Natural Language Processing with Deep Learning CS224N

Lecture 17
Semi-Supervised Learning for NLP

Kevin Clark

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!

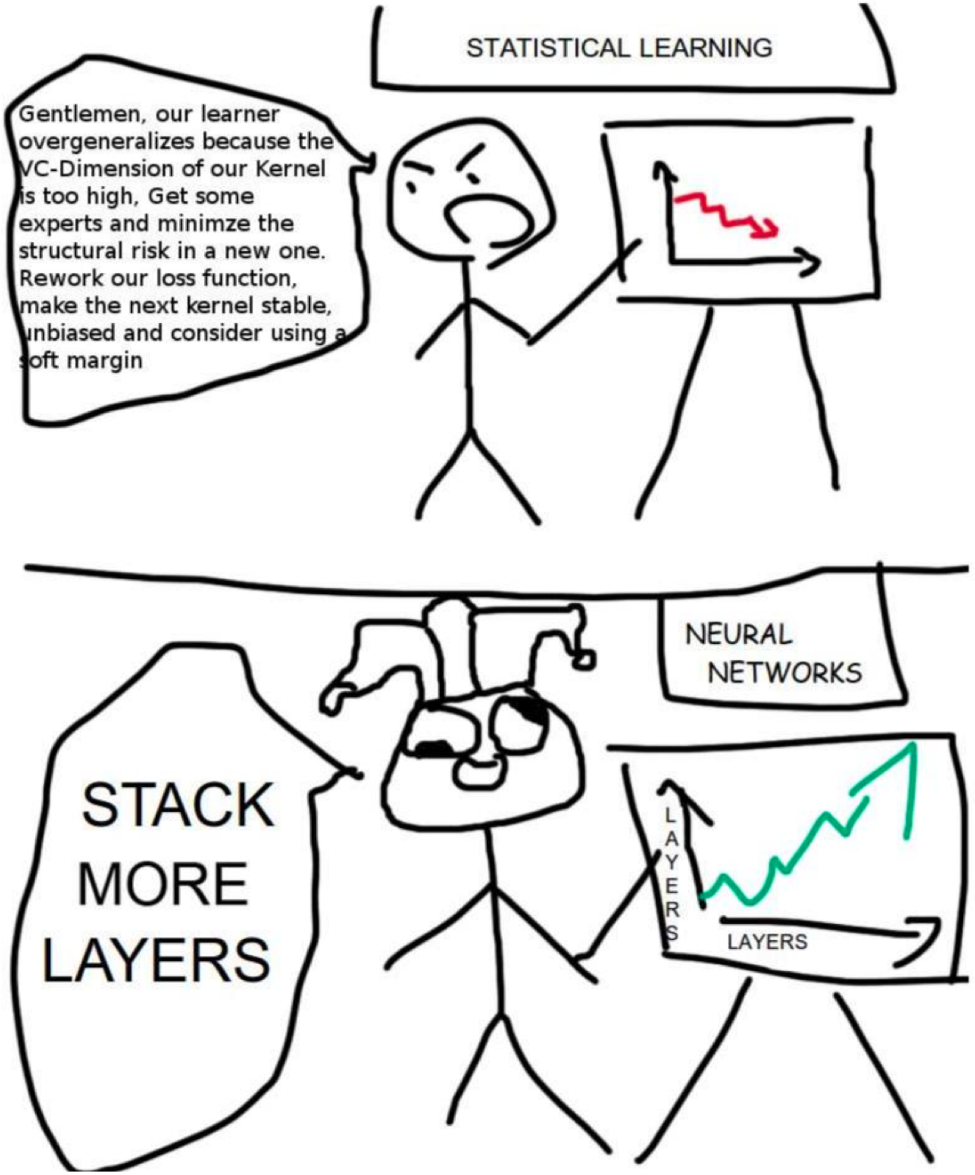
Lecture Plan:

- Why use semi-supervised learning for NLP?

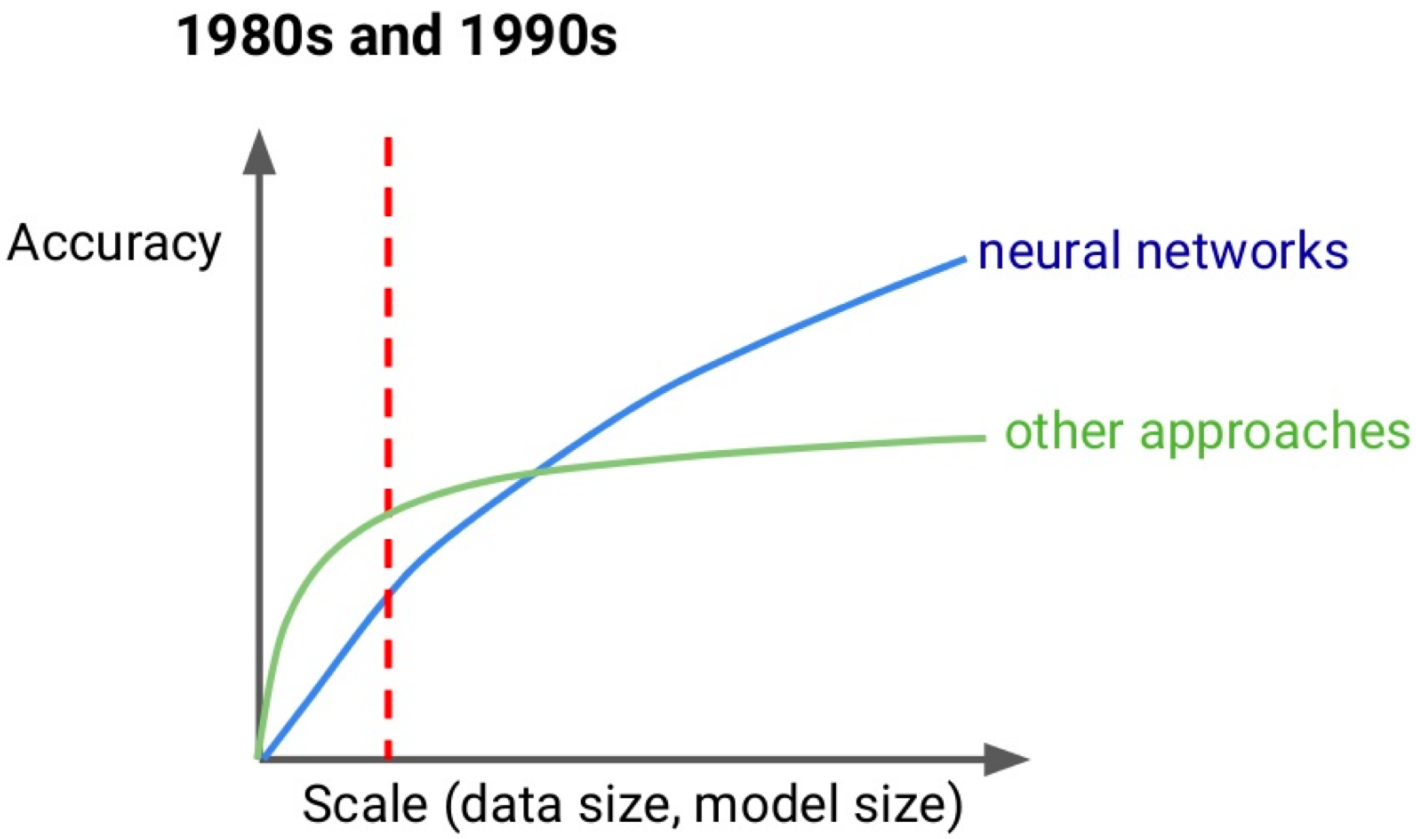
- Semi-supervised learning algorithms
  - Pre-training
    - Three recent papers
  - Self-training
  - Consistency regularization
    - Two recent papers

Why has deep learning been so successful recently?

Why has deep learning been so successful recently?



Why has deep learning been so successful recently?

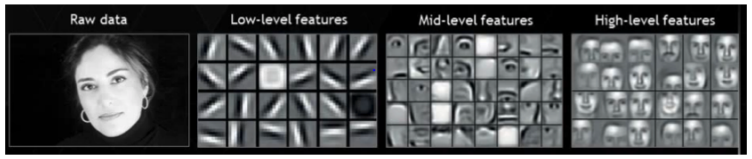


Why has deep learning been so successful recently?

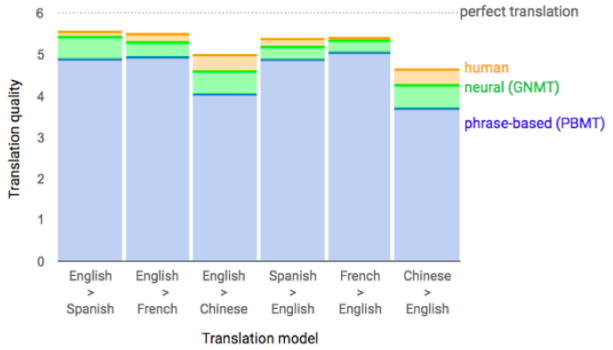
- Better “tricks” (dropout, improved optimizers (e.g., Adam), batch norm, attention)
- Better hardware (thanks video games!) -> larger models
- **Larger datasets**

Big deep learning successes

- Image Recognition:
  Widely used by Google, Facebook, etc.



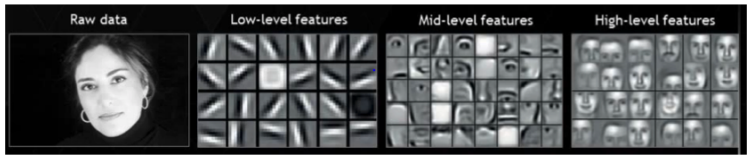
- Machine Translation:
  Google translate, etc.



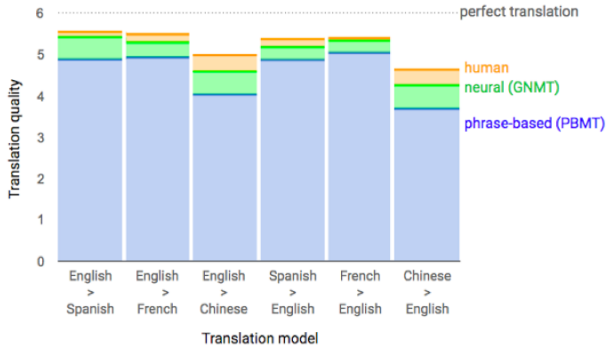
- Game Playing:
  Atari Games, AlphaGo, and more

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

Dataset (English)	Size (# sentences)
NER	15K (CoNLL 2003)
Coreference Resolution	75K (OntoNotes)
Parsing	40K (Penn Treebank)
Question Answering	100k questions (SQuAD)
Textual Entailment	570k (SNLI)
Sentiment Analysis	10k (SST)

…And that's for core tasks in English!

- Most tasks have less data

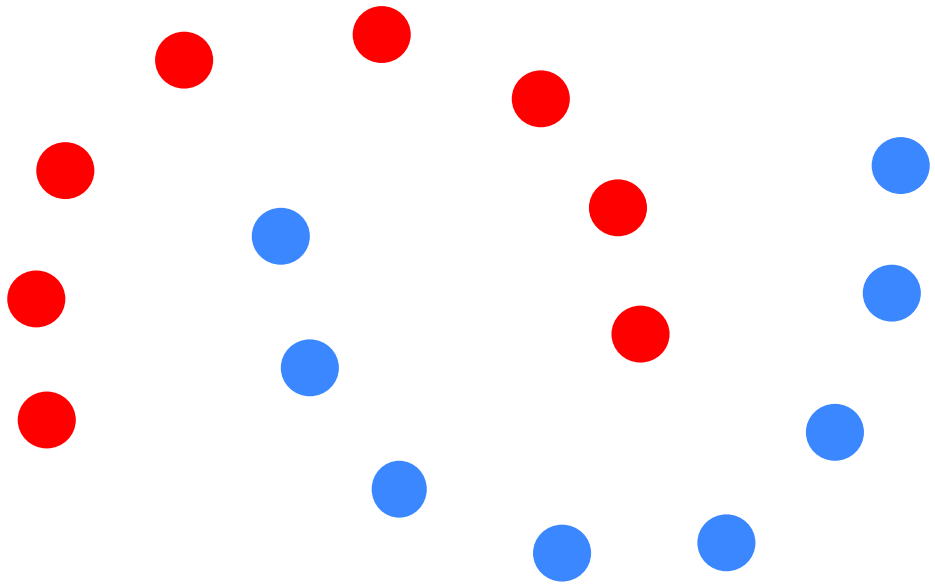
- There are thousands of languages
  - Hundreds with > 1 million native speakers
  - Less than 10% of people speak English as their first language
  - Little-to-no annotated data for many other languages

What to do?

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

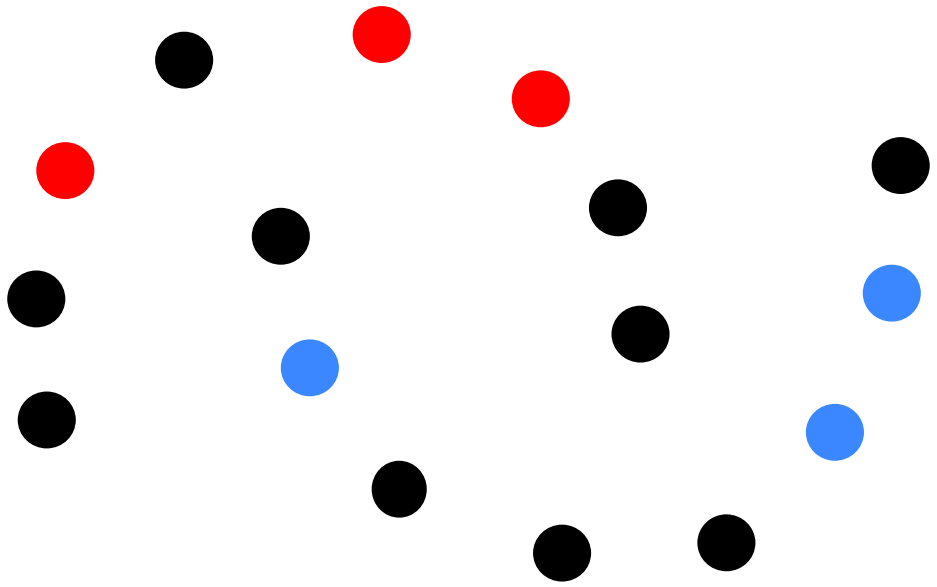
- Semi-Supervised Learning
  - Use **unlabeled** examples during training
  - Easy to find for NLP!

Two Moons Dataset



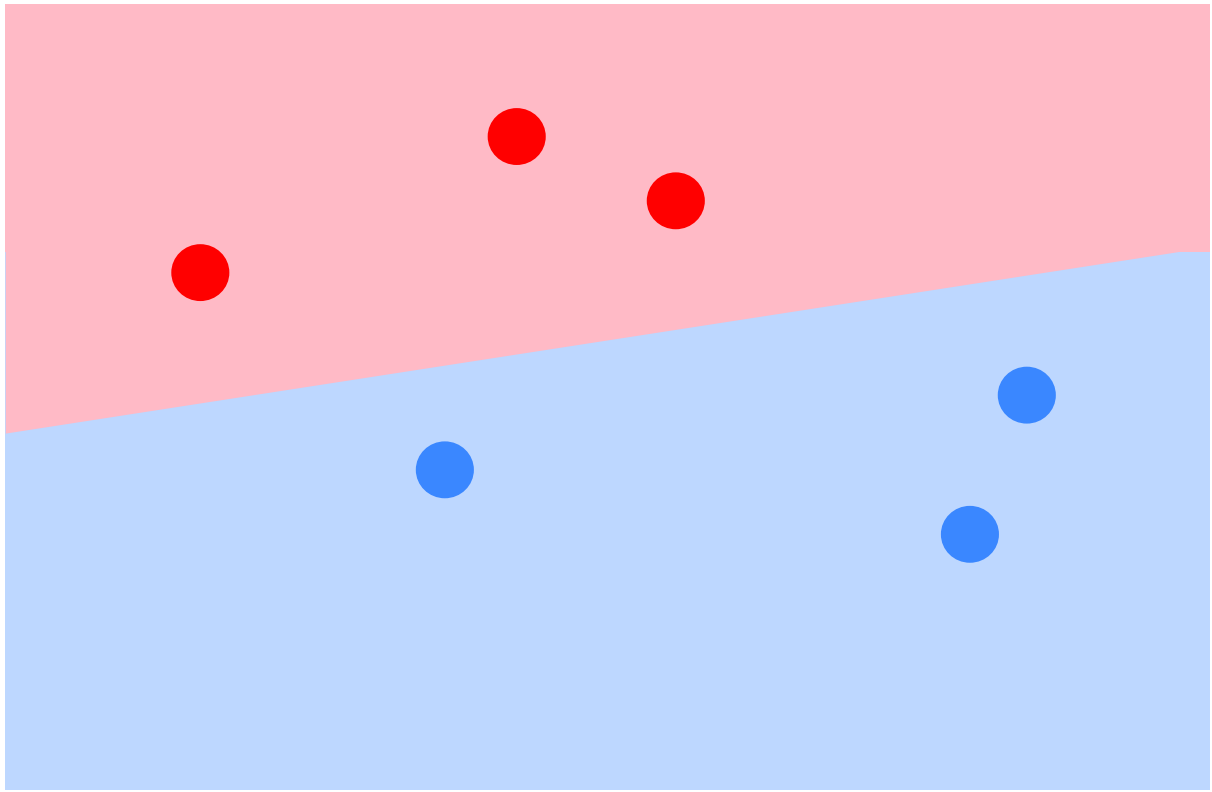
- Toy dataset that we will use as an example throughout the rest of the lecture

Two Moons Dataset



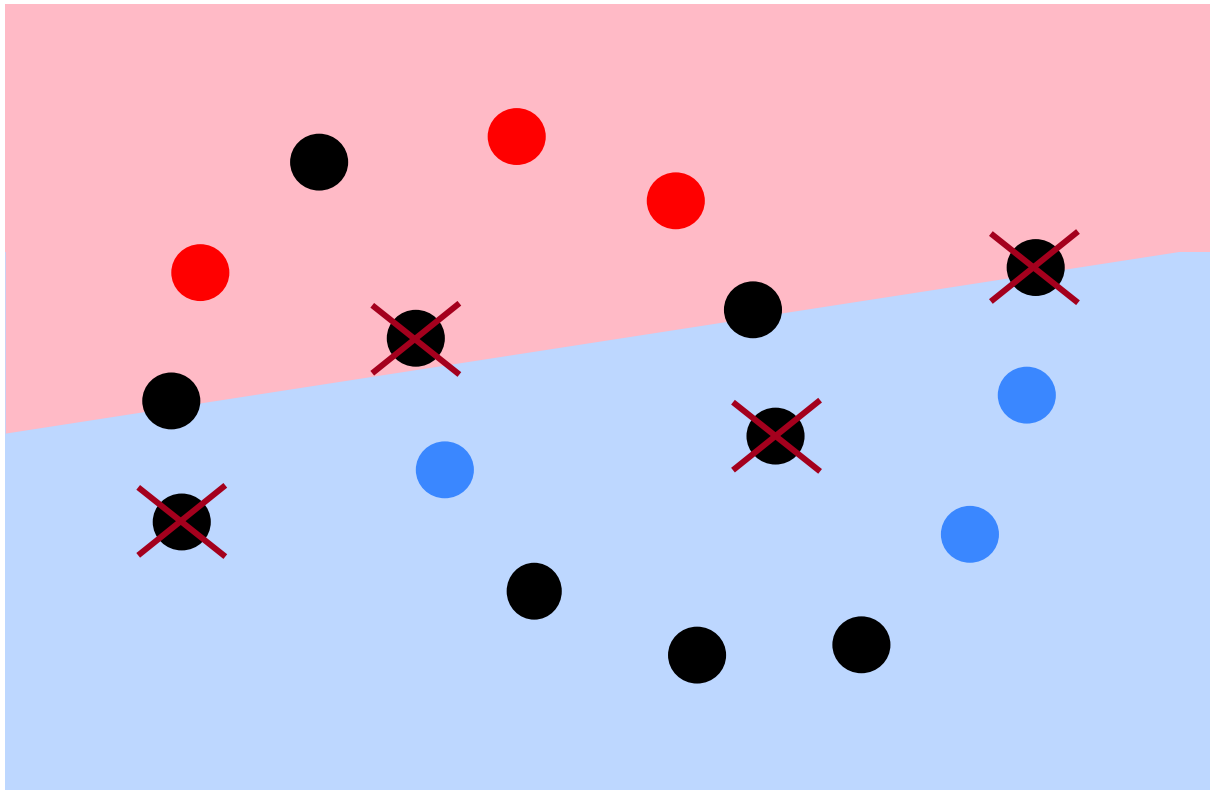
- Semi-supervised: most examples don’t have a label

Two Moons Dataset



- A supervised model would learn something like this

Two Moons Dataset



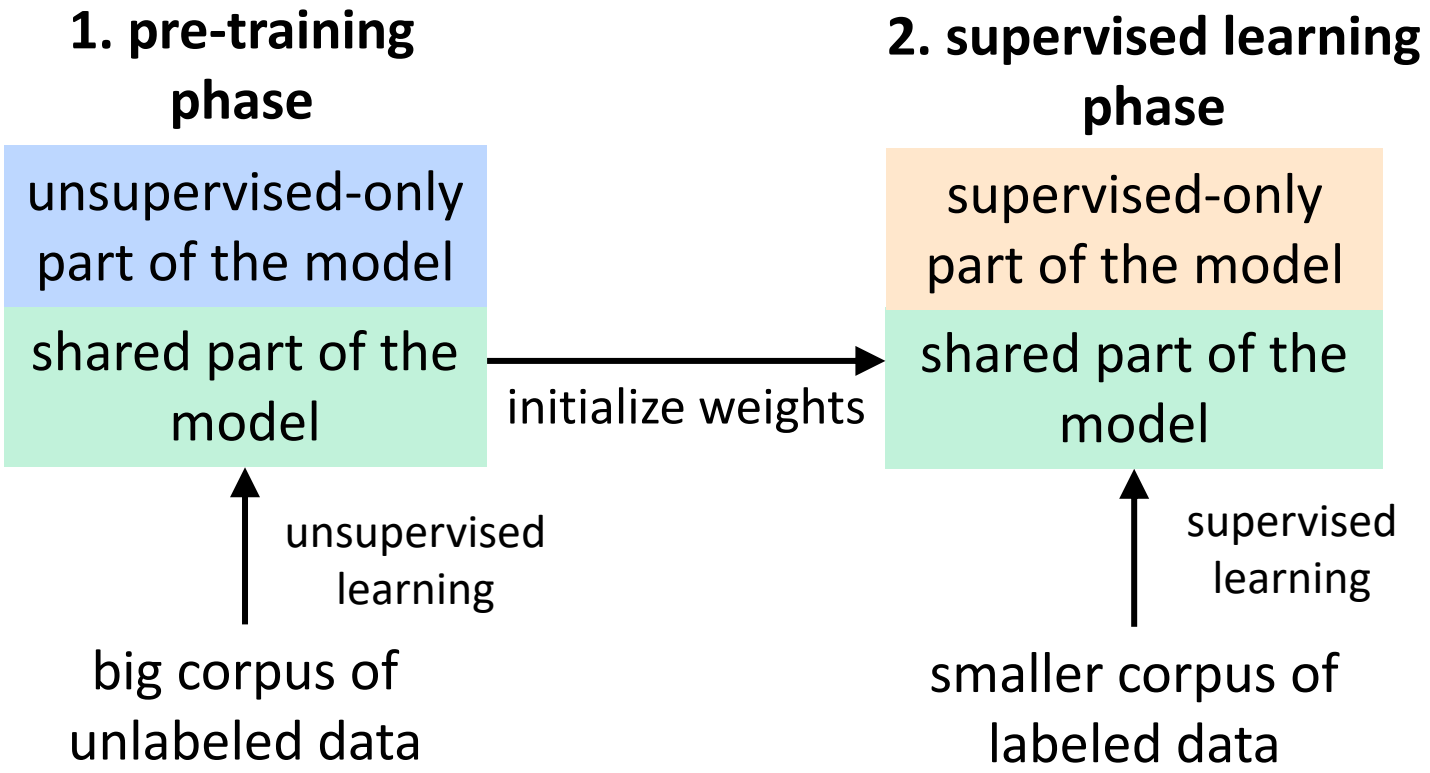
- But this of course is wrong, the model will make lots of mistakes

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

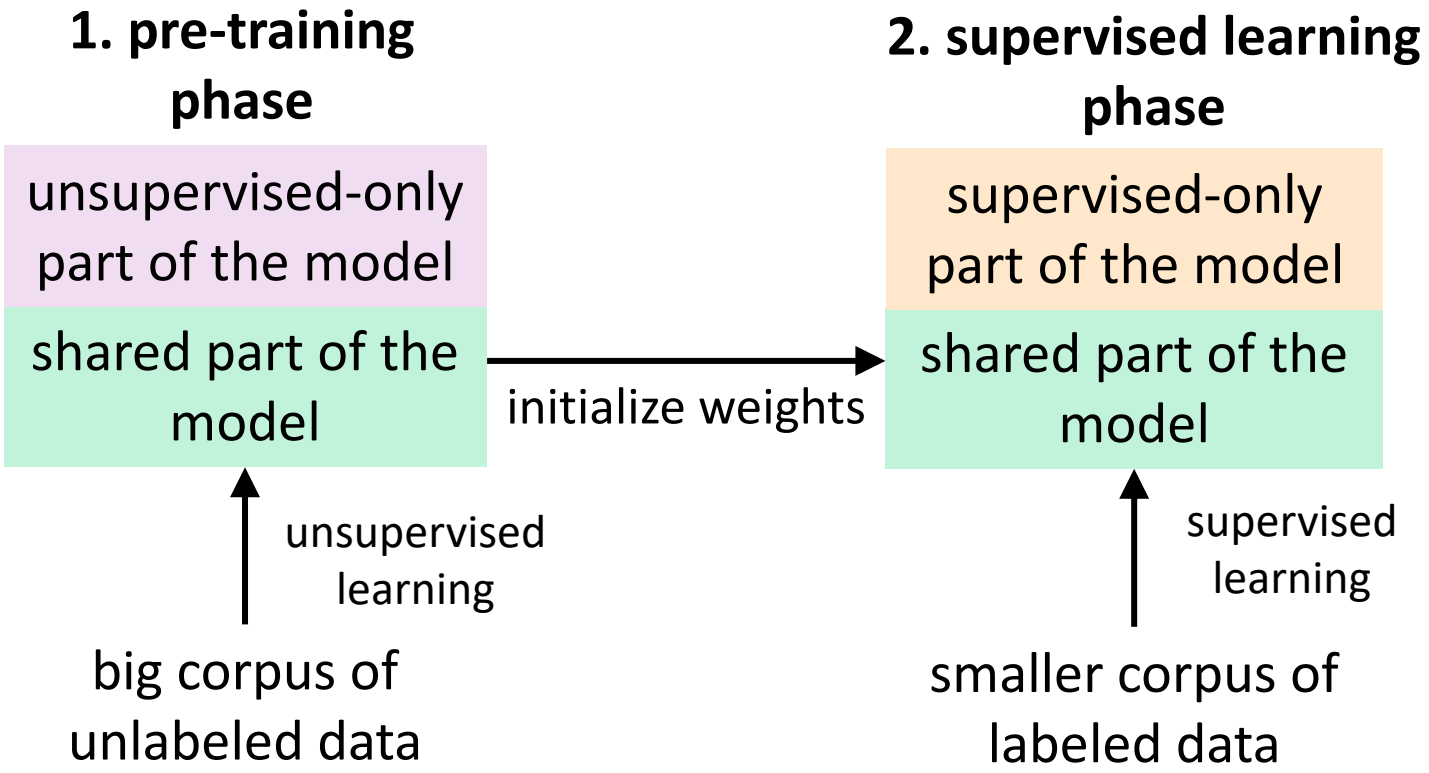
Pre-training

- 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.



Pre-training: Word2Vec

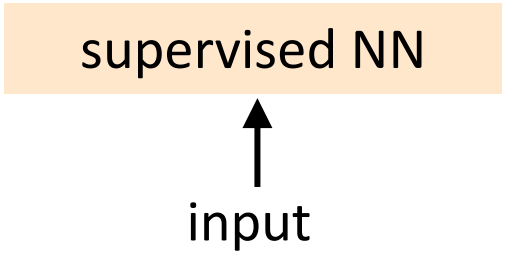
- Shared part is word embeddings
- No unsupervised-only part
- Supervised-only part is the rest of the model
- Unsupervised learning: skip-gram/cbow/glove/etc
- Supervised learning: training on some NLP task



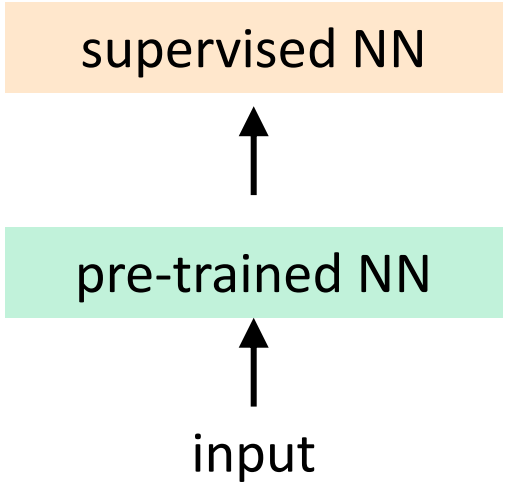
Why does pre-training work?

- ”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

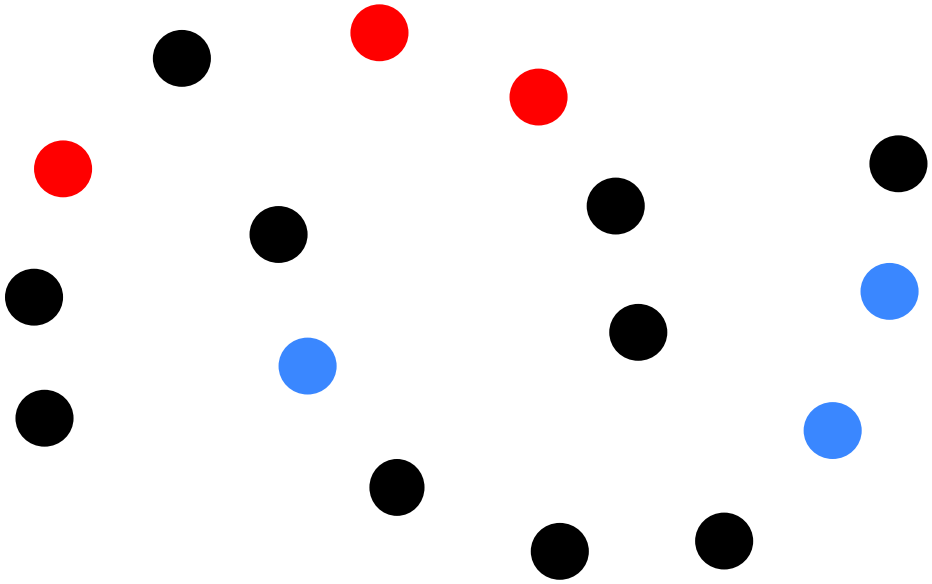


Pre-training: supervised part gets more useful representations as input

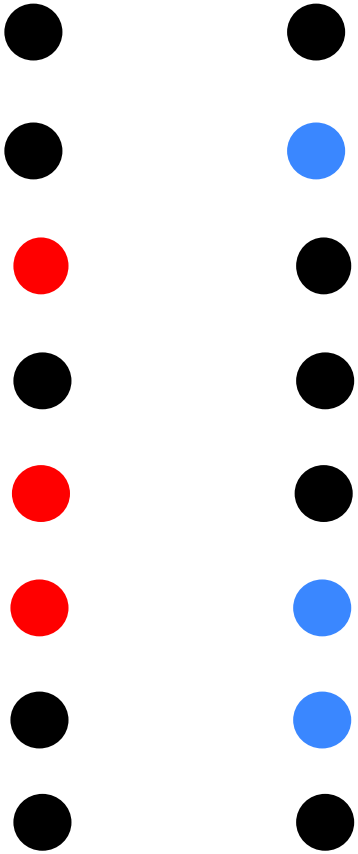


Why does pre-training work?

original representation space

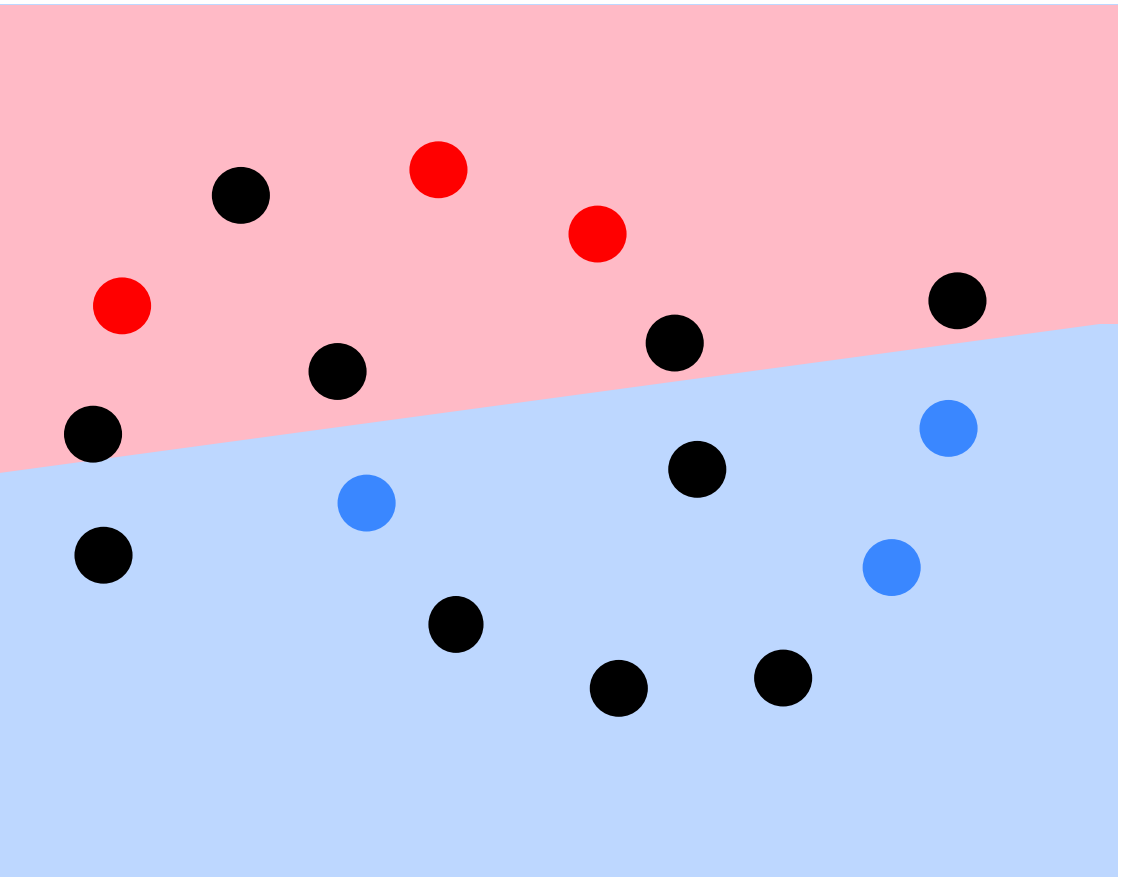


learned representation space after pre-training

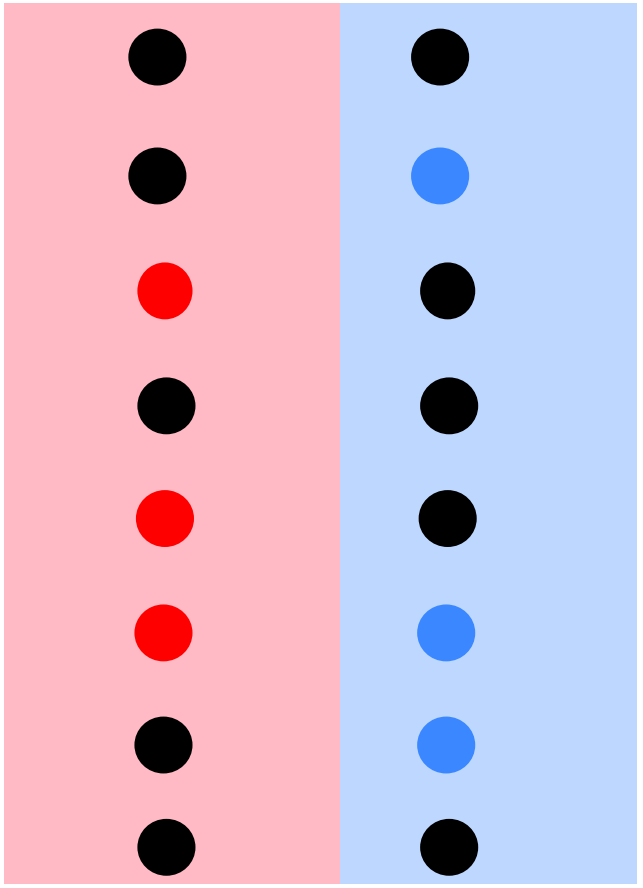


Why does pre-training work?

original representation space



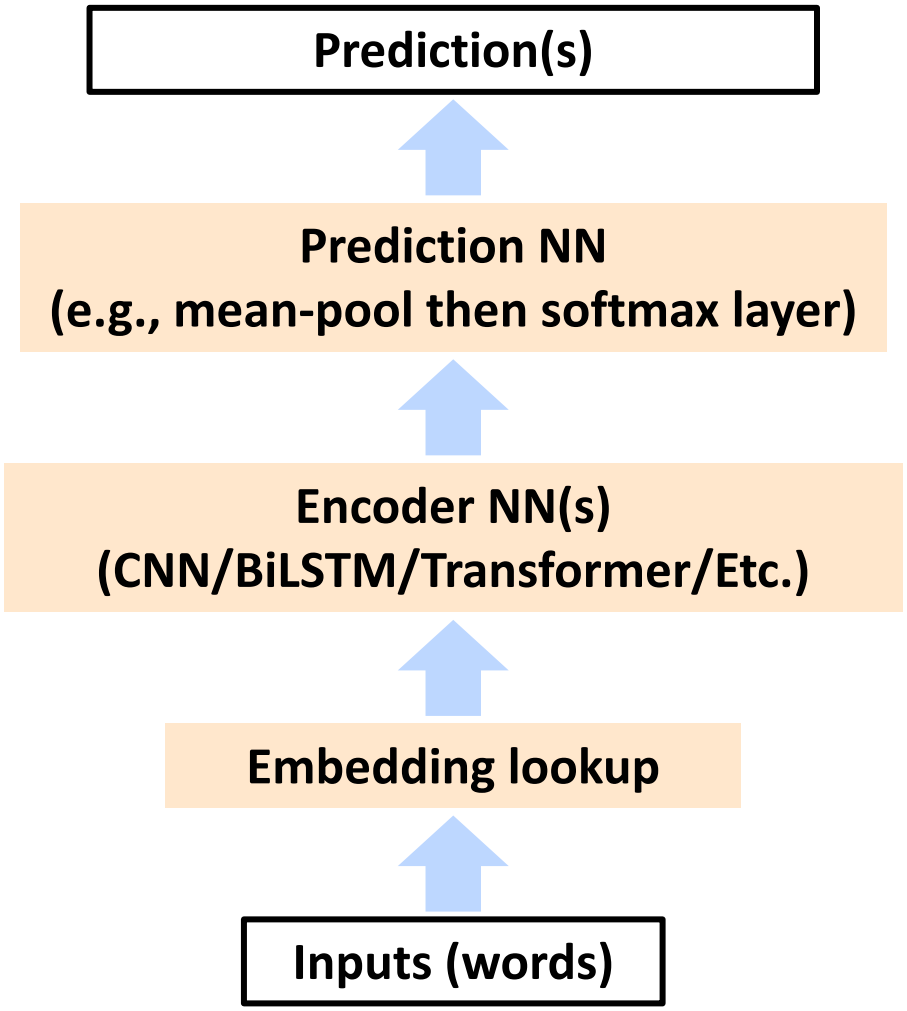
learned representation space after pre-training



Supervised part of the model has a much easier job after pre-training

Pre-Training for NLP

- Most neural NLP models looks (roughly) like this

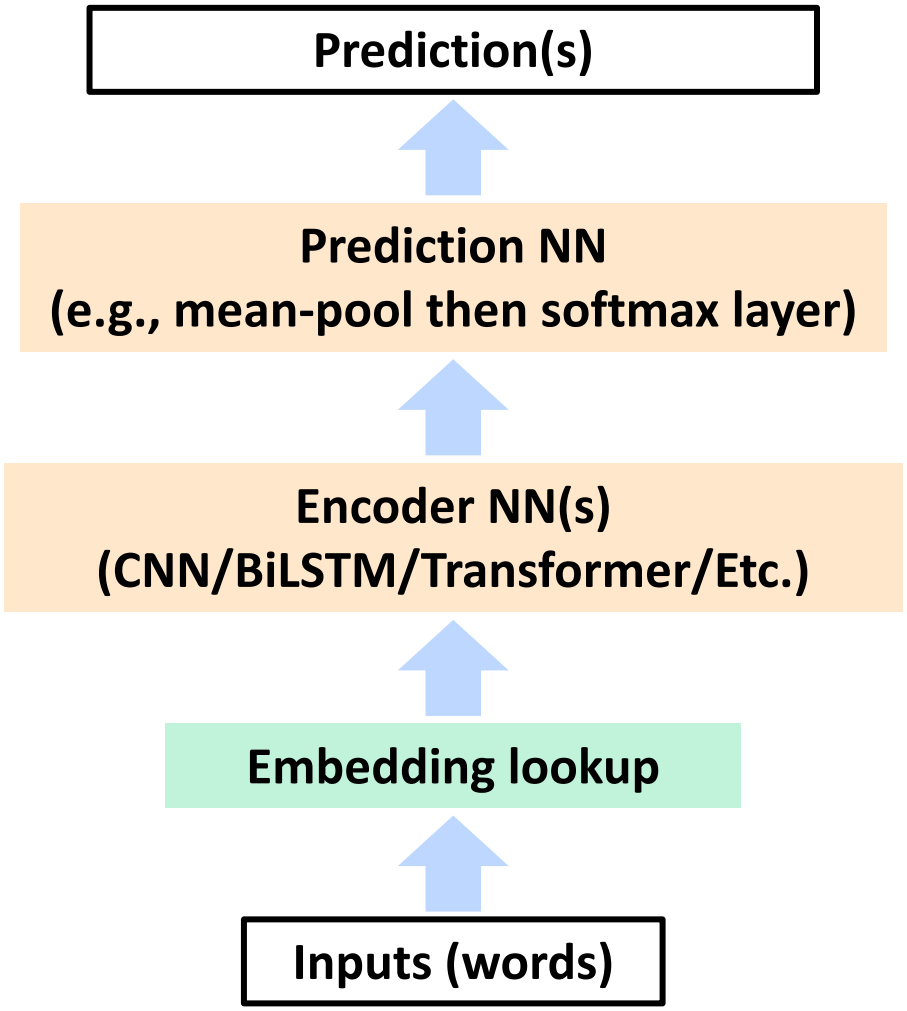


Pre-Training for NLP

- With pre-trained embeddings

Supervised

Pre-trained

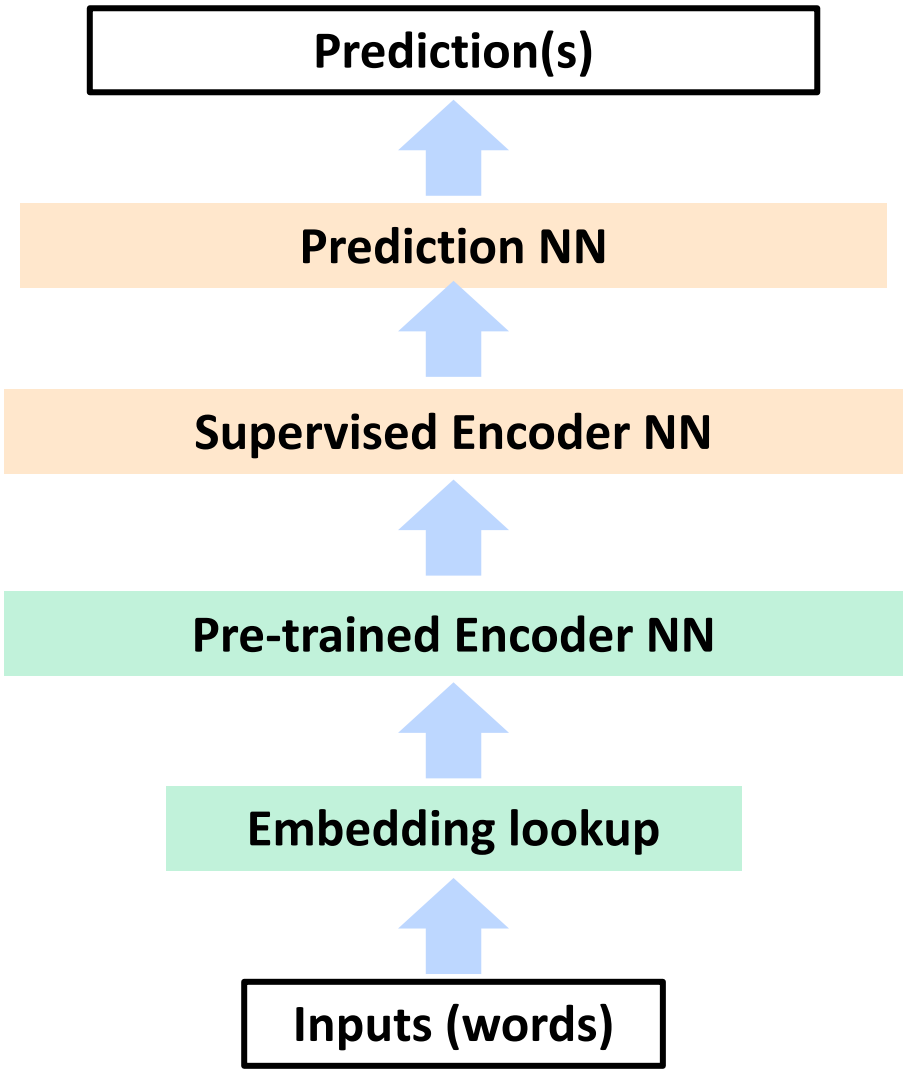


Pre-Training for NLP

- Recent research: pre-train more of the model (e.g., the first LSTM layer)

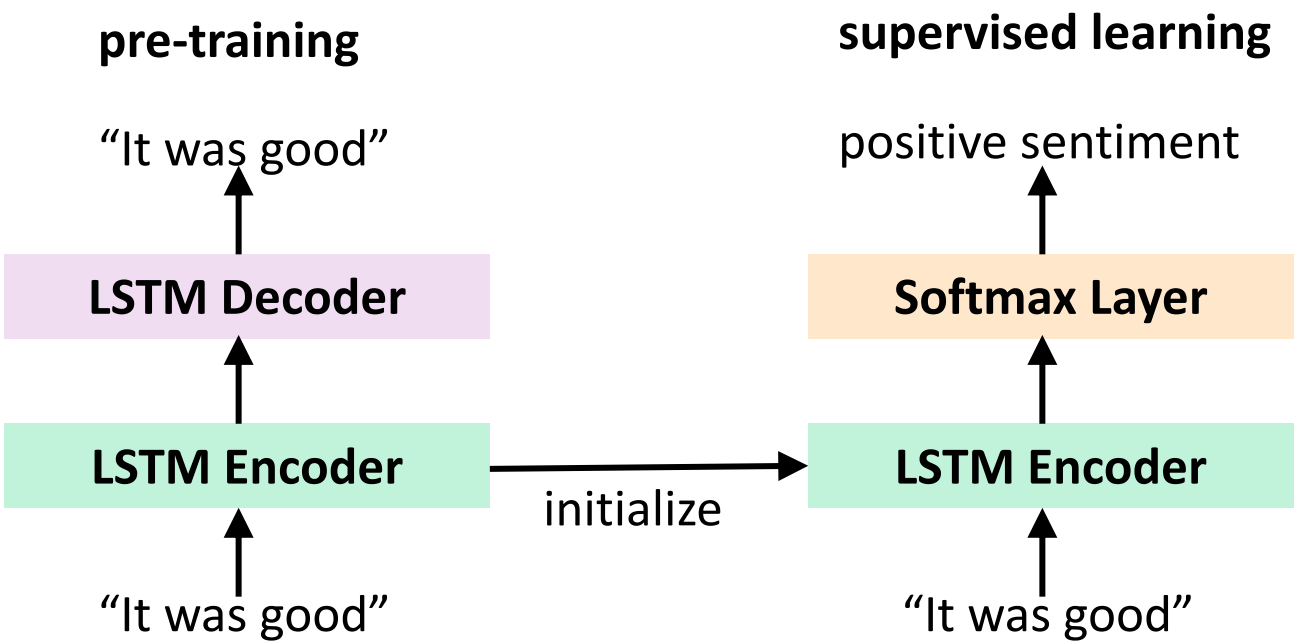
Supervised

Pre-trained



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



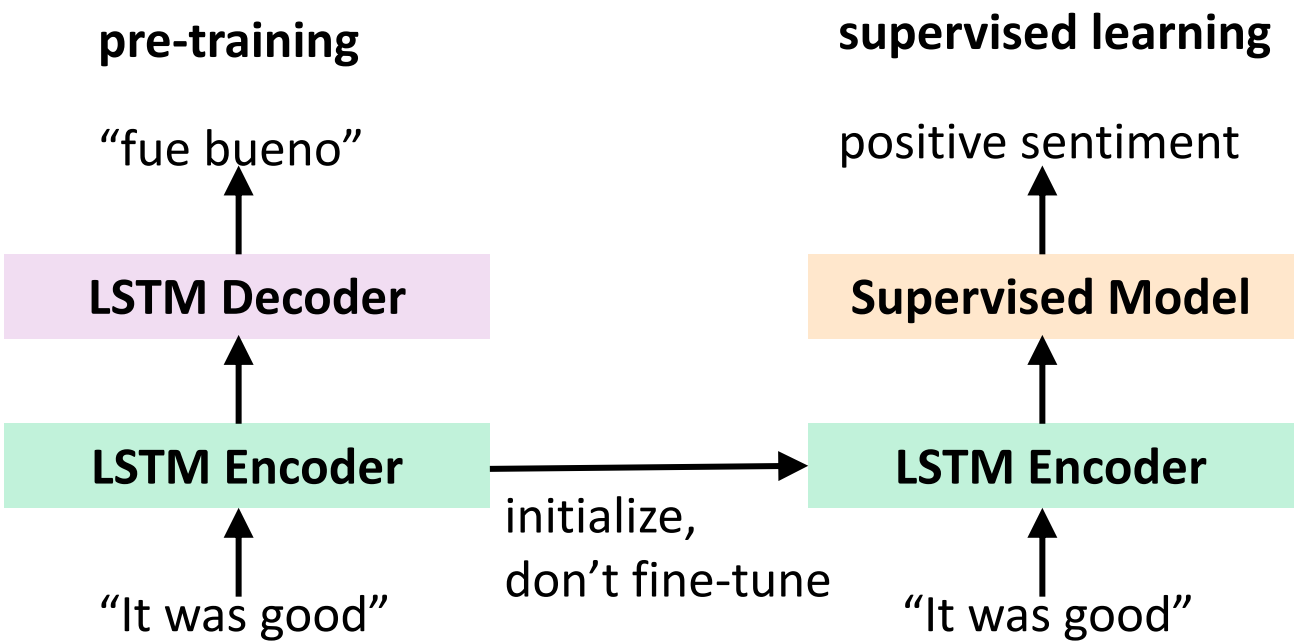
Pre-Training Strategies: Auto-Encoder (Dai & Le, 2015)

- 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

Dataset	Previous Best Result	Supervised Baseline	With Pretraining
IMDB	7.42	10.00	7.24
Rotten Tomatoes	18.5	20.5	16.7
20 Newsgroups	17.1	18.0	15.6

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



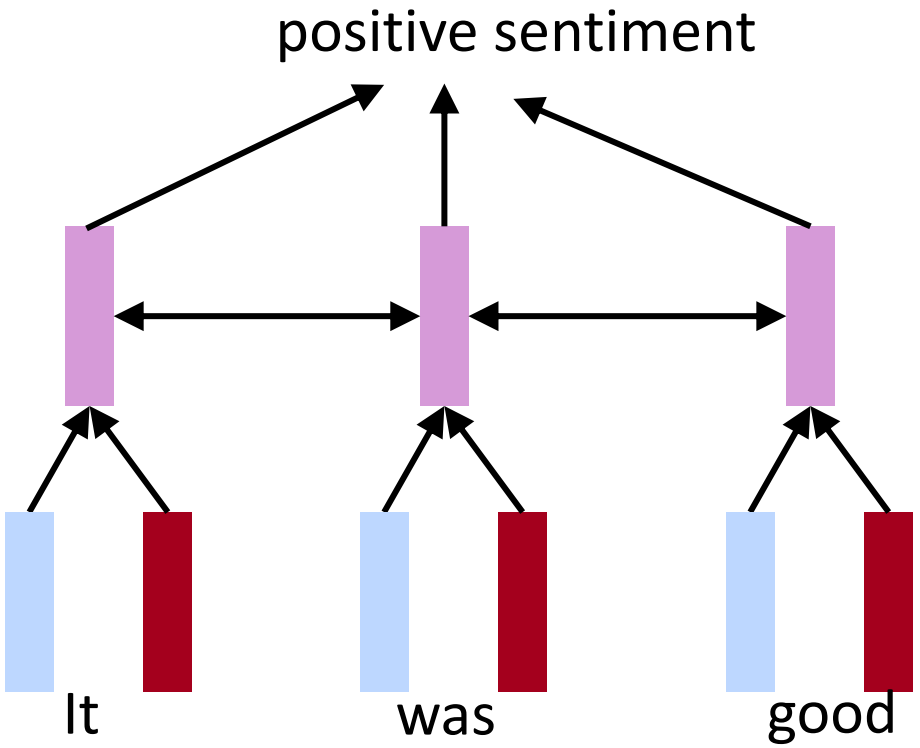
Pre-Training Strategies: CoVe (McCann et al., 2017)

- 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

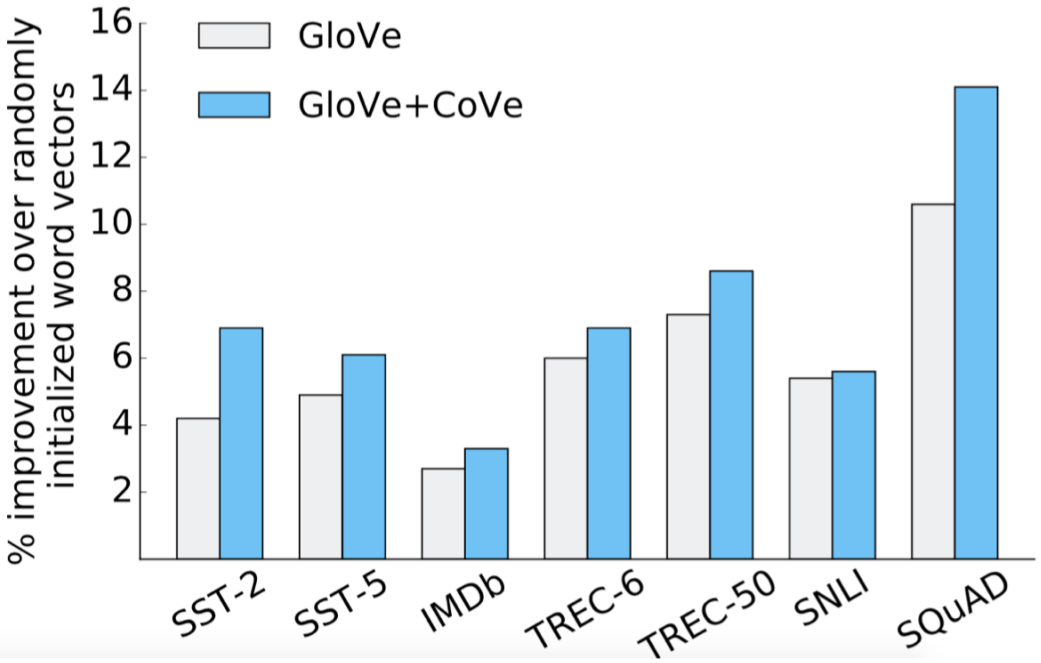
GloVe vectors

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



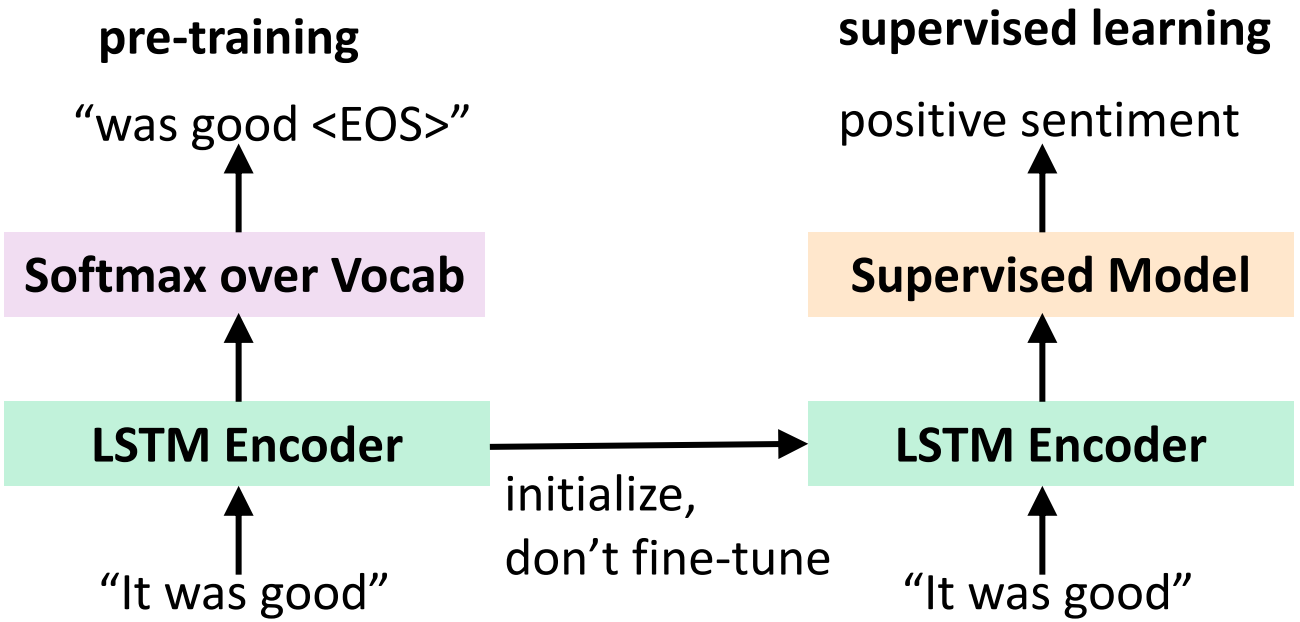
Pre-Training Strategies: CoVe (McCann et al., 2017)

- 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



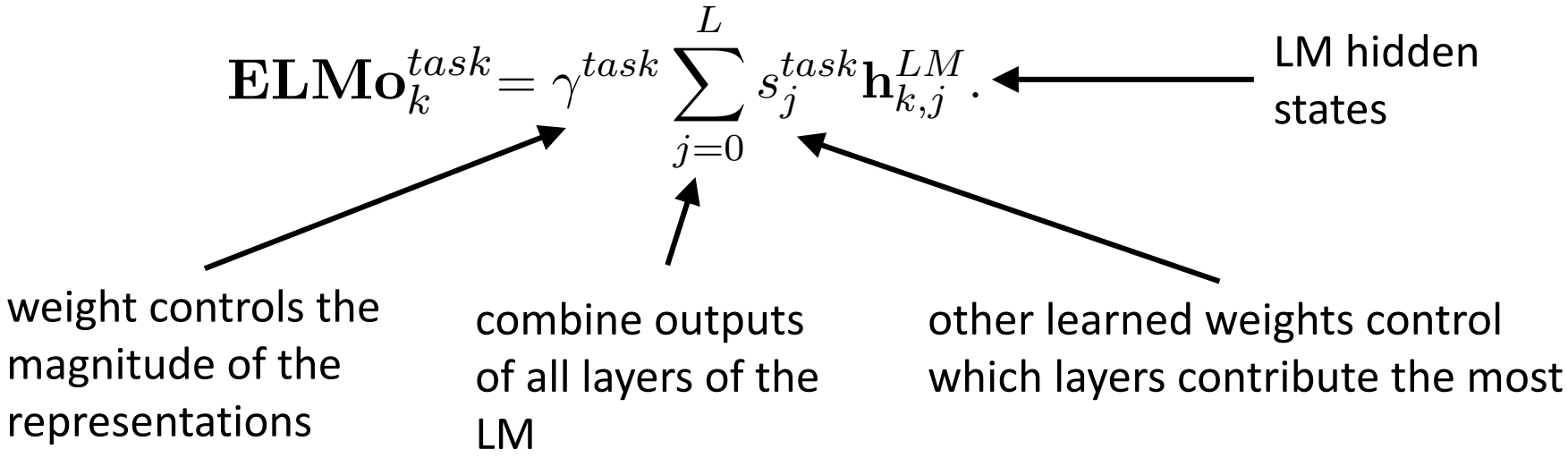
Pre-Training Strategies: ELMo (Peters et al., 2017)

- 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 Strategies: ELMo (Peters et al., 2018)

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



  - 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

TASK	PREVIOUS SOTA		OUR BASELINE	ELMO + BASELINE	INCREASE (ABSOLUTE/ RELATIVE)
SQuAD	SAN	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

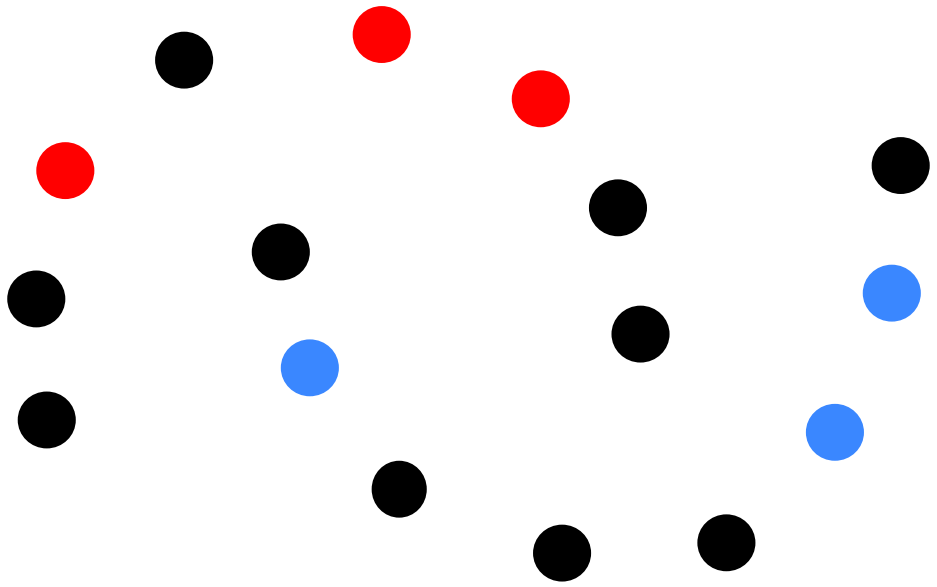
Pre-Training Overview

- 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

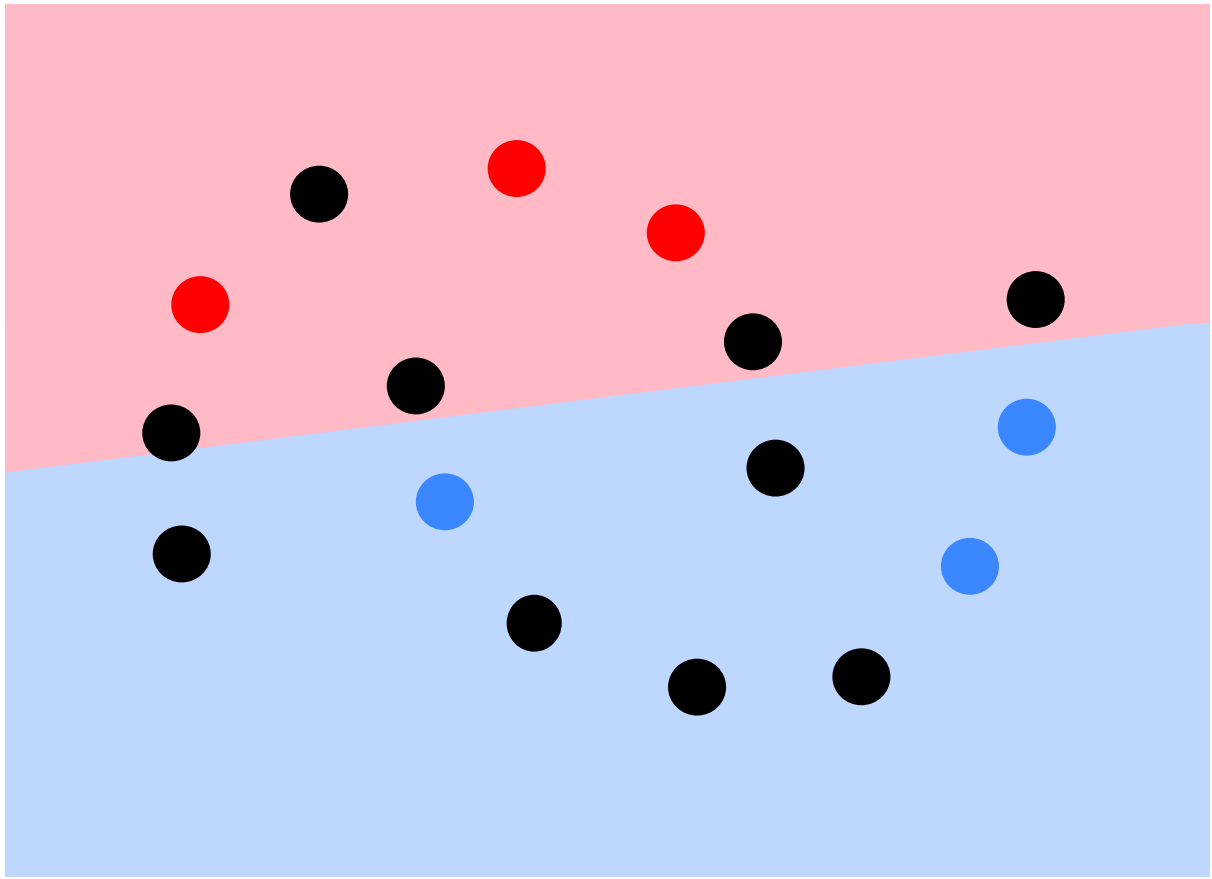
- Use unlabeled data without a giant model or long pretraining phase
- Old (1960s) and simple semi-supervised algorithm
- Algorithm:
  1. Train the model on the labeled data.
  2. Have the model label the unlabeled data.
     - Take some of examples the model is most confident about (i.e., the model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

Self-Training: Example



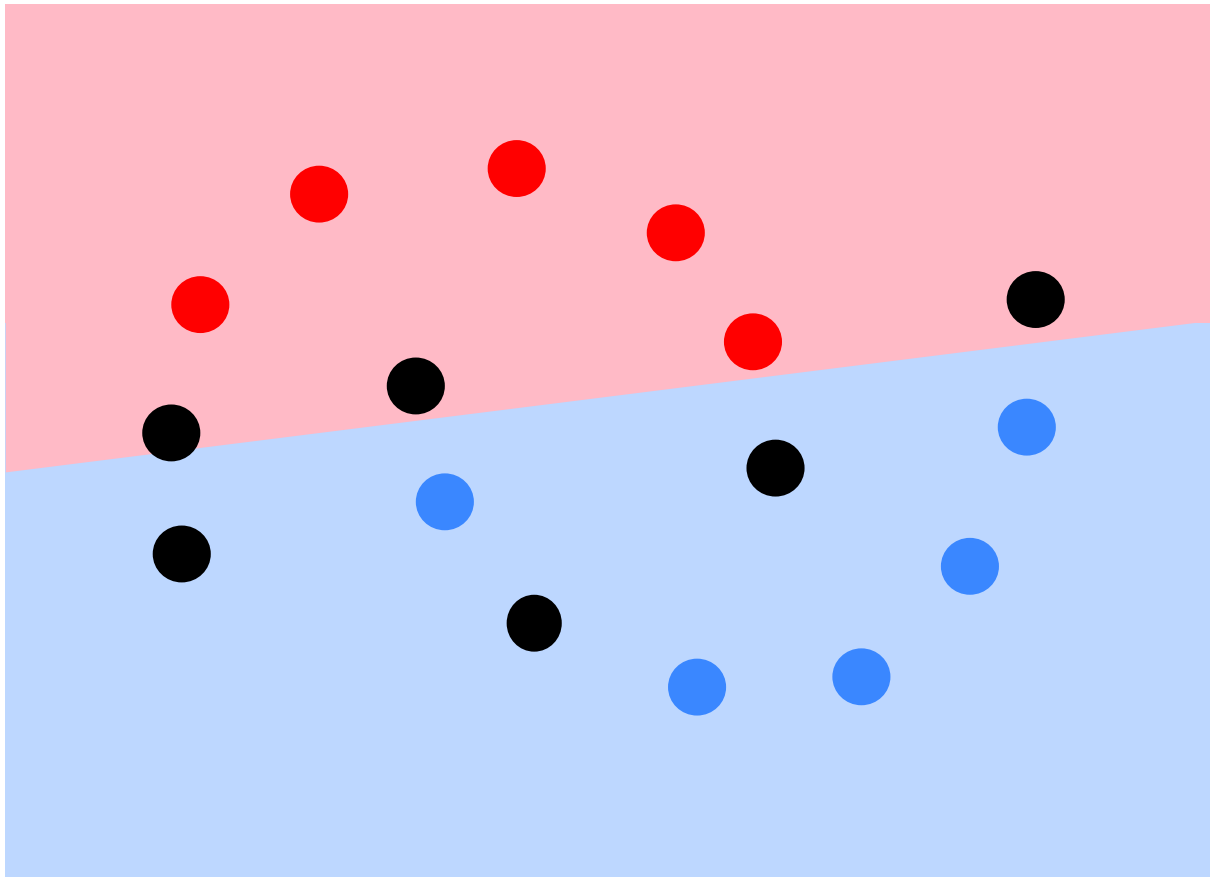
- 1. Train our model on the labeled data

Self-Training: Example



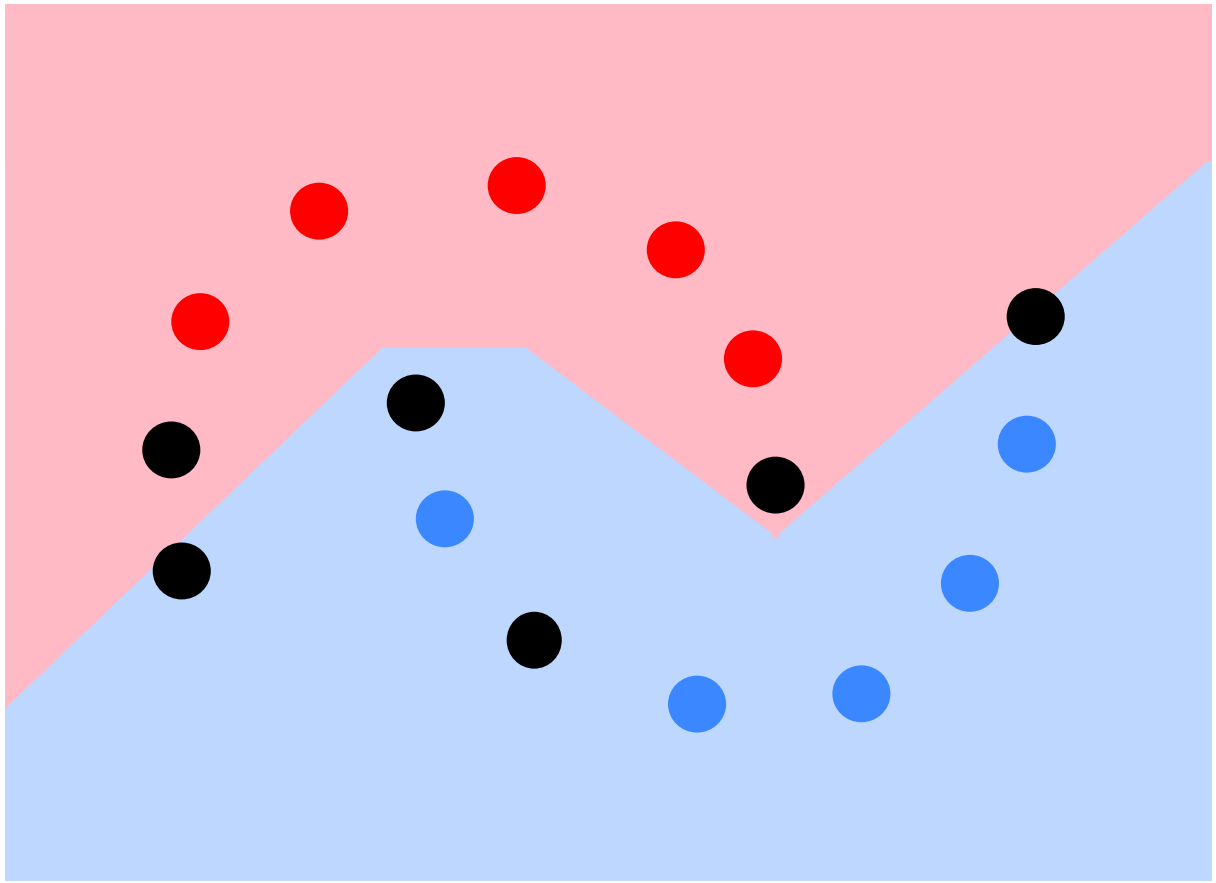
- 2. Label a few examples

Self-Training: Example



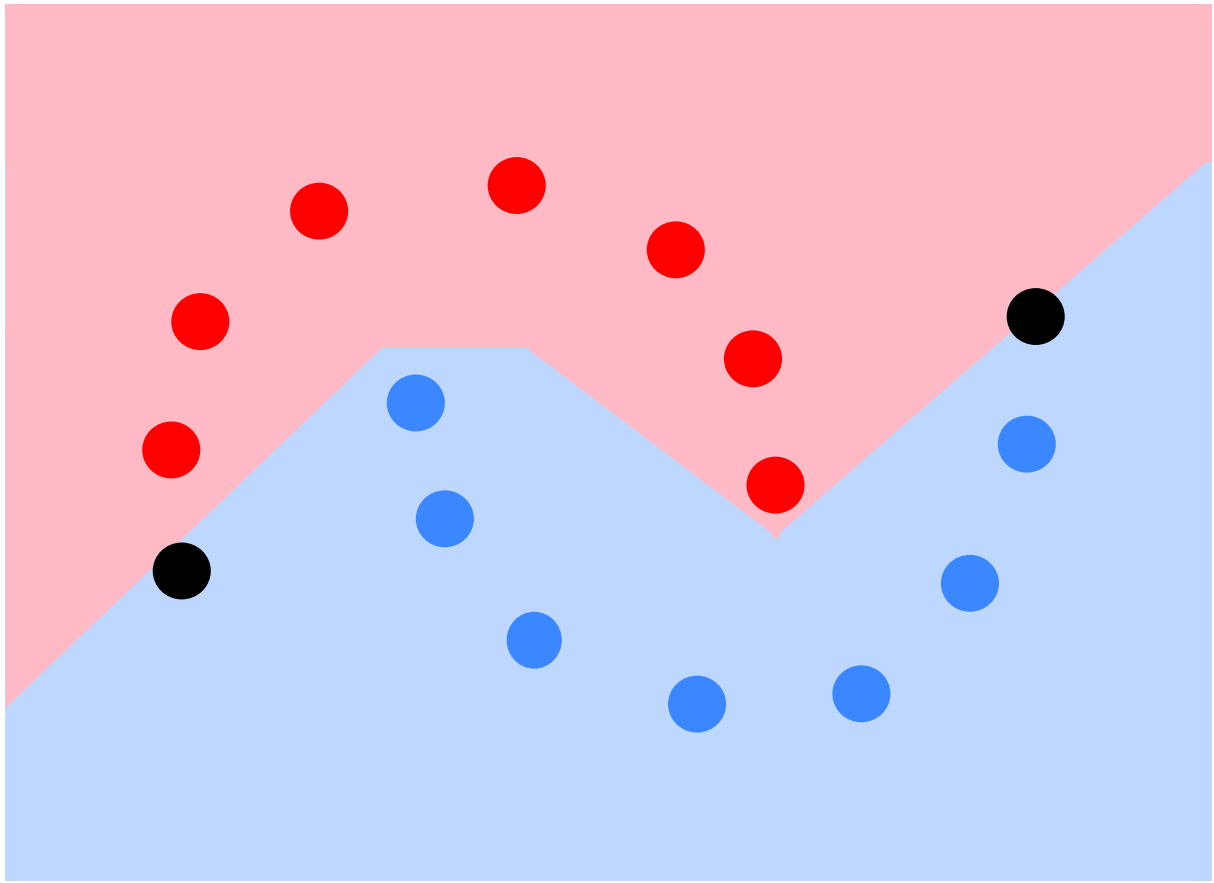
- 3. Re-train the model

Self-Training: Example

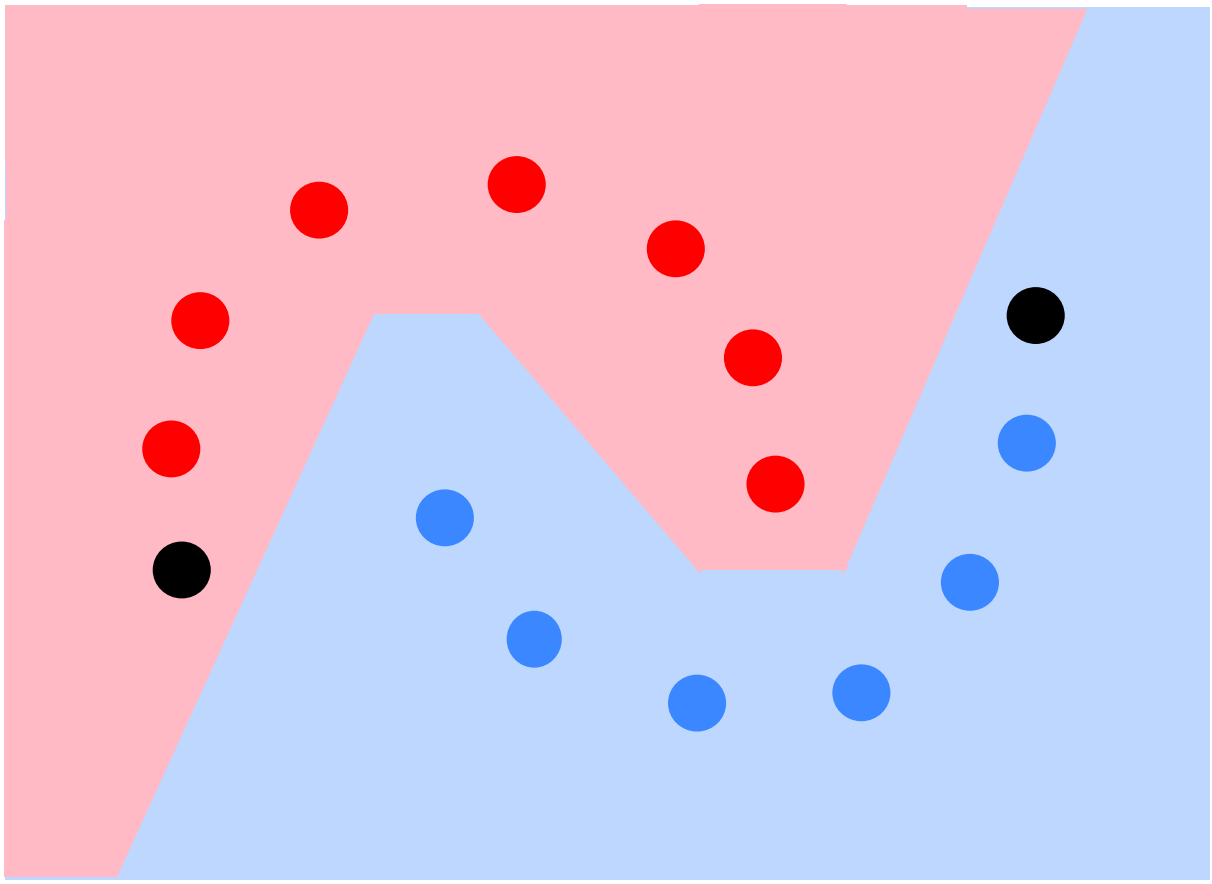


- Repeat!

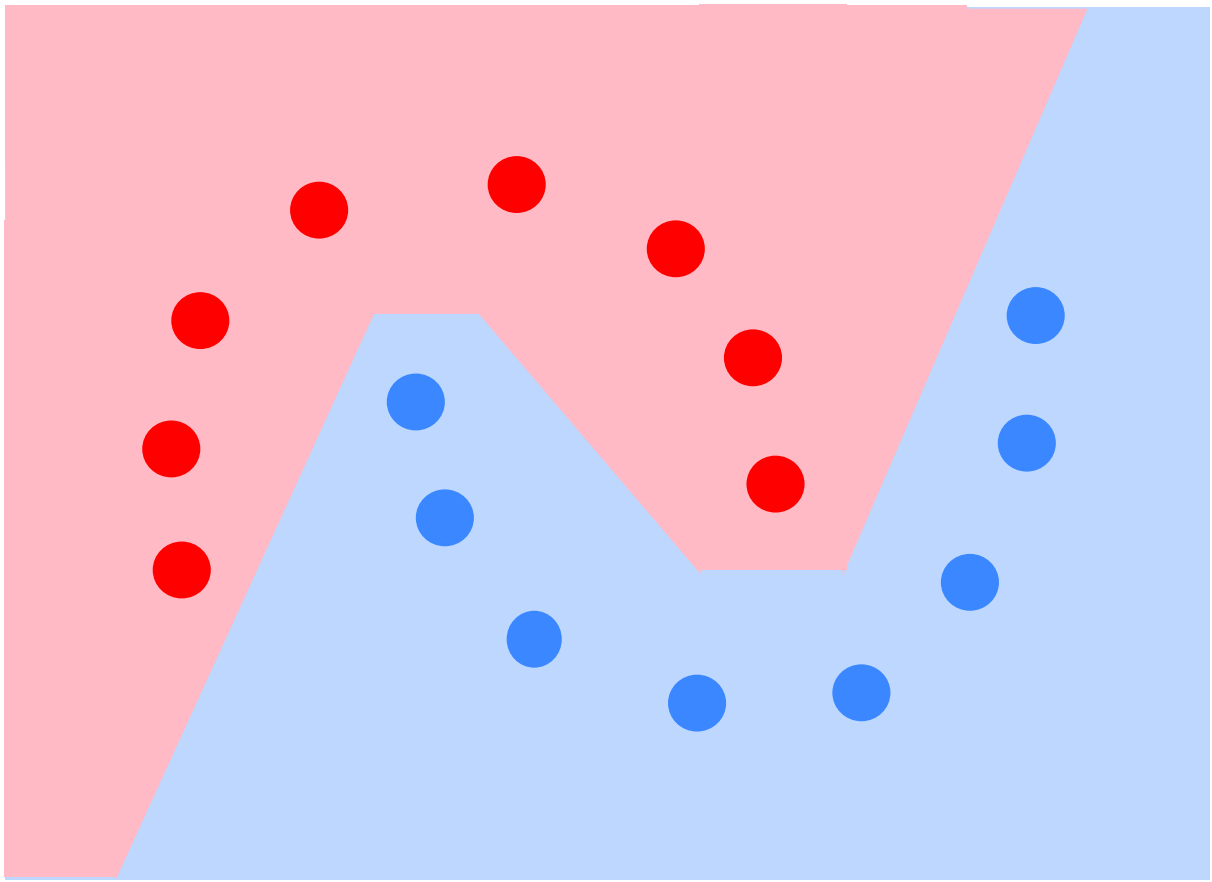
Self-Training: Example



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

Sentences added	1
0 (baseline)	91.8
50k	91.8
250k	91.8
500k	92.0
750k	92.0
1,000k	92.1
1,500k	92.1
1,750k	92.1
2,000k	92.2

Parsing F1 score vs amount of unlabeled data

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

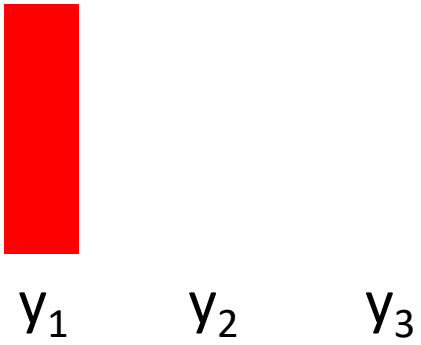
Target is a model-produced label. It will be noisy because the model isn't as accurate as a person, but hopefully the model can still learn from it

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

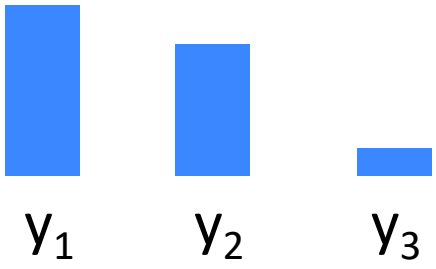
Rest of this lecture: how do we make this second term (the unsupervised one) better?

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 use 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?

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

$$\downarrow$$

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

  - A bit odd: the student already matches the targets!

Consistency Regularization

- Add noise to the student's inputs

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

Soft target

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

  - Where η is a vector with a random direction and a small magnitude ϵ

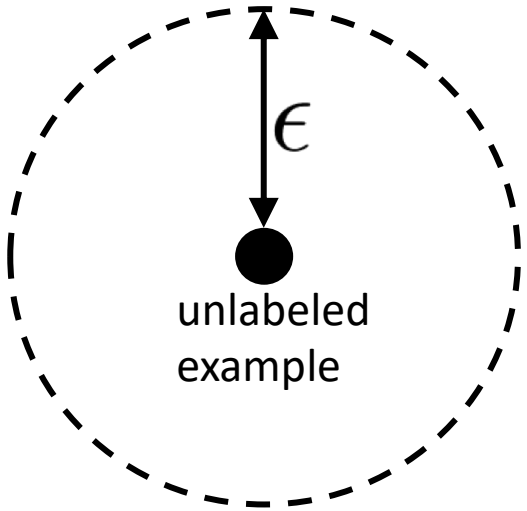
Consistency Regularization

- Add noise to the student's inputs

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

  - Where η is a vector with a random direction and a small magnitude ϵ

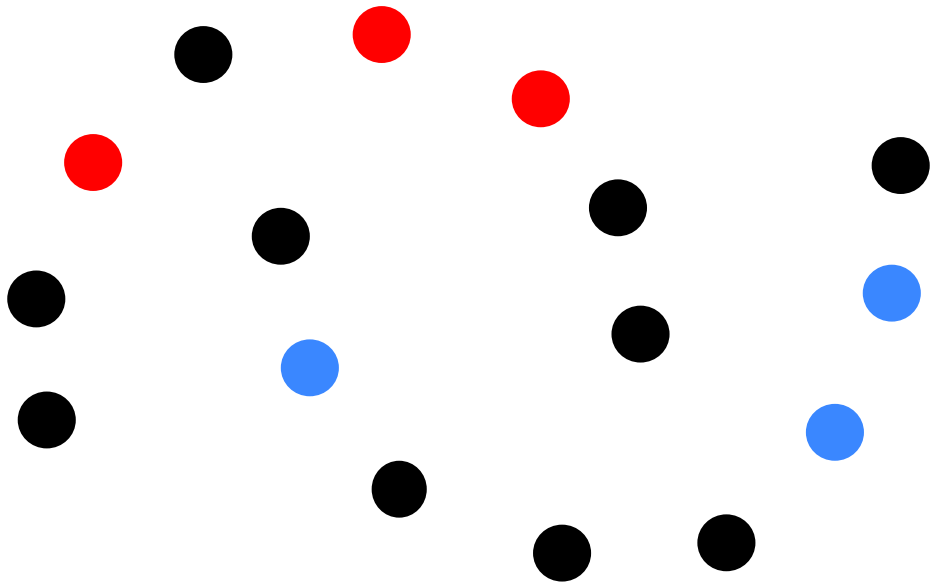
- Train the model so a bit of noise doesn't mess up its predictions
- Equivalently, the model must give consistent predictions to nearby data points



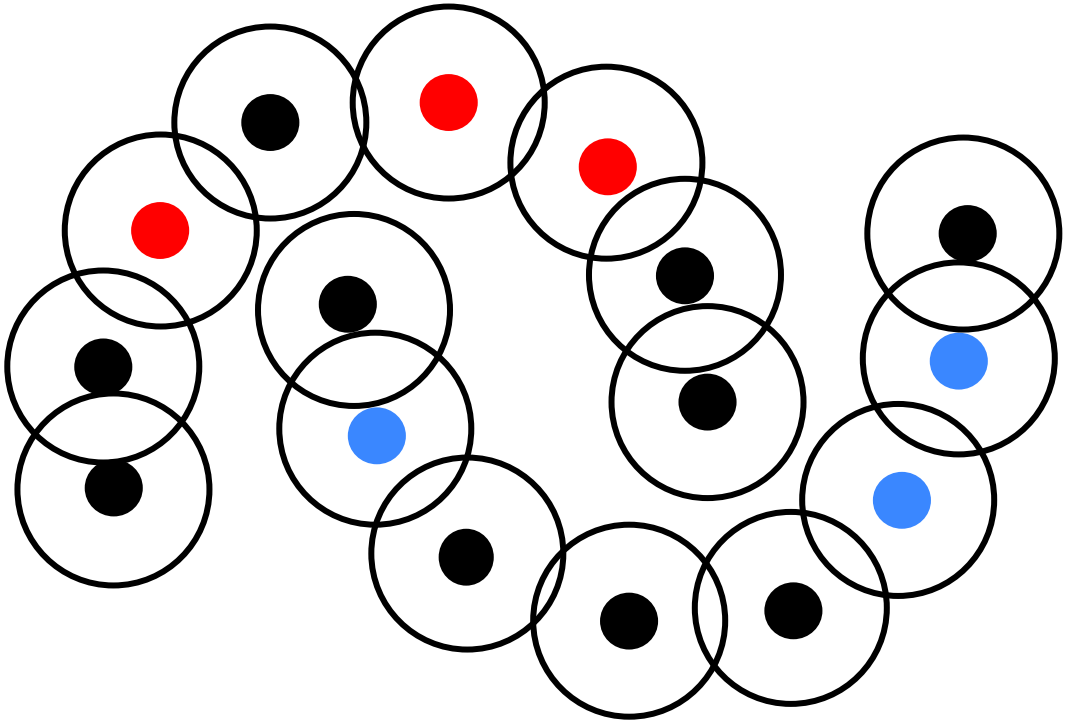
The model is trained to give the same prediction for any point in the circle

"distributional smoothing"

Consistency Regularization: Example

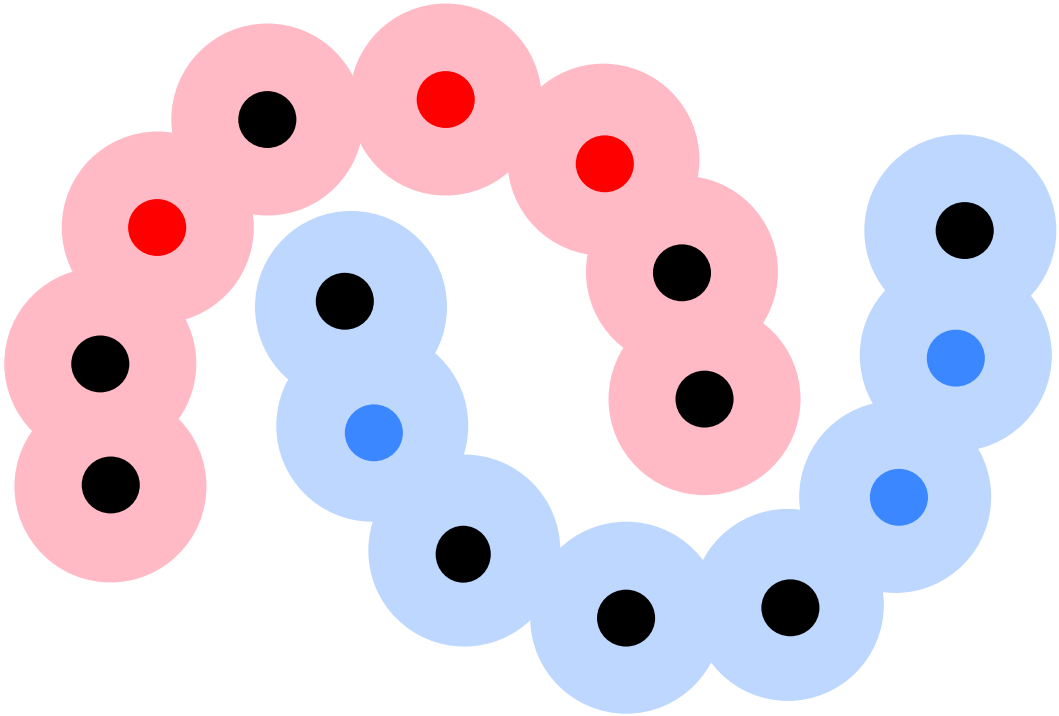


Consistency Regularization: Example



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

Model	Error Rate
Supervised	35.56
Ladder Netwosk (Rasmus et al, 2015)	20.40
CatGAN (Springenberger, 2016)	19.58
GAN (Salimans et al., 2016)	18.63
Consistency Regularization (Sajjadi et al., 2016)	**11.29**

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

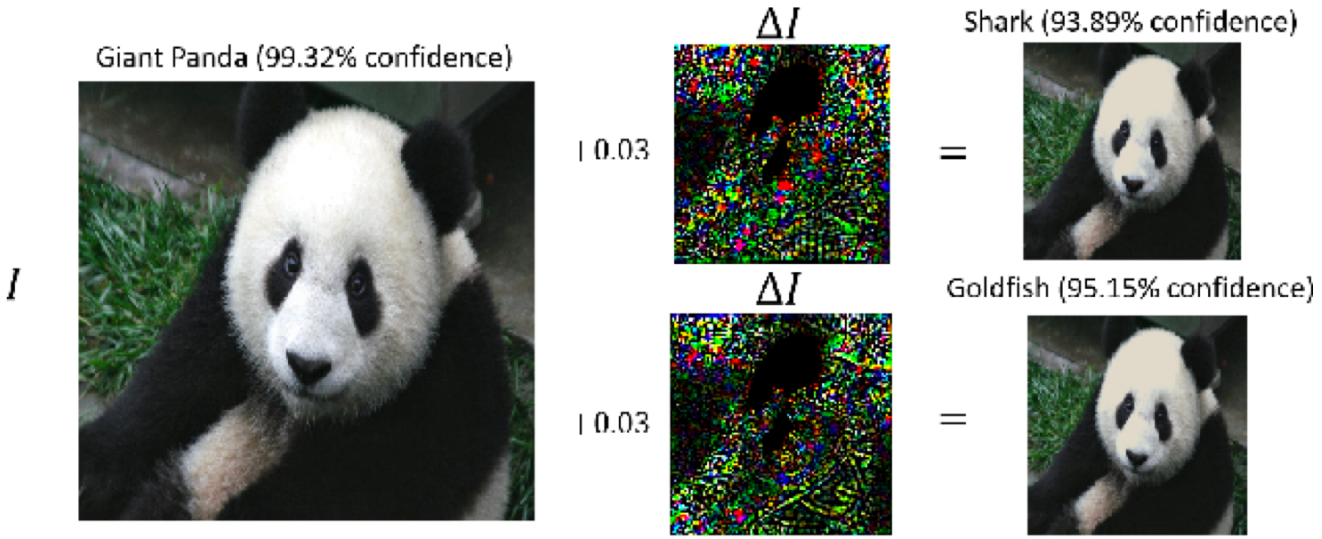
Virtual Adversarial Training (Miyato et al., 2017)

- 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**

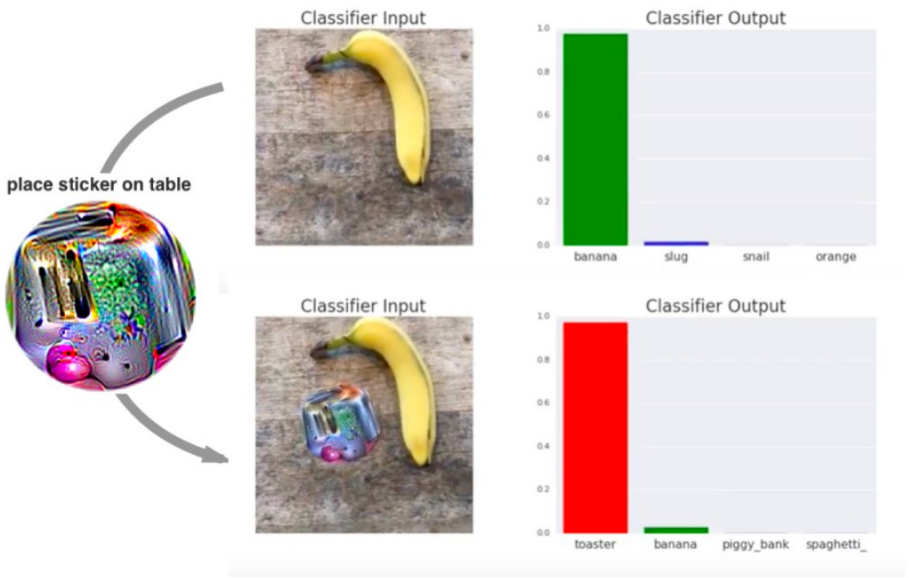
Adversarial Examples

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



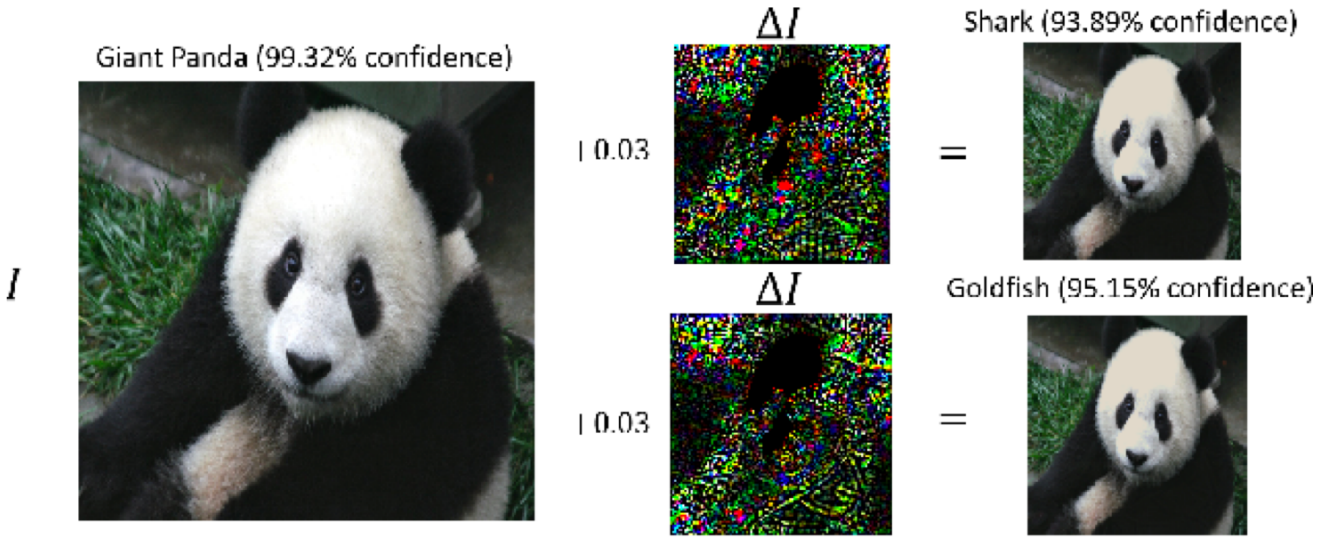
Adversarial Examples

- Security implications



Adversarial Examples

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



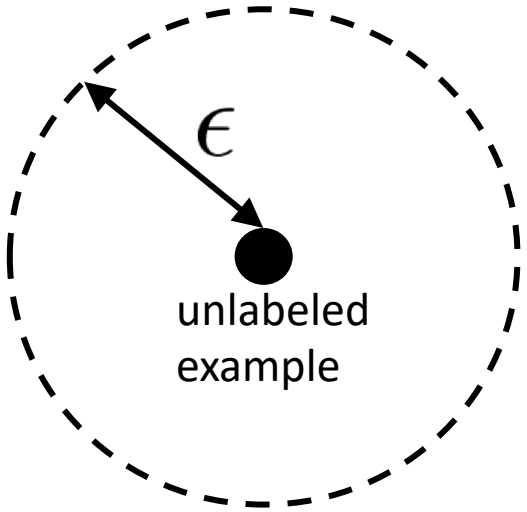
- **Creating an adversarial example:**
  - Compute the gradient of the loss with respect to the input
  - Add epsilon times the gradient to the input
    - Possibly repeat multiple times

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 η, pick the one that most increases the loss



Virtual Adversarial Training: Results

- Results on sentiment classification task

Model	Error
Pretraining Only	7.33
Pretraining + consistency reg. (random pertrubation)	6.78
Pretraining + consistency reg. (adversarial pertrubation)	5.91

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 `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 4*n* times as slow?

Cross-View Consistency (Clark et al., 2018)

- Sounds nice, but wouldn't this train 4*n* times as slow?

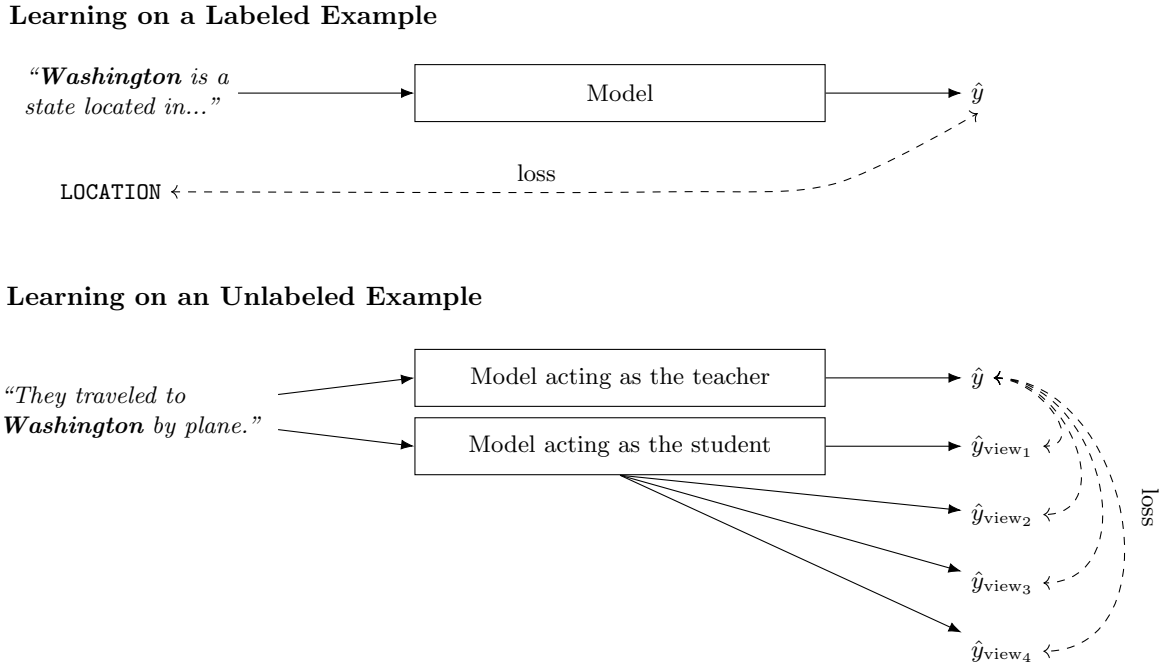
- Instead of running full the model multiple times, add multiple "auxiliary" softmax layers to the model
  - e.g., add one to the forward LSTM in the first BiLSTM layer. It doesn't see any words to the right of the current one
  - Trains these predictions to match the "primary" prediction from a softmax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

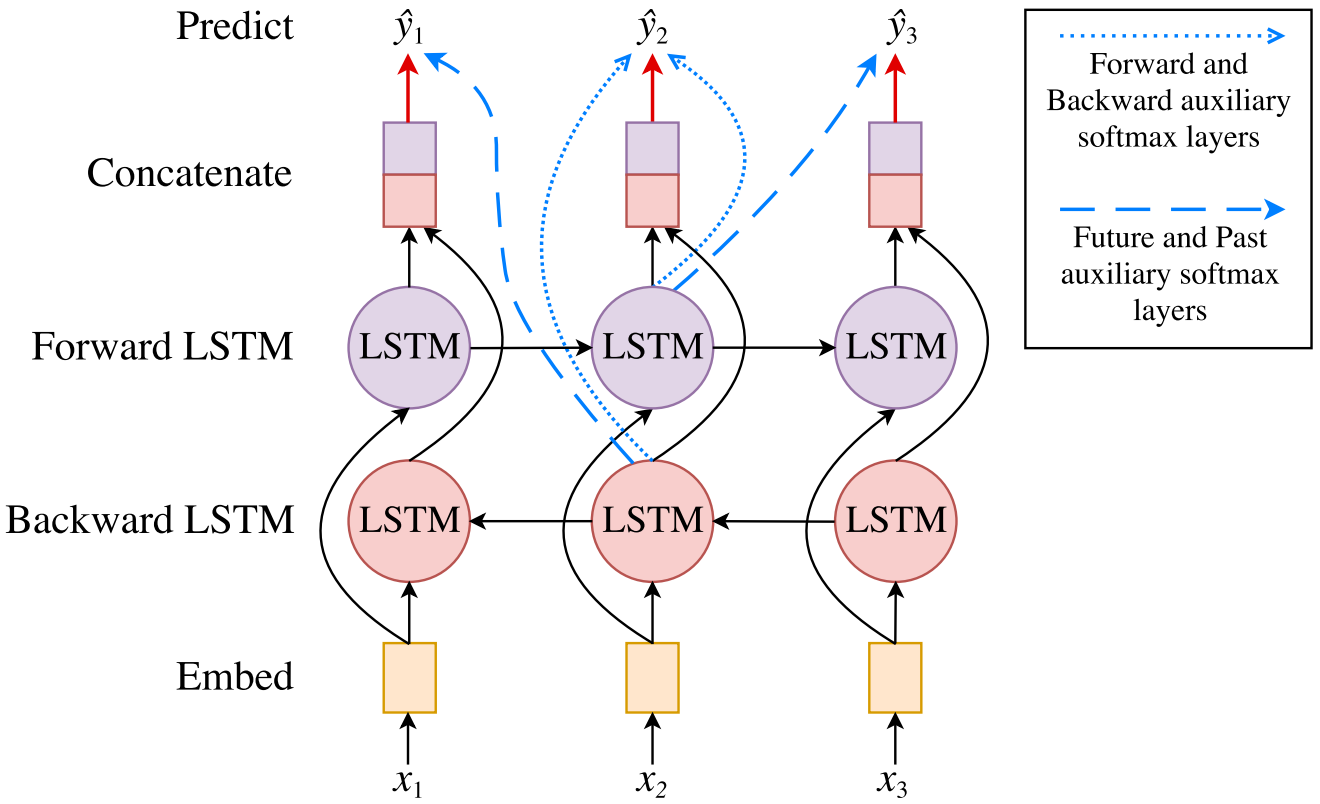
Auxiliary softmax layers that see restricted views of the input

Cross-View Consistency (Clark et al., 2018)



- 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"

Auxilliary Prediction Layers for Sequence Tagging

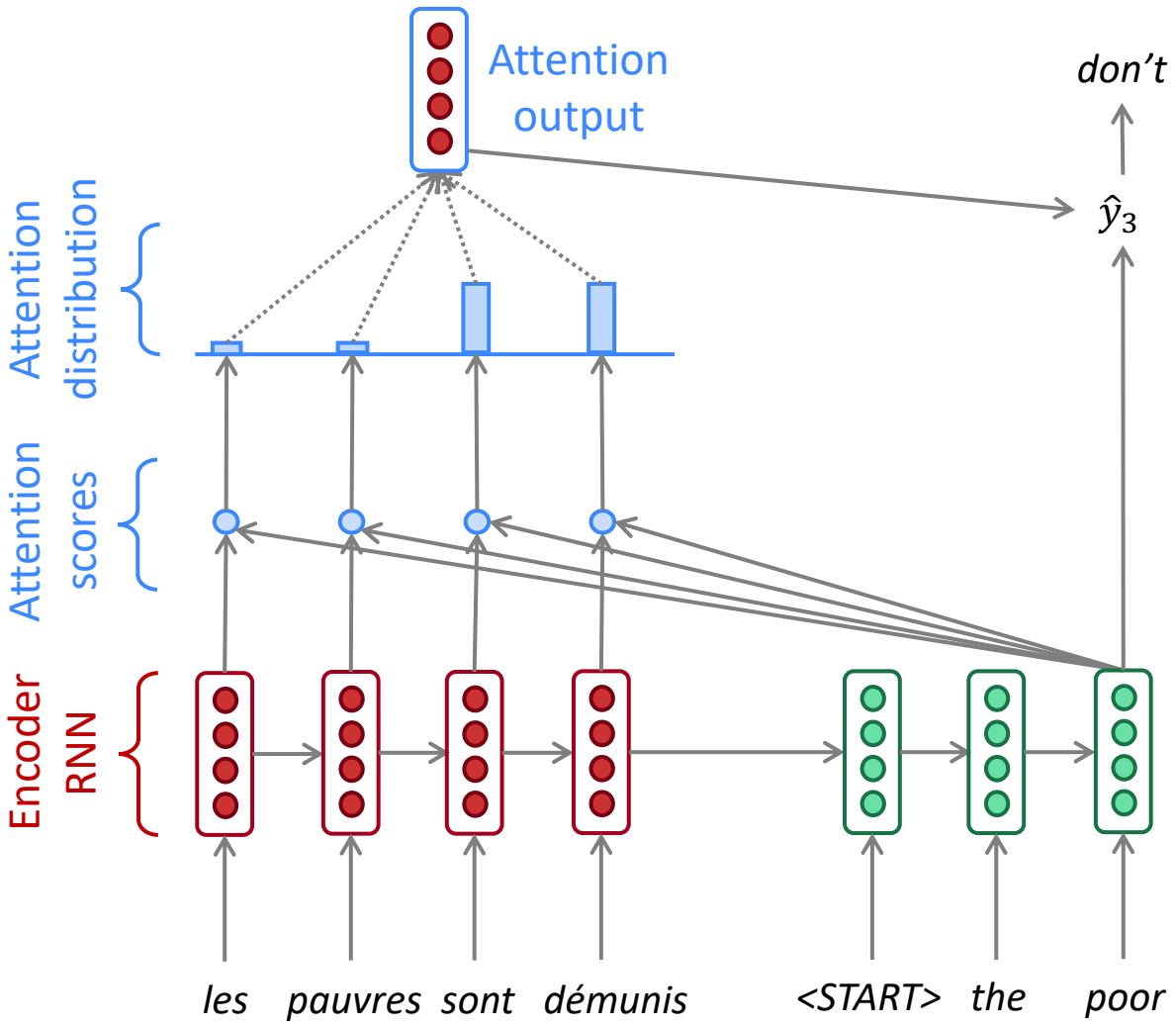


- **Forward**: attached to forward LSTM, produces predictions without seeing the right context of the current token.
- **Future**: attached to forward LSTM, produces prediction without seeing the right context or the current token itself

Auxiliary Prediction Layers for Machine Translation

Model: standard seq2seq with attention

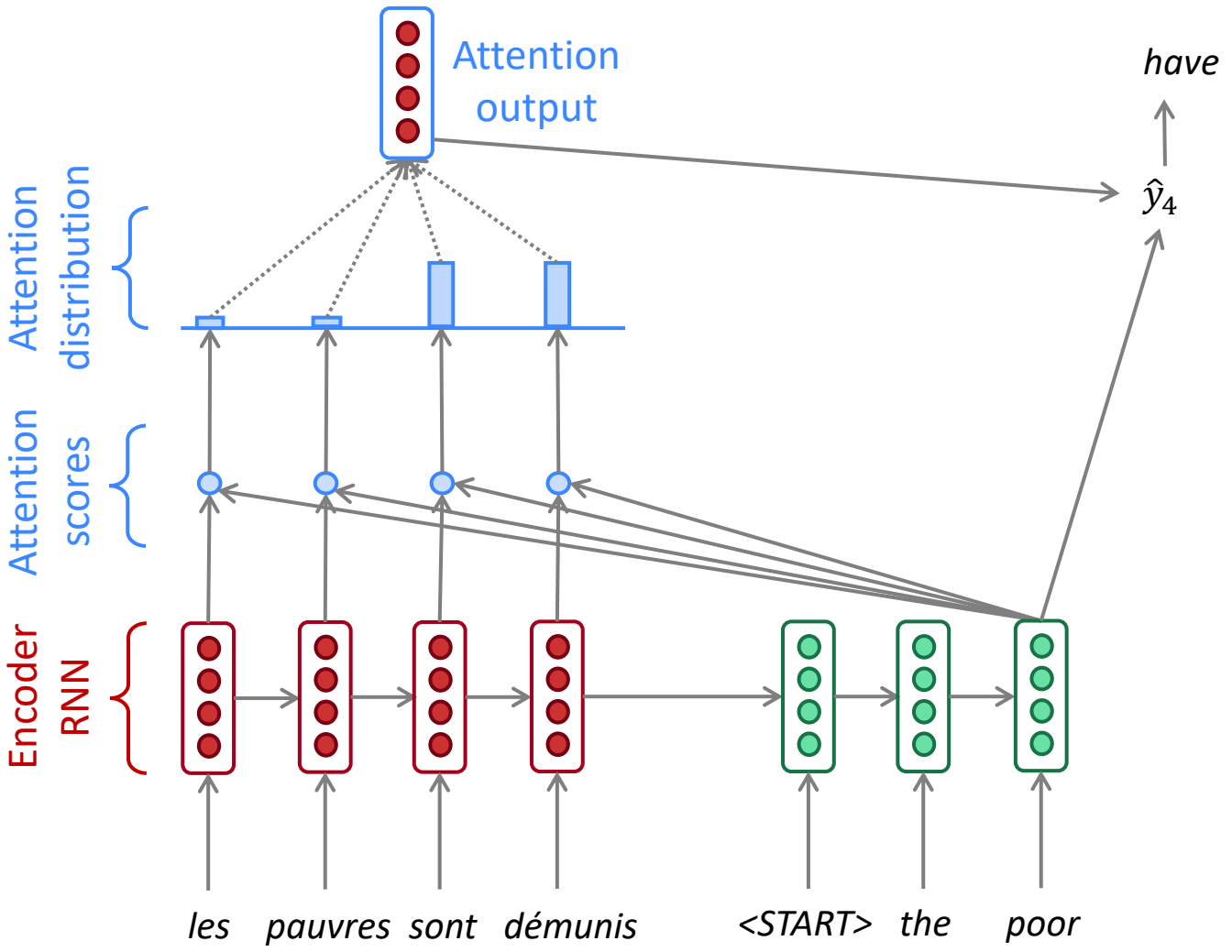
Auxiliary layers use the same LSTM but different attention and softmax weights



Auxiliary Prediction Layers for Machine Translation

Model: standard seq2seq with attention

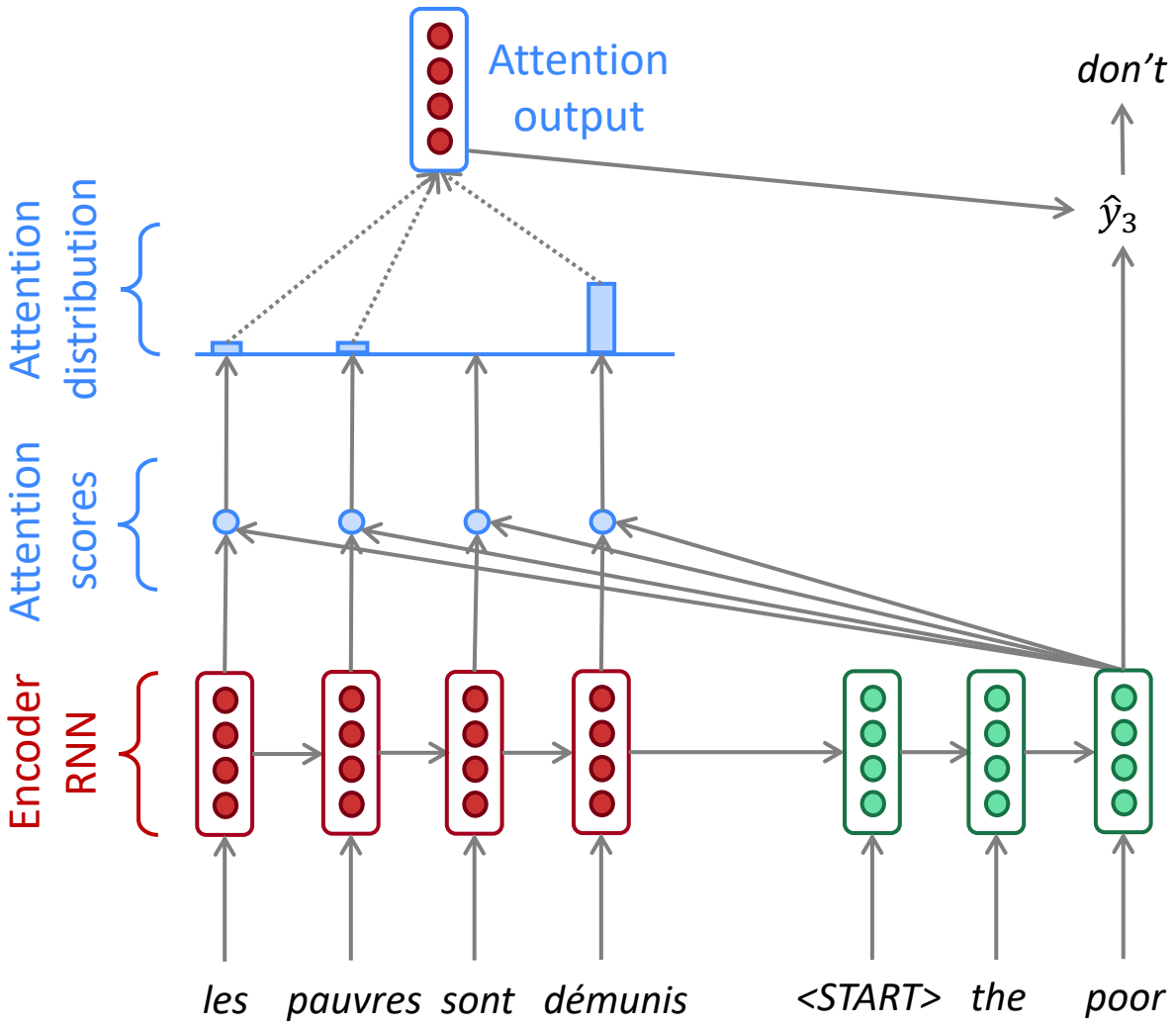
Auxiliary prediction layer 1: predict the word after next



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



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 layerr 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

Method	CCG	Chunk	NER	Dep. Parsing	MT (English-Vietnamese)
Previous state-of-the-art	**95.1**	**96.4**	**92.2**	**94.1**	26.1

- Chunking and NER results are using ELMo, rest are from supervised classifiers

Cross View Consistency: Results

Method	CCG	Chunk	NER	Dep. Parsing	MT (English-Vietnamese)
Previous state-of-the-art	**95.1**	**96.4**	**92.2**	**94.1**	26.1
Supervised	94.8	94.9	91.2	93.3	**28.9**

Cross View Consistency: Results

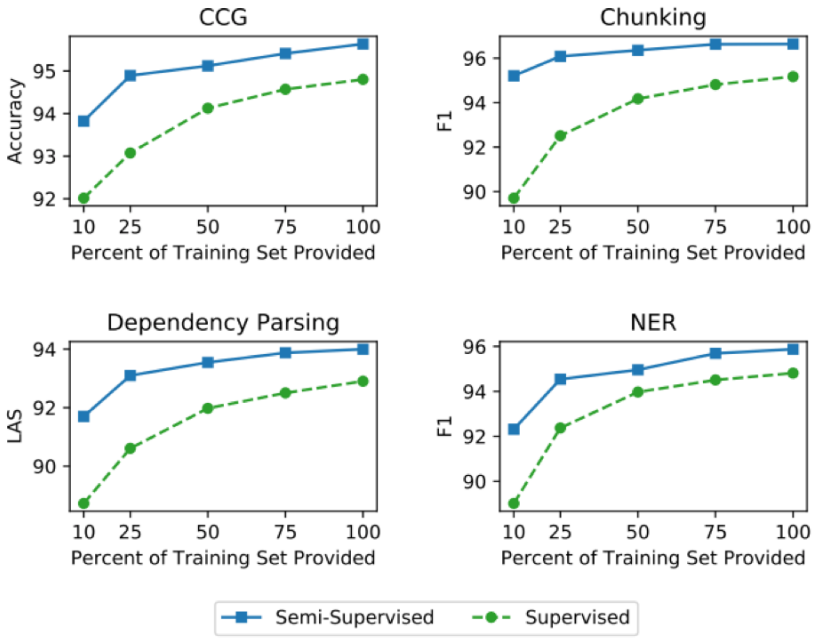
Method	CCG	Chunk	NER	Dep. Parsing	MT (English-Vietnamese)
Previous state-of-the-art	95.1	**96.4**	**92.2**	**94.1**	26.1
Supervised	94.8	94.9	91.2	93.3	28.9
Virtual Adversarial	95.1	95.2	91.6	93.7	--
Word Dropout	**95.2**	95.8	92.1	93.8	**29.3**

Cross View Consistency: Results

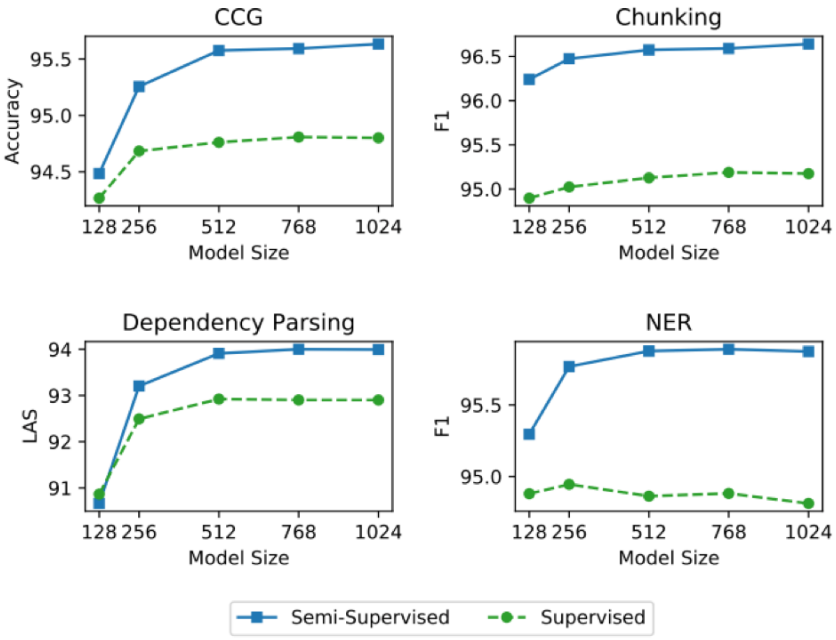
Method	CCG	Chunk	NER	Dep. Parsing	MT (English-Vietnamese)
Previous state-of-the-art	95.1	96.4	92.2	94.1	26.1
Supervised	94.8	94.9	91.2	93.3	28.9
Virtual Adversarial	95.1	95.2	91.6	93.7	--
Word Dropout	95.2	95.8	92.1	93.8	29.3
CVC	**95.6**	**96.5**	**92.4**	**94.2**	**29.7**

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