Neural models of language
Logistics

- Assignment 1: due Feb 11, 2022
- Assignment 2: release Feb 12, 2022
- Lecture delivery:
  - Online (as is) until Feb 18
  - Reading week break: Feb 21-25 (no lectures)
  - In-person Feb 28th onwards

- Final exam: planned in-person
Neural networks

• Introduction
• Word-level representations
• Neural language models
• Recurrent neural networks
• Sequence-to-sequence modelling
• Some recent developments

With material from Phil Blunsom, Piotr Mirowski, Adam Kalai, and James Zou
Artificial neural networks

- Artificial neural networks (ANNs) were (kind of) inspired from neurobiology (Widrow and Hoff, 1960).
  - Each unit has many inputs (dendrites), one output (axon).
  - The nucleus fires (sends an electric signal along the axon) given input from other neurons.
  - ‘Learning’ occurs at the synapses that connect neurons, either by amplifying or attenuating signals.
Perceptron: an artificial neuron

- Each neuron calculates a **weighted sum** of its inputs and compares this to a threshold, $\tau$. If the sum exceeds the threshold, the neuron fires.
- Inputs $a_i$ are activations from adjacent neurons, each weighted by a parameter $w_i$.

\[
x = \sum_{i=1}^{M} w_i a_i
\]

McCullogh-Pitts model

\[
\text{If } x > \tau, S := 1 \\
\text{Else, } S := 0
\]
Perceptron output

• Perceptron output is determined by activation functions, $g()$, which can be non-linear functions of weighted input.
• Popular activation functions include \text{tanh} and the sigmoid:

\[ g(x) = \sigma(x) = \frac{1}{1 + e^{\beta x}} \]

• The sigmoid’s derivative is the easily computable $\sigma' = \sigma \cdot (1 - \sigma)$.
Rectified Linear Units (ReLUs)

• Since 2011, the ReLU $S = g(x) = \max(0, x)$ has become more popular.
  • More biologically plausible, sparse activation, limited (vanishing) gradient problems, efficient computation.

• A smooth approximation is the softplus $\log(1 + e^x)$, which has a simple derivative $1/(1 + e^{-x})$

• Why do we care about the derivatives?

Perceptron learning

- Weights are adjusted in **proportion to the error** (i.e., the **difference** between the desired, $y$, and the actual output, $S$).
- The **derivative** $g'$ allows us to assign blame proportionally.

- Given a small learning rate, $\alpha$ (e.g., 0.05), we can repeatedly adjust each of the weight parameters by

$$w_j \leftarrow w_j + \alpha \cdot \sum_{i=1}^{R} Err_i \cdot g'(x_i) \cdot a_j[i]$$

where $Err_i = (y_i - S_i)$, among $R$ training examples.

Assumes mean-square error objective
Threshold perceptrons and XOR

- Some relatively simple logical functions cannot be learned by threshold perceptrons (since they are not linearly separable).
Artificial neural networks

• Complex functions can be represented by layers of perceptron (multi-layer perceptron, MLPs).

• Inputs are passed to the input layer.

• Activations are propagated through hidden layers to the output layer.

• MLPs are quite robust to noise, and are trained specifically to reduce error.
Deep

It's a cat. 'hidden' representations are learned here. Can we find hidden patterns in words?
Words

• Given a corpus with $D$ (e.g., $= 100K$) unique words, the classical approach is to uniquely assign each word with an index in $D$-dimensional vectors (‘one-hot’ representation).

| lugubrious | 0 | 0 | 0 | 0 | .. | 0 | 1 | 0 | ... | 0 |

• Classic word-feature representation assigns features to each index in a much denser vector.
  • E.g., ‘VBG’, ‘negative’, ‘age-of-acquisition’.

| 1 | 0.8 | 2.5 | 0.81 | ... | 99 |

| $d << D$ |

• Can we learn a dense representation? What will it give us?
Learning word semantics

"You shall know a word by the company it keeps."
— J.R. Firth (1957)

\[ P(w_t = \text{lugubrious}|w_{t-1} = \text{feeling}, w_{t-2} = \text{been}, ...) \]

been feeling lugubrious all day
felt a lugubrious sadness in

... Here, we’re predicting the center word given the context. This is called the ‘continuous bag of words’ (CBOW) model\(^1\).

Continuous bag of words (1 word context)

Note: we have two vector representations of each word:
\[ \nu_w = x^T W_i \quad (w^{th} \text{ row of } W_i) \]
\[ V_w = W_o^T y \quad (w^{th} \text{ col of } W_o) \]

‘softmax’: \[ P(w_o | w_i) = \frac{\exp(V^\top_w \nu_w)}{\sum_{w=1}^{W} \exp(V^\top_w \nu_w)} \]

Where \( \nu_w \) is the ‘input’ vector for word \( w \),
\( V_w \) is the ‘output’ vector for word \( w \),
Continuous bag of words \( (C \text{ words context}) \)

- If we want to use more context, \( C \), we need to change the network architecture somewhat.
  - Each input word will produce one of \( C \) embeddings
  - We just need to add an intermediate layer, usually this just averages the embeddings.

been feeling lugubrious all
felt a lugubrious sadness

...
Skip-grams

• **Skip-grams** invert the task – we predict context words given the current word.

• According to Mikolov, **Skip-gram**: works well with small amounts of training data, represents rare words.

**CBOW**: several times faster to train, slightly better accuracy for frequent words

Actually doing the learning

- Given $H$-dimensional embeddings, and $V$ word types, our parameters, $\theta$, are:

\[
\theta = \begin{bmatrix}
\nu_a \\
\nu_{aardvark} \\
\vdots \\
\nu_{zymurgy} \\
V_a \\
V_{aardvark} \\
\vdots \\
V_{zymurgy}
\end{bmatrix} \in \mathbb{R}^{2V \times H}
\]
Actually doing the learning

We have many options. Gradient descent is popular. We want to optimize, given $T$ tokens of training data,

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} \sum_{-c<j<c,j \neq 0} \log P(w_{t+j}|w_t)$$

And we want to update vectors $V_{w_{t+j}}$ then $v_{w_t}$ within $\theta$

$$\theta^{(new)} = \theta^{(old)} - \alpha \nabla_{\theta} J(\theta)$$

so we’ll need to take the derivative of the (log of the) softmax function:

$$P(w_o|w_i) = \frac{\exp(V_{w_o}^T v_{w_i})}{\sum_{w=1}^{W} \exp(V_w^T v_{w_i})}$$

Where $v_w$ is the ‘input’ vector for word $w$, and $V_w$ is the ‘output’ vector for word $w$. 
Actually doing the learning

We need the derivative of the (log of the) softmax function:

\[
\frac{\delta}{\delta \nu_{w_t}} \log P(w_{t+j}|w_t) = \frac{\delta}{\delta \nu_{w_t}} \log \left( \frac{\exp(V_{w_{t+j}}^T \nu_{w_t})}{\sum_{w=1}^W \exp(V_w^T \nu_{w_t})} \right) \\
= \frac{\delta}{\delta \nu_{w_t}} \left[ \log \exp(V_{w_{t+j}}^T \nu_{w_t}) - \log \sum_{w=1}^W \exp(V_w^T \nu_{w_t}) \right] \\
= V_{w_{t+j}} - \frac{\delta}{\delta \nu_{w_t}} \log \sum_{w=1}^W \exp(V_w^T \nu_{w_t}) \\
\text{[apply the chain rule } \frac{\delta f}{\delta \nu_{w_t}} = \frac{\delta f}{\delta z} \frac{\delta z}{\delta \nu_{w_t}} \text{]} \\
= V_{w_{t+j}} - \sum_{w=1}^W p(w|w_t) V_w
\]

Using word representations

Without a latent space,

\[
\text{lugubrious} = [0,0,0, \ldots, 0,1,0, \ldots, 0], \&
\]

\[
\text{sad} = [0,0,0, \ldots, 0,0,1, \ldots, 0] \text{ so}
\]

**Similarity** = \(\cos(x, y) = 0.0\)

In latent space,

\[
\text{lugubrious} = [0.8,0.69,0.4, \ldots, 0.05]_H, \&
\]

\[
\text{sad} = [0.9,0.7,0.43, \ldots, 0.05]_H \text{ so}
\]

**Similarity** = \(\cos(x, y) = 0.9\)

Reminder:

\[
\cos(u, v) = \frac{u \cdot v}{||u|| \times ||v||}
\]
Skip-grams with negative sampling

• The default process is inefficient.
  • For one – **what a waste of time!**
    We don’t want to update $H \times D$ weights!
  • For two – **we want to avoid confusion!**
    ‘Hallucinated’ (negative) contexts should be minimized.

• For the observed (true) pair (**lugubrious**, **sadness**),
  only the output neuron for **sadness** should be 1, and
  all $D − 1$ others should be 0.

• Mathematical Intuition:
  $$P (w_0 | w_c) = \frac{\exp(v_0^T v_c)}{\sum_{w=1}^D \exp(v_w^T v_c)}$$
  Computationally infeasible
Skip-grams with negative sampling

• We want to **maximize** the association of *observed* (positive) contexts:
  
  *lugubrious*  *sad*
  *lugubrious*  *feeling*
  *lugubrious*  *tired*

• We want to **minimize** the association of *‘hallucinated’* contexts:
  
  *lugubrious*  *happy*
  *lugubrious*  *roof*
  *lugubrious*  *truth*
Skip-grams with negative sampling

• Choose a small number $k$ of ‘negative’ words, and just update the weights for the ‘positive’ word plus the $k$ ‘negative’ words.
  • $5 \leq k \leq 20$ can work in practice for fewer data.
  • For $D = 100K$, we only update 0.006% of the weights in the output layer.

$$J(\theta) = \log \sigma(v_o^T v_c) + \sum_{i=1}^{k} E_{j \sim P(w)}[\log \sigma(-v_j^T v_c)]$$

• Mimno and Thompson (2017) choose the top $k$ words by modified unigram probability:

$$P^*(w_{t+1}) = \frac{C(w_{t+1})^3}{\sum_w C(w)^3}$$

Smell the GloVe

• **GloVe** (‘Global Vectors’) is an alternative method of obtaining word embeddings.
  - Instead of predicting words at particular positions, look at the **co-occurrence matrix**.

\[
X = \begin{bmatrix}
I & like & enjoy & deep & learning & NLP & flying \\
0 & 2 & 1 & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 \\
\end{bmatrix}
\]

- Word \( w_i \) occurs \( X_{i,j} (= X_{j,i}) \) times with word \( w_j \), within some context window (e.g., 10 words, a sentence, …).

Smell the GloVe

- Populating the co-occurrence matrix requires a complete pass through the corpus, but needs only be done once.
- Let $P_{i,j} = P(w_j|w_i) = X_{i,j}/X_i$.

Table 1: Co-occurrence probabilities for target words ice and steam with selected context words from a 6 billion token corpus. Only in the ratio does noise from non-discriminative words like water and fashion cancel out, so that large values (much greater than 1) correlate well with properties specific to ice, and small values (much less than 1) correlate well with properties specific of steam.

<table>
<thead>
<tr>
<th>Probability and Ratio</th>
<th>$k = solid$</th>
<th>$k = gas$</th>
<th>$k = water$</th>
<th>$k = fashion$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(k</td>
<td>ice)$</td>
<td>$1.9 \times 10^{-4}$</td>
<td>$6.6 \times 10^{-5}$</td>
<td>$3.0 \times 10^{-3}$</td>
</tr>
<tr>
<td>$P(k</td>
<td>steam)$</td>
<td>$2.2 \times 10^{-5}$</td>
<td>$7.8 \times 10^{-4}$</td>
<td>$2.2 \times 10^{-3}$</td>
</tr>
<tr>
<td>$P(k</td>
<td>ice)/P(k</td>
<td>steam)$</td>
<td>8.9</td>
<td>$8.5 \times 10^{-2}$</td>
</tr>
</tbody>
</table>

Aside – smell the GloVe

• Minimize $J = \sum_{i,j=1}^{V} f(X_{i,j}) \left( \nu_{wi}^T \nu_{wj} + b_i + \tilde{b}_j - \log X_{i,j} \right)^2$
  where, $b_i$ and $\tilde{b}_j$ are input and output bias terms associated with $w_i$ and $w_j$, respectively

• Weighting function $f(X_{i,j})$:

1. $f(0) = 0$. If $f$ is viewed as a continuous function, it should vanish as $x \to 0$ fast enough that the $\lim_{x\to 0} f(x) \log^2 x$ is finite.

2. $f(x)$ should be non-decreasing so that rare co-occurrences are not overweighted.

3. $f(x)$ should be relatively small for large values of $x$, so that frequent co-occurrences are not overweighted.
Aside – evaluation

- **Intrinsic evaluation**: popular though perhaps dishonest method was to cherry-pick a few $k$-nearest neighbours examples that match expectations.

- **Extrinsic evaluation**: embed resulting vectors into a variety of tasks\(^1,^2\).

1. https://gluebenchmark.com/tasks
Linguistic regularities in vector space

Trained on the Google news corpus with over 300 billion words.
Linguistic regularities in vector space

(from GloVe)
Linguistic regularities in vector space

<table>
<thead>
<tr>
<th>Expression</th>
<th>Nearest token</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paris – France + Italy</td>
<td>Rome</td>
</tr>
<tr>
<td>Bigger – big + cold</td>
<td>Colder</td>
</tr>
<tr>
<td>Sushi – Japan + Germany</td>
<td>bratwurst</td>
</tr>
<tr>
<td>Cu – copper + gold</td>
<td>Au</td>
</tr>
<tr>
<td>Windows – Microsoft + Google</td>
<td>Android</td>
</tr>
</tbody>
</table>

**Analogies:** apple:apples :: octopus:octopodes

**Hypernymy:** shirt:clothing :: chair:furniture

**Semantic:** queen – king ≈ woman – man
Importance of in-domain data

However, in word2vec trained on Google News, 
man:woman::programmer:homemaker.
Biases: let’s talk about gender

Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings

Tolga Bolukbasi¹, Kai-Wei Chang², James Zou², Venkatesh Saligrama¹,², Adam Kalai²
¹Boston University, 8 Saint Mary’s Street, Boston, MA
²Microsoft Research New England, 1 Memorial Drive, Cambridge, MA
tolgb@bu.edu, kw@kwchang.net, jameszou@gmail.com, srv@bu.edu, adam.kalai@microsoft.com

Abstract
The blind application of machine learning runs the risk of amplifying biases present in data. Such a danger is facing us with word embedding, a popular framework to represent text data as vectors which has been used in many machine learning and

<table>
<thead>
<tr>
<th>Extreme she</th>
<th>Extreme he</th>
<th>Gender stereotype she-he analogies</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. homemaker</td>
<td>1. maestro</td>
<td>registered nurse-physician</td>
</tr>
<tr>
<td>2. nurse</td>
<td>2. skipper</td>
<td>interior designer-architect</td>
</tr>
<tr>
<td>3. receptionist</td>
<td>3. protege</td>
<td>feminism-conservatism</td>
</tr>
<tr>
<td>4. librarian</td>
<td>4. philosopher</td>
<td>vocalism-guitarist</td>
</tr>
<tr>
<td>5. socialite</td>
<td>5. captain</td>
<td>diva-superstar</td>
</tr>
<tr>
<td>6. hairdresser</td>
<td>6. architect</td>
<td>fancy-pants</td>
</tr>
<tr>
<td>7. nanny</td>
<td>7. financier</td>
<td>petite-lanky</td>
</tr>
<tr>
<td>8. bookkeeper</td>
<td>8. warrior</td>
<td>charming-affable</td>
</tr>
<tr>
<td>9. stylist</td>
<td>9. broadcaster</td>
<td>lovely-brilliant</td>
</tr>
<tr>
<td>10. housekeeper</td>
<td>10. magician</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>housewife-shopkeeper</td>
</tr>
<tr>
<td></td>
<td></td>
<td>softball-baseball</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cosmetics-pharmaceuticalals</td>
</tr>
<tr>
<td></td>
<td></td>
<td>petite-lanky</td>
</tr>
<tr>
<td></td>
<td></td>
<td>charming-affable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>lovely-brilliant</td>
</tr>
</tbody>
</table>

Gender appropriate she-he analogies

|                          |                         |                                                               |
|                          |                         | sister-brother                                               |
|                          |                         | mother-father                                               |
|                          |                         | ovarian cancer-prostate cancer convent-monastery             |
1. Hand-pick words $S_0$ that are 'gender definitional'. 'Neutral' words are the complement, $N = V \setminus S_0$.
2. Project away gender subspace from gender-neutral words, $w := w - w \cdot B$ for $w \in N$, where $B$ is the gender subspace.
Solution?

2. Project away gender subspace from gender-neutral words, \( w := w - w \cdot B \) for \( w \in N \), where \( B \) is the gender subspace.
Results

- Generate many analogies, see which ones preserve gender stereotypes.
  
  He: *Blue* :: She: ?
  He: *Doctor* :: She: ?
  *He: Brother* :: She: ?

# stereotypic analogies

# appropriate analogies

# analogies generated
NEURAL LANGUAGE MODELS
Trigram models

- **CBOW**: prediction of current word $w_t$ given $w_{t-1}$.
- Let’s reconsider predicting $w_t$ given multiple $w_{t-j}$?
  - I.e., let’s think about **language modelling**.

Here:
- $w_i$ is a one-hot vector,
- $p_t$ is a distribution, and
- $|w_i| = |p_t| = |V|$ (i.e., the size of the vocabulary)

$$h = g(W_1 x + c)$$
$$y = W_0 h + b$$

$$h_t = g(W_t [w_{t-2} ; w_{t-1}] + c)$$
$$p_t = \text{softmax}(W_0 h_t + b)$$
Sampling from trigram models

- Since \( p_t \sim P(w_t | w_{t-2} \ w_{t-1}) \), we just feed forward and sample from the output vector.
Training trigram models

• Here’s one approach:

1. Randomly choose a batch (e.g., 10K consecutive words)
2. Propagate words through the current model
3. Obtain word likelihoods (loss)
4. Back-propagate loss
5. Gradient step to update model
6. Go to (1)
Training trigram models

• The typical training objective is the cross entropy (see Lecture 4) of the corpus $C$ given the model $M$:

\[
\mathcal{F} = H(C; M) = -\frac{\log_2 P_M(C)}{||C||}
\]

Minimize

Maximize

\[
\log_2 P_M(C) = \log_2 \prod_{t=0}^{T} P(w_t) = \sum_{t=0}^{T} \log_2 P(w_t)
\]

\[
\log_2 P(w_t) = w_t^\top \log p_t
\]

Here:

• $w_t$ is a one-hot vector, and
• $p_t$ is a distribution.

\[
h_t = g(W_t[w_{t-2}; w_{t-1}] + c)
\]

\[
p_t = \text{softmax}(W_0 h_t + b)
\]
Training trigram models

- Compute our gradients, using $\mathcal{F} = -\frac{\log_2 P_M(C)}{\|C\|}$ and $\log_2 P(w_t) = w_t^T \log p_t$ and back-propagate.

\[ h_t = g(W_I[w_{t-2}; w_{t-1}] + c) \]
\[ p_t = \text{softmax}(W_O h_t + b) \]

Here:
- $w_i$ is a one-hot vector, and
- $p_t$ is a distribution.
So what?

- 🎉 Neural language models of this type:
  - Can generalize better than MLE LMs to unseen $n$-grams,
  - Can use *semantic* information as in word2vec.

\[
P(\text{the cat sat on the } \text{mat}) \approx P(\text{the cat sat on the } \text{rug})\]

- 😞 Neural language models of this type:
  - Can take *relatively* long to train. “GPUs kill the Earth.”
  - Number of parameters scale poorly with increasing context.

*Let’s improve both of these issues...*
Dealing with that bottleneck

• Traditional datasets for neural language modeling include:
  • AP News (14M tokens, 17K types)
  • HUB-4 (1M tokens, 25K types)
  • Google News (6B tokens, 1M types)
  • Wikipedia (3.2B tokens, 2M types)

• Datasets for medical/clinical LM include:
  • EMRALD/ICES (3.5B tokens, 13M types)

• Much of the computational effort is in the initial embedding, and in the softmax.
  • Can we simplify and speed up the process?
Dealing with that bottleneck

• **Replace** rare words with `<out-of-vocabulary>` token.
• **Subsample** frequent words.

• Hierarchical softmax.
• Noise-contrastive estimation.
• Negative sampling.

Hierarchical softmax with grouping

• Group words into distinct classes, $c$, e.g., by frequency.
  • E.g., $c_1$ is top 5% of words by frequency, $c_2$ is the next 5%, ...

• Factorize $p(w_o | w_i) = p(c | w_i)p(w_o | w_i, c)$

\[
\text{softmax}: P(w_o | w_i) = \frac{\exp(V_{w_o}^T v_{w_i})}{\sum_{w=1}^W \exp(V_w^T v_{w_i})}
\]

Where

- $v_w$ is the ‘input’ vector for word $w$,
- $V_w$ is the ‘output’ vector for word $w$,

\[
\frac{\exp(c_j v_{w_i})}{\sum_c \exp(c v_{w_i})} \times \frac{\exp(V_{w_o}^T v_{w_i})}{\sum_{w \in c} \exp(V_w^T v_{w_i})}
\]

[Mikolov et al, 2011, Auli et al, 2013]
RECURRENT NEURAL NETWORKS
Statistical language models

• Probability is conditioned on (window of) n previous words*

• A necessary (but incorrect) Markov assumption: each observation only depends on a short linear history of length $L$.

$$P(w_n|w_{1:(n-1)}) \approx P(w_n|w_{(n-L+1):(n-1)})$$

• Probabilities are estimated by computing unigrams and bigrams

$$P(s) = \prod_{i=1}^{t} P(w_i|w_{i-1})$$

$$P(s) = \prod_{i=2}^{t} P(w_i|w_{i-2}w_{i-1})$$

*From Lecture 2
Statistical language models

• Using higher n-gram counts (with smoothing) improves performance*

• *Computational burden*: too many n-grams (combinations)
  • Infeasible RAM requirements

• *RNN intuition*:
  • Use the **same set of weight** parameters for each word (or across all time steps)
  • Condition the neural network on all previous words (or time steps)
  • Memory requirement now scales with number of words

*From Lecture 2
Recurrent neural networks (RNNs)

- An RNN has **feedback** connections in its structure so that it ‘remembers’ previous states, when reading a sequence.

Elman network feed hidden units back
Jordan network (not shown) feed output units back

Ground Truth

Backpropagate
RNNs: Unrolling the $h_i$

- Copies of the same network can be applied (i.e., unrolled) at each point in a time series.
- These can be applied to various tasks.

\[
h_t = g(W_1[h_{t-1}; x] + c) \\
y_t = W_0 h_t + b
\]
RNNs: One time step snapshot

Two riders approaching horses.

- Given a list of word vectors $X$: $x_1, x_2, \ldots, x_t, x_{t+1}, \ldots, x_T$

- At a single time-step:

  $h_t = g([W_{hh} h_{t-1} + W_{hx} x_t] + c)$

  $h_t = g(W_1 [h_{t-1}; x_t] + c)$ (equivalent notation)

  $\hat{y}_t = \text{softmax} (W_{hy} h_t + b)$

```python
import numpy as np

def softmax(x):
    f_x = np.exp(x) / np.sum(np.exp(x))
    return f_x

class RNN:
    # ...
    def step(self, x, is_normalized=False):
        # update the hidden state
        self.h = np.tanh(np.dot(self.W_hh, self.h) + np.dot(self.W_xh, x))

        # compute the output vector
        y = np.dot(self.W_yh, self.h)

        return softmax(y) if is_normalized else y
```

$P(x_{t+1} = v_j | x_t, \ldots, x_1) = \hat{y}_{t,j}$
RNNs: Training

- Given a list of word vectors $X$: $x_1, x_2, \ldots, x_t, x_{t+1}, \ldots, x_T$

Two riders .. approaching .. horses.

$\hat{y} \in \mathbb{R}^{|V|}$ is a probability distribution over the vocabulary

The output $\hat{y}_{t,j}$ is the word (index) prediction of the next word ($x_{t+1}$)

**Evaluation**

- Same **cross-entropy** loss function

$$J^{(t)}(\theta) = - \sum_{j=1}^{|V|} y_{t,j} \log \hat{y}_{t,j}$$

- **Perplexity**: $2^J$ (lower is better)
Sampling from a RNN LM

• If $|h_i| < |V|$, we’ve already reduced the number of parameters from the trigram NN.

• In ‘theory’, information is maintained in $h_i$ across arbitrary lengths of time...

$h_t = g([W_{hh}h_{t-1}; W_{hx}x_t] + c)$

$\hat{y}_t = \text{softmax} (W_{hy}h_t + b)$

Karpathy (2015),
The Unreasonable Effectiveness of Recurrent Neural Networks
RNNs and retrograde amnesia

• Unfortunately, **catastrophic forgetting** is common.

  • E.g., the **relevant** context in “The sushi the sister of your friend’s programming teacher told you about was...” has likely been **overwritten** by the time \( h_{13} \) is produced.

---

**Informational bottleneck**

---

RNNs and retrograde amnesia

• One challenge with RNNs is that the gradient decays quickly as one pushes it back in time. Can we store relevant information?

Imagery and sequence from http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Long short-term memory (LSTM)

- In each module, in an LSTM, there are four interacting neural network layers.

The cell state is a special vector stream that runs through the entire chain and stores the long-term information.
Long short-term memory (LSTM)

- In each **module**, in an LSTM, there are four interacting neural network layers.

  Gates decide what information should be withheld in the cell state. They are a **sigmoid** followed by a pointwise $\times$.
  Values near 0 block information; values near 1 pass information.
LSTM step 1: decide what to forget

- The **forget gate layer** compares \( h_{t-1} \) and the current input \( x_t \) to decide which elements in cell state \( C_{t-1} \) to keep and which to turn off.
  - E.g., the cell state might ‘remember’ the number (sing./plural) of the current subject, in order to predict appropriately conjugated verbs, but decide to forget it when a new subject is mentioned at \( x_t \).
  - (There’s scant evidence that such information is so explicit.)

\[
 f_t = \sigma \left( W_f \cdot [h_{t-1}, x_t] + b_f \right)
\]
LSTM step 2: decide what to store

- The **input gate layer** has two steps.
  - First, a sigmoid layer $\sigma$ decides which cell units to update.
  - Next, a `tanh` layer creates new candidate values $\tilde{C}_t$.
  - E.g., the $\sigma$ can turn on the ‘number’ units, and the `tanh` can push information on the current subject.
  - The $\sigma$ layer is important – we don’t want to push information on units (i.e., latent dimensions) for which we have no information.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$
LSTM step 3: update the cell state

• Update $C_{t-1}$ to $C_t$.
  • First, forget what we want to forget: multiply $C_{t-1}$ by $f_t$.
  • Then, create a ‘mask vector’ of information we want to store, $i_t \times \widetilde{C}_t$.
  • Finally, write this information to the new cell state $C_t$.

$$C_t = f_t \times C_{t-1} + i_t \times \widetilde{C}_t$$
LSTM step 4: output and feedback

• Output something, \( o_t \), based on the current \( x_t \) and \( h_{t-1} \).
• Combine the output with the cell to give your \( h_t \).
  • Normalize cell \( C_t \) on \([-1,1]\) using \( \tanh \) and combine with \( o_t \).

• In some sense, \( C_t \) is long-term memory and \( h_t \) is the short-term memory (hence the name).

\[
o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)
\]
\[
h_t = o_t \times \tanh(C_t)
\]
Variants of LSTMs

• There are various variations on LSTMs.
  • ‘Bidirectional LSTMs’ (and bidirectional RNNs generally), learn


CSC401/2511 – Spring 2022
Variants of LSTMs

• Gers & Schmidhuber (2000) add ‘peepholes’ that allow all sigmoids to read the cell state.

\[
\begin{align*}
    f_t &= \sigma (W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f) \\
    i_t &= \sigma (W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i) \\
    o_t &= \sigma (W_o \cdot [C_t, h_{t-1}, x_t] + b_o)
\end{align*}
\]

• We can **couple** the ‘forget’ and ‘input’ gates.
  • Joint decisioning is more efficient.

\[
C_t = f_t \cdot C_{t-1} + (1 - f_t) \cdot \tilde{C}_t
\]
Aside - Variants of LSTMs

- **Gated Recurrent units** (GRUs; *Cho et al. (2014)*) go a step further and also merge the cell and hidden states.

\[
\begin{align*}
    z_t &= \sigma(W_z \cdot [h_{t-1}, x_t]) \text{ Update gate} \\
    r_t &= \sigma(W_r \cdot [h_{t-1}, x_t]) \text{ Reset gate (0: replace units in } h_{t-1} \text{ with those in } x_t) \\
    \tilde{h}_t &= \tanh(W \cdot [r_t \cdot h_{t-1}, x_t]) \\
    h_t &= (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h}_t
\end{align*}
\]

- Which of these variants is best? Do the differences matter?
  - *Greff, et al. (2015)* do a nice comparison of popular variants, finding that they’re all about the same
  - *Jozefowicz, et al. (2015)* tested more than ten thousand RNN architectures, finding some that worked better than LSTMs on certain tasks.
Deep contextualized representations

- What does the word *play* mean?
NLM: the bigger is better trend

Cons:

- Deep learning == Deep pockets?
- Environmental impact: “training BERT on GPU is roughly equivalent to a trans-American flight”¹

¹ S. Emma, A. Ganesh, and A. McCallum. “Energy and policy considerations for deep learning in NLP. (2019)” [arxiv]
ELMo: Embeddings from Language Models

• Instead of a fixed embedding for each word type, ELMo considers the entire sentence before embedding each token.
• It uses a bi-directional LSTM trained on a specific task.
• Outputs are softmax probabilities on words, as before.
ELMo: Embeddings from Language Models

For each token, a L-layer biLM computes (2L+1) representations:

\[
R_k = \{x^{LM}_k, \overrightarrow{h}^{LM}_{k,j}, \overleftarrow{h}^{LM}_{k,j} \mid j = 1, \ldots, L\} \\
= \{h^{LM}_{k,j} \mid j = 0, \ldots, L\},
\]

- Task specific weighting produces the final embedding for word token \( k \).

\[
\text{ELMo}^{\text{task}}_{k} = E(R_k; \Theta^{\text{task}}) = \gamma^{\text{task}} \sum_{j=0}^{L} s^{\text{task}}_j h^{LM}_{k,j}
\]

- where \( R_K \) is the set of all \( L \) hidden layers, \( h_{k,j} \)
- \( s^{\text{task}}_j \) is the task’s weight on the layer, and
- \( \gamma^{\text{task}} \) is a weight on the entire task
ELMo: Embeddings from Language Models

1. Concatenate

2. Multiply by weight vectors

3. Sum

\[ \text{ELMO}_{k=1}^{\text{task}} \]
What does the word *play* mean?

<table>
<thead>
<tr>
<th>Source</th>
<th>Nearest Neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>GloVe</td>
<td>play, playing, game, games, played, players, plays, player, Play, football, multiplayer</td>
</tr>
<tr>
<td>biLM</td>
<td>Chico Ruiz made a spectacular <a href="https://arxiv.org/abs/1802.05365">play</a> on Alusik’s grounder {...}</td>
</tr>
<tr>
<td></td>
<td>Kieffer, the only junior in the group, was commended for his ability to hit in the clutch, as well as his all-round excellent play.</td>
</tr>
<tr>
<td></td>
<td>Olivia De Havilland signed to do a Broadway play for Garson {...}</td>
</tr>
<tr>
<td></td>
<td>{...} they were actors who had been handed fat roles in a successful <a href="https://arxiv.org/abs/1802.05365">play</a>, and had talent enough to fill the roles competently, with nice understatement.</td>
</tr>
</tbody>
</table>

Table 4: Nearest neighbors to “play” using GloVe and the context embeddings from a biLM.
ELMo: Embeddings from Language Models

<table>
<thead>
<tr>
<th>TASK</th>
<th>PREVIOUS SOTA</th>
<th>OUR BASELINE</th>
<th>ELMo + BASELINE</th>
<th>INCREASE (ABSOLUTE/RELATIVE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQuAD</td>
<td>Liu et al. (2017)</td>
<td>84.4</td>
<td>81.1</td>
<td>85.8</td>
</tr>
<tr>
<td>SNLI</td>
<td>Chen et al. (2017)</td>
<td>88.6</td>
<td>88.0</td>
<td>88.7 ± 0.17</td>
</tr>
<tr>
<td>SRL</td>
<td>He et al. (2017)</td>
<td>81.7</td>
<td>81.4</td>
<td>84.6</td>
</tr>
<tr>
<td>Coref</td>
<td>Lee et al. (2017)</td>
<td>67.2</td>
<td>67.2</td>
<td>70.4</td>
</tr>
<tr>
<td>NER</td>
<td>Peters et al. (2017)</td>
<td>91.93 ± 0.19</td>
<td>90.15</td>
<td>92.22 ± 0.10</td>
</tr>
<tr>
<td>SST-5</td>
<td>McCann et al. (2017)</td>
<td>53.7</td>
<td>51.4</td>
<td>54.7 ± 0.5</td>
</tr>
</tbody>
</table>

Table 1: Test set comparison of ELMo enhanced neural models with state-of-the-art single model baselines across six benchmark NLP tasks. The performance metric varies across tasks – accuracy for SNLI and SST-5; F1 for SQuAD, SRL and NER; average F1 for Coref. Due to the small test sizes for NER and SST-5, we report the mean and standard deviation across five runs with different random seeds. The “increase” column lists both the absolute and relative improvements over our baseline.

BERT: Bidirectional encoder representations from transformers

- Unlike ELMo, BERT is **deeply** bidirectional.
  - i.e., every embedding conditions every other in the next layer.

- This is difficult, because when predicting word $x_t$, you would already have ‘seen’ that word in modelling its own contexts.

Code and models: [https://github.com/google-research/bert](https://github.com/google-research/bert)

BERT: Bidirectional encoder representations from transformers

I’ve seen me already
BERT: Bidirectional encoder representations from transformers

• This can be solved by masking the word being predicted.

  Input: The man went to the [MASK] \_ \_ . He bought a [MASK] \_ \_ of milk .
  Labels: [MASK] \_ \_ = store; [MASK] \_ \_ = gallon

  • (actually, 80% we use [MASK]. 10% we replace the target word with another actual word; 10% we keep the word as-is, to bias ‘towards the observation’.)

• We can also predict other relationships, like whether one sentence follows another.

  Sentence A = The man went to the store.
  Sentence B = He bought a gallon of milk.
  Label = IsNextSentence

  Sentence A = The man went to the store.
  Sentence B = Penguins are flightless.
  Label = NotNextSentence

  • (actually, you can fine-tune on many different tasks)

**BERT: Bidirectional encoder representations from transformers**

What is the best contextualized embedding for “Help” in that context?
For named-entity recognition task CoNLL-2003 NER

<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Dev F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Layer</td>
<td>91.0</td>
</tr>
<tr>
<td>Last Hidden Layer</td>
<td>94.9</td>
</tr>
<tr>
<td>Sum All 12 Layers</td>
<td>95.5</td>
</tr>
<tr>
<td>Second-to-Last Hidden Layer</td>
<td>95.6</td>
</tr>
<tr>
<td>Sum Last Four Hidden</td>
<td>95.9</td>
</tr>
<tr>
<td>Concat Last Four Hidden</td>
<td>96.1</td>
</tr>
</tbody>
</table>

BERT: Bidirectional encoder representations from transformers

- The age of humans is over?

<table>
<thead>
<tr>
<th>Rank</th>
<th>Name</th>
<th>Model</th>
<th>URL Score</th>
<th>CoLA</th>
<th>SST-2</th>
<th>MRPC</th>
<th>STS-B</th>
<th>QQP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T5 Team - Google</td>
<td>T5</td>
<td>89.7</td>
<td>70.8</td>
<td>97.1</td>
<td>91.9/89.2</td>
<td>92.5/92.1</td>
<td>74.6/90.4</td>
</tr>
<tr>
<td>2</td>
<td>ALBERT-Team Google LanguageALBERT (Ensemble)</td>
<td></td>
<td>89.4</td>
<td>69.1</td>
<td>97.1</td>
<td>93.4/91.2</td>
<td>92.5/92.0</td>
<td>74.2/90.5</td>
</tr>
<tr>
<td>3</td>
<td>王玮</td>
<td>ALICE v2 large ensemble (Alibaba DAMO NLP)</td>
<td>89.0</td>
<td>69.2</td>
<td>97.1</td>
<td>93.6/91.5</td>
<td>92.7/92.3</td>
<td>74.4/90.7</td>
</tr>
<tr>
<td>4</td>
<td>Microsoft D365 AI &amp; UMD</td>
<td>FreeLB-RoBERTa (ensemble)</td>
<td>88.8</td>
<td>68.0</td>
<td>96.8</td>
<td>93.1/90.8</td>
<td>92.4/92.2</td>
<td>74.8/90.3</td>
</tr>
<tr>
<td>5</td>
<td>Facebook AI</td>
<td>RoBERTa</td>
<td>88.5</td>
<td>67.8</td>
<td>96.7</td>
<td>92.3/89.8</td>
<td>92.2/91.9</td>
<td>74.3/90.2</td>
</tr>
<tr>
<td>6</td>
<td>XLNet Team</td>
<td>XLNet-Large (ensemble)</td>
<td>88.4</td>
<td>67.8</td>
<td>96.8</td>
<td>93.0/90.7</td>
<td>91.6/91.1</td>
<td>74.2/90.3</td>
</tr>
<tr>
<td>7</td>
<td>Microsoft D365 AI &amp; MSR AI</td>
<td>MT-DNN-ensemble</td>
<td>87.6</td>
<td>68.4</td>
<td>96.5</td>
<td>92.7/90.3</td>
<td>91.1/90.7</td>
<td>73.7/89.9</td>
</tr>
<tr>
<td>8</td>
<td>GLUE Human Baselines</td>
<td>GLUE Human Baselines</td>
<td>87.1</td>
<td>66.4</td>
<td>97.8</td>
<td>86.3/80.8</td>
<td>92.7/92.6</td>
<td>59.5/80.4</td>
</tr>
<tr>
<td>9</td>
<td>Stanford Hazy Research</td>
<td>Snorkel MetaL</td>
<td>83.2</td>
<td>63.8</td>
<td>96.2</td>
<td>91.5/88.5</td>
<td>90.1/89.7</td>
<td>73.1/89.9</td>
</tr>
<tr>
<td>10</td>
<td>XLM Systems</td>
<td>XLM (English only)</td>
<td>83.1</td>
<td>62.9</td>
<td>95.6</td>
<td>90.7/87.1</td>
<td>88.8/88.2</td>
<td>73.2/89.8</td>
</tr>
</tbody>
</table>
Aside – ClosedAI

• There are, of course, alternatives.

• **FastText**: Represent each word as a bag of character-grams
  Code: [https://fasttext.cc](https://fasttext.cc)

• **ULMFit**: Model fine-tuning for classification tasks
  Code: [Here](https://arxiv.org/abs/1801.06146)

• **GPT-2/3**: Spooky, closed uni-directional model
  Paper: [Here](https://arxiv.org/abs/1607.04606)
  Blog: [Here](https://arxiv.org/abs/1607.04606)
Sharif Shameem @sharifshameem · Jul 19
Wow. I built a React dice component with GPT-3. This feels far more fun than writing JSX.

debuild.co
Describe your app. Clear Generate

- a button that says 'roll dice' and then displays its value as

The value is 3

// a button that says 'roll dice' and then displays its value
class App extends React.Component {
    constructor(props) {
        super(props);
        this.state = {
            value: 0
        }
    }
    render() {
        return (
            <div>
                <button onClick={this.turnDice}>Roll Dice</button>
                The value is {this.state.value}
            </div>
        );
    }
    turnDice() {
        this.setState({
            value: Math.floor(Math.random() * 6) + 1
        });
    }
}
The Open AI GPT Papers

- The GPT papers:
  - GPT (2018)
  - GPT2 (2019)
  - GPT3 (2020)

- Each builds on the predecessor
Approach: Model & Architectures

The three settings we explore for in-context learning:

**Zero-shot**
The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

- Translate English to French:
  - task description: "cheese =>
  - prompt

**One-shot**
In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

- Translate English to French:
  - task description: "see otter => loutre de mer
  - example: "cheese =>
  - prompt

**Few-shot**
In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

- Translate English to French:
  - task description: "see otter => loutre de mer, peppermint => menthe poivrée, plush giraffe => girafe peluche, cheese =>
  - examples
  - prompt

Traditional fine-tuning (not used for GPT3):

**Fine-tuning**
The model is trained via repeated gradient updates using a large corpus of example tasks.

- See otter => loutre de mer
- Peppermint => menthe poivrée
- Plush giraffe => girafe peluche
- Cheese =>

*Adapted from GPT3*

---

**Figure 1.1:** Language model meta-learning. During unsupervised pre-training, a language model develops a broad set of skills and pattern recognition abilities. It then uses these abilities at inference time to rapidly adapt to or recognize the desired task. We use the term “in-context learning” to describe the inner loop of this process, which occurs within the inner loop.

**Figure 2.1:** Zero-shot, one-shot and few-shot, contrasted with traditional fine-tuning. The panels above show

**Figure 1.2:** Larger models make increasingly efficient use of in-context information. We show in-context learning
Architecture evolution: GPT3 ← GPT2+mods ← GPT+mods

Core architecture is classic ‘language modeling’:

\[ p(x) = \prod_{i=1}^{n} p(s_n | s_1, \ldots, s_{n-1}) \]

Learning to perform a task as estimating distribution \( P(output \mid input) \)

Original GPT\textsuperscript{1} trains a standard LM objective to maximize the likelihood:

\[
L_1(U) = \sum_i \log P(u_i \mid u_{i-k}, \ldots, u_{i-1}; \Theta)
\]

- Given an unsupervised corpus of tokens \( \mu = \{\mu_1, \ldots, \mu_n\} \), where \( k \) is context window, \( P \) is modelled using a neural network with parameters \( \Theta \)

- GPT uses a multi-layer Transformer decoder for the language model

Aside: GPT Architecture – Transformer

- Also used in (w/ caveats) in current SOTA language modeling and NLP architectures like BERT and BERT-variants (RoBERTa, XLNet, Transformer XL etc.)

- GPT vs. BERT-variants:
  - GPT uses ‘transformer’ blocks as decoders, and BERT as encoders.
  - Underlying (block level) ideology is same
  - GPT (later Transformer XL, XLNet) is an **autoregressive** model, BERT is not
    - At the cost of auto-regression, BERT has bi-directional context awareness
  - GPT, like traditional LMs, outputs one token at a time

Research in neural networks is exciting, expansive, and explorative.

We have many hyperparameters we can tweak (e.g., activation functions, number and size of layers).

We have many architectures we can use (e.g., deep networks, LSTMs, attention mechanisms).

Given the fevered hype, it’s important to retain our scientific skepticism.

- What are our biases and expectations?
- When are neural networks the wrong choice?
- How are we actually evaluating these systems?