In this lecture series, leading research scientists from leading AI research lab, DeepMind, will give 12 lectures on an exciting selection of topics in Deep Learning, ranging from the fundamentals of training neural networks via advanced ideas around memory, attention, and generative modelling to the important topic of responsible innovation.

Please join us for a deep dive lecture series into Deep Learning!

#UCLxDeepMind
General information

**Exits:**
At the back, the way you came in

**Wifi:**
UCL guest
TODAY’S SPEAKER

Marta Garnelo

Marta is a research scientist at DeepMind working on deep generative models and meta learning. During her time at DM she has worked on Generative Query Networks as well as Neural Processes and recently her research focus has shifted towards multi-agent systems. In addition she is currently wrapping up her PhD with Prof Murray Shanahan at Imperial College London where she also did an MSc in Machine Learning.
In this lecture we will focus on sequential data and how machine learning methods have been adapted to process this particular type of structure. We will start by introducing some fundamentals of sequence modeling including common architectures designed for this task such as RNNs and LSTMs. We will then move on to sequence-to-sequence decoding and its applications before finishing with some examples of recent applications of sequence models.
Motivation
Why sequences matter and why the methods we have covered so far don’t work on them.

Fundamentals
Loss, optimisation and architectures of sequence models.

Generation
Applications and examples of sequence modelling.
Motivation
So far

Feed forward networks

Convolutional networks
Sequences

Collections of elements where:

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

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

Models discussed so far don’t do well with sequential data.
“Why do we care about sequences?”
“Why”, “do”, “we”, “care”, “about”, “sequences”, “?”
Why, do, we, care, about, sequences, ?
So far

Sequences

"Why", "do", "we", "care", "about", "sequences", "?"

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

Speech

Images

Programs

Videos

Decision making
Summary

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
Fundamentals

Sequences and Recurrent Neural Networks
Training machine learning models

Supervised learning

Data

\( \{x, y\}_i \)

Model

\( y \approx f_{\theta}(x) \)

Loss

\[
L(\theta) = \sum_{i=1}^{N} l(f_{\theta}(x_i), y_i)
\]

Optimisation

\( \theta^* = \arg\min_{\theta} L(\theta) \)
Training machine learning models

**Supervised learning**

- **Data**
  \[ \{x, y\}_i \]

- **Model**
  \[ y \approx f_\theta(x) \]

- **Loss**
  \[ \mathcal{L}(\theta) = \sum_{i=1}^{N} l(f_\theta(x_i), y_i) \]

- **Optimisation**
  \[ \theta^* = \arg \min_{\theta} \mathcal{L}(\theta) \]

**Sequence modelling**

- **Data**
  \[ \{x\}_i \]

- **Model**
  \[ p(x) \approx f_\theta(x) \]

- **Loss**
  \[ \mathcal{L}(\theta) = \sum_{i=1}^{N} \log p(f_\theta(x_i)) \]

- **Optimisation**
  \[ \theta^* = \arg \max_{\theta} \mathcal{L}(\theta) \]
Modeling $p(x)$

“Modeling word probabilities is really difficult”
Modeling $p(x)$

Simplest model:
Assume independence of words

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

\[
p(“modeling”) \times p(“word”) \times p(“probabilities”) \times p(“is”) \times p(“really”) \times p(“difficult”)
\]

<table>
<thead>
<tr>
<th>Word</th>
<th>$p(x_i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>0.049</td>
</tr>
<tr>
<td>be</td>
<td>0.028</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>really</td>
<td>0.0005</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Modeling $p(x)$

**Simplest model:**
Assume independence of words

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

$p(“modeling”) \times p(“word”) \times p(“probabilities”) \times p(“is”) \times p(“really”) \times p(“difficult”)$

<table>
<thead>
<tr>
<th>Word</th>
<th>$p(x_i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>0.049</td>
</tr>
<tr>
<td>be</td>
<td>0.028</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>really</td>
<td>0.0005</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**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

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

Modeling word probabilities is really difficult, hard, fun, easy...

| Context | Target | $p(x|\text{context})$ |
|---------|--------|-----------------------|
| difficult |        | 0.01                  |
| hard     |        | 0.009                 |
| fun      |        | 0.005                 |
| ...      |        | ...                   |
| easy     |        | 0.00001               |
Modeling $p(x)$

The chain rule
Computing the joint $p(x)$ from conditionals

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

$p(x_1)$
$p(x_2|x_1)$
$p(x_3|x_2, x_1)$
$p(x_4|x_3, x_2, x_1)$
$p(x_5|x_4, x_3, x_2, x_1)$
$p(x_6|x_5, x_4, x_3, x_2, x_1)$
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 $\text{vocabulary}^N$
Fixing a small context: N-grams

Only condition on N previous words

\[ p(x) \approx \prod_{i=1}^{T} p(x_t|x_{t-N-1}, \ldots, x_{t-1}) \]

Modeling

Modeling word

Modeling word probabilities

word probabilities is really difficult
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.

https://ai.googleblog.com/2006/08/all-our-n-gram-are-belong-to-you.html
Summary

Modeling probabilities of sequences scales badly given the non-independent structure of their elements.
Can this probability estimation be learned from data in a more efficient way?
Learning to model word probabilities

1. Vectorising the context

$f_\theta$ summarises the context in $h$ such that:

$$p(x_t | x_1, \ldots, x_{t-1}) \approx p(x_t | h)$$
Learning to model word probabilities

1. Vectorising the context

Desirable properties for $f_\theta$:

- Order matters
- Variable length
- Learnable (differentiable)
- Individual changes can have large effects (non-linear/deep)
Desirable properties

- Order matters
- Variable length
- Differentiable
- Pairwise encoding
- Preserves long-term
N-grams

\[ p(x) \approx \prod_{t=1}^{T} p(x_t|x_{t-N-1}, \ldots, x_{t-1}) \]

Modeling word probabilities is really difficult

Modeling word probabilities

\[ p(x_1) \]
\[ p(x_2|x_1) \]
\[ p(x_3|x_2, x_1) \]
\[ p(x_4|x_3, x_2) \]
\[ p(x_5|x_4, x_3) \]
\[ p(x_6|x_5, x_4) \]

\( f_\theta \) concatenates the N last words
## Properties of N-grams as $f_\theta$

<table>
<thead>
<tr>
<th>N-gram</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Order matters</td>
<td>✓</td>
</tr>
<tr>
<td>Variable length</td>
<td>✘</td>
</tr>
<tr>
<td>Differentiable</td>
<td>✘</td>
</tr>
<tr>
<td>Pairwise encoding</td>
<td>✓</td>
</tr>
<tr>
<td>Preserves long-term</td>
<td>✘</td>
</tr>
</tbody>
</table>
Addition

"Modeling" "word" "probabilities" "is" "really"

\[ f_{\theta} \rightarrow h \]
## Properties of addition as $f_θ$

<table>
<thead>
<tr>
<th></th>
<th>N-gram</th>
<th>Addition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order matters</td>
<td>✔</td>
<td>✘</td>
</tr>
<tr>
<td>Variable length</td>
<td>✘</td>
<td>✔</td>
</tr>
<tr>
<td>Differentiable</td>
<td>✘</td>
<td>✔</td>
</tr>
<tr>
<td>Pairwise encoding</td>
<td>✔</td>
<td>✘</td>
</tr>
<tr>
<td>Preserves long-term</td>
<td>✘</td>
<td>✔</td>
</tr>
</tbody>
</table>
Learning to model word probabilities

2. Modeling conditional probabilities
Learning to model word probabilities

2. Modeling conditional probabilities

Desirable properties for $g_\theta$:

- Individual changes can have large effects (non-linear/deep)
- Returns a probability distribution
Summary

N-grams and simple aggregation do not meet the requirements for modeling sequences.
How can we build a deep network that meets our requirements?
Recurrent Neural Networks (RNNs)

Persistent state variable $h$ stores information from the context observed so far.

\[ h_t = \tanh(W_h h_{t-1} + W_x x_t) \]
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.

Elman (1991)
Recurrent Neural Networks (RNNs)

Input next word in sentence $x_1$

Elman (1991)
Recurrent Neural Networks (RNNs)

Elman (1991)
Recurrent Neural Networks (RNNs)
Recurrent Neural Networks (RNNs)

Weights are shared over time steps

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

For the sentence: $$\mathcal{L}_\theta(y, \hat{y}) = -\sum_{t=1}^{T} y_t \log \hat{y}_t$$

With parameters $$\theta = \{W_y, W_x, W_h\}$$
Differentiating wrt $W_y$, $W_x$ and $W_h$

\[
    h_t = \tanh(W_h h_{t-1} + W_x x_t)
\]

\[
    p(x_{t+1}) = \text{softmax}(W_y h_t)
\]

\[
    \mathcal{L}_\theta(y, \hat{y})_t = -y_t \log \hat{y}_t
\]
Differentiating wrt $W_y$

\[
 h_t = \tanh(W_h h_{t-1} + W_x x_t)
\]

\[
 p(x_{t+1}) = \text{softmax}(W_y h_t)
\]

\[
 L_\theta(y, \hat{y})_t = -y_t \log \hat{y}_t
\]

\[
 \frac{\partial L_\theta,t}{\partial W_y} = \frac{\partial L_\theta,t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial W_y} = (y_t - \hat{y}_t) h_t
\]
Differentiating wrt $W_h$

\[ h_t = \tanh(W_h h_{t-1} + W_x x_t) \]

\[ p(x_{t+1}) = \text{softmax}(W_y h_t) \]

\[ L_{\theta}(y, \hat{y})_t = -y_t \log \hat{y}_t \]

\[
\frac{\partial L_{\theta,t}}{\partial W_h} = \frac{\partial L_{\theta,t}}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial W_h}
\]
Differentiating wrt $W_h$

$$h_t = \tanh(W_h h_{t-1} + W_x x_t)$$

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

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

$$\frac{\partial \mathcal{L}_{\theta,t}}{\partial W_h} = \frac{\partial \mathcal{L}_{\theta,t}}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial W_h}$$
Back propagating through time

\[
\frac{\partial h_t}{\partial W_h} = \frac{\partial h_t}{\partial W_h} + \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial W_h}
\]

\[
= \frac{\partial h_t}{\partial W_h} + \frac{\partial h_t}{\partial h_{t-1}} \left[ \frac{\partial h_{t-1}}{\partial W_h} + \frac{\partial h_{t-1}}{\partial h_{t-2}} \frac{\partial h_{t-2}}{\partial W_h} \right]
\]

\[
\ldots
\]

\[
= \sum_{k=1}^{t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W_h}
\]
Differentiating wrt $W_h$

\[ h_t = \tanh(W_h h_{t-1} + W_x x_t) \]
\[ p(x_{t+1}) = \text{softmax}(W_y h_t) \]
\[ \mathcal{L}_\theta(y, \hat{y})_t = -y_t \log \hat{y}_t \]

\[
\frac{\partial \mathcal{L}_\theta, t}{\partial W_h} = \frac{\partial \mathcal{L}_\theta, t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial W_h} 
\]

\[
\frac{\partial h_t}{\partial W_h} = \sum_{k=1}^{t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W_h} 
\]
\[
= \sum_{k=1}^{t} \frac{\partial \mathcal{L}_\theta, t}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W_h} 
\]
Vanishing gradients

A simple example

\[ h_t = W_h h_{t-1} \]

\[ h_t = (W_h)^t h_0 \]

\[ h_t \to \infty \text{ if } |W_h| > 1 \]

\[ h_t \to 0 \text{ if } |W_h| < 1 \]

Hochreiter (1991), Bengio et al. (1994)
Vanishing gradients

A simple example

But RNNs bound $h$ with a tanh!

$$h_t = W_h h_{t-1} \quad \rightarrow \quad h_t = \tanh(W_h h_{t-1})$$
Vanishing gradients

\[ h_t = \tanh(W_h h_{t-1}) \]

\[
\frac{\partial h_t}{\partial h_{t-T}} = \left(1 - \tanh^2(W_h h_{t-1})\right) W_h \frac{\partial h_{t-1}}{\partial h_{t-T}}
\]

Python code snippet:

```python
import numpy as np
import matplotlib.pyplot as plt

def forward_backward_prop(w, T):
    hs = [0.5]
    for _ in range(T):
        hs.append(np.tanh(w*hs[-1]))
    dh = 1
    for t in range(T):
        dh = (1-hs[-1-t]**2) * w * dh
    return hs[:-1], dh

T = 10  # sequence length
wlim = 4  # limit of interval over weights w
results = []
ws = np.linspace(-wlim, wlim, 1000)
for w in ws:
    results.append(forward_backward_prop(w, T))

plt.plot(ws, [r[0] for r in results], label='RNN state')
plt.plot(ws, [r[1] for r in results], label='Gradients')
plt.legend()
```
<table>
<thead>
<tr>
<th></th>
<th>N-gram</th>
<th>Addition</th>
<th>RNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order matters</td>
<td>✔</td>
<td>✘</td>
<td>✔</td>
</tr>
<tr>
<td>Variable length</td>
<td>✘</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Differentiable</td>
<td>✘</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Pairwise encoding</td>
<td>✔</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>Preserves long-term</td>
<td>✘</td>
<td>✔</td>
<td>✘</td>
</tr>
</tbody>
</table>
Summary

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.
Long term dependencies are important

... Finally, Tim was planning to visit France on the final week of his journey. He was quite excited to try the local delicacies and had lots of recommendations for good restaurants and exhibitions. His first stop was, of course, the capital where he would meet his long-time Friend Jean-Pierre. In order to arrive for breakfast he took the early 5 AM train from London to ...
Long term dependencies are important

... Finally, Tim was planning to visit France on the final week of his journey. He was quite excited to try the local delicacies and had lots of recommendations for good restaurants and exhibitions. His first stop was, of course, the capital where he would meet his long-time Friend Jean-Pierre. In order to arrive for breakfast he took the early 5 AM train from London to ...
Long term dependencies are important

... Finally, Tim was planning to visit France on the final week of his journey. He was quite excited to try the local delicacies and had lots of recommendations for good restaurants and exhibitions. His first stop was, of course, the capital where he would meet his long-time Friend Jean-Pierre. In order to arrive for breakfast he took the early 5 AM train from London to ...

PARIS!
How can we capture long-term dependencies?
Long Short-Term Memory (LSTM) networks

RNN state update
Long Short-Term Memory (LSTM) networks

Hochreiter (1997), Gers et al. (1999)
1) Forget gate

\[ f_t^1 = \sigma(W_{f1} \cdot [h_{t-1}, x_t] + b_{f1}) \]
Long Short-Term Memory (LSTM) networks

2) Input gates

\[ f_t^2 = \sigma(W_{f2} \cdot [h_{t-1}, x_t] + b_{f2}) \odot \tanh(W_{f2} \cdot [h_{t-1}, x_t] + b_{f2}) \]
Long Short-Term Memory (LSTM) networks

3) Output gate

\[ h'_t = \sigma(W_{h'_t} \cdot [h_{t-1}, x_t] + b_{h'_t}) \odot \tanh(c_t) \]
Long Short-Term Memory (LSTM) networks

LSTM state update
Gated Recurrent Unit nets

GRU state update

GRU can be seen as a simplified LSTM.

Cho et al. (2014)
# Properties of LSTMs as $f_\theta$

<table>
<thead>
<tr>
<th></th>
<th>N-gram</th>
<th>Addition</th>
<th>RNN</th>
<th>LSTM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order matters</td>
<td>✔</td>
<td>✘</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Variable length</td>
<td>✘</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Differentiable</td>
<td>✘</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Pairwise encoding</td>
<td>✔</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
</tr>
<tr>
<td>Preserves long-term</td>
<td>✘</td>
<td>✔</td>
<td>✘</td>
<td>✔</td>
</tr>
</tbody>
</table>
Summary

LSTMs and GRUs overcome the vanishing gradient problem by making use of sophisticated gating mechanisms.

As a result these models are ubiquitous across machine learning research.
Generating Sequences

Sequences and Recurrent Neural Networks
Using a trained model

During training we focussed on optimising the log probability estimates produced by our model.

\[ \mathcal{L}_\theta(y, \hat{y})_t = -y_t \log \hat{y}_t \]

This means at test time we can use it to evaluate the probability of a new sentence. This, however, is arguably not very interesting.

An alternative use case of our trained model is sequence generation.
Generating sequences with RNNs

ŷ is a probability distribution over possible words that we can sample from.
Generating sequences with RNNs

The sampled $\hat{y}$ is the input to the next iteration of the network.
Generating sequences with RNNs
Generating sequences with RNNs
Images as sequences
Images as sequences: PixelRNN

\[ p(x) = \prod_{i=1}^{n^2} p(x_i | x_1, \ldots, x_{i-1}) \]
Softmax Sampling

$P(w_1)$
Softmax Sampling

\[ P(w_1) \]

\[ P(w_2 | w_1) \]
Softmax Sampling

\[
P(w_1)
\]
\[
P(w_2|w_1)
\]
\[
P(w_3|w_2, w_1)
\]
Softmax Sampling

\[
P(w_1) \\
P(w_2|w_1) \\
P(w_3|w_2, w_1) \\
P(w_4|w_3, w_2, w_1)
\]
Softmax Sampling
Softmax Sampling
Softmax Sampling
Softmax Sampling
Softmax Sampling
Softmax Sampling
Softmax Sampling
Softmax Sampling
Softmax Sampling
Softmax Sampling
Softmax Sampling
Softmax Sampling
Softmax Sampling
Softmax Sampling
Softmax Sampling
Softmax Sampling

![Image of a bird with a histogram graph]
Softmax Sampling
Softmax Sampling

![Image of a bird](image)

![Histogram](histogram)
Softmax Sampling
Images as sequences: PixelRNN

Van den Oord et al. (2016)
Images as sequences

Pixel-by-pixel

Group-by-group
Reed et al. “Parallel Multiscale Autoregressive Density Estimation.”
Natural language as sequences
Language as sequences:
Sequence-to-sequence models
Language as sequences: Sequence-to-sequence models
Language as sequences:
Sequence-to-sequence models
This setup is flexible

One to one

One to many

Many to one

Many to many
Seq2seq has a wide range of applications

3. **Speech** [Chorowsky et al, NIPS DL 2014][Chan et al, arxiv 2015]
4. **Parsing** [Vinyals & Kaiser et al, NIPS 2015]
6. **Video Generation** [Srivastava et al, ICML 2015]
7. **Geometry** [Vinyals & Fortunato & Jaitly, NIPS 2015]
Google Neural Machine Translation

Wu et al, 2016
(Kalchbrenner et al, 2013; Sutskever et al, 2014; Cho et al, 2014; Bhadanau et al, 2014; ...)

Encoder

$e_0 \rightarrow e_1 \rightarrow e_2 \rightarrow e_3 \rightarrow e_4 \rightarrow e_5 \rightarrow e_6$

Decoder

$d_0 \rightarrow d_1 \rightarrow d_2 \rightarrow d_3$

Knowledge is power <end>
Google Neural Machine Translation

Closes gap between old system and human-quality translation by 58% to 87%.
Image captioning

\[ p(\text{language}_1 \mid \text{language}_2) \to p(\text{language}_1 \mid \text{image}) \]
Image captioning

Human: A brown dog laying in a red wicker bed.

Best Model: A small dog is sitting on a chair.

Initial Model: A large brown dog laying on top of a couch.
Human: A man outside cooking with a sub in his hand.

Best Model: A man is holding a sandwich in his hand.

Initial Model: A man cutting a cake with a knife.
Human: Someone is using a small grill to melt his sandwich.

Best Model: A person is cooking some food on a grill.

Initial Model: A pizza sitting on top of a white plate.
Image captioning

Human: A woman holding up a yellow banana to her face.

Best Model: A woman holding a banana up to her face.

Initial Model: A close up of a person eating a hot dog.
Audio waves as sequences
Audio waves as sequences: convolutions

Van den Oord et al. (2016)
## Properties of Convolutions as $f_\theta$

<table>
<thead>
<tr>
<th>Feature</th>
<th>N-gram</th>
<th>Addition</th>
<th>RNN</th>
<th>LSTM</th>
<th>Conv</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order matters</td>
<td>✔</td>
<td>❌</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Variable length</td>
<td>❌</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Differentiable</td>
<td>❌</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Pairwise encoding</td>
<td>✔</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
<td>❌</td>
</tr>
<tr>
<td>Preserves long-term</td>
<td>❌</td>
<td>✔</td>
<td>❌</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
Policies as sequences
Policies as sequences

AlphaStar Architecture

Observation
- Spatial
- General
- Units

Torsos
- ResNet Stack
- Scalar Torso
- Transformer

Core
- Core LSTM
- Avg

Heads
- Function Head
- Args Head

Action
- Function
- Arguments

Embedded Units
Attention for sequences: transformers
Transformers

Convolution

Transformer

Vaswani et al. (2017)
Transformers

Vaswani et al. (2017)
Transformers

Vaswani et al. (2017)
Transformers

Vaswani et al. (2017)
Transformers

Vaswani et al. (2017)
GPT2

- Transformer-based language model with 1.5 billion parameters for next-word prediction
- Dataset: 40GB of text data from 8M websites
- Adapts to style and content of arbitrary conditioning input

https://openai.com/blog/better-language-models/#sample1

Radford et al. (2019)
<table>
<thead>
<tr>
<th>SYSTEM PROMPT (HUMAN-WRITTEN)</th>
<th>MODEL COMPLETION (MACHINE-WRITTEN, 10 TRIES)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>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.</strong></td>
<td>The scientist named the population, after their distinctive horn, Ovid’s Unicorn. These four-horned, silver-white unicorns were previously unknown to science.</td>
</tr>
<tr>
<td>Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.</td>
<td></td>
</tr>
<tr>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>Pérez and the others then ventured further into the valley. “By the time we reached the top of one peak, the water looked blue, with some crystals on top,” said Pérez.</td>
<td></td>
</tr>
<tr>
<td>Pérez and his friends were astonished to see the unicorn herd. These creatures could be seen from the air without having to move too much to see them – they were so close they could touch their horns.</td>
<td></td>
</tr>
</tbody>
</table>
## Properties of Transformers as $f_\theta$

<table>
<thead>
<tr>
<th></th>
<th>N-gram</th>
<th>Addition</th>
<th>RNN</th>
<th>LSTM</th>
<th>Conv</th>
<th>Transf.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order matters</td>
<td>✔</td>
<td>✘</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Variable length</td>
<td>✘</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Differentiable</td>
<td>✘</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Pairwise encoding</td>
<td>✔</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✘</td>
<td>✔</td>
</tr>
<tr>
<td>Preserves long-term</td>
<td>✘</td>
<td>✔</td>
<td>✘</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
Evolution of language modeling

Sutskever et al, 2011, RNNs

[ ] while he was giving attention to the second advantage of school building a 2-for-2 stool killed by the Cultures saddled with a halfsuit defending the Bharatiya Fernall’s office

Radford et al, 2019, GPT2

[ Miley Cyrus was caught shoplifting from Abercrombie and Fitch on Hollywood Boulevard today.] The singer was wearing a black hoodie with the label ‘Blurred Lines’ on the front and ‘Fashion Police’ on the back.
Summary

1 Motivation: Sequences are everywhere but modeling them is hard!

2 Covered different approaches:
   a. N-Grams
   b. RNNs
   c. LSTMs & GRUs
   d. Dilated convolutions and Transformers

3 These models are flexible and can be applied to a wide range of tasks across machine learning.
Thank you
Questions