Announcements

• Final project due soon: go to office hours!

• More poster session details on the website
  • If you can’t make some or all of the poster session, fill out the Google form ASAP!
Lecture Plan:

• Why use semi-supervised learning for NLP?

• Semi-supervised learning algorithms
  • Pre-training
    • Three recent papers
  • Self-training
  • Consistency regularization
    • Two recent papers
Why has deep learning been so successful recently?
Why has deep learning been so successful recently?

Gentlemen, our learner overgeneralizes because the VC-Dimension of our Kernel is too high. Get some experts and minimize the structural risk in a new one. Rework our loss function, make the next kernel stable, unbiased and consider using a soft margin.

Lecture 1, Slide 4
Why has deep learning been so successful recently?
Why has deep learning been so successful recently?

• Better “tricks” (dropout, improved optimizers (e.g., Adam), batch norm, attention)

• Better hardware (thanks video games!) -> larger models

• Larger datasets
Big deep learning successes

- Image Recognition: Widely used by Google, Facebook, etc.
- Machine Translation: Google translate, etc.
- Game Playing: Atari Games, AlphaGo, and more
Big deep learning successes

• Image Recognition:
  ImageNet: 14 million examples

• Machine Translation:
  WMT: Millions of sentence pairs

• Game Playing:
  10s of millions of frames for Atari AI
  10s of millions of self-play games for AlphaZero
## NLP Dataset Sizes

<table>
<thead>
<tr>
<th>Dataset (English)</th>
<th>Size (# sentences)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NER</td>
<td>15K (CoNLL 2003)</td>
</tr>
<tr>
<td>Coreference Resolution</td>
<td>75K (OntoNotes)</td>
</tr>
<tr>
<td>Parsing</td>
<td>40K (Penn Treebank)</td>
</tr>
<tr>
<td>Question Answering</td>
<td>100k questions (SQuAD)</td>
</tr>
<tr>
<td>Textual Entailment</td>
<td>570k (SNLI)</td>
</tr>
<tr>
<td>Sentiment Analysis</td>
<td>10k (SST)</td>
</tr>
</tbody>
</table>
And that’s for core tasks in English!

- Most tasks have less data

- There are thousands of languages
  - Hundreds with > 1 million native speakers
  - Less than 10% of people speak English as their first language
  - Little-to-no annotated data for many other languages
What to do?

- Just collect more data?
  - Expensive!
    - Crowdsourcing alleviates this a bit
  - Requires linguistic knowledge for some tasks

- Semi-Supervised Learning
  - Use **unlabeled** examples during training
  - Easy to find for NLP!
Two Moons Dataset

- Toy dataset that we will use as an example throughout the rest of the lecture
Two Moons Dataset

- Semi-supervised: most examples don’t have a label
• A supervised model would learn something like this
But this of course is wrong, the model will make lots of mistakes
Semi-Supervised Learning

- We will cover three semi-supervised learning techniques
  - Pre-training
    - One of the tricks that started to make NNs successful
    - You learned about this in week 1 (word2vec)!
  - Self-training
    - One of the oldest and simplest semi-supervised learning algorithms (1960s)
  - Consistency regularization
    - Recent idea (2014, lots of active research)
    - Great results for both computer vision and NLP
Pre-training

- First train an unsupervised model on unlabeled data
- Then incorporate the model’s learned weights into a supervised model and train it on the labeled data
  - Optional: continue fine-tuning the unsupervised weights.

1. **pre-training phase**
   - unsupervised-only part of the model
   - shared part of the model
   - big corpus of unlabeled data
   - unsupervised learning

2. **supervised learning phase**
   - supervised-only part of the model
   - shared part of the model
   - initialized weights
   - smaller corpus of labeled data
   - supervised learning
Pre-training: Word2Vec

- Shared part is word embeddings
- No unsupervised-only part
- Supervised-only part is the rest of the model
- Unsupervised learning: skip-gram/cbow/glove/etc
- Supervised learning: training on some NLP task

1. pre-training phase
   - unsupervised-only part of the model
   - shared part of the model
   - big corpus of unlabeled data
   - unsupervised learning

2. supervised learning phase
   - supervised-only part of the model
   - shared part of the model
   - initialize weights
   - supervised learning
   - smaller corpus of labeled data
Why does pre-training work?

• "Smart" initialization for the model
• More meaningful representations in the model
  • e.g., GloVe vectors capture a lot about word meaning, our model no longer has to learn the meanings itself

Supervised learning: have to learn everything from “raw” input

Pre-training: supervised part gets more useful representations as input
Why does pre-training work?

original representation space

learned representation space after pre-training
Why does pre-training work?

original representation space

learned representation space after pre-training

Supervised part of the model has a much easier job after pre-training
Pre-Training for NLP

- Most neural NLP models look (roughly) like this
Pre-Training for NLP

- With pre-trained embeddings

Supervised

Pre-trained

- Inputs (words)
  - Embedding lookup
  - Encoder NN(s) (CNN/BiLSTM/Transformer/Etc.)
  - Prediction NN (e.g., mean-pool then softmax layer)
  - Prediction(s)
Pre-Training for NLP

- Recent research: pre-train more of the model (e.g., the first LSTM layer)
Pre-Training Strategies: Auto-Encoder (Dai & Le, 2015)

- For pre-training, train an autoencoder: seq2seq model (without attention) where the target sequence is the input sequence
  - the encoder converts the input into a vector that contains enough information that the input can be recovered
- Initialize the LSTM for a sentence classification model with the encoder

![Diagram showing pre-training and supervised learning stages with LSTM Encoder and Decoder, Softmax Layer, and positive sentiment initialization.]
Pre-Training Strategies: Auto-Encoder (Dai & Le, 2015)

- For pre-training, train an **autoencoder**: seq2seq model (without attention) where the target sequence is the input sequence
  - the encoder converts the input into a vector that contains enough information that the input can be recovered
- Initialize the LSTM for a sentence classification model with the encoder

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Previous Best Result</th>
<th>Supervised Baseline</th>
<th>With Pretraining</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMDB</td>
<td>7.42</td>
<td>10.00</td>
<td>7.24</td>
</tr>
<tr>
<td>Rotten Tomatoes</td>
<td>18.5</td>
<td>20.5</td>
<td>16.7</td>
</tr>
<tr>
<td>20 Newsgroups</td>
<td>17.1</td>
<td>18.0</td>
<td>15.6</td>
</tr>
</tbody>
</table>
Pre-Training Strategies: CoVe (McCann et al., 2017)

- Pre-train the encoder for machine translation
  - So really this could be considered transfer learning, not semi-supervised learning
- Don’t update the pre-trained part of the model during training
- Train bigger NN on top of the pre-trained encoder
  - e.g., BiLSTM layers as well as just a softmax layer

![Diagram of pre-training and supervised learning with LSTM Encoder and Decoder, initializing and fine-tuning.]
Pre-Training Strategies: CoVe (McCann et al., 2017)

• Why not fine-tune the pre-trained encoder?
  • Much faster during training! In a preprocessing step run the encoder once over each example
  • Then treat the outputs as fixed vectors (like GloVe vectors) that the model takes as input

Supervised BiLSTM

GloVe vectors

CoVe vectors (produced by NMT encoder running over the sentence)

It was good positive sentiment
Pre-Training Strategies: CoVe (McCann et al., 2017)

- Why not fine-tune the pre-trained encoder?
  - Much faster during training! In a preprocessing step run the encoder once over each example
  - Then treat the outputs as fixed vectors (like GloVe vectors) that the model takes as input
Pre-Training Strategies: ELMo (Peters et al., 2017)

- Similar to CoVe but
  - Pre-train the model for language modeling (both forwards and backwards)
  - Scaled-up: much larger model, much more data
  - A few other tricks...

pre-training
“was good <EOS>”

Softmax over Vocab

LSTM Encoder

supervised learning
positive sentiment

initialize, don’t fine-tune

“It was good”

LSTM Encoder

“It was good”

Supervised Model
Pre-Training Strategies: ELMo (Peters et al., 2018)

- Tricks:
  - Fine-tune the LM on the supervised dataset
  - Combine the LM representations in a smart way

\[
\text{ELMo}_{k}^{\text{task}} = \gamma^{\text{task}} \sum_{j=0}^{L} s_{j}^{\text{task}} h_{k,j}^{\text{LM}}.
\]

- Pass the ELMo representations into multiple layers of the supervised model, not just the first layer
Pre-Training Strategies: ELMo (Peters et al., 2018)

- Amazing Results
  - This kind of method may become standard-practice in NLP the way pretrained embeddings is standard-practice currently

<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>SAN</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>
Pre-Training Overview

- Initialize part of the model with a network trained using unsupervised learning
- Works great!
- But requires training a separate (usually extremely large) model
  - e.g., ELMo uses a 2-layer BiLSTM with 4096 units in each layer, also incorporated a size 2048 character-level CNN, pre-trained with 10 passes over a 1 billion word corpus
Self-Training

• Use unlabeled data without a giant model or long pretraining phase
• Old (1960s) and simple semi-supervised algorithm
• Algorithm:
  1. Train the model on the labeled data.
  2. Have the model label the unlabeled data.
     • Take some of examples the model is most confident about (i.e., the model gives them high probability). Add those examples with the model’s labels to the training set
  3. Go back to 1.
Self-Training: Example

• 1. Train our model on the labeled data
• 2. Label a few examples
Self-Training: Example

• 3. Re-train the model
Self-Training: Example

- Repeat!
Self-Training: Example
Self-Training: Example
Self-Training: Example
Self-Training:

- Good results on quite a few NLP tasks in the 1990s and 2000s
  - e.g., for constituency parsing (McClosky et al., 2006)
- However, not used as much lately (especially with NNs) because other methods work better

<table>
<thead>
<tr>
<th>Sentences added</th>
<th>F1-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (baseline)</td>
<td>91.8</td>
</tr>
<tr>
<td>50k</td>
<td>91.8</td>
</tr>
<tr>
<td>250k</td>
<td>91.8</td>
</tr>
<tr>
<td>500k</td>
<td>92.0</td>
</tr>
<tr>
<td>750k</td>
<td>92.0</td>
</tr>
<tr>
<td>1,000k</td>
<td>92.1</td>
</tr>
<tr>
<td>1,500k</td>
<td>92.1</td>
</tr>
<tr>
<td>1,750k</td>
<td>92.1</td>
</tr>
<tr>
<td>2,000k</td>
<td>92.2</td>
</tr>
</tbody>
</table>

Table 2: f-scores from evaluating the reranking parser on three held-out sections after adding reranked sentences from NANC to WSJ training.

These evaluations were performed on all sentences. In Table 2, we see that the new NANC data contains some information orthogonal to the reranker and improves parsing accuracy of the reranking parser.

Up to this point, we have only considered giving our true training data a relative weight of one. Increasing the weight of the Wall Street Journal data should improve, or at least not hurt, parsing performance. Indeed, this is the case for both the parser (figure not shown) and reranking parser (Figure 1).

Adding more weight to the Wall Street Journal data ensures that the counts of our events will be closer to our more accurate data source while still incorporating new data from NANC. While it appears that the performance still levels off after adding about one million sentences from NANC, the curve corresponding to higher WSJ weights achieve a higher asymptote. Looking at the performance of various weights across sections 1, 22, and 24, we decided that the best combination of training data is to give WSJ a relative weight of 5 and use the first 1,750k reranker-best sentences from NANC.

Finally, we evaluate our new model on the test section of Wall Street Journal. In Table 3, we note that baseline system (i.e. the parser and reranker trained purely on Wall Street Journal) has improved by 0.3% over Charniak and Johnson (2005). The 92.1% f-score is the 1.1% absolute improvement mentioned in the abstract. The improvement from self-training is significant in both macro and micro tests (p<10⁻⁵).
Online Self-Training

1. Sample a labeled minibatch \((x_i, y_i)\) and unlabeled minibatch \(x_j\)
2. Take a step of gradient descent minimizing

\[
J(\theta) = CE(y_i, p(y|x_i, \theta)) + CE(\text{one-hot}(\text{argmax}(p(y|x_j, \theta))), p(y|x_j, \theta))
\]

- Regular supervised loss
- Target output is a human-produced label
- Model acts as a “teacher” and labels the examples
- Target is a model-produced label. It will be noisy because the model isn’t as accurate as a person, but hopefully the model can still learn from it
- Then model acts as a “student” and learns to match the target
Online Self-Training

1. Sample a labeled minibatch \((x_i, y_i)\) and unlabeled minibatch \(x_j\)

2. Take a step of gradient descent minimizing

\[
J(\theta) = CE(y_i, p(y|x_i, \theta)) + CE(\text{one-hot} (\text{argmax}(p(y|x_j, \theta))), p(y|x_j, \theta))
\]

Regular supervised loss

Target output is a human-produced label

Model acts as a “teacher” and labels the examples

Target is a model-produced label. It will be noisy because the model isn’t as accurate as a person, but hopefully the model can still learn from it

Then model acts as a “student” and learns to match the target

Rest of this lecture: how do we make this second term (the unsupervised one) better?
**Hard vs Soft Targets**

- Why use “hard” one-hot label as the target on unlabeled examples? Wouldn’t a “soft” probability distribution work better?

**One-hot vector** just tells us $y_1$ is the most likely class

**Probability distribution** also tells us the model isn’t very confident about its prediction and that $y_2$ is the second-most-likely class
Hard vs Soft Targets

• Why use “hard” one-hot label as the target on unlabeled examples? Wouldn’t a “soft” probability distribution work better?

\[
J(\theta) = CE(\text{one\_hot}(\text{argmax}(p(y|x_j, \theta))), p(y|x_j, \theta))
\]

\[
J(\theta) = CE(p(y|x_j, \theta), p(y|x_j, \theta))
\]
Hard vs Soft Targets

• Why use “hard” one-hot label as the target on unlabeled examples? Wouldn’t a “soft” probability distribution work better?

\[
J(\theta) = CE(\text{one}_\text{hot}(\text{argmax}(p(y|x_j, \theta))), p(y|x_j, \theta))
\]

\[
J(\theta) = CE(p(y|x_j, \theta), p(y|x_j, \theta))
\]

• A bit odd: the student already matches the targets!
Consistency Regularization

- Add noise to the student’s inputs

\[ J(\theta) = CE(p(y|x_j, \theta), p(y|x_j + \eta, \theta)) \]

- Where \( \eta \) is a vector with a random direction and a small magnitude \( \epsilon \)

- Soft target

- Model learns to produce target even when noise is added to its input
Consistency Regularization

- Add noise to the student’s inputs
  \[ J(\theta) = CE(p(y|x_j, \theta), p(y|x_j + \eta, \theta)) \]
  - Where \( \eta \) is a vector with a random direction and a small magnitude \( \epsilon \)

- Train the model so a bit of noise doesn’t mess up its predictions
- Equivalently, the model must give consistent predictions to nearby data points

The model is trained to give the same prediction for any point in the circle

"distributional smoothing"
Consistency Regularization: Example
Consistency Regularization: Example

Model should produce the same predictions everywhere in the circles -> overlapping circles should have the same prediction
Consistency Regularization: Example

Decision boundary will look like this
Amazing Results for Computer Vision!

- Results on small image recognition dataset: 4K labeled examples, 46K unlabeled examples

<table>
<thead>
<tr>
<th>Model</th>
<th>Error Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised</td>
<td>35.56</td>
</tr>
<tr>
<td>Ladder Network (Rasmus et al, 2015)</td>
<td>20.40</td>
</tr>
<tr>
<td>CatGAN (Springenberger, 2016)</td>
<td>19.58</td>
</tr>
<tr>
<td>GAN (Salimans et al., 2016)</td>
<td>18.63</td>
</tr>
<tr>
<td>Consistency Regularization (Sajjadi et al., 2016)</td>
<td>11.29</td>
</tr>
</tbody>
</table>
How to Apply Consistency Regularization to NLP?

- \( J(\theta) = CE(p(y|x_j, \theta), p(y|x_j + \eta, \theta)) \)

- In NLP, \( x_j \) is a sequence of words
- Unlike with pixels in an image, words are discrete. How can we add random noise to them?
- 3 ideas:
  - Miyato et al. (2017)
    - Add noise to the word embeddings
  - Clark et al. (2018)
    - Word dropout
    - Cross-view Consistency
Virtual Adversarial Training (Miyato et al., 2017)

• Apply consistency regularization to text classification
  • First embed the words
  • Add the noise to the word embeddings
  • Have to constrain the word embeddings (e.g., make them have zero mean and unit variance)
    • Otherwise the model could just make them have really large magnitude so the noise doesn’t change anything

• Noise added to the word embeddings is not chosen randomly: it is chosen adversarially
Adversarial Examples

- **Adversarial examples**: Small (imperceptible to humans) tweak to neural network inputs can change its output

![Image](image_url)
Adversarial Examples

- Security implications
Adversarial Examples

- **Adversarial examples**: Small (imperceptible to humans) tweak to neural network inputs can change its output

![Image of adversarial examples](image.png)

- **Creating an adversarial example**: 
  - Compute the gradient of the loss with respect to the input 
  - Add epsilon times the gradient to the input 
    - Possibly repeat multiple times

Lecture 1, Slide 58
Virtual Adversarial Training

\[ J(\theta) = CE(p(y|x_j, \theta), p(y|x_j + \eta, \theta)) \]

\[ \eta = \epsilon \frac{\nabla_x J}{\|\nabla_x J\|} \]

Instead of picking a random direction for \( \eta \), pick the one that most increases the loss.
Virtual Adversarial Training: Results

- Results on sentiment classification task

<table>
<thead>
<tr>
<th>Model</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pretraining Only</td>
<td>7.33</td>
</tr>
<tr>
<td>Pretraining + consistency reg. (random perturbation)</td>
<td>6.78</td>
</tr>
<tr>
<td>Pretraining + consistency reg. (adversarial perturbation)</td>
<td>5.91</td>
</tr>
</tbody>
</table>
Word Dropout

- Much simpler idea:
  - We can’t add noise to words easily
  - Instead let’s randomly (10-20% probability) replace words in the input with a special \texttt{REMOVED} token

\[
J(\theta) = CE(p(y|x_j, \theta), p(y|\text{drop\_words}(x_j), \theta))
\]

- A lot simpler than Virtual Adversarial Training!
- And actually works better in many cases
Cross-View Consistency (Clark et al., 2018)

- Word dropout causes the model (when acting as the “student”) to see a restricted view of the input
  
  Original input: “They traveled to Washington by plane”
  
  Restricted view: “They ______ to Washington by ____”

- Cross-view Consistency: instead train the model across many different views of the input at once

When making a prediction about “Washington”:

View 1: They traveled to Washington _____
View 2: They traveled to _____________
View 3: __________ Washington by plane
View 4: __________________________ by plane

When making a prediction about “to”:

View 1: They traveled to ______________
View 2: They traveled __________________
View 3: _________ to Washington by plane
View 4: __________ Washington by plane

- Sounds nice, but wouldn’t this train $4n$ times as slow?
Cross-View Consistency (Clark et al., 2018)

- Sounds nice, but wouldn’t this train \(4n\) times as slow?

- Instead of running full the model multiple times, add multiple “auxiliary” softmax layers to the model
  - e.g., add one to the forward LSTM in the first BiLSTM layer. It doesn’t see any words to the right of the current one
  - Trains these predictions to match the “primary” prediction from a softmax layer that sees all of the input

\[
J(\theta) = \sum_{i=1}^{k} CE(p(y|x_j, \theta), p_{\text{view}_i}(y|x_j, \theta))
\]

Primary softmax layer that sees all the input

Auxiliary softmax layers that see restricted views of the input
Cross-View Consistency (Clark et al., 2018)

Learning on a Labeled Example

“Washington is a state located in...”

Model

\(\hat{y}\)

Learning on an Unlabeled Example

“They traveled to Washington by plane.”

Model acting as the teacher

\(\hat{y}\)

Model acting as the student

\(\hat{y}_{view_1}\), \(\hat{y}_{view_2}\), \(\hat{y}_{view_3}\), \(\hat{y}_{view_4}\)

Inputs Seen by Student Prediction Layers:

view 1: “They traveled to _______”
view 2: “___________ by plane”
view 3: “They traveled to Washington ______”
view 4: “___________ Washington by plane”

- Model first learns “Washington” is usually a location from the labeled data. So on the unlabeled example it can guess “Washington” refers to a location

- Then on the unlabeled example it learns a location usually follows “They traveled to”
• **Forward**: attached to forward LSTM, produces predictions without seeing the right context of the current token.

• **Future**: attached to forward LSTM, produces prediction without seeing the right context or the current token itself.
**Model:** standard seq2seq with attention

Auxiliary layers use the same LSTM but different attention and softmax weights.
**Model:** standard seq2seq with attention

**Auxiliary prediction layer 1:** predict the word after next

---

The diagram illustrates the auxiliary prediction layers for machine translation. The model is a standard seq2seq with attention. The auxiliary prediction layer 1 is designed to predict the word after the next word in the sequence. The diagram shows the attention scores and attention distribution, with the encoder RNN processing the input sequence and the decoder predicting the output sequence. The input sequence includes the words "les pauvres sont démunis" and the output sequence includes "have the poor."
Auxiliary Prediction Layers for Machine Translation

**Model:** standard seq2seq with attention

**Auxiliary prediction layer 2:**
- attention dropout: model only attends to a subset of the source sentence

Diagram:
- Encoder RNN
- Attention scores
- Attention distribution
- Attention output
- don’t
- ŷ₃
- les pauvres sont démunis
- <START> the poor
Cross-View Consistency (Clark et al., 2018)

- Model makes multiple predictions $\hat{y}_{\text{view}_1}, \hat{y}_{\text{view}_2}, \ldots, \hat{y}_{\text{view}_k}$
  - Each one using a different softmax layer
- Trains these predictions to match the “primary” prediction $\hat{y}$ from a softmax layer that sees all of the input
- Loss function:

$$J(\theta) = \sum_{i=1}^{k} CE(p(y|x_j, \theta), p_{\text{view}_i}(y|x_j, \theta))$$

Primary softmax layer that sees all the input

Auxiliary softmax layer that see restricted views of the input
Cross View Consistency: Advantages

• Much more data efficient than word dropout because the model learns to produce good predictions across many views of the input at once instead of just one

• Not much slower because a few extra softmax layers are computationally cheap compared to the LSTMs
Cross View Consistency: Results

<table>
<thead>
<tr>
<th>Method</th>
<th>CCG</th>
<th>Chunk</th>
<th>NER</th>
<th>Dep. Parsing</th>
<th>MT (English-Vietnamese)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous state-of-the-art</td>
<td>95.1</td>
<td>96.4</td>
<td>92.2</td>
<td>94.1</td>
<td>26.1</td>
</tr>
</tbody>
</table>

- Chunking and NER results are using ELMo, rest are from supervised classifiers
# Cross View Consistency: Results

<table>
<thead>
<tr>
<th>Method</th>
<th>CCG</th>
<th>Chunk</th>
<th>NER</th>
<th>Dep. Parsing</th>
<th>MT (English-Vietnamese)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous state-of-the-art</td>
<td>95.1</td>
<td>96.4</td>
<td>92.2</td>
<td>94.1</td>
<td>26.1</td>
</tr>
<tr>
<td>Supervised</td>
<td>94.8</td>
<td>94.9</td>
<td>91.2</td>
<td>93.3</td>
<td>28.9</td>
</tr>
</tbody>
</table>
## Cross View Consistency: Results

<table>
<thead>
<tr>
<th>Method</th>
<th>CCG</th>
<th>Chunk</th>
<th>NER</th>
<th>Dep. Parsing</th>
<th>MT (English-Vietnamese)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous state-of-the-art</td>
<td>95.1</td>
<td>96.4</td>
<td>92.2</td>
<td>94.1</td>
<td>26.1</td>
</tr>
<tr>
<td>Supervised</td>
<td>94.8</td>
<td>94.9</td>
<td>91.2</td>
<td>93.3</td>
<td>28.9</td>
</tr>
<tr>
<td>Virtual Adversarial</td>
<td>95.1</td>
<td>95.2</td>
<td>91.6</td>
<td>93.7</td>
<td>--</td>
</tr>
<tr>
<td>Word Dropout</td>
<td><strong>95.2</strong></td>
<td>95.8</td>
<td>92.1</td>
<td>93.8</td>
<td><strong>29.3</strong></td>
</tr>
</tbody>
</table>
## Cross View Consistency: Results

<table>
<thead>
<tr>
<th>Method</th>
<th>CCG</th>
<th>Chunk</th>
<th>NER</th>
<th>Dep. Parsing</th>
<th>MT (English-Vietnamese)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Previous state-of-the-art</td>
<td>95.1</td>
<td>96.4</td>
<td>92.2</td>
<td>94.1</td>
<td>26.1</td>
</tr>
<tr>
<td>Supervised</td>
<td>94.8</td>
<td>94.9</td>
<td>91.2</td>
<td>93.3</td>
<td>28.9</td>
</tr>
<tr>
<td>Virtual Adversarial</td>
<td>95.1</td>
<td>95.2</td>
<td>91.6</td>
<td>93.7</td>
<td>--</td>
</tr>
<tr>
<td>Word Dropout</td>
<td>95.2</td>
<td>95.8</td>
<td>92.1</td>
<td>93.8</td>
<td>29.3</td>
</tr>
<tr>
<td>CVC</td>
<td>95.6</td>
<td>96.5</td>
<td>92.4</td>
<td>94.2</td>
<td>29.7</td>
</tr>
</tbody>
</table>
Cross View Consistency: Results

Accuracy vs Amount of labeled data

Accuracy vs Model size
Conclusion

- Lots of recent work on semi-supervised learning resulting in big improvements on many tasks!

- Provides a way to scale up models even when there isn’t much labeled data