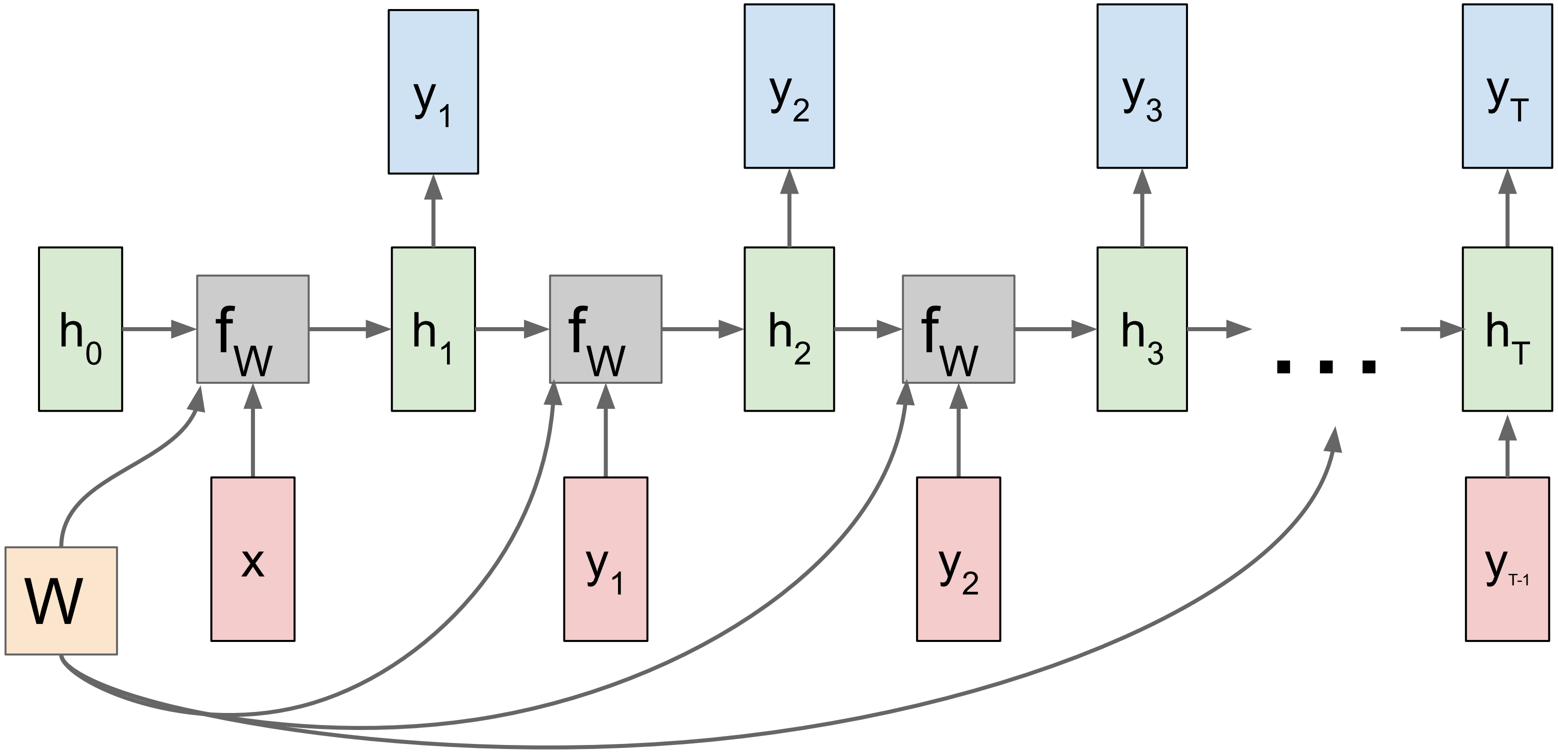
RNN: Computational Graph: One to Many



RNN: Computational Graph: One to Many

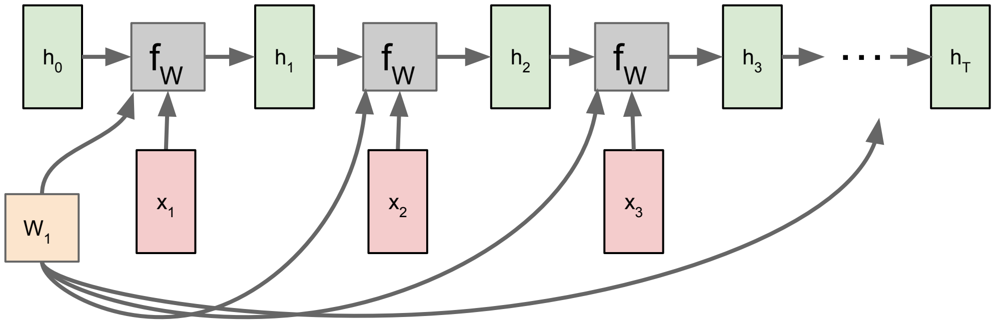


RNN: Computational Graph: One to Many



Sequence to Sequence: Many-to-one + one-to-many

Many to one: Encode input sequence in a single vector

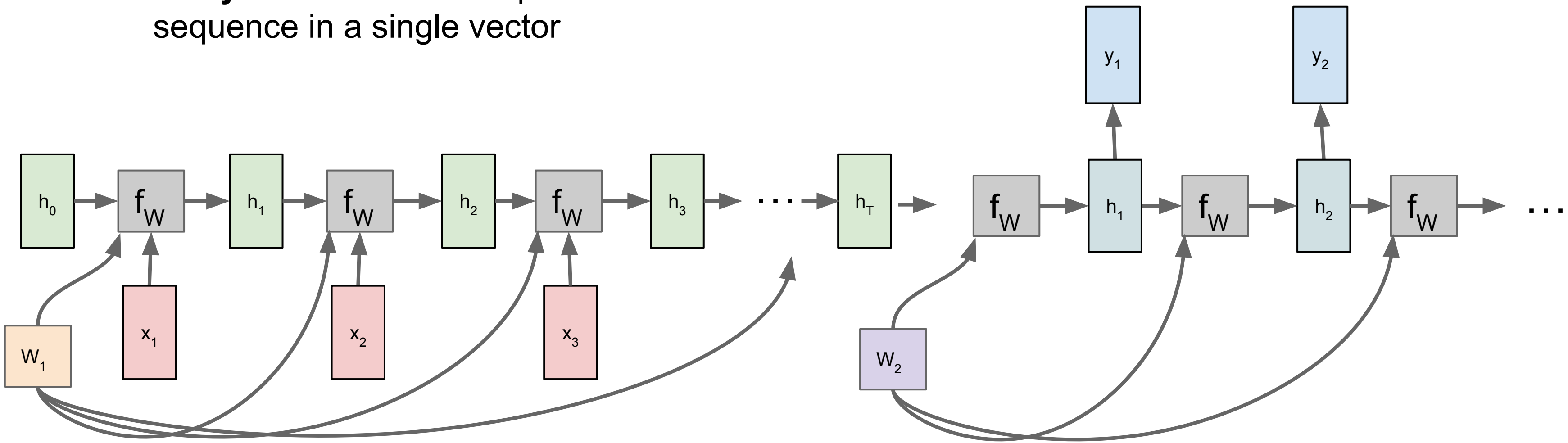


Sutskever et al, “Sequence to Sequence Learning with Neural Networks”, NIPS 2014

Sequence to Sequence: Many-to-one + one-to-many

One to many: Produce output sequence from single input vector

Many to one: Encode input sequence in a single vector

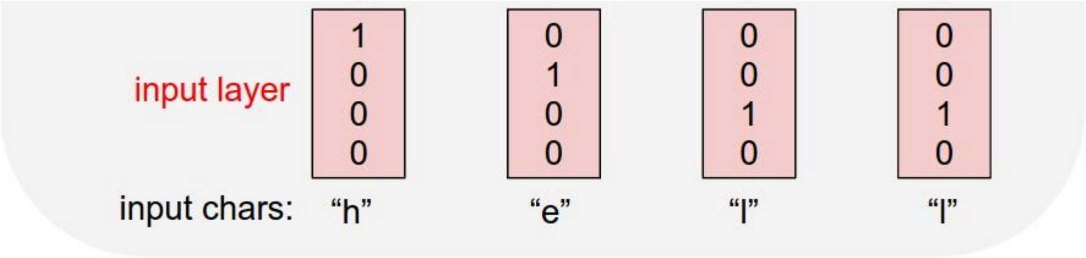


Sutskever et al, “Sequence to Sequence Learning with Neural Networks”, NIPS 2014

Example:
Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training sequence:
“hello”

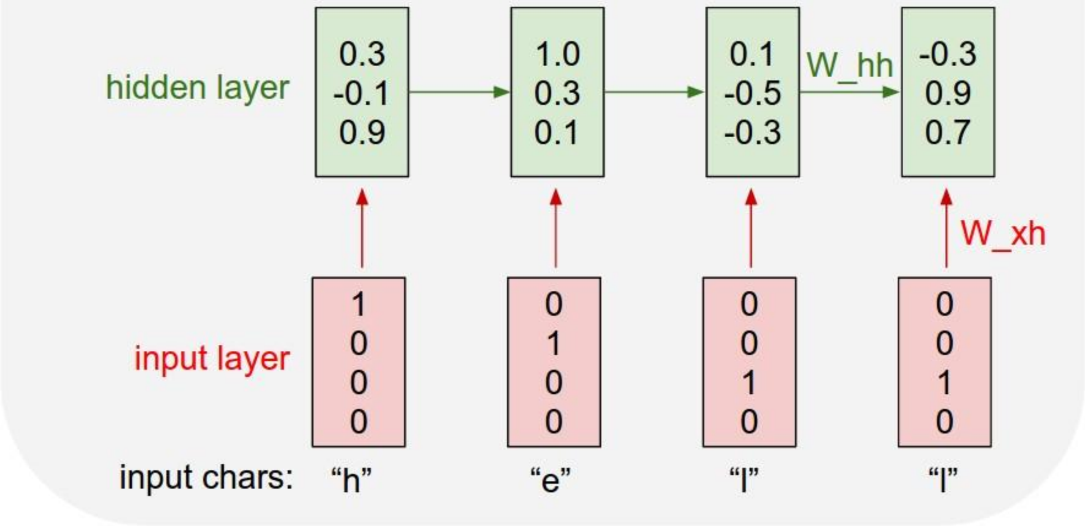


Example: Character-level Language Model

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Vocabulary: [h,e,l,o]

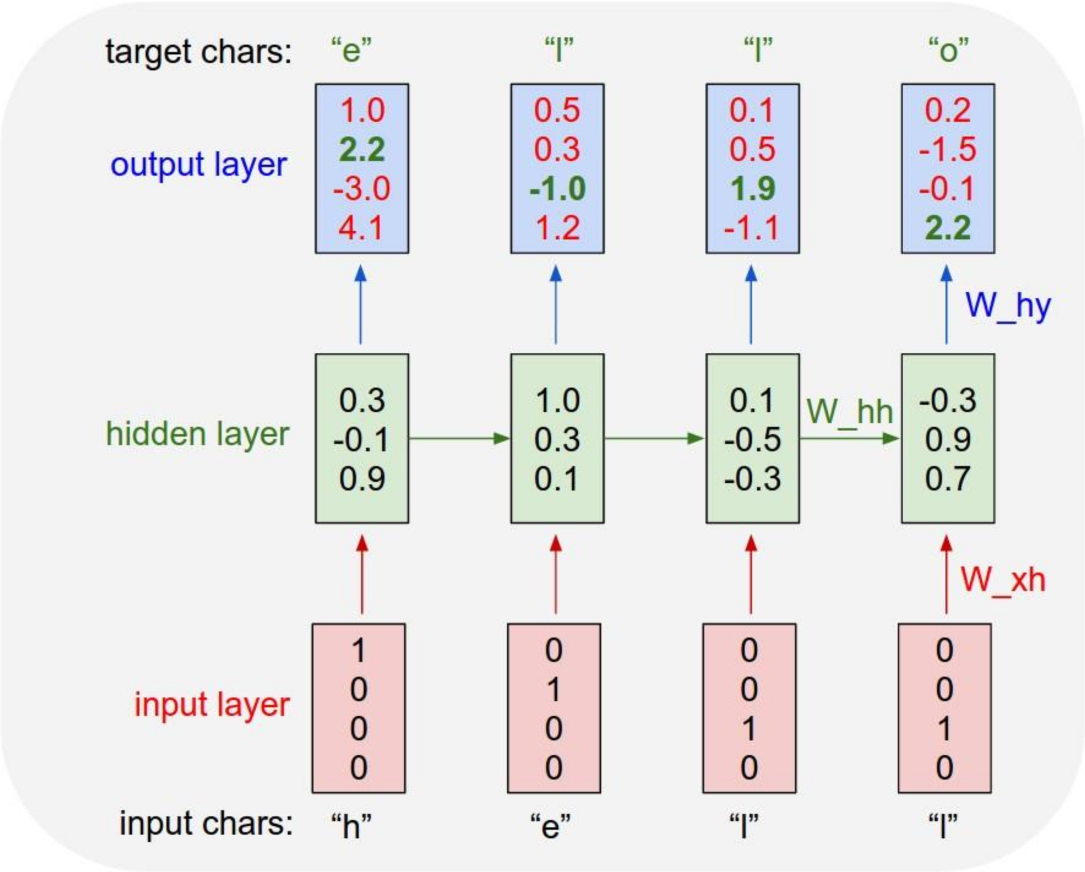
Example training sequence: **“hello”**



Example: Character-level Language Model

Vocabulary: [h,e,l,o]

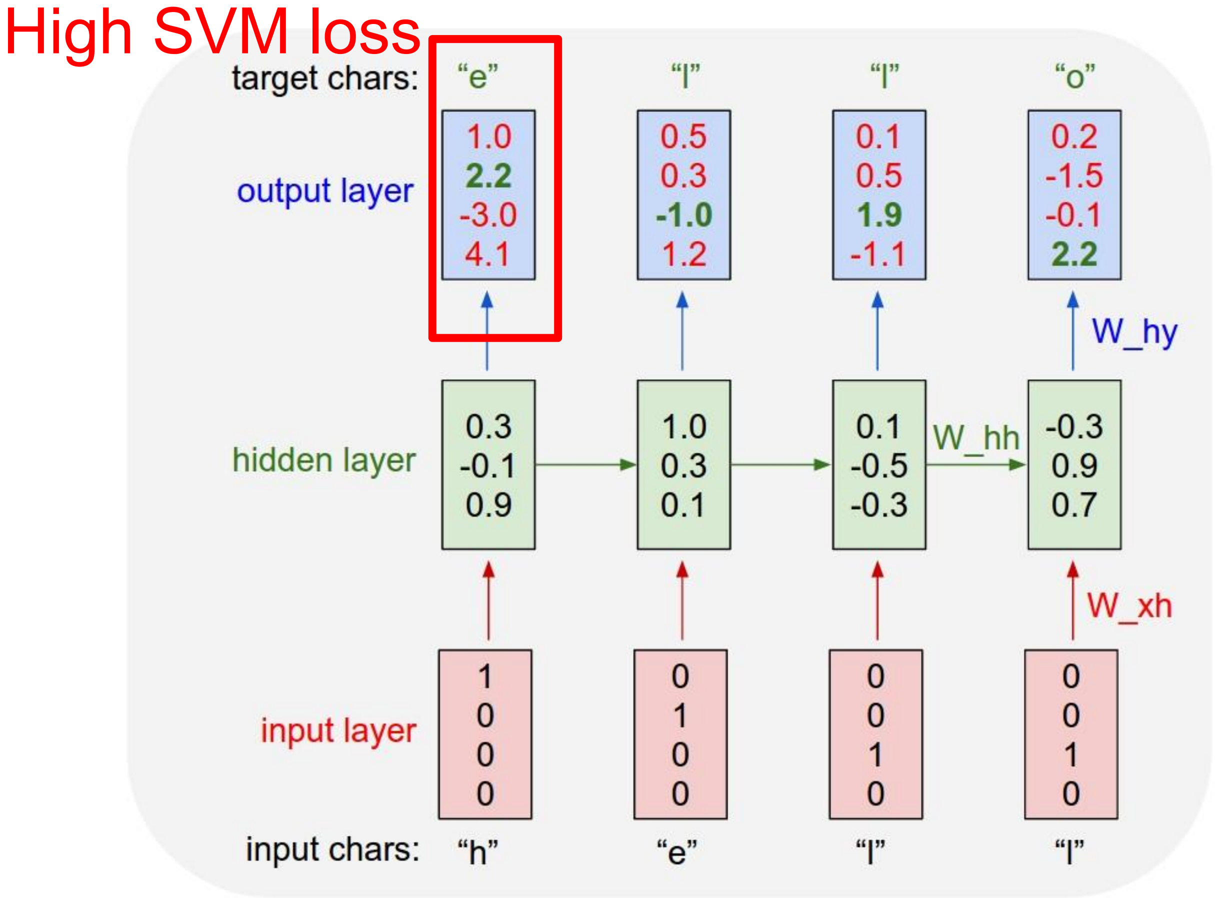
Example training sequence: **“hello”**



Example: Character-level Language Model

Vocabulary: [h,e,l,o]

Example training sequence: **“hello”**

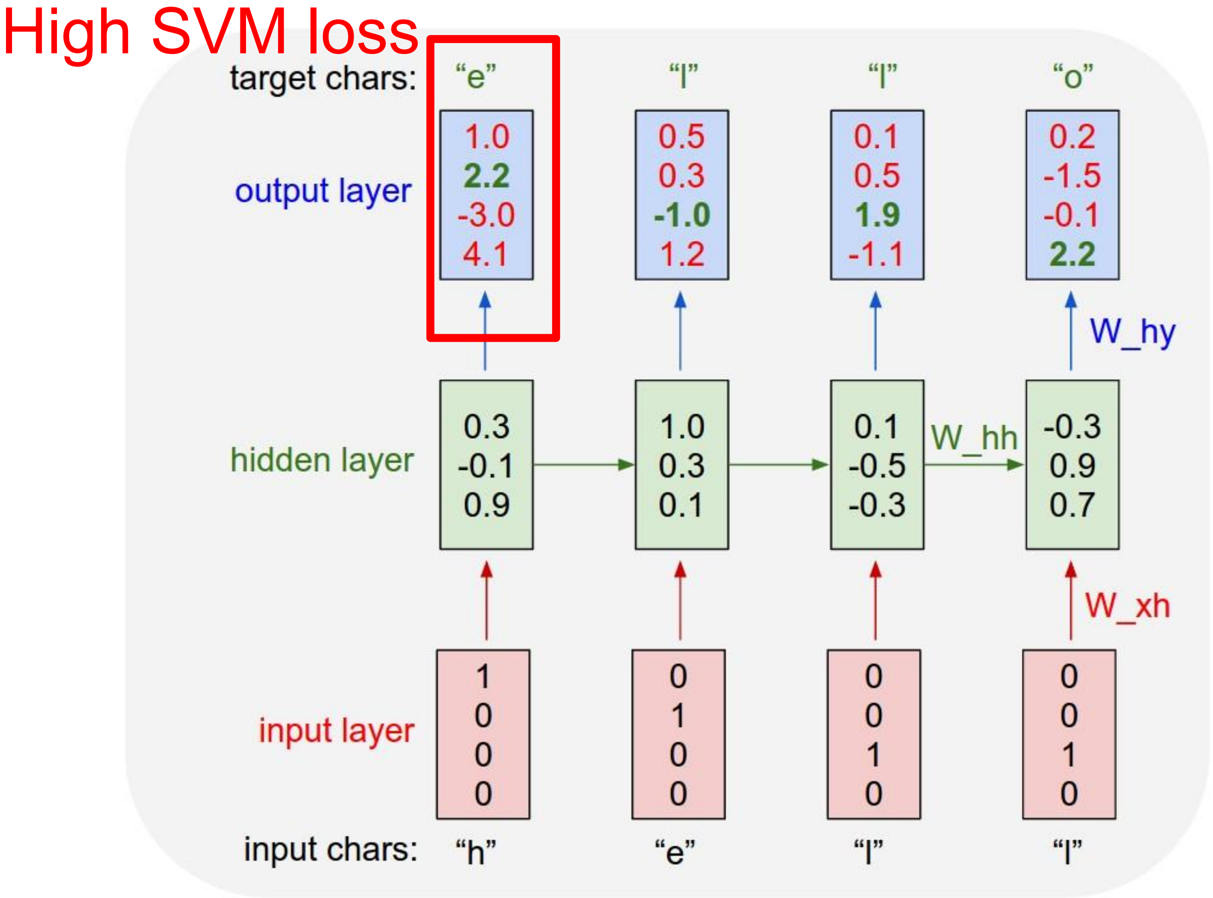


Example: Character-level Language Model

So far: encode inputs as one-hot-vector

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{14} \\ w_{31} & w_{32} & w_{33} & w_{14} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{31} \end{bmatrix}$$

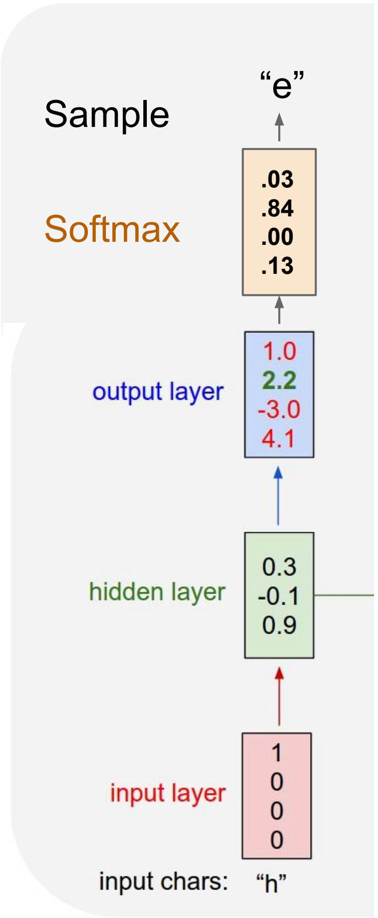
Matrix multiply with a one-hot vector just extracts a column from the weight matrix. Now a days, we extract this into a separate embedding layer



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

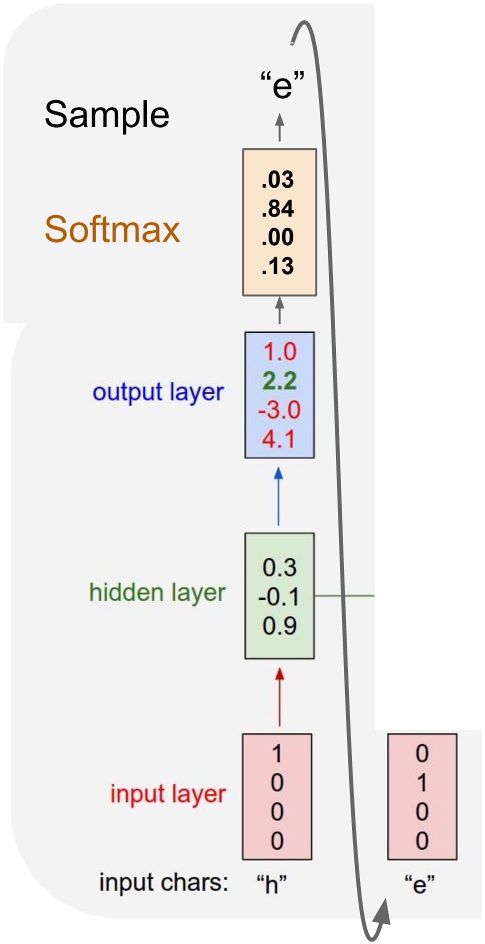
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

Vocabulary: [h,e,l,o]

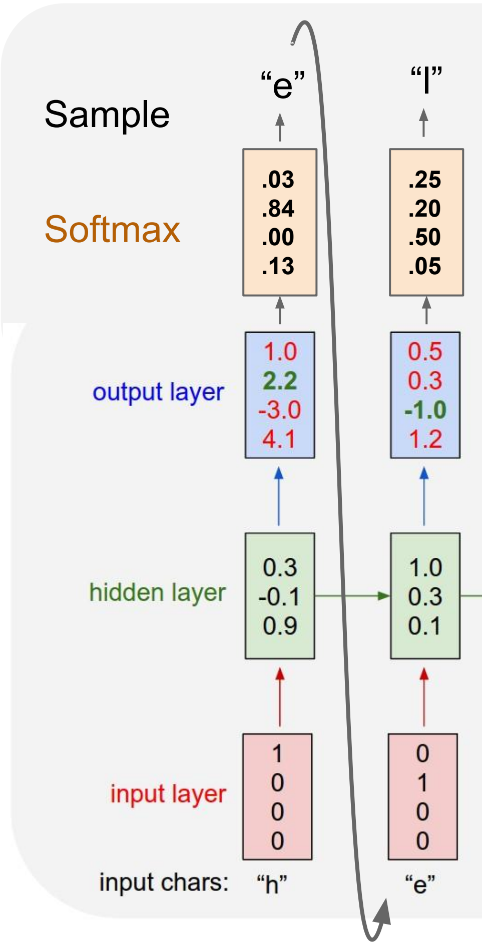
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

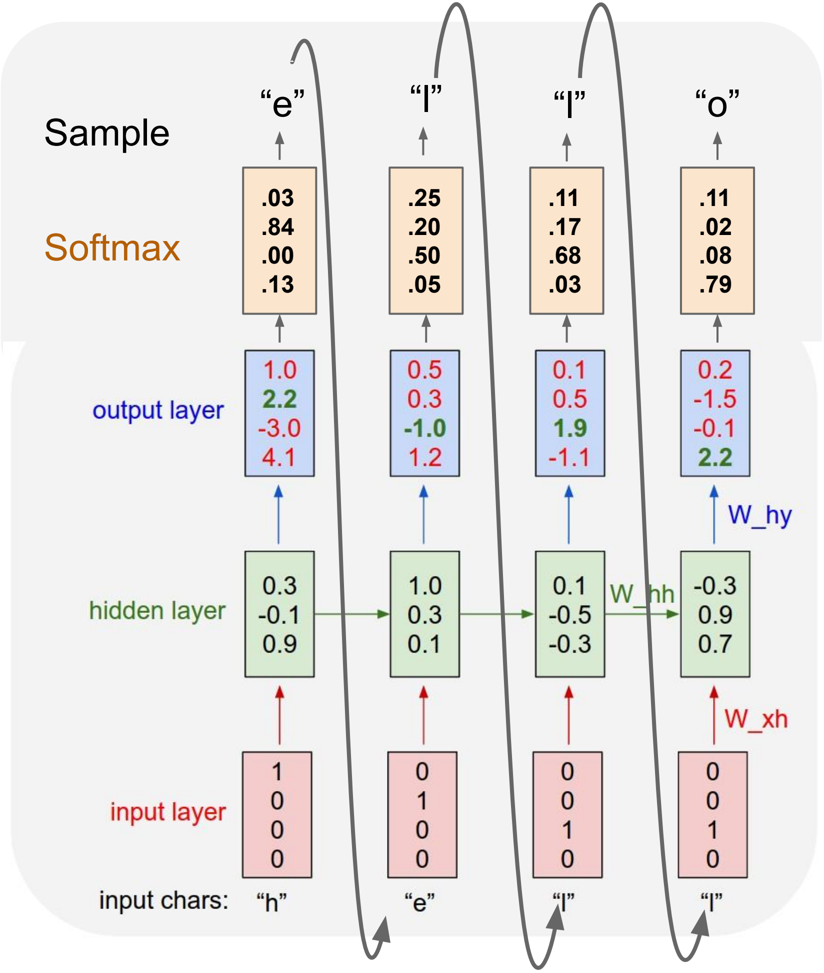
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

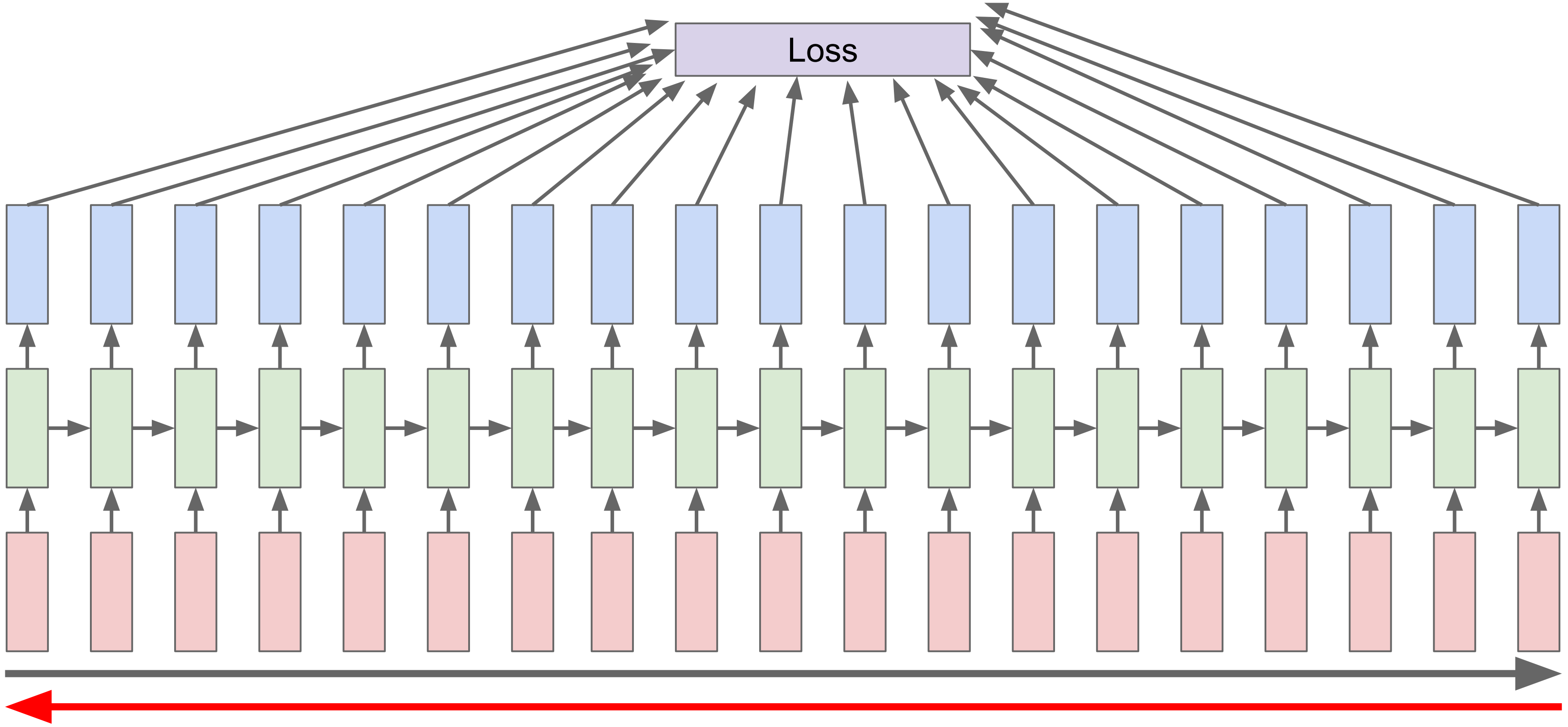
Vocabulary:
[h,e,l,o]

At test-time sample characters one at a time, feed back to model

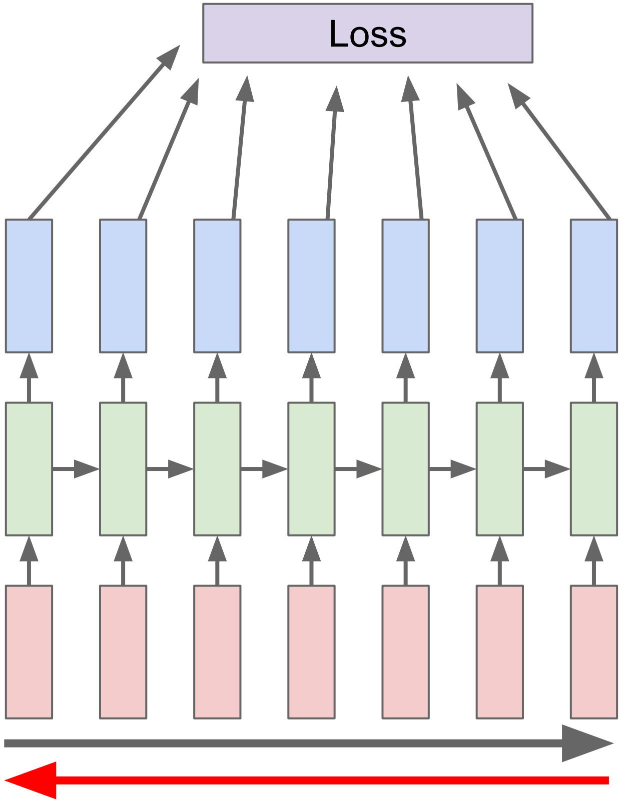


Backpropagation through time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

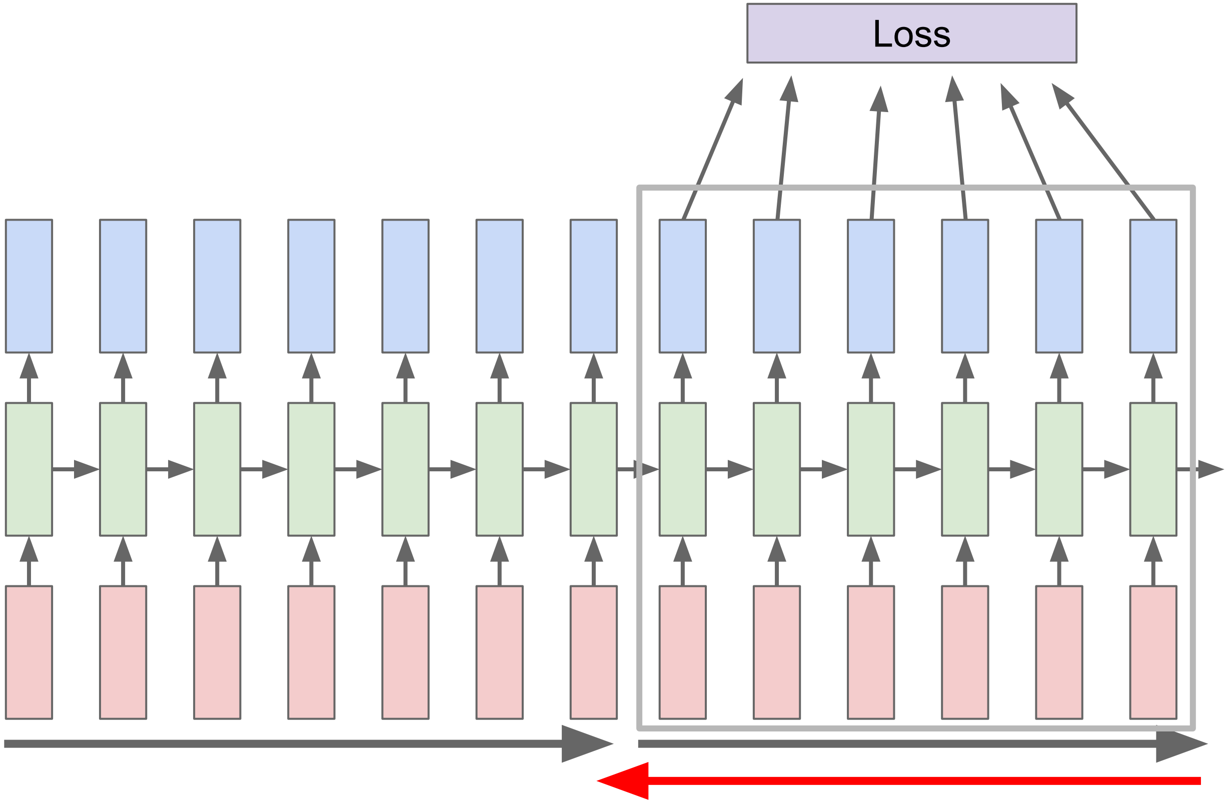


Truncated Backpropagation through time



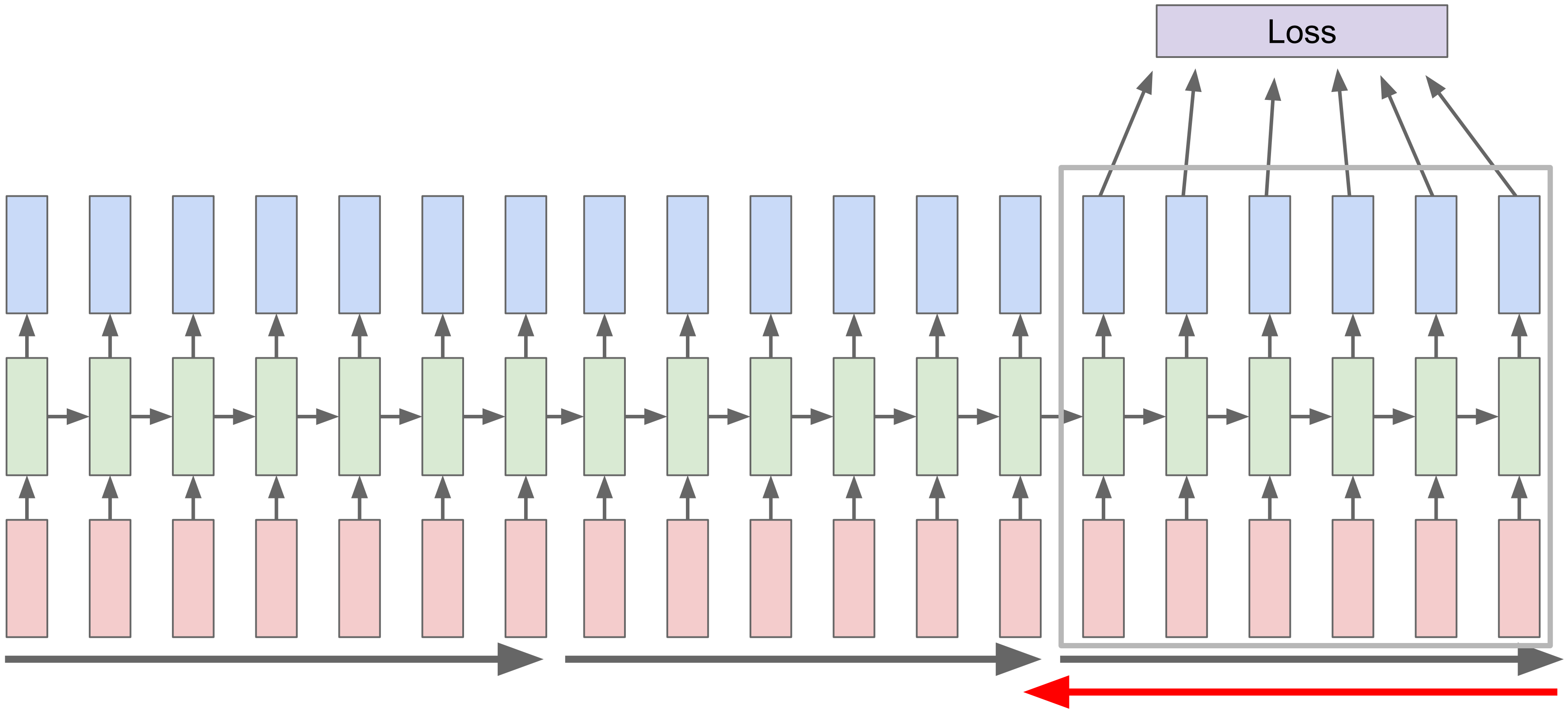
Run forward and backward through chunks of the sequence instead of whole sequence

Truncated Backpropagation through time



Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

Truncated Backpropagation through time



min-char-rnn.py gist: 112 lines of Python

```python
"""
Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
BSD License
"""
import numpy as np

# data I/O
data = open('input.txt', 'r').read() # should be simple plain text file
chars = list(set(data))
data_size, vocab_size = len(data), len(chars)
print 'data has %d characters, %d unique.' % (data_size, vocab_size)
char_to_ix = { ch:i for i,ch in enumerate(chars) }
ix_to_char = { i:ch for i,ch in enumerate(chars) }

# hyperparameters
hidden_size = 100 # size of hidden layer of neurons
seq_length = 25 # number of steps to unroll the RNN for
learning_rate = 1e-1

# model parameters
Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
bh = np.zeros((hidden_size, 1)) # hidden bias
by = np.zeros((vocab_size, 1)) # output bias

def lossFun(inputs, targets, hprev):
  """
  inputs,targets are both list of integers.
  hprev is Hx1 array of initial hidden state
  returns the loss, gradients on model parameters, and last hidden state
  """
  xs, hs, ys, ps = {}, {}, {}, {}
  hs[-1] = np.copy(hprev)
  loss = 0
  # forward pass
  for t in xrange(len(inputs)):
    xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
    xs[t][inputs[t]] = 1
    hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
    ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
    ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
    loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
  # backward pass: compute gradients going backwards
  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
  dbh, dby = np.zeros_like(bh), np.zeros_like(by)
  dhnext = np.zeros_like(hs[0])
  for t in reversed(xrange(len(inputs))):
    dy = np.copy(ps[t])
    dy[targets[t]] -= 1 # backprop into y
    dWhy += np.dot(dy, hs[t].T)
    dby += dy
    dh = np.dot(Why.T, dy) + dhnext # backprop into h
    dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity
    dbh += dhraw
    dWxh += np.dot(dhraw, xs[t].T)
    dWhh += np.dot(dhraw, hs[t-1].T)
    dhnext = np.dot(Whh.T, dhraw)
  for dparam in [dWxh, dWhh, dWhy, dbh, dby]:
    np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
  return loss, dWxh, dWhh, dWhy, dbh, dby, hs[len(inputs)-1]
```

```python
def sample(h, seed_ix, n):
  """
  sample a sequence of integers from the model
  h is memory state, seed_ix is seed letter for first time step
  """
  x = np.zeros((vocab_size, 1))
  x[seed_ix] = 1
  ixes = []
  for t in xrange(n):
    h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
    y = np.dot(Why, h) + by
    p = np.exp(y) / np.sum(np.exp(y))
    ix = np.random.choice(range(vocab_size), p=p.ravel())
    x = np.zeros((vocab_size, 1))
    x[ix] = 1
    ixes.append(ix)
  return ixes

n, p = 0, 0
mWxh, mWhh, mWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)
mbh, mby = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
while True:
  # prepare inputs (we're sweeping from left to right in steps seq_length long)
  if p+seq_length+1 >= len(data) or n == 0:
    hprev = np.zeros((hidden_size,1)) # reset RNN memory
    p = 0 # go from start of data
  inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
  targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]

  # sample from the model now and then
  if n % 100 == 0:
    sample_ix = sample(hprev, inputs[0], 200)
    txt = ''.join(ix_to_char[ix] for ix in sample_ix)
    print '----\n %s \n----' % (txt, )

  # forward seq_length characters through the net and fetch gradient
  loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
  smooth_loss = smooth_loss * 0.999 + loss * 0.001
  if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress

  # perform parameter update with Adagrad
  for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
                                [dWxh, dWhh, dWhy, dbh, dby],
                                [mWxh, mWhh, mWhy, mbh, mby]):
    mem += dparam * dparam
    param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

  p += seq_length # move data pointer
  n += 1 # iteration counter
```

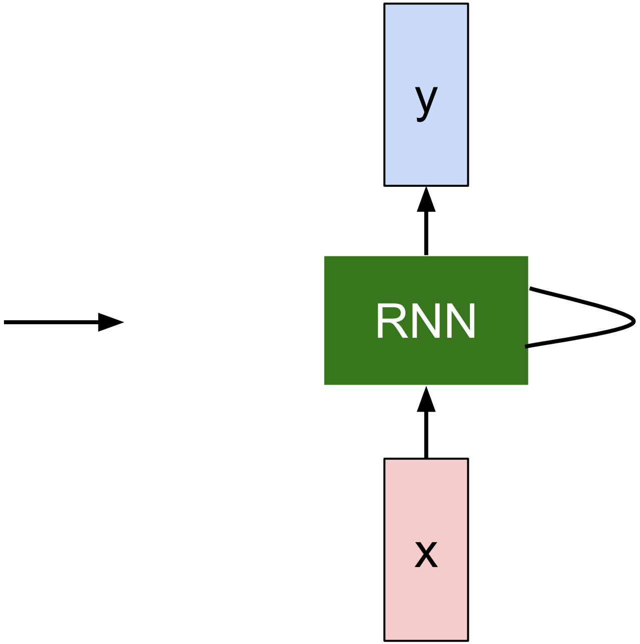
Simple python implementation

THE SONNETS

by William Shakespeare

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And tender churl mak'st waste in niggarding:
  Pity the world, or else this glutton be,
  To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a tatter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thriftless praise.
How much more praise deserv'd thy beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
  This were to be new made when thou art old,
  And see thy blood warm when thou feel'st it cold.



at first:

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

↓ train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

↓ train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

↓ train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

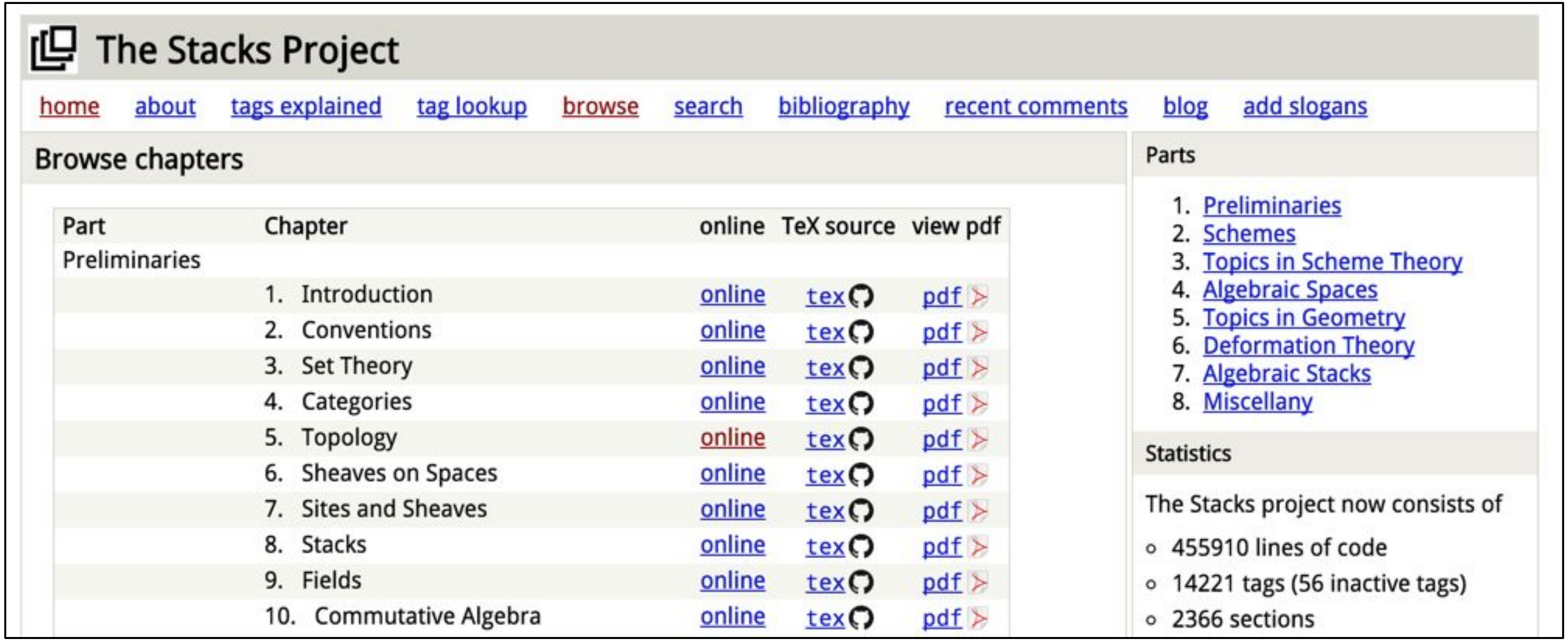
Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.

VIOLA:
Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:
O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.

The Stacks Project: open source algebraic geometry textbook



The Stacks Project

home | about | tags explained | tag lookup | browse | search | bibliography | recent comments | blog | add slogans

Browse chapters

Part	Chapter	online	TeX source	view pdf
Preliminaries				
	1. Introduction	online	tex	pdf
	2. Conventions	online	tex	pdf
	3. Set Theory	online	tex	pdf
	4. Categories	online	tex	pdf
	5. Topology	online	tex	pdf
	6. Sheaves on Spaces	online	tex	pdf
	7. Sites and Sheaves	online	tex	pdf
	8. Stacks	online	tex	pdf
	9. Fields	online	tex	pdf
	10. Commutative Algebra	online	tex	pdf

Parts

1. Preliminaries
2. Schemes
3. Topics in Scheme Theory
4. Algebraic Spaces
5. Topics in Geometry
6. Deformation Theory
7. Algebraic Stacks
8. Miscellany

Statistics

The Stacks project now consists of

- 455910 lines of code
- 14221 tags (56 inactive tags)
- 2366 sections

Latex source

http://stacks.math.columbia.edu/

For $\bigoplus_{n=1,\dots,m}$ where $\mathcal{L}_{m_\bullet} = 0$, hence we can find a closed subset $\mathcal{H}$ in $\mathcal{H}$ and any sets $\mathcal{F}$ on X, U is a closed immersion of S, then $U \to T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \mathrm{Spec}(R) = U \times_X U \times_X U$$

and the comparicoly in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \to V$. Consider the maps M along the set of points Sch_{fppf} and $U \to U$ is the fibre category of S in U in Section, **??** and the fact that any U affine, see Morphisms, Lemma **??**. Hence we obtain a scheme S and any open subset $W \subset U$ in $Sh(G)$ such that $\mathrm{Spec}(R') \to S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S. We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \to \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma **??** we can define a map of complexes $\mathrm{GL}_{S'}(x'/S'')$ and we win. □

To prove study we see that $\mathcal{F}|_U$ is a covering of $\mathcal{X}'$, and $\mathcal{T}_i$ is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and $\mathcal{F}_p$ exists and let $\mathcal{F}_i$ be a presheaf of $\mathcal{O}_X$-modules on $\mathcal{C}$ as a $\mathcal{F}$-module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\mathrm{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1}\mathcal{F})$$

is a unique morphism of algebraic stacks. Note that

$$\mathrm{Arrows} = (Sch/S)^{opp}_{fppf}, (Sch/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \longmapsto (U, \mathrm{Spec}(A))$$

is an open subset of X. Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S.

Proof. See discussion of sheaves of sets. □

The result for prove any open covering follows from the less of Example **??**. It may replace S by $X_{spaces,étale}$ which gives an open subspace of X and T equal to S_{Zar}, see Descent, Lemma **??**. Namely, by Lemma **??** we see that R is geometrically regular over S.

Lemma 0.1. *Assume (3) and (3) by the construction in the description.*

Suppose $X = \lim |X|$ (by the formal open covering X and a single map $\underline{Proj}_X(\mathcal{A}) = \mathrm{Spec}(B)$ over U compatible with the complex

$$Set(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

*When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition **??** (without element is when the closed subschemes are catenary. If T is surjective we may assume that T is connected with residue fields of S. Moreover there exists a closed subspace $Z \subset X$ of X where U in X' is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem*

(1) *f is locally of finite type. Since $S = \mathrm{Spec}(R)$ and $Y = \mathrm{Spec}(R)$.*

Proof. This is form all sheaves of sheaves on X. But given a scheme U and a surjective étale morphism $U \to X$. Let $U \cap U = \coprod_{i=1,\dots,n} U_i$ be the scheme X over S at the schemes $X_i \to X$ and $U = \lim_i X_i$. □

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{\mathcal{X},\dots,0}$.

Lemma 0.2. *Let X be a locally Noetherian scheme over S, $E = \mathcal{F}_{X/S}$. Set $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. Since $\mathcal{I}^n \subset \mathcal{I}^n$ are nonzero over $i_0 \leq \mathfrak{p}$ is a subset of $\mathcal{J}_{n,0} \circ \overline{A}_2$ works.*

Lemma 0.3. *In Situation **??**. Hence we may assume $\mathfrak{q}' = 0$.*

Proof. We will use the property we see that $\mathfrak{p}$ is the mext functor (**??**). On the other hand, by Lemma **??** we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F-algebra where δ_{n+1} is a scheme over S. □

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

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

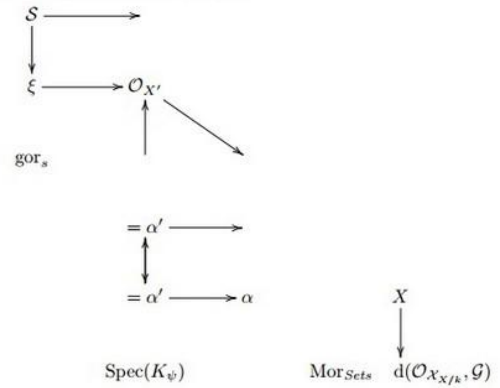
be a morphism of algebraic spaces over S and Y.

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. (1) $\mathcal{F}$ is an algebraic space over S.
2. (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.

□

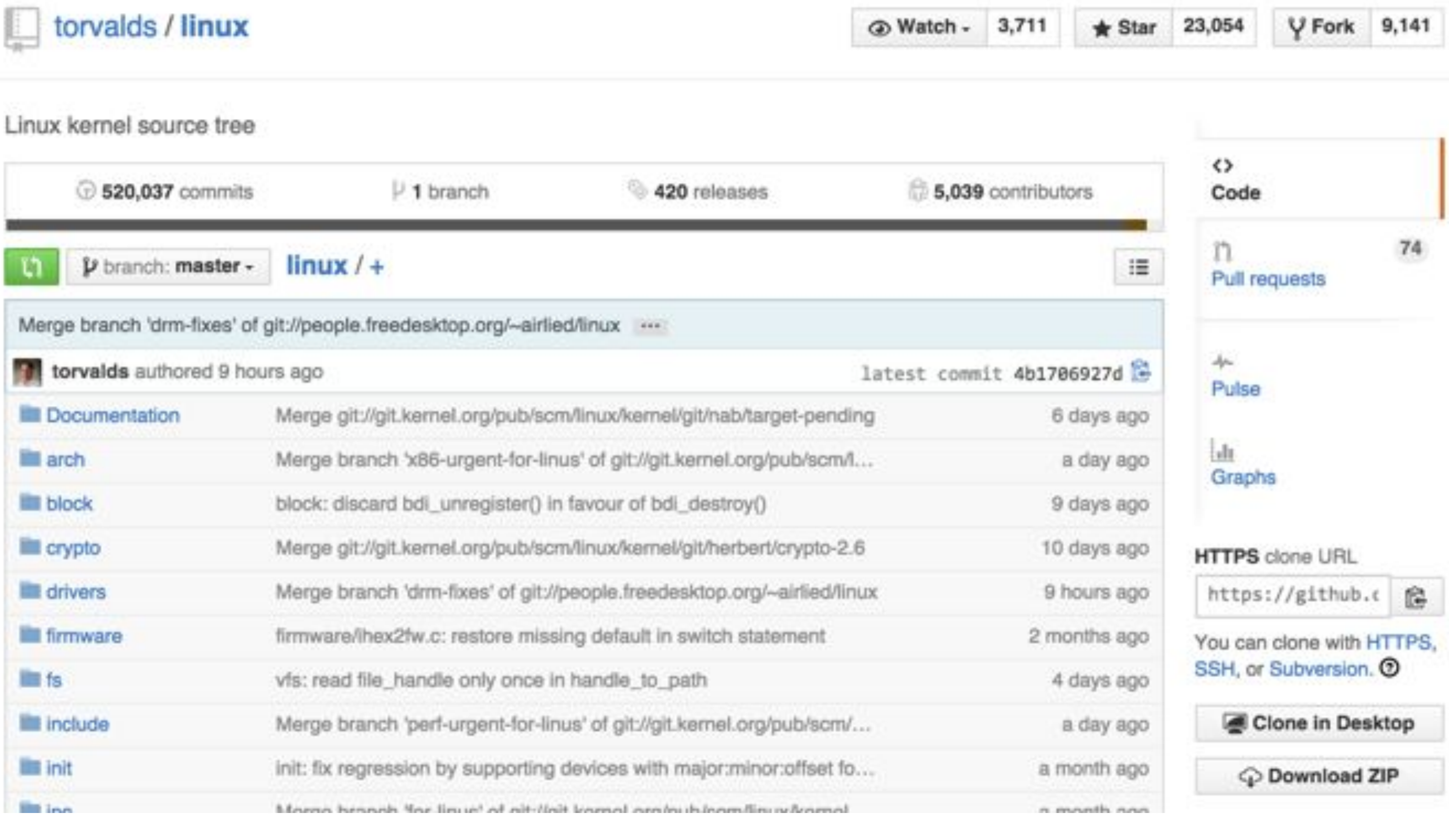
Proof. We have see that $X = \text{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

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.


torvalds / linux
Watch 3,711
Star 23,054
Fork 9,141
Linux kernel source tree
520,037 commits
1 branch
420 releases
5,039 contributors
branch: master
linux / +
Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux
torvalds authored 9 hours ago
latest commit 4b1706927d
Documentation
Merge git://git.kernel.org/pub/scm/linux/kernel/git/nab/target-pending
6 days ago
arch
Merge branch 'x86-urgent-for-linus' of git://git.kernel.org/pub/scm/l…
a day ago
block
block: discard bdi_unregister() in favour of bdi_destroy()
9 days ago
crypto
Merge git://git.kernel.org/pub/scm/linux/kernel/git/herbert/crypto-2.6
10 days ago
drivers
Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux
9 hours ago
firmware
firmware/ihex2fw.c: restore missing default in switch statement
2 months ago
fs
vfs: read file_handle only once in handle_to_path
4 days ago
include
Merge branch 'perf-urgent-for-linus' of git://git.kernel.org/pub/scm/…
a day ago
init
init: fix regression by supporting devices with major:minor:offset fo…
a month ago
Code
Pull requests 74
Pulse
Graphs
HTTPS clone URL
https://github.c
You can clone with HTTPS, SSH, or Subversion.
Clone in Desktop
Download ZIP

```c
static void do_command(struct seq_file *m, void *v)
{
  int column = 32 << (cmd[2] & 0x80);
  if (state)
    cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
  else
    seq = 1;
  for (i = 0; i < 16; i++) {
    if (k & (1 << 1))
      pipe = (in_use & UMXTHREAD_UNCCA) +
        ((count & 0x00000000fffffff8) & 0x000000f) << 8;
    if (count == 0)
      sub(pid, ppc_md.kexec_handle, 0x20000000);
    pipe_set_bytes(i, 0);
  }
  /* Free our user pages pointer to place camera if all dash */
  subsystem_info = &of_changes[PAGE_SIZE];
  rek_controls(offset, idx, &soffset);
  /* Now we want to deliberately put it to device */
  control_check_polarity(&context, val, 0);
  for (i = 0; i < COUNTER; i++)
    seq_puts(s, "policy ");
}
```

Generated C code

```
/*
 *  Copyright (c) 2006-2010, Intel Mobile Communications.  All rights reserved.
 *
 *   This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 as published by
 * the Free Software Foundation.
 *
 *        This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 *   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *
 *  GNU General Public License for more details.
 *
 *   You should have received a copy of the GNU General Public License
 *    along with this program; if not, write to the Free Software Foundation,
 *  Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <linux/kexec.h>
#include <linux/errno.h>
#include <linux/io.h>
#include <linux/platform_device.h>
#include <linux/multi.h>
#include <linux/ckevent.h>

#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>
```

```
#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/setew.h>
#include <asm/pgproto.h>

#define REG_PG    vesa_slot_addr_pack
#define PFM_NOCOMP  AFSR(0, load)
#define STACK_DDR(type)     (func)

#define SWAP_ALLOCATE(nr)     (e)
#define emulate_sigs()  arch_get_unaligned_child()
#define access_rw(TST)  asm volatile("movd %%esp, %0, %3" : : "r" (0));   \
  if (__type & DO_READ)

static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \
          pC>[1]);

static void
os_prefix(unsigned long sys)
{
#ifdef CONFIG_PREEMPT
  PUT_PARAM_RAID(2, sel) = get_state_state();
  set_pid_sum((unsigned long)state, current_state_str(),
           (unsigned long)-1->lr_full; low;
}
```

OpenAI GPT-2 generated text

source

Input: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Output: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

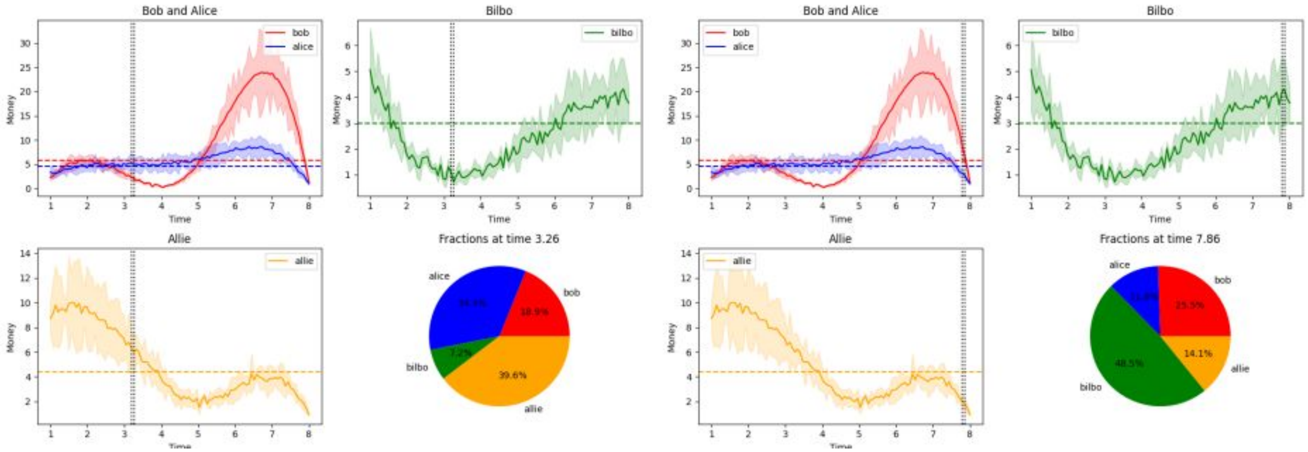
Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

GPT-4

Prompt: Can you generate a pyplot for the following data: $x = [1, 3, 5, 6, 8], y1 = [2, 3, 5, 18, 1], y2 = [3, 5, 6, 8, 1], y3 = [5, 1, 2, 3, 4], y4 = [9, 7, 2, 3, 1]$. I want $y1, y2$ to be in the same plot, but $y3$ is in another plot next to that plot, $y4$ is in below. I want the legend of $y1$ to be "bob", $y2$ to be "alice", $y3$ to be "bilbo", $y4$ to be "allie". I want the x-axis to be labeled with "time" and y axis to be labeled with "money". I want to add a 10 %-40% random error bar to each curve, through all times (including non-integers). I want smoothed curves to show the plot, and smoothed error bar. Do not use linear interpolation, use smooth interpolation! I want to also add some small zig-zag to the smoothed curve to make it look more real. I want to put a baseline as the mean of each line. I want to put a pie chart below indicating the fraction of the four people in each time step. I also want an animation to show how the fractions are changing in the pie chart in **continuous time**. Interpolate the missing fractions! I also want vertical line animation in other three plots to match the pie chart. I want the fanciest plot. Please add as many fancy things as possible.

Here is a possible code to generate a pyplot for the following data, with some comments to explain the steps:

GPT-4: [We give two snapshots from the animation resulting from the produced code]



Bubek et al. Sparks of AGI. ArXiv 2023

Language modeling leads to reasoning capabilities for GPT-4

Bubek et al. Sparks of AGI. ArXiv 2023

GPT-4 vs ChatGPT

Prompt: Here we have a book, 9 eggs, a laptop, a bottle and a nail. Please tell me how to stack them onto each other in a stable manner.

GPT-4:

```
One possible way to stack the objects onto each other in a stable manner is:

- Place the book flat on a level surface, such as a table or a floor. The book will serve as the base of
the stack and provide a large and sturdy support.
- Arrange the 9 eggs in a 3 by 3 square on top of the book, leaving some space between them. The eggs
will
form a second layer and distribute the weight evenly. Make sure the eggs are not cracked or broken, and
handle them gently to avoid dropping them.
- Place the laptop on top of the eggs, with the screen facing down and the keyboard facing up. The
laptop will fit snugly within the boundaries of the book and the eggs, and its flat and rigid surface
will provide a stable platform for the next layer. Make sure the laptop is closed and turned off, and do
not press any keys or buttons.
- Place the bottle on top of the laptop, with the cap facing up and the bottom facing down. The bottle
will add some height and balance to the stack, and its cylindrical shape will reduce the contact area
with the laptop. Make sure the bottle is empty and closed, and do not shake or tilt it.
- Place the nail on top of the bottle cap, with the pointy end facing up and the flat end facing down.
The nail will be the final and smallest object in the stack, and its sharp tip will prevent it from
rolling or sliding off the bottle. Make sure the nail is clean and straight, and do not touch or move
it.
```

ChatGPT:

```
It would be difficult to stack all of these objects on top of each other in a stable manner without
using any extra objects. The book and laptop are likely too large and heavy to stack on top of the
eggs without damaging them. The bottle and nail may be too small to provide a stable base for the other
objects.

Here is one possible way to stack the objects:

- Place the bottle on a flat surface.
- Carefully balance the nail on top of the bottle.
- Place the eggs on top of the nail, making sure they are balanced and not tilting to one side.
- Place the laptop on top of the eggs, again making sure it is balanced and not tilting to one side.
- Finally, place the book on top of the laptop.

This stack may not be very stable, so it is important to be careful when handling it. Alternatively, you
could try stacking the objects in a different order or using some of the objects as supports for others
to create a more stable stack.
```

RNN tradeoffs

RNN Advantages:

- Can process any length input
- Computation for step *t* can (in theory) use information from many steps back
- Model size doesn't increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

RNN Disadvantages:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

Image Captioning

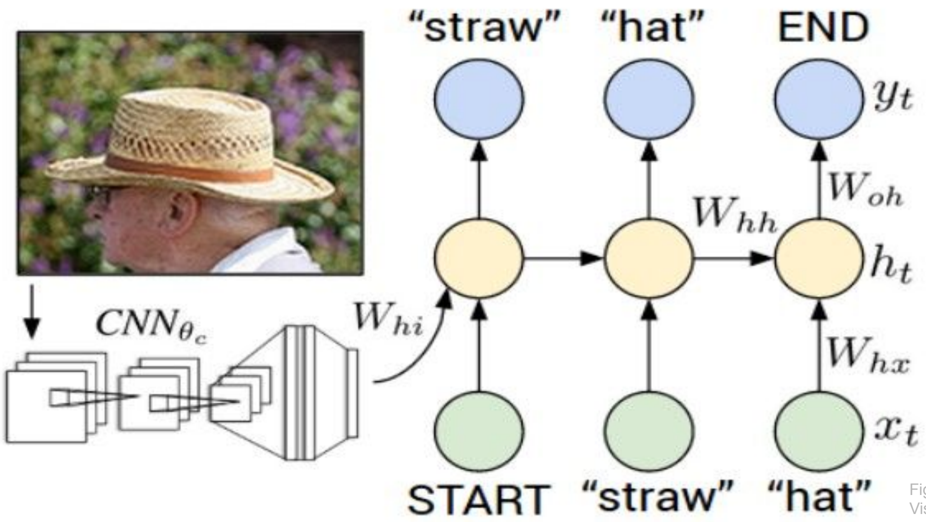
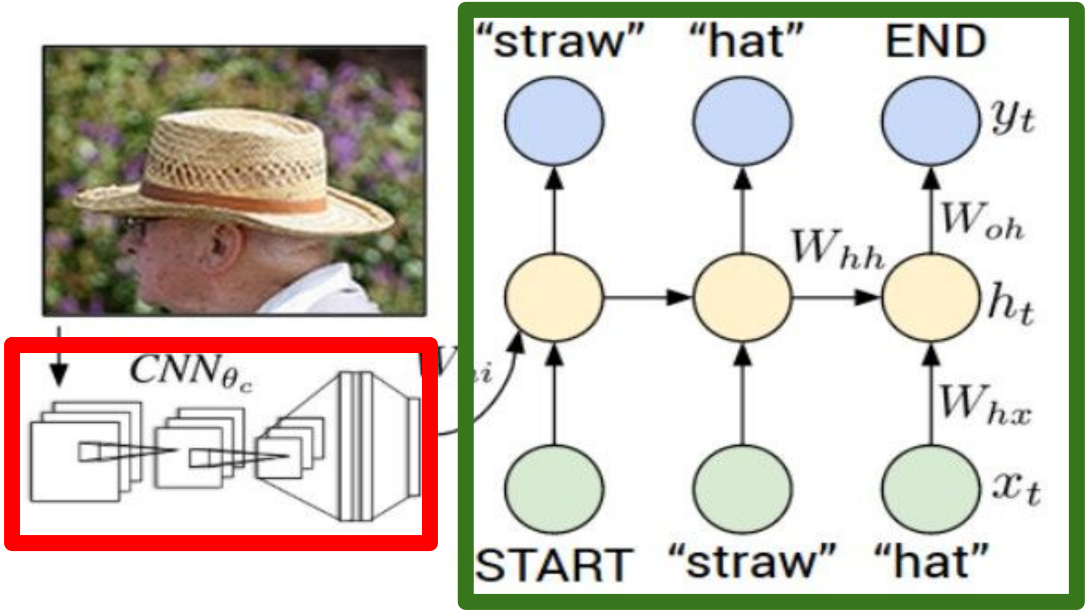


Figure from Karpathy et a, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure copyright IEEE, 2015.
Reproduced for educational purposes.

Explain Images with Multimodal Recurrent Neural Networks, Mao et al.
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei
Show and Tell: A Neural Image Caption Generator, Vinyals et al.
Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.
Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

Recurrent Neural Network



Convolutional Neural Network

test image


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000
softmax

test image


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
X

test image

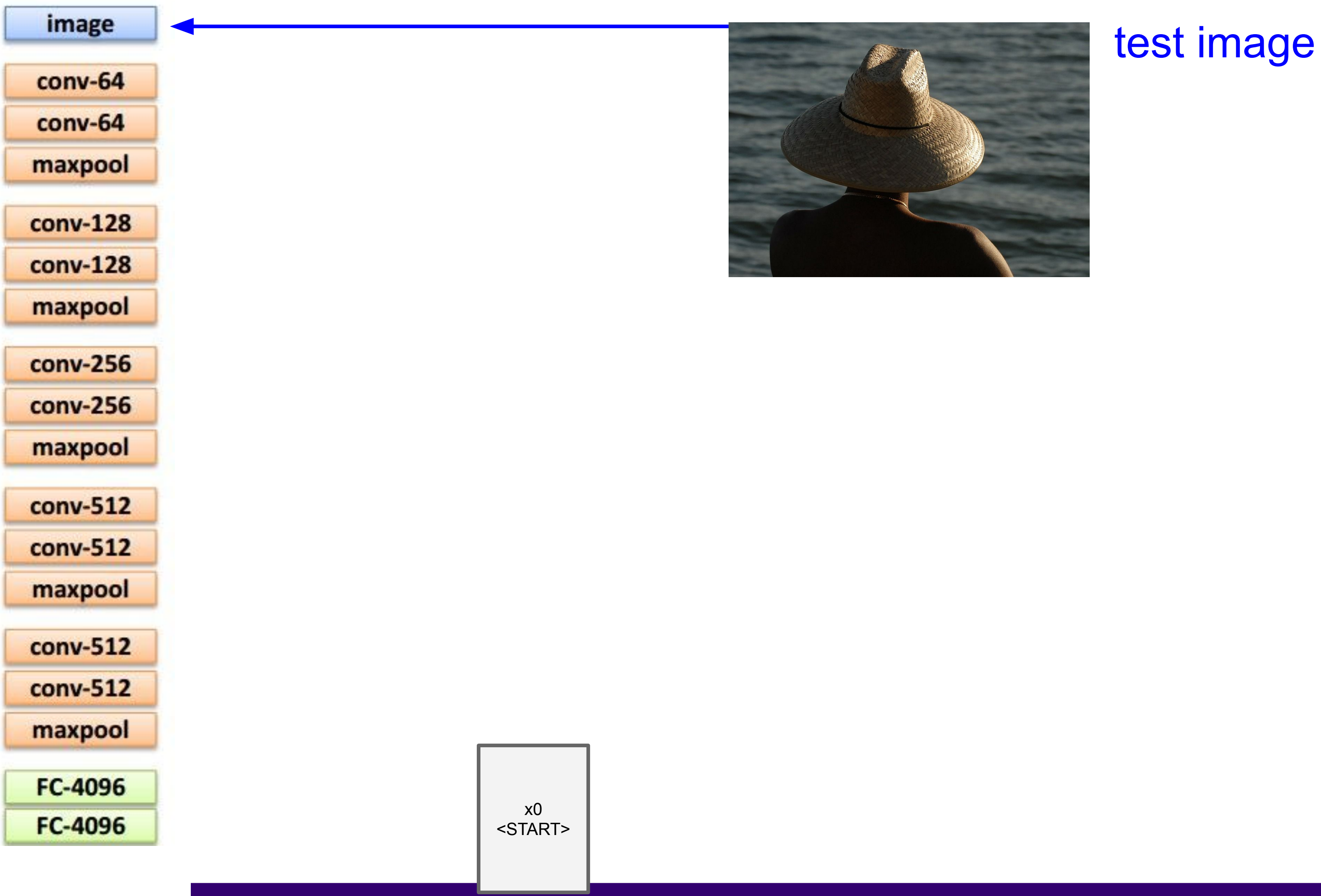
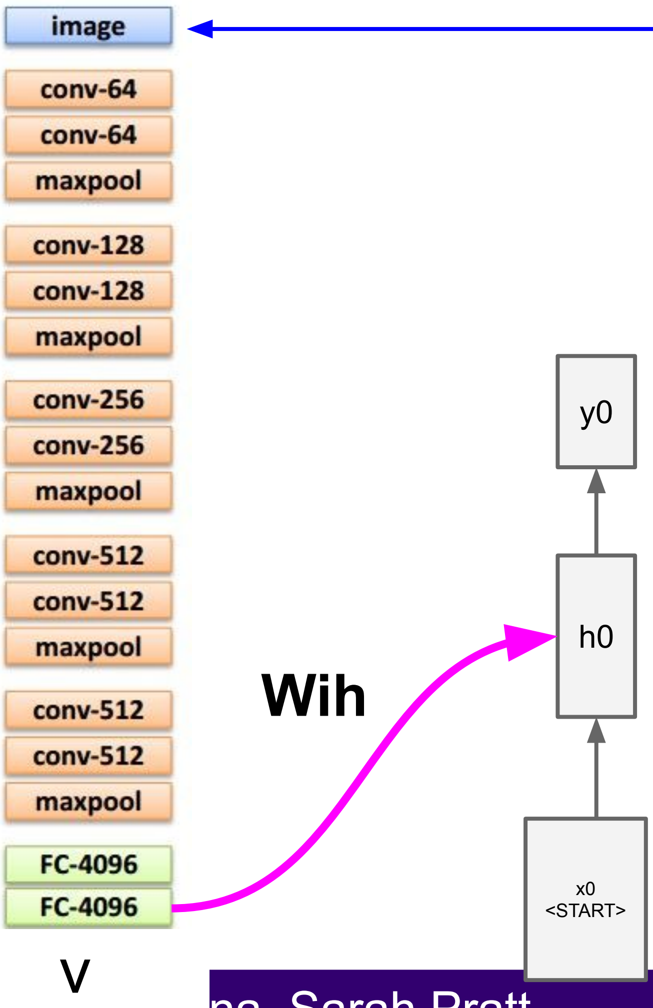


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
x0
<START>



test image

before:

$$h = \tanh(W_{xh} * x + W_{hh} * h)$$

now:

$$h = \tanh(W_{xh} * x + W_{hh} * h + W_{ih} * v)$$

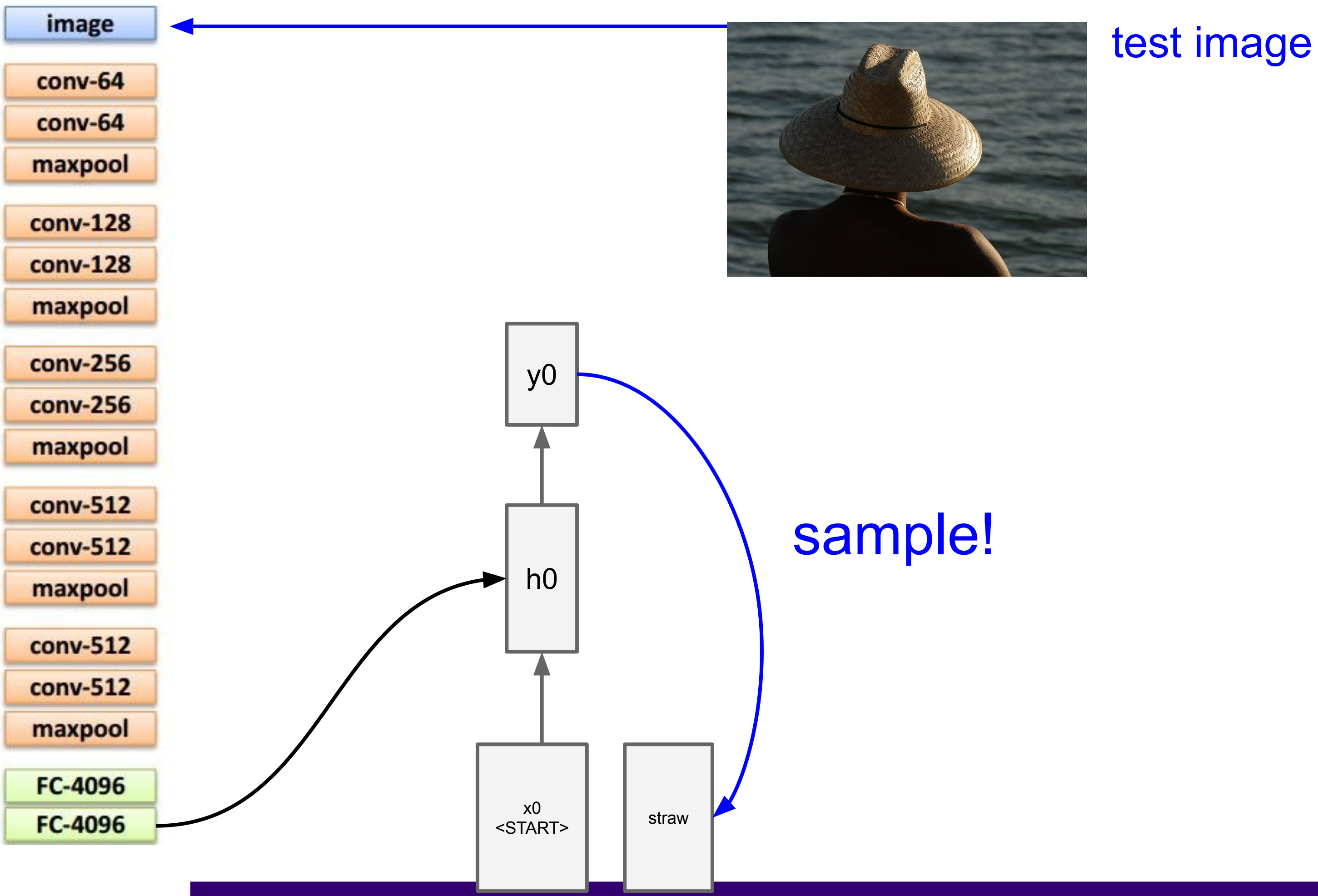


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
h0
sample!
x0
<START>
straw

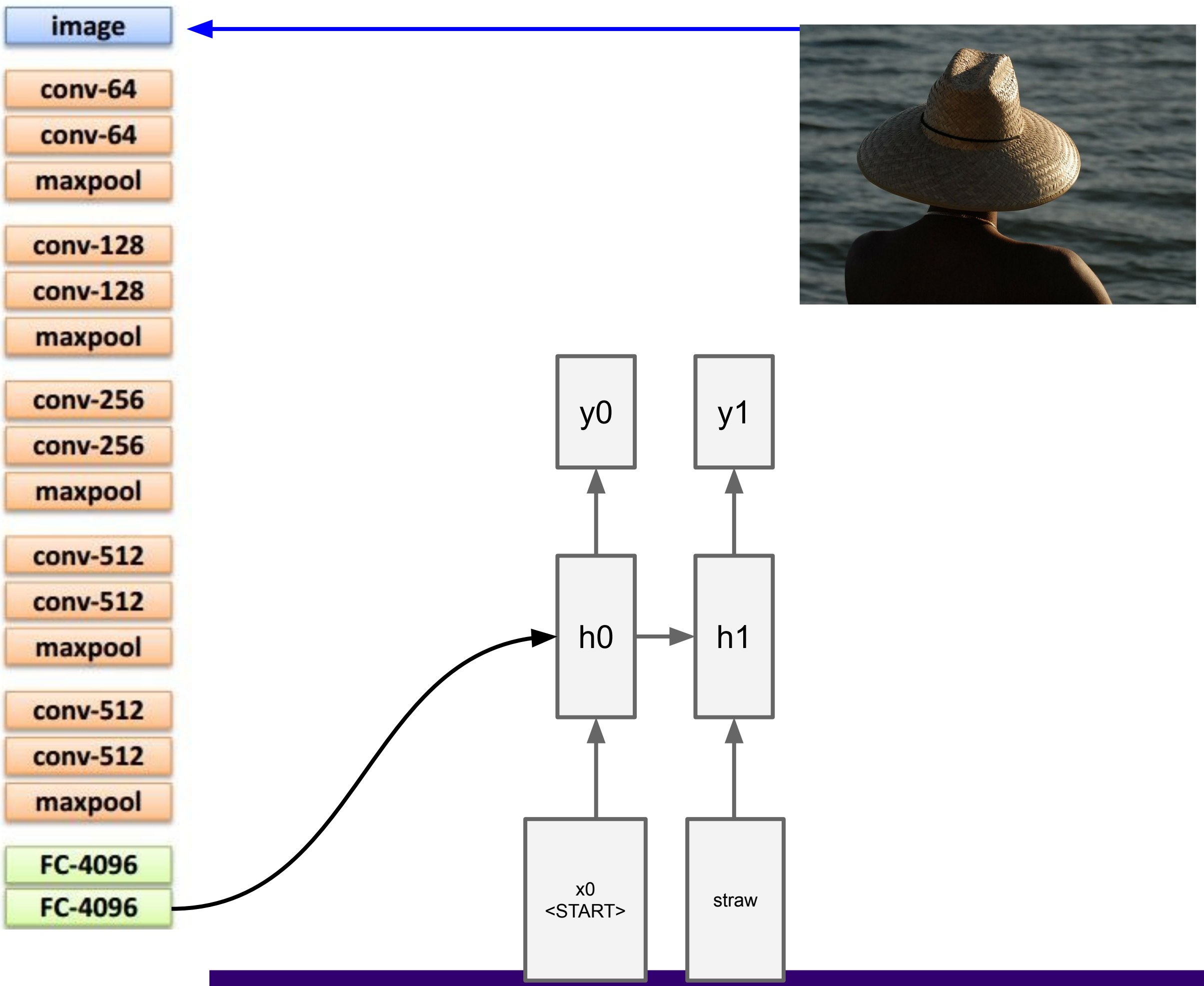


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
y0
y1
h0
h1
x0
<START>
straw

test image

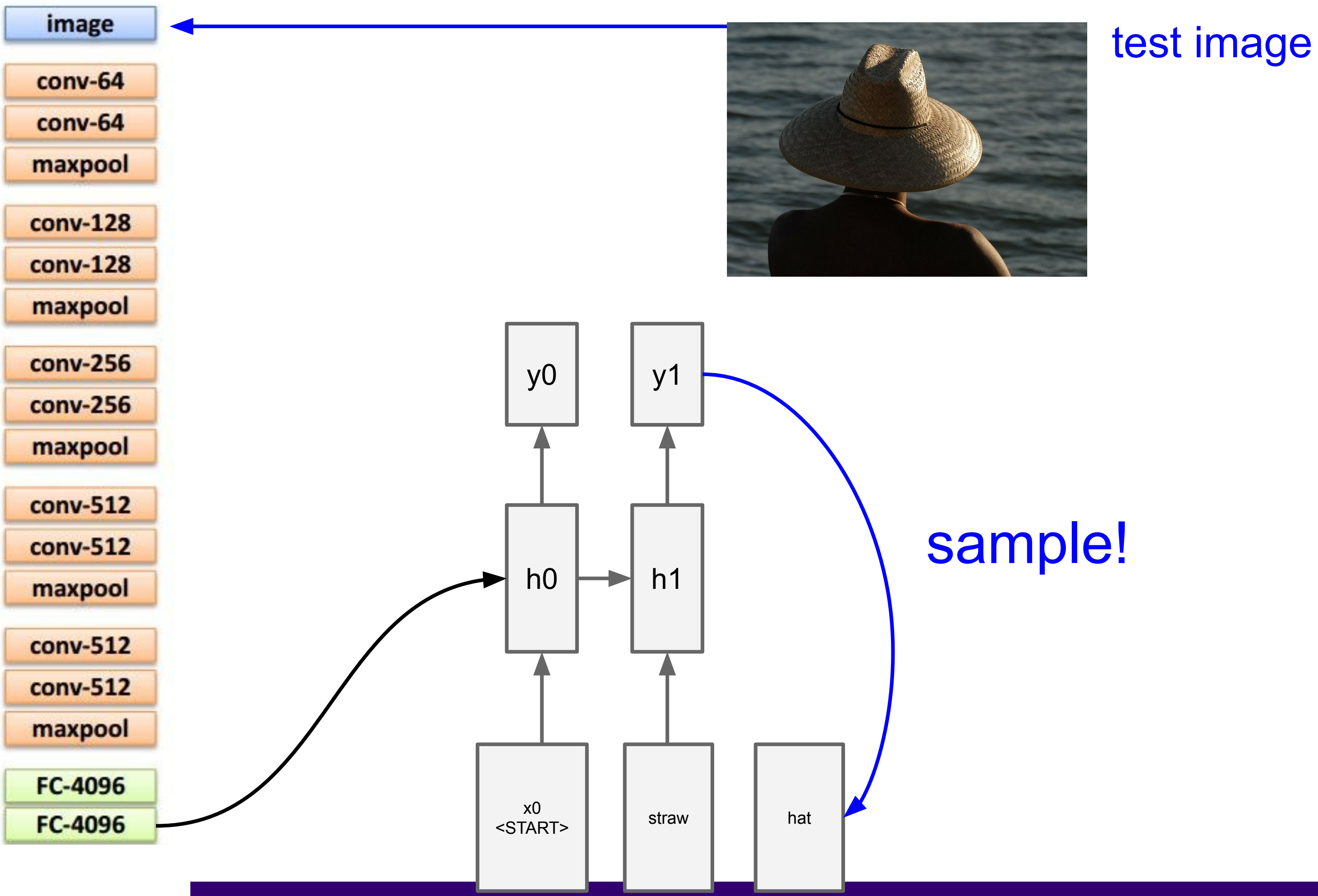


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
h0
h1
sample!
x0
<START>
straw
hat

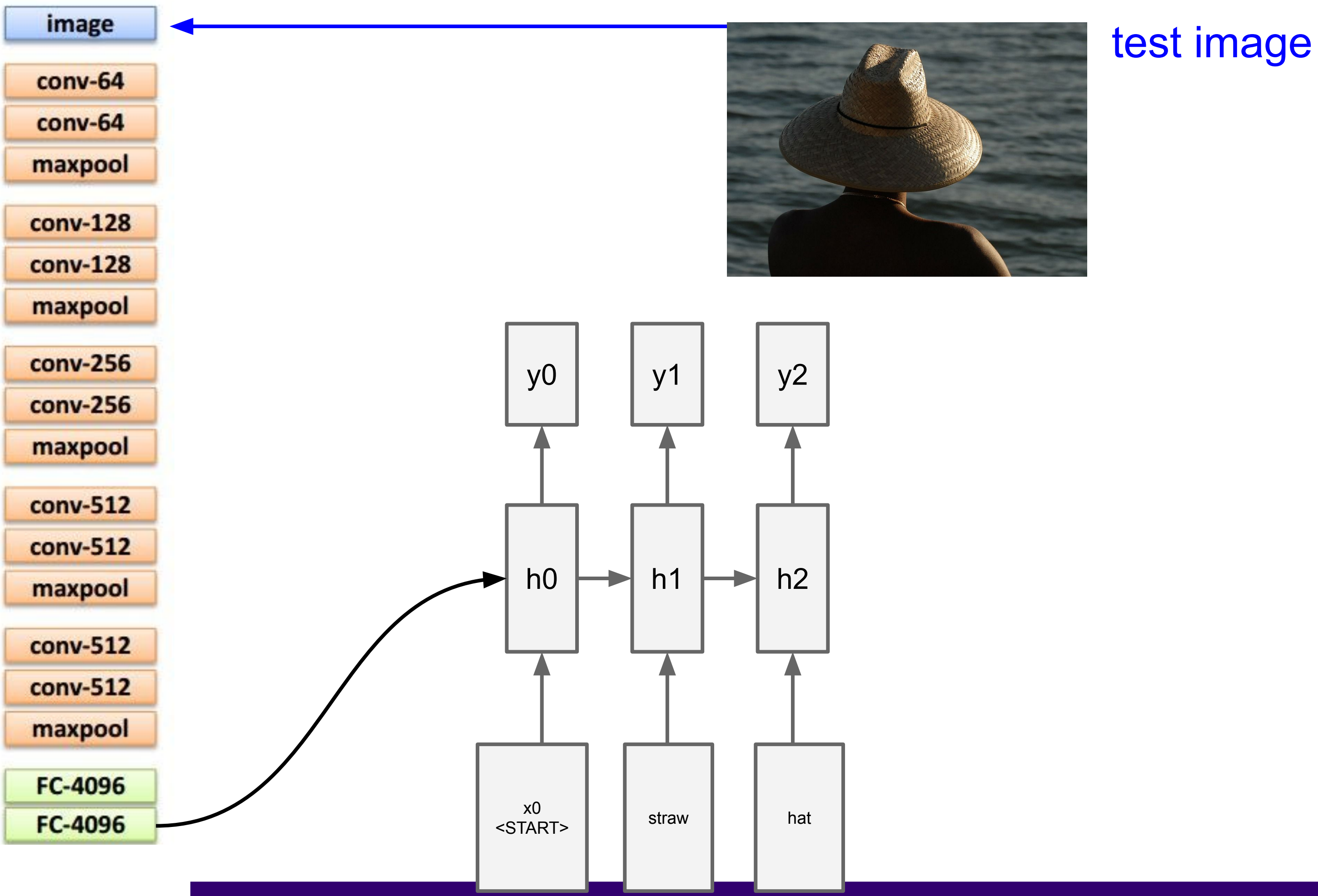
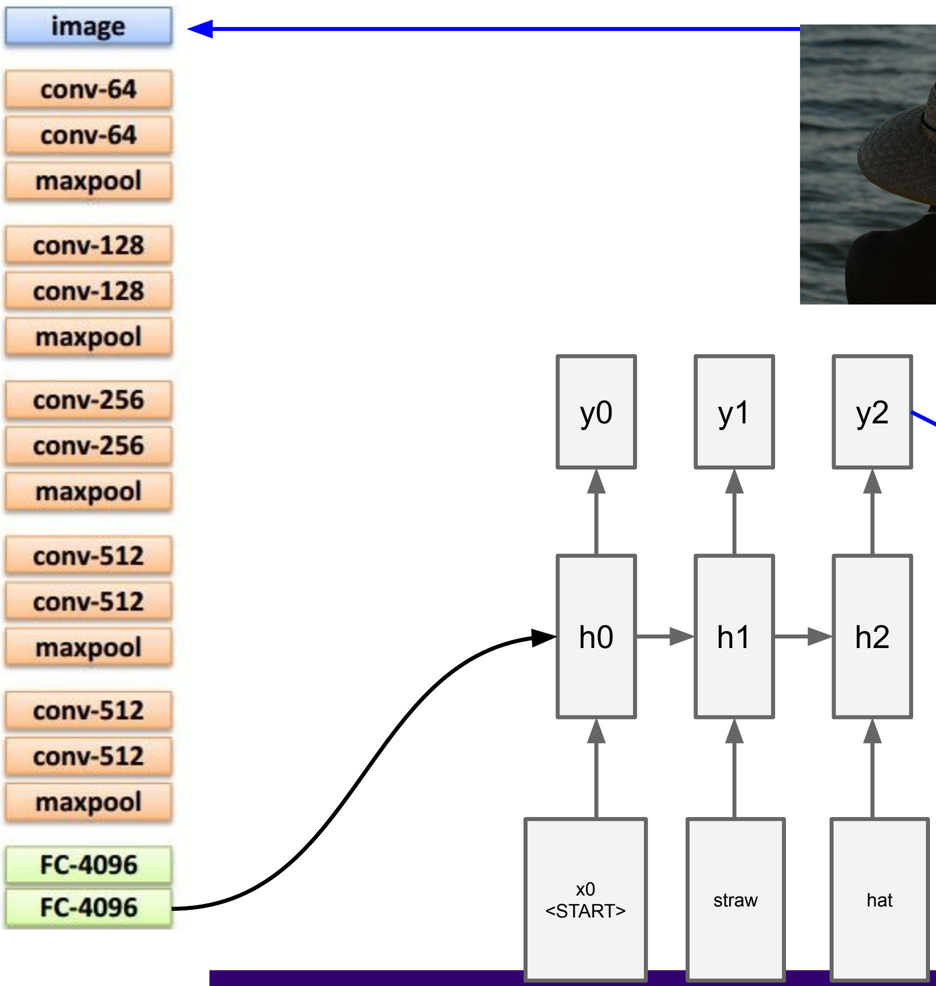


image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
test image
y0
y1
y2
h0
h1
h2
x0
<START>
straw
hat



test image

sample
<END> token
=> finish.

Image Captioning: Example Results

Captions generated using neuraltalk2
All images are CC0 Public domain: cat suitcase, cat tree, dog, bear, surfers, tennis, giraffe, motorcycle

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A dog is running in the grass with a frisbee

A white teddy bear sitting in the grass

Two people walking on the beach with surfboards

A tennis player in action on the court

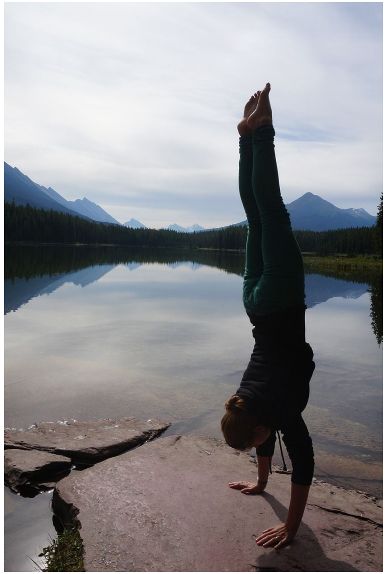
Two giraffes standing in a grassy field

A man riding a dirt bike on a dirt track

Image Captioning: Failure Cases

A woman is holding a cat in her hand

A person holding a computer mouse on a desk



A woman standing on a beach holding a surfboard

A bird is perched on a tree branch

A man in a baseball uniform throwing a ball

Visual Question Answering (VQA)

Q: What endangered animal is featured on the truck?

A: A bald eagle.
A: A sparrow.
A: A humming bird.
A: A raven.

Q: Where will the driver go if turning right?

A: Onto 24 ¾ Rd.
A: Onto 25 ¾ Rd.
A: Onto 23 ¾ Rd.
A: Onto Main Street.

Q: When was the picture taken?

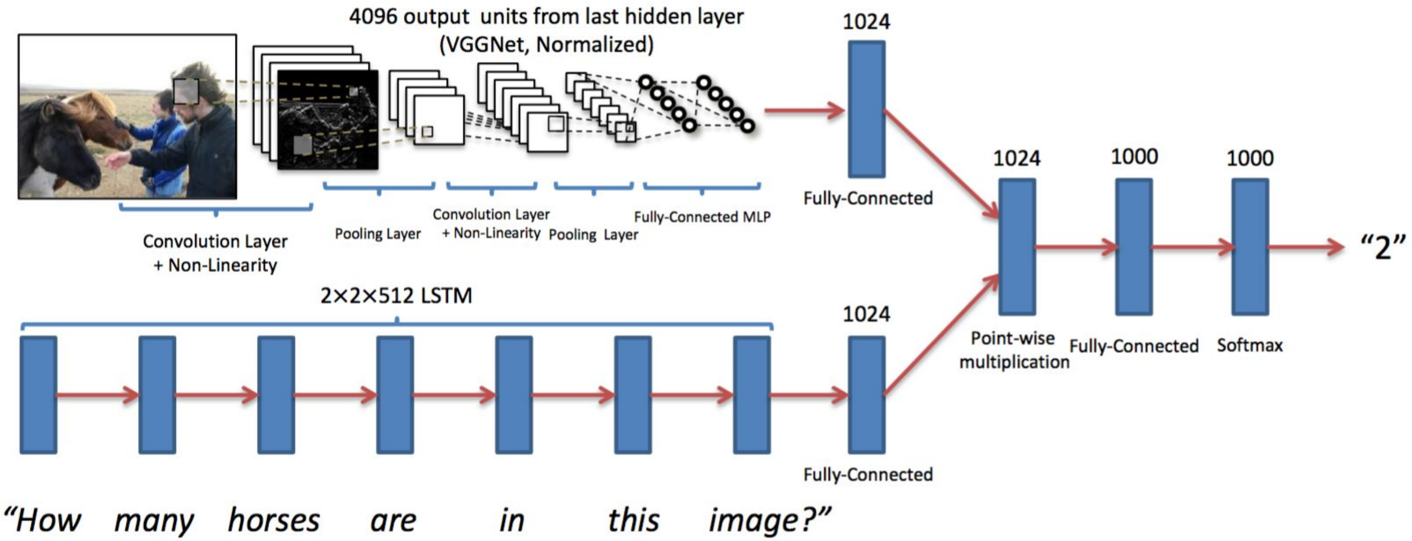
A: During a wedding.
A: During a bar mitzvah.
A: During a funeral.
A: During a Sunday church service

Q: Who is under the umbrella?

A: Two women.
A: A child.
A: An old man.
A: A husband and a wife.

Agrawal et al, "VQA: Visual Question Answering", ICCV 2015
Zhu et al, "Visual 7W: Grounded Question Answering in Images", CVPR 2016

Visual Question Answering: RNNs with Attention



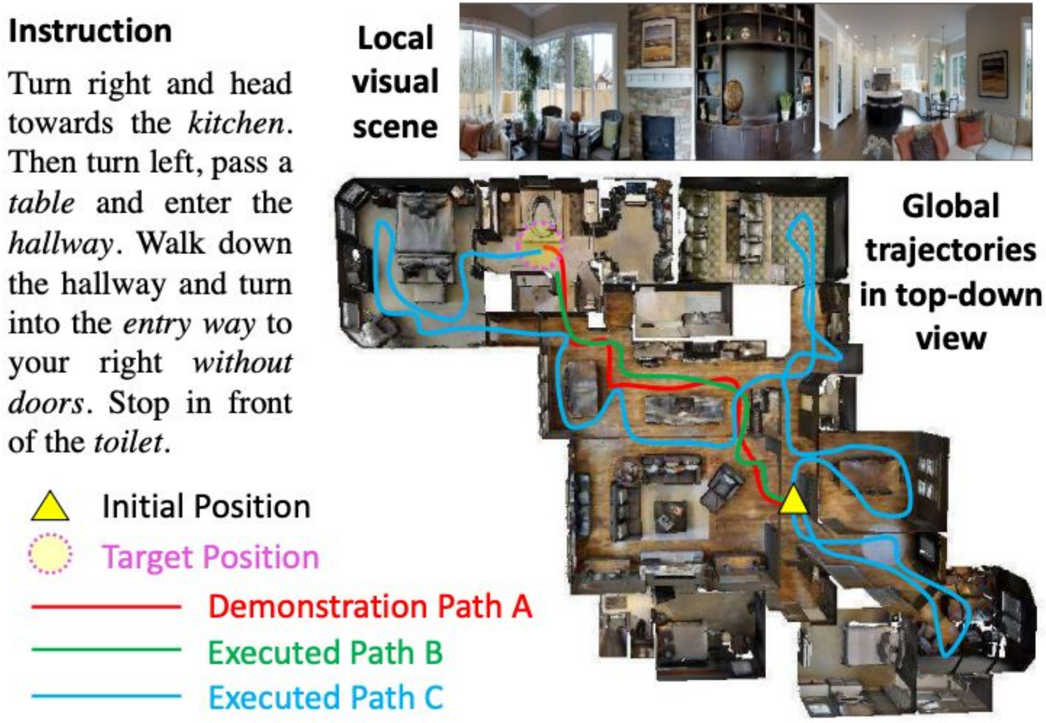
Agrawal et al, "Visual 7W: Grounded Question Answering in Images", CVPR 2015

Visual Dialog: Conversations about images

Das et al, “Visual Dialog”, CVPR 2017

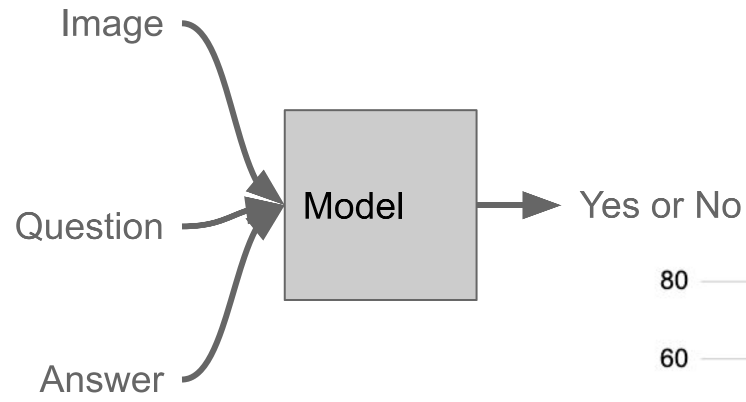
Visual Language Navigation: Go to the living room

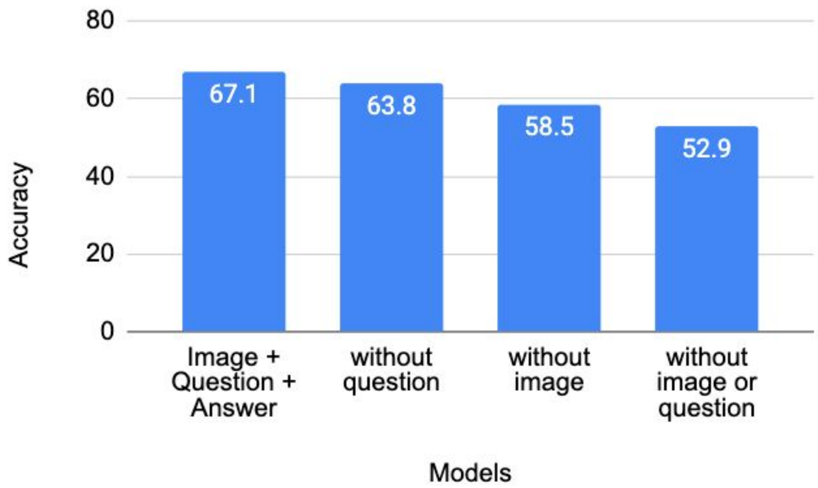
Agent encodes instructions in language and uses an RNN to generate a series of movements as the visual input changes after each move.



Wang et al, "Reinforced Cross-Modal Matching and Self-Supervised Imitation Learning for Vision-Language Navigation", CVPR 2018

Visual Question Answering: Dataset Bias



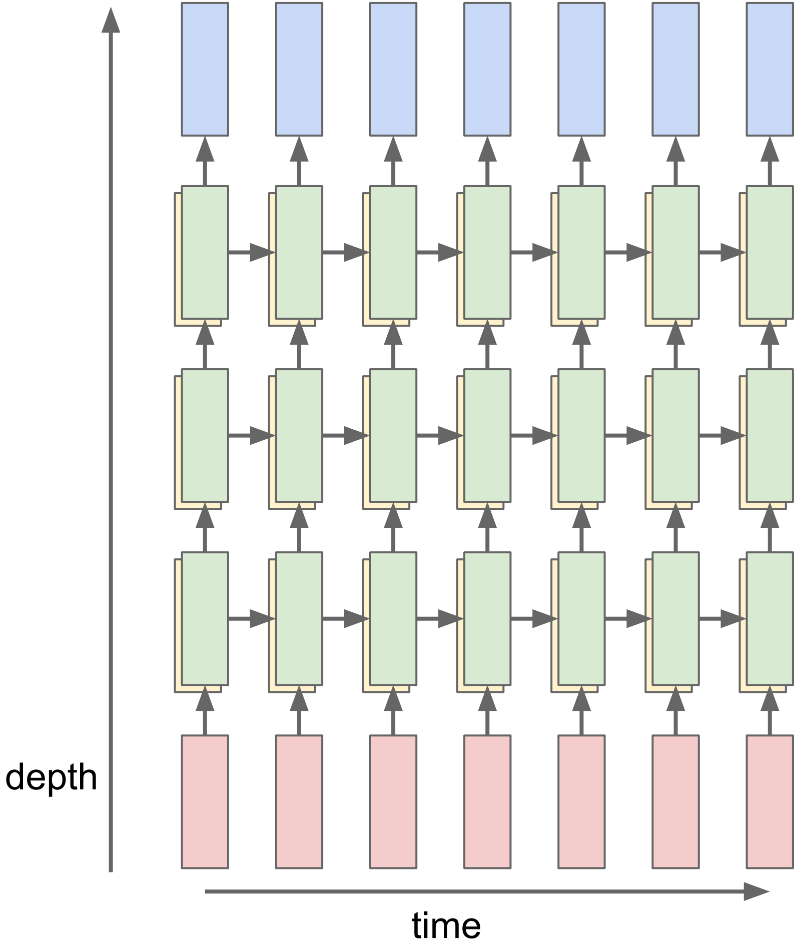


Jabri et al. "Revisiting Visual Question Answering Baselines" ECCV 2016

Multilayer RNNs

Each layer has a different set of weights

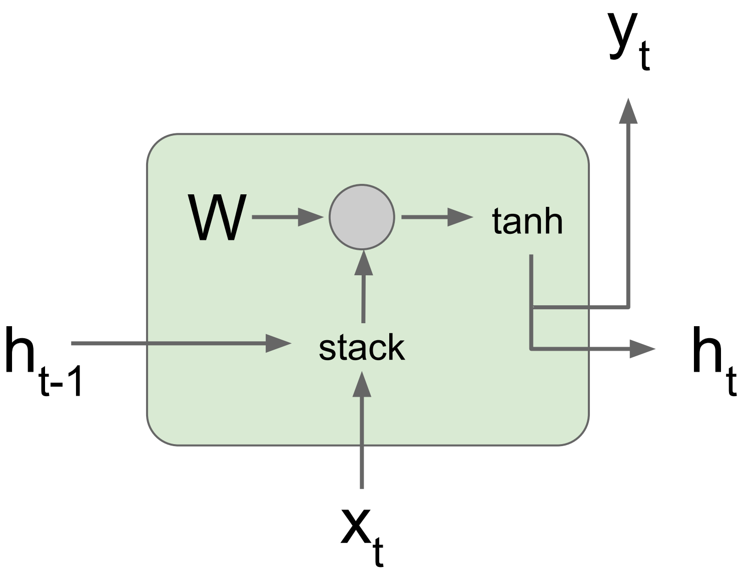
Outputs from one layer become inputs to the layer above.



Now, let's talk about why RNNs are not as popular anymore.

Vanilla RNN Gradient Flow

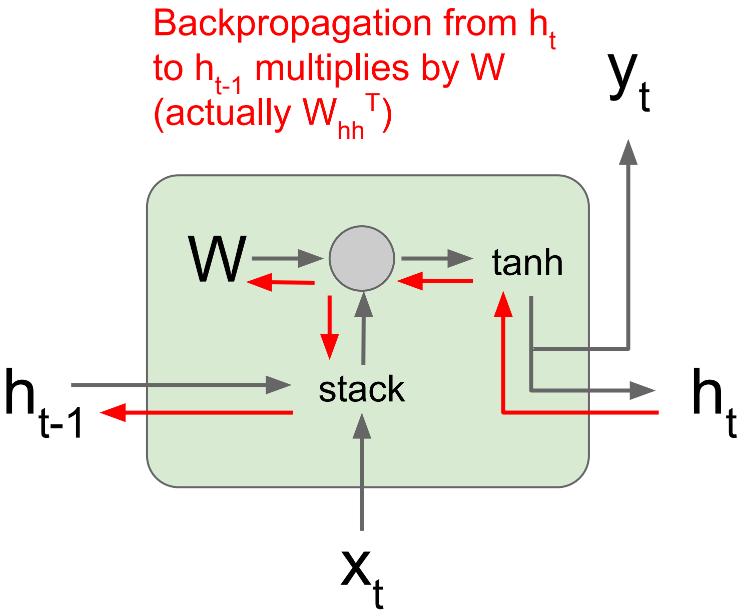
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

Vanilla RNN Gradient Flow

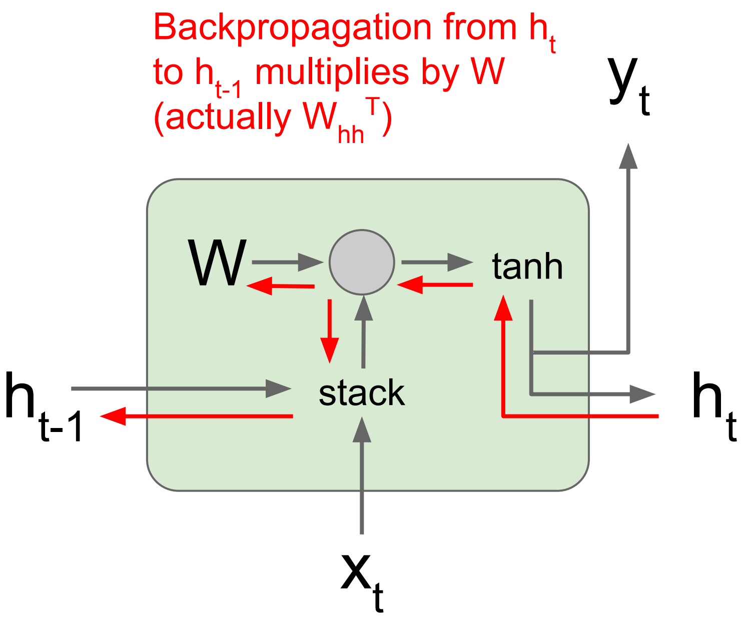
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left(\begin{pmatrix}W_{hh} & W_{hx}\end{pmatrix}\begin{pmatrix}h_{t-1}\\ x_t\end{pmatrix}\right)$$
$$= \tanh\left(W\begin{pmatrix}h_{t-1}\\ x_t\end{pmatrix}\right)$$

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

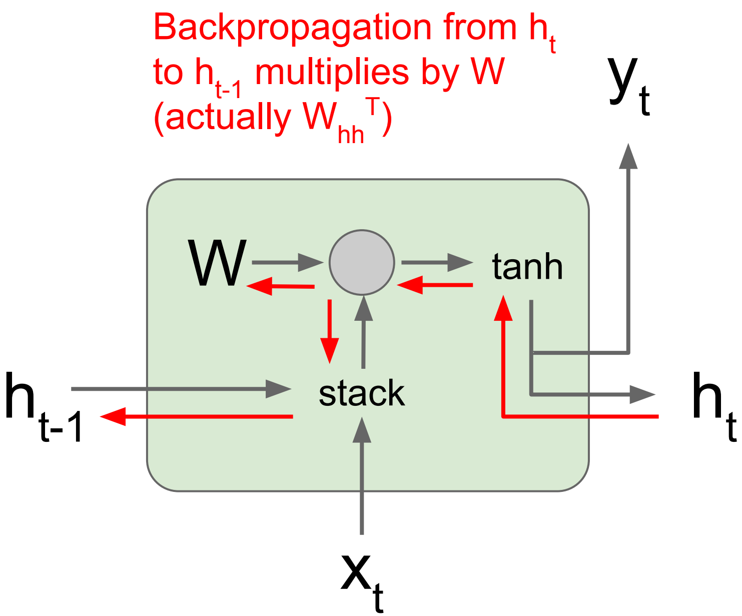


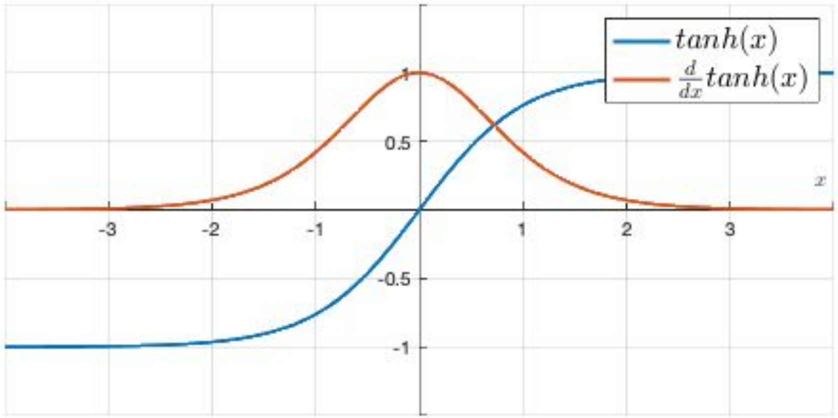
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$= \tanh\left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$
$$= \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

$$\frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}$$

Vanilla RNN Gradient Flow

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013

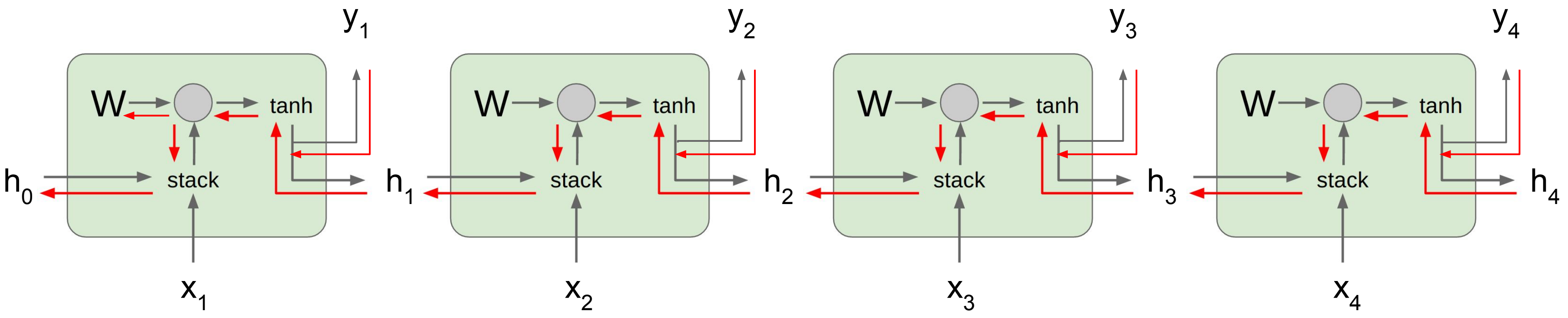




$$\frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}$$

Vanilla RNN Gradient Flow

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013

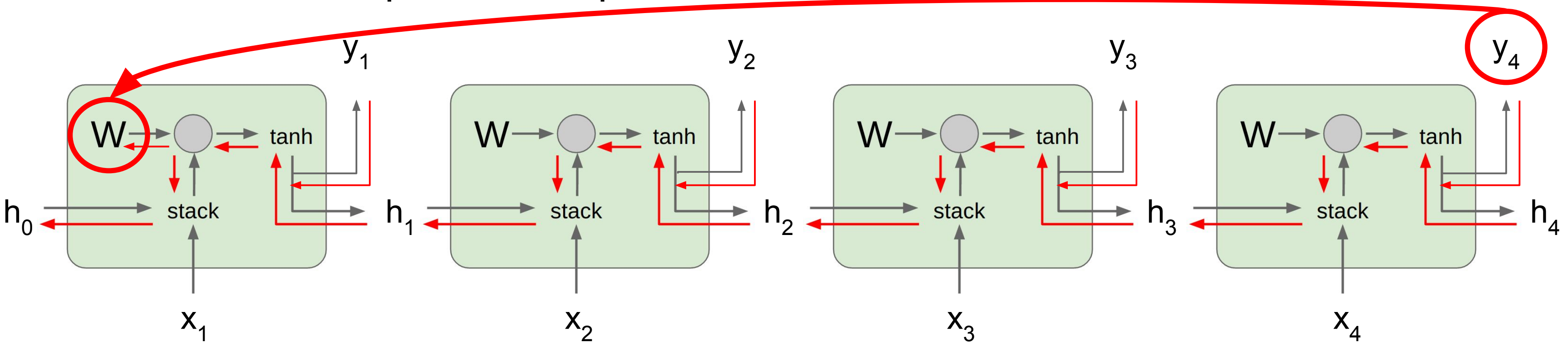


$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Gradients over multiple time steps:



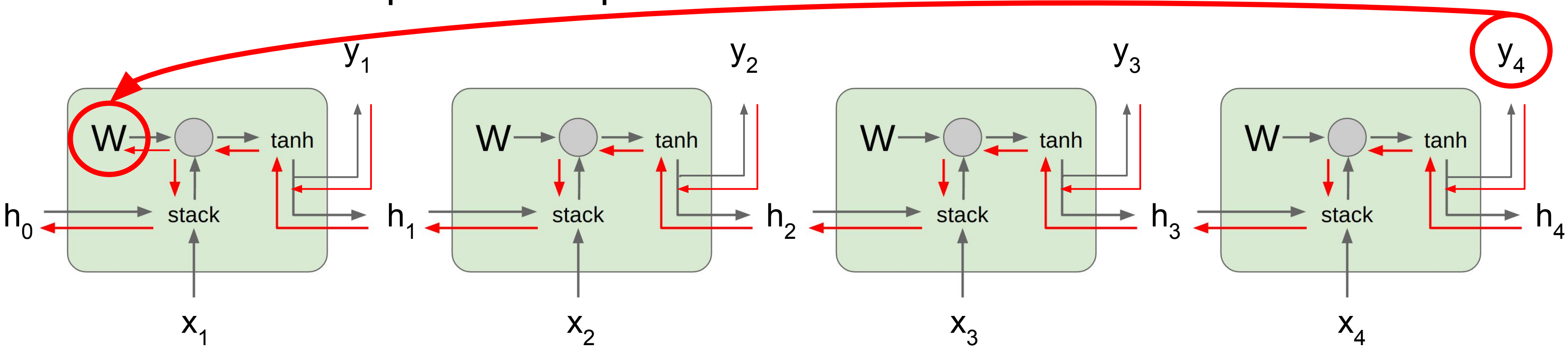
$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W}$$

Vanilla RNN Gradient Flow

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013

Gradients over multiple time steps:



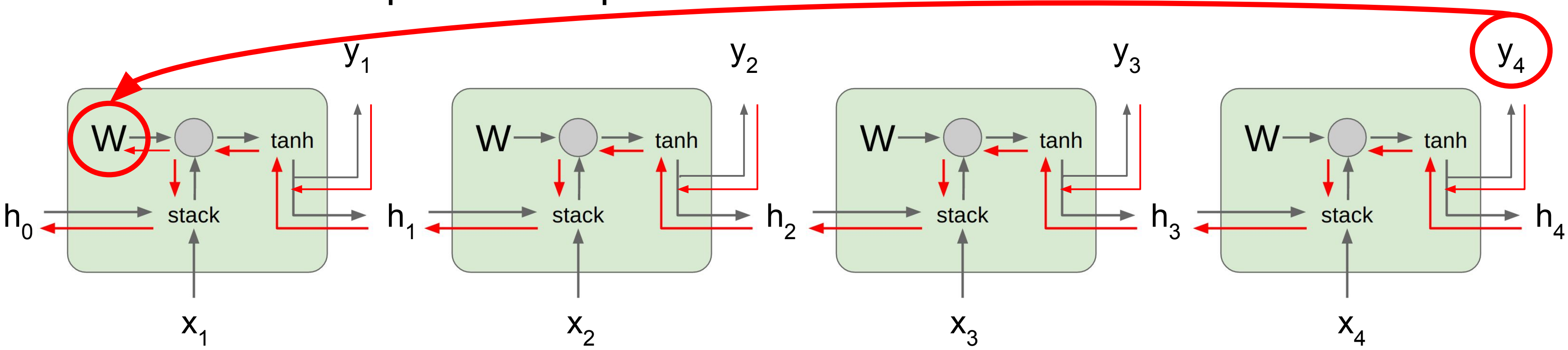
$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}}\right) \frac{\partial h_1}{\partial W}$$

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Gradients over multiple time steps:



$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

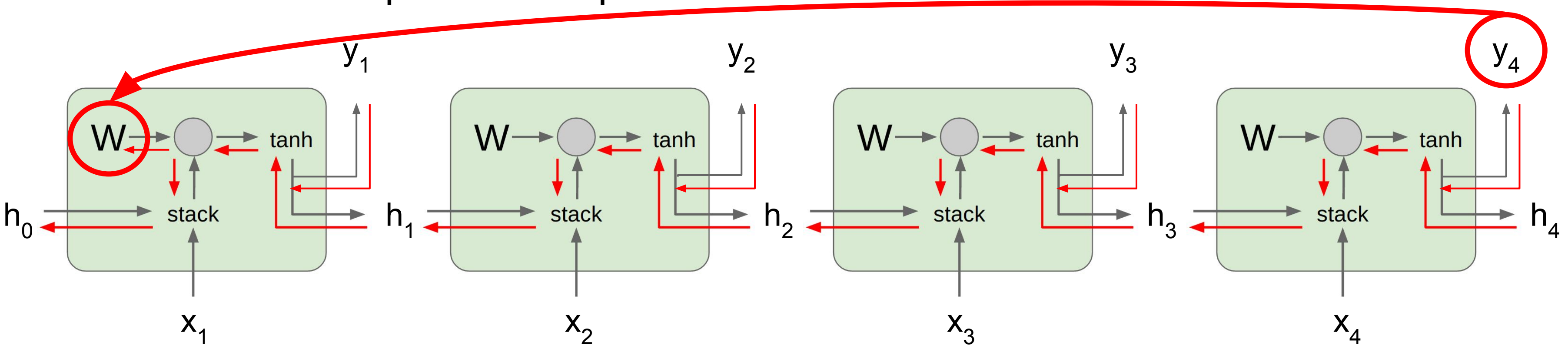
$$\frac{\partial h_t}{\partial h_{t-1}} = tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^{T} \frac{\partial h_t}{\partial h_{t-1}}\right) \frac{\partial h_1}{\partial W}$$

Vanilla RNN Gradient Flow

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013

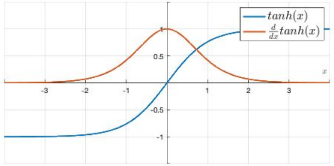
Gradients over multiple time steps:



$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

Almost always < 1

Vanishing gradients

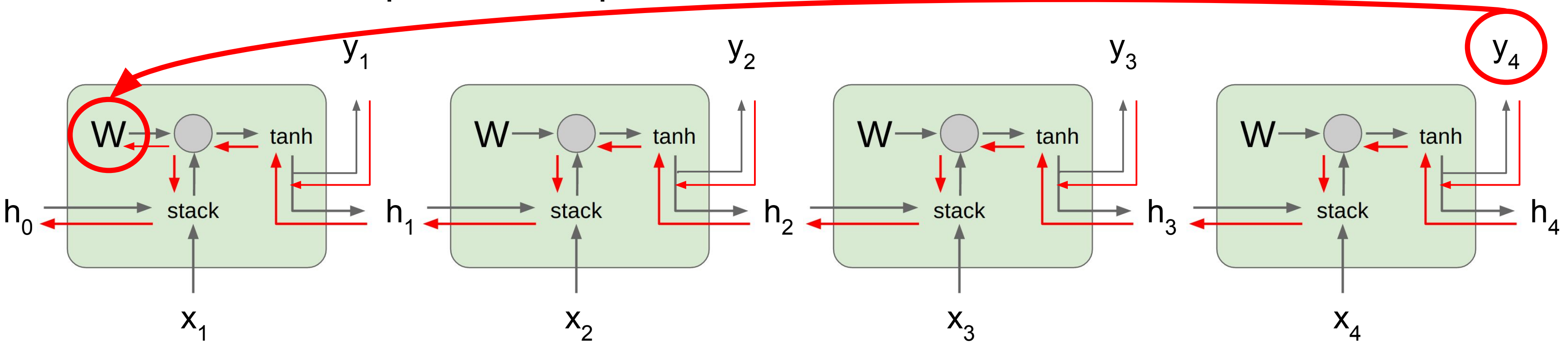


$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^{T} \boxed{tanh'(W_{hh}h_{t-1} + W_{xh}x_t)}\right) W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

Vanilla RNN Gradient Flow

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013

Gradients over multiple time steps:



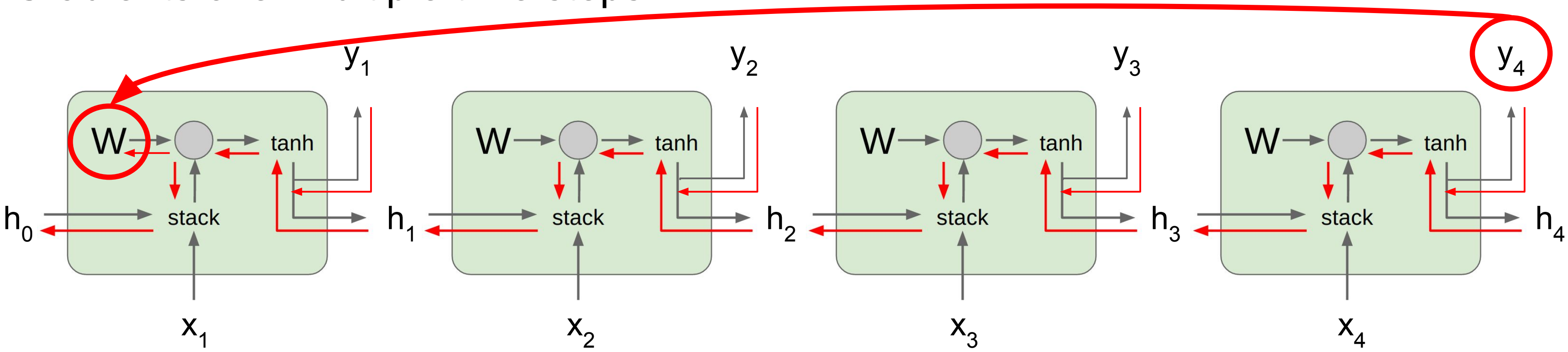
$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

What if we assumed no non-linearity?

Vanilla RNN Gradient Flow

Bengio et al, “Learning long-term dependencies with gradient descent is difficult”, IEEE Transactions on Neural Networks, 1994
Pascanu et al, “On the difficulty of training recurrent neural networks”, ICML 2013

Gradients over multiple time steps:



What if we assumed no non-linearity?

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \boxed{W_{hh}^{T-1}} \frac{\partial h_1}{\partial W}$$

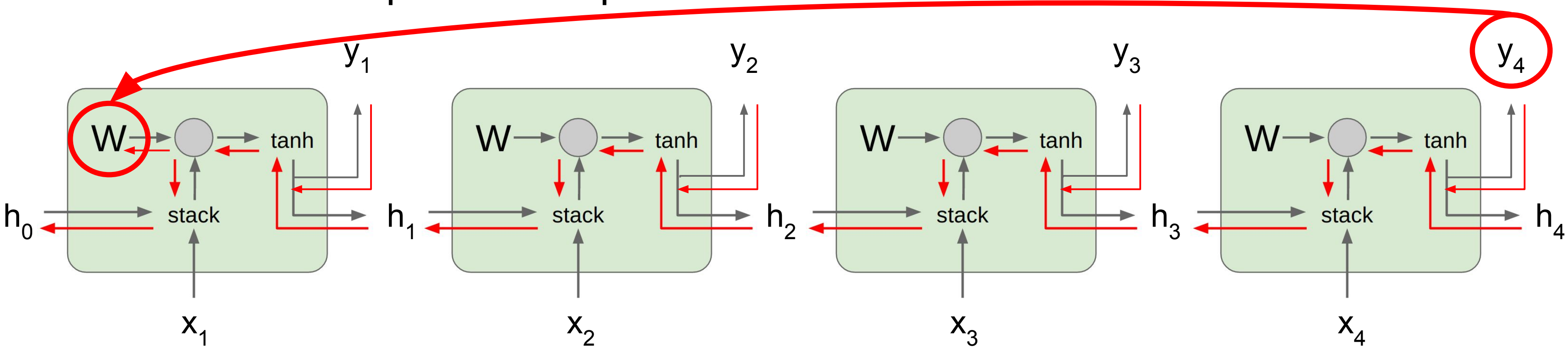
Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

Gradients over multiple time steps:



What if we assumed no non-linearity?

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

Largest singular value > 1:
Exploding gradients

→ **Gradient clipping**: Scale gradient if its norm is too big

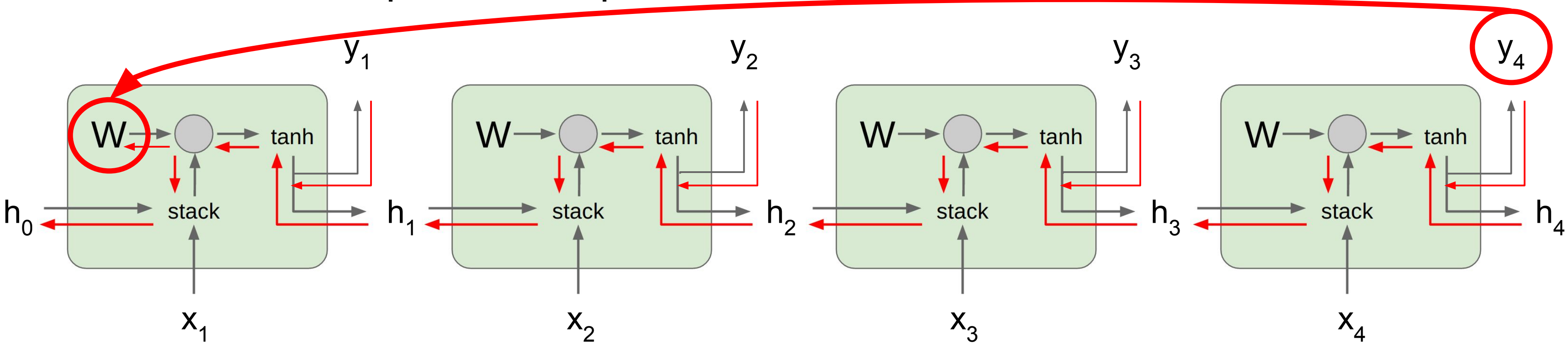
```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
  grad *= (threshold / grad_norm)
```

Largest singular value < 1:
Vanishing gradients

Vanilla RNN Gradient Flow

Gradients over multiple time steps:

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



What if we assumed no non-linearity?

$$\frac{\partial L}{\partial W} = \sum_{t=1}^{T} \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

Largest singular value > 1:
Exploding gradients

Largest singular value < 1:
Vanishing gradients

→ Change RNN architecture

Long Short Term Memory (LSTM)

Vanilla RNN

$$h_t = \tanh\left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}\right)$$

LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997

RNNs have a single hidden state (h_t)
LSTMs have two: cell memory c_t and hidden state h_t

Vanilla RNN

$$h_t = \tanh\left(W\begin{pmatrix}h_{t-1}\\x_t\end{pmatrix}\right)$$

LSTM

$$\begin{pmatrix}i\\f\\o\\g\end{pmatrix} = \begin{pmatrix}\sigma\\\sigma\\\sigma\\\tanh\end{pmatrix} W \begin{pmatrix}h_{t-1}\\x_t\end{pmatrix}$$

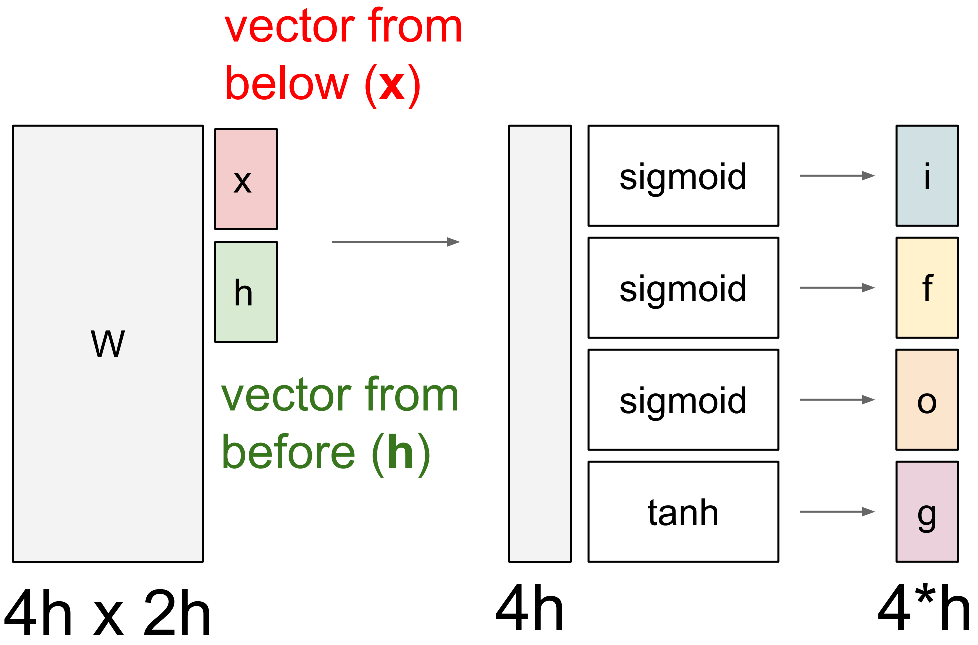
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Hochreiter and Schmidhuber, “Long Short Term Memory”, Neural Computation 1997

Long Short Term Memory (LSTM)

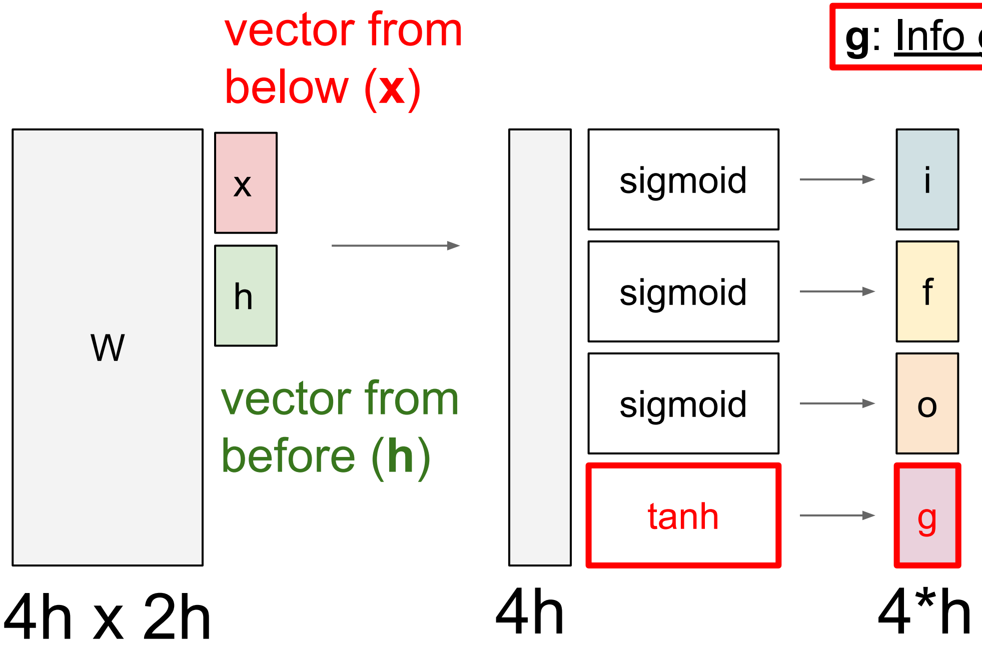
[Hochreiter et al., 1997]



Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

g: Info gate, How much to write to cell



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

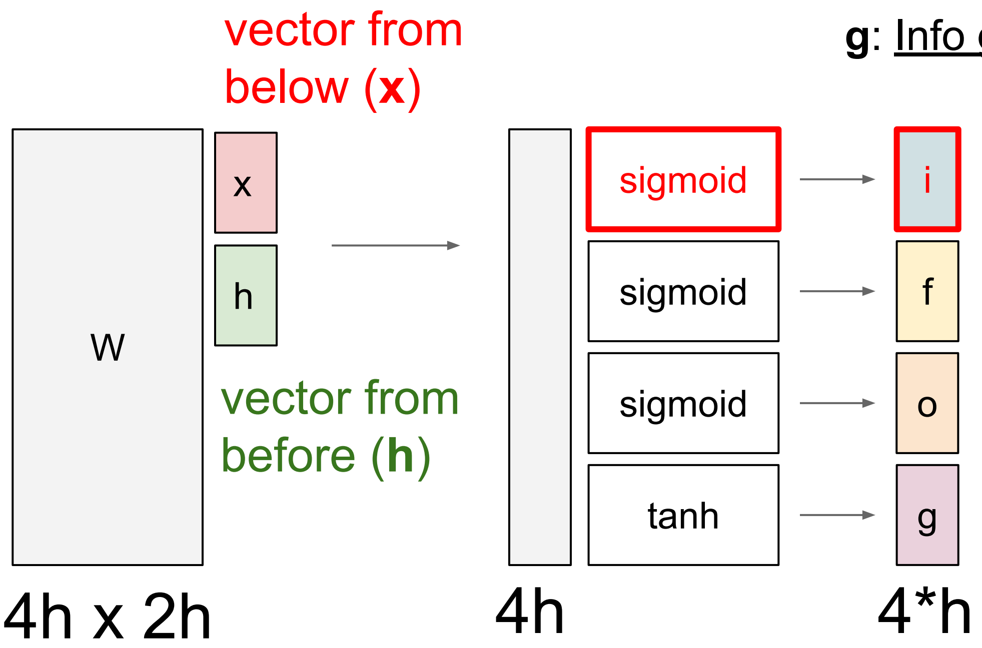
$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

i: Input gate, whether to write to cell

g: Info gate, How much to write to cell



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

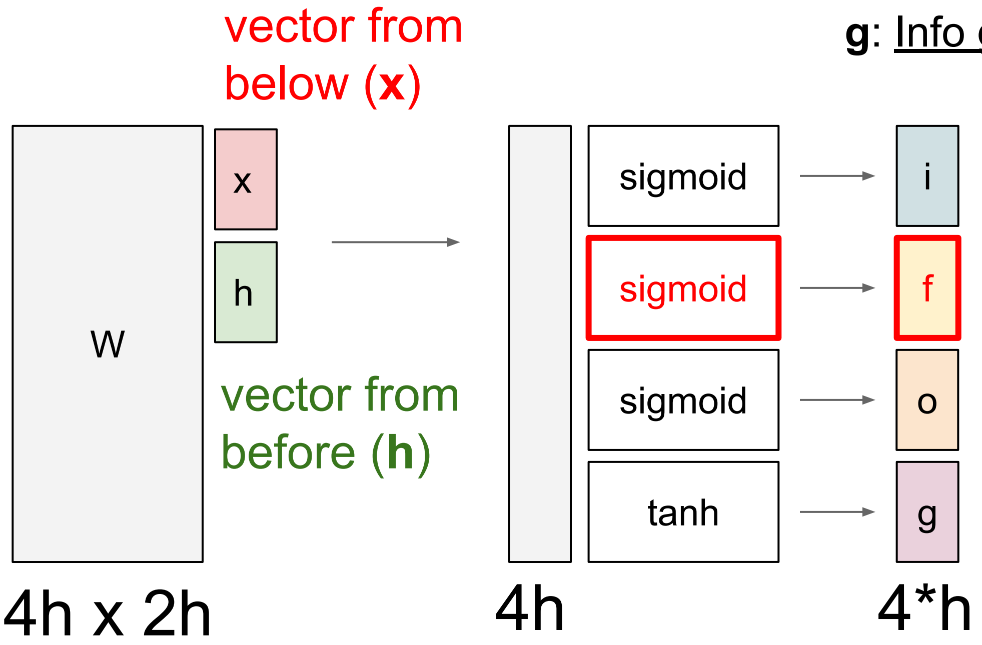
Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

i: Input gate, whether to write to cell

f: Forget gate, Whether to erase cell

g: Info gate, How much to write to cell



$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

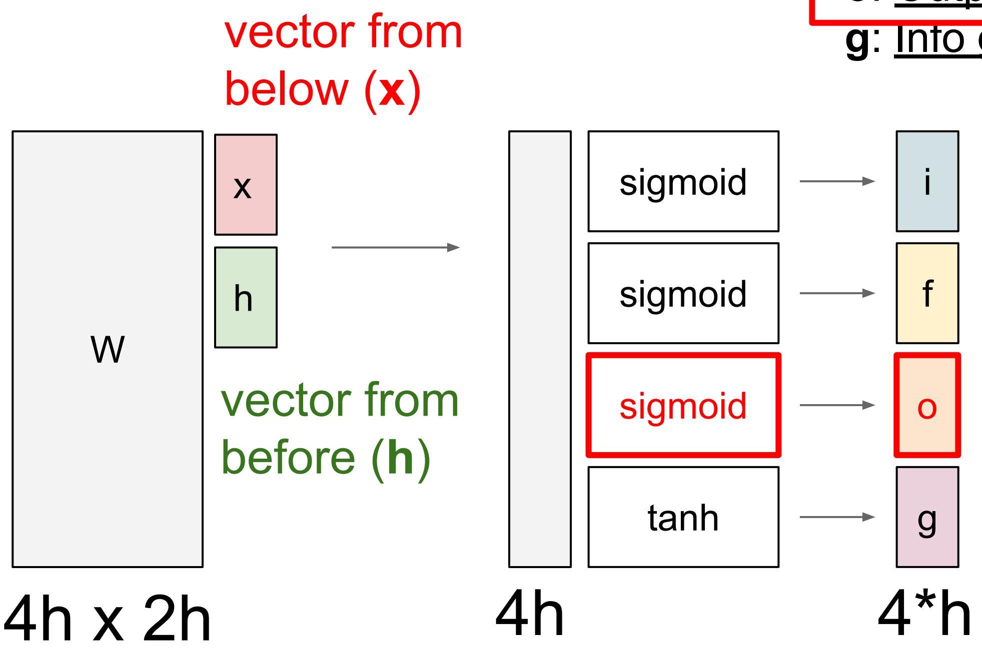
$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

i: Input gate, whether to write to cell
f: Forget gate, Whether to erase cell
o: Output gate, How much to reveal cell
g: Info gate, How much to write to cell



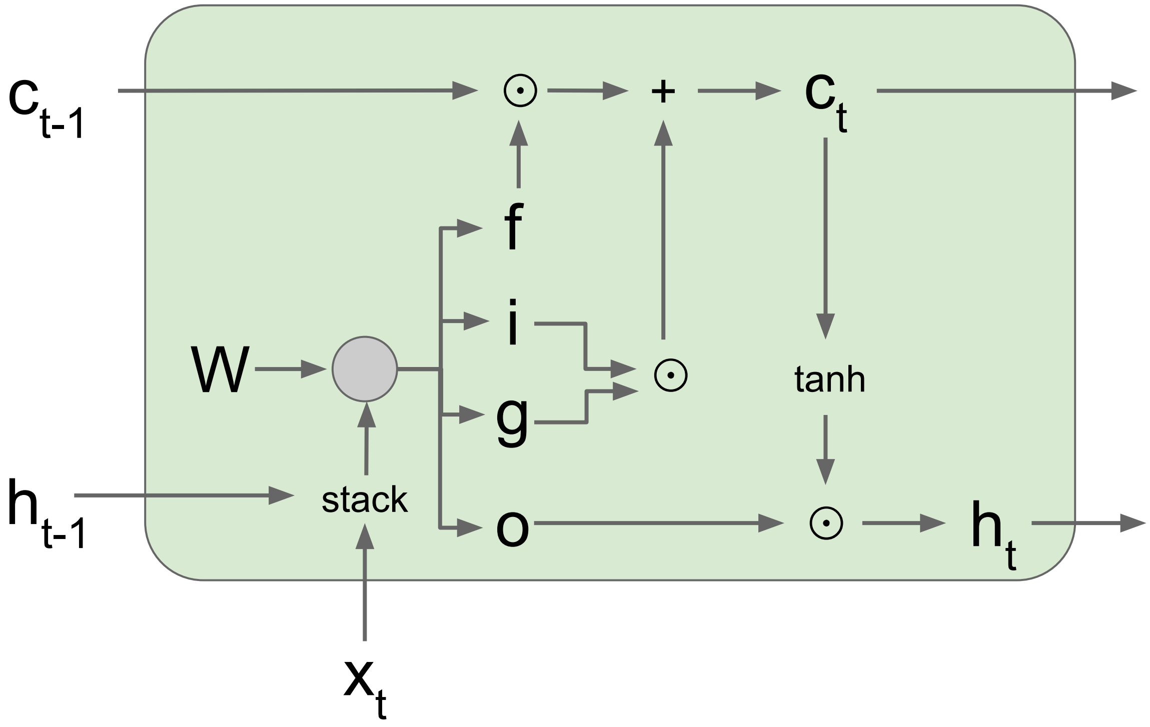
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



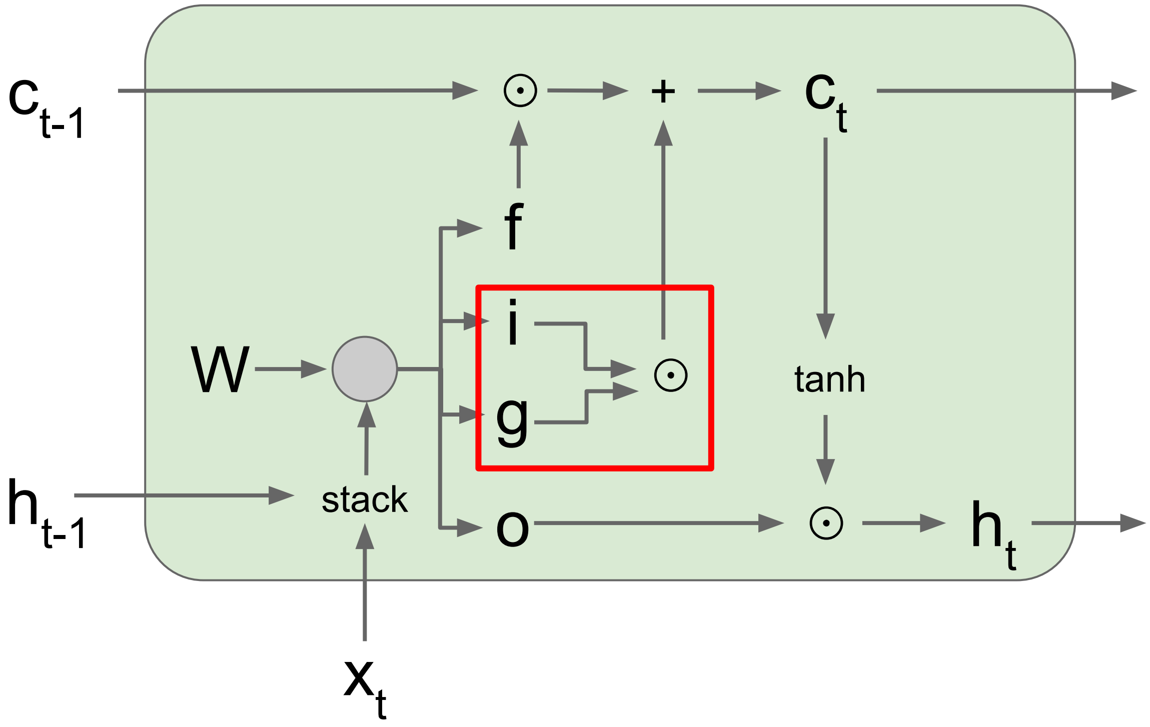
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



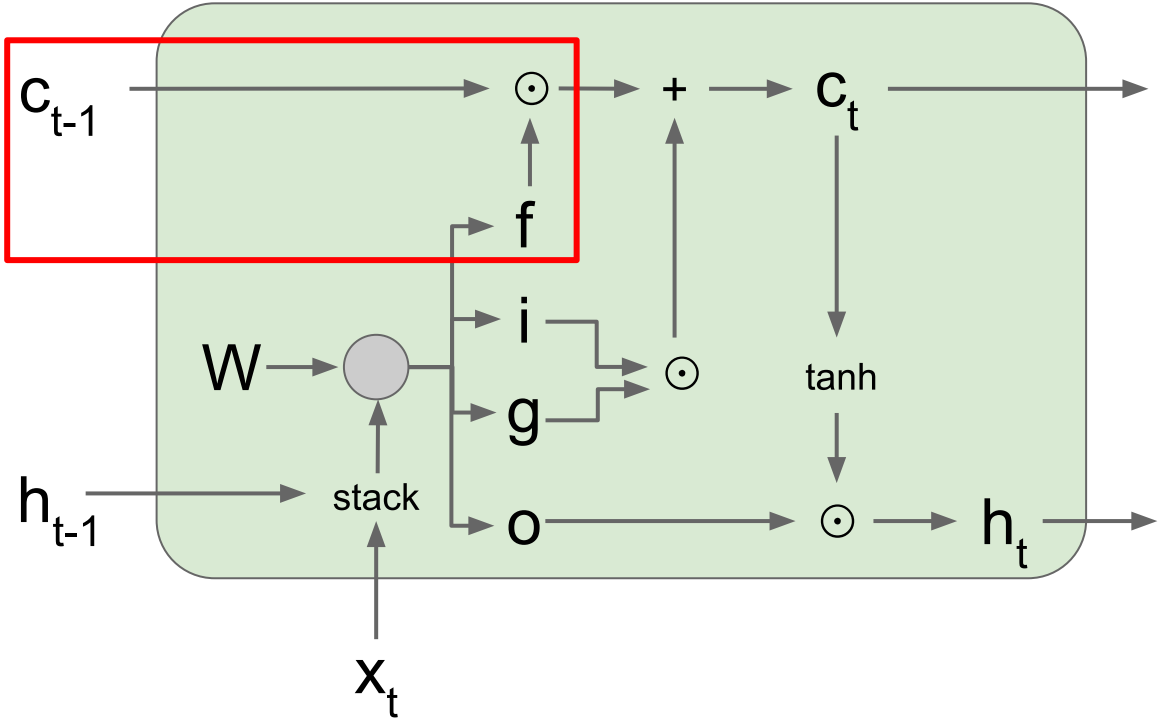
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



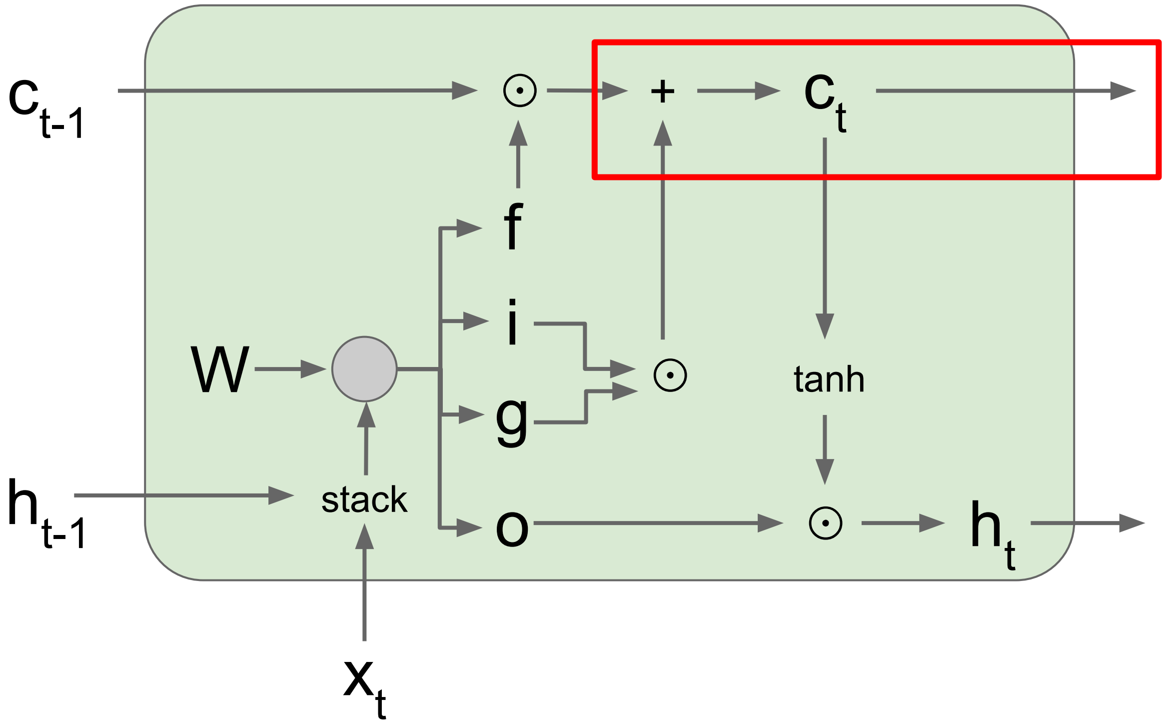
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



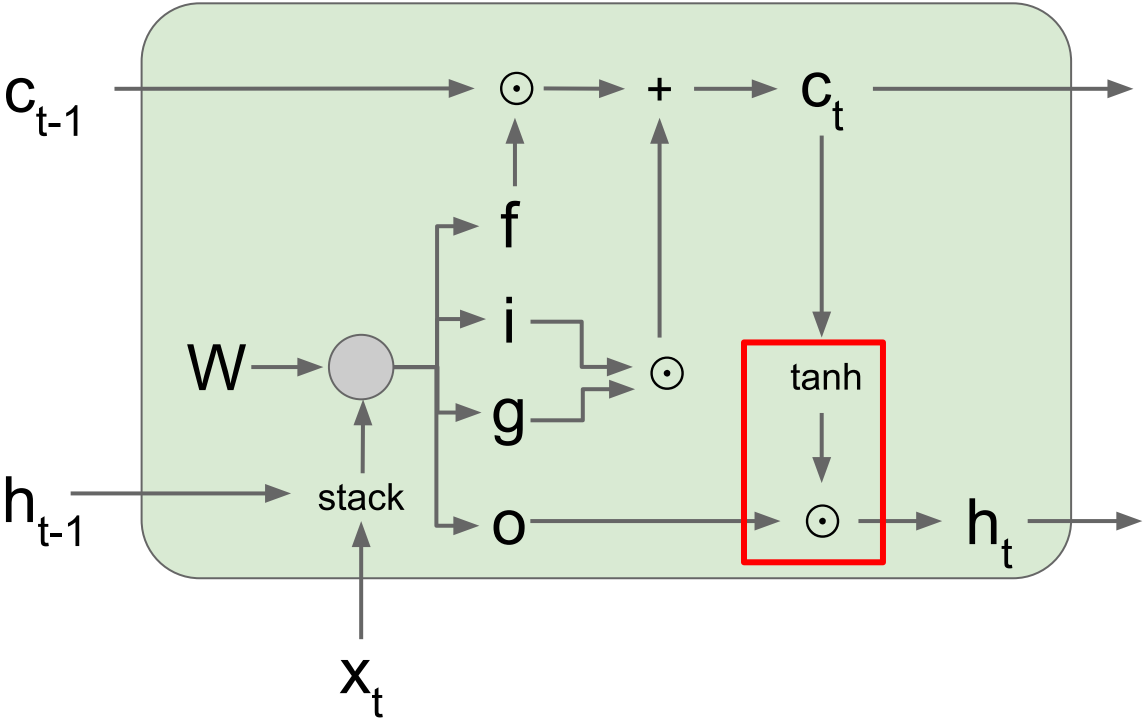
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



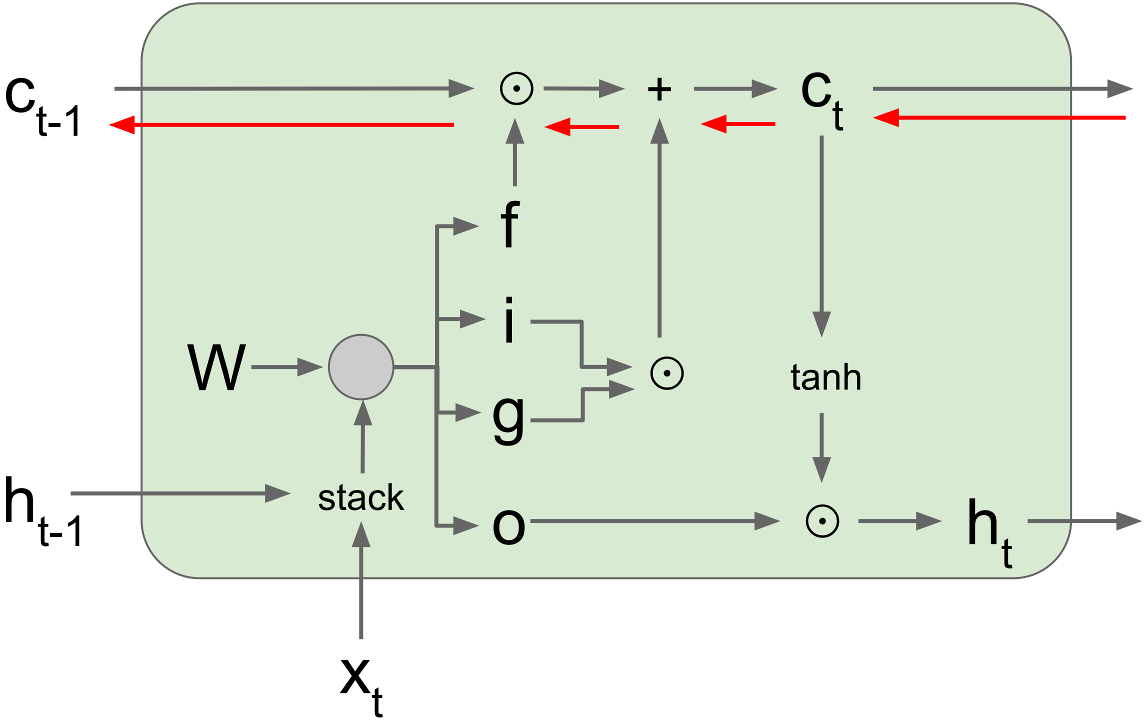
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]



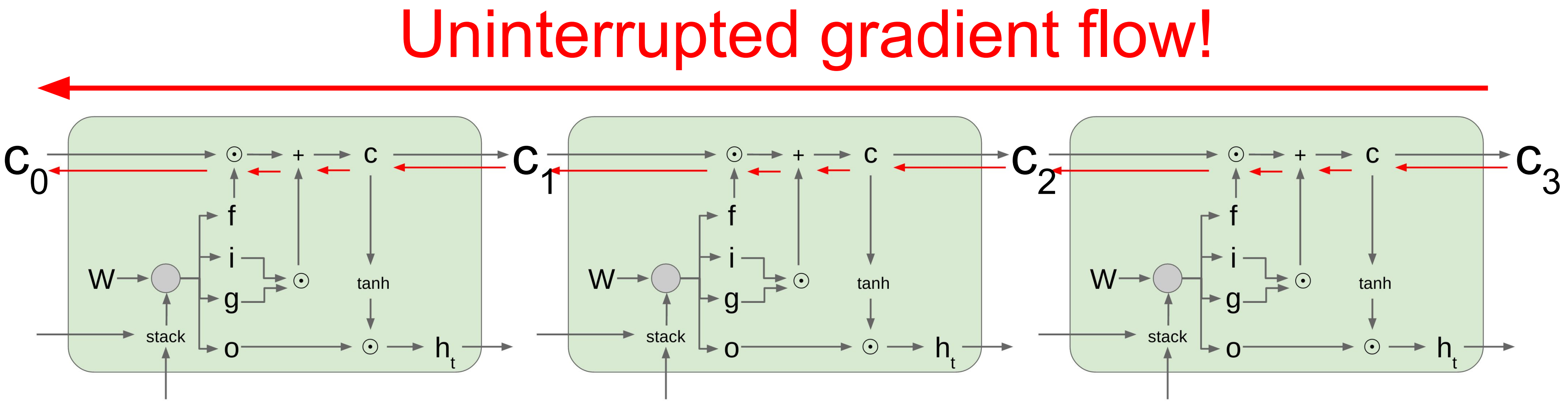
Backpropagation from c_t to c_{t-1} only elementwise multiplication by f, no matrix multiply by W

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

$$h_t = o \odot \tanh(c_t)$$

Long Short Term Memory (LSTM): Gradient Flow
[Hochreiter et al., 1997]



Notice that the gradient contains the **f** gate's vector of activations

- allows better control of gradients values, using suitable parameter updates of the forget gate.

Also notice that are added through the **f, i, g,** and **o** gates

- better balancing of gradient values

Do LSTMs solve the vanishing gradient problem?

The LSTM architecture makes it easier for the RNN to preserve information over many timesteps

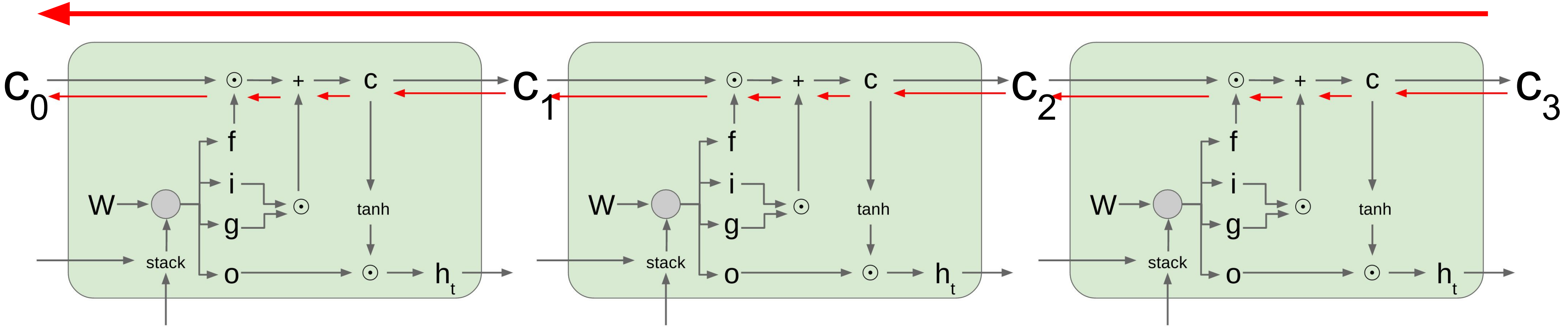
- e.g. **if the f = 1 and the i = 0**, then the information of that cell is preserved indefinitely.
- By contrast, it's harder for vanilla RNN to learn a recurrent weight matrix Wh that preserves info in hidden state

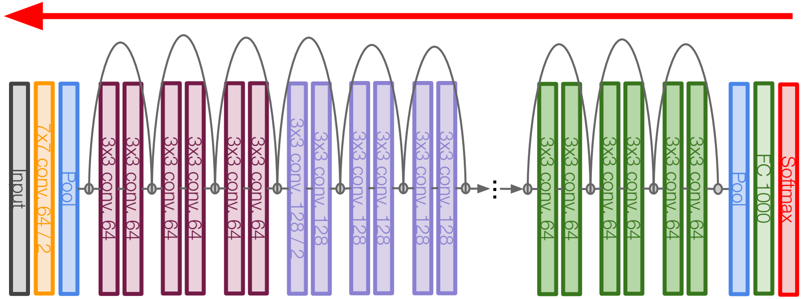
LSTM **doesn't guarantee** that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies

Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]

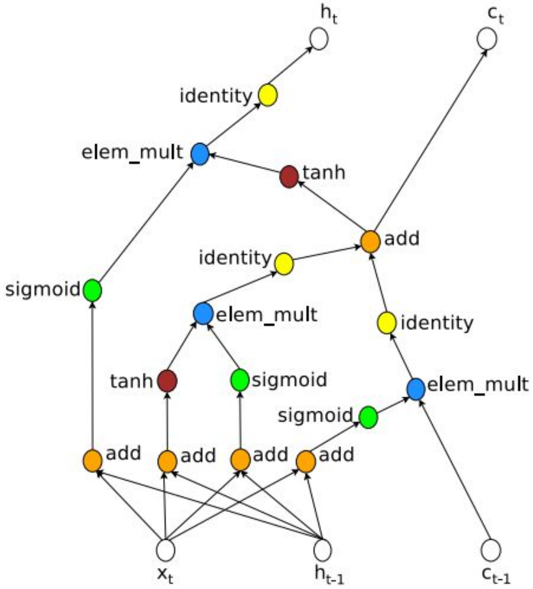
Uninterrupted gradient flow!



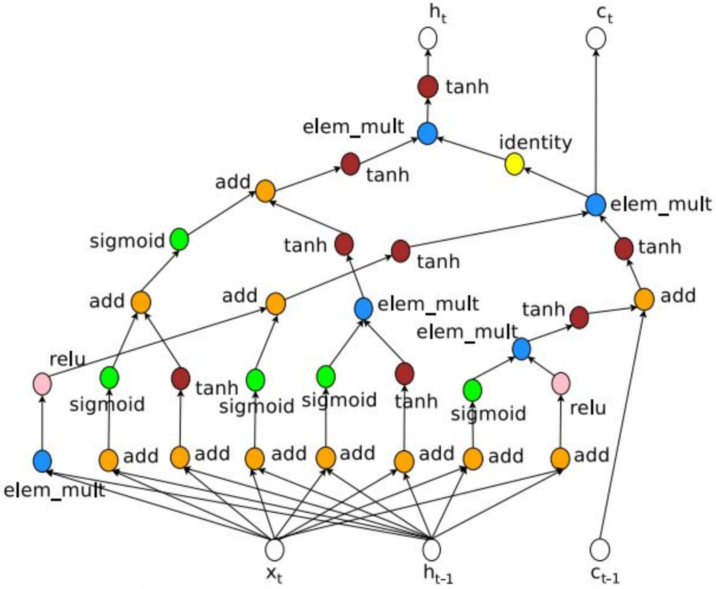


Similar to residual connections (e.g. in ResNets and Transformers), which we will learn about soon!

Neural Architecture Search for RNN architectures



LSTM cell



Cell they found

Zoph et Le, “Neural Architecture Search with Reinforcement Learning”, ICLR 2017

Other RNN Variants

[*An Empirical Exploration of Recurrent Network Architectures*, Jozefowicz et al., 2015]

GRU [*Learning phrase representations using rnn encoder-decoder for statistical machine translation*, Cho et al. 2014]

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$
$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$
$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h)$$
$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t$$

[*LSTM: A Search Space Odyssey*, Greff et al., 2015]

MUT1:

$$\begin{aligned} z &= \mathrm{sigm}(W_{\mathrm{xz}}x_t + b_{\mathrm{z}}) \\ r &= \mathrm{sigm}(W_{\mathrm{xr}}x_t + W_{\mathrm{hr}}h_t + b_{\mathrm{r}}) \\ h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + \tanh(x_t) + b_{\mathrm{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

MUT2:

$$\begin{aligned} z &= \mathrm{sigm}(W_{\mathrm{xz}}x_t + W_{\mathrm{hz}}h_t + b_{\mathrm{z}}) \\ r &= \mathrm{sigm}(x_t + W_{\mathrm{hr}}h_t + b_{\mathrm{r}}) \\ h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + W_{xh}x_t + b_{\mathrm{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

MUT3:

$$\begin{aligned} z &= \mathrm{sigm}(W_{\mathrm{xz}}x_t + W_{\mathrm{hz}}\tanh(h_t) + b_{\mathrm{z}}) \\ r &= \mathrm{sigm}(W_{\mathrm{xr}}x_t + W_{\mathrm{hr}}h_t + b_{\mathrm{r}}) \\ h_{t+1} &= \tanh(W_{\mathrm{hh}}(r \odot h_t) + W_{xh}x_t + b_{\mathrm{h}}) \odot z \\ &+ h_t \odot (1 - z) \end{aligned}$$

Recurrence for Vision

- LSTM wer a good default choice until this year
- Use variants like GRU if you want faster compute and less parameters
- Use transformers (next lecture) as they are dominating NLP and also vision models
    - almost everyday there is a new transformer model

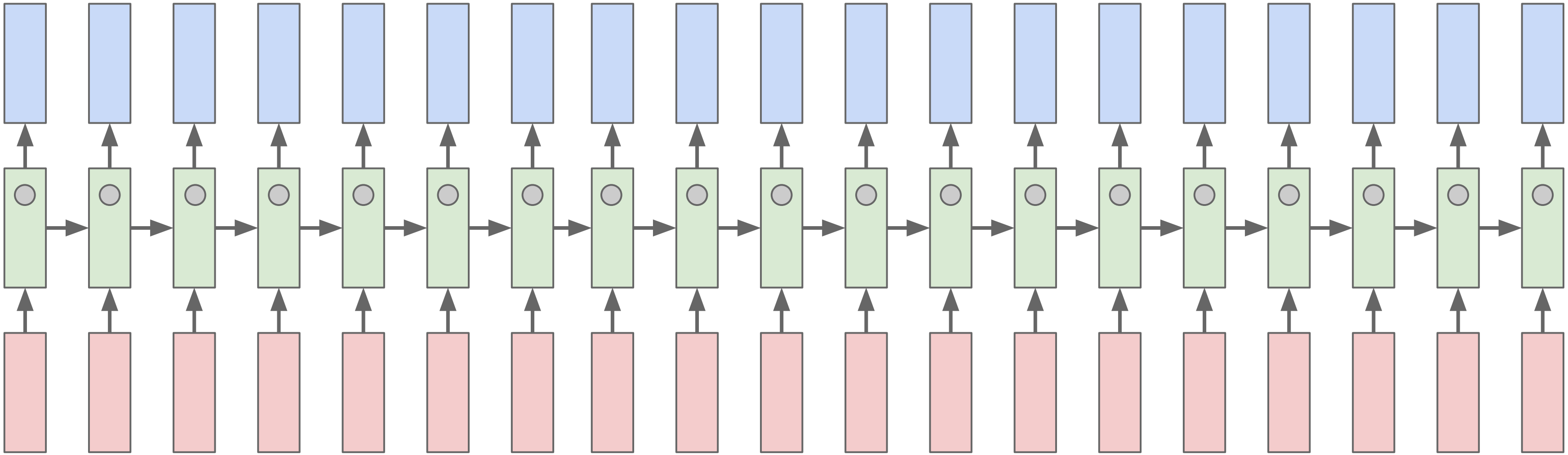
Su et al. "Vl-bert: Pre-training of generic visual-linguistic representations." ICLR 2020
Lu et al. "Vilbert: Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks." NeurIPS 2019
Li et al. "Visualbert: A simple and performant baseline for vision and language." *arXiv* 2019

Summary

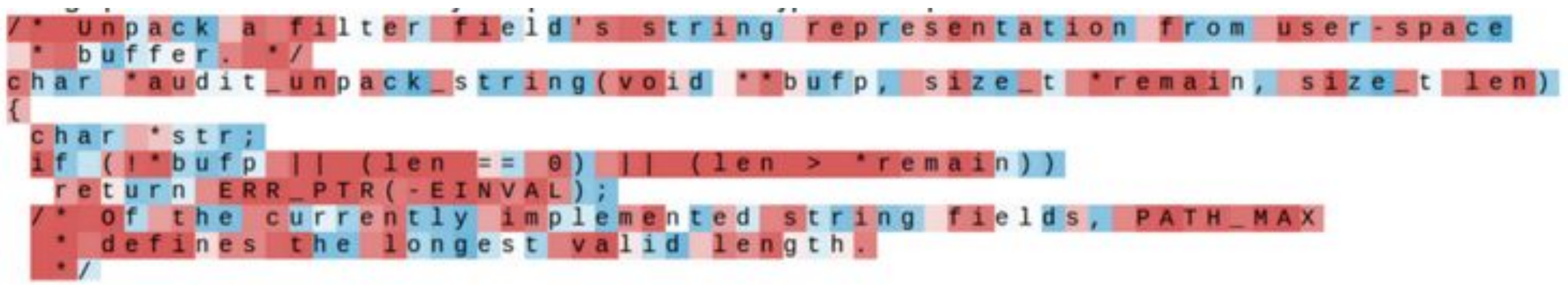
- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don’t work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research, as well as new paradigms for reasoning over sequences
- Better understanding (both theoretical and empirical) is needed.

Next time: Attention and transformers!

Searching for interpretable cells

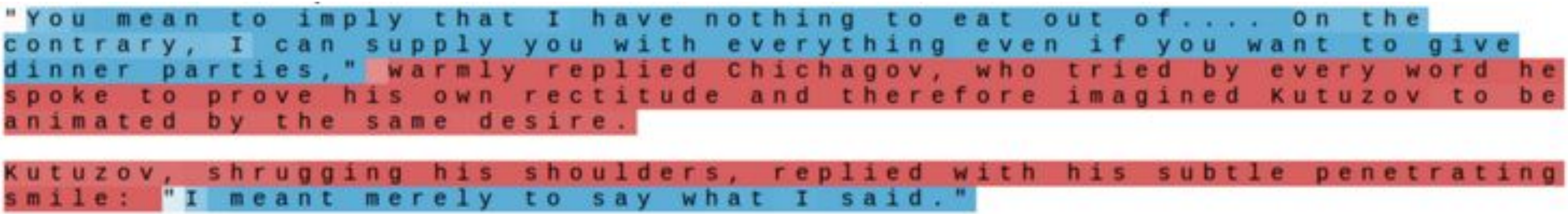


Searching for interpretable cells



Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

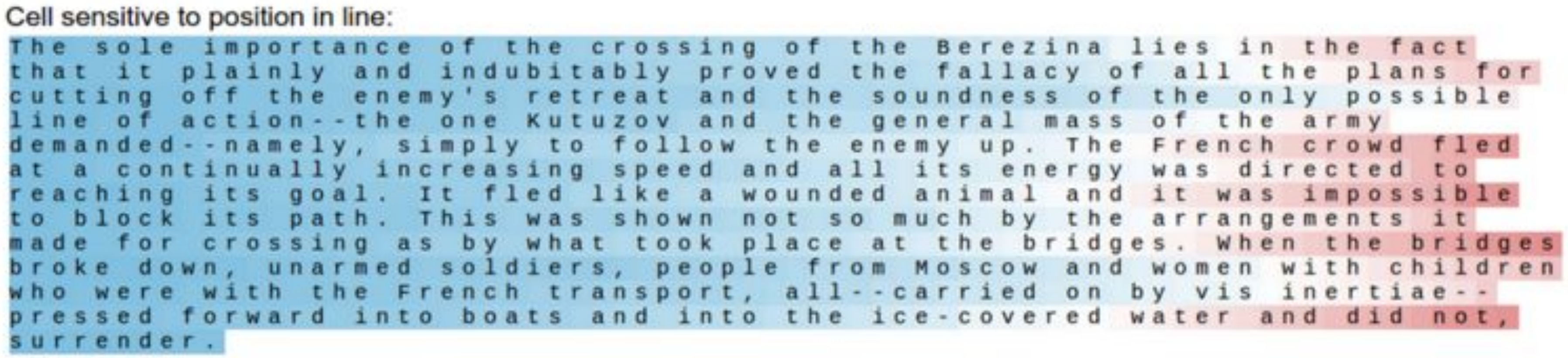
Searching for interpretable cells



quote detection cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

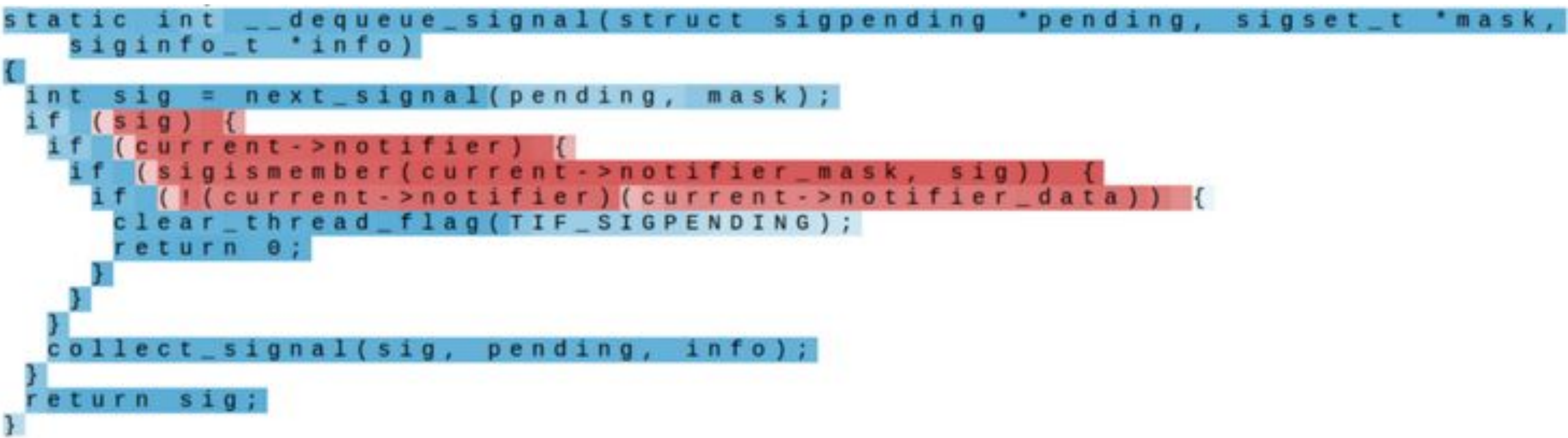
Searching for interpretable cells



line length tracking cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

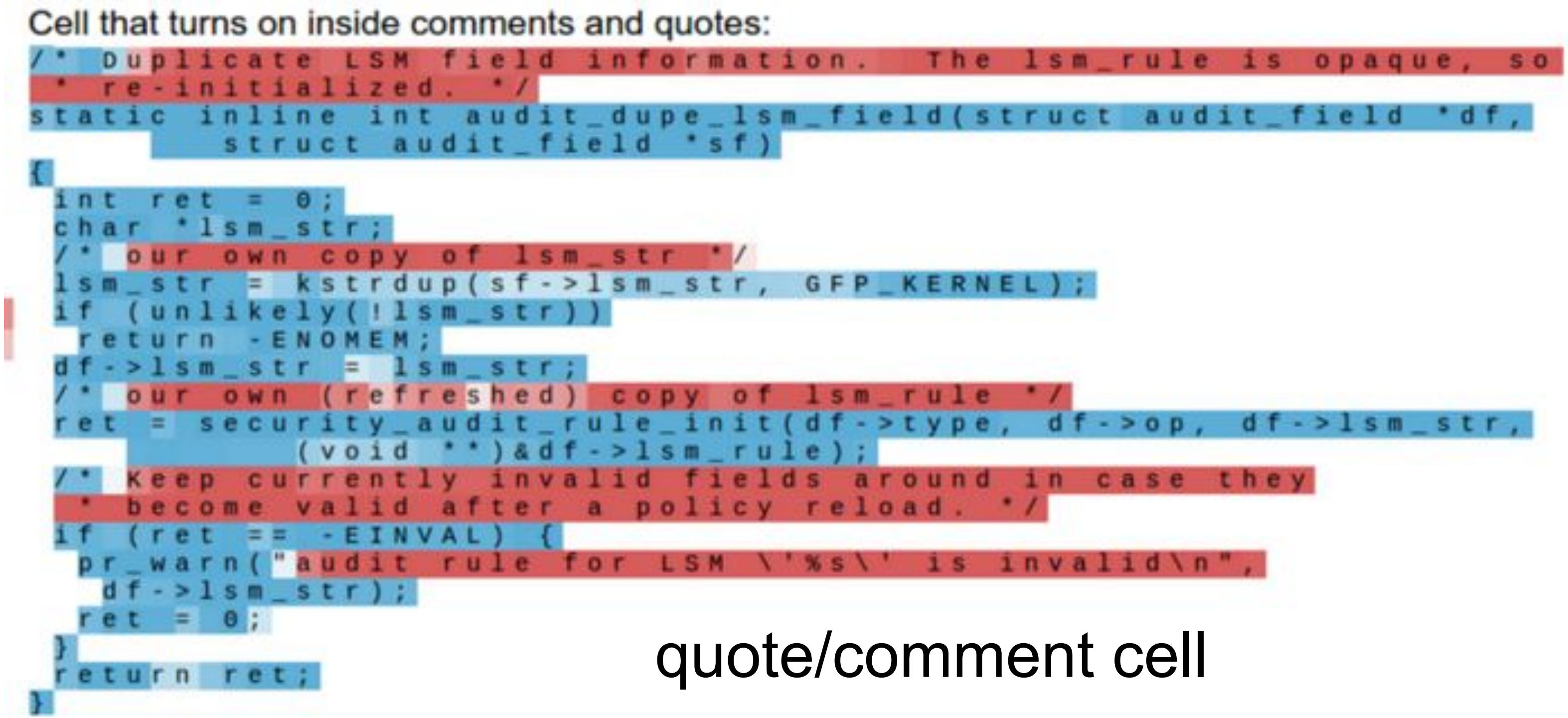
Searching for interpretable cells



if statement cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

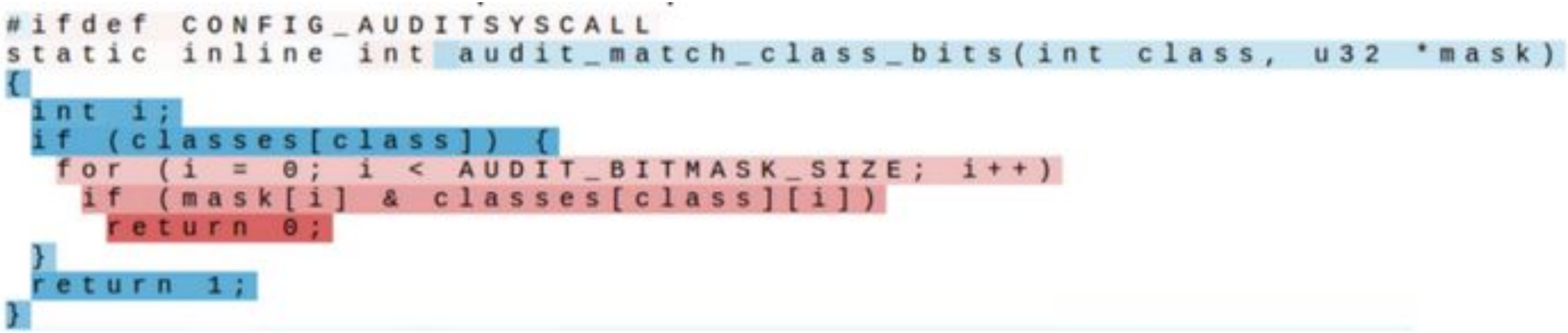
Searching for interpretable cells



quote/comment cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Searching for interpretable cells



code depth cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016