

ITCS 6156/8156 Fall 2023 Machine Learning

Recurrent Neural Networks

Instructor: Hongfei Xue

Email: hongfei.xue@charlotte.edu

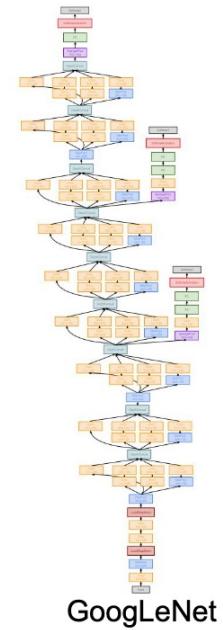
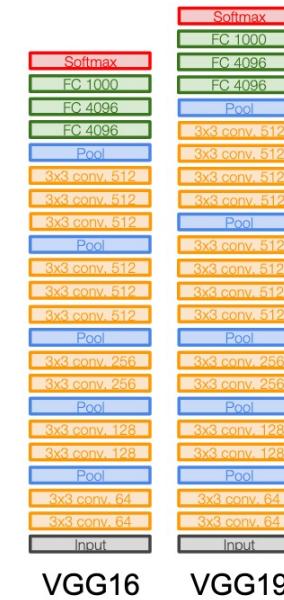
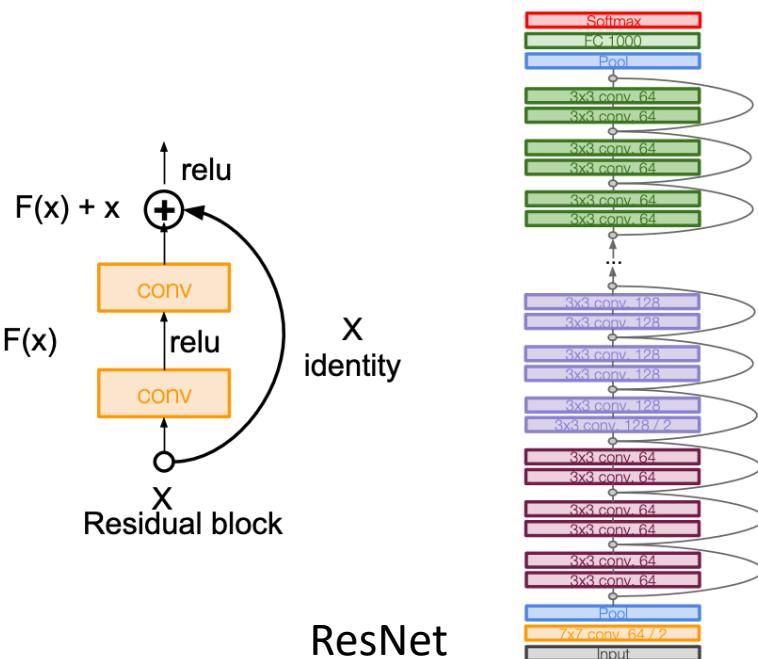
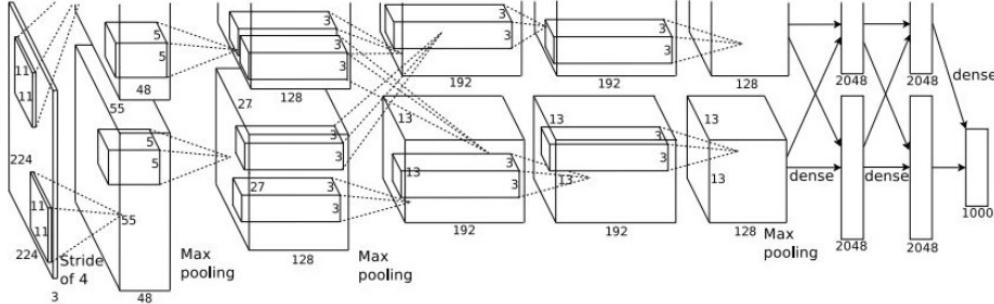
Class Meeting: Mon & Wed, 4:00 PM – 5:15 PM, CHHS 376



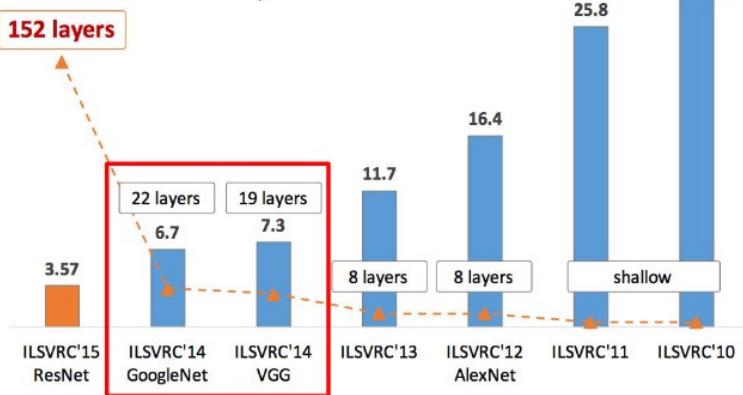
Some content in the slides is based on DeepMind's and Dr. Fei-Fei Li's lectures

Recap: CNN Architecture

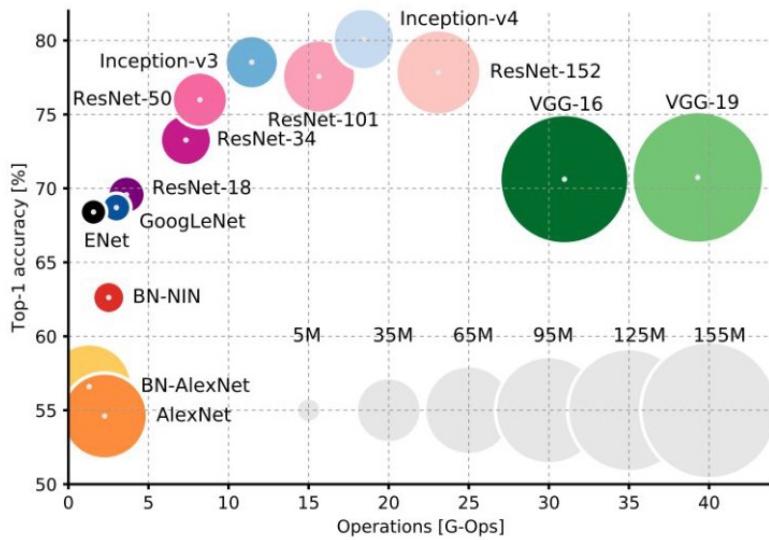
AlexNet



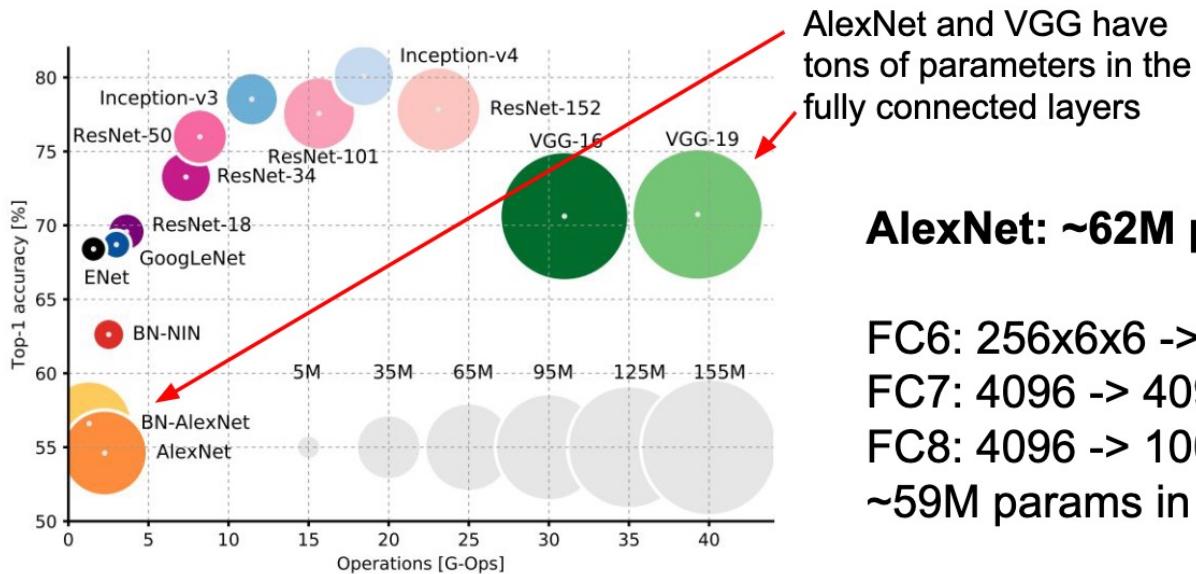
Revolution of Depth



Recap: CNN Architecture



Recap: CNN Architecture



AlexNet and VGG have tons of parameters in the fully connected layers

AlexNet: ~62M parameters

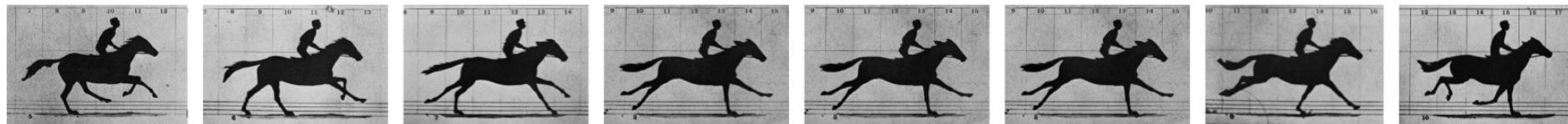
FC6: 256x6x6 -> 4096: 38M params

FC7: 4096 -> 4096: 17M params

FC8: 4096 -> 1000: 4M params

~59M params in FC layers!

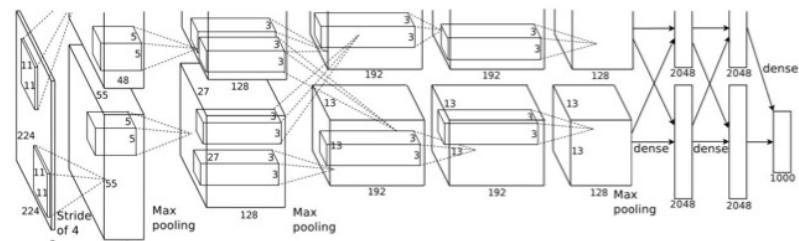
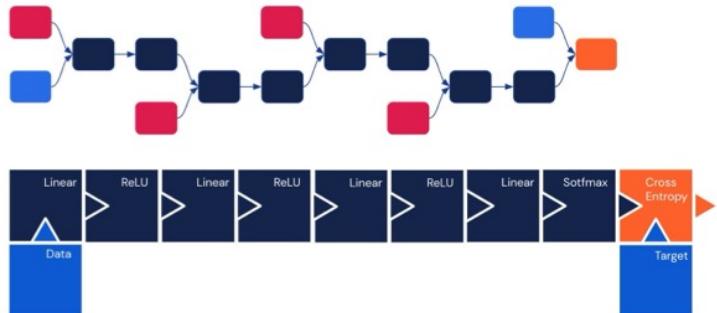
Modeling Sequences



Collections of elements where:

- Elements can be **repeated**
- **Order** matters
- Of **variable** (potentially infinite) length

Models discussed so far don't do well with sequential data.



Modeling Sequences

“Why do we care about sequences?”



“Why”, “do”, “we”, “care”, “about”, “sequences”, “?”



“D”, “O”

“C”, “A”, “R”, “E”

“S”, “E”, “Q”, “U”, “E”, “N”, “C”, “E”, “S”

“Why”, “do”, “we”, “care”, “about”, “sequences”, “?”

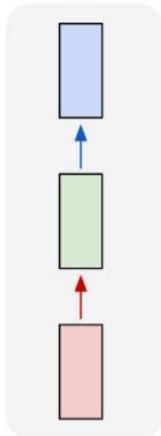
“W”, “H”, “Y”

“W”, “E”

“A”, “B”, “O”, “U”, “T”

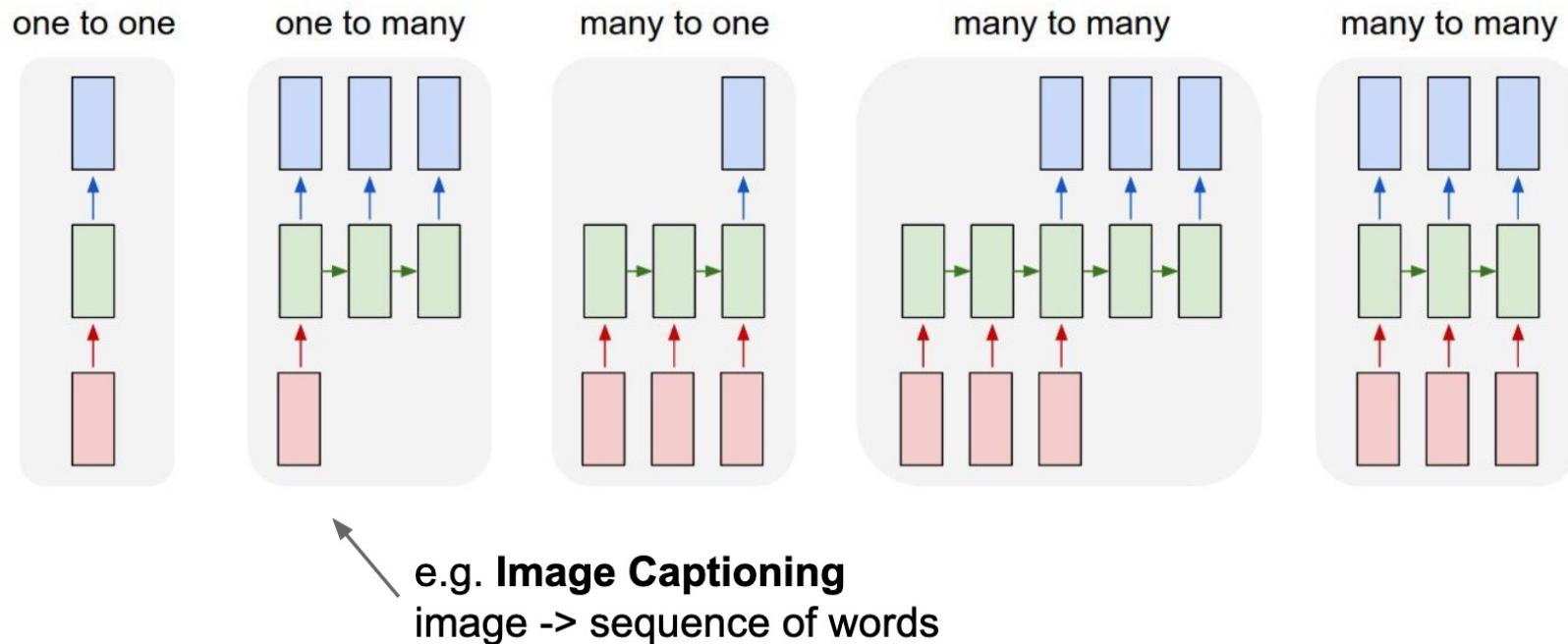
Process Sequence

one to one

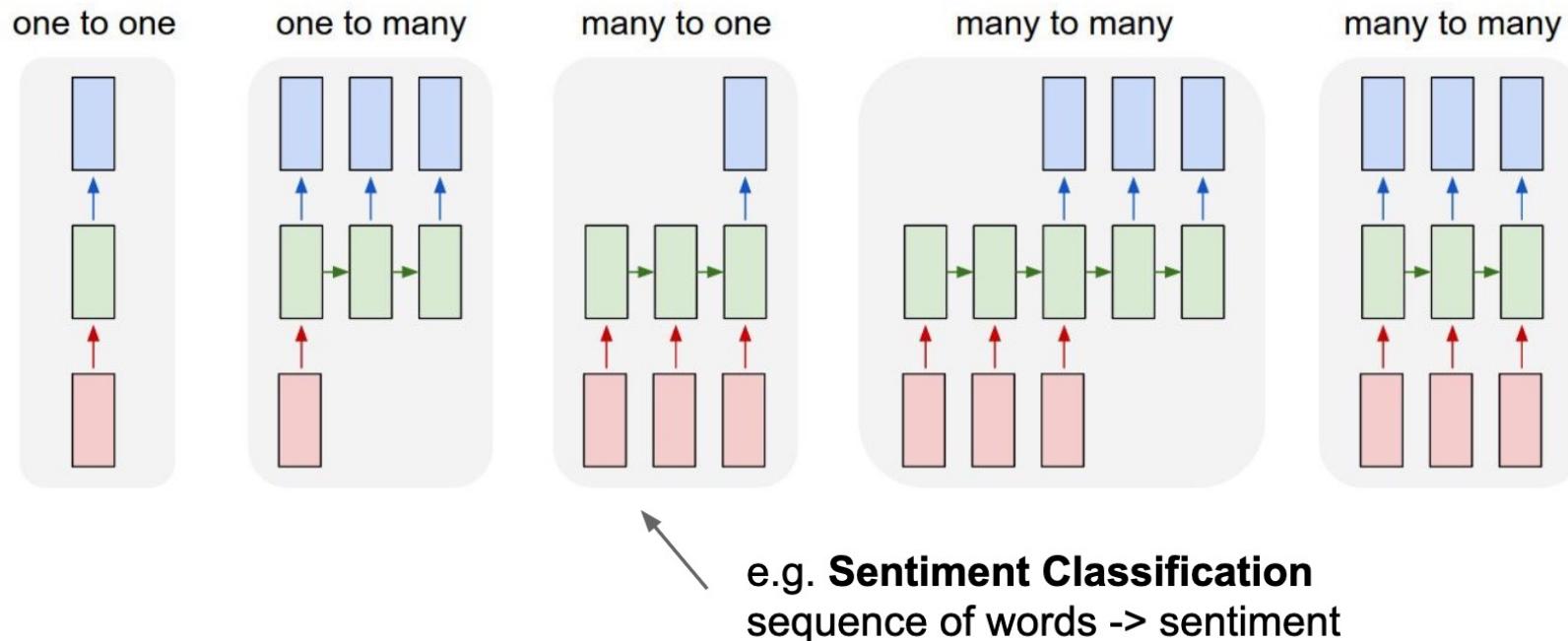


← **Vanilla Neural Networks**

Process Sequence

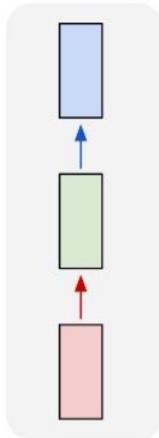


Process Sequence

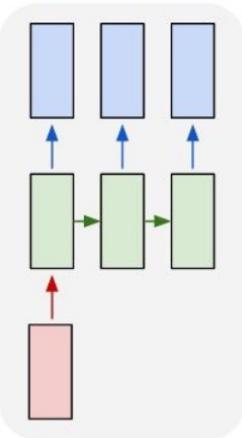


Process Sequence

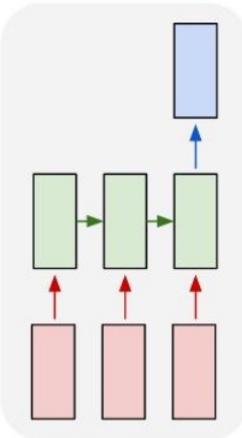
one to one



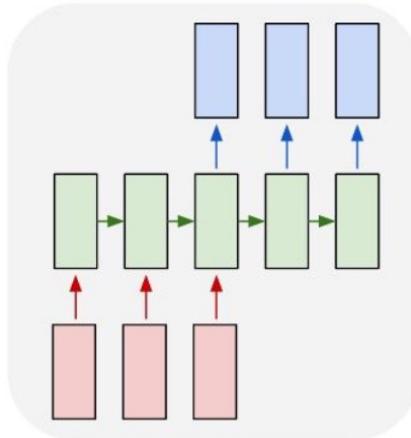
one to many



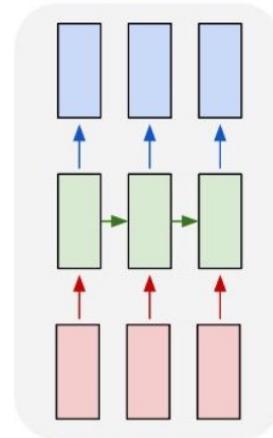
many to one



many to many

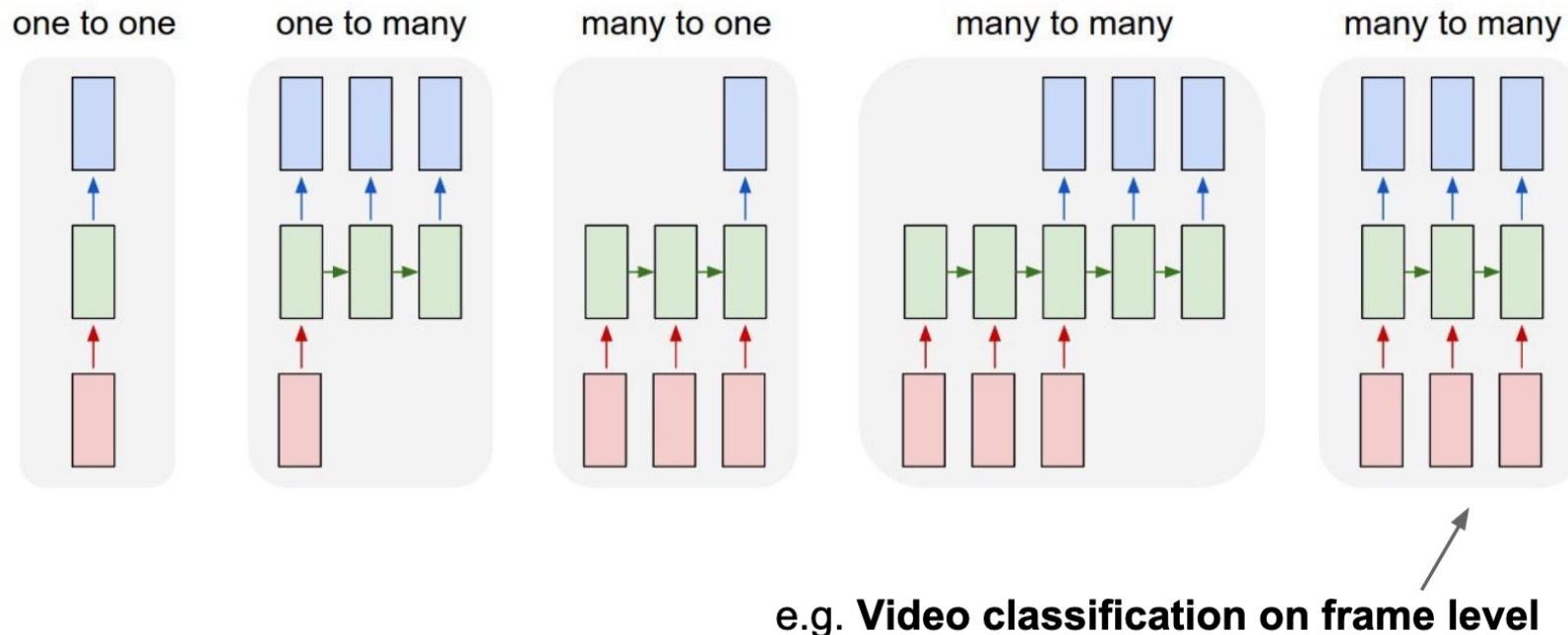


many to many

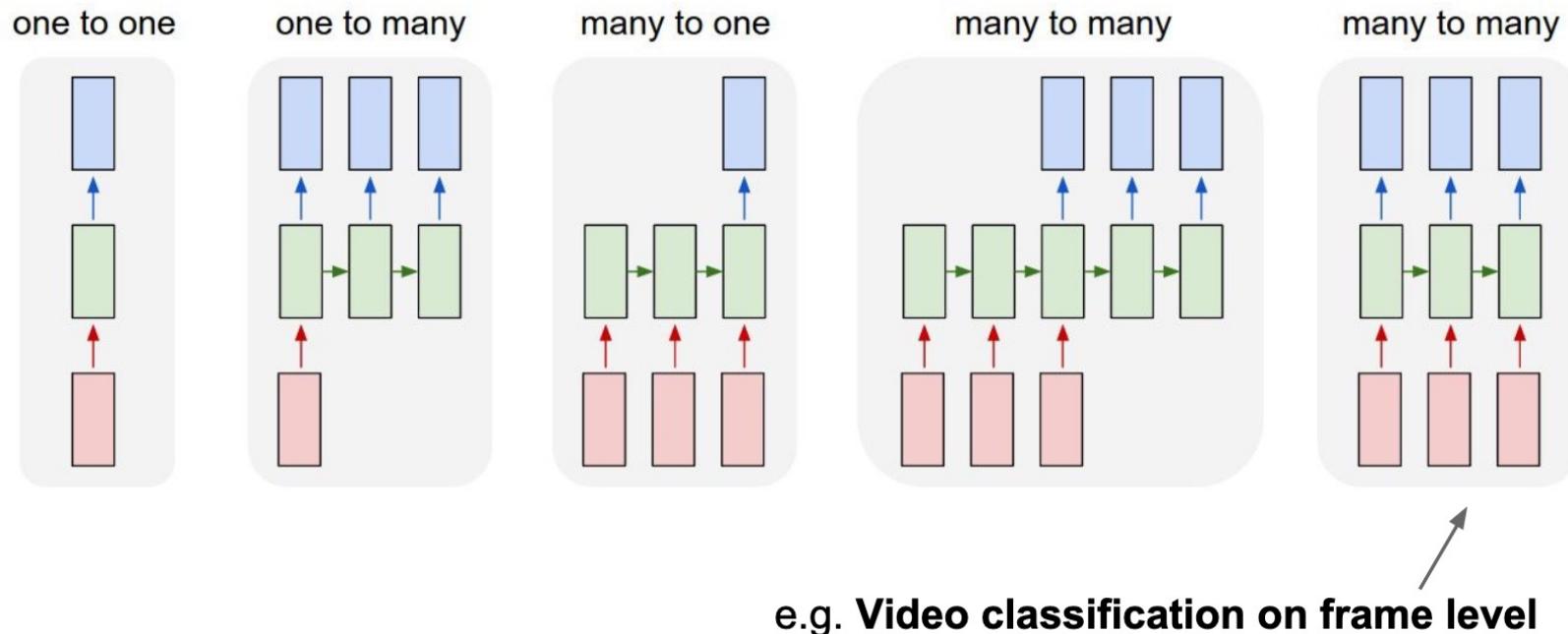


e.g. **Machine Translation (Chatbots)**
seq of words \rightarrow seq of words

Process Sequence



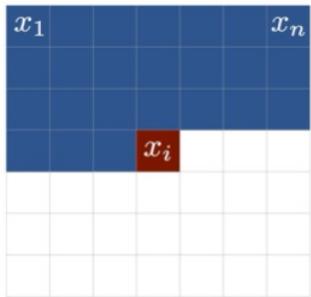
Process Sequence



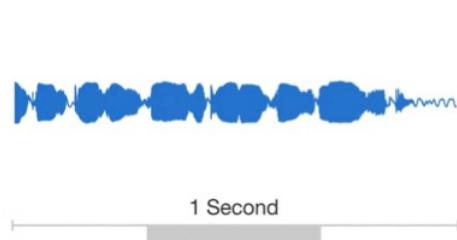
Sequences Are Everywhere

"Sequences really seem to be everywhere! We should learn how to model them. What is the best way to do that? Stay tuned!"

Words, letters



Images



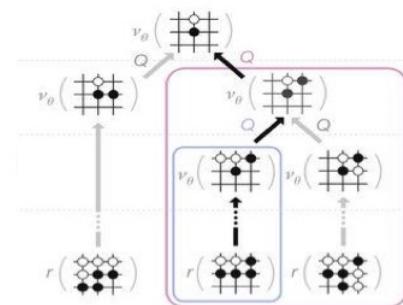
Speech

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def forward_backward_prop(w, T):
5     hs = [0.5]
6     for _ in range(T):
7         hs.append(np.tanh(w*hs[-1]))
8
9     dh = 1
10    for t in range(T):
11        dh = (1-hs[-1-t])**2 * w * dh
12
13    return hs[-1], dh
14
15 T = 10 # sequence length
16 wlim = 4 # limit of interval over weights w
17
18 results = []
19 ws = np.linspace(-wlim, wlim, 1000)
20 for w in ws:
21     results.append(forward_backward_prop(w, T))
22
23 plt.plot(ws, [r[0] for r in results], label='MM state')
24 plt.plot(ws, [r[1] for r in results], label='Gradients')
```

Programs



Videos



Decision making

- Sequences are collections of variable length where order matters.
- Sequences are widespread across machine learning applications.
- Not all deep learning models can handle sequential data.

Training Machine Learning Models

	Supervised learning	Sequence modelling
Data	$\{x, y\}_i$	$\{x\}_i$
Model	$y \approx f_\theta(x)$	$p(x) \approx f_\theta(x)$
Loss	$\mathcal{L}(\theta) = \sum_{i=1}^N l(f_\theta(x_i), y_i)$	$\mathcal{L}(\theta) = \sum_{i=1}^N \log p(f_\theta(x_i))$
Optimisation	$\theta^* = \arg \min_{\theta} \mathcal{L}(\theta)$	$\theta^* = \arg \max_{\theta} \mathcal{L}(\theta)$

Modeling $p(x)$

“Modeling word probabilities is really difficult”

Simplest model:

Assume independence of words

$$p(\mathbf{x}) = \prod_{t=1}^T p(x_t)$$

$p(\text{"modeling"}) \times p(\text{"word"}) \times p(\text{"probabilities"}) \times p(\text{"is"}) \times p(\text{"really"}) \times p(\text{"difficult"})$

Word	$p(x_i)$
the	0.049
be	0.028
...	...
really	0.0005
...	...

However:

Most likely 6-word sentence:

“The the the the the the.”

→ Independence assumption does not match sequential structure of language.

Modeling $p(x)$

More realistic model:

Assume conditional dependence of words

$$p(x_T) = p(x_T | x_1, \dots, x_{T-1})$$

Modeling word probabilities is really

?

Context

Target

$p(x|context)$

difficult

0.01

hard

0.009

fun

0.005

...

easy

...

0.00001

Modeling $p(\mathbf{x})$

The chain rule

Computing the joint $p(\mathbf{x})$ from conditionals

$$p(\mathbf{x}) = \prod_{t=1}^T p(x_t | x_1, \dots, x_{t-1})$$

Modeling

Modeling word

Modeling word probabilities

Modeling word probabilities is

Modeling word probabilities is really

Modeling word probabilities is really difficult

$$p(x_1)$$

$$p(x_2 | x_1)$$

$$p(x_3 | x_2, x_1)$$

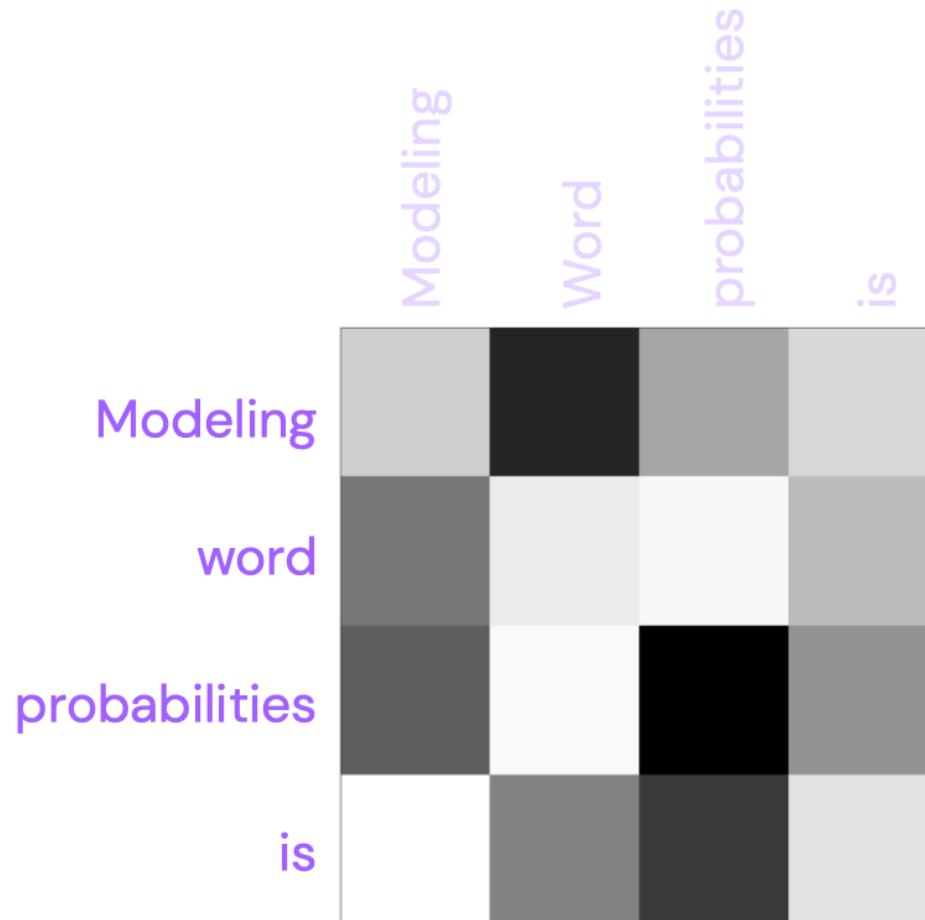
$$p(x_4 | x_3, x_2, x_1)$$

$$p(x_5 | x_4, x_3, x_2, x_1)$$

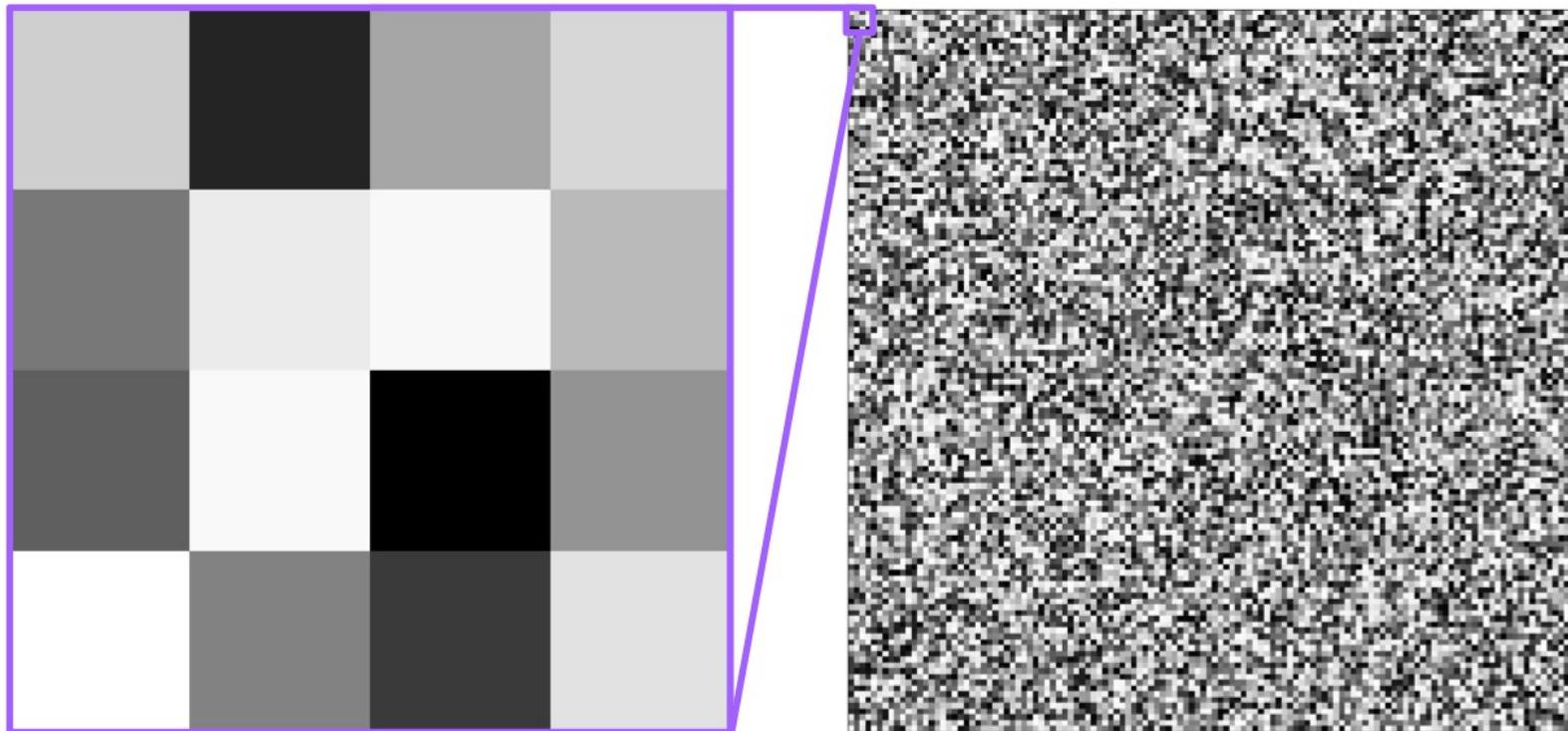
$$p(x_6 | x_5, x_4, x_3, x_2, x_1)$$

Scalability Issues

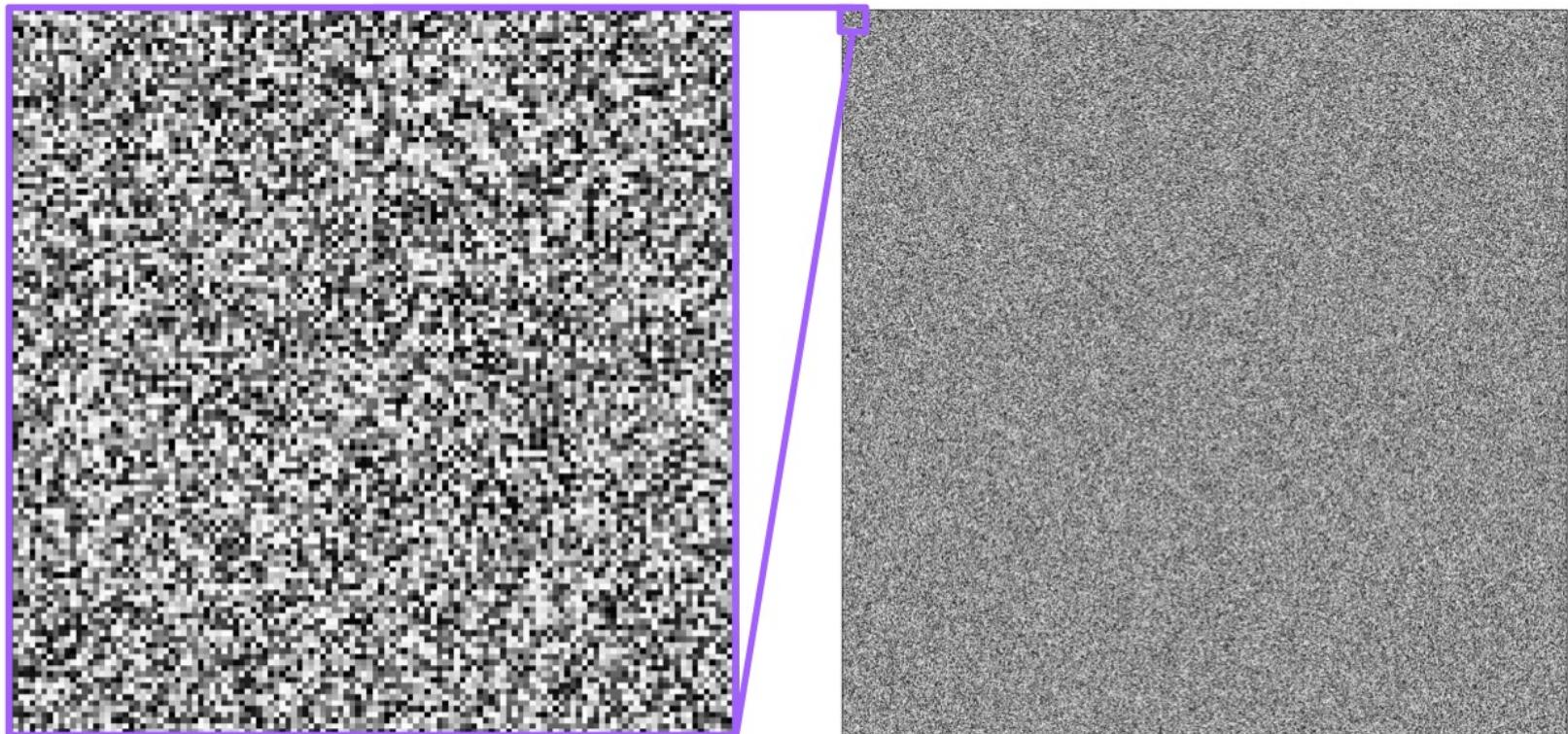
$$p(x_2|x_1)$$



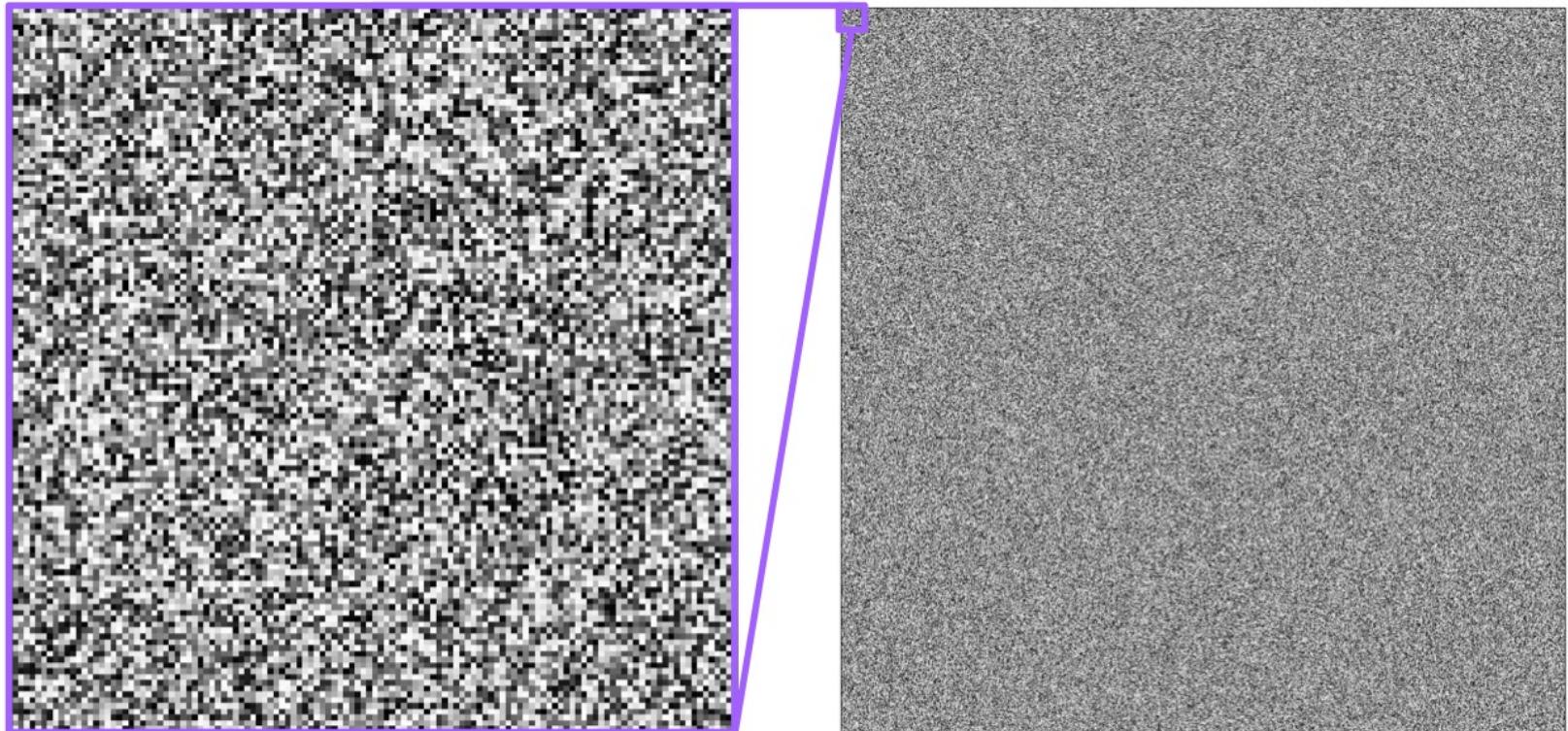
Scalability Issues



Scalability Issues



Scalability Issues



These images are only for context of size $N=1$!
The table size of larger contexts will grow with **vocabulary^N**

Fixing a small context: N-grams

Only condition on N previous words

$$p(\mathbf{x}) \approx \prod_{t=1}^T p(x_t | x_{t-N-1}, \dots, x_{t-1})$$

Modeling

Modeling word

Modeling word probabilities

word probabilities is

probabilities is really

is really difficult

$$p(x_1)$$

$$p(x_2 | x_1)$$

$$p(x_3 | x_2, x_1)$$

$$p(x_4 | x_3, x_2)$$

$$p(x_5 | x_4, x_3)$$

$$p(x_6 | x_5, x_4)$$

Downsides of Using N-grams

1 Doesn't take into account words that are more than N words away

2 Data table is still very, very large

All Our N-gram are Belong to You

Thursday, August 3, 2006

Posted by Alex Franz and Thorsten Brants, Google Machine Translation Team

Here at Google Research we have been using word [n-gram models](#) for a variety of R&D projects, such as [statistical machine translation](#), speech recognition, [spelling correction](#), entity detection, information extraction, and others. While such models have usually been estimated from training corpora containing at most a few billion words, we have been harnessing the vast power of Google's datacenters and distributed processing [infrastructure](#) to process larger and larger training corpora. We found that there's no data like more data, and scaled up the size of our data by one order of magnitude, and then another, and then one more - resulting in a training corpus of [one trillion words](#) from public Web pages.

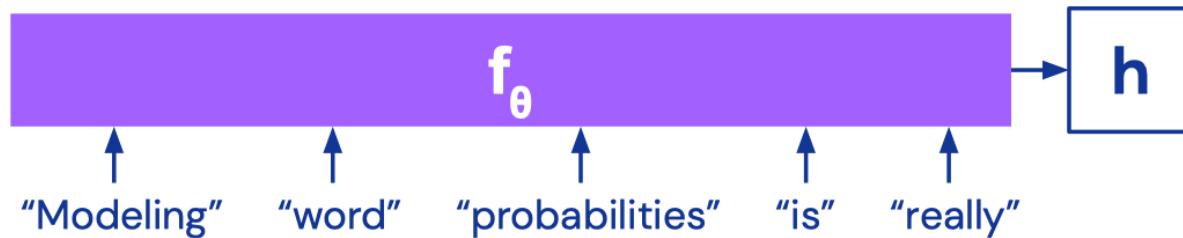
We believe that the entire research community can benefit from access to such massive amounts of data. It will advance the state of the art, it will focus research in the promising direction of large-scale, data-driven approaches, and it will allow all research groups, no matter how large or small their computing resources, to play together. That's why we decided to share this enormous dataset with everyone. We processed 1,024,908,267,229 words of running text and are publishing the counts for all 1,176,470,663 five-word sequences that appear at least 40 times. There are 13,588,391 unique words, after discarding words that appear less than 200 times.

- Modeling probabilities of sequences scales badly given the non-independent structure of their elements.

Learning to Model Word Probabilities

- Can this probability estimation be learned from data in a more efficient way?

1. Vectorising the context



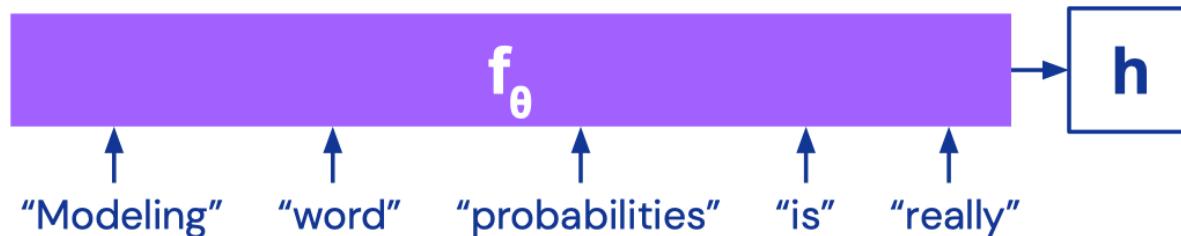
f_θ summarises the context in h such that:

$$p(x_t | x_1, \dots, x_{t-1}) \approx p(x_t | h)$$

Learning to Model Word Probabilities

- Can this probability estimation be learned from data in a more efficient way?

1. Vectorising the context



Desirable properties for f_θ :

Order matters

Variable length

Differentiable

Pairwise encoding

Preserves long-term

N-grams

$$p(\mathbf{x}) \approx \prod_{t=1}^T p(x_t | x_{t-N+1}, \dots, x_{t-1})$$

Modeling

$$p(x_1)$$

Modeling word

$$p(x_2 | x_1)$$

Modeling word probabilities

$$p(x_3 | x_2, x_1)$$

word probabilities is

$$p(x_4 | x_3, x_2)$$

probabilities is really

$$p(x_5 | x_4, x_3)$$

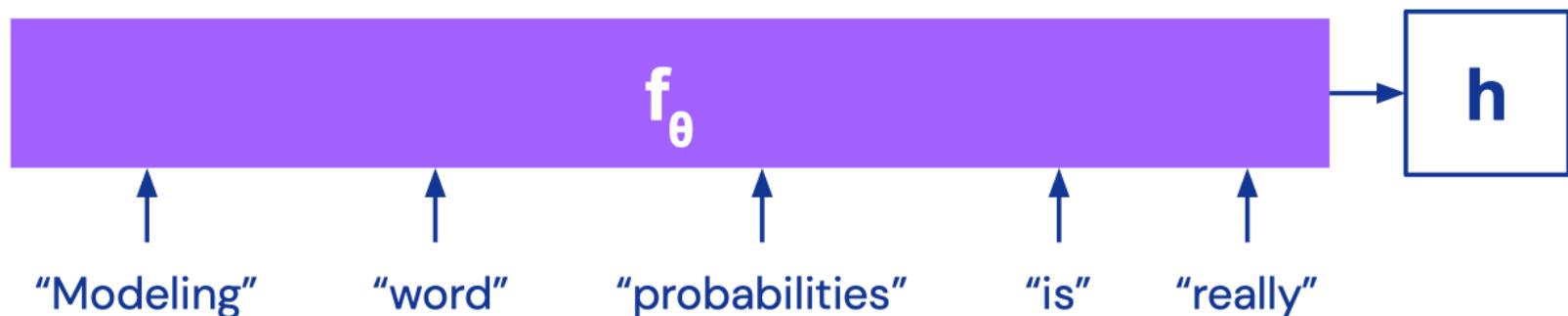
is really difficult

$$p(x_6 | x_5, x_4)$$

f_θ concatenates the N last words

N-gram	
Order matters	✓
Variable length	✗
Differentiable	✗
Pairwise encoding	✓
Preserves long-term	✗

Addition



	N-gram	Addition
Order matters	✓	✗
Variable length	✗	✓
Differentiable	✗	✓
Pairwise encoding	✓	✗
Preserves long-term	✗	✓

- N-grams and simple aggregation do not meet the requirements for modeling sequences.

Learning to Model Word Probabilities

Modeling conditional probabilities



Desirable properties for g_θ :

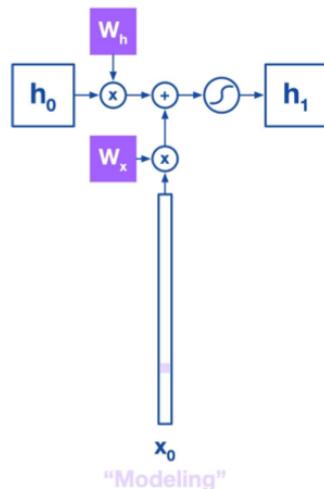
Individual changes can have large effects
(non-linear/deep)

Returns a probability distribution

Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs)

Persistent state variable \mathbf{h} stores information from the context observed so far.



$$\mathbf{h}_t = \tanh(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t)$$

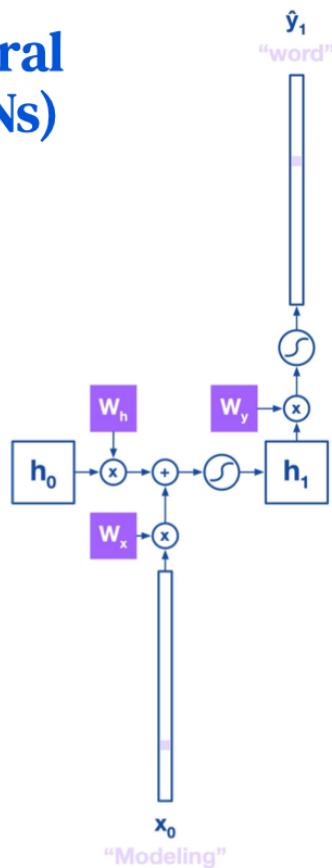
Word representation

Words are indexed and represented as 1-hot vectors

Large Vocabulary of possible words $|V|$

Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs)



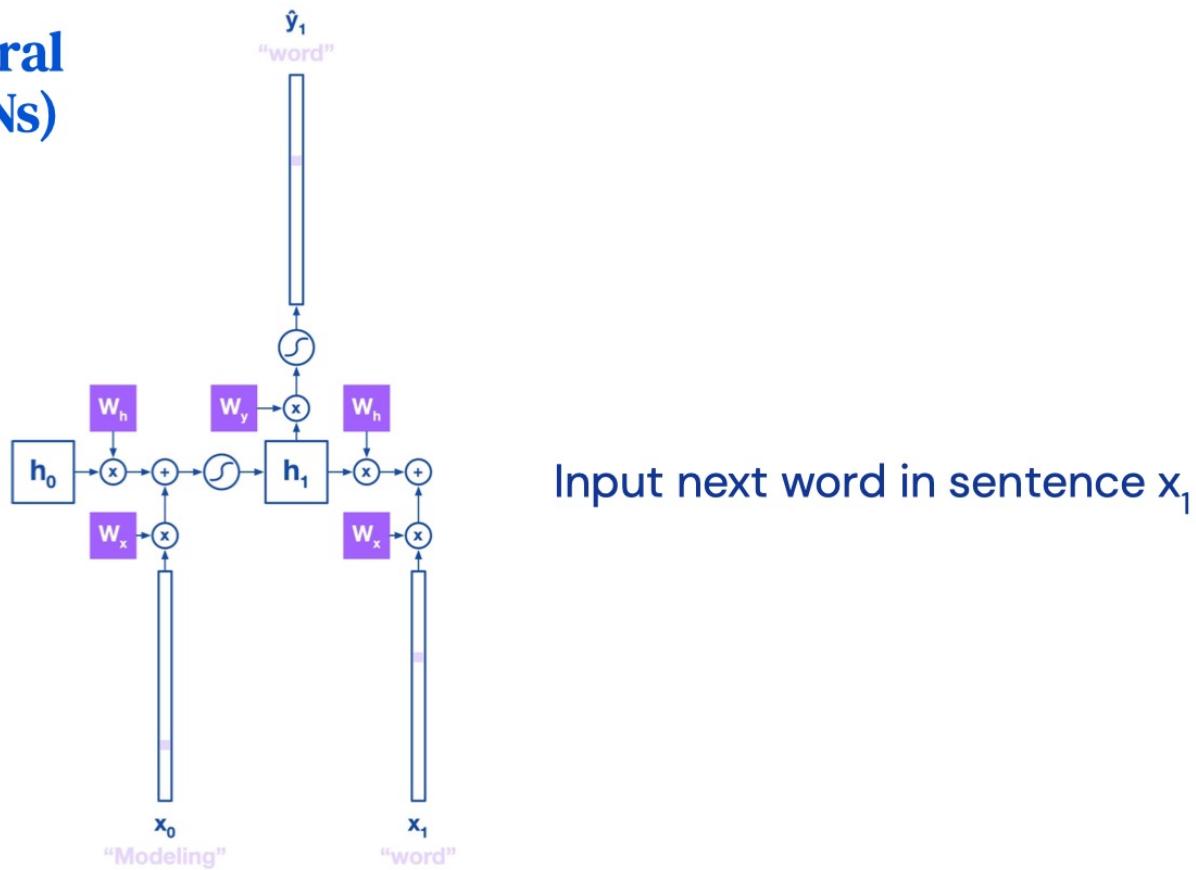
RNNs predict the target y (the next word) from the state h .

$$p(y_{t+1}) = \text{softmax}(W_y h_t)$$

Softmax ensures we obtain a distribution over all possible words.

Recurrent Neural Networks (RNNs)

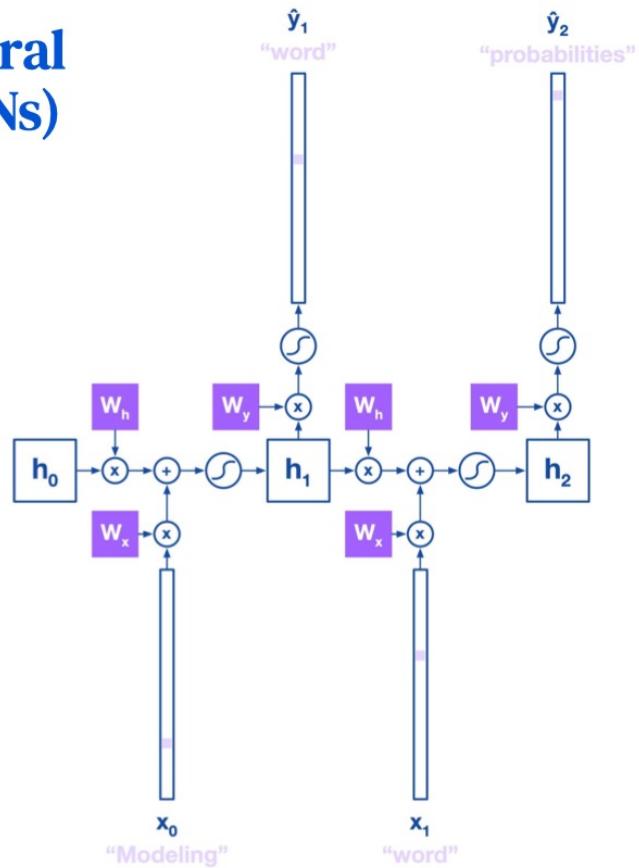
Recurrent Neural Networks (RNNs)



Input next word in sentence x_1

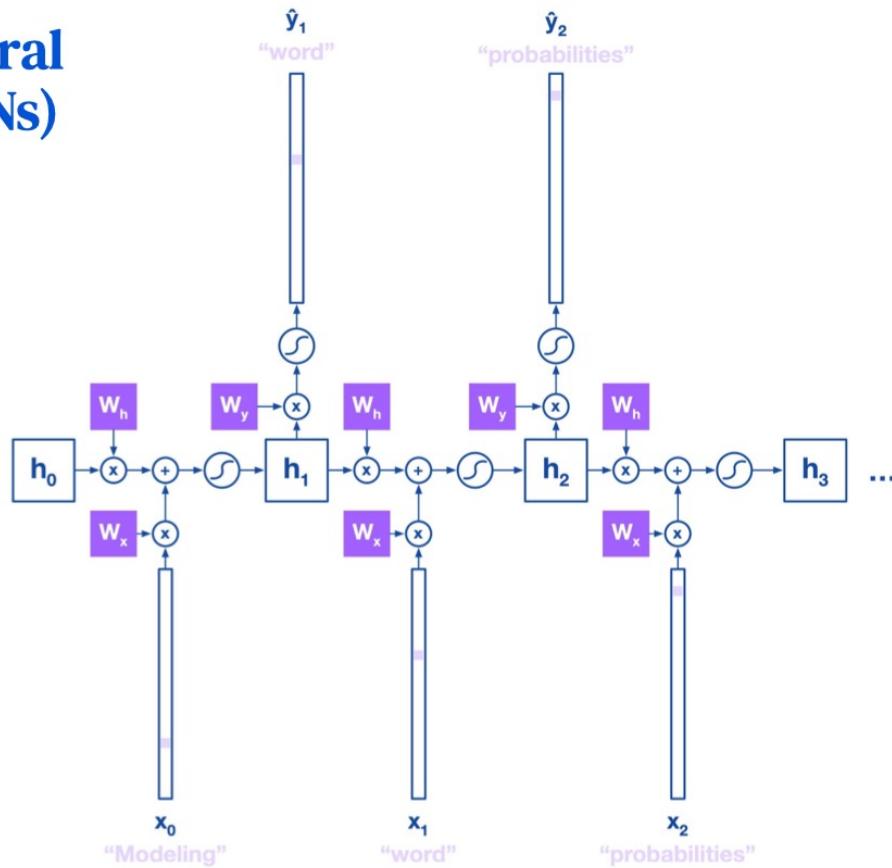
Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs)



Recurrent Neural Networks (RNNs)

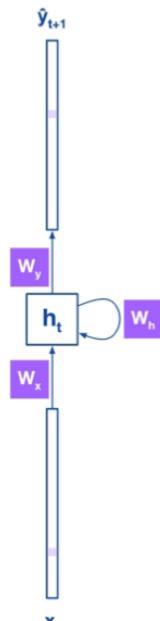
Recurrent Neural Networks (RNNs)



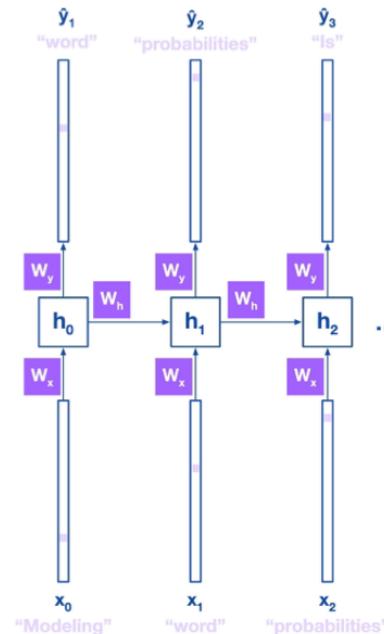
Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs)

Weights are shared over time steps



RNN



RNN rolled out over time

Loss: Cross Entropy

Next word prediction is essentially a classification task where the number of classes is the size of the vocabulary.

As such we use the cross-entropy loss:

For one word:

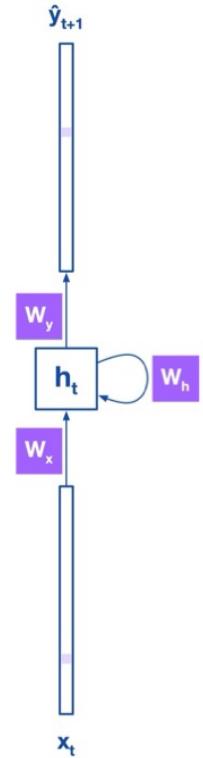
$$\mathcal{L}_\theta(\mathbf{y}, \hat{\mathbf{y}})_t = -\mathbf{y}_t \log \hat{\mathbf{y}}_t$$

For the

sentence:

$$\mathcal{L}_\theta(\mathbf{y}, \hat{\mathbf{y}}) = - \sum_{t=1}^T \mathbf{y}_t \log \hat{\mathbf{y}}_t$$

With parameters $\theta = \{\mathbf{W}_y, \mathbf{W}_x, \mathbf{W}_h\}$



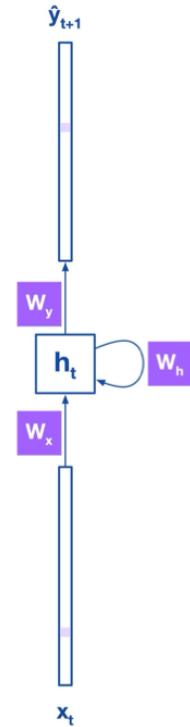
Differentiating

Parameters: \mathbf{W}_y , \mathbf{W}_x and \mathbf{W}_h

$$\mathbf{h}_t = \tanh(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t)$$

$$p(\mathbf{x}_{t+1}) = \text{softmax}(\mathbf{W}_y \mathbf{h}_t)$$

$$\mathcal{L}_\theta(\mathbf{y}, \hat{\mathbf{y}})_t = -\mathbf{y}_t \log \hat{\mathbf{y}}_t$$



Differentiating

Parameters: \mathbf{W}_y , \mathbf{W}_x and \mathbf{W}_h

$$\mathbf{h}_t = \tanh(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t)$$

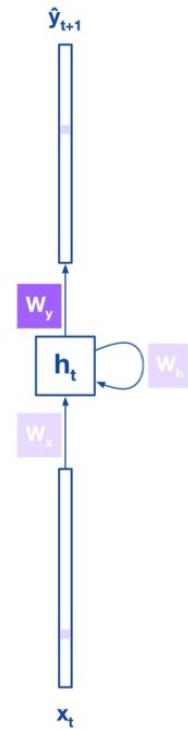
$$p(\mathbf{x}_{t+1}) = \text{softmax}(\mathbf{W}_y \mathbf{h}_t)$$

$$\mathcal{L}_\theta(\mathbf{y}, \hat{\mathbf{y}})_t = -\mathbf{y}_t \log \hat{\mathbf{y}}_t$$

\mathbf{W}_y :

$$\frac{\partial \mathcal{L}_{\theta,t}}{\partial \mathbf{W}_y} = \frac{\partial \mathcal{L}_{\theta,t}}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{W}_y}$$

$$= (\mathbf{y}_t - \hat{\mathbf{y}}_t) \mathbf{h}_t$$



Differentiating

Parameters: \mathbf{W}_y , \mathbf{W}_x and \mathbf{W}_h

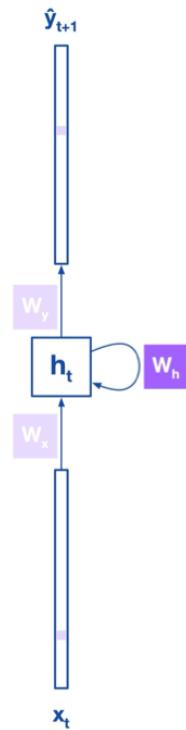
$$\mathbf{h}_t = \tanh(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t)$$

$$p(\mathbf{x}_{t+1}) = \text{softmax}(\mathbf{W}_y \mathbf{h}_t)$$

$$\mathcal{L}_\theta(\mathbf{y}, \hat{\mathbf{y}})_t = -\mathbf{y}_t \log \hat{\mathbf{y}}_t$$

\mathbf{W}_h :

$$\frac{\partial \mathcal{L}_{\theta,t}}{\partial \mathbf{W}_h} = \frac{\partial \mathcal{L}_{\theta,t}}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_h}$$



Differentiating

Parameters: \mathbf{W}_y , \mathbf{W}_x and \mathbf{W}_h

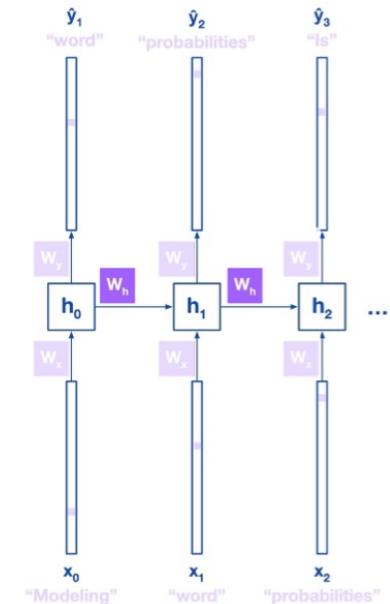
$$\mathbf{h}_t = \tanh(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t)$$

$$p(\mathbf{x}_{t+1}) = \text{softmax}(\mathbf{W}_y \mathbf{h}_t)$$

$$\mathcal{L}_\theta(\mathbf{y}, \hat{\mathbf{y}})_t = -\mathbf{y}_t \log \hat{\mathbf{y}}_t$$

\mathbf{W}_h :

$$\frac{\partial \mathcal{L}_{\theta,t}}{\partial \mathbf{W}_h} = \frac{\partial \mathcal{L}_{\theta,t}}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_h}$$



Differentiating

Back propagating through time

$$\begin{aligned}\frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_h} &= \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_h} + \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{W}_h} \\ &= \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_h} + \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \left[\frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{W}_h} + \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{h}_{t-2}} \frac{\partial \mathbf{h}_{t-2}}{\partial \mathbf{W}_h} \right] \\ &\dots \\ &= \sum_{k=1}^t \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \mathbf{W}_h}\end{aligned}$$

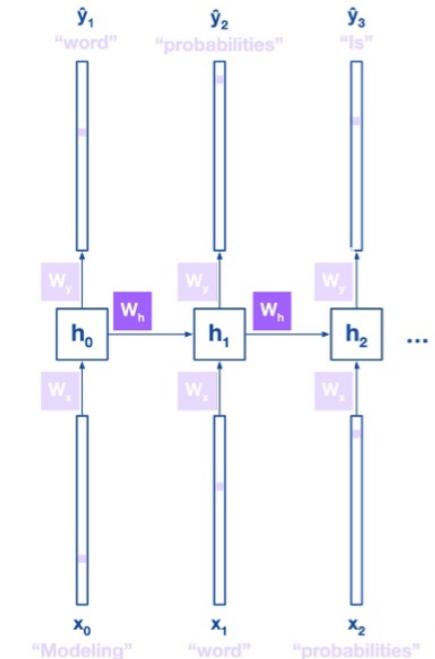
Differentiating

$$\mathbf{h}_t = \tanh(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t)$$

$$p(\mathbf{x}_{t+1}) = \text{softmax}(\mathbf{W}_y \mathbf{h}_t)$$

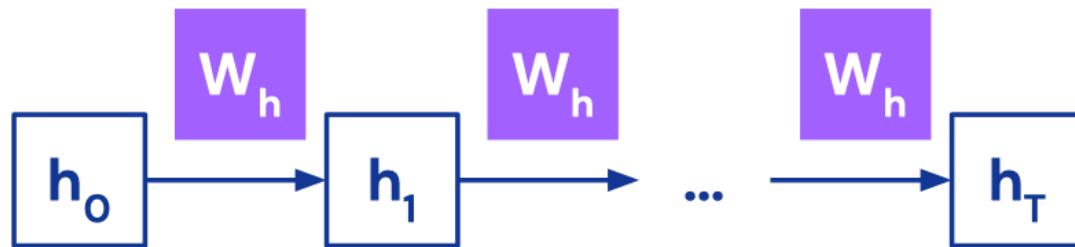
$$\mathcal{L}_\theta(\mathbf{y}, \hat{\mathbf{y}})_t = -\mathbf{y}_t \log \hat{\mathbf{y}}_t$$

$$\begin{aligned} \frac{\partial \mathcal{L}_{\theta,t}}{\partial \mathbf{W}_h} &= \frac{\partial \mathcal{L}_{\theta,t}}{\partial \hat{\mathbf{y}}_t} \frac{\partial \hat{\mathbf{y}}_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_h} \\ \frac{\partial \mathbf{h}_t}{\partial \mathbf{W}_h} &= \sum_{k=1}^t \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_k} \frac{\partial \mathbf{h}_k}{\partial \mathbf{W}_h} \end{aligned}$$



Vanishing Gradients

A simple example



$$h_t = W_h h_{t-1}$$

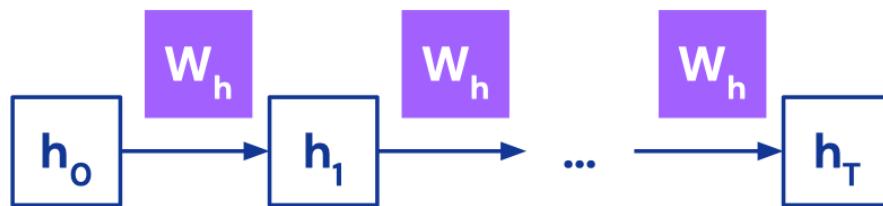
$$h_t = (W_h)^t h_0$$

$$h_t \rightarrow \infty \text{ if } |W_h| > 1$$

$$h_t \rightarrow 0 \text{ if } |W_h| < 1$$

Vanishing Gradients

A simple example



But RNNs bound h with a tanh!

$$h_t = \mathbf{W}_h h_{t-1} \rightarrow h_t = \tanh(\mathbf{W}_h h_{t-1})$$

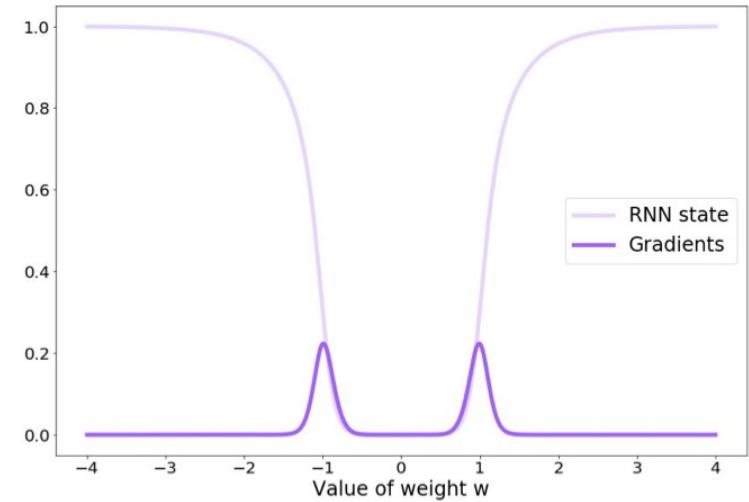
Vanishing Gradients

$$\mathbf{h}_t = \tanh(\mathbf{W}_h \mathbf{h}_{t-1})$$



```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 def forward_backward_prop(w, T):
5     hs = [0.5]
6     for _ in range(T):
7         hs.append(np.tanh(w*hs[-1]))
8
9     dh = 1
10    for t in range(T):
11        dh = (1-hs[-1-t]**2) * w * dh
12
13    return hs[-1], dh
14
15 T = 10 # sequence length
16 wlim = 4 # limit of interval over weights w
17
18 results = []
19 ws = np.linspace(-wlim, wlim, 1000)
20 for w in ws:
21     results.append(forward_backward_prop(w, T))
22
23 plt.plot(ws, [r[0] for r in results], label='RNN state')
24 plt.plot(ws, [r[1] for r in results], label='Gradients')
```

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-T}} = (1 - \tanh^2(\mathbf{W}_h \mathbf{h}_{t-1})) \mathbf{W}_h \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{h}_{t-T}}$$



Properties of RNNs

	N-gram	Addition	RNN
Order matters	✓	✗	✓
Variable length	✗	✓	✓
Differentiable	✗	✓	✓
Pairwise encoding	✓	✗	✗
Preserves long-term	✗	✓	✗

- Recurrent neural networks can model sequences of variable length and can be trained via back-propagation.
- They do, however suffer from the vanishing gradients problem, which stops them from capturing long-term dependencies.

Questions?