Deep Learning for NLP
Paul Michel
The Plan

1. Introduction
2. Building Blocks of Deep Learning for NLP
3. Training
4. Some Classical Results
5. Deep Learning for NLP Today
Deep Learning:
Introduction
The “Classical” NLP Pipeline

Input x

This movie was a lot of fun!
The “Classical” NLP Pipeline

This movie was a lot of fun!

Feature extraction

Features $f(x)$

- Contains word “bad”
- Contains word “fun”
- Contains bigram “very boring”
- Ends with “!”
The “Classical” NLP Pipeline

Input $x$

This movie was a lot of fun!

Feature extraction

Features $f(x)$

Contains word “bad”
Contains word “fun”
Contains bigram “very boring”
Ends with “!”

Weights $w$

-2.0
3.5
-6.7
...

0.3
The Linear Model

$$\log p(y) \propto w_y^T \cdot f(x)$$
The Linear Model

\[ \log p(y) \propto w_y^T \cdot f(x) \]

Output probability
weights
features
The Linear Model

\[ \log p(y) \propto w_y^T \cdot f(x) \]
The Linear Model

$$\log p(y) \propto w_y^T f(x)$$

Output probability

Weights learned

Features hand-crafted
The Linear Model

\[
\log p(y) \propto w^T y f(x)
\]

- Easy to train (convex optimization)
- Interpretable (1 weight = 1 feature)
The Linear Model

\[ \log p(y) \propto w^T y \]

Output probability

- Easy to train (convex optimization)
- Interpretable (1 weight = 1 feature)

Weights

- Expressivity
- Feature engineering

Learned

Hand-crafted

\[ f(x) \]
The Linear Model: Expressivity

- $h^*$: function we are trying to approximate
The Linear Model: Expressivity

- \( h^* \): function we are trying to approximate

- \( h_\theta \): function (model) parametrized by \( \theta \)
The Linear Model: Expressivity

- $h^*$: function we are trying to approximate

- $h_\theta$: function (model) parametrized by $\theta$

- We want there to be a $\theta^*$ such that
  - $h^* = h_{\theta^*}$
  - Or at least $|h^* - h_{\theta^*}|$ is small
The Linear Model: Expressivity

○ $h^*$: function we are trying to approximate

○ $h_\theta$: function (model) parametrized by $\theta$

○ We want there to be a $\theta^*$ such that
  > $h^* = h_{\theta^*}$
  > Or at least $|h^* - h_{\theta^*}|$ is small
The Linear Model

\[ \log p(y) \propto \mathbf{w}_y^T \cdot f(x) \]
Deep Learning

\[ \log p_{\theta}(y) \propto w_y^T \cdot f_{\theta}(x) \]

- Output probability
- Weights
- Features
Deep Learning

$$\log p_\theta(y)$$

Output probability

$$w_y^T \cdot f_\theta(x)$$

Weights

Features

Learned

Also learned!
Deep Learning

\[ \log p_{\theta}(y) \]

Output probability

- More expressive
- Less feature engineering
Deep Learning

\[ \log p_{\theta}(y) \]

Output probability

More expressive
Less feature engineering

- Harder to train
- Difficult to interpret what the weights mean
Deep Learning: Expressivity
Deep Learning: Expressivity
Deep Learning: Building Blocks
Model Architecture

○ In theory $f_\theta$ could be anything expressive enough
  > Polynomial? $f_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_n x^n$
  > Sinusoidal? $f_\theta(x) = \theta_0 + \theta_{1,1} \cos(x) + \theta_{1,2} \sin(x) + \ldots + \theta_{n,1} \cos(nx) + \theta_{n,2} \sin(nx)$
  > ...
Model Architecture

○ In theory $f_\theta$ could be anything expressive enough
  > Polynomial? $f_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_n x^n$
  > Sinusoidal? $f_\theta(x) = \theta_0 + \theta_{1,1} \cos(x) + \theta_{1,2} \sin(x) + \ldots + \theta_{n,1} \cos(nx) + \theta_{n,2} \sin(nx)$
  > ...

○ In practice: “stacked layers”
  > Combine simple functions (“layers”) together:
  > $f_\theta(x) = h^n_{\theta_n}(\ldots h^2_{\theta_2}(h^1_{\theta_1}(x))\ldots)$
  > “Multilayer perceptron”
Model Architecture

○ In theory $f_\theta$ could be anything expressive enough
  > Polynomial? $f_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_n x^n$
  > Sinusoidal? $f_\theta(x) = \theta_0 + \theta_{1,1} \cos(x) + \theta_{1,2} \sin(x) + \ldots + \theta_{n,1} \cos(nx) + \theta_{n,2} \sin(nx)$
  > ...

○ In practice: “stacked layers”
  > Combine simple functions (“layers”) together:
    $f_\theta(x) = h^n_{\theta_n}( \ldots h^2_{\theta_2}( h^1_{\theta_1}(x) ) \ldots )$
  > “Multilayer perceptron”
Model Architecture: Stacked Layers
Model Architecture: Stacked Layers
Model Architecture: Stacked Layers

\[ f_\theta(x) = h^n_\theta_n( ... h^2_\theta_2( h^1_\theta_1( x ) ) ... ) \]
Model Architecture: Stacked Layers

\[ f_\theta(x) = h^n_{\theta_n}( ... h^2_{\theta_2}( h^1_{\theta_1}( x ) ) ... ) \]

○ Expressivity:
  > Universal approximation theorem:

For well chosen \( h \), any function \( f \) can be approximated with arbitrary precision by a multilayer perceptron.
Model Architecture: Stacked Layers

\[ f_\theta(x) = h^{n}_{\theta_n} \left( ... h^{2}_{\theta_2} \left( h^{1}_{\theta_1}(x) \right) ... \right) \]

- **Expressivity:**
  - Universal approximation theorem:
    - For well chosen \( h \), any function \( f \) can be approximated with arbitrary precision by a multilayer perceptron
  - NOTE: this doesn’t mean that this “best approximating perceptron” is easy to find...
Model Architecture: Stacked Layers

\[ f_\theta(x) = h^n_{\theta_n}( \ldots h^2_{\theta_2}( h^1_{\theta_1}( x ) ) \ldots ) \]

○ Expressivity:
  > Universal approximation theorem:
    
    For well chosen \( h \), any function \( f \) can be approximated with arbitrary precision by a multilayer perceptron
  
  > NOTE: this doesn’t mean that this “best approximating perceptron” is easy to find…

○ Learning
  > Computing the gradients is inexpensive with the chain rule!
    (=as easy as computing \( f_\theta(x) \))
  > In practice “easy” to train
What is h?

1. Dense layers
   > The workhorse of deep learning

2. Embeddings
   > How to go from categorical variables to vectors

3. Recurrent cells
   > How to tackle sequences

4. Attention
   > How to combine multiple outputs into one

5. Many others I won’t cover...
The Dense Layer

○ Input vector: $\mathbf{x} = [x_1, \ldots, x_n]$

○ "Neuron" $i$: $a_i = \sigma(w_{i1}x_1 + \ldots + w_{in}x_n + b_i)$
The Dense Layer

- Input vector: \( \mathbf{x} = [x_1, \ldots, x_n] \)
- “Neuron” i: \( a_i = \sigma(w_{i1}x_1 + \ldots + w_{in}x_n + b_i) \)
- \( m \) neurons in parallel:

\[ \mathbf{a} = \sigma(\mathbf{Wx} + \mathbf{b}) \]
The Dense Layer

- **Input vector:** \( \mathbf{x} = [x_1, \ldots, x_n] \)
- **“Neuron”** \( i \): \( a_i = \sigma(w_{i1}x_1 + \ldots + w_{in}x_n + b_i) \)
- **m neurons in parallel:**
  \[
  \mathbf{a} = \sigma(\mathbf{Wx} + \mathbf{b})
  \]
  ![Diagram of the dense layer with parameters](image)

- **With parameters:**
  - **\( \mathbf{W} \):** a \( m \times n \) weight matrix
  - **\( \mathbf{b} \):** a \( m \)-dimensional bias vector
  - **\( \sigma \):** an activation function
The Dense Layer

- Input vector: $\mathbf{x} = [x_1, \ldots, x_n]$
- “Neuron” $i$: $a_i = \sigma(w_{i1}x_1 + \ldots + w_{in}x_n + b_i)$
- $m$ neurons in parallel:
  $$\mathbf{a} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$
- With parameters:
  - $\mathbf{W}$: a $m \times n$ weight matrix
  - $\mathbf{b}$: a $m$-dimensional bias vector
  - $\sigma$: an activation function

$$\mathbf{a} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$
The Dense Layer

- Input vector: \( \mathbf{x} = [x_1, \ldots, x_n] \)
- “Neuron” \( i \): \( a_i = \sigma(w_{i1}x_1 + \ldots + w_{in}x_n + b_i) \)
- \( m \) neurons in parallel:
  \[ \mathbf{a} = \sigma(\mathbf{Wx} + \mathbf{b}) \]

- With parameters:
  - \( \mathbf{W} \): an \( m \times n \) weight matrix
  - \( \mathbf{b} \): a \( m \)-dimensional bias vector
  - \( \sigma \): an activation function

- Sufficient for the universal approximation theorem!
Embeddings

- In NLP our inputs are words, not vectors
Embeddings

- In NLP our inputs are words, not vectors
- How do we turn words into vectors?
Embeddings

- In NLP our inputs are words, not vectors
- How do we turn words into vectors?
- First attempt: **features**

Gradient descent rocks!

Feature extraction

- Contains character “s”
- Contains character trigram “ent”
- Starts with a lowercase letter
- Is mathematical term
Embeddings

- In NLP our inputs are words, not vectors
- How do we turn words into vectors?
- First attempt: **features**

- Problems:
  - Feature engineering
  - Need many features to be effective

Gradient descent rocks!

Feature extraction

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contains character “s”</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Contains character trigram “ent”</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Starts with a lowercase letter</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Is mathematical term</td>
<td></td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
Embeddings

- In NLP our inputs are words, not vectors
- How do we turn words into vectors?
- Second attempt: **learned word embeddings**
  - Fix a dimension \( d \)
  - Assign a \( d \)-dimensional vector to each word
  - Initialize at random and learn like any other parameters

Gradient descent rocks!

```
0.3  0.2  0.2
-1.2 -0.5 -0.5
-0.4 -1.0 -1.0
...  ...  ...
```
Embeddings

- In NLP our inputs are words, not vectors
- How do we turn words into vectors?
- Second attempt: **learned word embeddings**
  - Fix a dimension $d$
  - Assign a $d$-dimensional vector to each word
  - Initialize at random and learn like any other parameters

- Problems
  - Sensitive to initialization
  - Not interpretable
  - Sparse gradients for low-frequency words

Gradient descent rocks!
Processing Sequences

Gradient descent rocks!
Processing Sequences

Gradient descent rocks!

0.3
-1.2
-0.4
...

0.2
-0.5
-1.0
...

0.2
-0.5
-1.0
...
Processing Sequences

Gradient descent rocks!

```
0.3  0.2  0.2
-1.2 -0.5 -0.5
-0.4 -1.0 -1.0
...  ...  ...
```
Processing Sequences

Gradient descent rocks!

\[ f_\theta \]
Processing Sequences

Gradient descent rocks!

\[ f_\theta(x) \]
Processing Sequences

- Gradient descent rocks!

○ How do we turn a sequence of vectors into a vector for classification?
  > How do we capture the relationship between words?
Recurrent Layers

- Idea: like an automaton
  - $h_i = F(x_i, h_{i-1})$
  - The last $h$ is a function of the whole sentence
Recurrent Layers

○ Idea: like an automaton
  > $h_i = \text{RNN}(x_i, h_{i-1})$
  > The last $h$ is a function of the whole sentence

Gradient descent rocks!

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>-1.2</td>
<td>-0.5</td>
<td>-0.5</td>
</tr>
<tr>
<td>-0.4</td>
<td>-1.0</td>
<td>-1.0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
```
Recurrent Layers

- Idea: like an automaton
  - $h_i = \text{RNN}(x_i, h_{i-1})$
  - The last $h$ is a function of the whole sentence

Gradient descent rocks!

- $h_0$
- $0.3$
- $-1.2$
- $-0.4$
- $...$
- $0.2$
- $-0.5$
- $-1.0$
- $...$
- $0.2$
- $-0.5$
- $-1.0$
- $...$
Recurrent Layers

- Idea: like an automaton
  - \( h_i = \text{RNN}(x_i, h_{i-1}) \)
  - The last \( h \) is a function of the whole sentence

Gradient descent rocks!
Recurrent Layers

- Idea: like an automaton
  - $h_i = \text{RNN}(x_i, h_{i-1})$
  - The last $h$ is a function of the whole sentence

Gradient descent rocks!
Recurrent Layers

- Idea: like an automaton
  - $h_i = \text{RNN}(x_i, h_{i-1})$
  - The last $h$ is a function of the whole sentence

Gradient descent rocks!

```
0.3  -1.2  -0.4  ...
0.2  -0.5  -1.0  ...
0.2  -0.5  -1.0  ...
```
Recurrent Layers

- **Idea**: like an automaton
  - $h_i = \text{RNN}(x_i, h_{i-1})$
  - The last $h$ is a function of the whole sentence

Gradient descent rocks!

$$
\begin{array}{c}
0.3 \\
-1.2 \\
-0.4 \\
... \\
\end{array} \\
\begin{array}{c}
0.2 \\
-0.5 \\
-1.0 \\
... \\
\end{array} \\
\begin{array}{c}
0.2 \\
-0.5 \\
-1.0 \\
... \\
\end{array}
$$
Recurrent Layers

- Idea: like an automaton
  - $h_i = \text{RNN}(x_i, h_{i-1})$
  - The last $h$ is a function of the whole sentence

- What is the “RNN” function?
Recurrent Layers

- Idea: like an automaton
  - \( h_i = \text{RNN}(x_i, h_{i-1}) \)
  - The last \( h \) is a function of the whole sentence

- What is the “RNN” function?

\[
\begin{align*}
  h_i & = \sigma \\
  \sigma & = W_x x_i + b
\end{align*}
\]
Recurrent Layers

- **Idea:** like an automaton
  - \( h_i = \text{RNN}(x_i, h_{i-1}) \)
  - The last \( h \) is a function of the whole sentence

- **What is the “RNN” function?**
Recurrent Layers

- **Idea: like an automaton**
  - \( h_i = \text{RNN}(x_i, h_{i-1}) \)
  - The last \( h \) is a function of the whole sentence

- **In practice this is not used anymore**
  - Too hard to train (exploding/vanishing gradients)
Recurrent Layers

- **Idea: like an automaton**
  - $h_i = \text{RNN}(x_i, h_{i-1})$
  - The last $h$ is a function of the whole sentence

- **In practice this is not used anymore**
  - Too hard to train (exploding/vanishing gradients)

- **State of the art: LSTM**
  - Additive update of the hidden state
  - Better gradient flow (no exponential decay)
  - Ubiquitous in NLP up until ca. 2018

Figure credits: Christopher Olah http://colah.github.io/posts/2015-08-Understanding-LSTMs/
Bidirectional LSTM

Cornegruta et al. (2016)
Attention Layers

- How do we turn a sequence of vectors into a vector for classification?
  - Why not just sum/average the word vectors?
  - We treat all words equally: bad
Attention Layers

○ How do we turn a sequence of vectors into a vector for classification?
  > Why not just sum/average the word vectors?
  > We treat all words equally: bad

○ Attention: weighted sum of vectors
Attention Layers

- How do we turn a sequence of vectors into a vector for classification?
  - Why not just sum/average the word vectors?
  - We treat all words equally: bad

- Attention: weighted sum of vectors

```
0.3  0.2  0.2
-1.2 -0.5 -0.5
-0.4 -1.0 -1.0
...
...
```
Attention Layers

- How do we turn a sequence of vectors into a vector for classification?
  - Why not just sum/average the word vectors?
  - We treat all words equally: bad

- Attention: weighted sum of vectors

```
x_1
0.3
-1.2
-0.4
...

x_2
0.2
-0.5
-1.0
...

x_3
0.2
-0.5
-1.0
...
```

Gradient descent rocks!
Attention Layers

○ How do we turn a sequence of vectors into a vector for classification?
  > Why not just sum/average the word vectors?
  > We treat all words equally: bad

○ Attention: weighted sum of vectors

\[
\alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3
\]

Gradient descent rocks!

\[
\begin{align*}
0.3 & \quad -1.2 & \quad -0.4 \\
-1.2 & \quad 0.2 & \quad 0.2 \\
-0.4 & \quad -0.5 & \quad -0.5 \\
-1.0 & \quad -1.0 & \quad -1.0 \\
\ldots & \quad \ldots & \quad \ldots \\
\end{align*}
\]
Attention Layers

- How do we turn a sequence of vectors into a vector for classification?
  - Why not just sum/average the word vectors?
  - We treat all words equally: bad

- Attention: weighted sum of vectors

\[ \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 \]

Gradient descent rocks!
Attention Layers

○ How do we turn a sequence of vectors into a vector for classification?
  > Why not just sum/average the word vectors?
  > We treat all words equally: bad

○ Attention: weighted sum of vectors

\[ f_\theta(x) = \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 = f_\theta(x) \]
Attention Layers Example: Machine Translation

- **Key Vectors**
- **Query Vector**
- **Example:**
  - Input: `hate`
  - Output: `kono eiga ga kirai`
  - Attention weights: $a_1 = 2.1$, $a_2 = -0.1$, $a_3 = 0.3$, $a_4 = -1.0$
  - Softmax output: $\alpha_1 = 0.76$, $\alpha_2 = 0.08$, $\alpha_3 = 0.13$, $\alpha_4 = 0.03$
Attention Layers Example: Machine Translation

\[ \alpha_1 = 0.76 \quad \alpha_2 = 0.08 \quad \alpha_3 = 0.13 \quad \alpha_4 = 0.03 \]
Attention Layers Example: Machine Translation

良いレストランを紹介していただけますか。
could you recommend an inexpensive restaurant?

CS 11-747
Transformer

- Fairly recent (2017)
  > Originally proposed for machine translation
  > State of the art in many NLP tasks

Vaswani et al. (2017)
Transformer

- Fairly recent (2017)
  - Originally proposed for machine translation
  - State of the art in many NLP tasks
- Main characteristics
  - Uses mostly attention/dense layers
  - No recurrent layers at all!
  - “Attention is all you need”

Vaswani et al. (2017)
Transformer

○ Fairly recent (2017)
  > Originally proposed for machine translation
  > State of the art in many NLP tasks

○ Main characteristics
  > Uses mostly attention/dense layers
  > No recurrent layers at all!
  > “Attention is all you need”

Vaswani et al. (2017)
Transformer

○ Fairly recent (2017)
  > Originally proposed for machine translation
  > State of the art in many NLP tasks

○ Main characteristics
  > Uses mostly attention/dense layers
  > No recurrent layers at all!
  > “Attention is all you need”

○ Pros
  > Easy to parallelize on the GPU (large matrix multiplications)
  > “Equal path length” between any two words

Vaswani et al. (2017)
Transformer

○ Fairly recent (2017)
  > Originally proposed for machine translation
  > State of the art in many NLP tasks
○ Main characteristics
  > Uses mostly attention/dense layers
  > No recurrent layers at all!
  > “Attention is all you need”
○ Pros
  > Easy to parallelize on the GPU (large matrix multiplications)
  > “Equal path length” between any two words
○ Cons
  > Still not completely figured out
  > Seemingly harder to train in a low resource scenario

Vaswani et al. (2017)
Things I didn’t cover

○ Convolutional layers
  > Mainly for vision and speech but also used in text processing

○ Residual connections
  > For gradient flow

○ Normalization layers
  > For stability of deep networks

○ Stochastic layers
  > Variational inference, etc...

○ Many others...
Deep Learning: Training
Deep Learning: Training

- Loss function (here likelihood):
  \[ L(\theta) = \sum_{x,y \text{ in data}} \log p(y|x;\theta) \]
  > Note: \( L \) is a function of \( \theta \)
Deep Learning: Training

○ Loss function (here likelihood):
  > Note: L is a function of θ

\[ L(\theta) = \sum_{x,y \text{ in data}} \log p(y|x;\theta) \]

○ Training:

\[ \theta^* = \arg\min_{\theta} L(\theta) \]
Deep Learning: Training

○ Loss function (here likelihood): 
  \[ L(\theta) = \sum_{x,y \text{ in data}} \log p(y|x;\theta) \]
  > Note: \( L \) is a function of \( \theta \)

○ Training: 
  \[ \theta^* = \arg\min_{\theta} L(\theta) \]

○ Gradient descent:
  > Take small steps following the direction of steepest descent of \( L \)
  > \( \theta_0 \) random
  > \( \theta_{t+1} = \theta_t - \lambda \nabla L(\theta_t) \) with small \( \lambda \)
Gradient Descent

\[ \theta_0 \] \quad \theta^*
Gradient Descent
Gradient Descent
Gradient descent

- **Gradient descent:**
  - Take small steps following the direction of steepest descent of $L$
  - $\theta_0$ random
  - $\theta_{t+1} = \theta_t - \lambda \nabla L(\theta_t)$ with small $\lambda$
Gradient descent

- Gradient descent:
  - Take small steps following the direction of steepest descent of $L$
  - $\theta_0$ random
  - $\theta_{t+1} = \theta_t - \lambda \nabla L(\theta_t)$ with small $\lambda$

- How is $\nabla L(\theta_t)$ computed?
Gradient descent

- **Gradient descent:**
  - Take small steps following the direction of steepest descent of $L$
  - $\theta_0$ random
  - $\theta_{t+1} = \theta_t - \lambda \nabla L(\theta_t)$ with small $\lambda$

- How is $\nabla L(\theta_t)$ computed? Backpropagation
  - $y = h_2(h_1(x))$
  - $\frac{dy}{dx} = [\frac{dh_2}{dh_1}] \cdot [\frac{dh_1}{dx}]$
Gradient descent

- **Gradient descent:**
  - Take small steps following the direction of steepest descent of $L$
  - $\theta_0$ random
  - $\theta_{t+1} = \theta_t - \lambda \nabla L(\theta_t)$ with small $\lambda$

- **How is $\nabla L(\theta_t)$ computed?** Backpropagation
  - $y = h_2(h_1(x))$
  - $dy/dx = [dh_2/dh_1] . [dh_1/dx]$
Gradient descent

- **Gradient descent:**
  - Take small steps following the direction of steepest descent of $L$
  - $\theta_0$ random
  - $\theta_{t+1} = \theta_t - \lambda \nabla L(\theta_t)$ with small $\lambda$

- How is $\nabla L(\theta_t)$ computed? Backpropagation

  - $y = h_2(h_1(x))$
  - $\frac{dy}{dx} = \left[\frac{dh_2}{dh_1}\right] \cdot \left[\frac{dh_1}{dx}\right]$
Difficulties of Training Neural Networks

- Non-convexity
  - No guarantee that gradient descent converges to a global minimum
Gradient Descent: Convex Function

\[ \theta^* - \lambda \nabla L(\theta_0) - \lambda \nabla L(\theta_1) \]
Gradient Descent: Non convexity
Difficulties of Training Neural Networks

- Non-convexity
  - No guarantee that gradient descent converges to a global minimum

- Gradient computation
  - Requires forward+backward pass for all examples: lot of memory
  - Minibatch training compute the loss and gradient on a subset of the data at each step
  - “Stochastic” Gradient Descent (SGD)
Stochastic Gradient Descent
Stochastic Gradient Descent

Stochastic Gradient Descent

Gradient Descent
Stochastic Gradient Descent
Stochastic Gradient Descent: Variants

- Many variants have been developed to address the shortcomings of SGD
Many variants have been developed to address the shortcomings of SGD
Difficulties of Training Neural Networks

- **Non-convexity**
  - No guarantee that gradient descent converges to a global minimum

- **Gradient computation**
  - Requires forward+backward pass for all examples: lot of memory
  - Minibatch training compute the loss and gradient on a subset of the data at each step
  - “Stochastic”/Minibatch Gradient Descent (SGD)

- **Overfitting**
  - Deep learning models have a LOT of parameters (sometimes more than the amount of data)
  - Training error goes down but test error goes up!
Overfitting
Regularization

- Combat overfitting with regularization
Regularization

○ Combat overfitting with regularization

○ L2 regularization
  > Add a penalty to prevent parameters from being too big
  > Well motivated for linear models
Regularization

○ Combat overfitting with regularization

○ L2 regularization
  > Add a penalty to prevent parameters from being too big
  > Well motivated for linear models

○ Dropout
  > Randomly drop (=set to zero) entire neurons (rows in the weight matrices)
  > Very strong regularizer!
Regularization

○ Combat overfitting with regularization

○ L2 regularization
  > Add a penalty to prevent parameters from being too big
  > Well motivated for linear models

○ Dropout
  > Randomly drop (=set to zero) entire neurons (rows in the weight matrices)
  > Very strong regularizer!

○ Implicit regularization?
  > Recent work suggest that minibatch-training and architecture may implicitly regularize
  > A lot of ongoing research to understand why neural networks generalize so surprisingly well.
    Can’t be explained by standard learning theory
Deep Learning:
Classical Major Results
2012: ImageNet (Image Classification)

- AlexNet (Krizhevsky et al.)
  - Wins the ImageNet competition with a neural network
  - 15.3% top-5 error rate
  - 10.8% points better than the runner up!

Krizhevsky et al., 2012
2012: ImageNet (Image Classification)

- AlexNet (Krizhevsky et al.)
  - Wins the ImageNet competition with a neural network
  - 15.3% top-5 error rate
  - 10.8% points better than the runner up!

- Arguably started the golden age of deep learning
2011: Language Modeling

Table 2: Comparison of various configurations of RNN LMs and combinations with backoff models while using 6.4M words in training data (WSJ DEV).

<table>
<thead>
<tr>
<th>Model</th>
<th>PPL</th>
<th>WER</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PPL</td>
<td>WER</td>
</tr>
<tr>
<td></td>
<td>RNN</td>
<td>RNN+KN</td>
</tr>
<tr>
<td>KN5 - baseline</td>
<td>-</td>
<td>221</td>
</tr>
<tr>
<td>RNN 60/20</td>
<td>229</td>
<td>186</td>
</tr>
<tr>
<td>RNN 90/10</td>
<td>202</td>
<td>173</td>
</tr>
<tr>
<td>RNN 250/5</td>
<td>173</td>
<td>155</td>
</tr>
<tr>
<td>RNN 250/2</td>
<td>176</td>
<td>156</td>
</tr>
<tr>
<td>RNN 400/10</td>
<td>171</td>
<td>152</td>
</tr>
<tr>
<td>3xRNN static</td>
<td>151</td>
<td>143</td>
</tr>
<tr>
<td>3xRNN dynamic</td>
<td>128</td>
<td>121</td>
</tr>
</tbody>
</table>

Mikolov et al., 2011
2014: Machine Translation

Table 1: The performance of the LSTM on WMT’14 English to French test set (ntst14). Note that an ensemble of 5 LSTMs with a beam of size 2 is cheaper than of a single LSTM with a beam of size 12.
2014: Dependency Parsing

Figure 2: Our neural network architecture.

Table 5: Accuracy and parsing speed on PTB + Stanford dependencies.

<table>
<thead>
<tr>
<th>Parser</th>
<th>Dev UAS</th>
<th>LAS</th>
<th>Test UAS</th>
<th>LAS</th>
<th>Speed (sent/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>standard eager</td>
<td>90.2</td>
<td>87.8</td>
<td>89.4</td>
<td>87.3</td>
<td>26</td>
</tr>
<tr>
<td>Maltsp</td>
<td>89.8</td>
<td>87.2</td>
<td>89.3</td>
<td>86.9</td>
<td>469</td>
</tr>
<tr>
<td>Malteager</td>
<td>89.6</td>
<td>86.9</td>
<td>89.4</td>
<td>86.8</td>
<td>448</td>
</tr>
<tr>
<td>MSTParser</td>
<td>91.4</td>
<td>88.1</td>
<td>90.7</td>
<td>87.6</td>
<td>10</td>
</tr>
<tr>
<td>Our parser</td>
<td><strong>92.0</strong></td>
<td><strong>89.7</strong></td>
<td><strong>91.8</strong></td>
<td><strong>89.6</strong></td>
<td><strong>654</strong></td>
</tr>
</tbody>
</table>

Table 6: Accuracy and parsing speed on CTB.

<table>
<thead>
<tr>
<th>Parser</th>
<th>Dev UAS</th>
<th>LAS</th>
<th>Test UAS</th>
<th>LAS</th>
<th>Speed (sent/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>standard eager</td>
<td>82.4</td>
<td>80.9</td>
<td>82.7</td>
<td>81.2</td>
<td>72</td>
</tr>
<tr>
<td>Maltsp</td>
<td>82.4</td>
<td>80.5</td>
<td>82.4</td>
<td>80.6</td>
<td>420</td>
</tr>
<tr>
<td>Malteager</td>
<td>81.2</td>
<td>79.3</td>
<td>80.2</td>
<td>78.4</td>
<td>393</td>
</tr>
<tr>
<td>MSTParser</td>
<td><strong>84.0</strong></td>
<td><strong>82.1</strong></td>
<td><strong>83.0</strong></td>
<td><strong>81.2</strong></td>
<td><strong>6</strong></td>
</tr>
<tr>
<td>Our parser</td>
<td><strong>84.0</strong></td>
<td><strong>82.4</strong></td>
<td><strong>83.9</strong></td>
<td><strong>82.4</strong></td>
<td><strong>936</strong></td>
</tr>
</tbody>
</table>

Chen & Manning (2014)
Deep Learning: Today
Downsides of Deep Learning

○ Need a lot of data
  > Otherwise: overfitting, etc...
  > Problem for low resource languages
  > Or tasks where annotation is expensive

○ Compute intensive
  > Training can take many GPU hours
  > Large number of parameters

○ Hard to interpret
  > Hard to explain why the model is making a prediction
  > Not clear what the model is learning

○ Robustness
  > Can be brittle in the face of (well chosen) noise
  > Doesn’t work as well on different domains
The “Pretrain & Finetune” Paradigm
The “Pretrain & Finetune” Paradigm

Standard NLP training pipeline
The “Pretrain & Finetune” Paradigm

Standard NLP training pipeline

1. Collect/Annotate data for your task
   > Can be expensive/slow
   > More data is always better
The “Pretrain & Finetune” Paradigm

Standard NLP training pipeline

1. Collect/Annotate data for your task
   - Can be expensive/slow
   - More data is always better

2. **Train** the model on your dataset
   - Gradient descent, etc...
The “Pretrain & Finetune” Paradigm

Standard NLP training pipeline

1. Collect/Annotate data for your task
   > Can be expensive/slow
   > More data is always better

2. **Train** the model on your dataset
   > Gradient descent, etc...

3. Test and deploy!
The “Pretrain & Finetune” Paradigm

Standard NLP training pipeline

1. Collect/Annotate data for your task
   - Can be expensive/slow
   - More data is always better

2. **Train** the model on your dataset
   - Gradient descent, etc...

3. Test and deploy!

4. Need to perform a different task?
   - Start over
The “Pretrain & Finetune” Paradigm
The “Pretrain & Finetune” Paradigm

Pretrain & Finetune

1. Collect raw text
   > Crawled at a minimal cost from the internet (wikipedia, news,...)
   > A LOT of data
The “Pretrain & Finetune” Paradigm

Pretrain & Finetune

1. Collect raw text
   - Crawled at a minimal cost from the internet (wikipedia, news,...)
   - A LOT of data

2. **Pre-train** your model on the unannotated corpus
   - Using an objective that doesn’t require labels (language modeling, etc...)
The “Pretrain & Finetune” Paradigm

Pretrain & Finetune

1. Collect raw text
   > Crawled at a minimal cost from the internet (wikipedia, news,...)
   > A LOT of data

2. **Pre-train** your model on the unannotated corpus
   > Using an objective that doesn’t require labels (language modeling, etc...)

3. **Fine-tuning**: proceed with the standard NLP pipeline using the pre-trained model as a starting point
The “Pretrain & Finetune” Paradigm

Pretrain & Finetune

1. Collect raw text
   > Crawled at a minimal cost from the internet (wikipedia, news, ...)
   > A LOT of data

2. **Pre-train** your model on the unannotated corpus
   > Using an objective that doesn’t require labels (language modeling, etc...)

3. **Fine-tuning**: proceed with the standard NLP pipeline using the pre-trained model as a starting point

4. Need to perform a different task?
   > Start over from the pre-trained model
The “Pretrain & Finetune” Paradigm

- Just the standard pipeline with extra steps?
The “Pretrain & Finetune” Paradigm

○ Just the standard pipeline with extra steps?
○ NO: using a pre-trained model makes it possible to use much less data during fine tuning
The “Pretrain & Finetune” Paradigm

- Just the standard pipeline with extra steps?
- NO: using a pre-trained model makes it possible to use much less data during fine tuning
The “Pretrain & Finetune” Paradigm

○ Just the standard pipeline with extra steps?
○ NO: using a pre-trained model makes it possible to use much less data during fine tuning
The “Pretrain & Finetune” Paradigm

- Just the standard pipeline with extra steps?
- NO: using a pre-trained model makes it possible to use much less data during fine tuning
The “Pretrain & Finetune” Paradigm

- Just the standard pipeline with extra steps?
- NO: using a pre-trained model makes it possible to use much less data during fine tuning
The “Pretrain & Finetune” Paradigm

- Just the standard pipeline with extra steps?
- NO: using a pre-trained model makes it possible to use much less data during fine tuning
The “Pretrain & Finetune” Paradigm

- Just the standard pipeline with extra steps?
- NO: using a pre-trained model makes it possible to use much less data during fine tuning
The “Pretrain & Finetune” Paradigm

- Just the standard pipeline with extra steps?
- NO: using a pre-trained model makes it possible to use much less data during fine tuning
The “Pretrain & Finetune” Paradigm

<table>
<thead>
<tr>
<th>System</th>
<th>MNLI-(m/mm)</th>
<th>QQP</th>
<th>QNLI</th>
<th>SST-2</th>
<th>CoLA</th>
<th>STS-B</th>
<th>MRPC</th>
<th>RTE</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>392k</td>
<td>363k</td>
<td>108k</td>
<td>67k</td>
<td>8.5k</td>
<td>5.7k</td>
<td>3.5k</td>
<td>2.5k</td>
<td></td>
</tr>
<tr>
<td>Pre-OpenAI SOTA</td>
<td>80.6/80.1</td>
<td>66.1</td>
<td>82.3</td>
<td>93.2</td>
<td>35.0</td>
<td>81.0</td>
<td>86.0</td>
<td>61.7</td>
<td>74.0</td>
</tr>
<tr>
<td>BiLSTM+ELMo+Attn</td>
<td>76.4/76.1</td>
<td>64.8</td>
<td>79.8</td>
<td>90.4</td>
<td>36.0</td>
<td>73.3</td>
<td>84.9</td>
<td>56.8</td>
<td>71.0</td>
</tr>
<tr>
<td>OpenAI GPT</td>
<td>82.1/81.4</td>
<td>70.3</td>
<td>87.4</td>
<td>91.3</td>
<td>45.4</td>
<td>80.0</td>
<td>82.3</td>
<td>56.0</td>
<td>75.1</td>
</tr>
<tr>
<td>BERT&lt;sub&gt;BASE&lt;/sub&gt;</td>
<td>84.6/83.4</td>
<td>71.2</td>
<td>90.5</td>
<td>93.5</td>
<td>52.1</td>
<td>85.8</td>
<td>88.9</td>
<td>66.4</td>
<td>79.6</td>
</tr>
<tr>
<td>BERT&lt;sub&gt;LARGE&lt;/sub&gt;</td>
<td>&lt;strong&gt;86.7/85.9&lt;/strong&gt;</td>
<td>&lt;strong&gt;72.1&lt;/strong&gt;</td>
<td>&lt;strong&gt;92.7&lt;/strong&gt;</td>
<td>&lt;strong&gt;94.9&lt;/strong&gt;</td>
<td>&lt;strong&gt;60.5&lt;/strong&gt;</td>
<td>&lt;strong&gt;86.5&lt;/strong&gt;</td>
<td>&lt;strong&gt;89.3&lt;/strong&gt;</td>
<td>&lt;strong&gt;70.1&lt;/strong&gt;</td>
<td>&lt;strong&gt;82.1&lt;/strong&gt;</td>
</tr>
</tbody>
</table>

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The “Average” column is slightly different than the official GLUE score, since we exclude the problematic WNLI set. BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.
Pre-training: Architectures

Gradient descent rocks!

Pretrained model

-0.4

...
Pre-training: Architectures

Gradient descent rocks!

Pretrained model

Pre-training specific
Pre-training: Architectures

Gradient descent rocks!

Pretrained model

Pre-training specific
Pre-training: Architectures

Pretrained model

Gradient descent rocks!

Pre-training specific

Fine-tuning

Pretrained model

Gradient descent rocks!
Pre-training: Architectures

Gradient descent rocks!

Pretrained model

Pre-training specific

0.3
-1.2
-0.4
...

0.2
-0.5
-1.0
...

0.2
-0.5
-1.0
...

Gradient descent rocks!

Pretrained model

Task specific

Fine-tuning

0.3
-1.2
-0.4
...

0.2
-0.5
-1.0
...

0.2
-0.5
-1.0
...
Pre-training: Architectures

- **Word vectors: word2vec, GLoVE**
  - Just pre-train word embeddings
  - Most of the “hard work” will be done by the task-specific model
  - But still good because training word embeddings is hard (due to sparsity of low-frequency words)
Pre-training: Architectures

- **Word vectors: word2vec, GLoVE**
  - Just pre-train word embeddings
  - Most of the “hard work” will be done by the task-specific model
  - But still good because training word embeddings is hard (due to sparsity of low-frequency words)

- **Many-to-one: skip-thought vectors, BiLSTM-max...**
  - BiLSTM or self-attention network that returns one vector for a whole sentence/string
  - “Bottleneck” the representation
  - Not adapted for structured prediction (NER, POS tagging, parsing...)
Pre-training: Architectures

- **Word vectors: word2vec, GLoVE**
  - Just pre-train word embeddings
  - Most of the “hard work” will be done by the task-specific model
  - But still good because training word embeddings is hard (due to sparsity of low-frequency words)

- **Many-to-one: skip-thought vectors, BiLSTM-max**
  - BiLSTM or self-attention network that returns one vector for a whole sentence/string
  - “Bottleneck” the representation
  - Not adapted for structured prediction (NER, POS tagging, parsing...)

- **Many-to-many (“Contextual word embeddings”): ULMFit, ELMo, BERT**
  - Takes a sequence of words and returns a sequence of vectors where each vector is a function of ALL the words in the sentence
Pre-training: Architectures

Figure 3: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. Among the three, only BERT representations are jointly conditioned on both left and right context in all layers. In addition to the architecture differences, BERT and OpenAI GPT are fine-tuning approaches, while ELMo is a feature-based approach.
Pre-training Objectives

○ What objective to use during pre-training?
  > Shouldn’t require annotation
  > Should transfer well (“the model should learn something about language”)
Pre-training Objectives

○ What objective to use during pre-training?
  > Shouldn’t require annotation
  > Should transfer well (“the model should learn something about language”)

○ Neighboring word prediction
  > Distributional hypothesis: “a word is characterized by the company it keeps” (Firth, 1957)
  > Learn to predict neighboring words
  > Words that occur in similar contexts will have similar embeddings
Pre-training Objectives

○ What objective to use during pre-training?
  > Shouldn’t require annotation
  > Should transfer well (“the model should learn something about language”)

○ Neighboring sentence prediction
  > Similar as before, but for sentences
Pre-training Objectives

○ What objective to use during pre-training?
  > Shouldn’t require annotation
  > Should transfer well (“the model should learn something about language”)

○ Neighboring sentence prediction
  > Similar as before, but for sentences
## Pre-training Objectives

<table>
<thead>
<tr>
<th>Query and nearest sentence</th>
</tr>
</thead>
<tbody>
<tr>
<td>he ran his hand inside his coat, double-checking that the unopened letter was still there.</td>
</tr>
<tr>
<td>he slipped his hand between his coat and his shirt, where the folded copies lay in a brown envelope.</td>
</tr>
<tr>
<td>im sure youll have a glamorous evening, she said, giving an exaggerated wink.</td>
</tr>
<tr>
<td>im really glad you came to the party tonight, he said, turning to her.</td>
</tr>
<tr>
<td>although she could tell he had n’t been too invested in any of their other chitchat, he seemed genuinely curious about this.</td>
</tr>
<tr>
<td>although he had n’t been following her career with a microscope, he ’d definitely taken notice of her appearances.</td>
</tr>
<tr>
<td>an annoying buzz started to ring in my ears, becoming louder and louder as my vision began to swim.</td>
</tr>
<tr>
<td>a weighty pressure landed on my lungs and my vision blurred at the edges, threatening my consciousness altogether.</td>
</tr>
<tr>
<td>if he had a weapon, he could maybe take out their last imp, and then beat up errol and vanessa.</td>
</tr>
<tr>
<td>if he could ram them from behind, send them sailing over the far side of the levee, he had a chance of stopping them.</td>
</tr>
<tr>
<td>then, with a stroke of luck, they saw the pair head together towards the portaloos.</td>
</tr>
<tr>
<td>then, from out back of the house, they heard a horse scream probably in answer to a pair of sharp spurs digging deep into its flanks.</td>
</tr>
<tr>
<td>“i’ll take care of it,” goodman said, taking the phonebook.</td>
</tr>
<tr>
<td>“i’ll do that,” julia said, coming in.</td>
</tr>
<tr>
<td>he finished rolling up scrolls and, placing them to one side, began the more urgent task of finding ale and tankards.</td>
</tr>
<tr>
<td>he righted the table, set the candle on a piece of broken plate, and reached for his flint, steel, and tinder.</td>
</tr>
</tbody>
</table>

Table 2: In each example, the first sentence is a query while the second sentence is its nearest neighbour. Nearest neighbours were scored by cosine similarity from a random sample of 500,000 sentences from our corpus.
Pre-training Objectives

- **What objective to use during pre-training?**
  - Shouldn’t require annotation
  - Should transfer well ("the model should learn something about language")

- **Unidirectional language modeling**
  - Predict next word in a sentence
  - Can be reversed as well
  - Huge literature on language modeling
Pre-training Objectives

○ What objective to use during pre-training?
  > Shouldn’t require annotation
  > Should transfer well ("the model should learn something about language")

○ Unidirectional language modeling
  > Predict next word in a sentence
  > Can be reversed as well
  > Huge literature on language modeling
Pre-training Objectives

○ What objective to use during pre-training?
  > Shouldn’t require annotation
  > Should transfer well (“the model should learn something about language”)

○ Masked Language Modeling
  > Predict missing word in a sentence
  > More flexibility as far as architecture is concerned
Pre-training Objectives

○ What objective to use during pre-training?
  > Shouldn’t require annotation
  > Should transfer well (“the model should learn something about language”)

○ Masked Language Modeling
  > Predict missing word in a sentence
  > More flexibility as far as architecture is concerned
Some Results

○ BERT (Devlin et al., 2018)
  > Transformer trained with masked language modeling + next sentence prediction
  > Fine-tuned on a variety of tasks (sentiment, entailment, QA...)
  > State-of-the-art almost everywhere
Some Results

- **BERT (Devlin et al., 2018)**
  - Transformer trained with masked language modeling + next sentence prediction
  - Fine-tuned on a variety of tasks (sentiment, entailment, QA...)
  - State-of-the-art almost everywhere

<table>
<thead>
<tr>
<th>System</th>
<th>MNLI-(m/mm)</th>
<th>QQP</th>
<th>QNLI</th>
<th>SST-2</th>
<th>CoLA</th>
<th>STS-B</th>
<th>MRPC</th>
<th>RTE</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>392k/363k</td>
<td>108k</td>
<td>67k</td>
<td>8.5k</td>
<td>5.7k</td>
<td>3.5k</td>
<td>2.5k</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Pre-OpenAI SOTA</td>
<td>80.6/80.1</td>
<td>66.1</td>
<td>82.3</td>
<td>93.2</td>
<td>35.0</td>
<td>81.0</td>
<td>86.0</td>
<td>61.7</td>
<td>74.0</td>
</tr>
<tr>
<td>BiLSTM+ELMo+Attn</td>
<td>76.4/76.1</td>
<td>64.8</td>
<td>79.8</td>
<td>90.4</td>
<td>36.0</td>
<td>73.3</td>
<td>84.9</td>
<td>56.8</td>
<td>71.0</td>
</tr>
<tr>
<td>OpenAI GPT</td>
<td>82.1/81.4</td>
<td>70.3</td>
<td>87.4</td>
<td>91.3</td>
<td>45.4</td>
<td>80.0</td>
<td>82.3</td>
<td>56.0</td>
<td>75.1</td>
</tr>
<tr>
<td>BERT&lt;sub&gt;BASE&lt;/sub&gt;</td>
<td>84.6/83.4</td>
<td>71.2</td>
<td>90.5</td>
<td>93.5</td>
<td>52.1</td>
<td>85.8</td>
<td>88.9</td>
<td>66.4</td>
<td>79.6</td>
</tr>
<tr>
<td>BERT&lt;sub&gt;LARGE&lt;/sub&gt;</td>
<td>86.7/85.9</td>
<td>72.1</td>
<td>92.7</td>
<td>94.9</td>
<td>60.5</td>
<td>86.5</td>
<td>89.3</td>
<td>70.1</td>
<td>82.1</td>
</tr>
</tbody>
</table>

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The “Average” column is slightly different than the official GLUE score, since we exclude the problematic WNLI set. BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.
Understanding Deep Learning Models for NLP
Understanding Deep Learning Models for NLP

- Neural models are good but they suffer from a “black-box” effect
  - Feed an input, get an output.
  - Hard to know what happens in between
  - (This was one of the motivations: no feature engineering)
Understanding Deep Learning Models for NLP

- Neural models are good but they suffer from a “black-box” effect
  - Feed an input, get an output.
  - Hard to know what happens in between
  - (This was one of the motivations: no feature engineering)

- This can be a problem!
  - For critical decision making, having a rationale can be necessary (e.g. recidivism prediction, loan applications...) to ensure that there is no bias (e.g. race, gender...)
  - Idiosyncratic behaviour (adversarial attacks)
Shortcomings: Adversarial Attacks

Figure from Goodfellow et al. (2014)
Shortcomings: Adversarial Attacks

Figure from Goodfellow et al. (2014)
Shortcomings: Adversarial Attacks

Original $x$: Ils le réinvestissent directement en engageant plus de procès.

Attack

Adv. src $\hat{x}$: Ils le réinvestissent directement en engageant plus de procès.

Reference $y$

They plow it right back into filing more troll lawsuits.

Base output $y_M$

They direct it directly by engaging more cases.

Adv. output $\hat{y}_M$

.. de plus.

"On Evaluation of Adversarial Perturbations for Sequence-to-Sequence Models" Michel et al. 2019
Understanding Deep Learning: Probing

- What does my model learn?
  - Does it learn syntax/POS? Does it learn gender/number/tense?
Understanding Deep Learning: Probing

○ What does my model learn?
  > Does it learn syntax/POS? Does it learn gender/number/tense?

○ Probing: Predict attributes of the data
  > Idea: If the model learned the tense of the verb, it should be easy to retrieve it from the learned representation
Understanding Deep Learning: Probing

○ What does my model learn?
  > Does it learn syntax/POS? Does it learn gender/number/tense?

○ Probing: Predict attributes of the data
  > Idea: If the model learned the tense of the verb, it should be easy to retrieve it from the learned representation
  > In practice:
Understanding Deep Learning: Probing

○ What does my model learn?
   > Does it learn syntax/POS? Does it learn gender/number/tense?

○ Probing: Predict attributes of the data
   > Idea: If the model learned the tense of the verb, it should be easy to retrieve it from the learned representation
   > In practice:
Understanding Deep Learning: Probing

- **Probing:** Predict attributes of the data
  - Idea: If the model learned the tense of the verb, it should be easy to retrieve it from the learned representation

<table>
<thead>
<tr>
<th>Task</th>
<th>SentLen</th>
<th>WC</th>
<th>TreeDepth</th>
<th>TopConst</th>
<th>BShift</th>
<th>Tense</th>
<th>SubjNum</th>
<th>ObjNum</th>
<th>SOMO</th>
<th>CoordInv</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline representations</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Majority vote</td>
<td>20.0</td>
<td>0.5</td>
<td>17.9</td>
<td>5.0</td>
<td>50.0</td>
<td>50.0</td>
<td>50.0</td>
<td>50.0</td>
<td>50.0</td>
<td>50.0</td>
</tr>
<tr>
<td>Hum. Eval.</td>
<td>100</td>
<td>100</td>
<td>84.0</td>
<td>84.0</td>
<td>98.0</td>
<td>85.0</td>
<td>88.0</td>
<td>86.5</td>
<td>81.2</td>
<td>85.0</td>
</tr>
<tr>
<td>Length</td>
<td>100</td>
<td>0.2</td>
<td>18.1</td>
<td>9.3</td>
<td>50.6</td>
<td>56.5</td>
<td>50.3</td>
<td>50.1</td>
<td>50.2</td>
<td>50.0</td>
</tr>
<tr>
<td>NB.uni-tfidf</td>
<td>22.7</td>
<td>97.8</td>
<td>24.1</td>
<td>41.9</td>
<td>49.5</td>
<td>77.7</td>
<td>68.9</td>
<td>64.0</td>
<td>38.0</td>
<td>50.5</td>
</tr>
<tr>
<td>NB.bi-tfidf</td>
<td>23.0</td>
<td>95.0</td>
<td>24.6</td>
<td>53.0</td>
<td>63.8</td>
<td>75.9</td>
<td>69.1</td>
<td>65.4</td>
<td>39.9</td>
<td><strong>55.7</strong></td>
</tr>
<tr>
<td>BoV-fastText</td>
<td>66.6</td>
<td>91.6</td>
<td><strong>37.1</strong></td>
<td><strong>68.1</strong></td>
<td>50.8</td>
<td><strong>89.1</strong></td>
<td><strong>82.1</strong></td>
<td><strong>79.8</strong></td>
<td><strong>54.2</strong></td>
<td><strong>54.8</strong></td>
</tr>
<tr>
<td><strong>BiLSTM-last encoder</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Untrained</td>
<td>36.7</td>
<td>43.8</td>
<td>28.5</td>
<td>76.3</td>
<td>49.8</td>
<td>84.9</td>
<td>84.7</td>
<td>74.7</td>
<td>51.1</td>
<td>64.3</td>
</tr>
<tr>
<td>AutoEncoder</td>
<td><strong>99.3</strong></td>
<td>23.3</td>
<td>35.6</td>
<td>78.2</td>
<td>62.0</td>
<td>84.3</td>
<td>84.7</td>
<td>82.1</td>
<td>49.9</td>
<td>65.1</td>
</tr>
<tr>
<td>NMT En-Fr</td>
<td>83.5</td>
<td>55.6</td>
<td>42.4</td>
<td>81.6</td>
<td>62.3</td>
<td>88.1</td>
<td>89.7</td>
<td>89.5</td>
<td>52.0</td>
<td>71.2</td>
</tr>
<tr>
<td>NMT En-De</td>
<td>83.8</td>
<td>53.1</td>
<td>42.1</td>
<td>81.8</td>
<td>60.6</td>
<td>88.6</td>
<td>89.3</td>
<td>87.3</td>
<td>51.5</td>
<td><strong>71.3</strong></td>
</tr>
<tr>
<td>NMT En-Fi</td>
<td>82.4</td>
<td>52.6</td>
<td>40.8</td>
<td>81.3</td>
<td>58.8</td>
<td>88.4</td>
<td>86.8</td>
<td>85.3</td>
<td>52.1</td>
<td>71.0</td>
</tr>
<tr>
<td>Seq2Tree</td>
<td>94.0</td>
<td>14.0</td>
<td><strong>59.6</strong></td>
<td><strong>89.4</strong></td>
<td><strong>78.6</strong></td>
<td><strong>89.9</strong></td>
<td><strong>94.4</strong></td>
<td><strong>94.7</strong></td>
<td>49.6</td>
<td>67.8</td>
</tr>
<tr>
<td>SkipThought</td>
<td>68.1</td>
<td>35.9</td>
<td>33.5</td>
<td>75.4</td>
<td>60.1</td>
<td>89.1</td>
<td>80.5</td>
<td>77.1</td>
<td><strong>55.6</strong></td>
<td>67.7</td>
</tr>
<tr>
<td>NLI</td>
<td>75.9</td>
<td>47.3</td>
<td>32.7</td>
<td>70.5</td>
<td>54.5</td>
<td>79.7</td>
<td>79.3</td>
<td>71.3</td>
<td>53.3</td>
<td>66.5</td>
</tr>
</tbody>
</table>

Figure from Conneau et al. (2018)
Understanding Deep Learning: Probing

○ Probing: Predict attributes of the data
  > More complicated probing tasks
  > Eg. probe language models for syntactic abilities via word prediction
    ■ “The keys to the cabinet [is/are]”
    ■ “Yet the ratio of men who survive to the women and children who survive [is/are]”

Linzen et al. (2016)
Understanding Deep Learning: Probing

- Probing: Predict attributes of the data
  - More complicated probing tasks
  - Eg. probe language models for syntactic abilities via word prediction
    - “The keys to the cabinet [is/are]”
    - “Yet the ratio of men who survive to the women and children who survive [is/are]”

Linzen et al. (2016)
Understanding Deep Learning: Probing

- Caveats of probing
  - Just because we can retrieve information doesn’t mean the model uses it.
  - Not clear how much the model is responsible vs the probing network

Zhang & Bowman (2018)
Understanding Deep Learning: Probing

○ Caveats of probing
  > Just because we can retrieve information doesn’t mean the model uses it.
  > Not clear how much the model is responsible vs the probing network

○ Example: Zhang & Bowman, 2018
  > Compare probing accuracy after the model has been trained on different task
  > Also report probing accuracy for randomly initialized model (no training)
  > Vary amount of training data for the probe
Understanding Deep Learning: Probing

○ Caveats of probing
  > Just because we can retrieve information doesn’t mean the model uses it.
  > Not clear how much the model is responsible vs the probing network

○ Example: Zhang & Bowman, 2018
  > Compare probing accuracy after the model has been trained on different task
  > Also report probing accuracy for randomly initialized model (no training)
  > Vary amount of training data for the probe

Zhang & Bowman (2018)
Why does my model return this prediction?

> “Easy” in a linear model: look at the active features and their weights

\[
\log p(y) \propto w^T_y \cdot f(x)
\]
Understanding Deep Learning: Explaining

○ Why does my model return this prediction?
  > “Easy” in a linear model: look at the active features and their weights
    \[
    \log p(y) \propto w_y^T \cdot f(x)
    \]
  > In neural networks the features don’t “mean” anything: just arbitrary values
Understanding Deep Learning: Explaining

- Why does my model return this prediction?
  - “Easy” in a linear model: look at the active features and their weights
    \[
    \log p(y) \propto w_y^T \cdot f(x)
    \]
  - In neural networks the features don’t “mean” anything: just arbitrary values

- Possible solutions:
  - Word importance scores: what is the word that was the most important for this decision
  - Influence functions: which was the training example that was most responsible for teaching the model this decision
  - Build models that are easier to interpret (but still as good): Attention \( \sim \) baked in word importance?
Understanding Deep Learning: Explaining

○ Word importance scores demo

https://demo.allennlp.org/masked-Lm
Downsides of Deep Learning

- **Need a lot of data**
  - Otherwise: overfitting, etc...
  - Problem for low resource languages
  - Or tasks where annotation is expensive
  - Pretrained models help

- **Compute intensive**
  - Training can take many GPU hours
  - Large number of parameters
  - Transformer, hardware...
  - Distillation, pruning...

- **Hard to interpret**
  - Hard to explain why the model is making a prediction
  - Not clear what the model is learning
  - Word importance scores, etc...
  - Probing, etc...

- **Robustness**
  - Can be brittle in the face of (well chosen) noise
  - Doesn’t work as well on different domains
  - Robustness
  - Domain adaptation
Would you like to know more?
Questions?