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 $\mathbf{x}$

This movie was a lot of fun!

The “Classical” NLP Pipeline

Input x

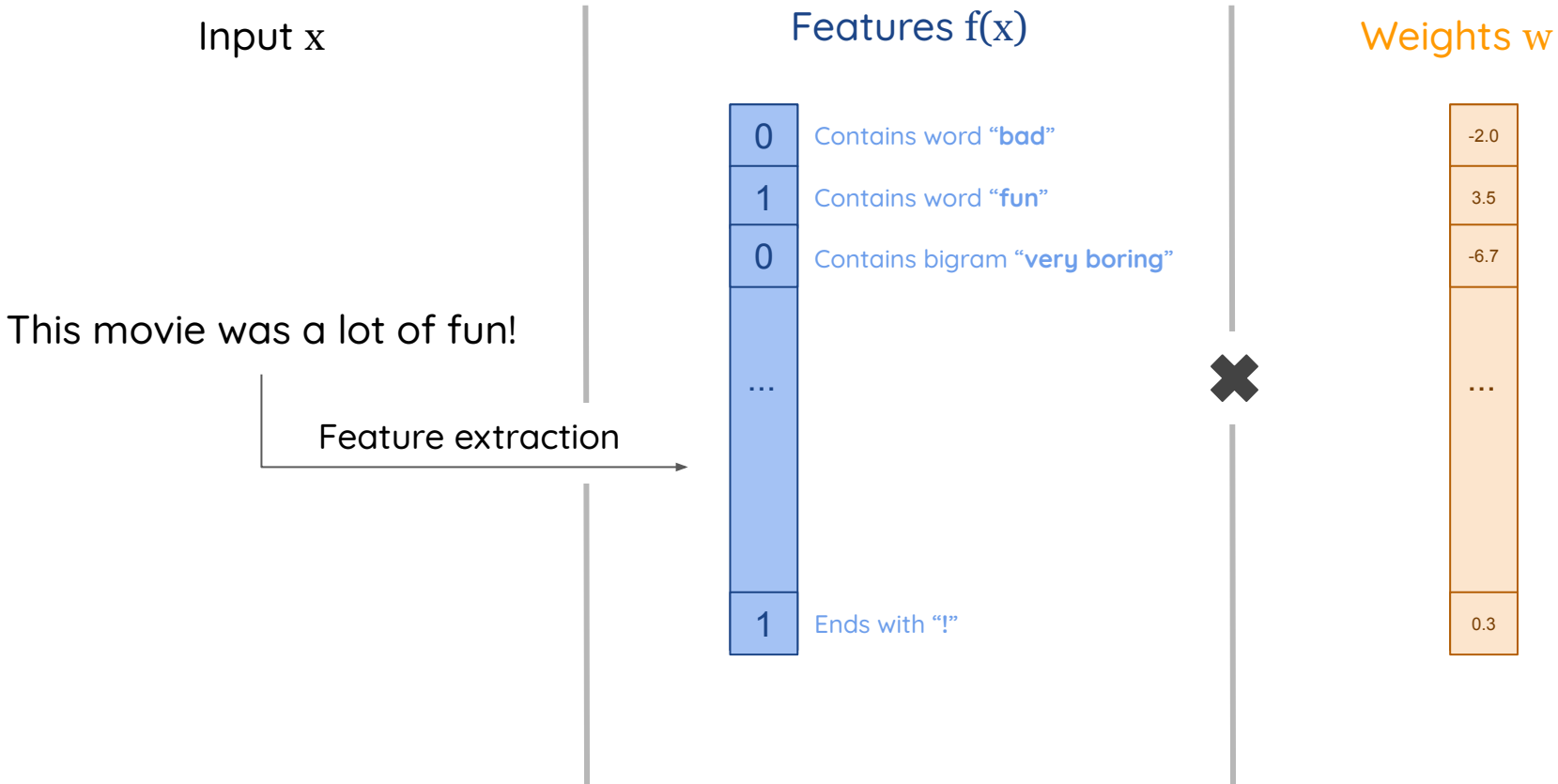
This movie was a lot of fun!

Feature extraction

Features $f(x)$

0	Contains word “**bad**”
1	Contains word “**fun**”
0	Contains bigram “**very boring**”
…	
1	Ends with “!”

The “Classical” NLP Pipeline



The Linear Model

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

The Linear Model

$$\underbrace{\log p(y)}_{\text{Output probability}} \propto \underbrace{w_y^T}_{\text{weights}} \cdot \underbrace{f(x)}_{\text{features}}$$

The Linear Model

$$\underbrace{\log p(y)}_{\text{Output probability}} \propto \boxed{\underbrace{w_y^T}_{\text{weights}}} \cdot \underbrace{f(x)}_{\text{features}}$$

learned

The Linear Model

$$\underbrace{\log p(y)}_{\text{Output probability}} \propto \boxed{\underbrace{w_y^T}_{\text{weights}}} \cdot \boxed{\underbrace{f(x)}_{\text{features}}}$$

learned

hand-crafted

The Linear Model

$$\underbrace{\log p(y)}_{\text{Output probability}} \propto \boxed{\underbrace{w_y^T}_{\text{weights}}} \cdot \boxed{\underbrace{f(x)}_{\text{features}}}$$

learned

hand-crafted

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

The Linear Model

$$\underbrace{\log p(y)}_{\text{Output probability}} \propto \boxed{\underbrace{w_y^T}_{\text{weights}}} \cdot \boxed{\underbrace{f(x)}_{\text{features}}}$$

learned

hand-crafted

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

- Expressivity
- Feature engineering

The Linear Model: Expressivity

- h^*: function we are trying to approximate

The Linear Model: Expressivity

- h^*: function we are trying to approximate

- h_θ: function (model) parametrized by θ

The Linear Model: Expressivity

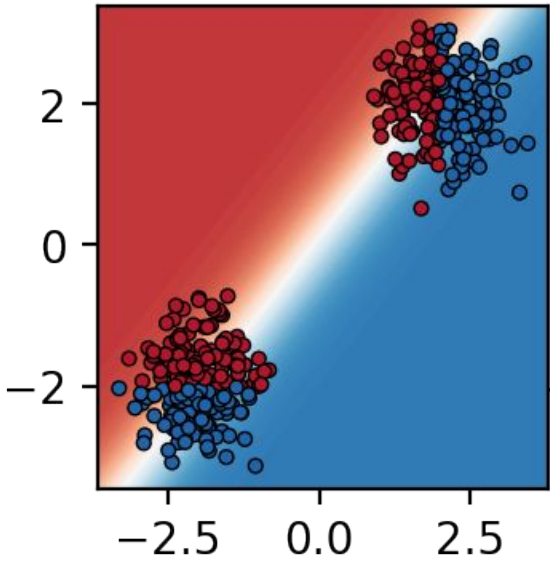
- h^*: function we are trying to approximate

- h_θ: function (model) parametrized by θ

- We want there to be a θ* 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_θ: function (model) parametrized by θ
- We want there to be a θ* such that
  - > $h^* = h_{\theta^*}$
  - > Or at least $|h^* - h_{\theta^*}|$ is small



The Linear Model

$$\underbrace{\log p(y)}_{\text{Output probability}} \propto \underbrace{w_y^T}_{\text{weights}} \cdot \underbrace{f(x)}_{\text{features}}$$

Deep Learning

$$\underbrace{\log p_\theta(y)}_{\text{Output probability}} \propto \underbrace{w_y^T}_{\text{weights}} \cdot \underbrace{f_\theta(x)}_{\text{features}}$$

Deep Learning

$$\underbrace{\log p_\theta(y)}_{\text{Output probability}} \propto \boxed{\underbrace{w_y^T}_{\text{weights}}} \cdot \boxed{\underbrace{f_\theta(x)}_{\text{features}}}$$

learned

also learned!

Deep Learning

$$\underbrace{\log p_\theta(y)}_{\text{Output probability}} \propto \boxed{\underbrace{w_y^T}_{\text{weights}}} \cdot \boxed{\underbrace{f_\theta(x)}_{\text{features}}}$$

learned

also learned!

+ More expressive
+ Less feature engineering

Deep Learning

$$\underbrace{\log p_\theta(y)}_{\text{Output probability}} \propto \boxed{\underbrace{w_y^T}_{\text{weights}}} \cdot \boxed{\underbrace{f_\theta(x)}_{\text{features}}}$$

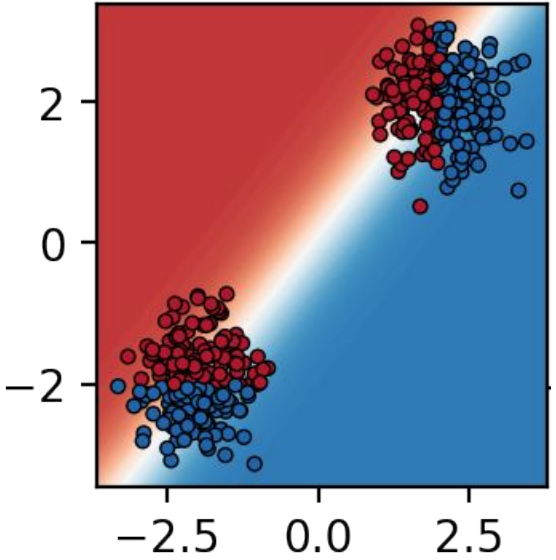
learned

also learned!

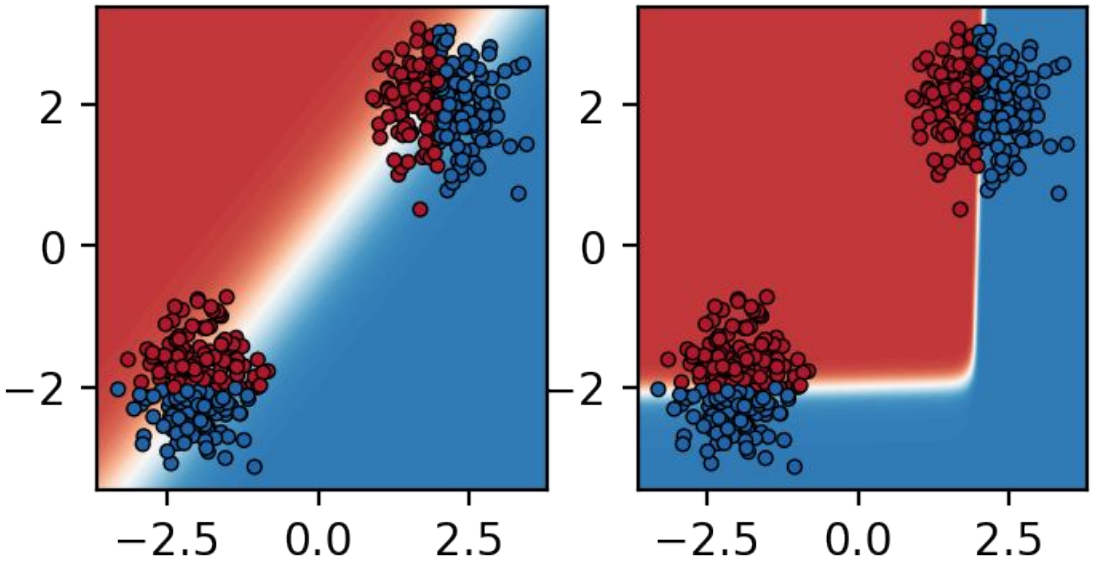
+ More expressive
+ Less feature engineering

- 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_θ 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

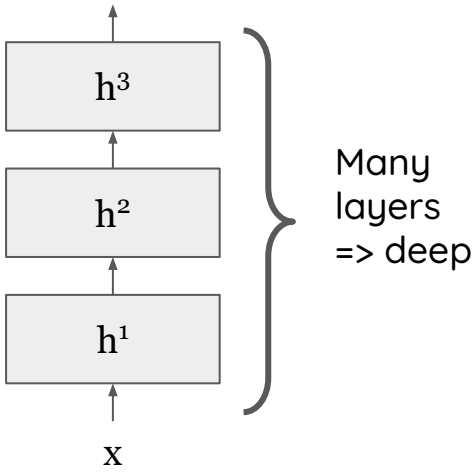
- In theory f_θ could be anything expressive enough
  - > Polynomial? $f_\theta(x) = \theta_o + \theta_1 x + \theta_2 x^2 + \ldots + \theta_n x^n$
  - > Sinusoidal? $f_\theta(x) = \theta_o + \theta_{1,1}\cos(x) + \theta_{1,2}\sin(x) + \ldots + \theta_{n,1}\cos(nx) + \theta_{n,2}\sin(nx)$
  - > ...

- 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”

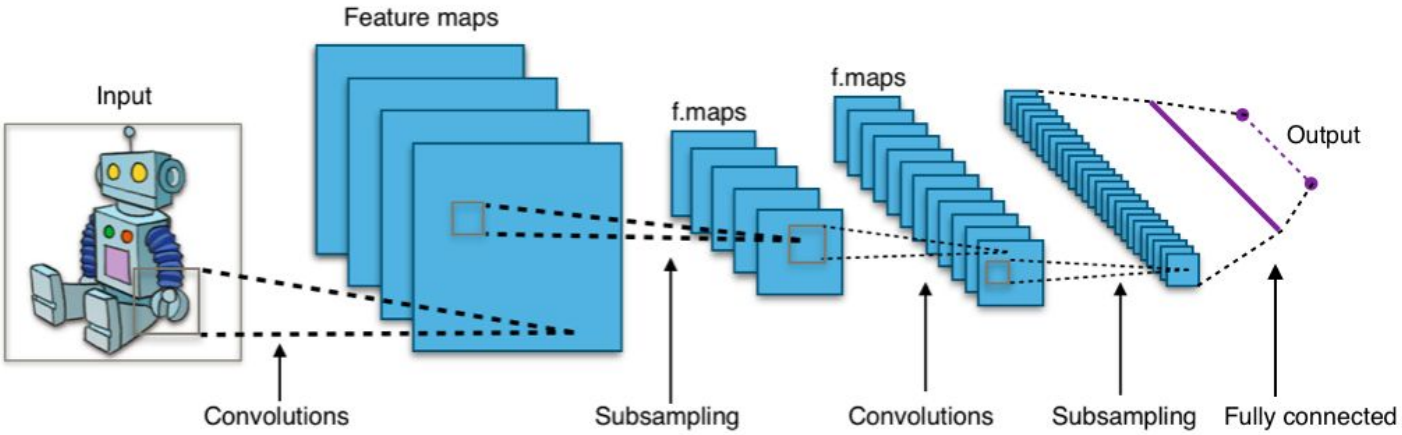
Model Architecture

- In theory f_θ 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”

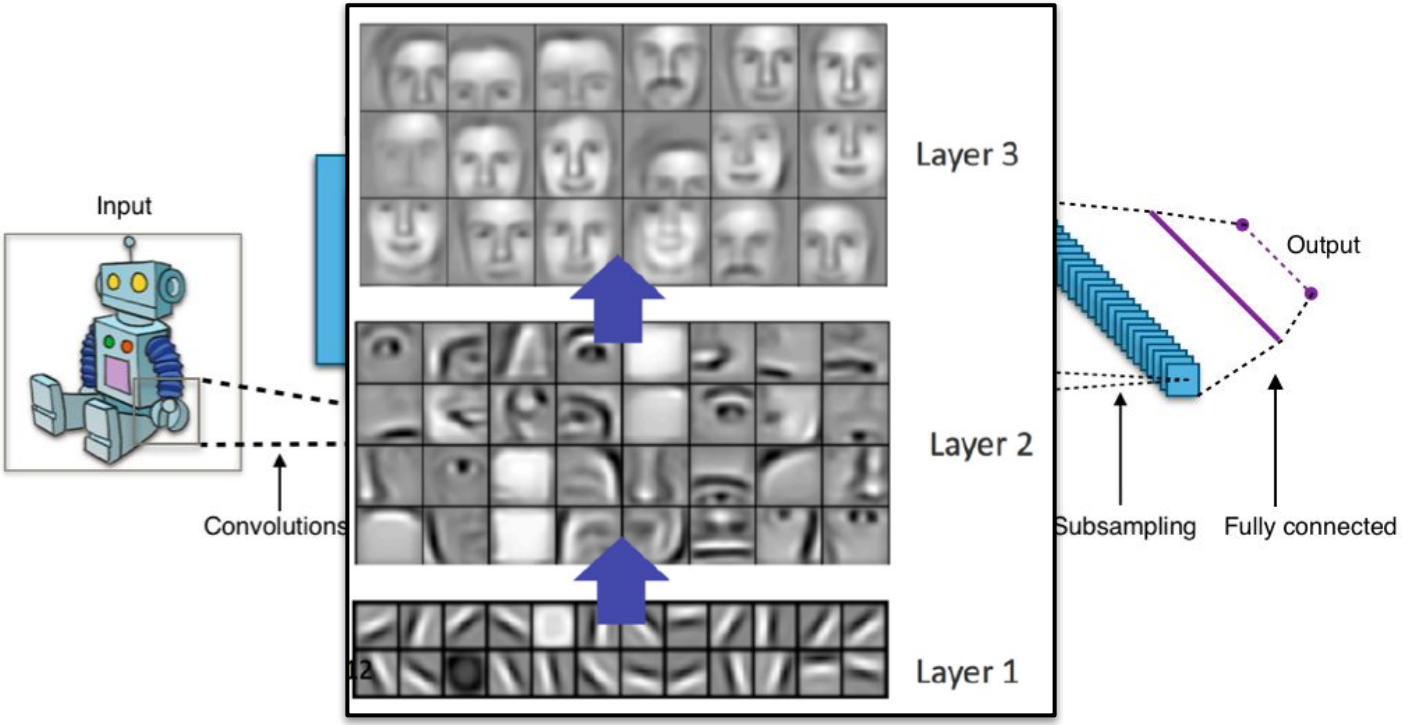


Model Architecture: Stacked Layers



Wikipedia

Model Architecture: Stacked Layers



???

Model Architecture: Stacked Layers

$$f_\theta(x) = h^n_{\theta n}(\dots h^2_{\theta 2}(h^1_{\theta 1}(x)) \dots)$$

Model Architecture: Stacked Layers

$$f_\theta(x) = h^n_{\theta n}(\dots h^2_{\theta 2}(h^1_{\theta 1}(x)) \dots)$$

- 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

$$f_\theta(x) = h^n_{\theta n}(\dots h^2_{\theta 2}(h^1_{\theta 1}(x)) \dots)$$

- 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

$$f_\theta(x) = h^n_{\theta n}(\dots h^2_{\theta 2}(h^1_{\theta 1}(x)) \dots)$$

- 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 $\mathbf{h}$?

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

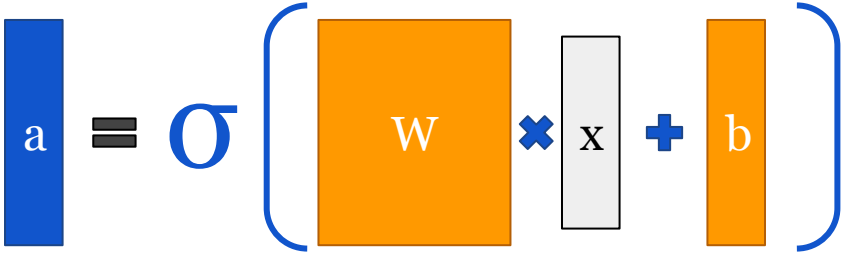
- Input vector: $\mathbf{x} = [x_1,\ldots,x_n]$
- "Neuron" i: $a_i = \sigma(w_{i1}x_1 + \ldots + w_{in}x_n + b_i)$
- m neurons in parallel:

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

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:

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

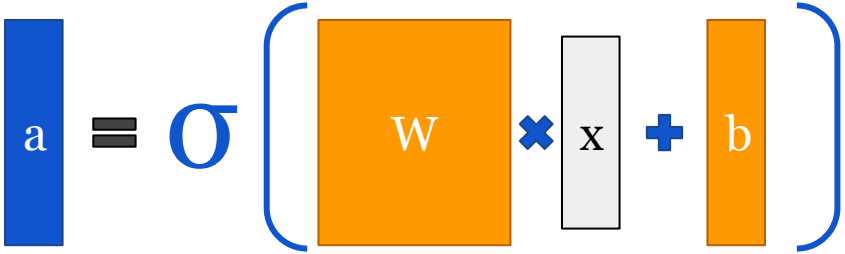


- With parameters:
  - > $\mathbf{W}$: a m x n **weight** matrix
  - > $\mathbf{b}$: a m-dimensional **bias** vector
  - > σ: an **activation** function

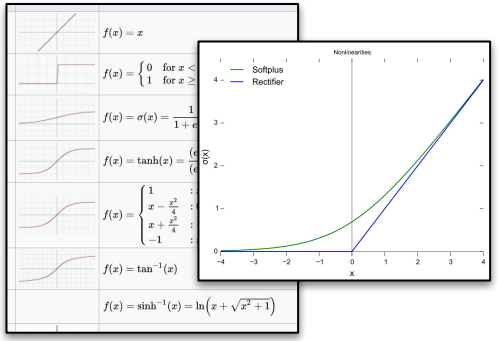
The Dense Layer

- 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:

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



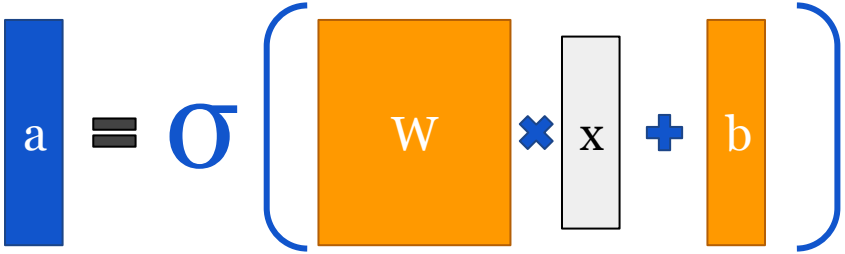
- With parameters:
  - > **W**: a m x n **weight** matrix
  - > **b**: a m-dimensional **bias** vector
  - > σ: an **activation** function



The Dense Layer

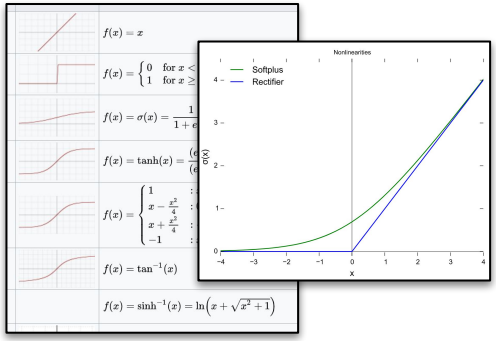
- Input vector: $\mathbf{x} = [x_1,\ldots,x_n]$
- "Neuron" i: $a_i = \sigma(w_{i1}x_1 + \ldots + w_{in}x_n + b_i)$
- m neurons in parallel:

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



- With parameters:
  - > **W**: a m x n **weight** matrix
  - > **b**: a m-dimensional **bias** vector
  - > σ: an **activation** function

- Sufficient for the universal approximation theorem!



Embeddings

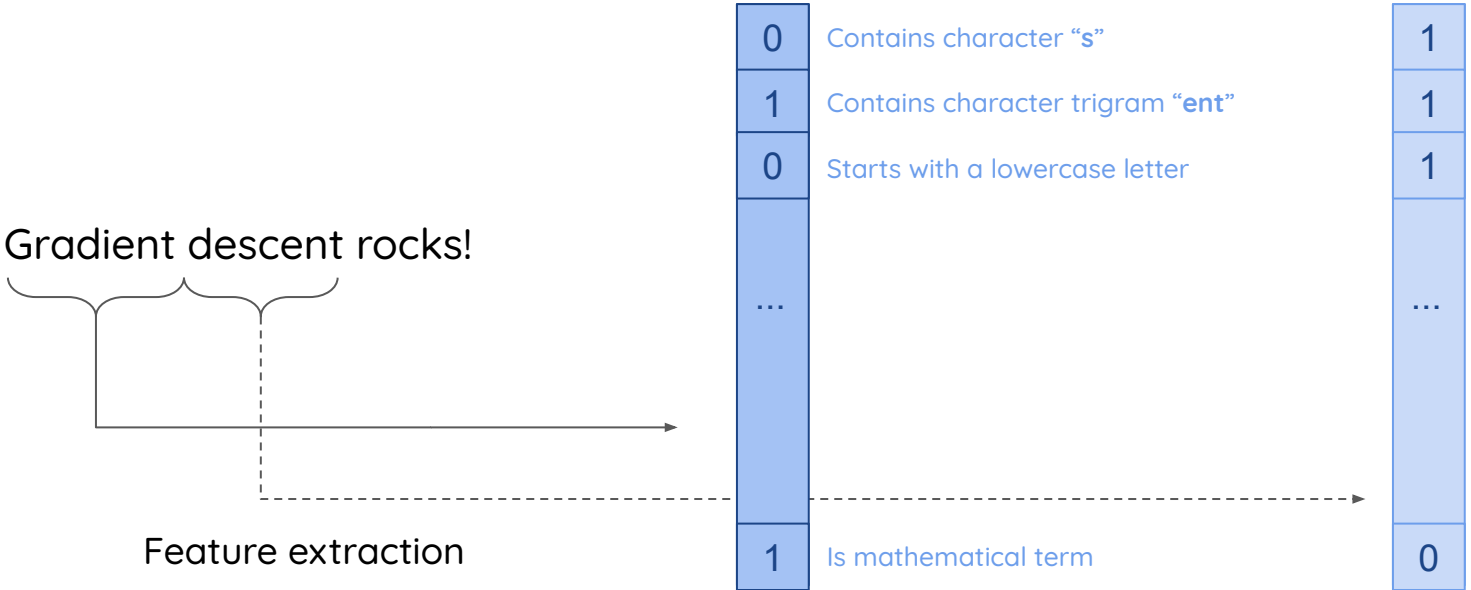
- In NLP our inputs are words, not vectors

Embeddings

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

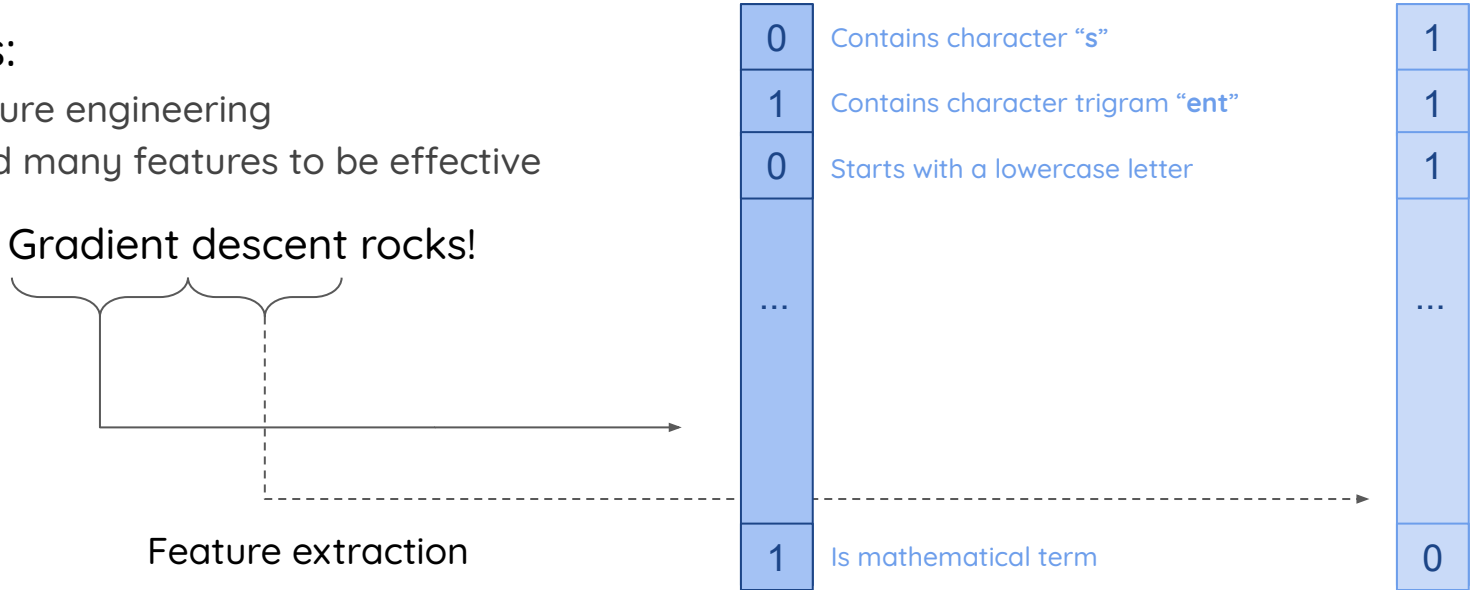
Embeddings

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



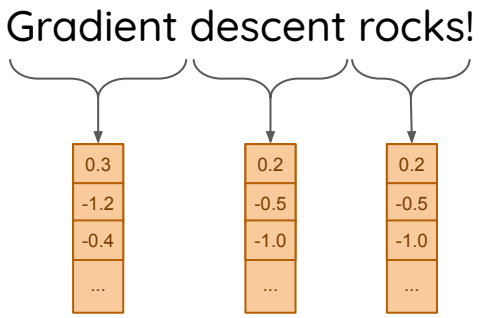
Embeddings

- 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



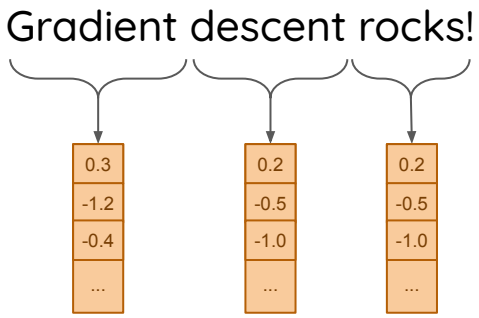
Embeddings

- 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



Embeddings

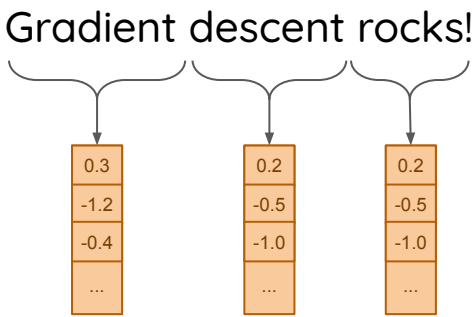
- In NLP our inputs are words, not vectors
- How do we turn words into vectors?
- Second attempt: **learned word embeddings**
  - Fix a dimension d
  - Assign a d-dimensional vector to each word
  - Initialize at random and learn like any other parameters
- Problems
  - Sensitive to initialization
  - Not interpretable
  - Sparse gradients for low-frequency words



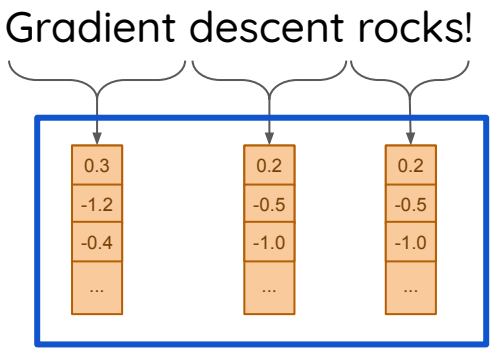
Processing Sequences

Gradient descent rocks!

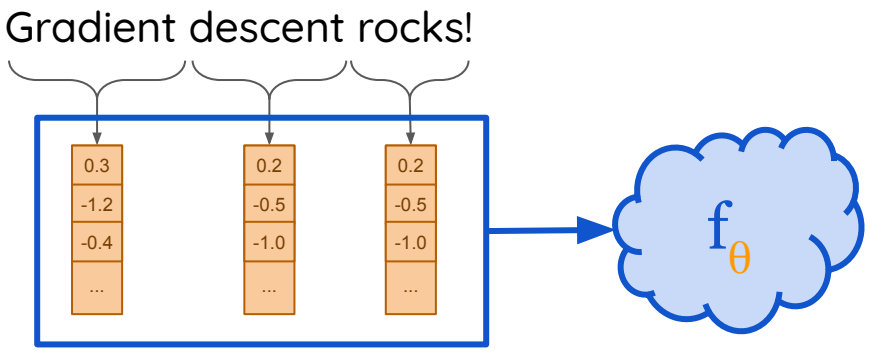
Processing Sequences



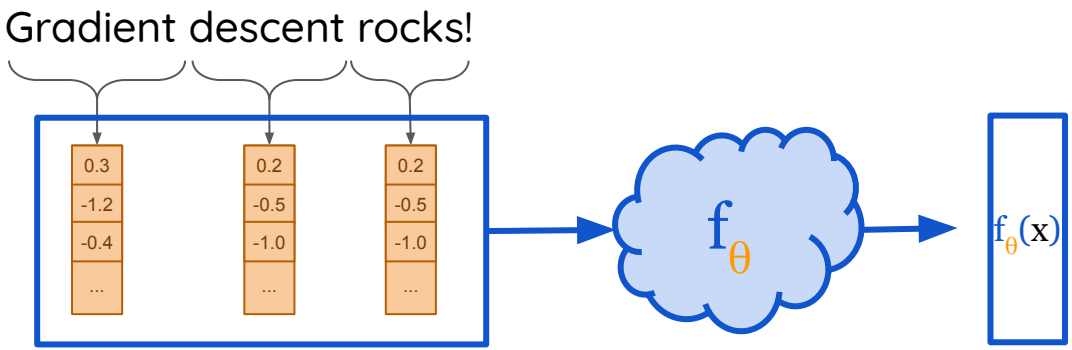
Processing Sequences



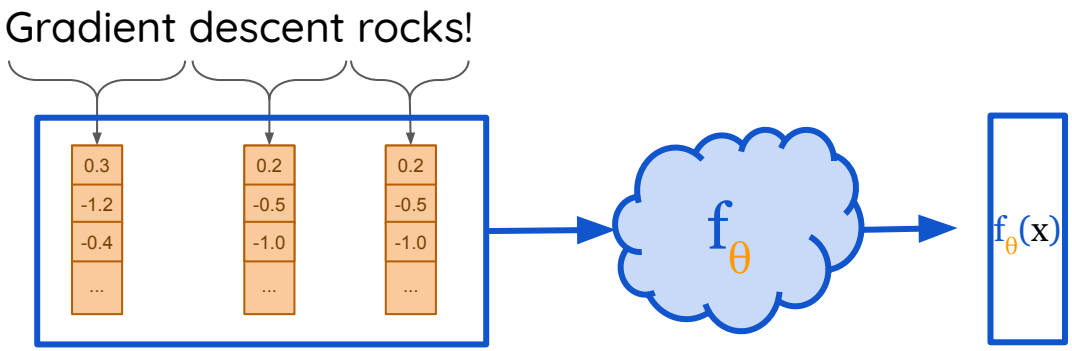
Processing Sequences



Processing Sequences



Processing Sequences



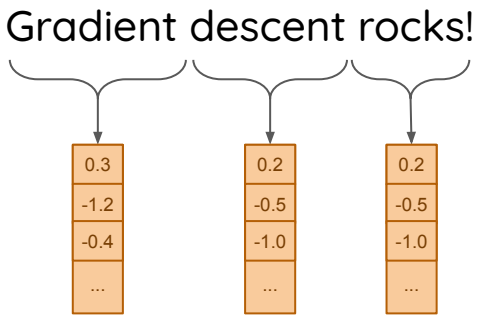
- How do we turn a sequence of vectors into a vector for classification?
  - How do we capture the relationship between words?

Recurrent Layers

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

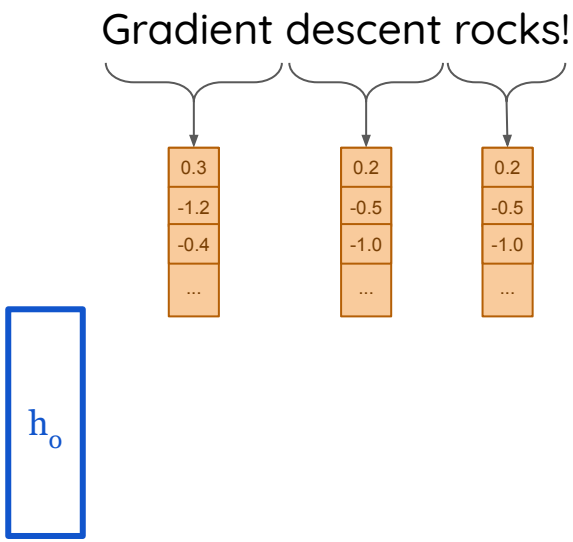
Recurrent Layers

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



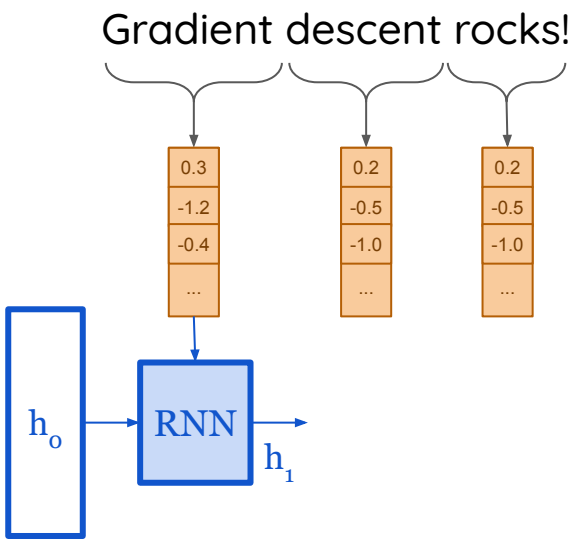
Recurrent Layers

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



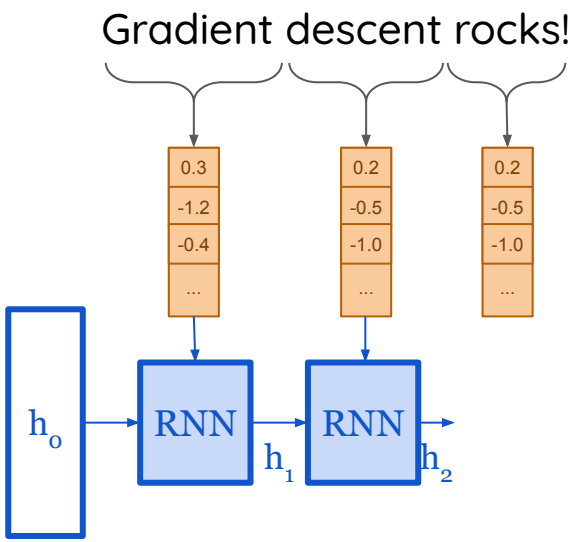
Recurrent Layers

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



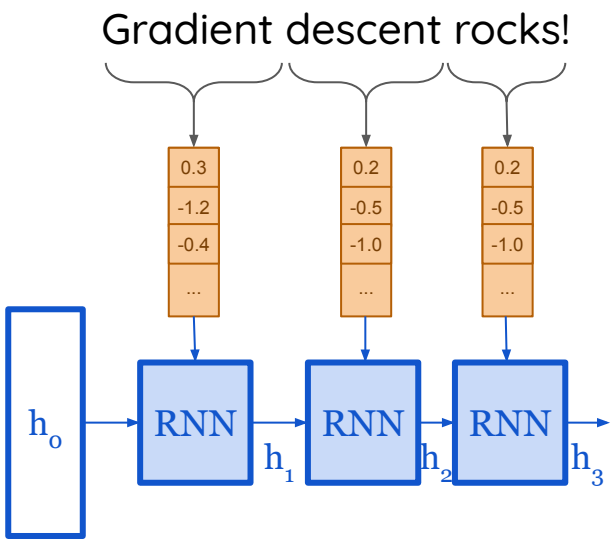
Recurrent Layers

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



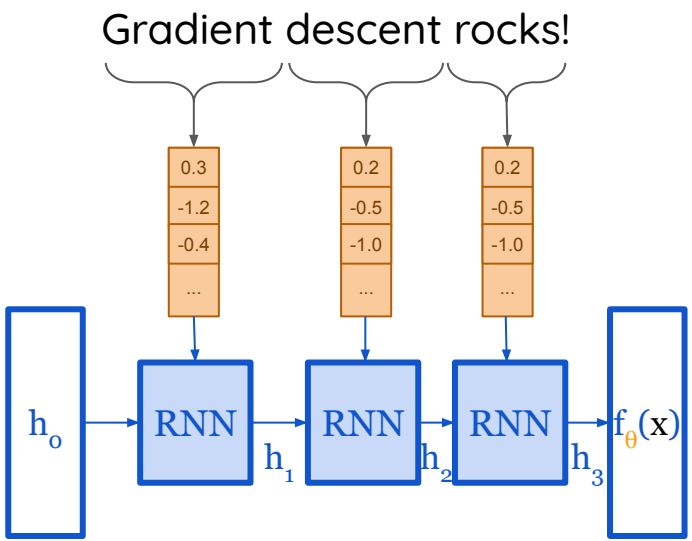
Recurrent Layers

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



Recurrent Layers

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



Recurrent Layers

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

Recurrent Layers

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

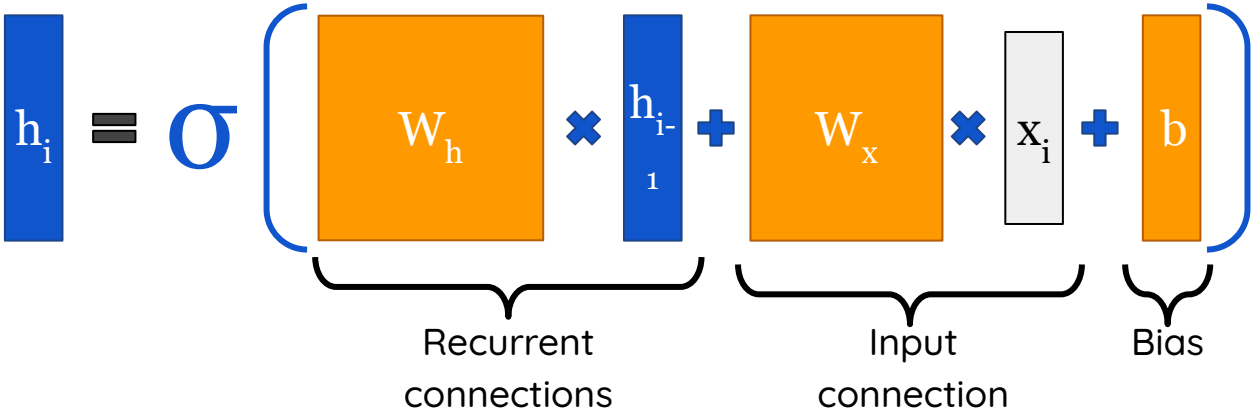
$$h_i = \sigma\left(W_x \times x_i + b \right)$$

Input connection

Bias

Recurrent Layers

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

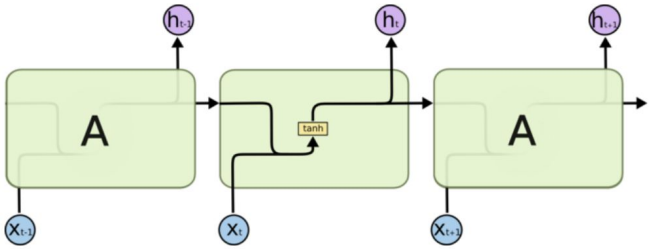


Recurrent Layers

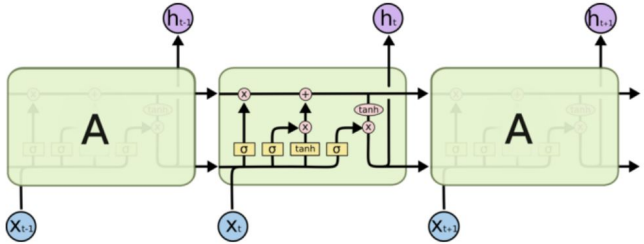
- Idea: like an automaton
  - $h_i = RNN(x_i, h_{i-1})$
  - The last h is a function of the whole sentence
- In practice this is not used anymore
  - Too hard to train (exploding/vanishing gradients)

Recurrent Layers

- Idea: like an automaton
  - > $h_i = RNN(x_i, h_{i-1})$
  - > 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



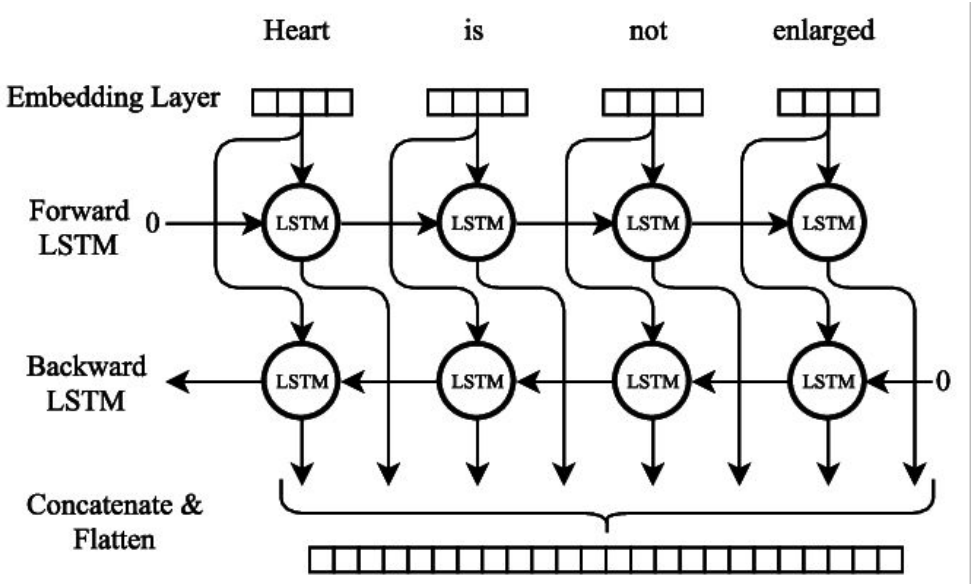
Regular RNNs



LSTMs

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

Bidirectional LSTM



Cornegruta et al. (2016)

Attention Layers

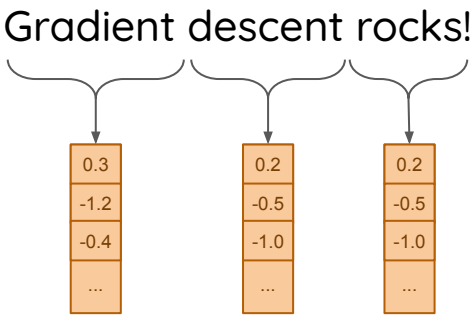
- How do we turn a sequence of vectors into a vector for classification?
  - Why not just sum/average the word vectors?
  - We treat all words equally: bad

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

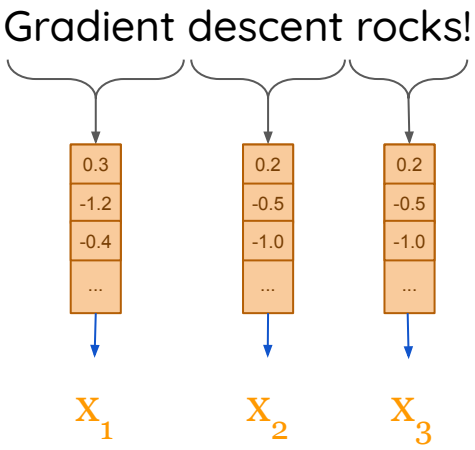
Attention Layers

- 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



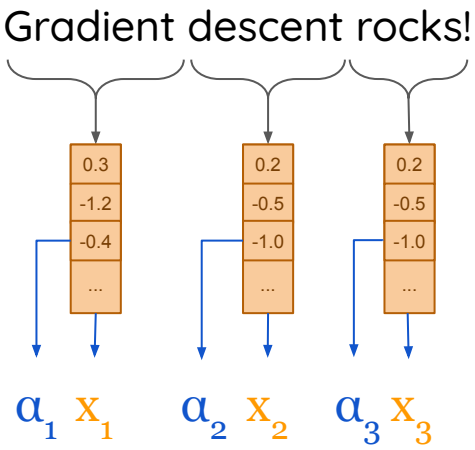
Attention Layers

- 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



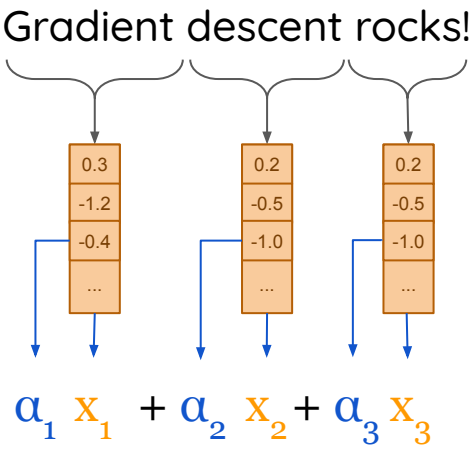
Attention Layers

- 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



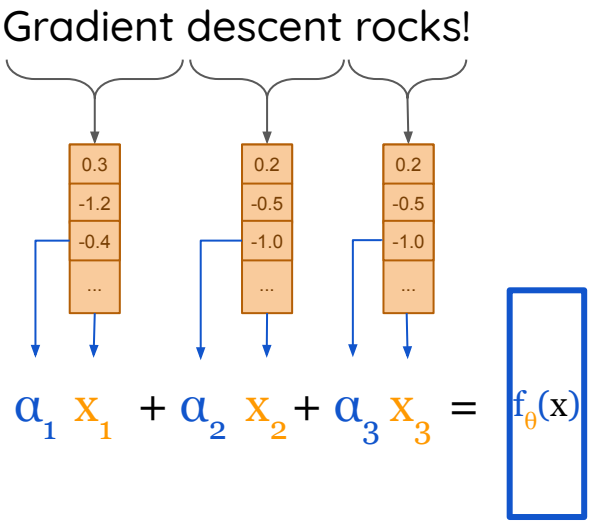
Attention Layers

- 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

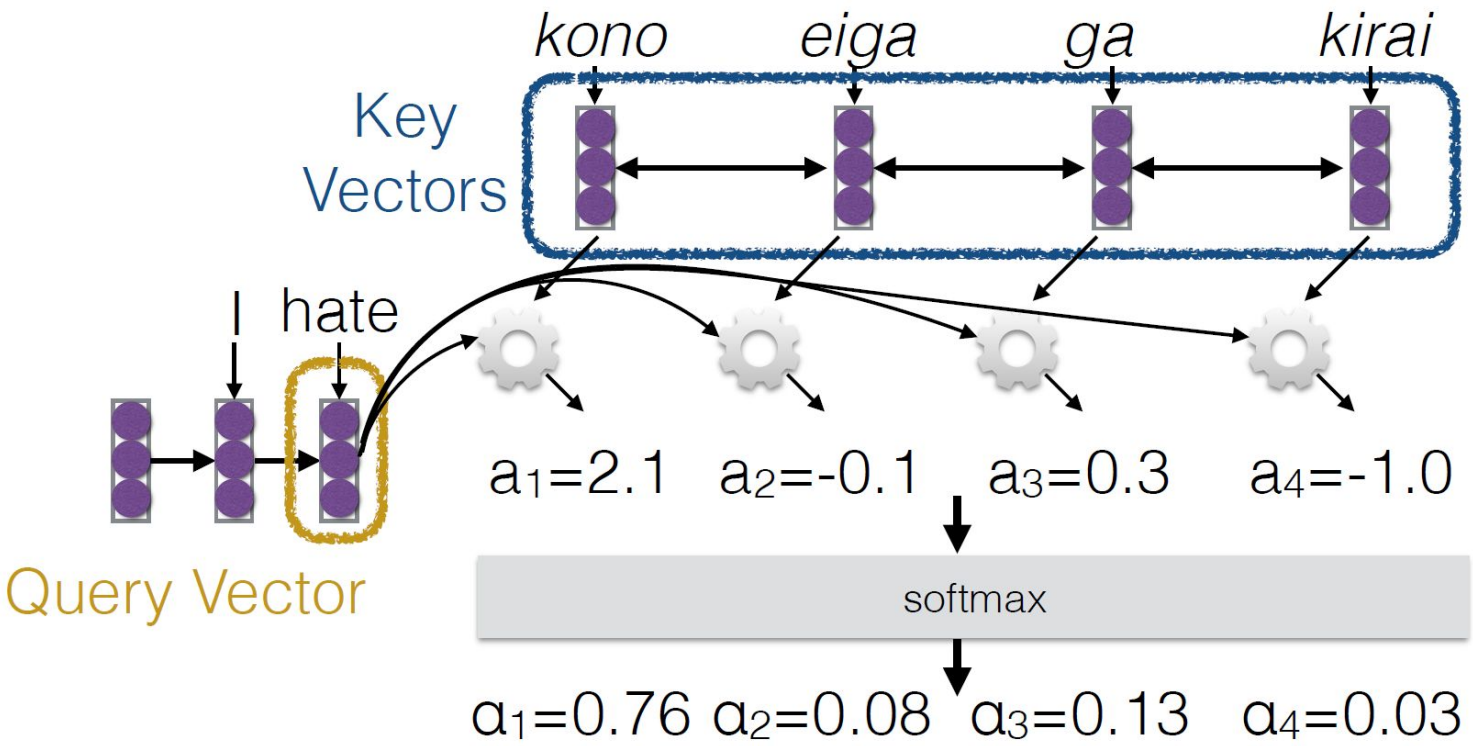


Attention Layers

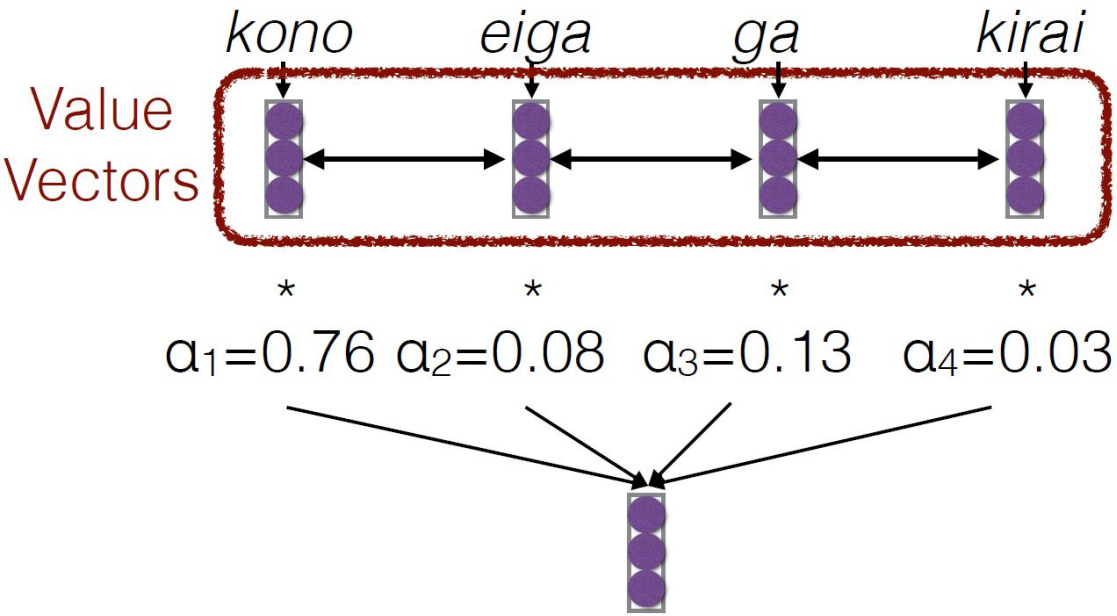
- How do we turn a sequence of vectors into a vector for classification?
  - > Why not just sum/average the word vectors?
  - > We treat all words equally: bad
- Attention: weighted sum of vectors



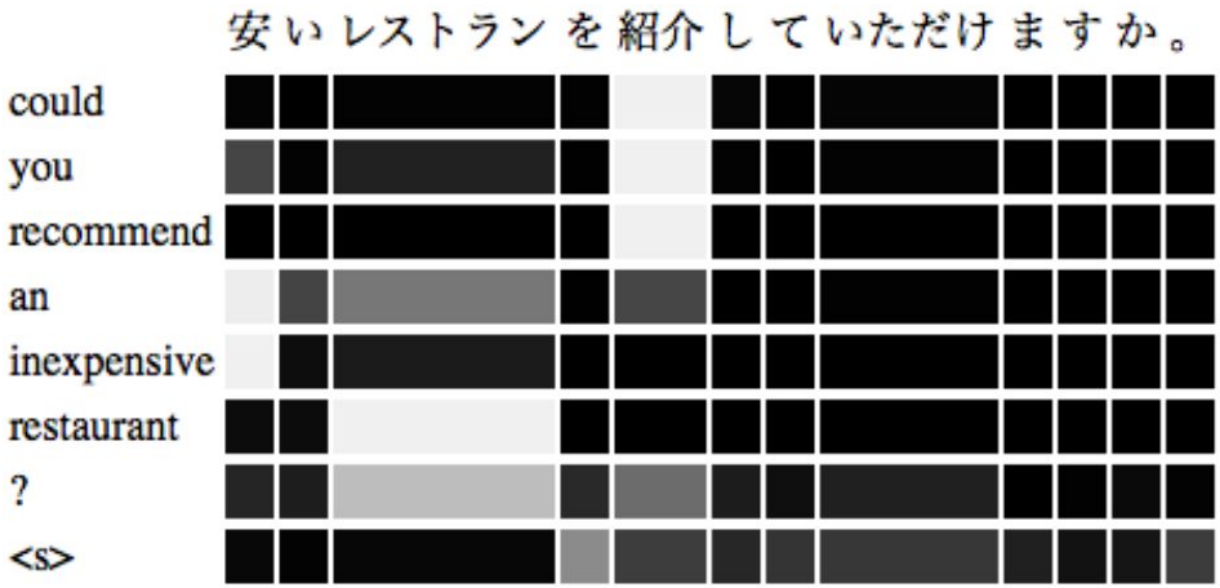
Attention Layers Example: Machine Translation



Attention Layers Example: Machine Translation

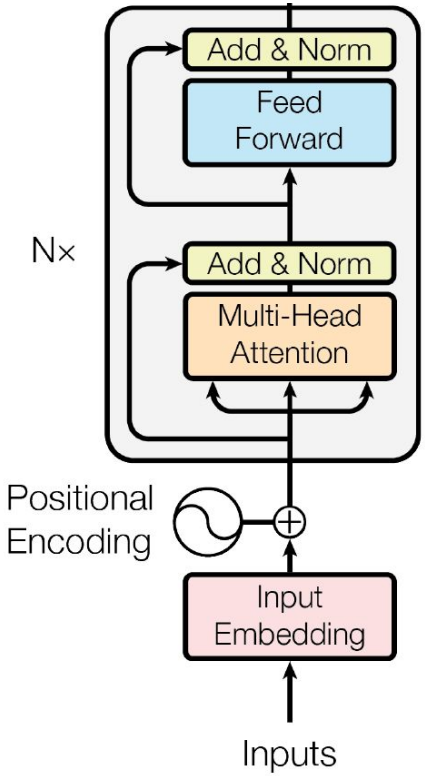


Attention Layers Example: Machine Translation



Transformer

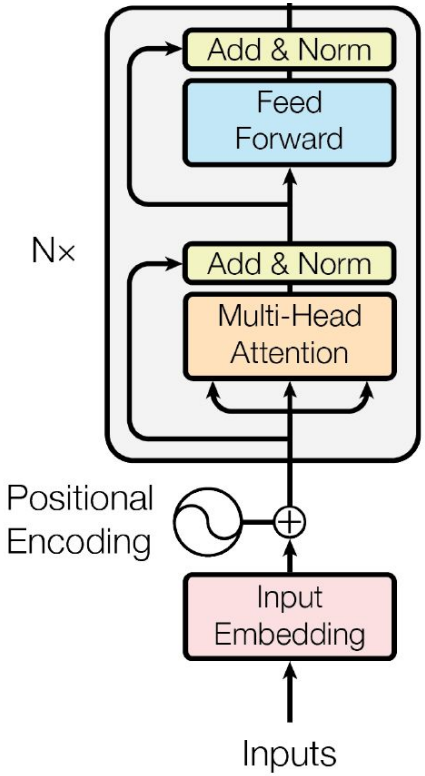
- Fairly recent (2017)
  - Originally proposed for machine translation
  - State of the art in many NLP tasks



Vaswani et al. (2017)

Transformer

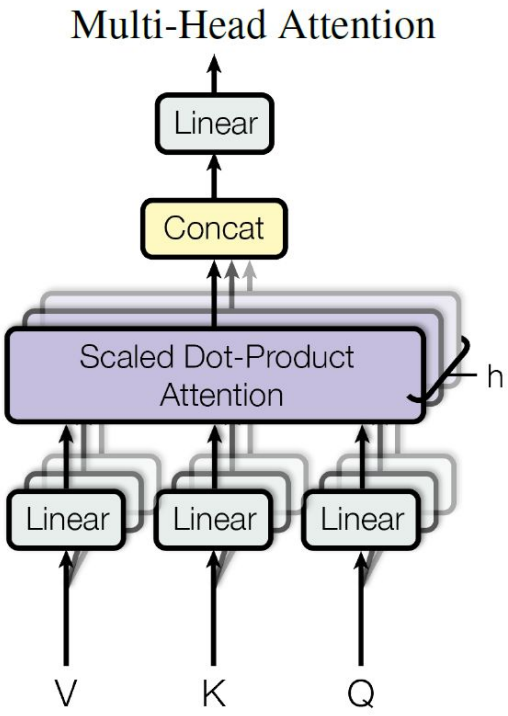
- Fairly recent (2017)
  - Originally proposed for machine translation
  - State of the art in many NLP tasks
- Main characteristics
  - Uses mostly attention/dense layers
  - No recurrent layers at all!
  - “Attention is all you need”



Vaswani et al. (2017)

Transformer

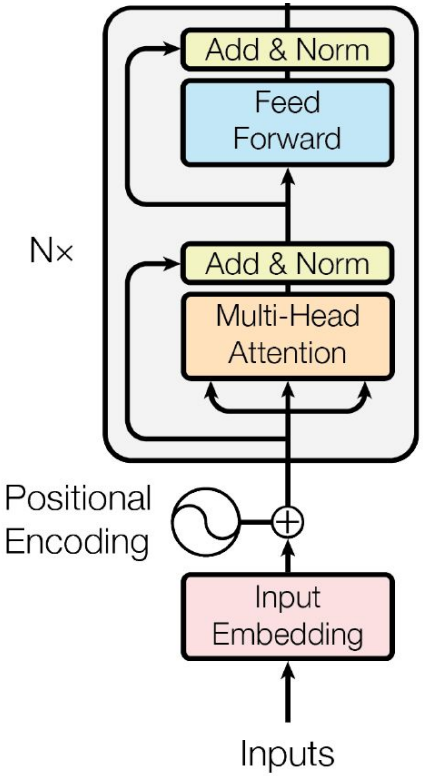
- Fairly recent (2017)
  - Originally proposed for machine translation
  - State of the art in many NLP tasks
- Main characteristics
  - Uses mostly attention/dense layers
  - No recurrent layers at all!
  - “Attention is all you need”



Vaswani et al. (2017)

Transformer

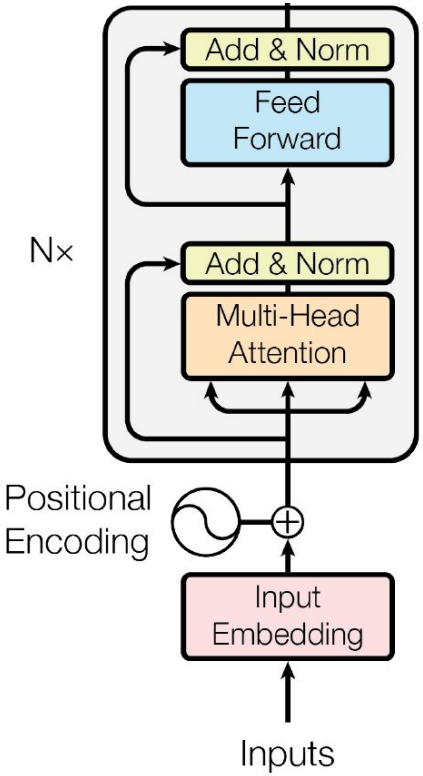
- Fairly recent (2017)
  - Originally proposed for machine translation
  - State of the art in many NLP tasks
- Main characteristics
  - Uses mostly attention/dense layers
  - No recurrent layers at all!
  - “Attention is all you need”
- Pros
  - Easy to parallelize on the GPU (large matrix multiplications)
  - “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 θ

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 θ

- Training: $\theta^{*}=\text{argmin}_{\theta} L(\theta)$

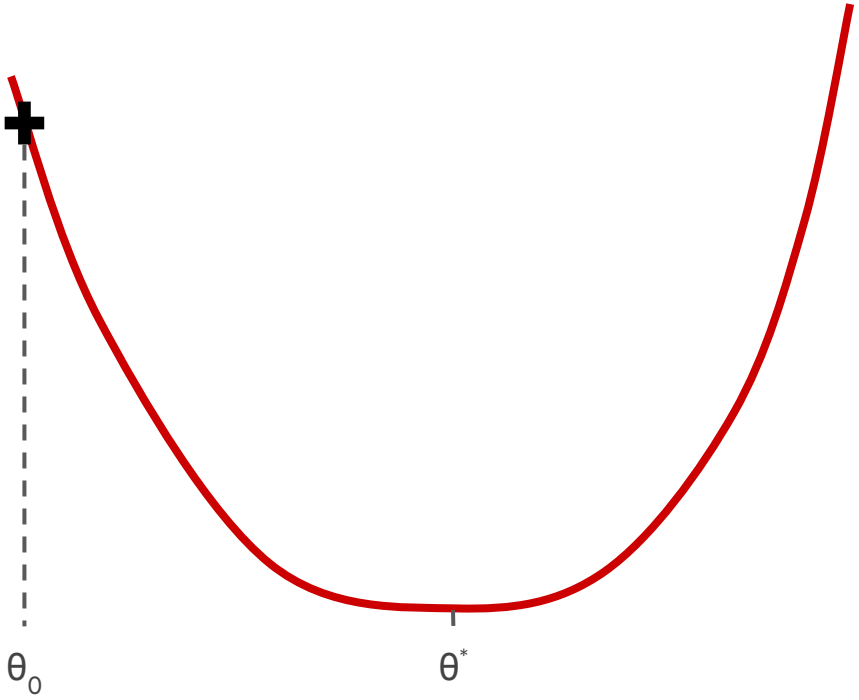
Deep Learning: Training

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

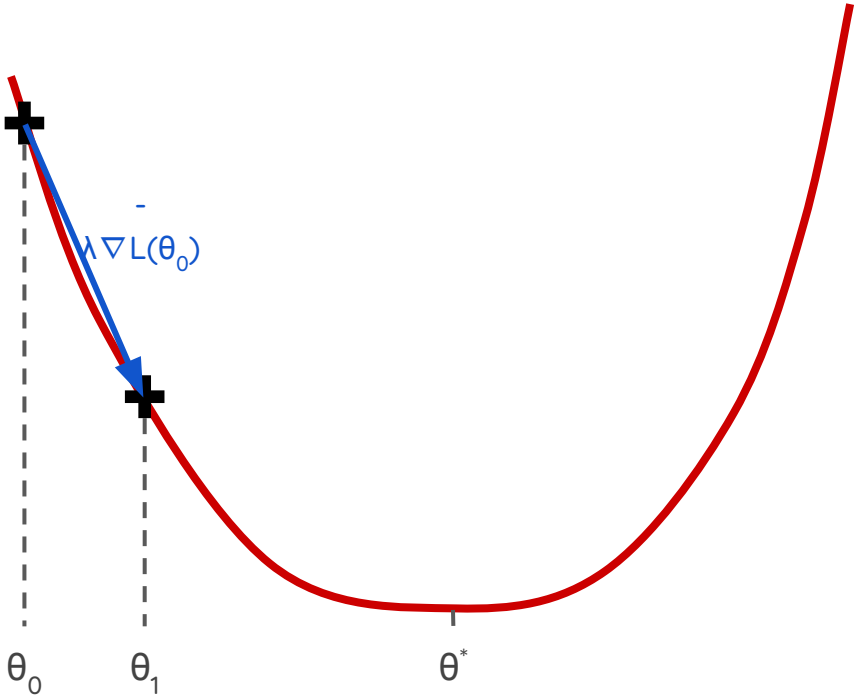
- Training: $\theta^{*}=\text{argmin}_{\theta} L(\theta)$

- Gradient descent:
  - Take small steps following the direction of steepest descent of L
  - θ_0 random
  - $\theta_{t+1} = \theta_t - \lambda \nabla L(\theta_t)$ with small λ

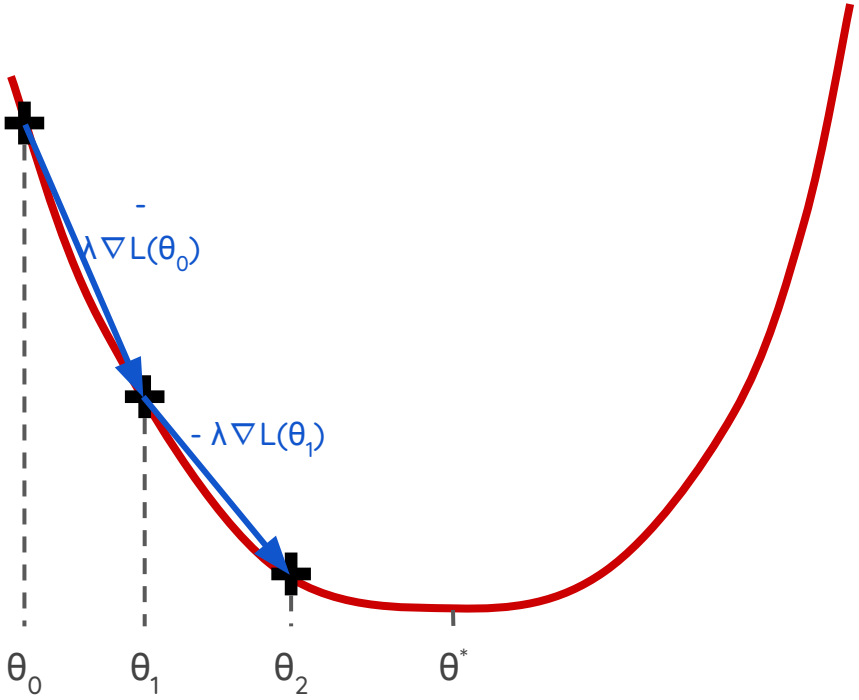
Gradient Descent



Gradient Descent



Gradient Descent



Gradient descent

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

Gradient descent

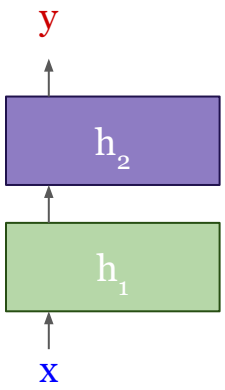
- Gradient descent:
  - > Take small steps following the direction of steepest descent of L
  - > θ_o random
  - > $\theta_{t+1} = \theta_t - \lambda \nabla L(\theta_t)$ with small λ
- How is $\nabla L(\theta_t)$ computed?

Gradient descent

- Gradient descent:
  - Take small steps following the direction of steepest descent of L
  - θ_o random
  - $\theta_{t+1} = \theta_t - \lambda \nabla L(\theta_t)$ with small λ
- How is $\nabla L(\theta_t)$ computed? Backpropagation
  - $y = h_2(h_1(x))$
  - $dy/dx = [dh_2/dh_1] . [dh_1/dx]$

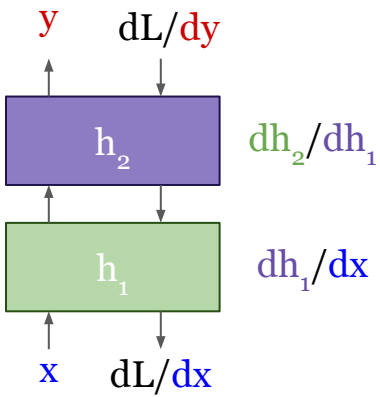
Gradient descent

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

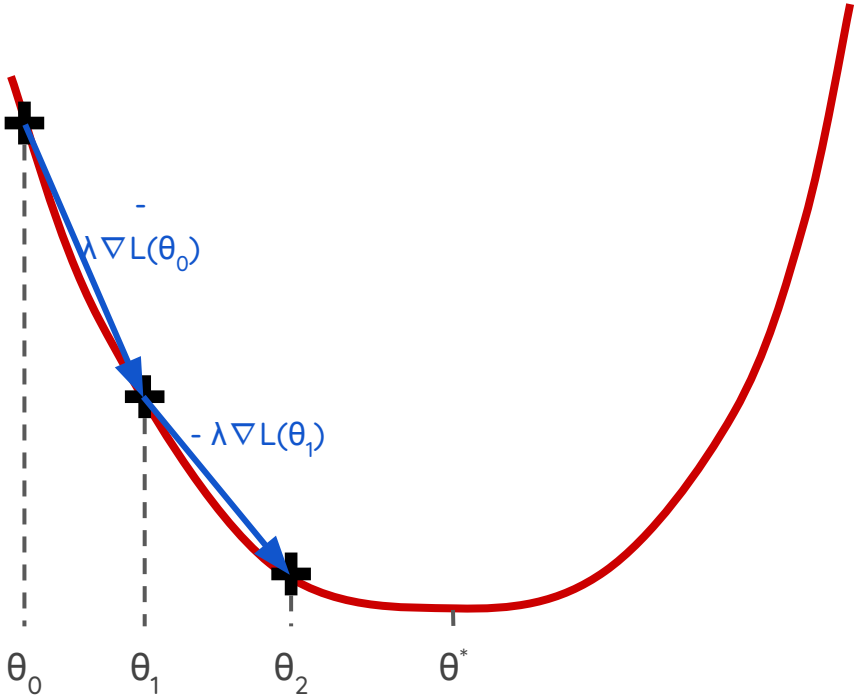
- Gradient descent:
  - > Take small steps following the direction of steepest descent of L
  - > θ_o random
  - > $\theta_{t+1} = \theta_t - \lambda \nabla L(\theta_t)$ with small λ
- How is $\nabla L(\theta_t)$ computed? Backpropagation
  - > $y = h_2(h_1(x))$
  - > $dy/dx = [dh_2/dh_1] . [dh_1/dx]$



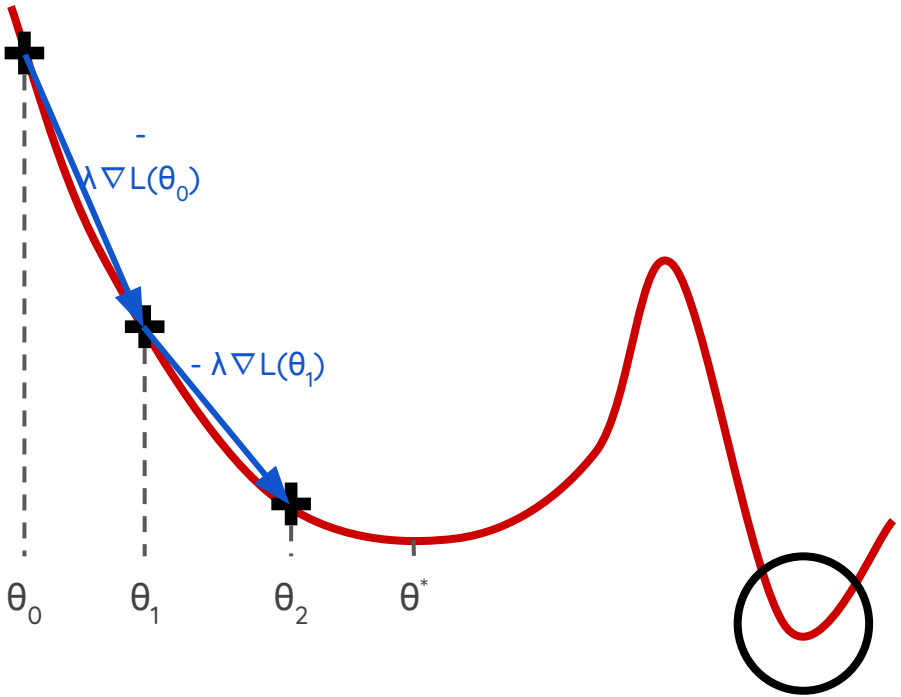
Difficulties of Training Neural Networks

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

Gradient Descent: Convex Function



Gradient Descent: Non convexity

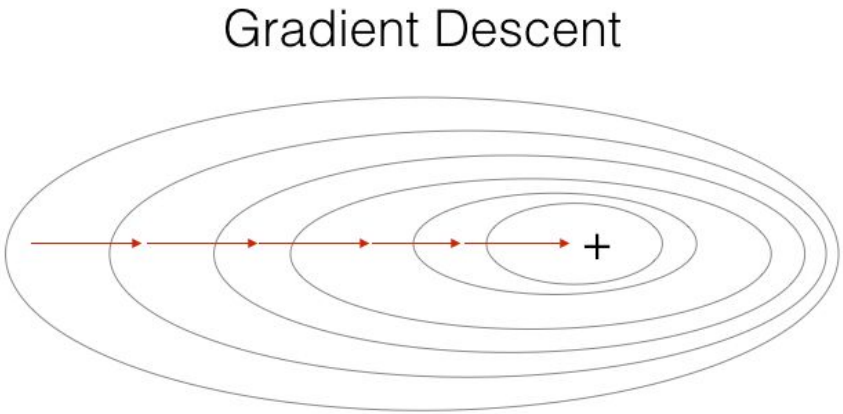


Difficulties of Training Neural Networks

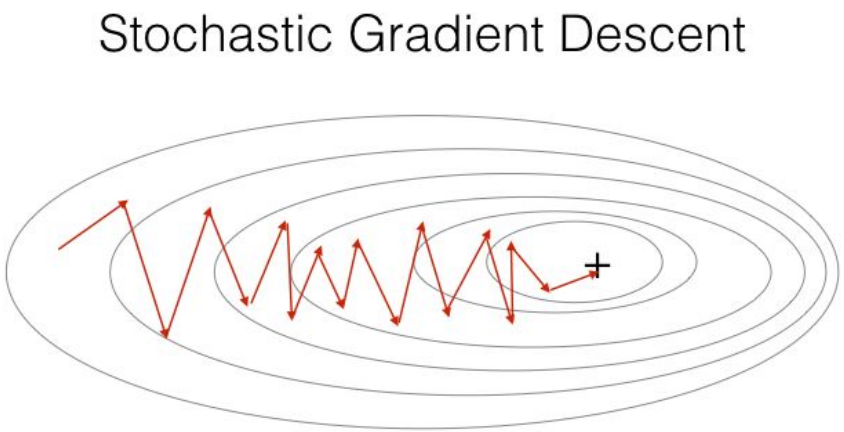
- Non-convexity
  - No guarantee that gradient descent converges to a global minimum

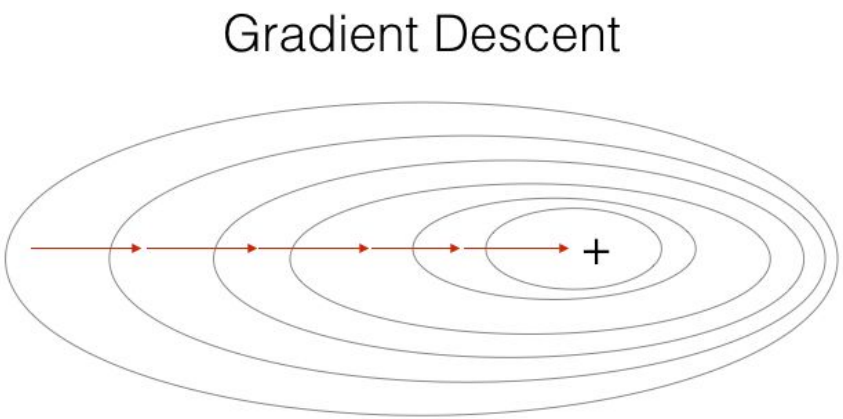
- Gradient computation
  - Requires forward+backward pass for all examples: lot of memory
  - Minibatch training compute the loss and gradient on a subset of the data at each step
  - “Stochastic” Gradient Descent (SGD)

Stochastic Gradient Descent

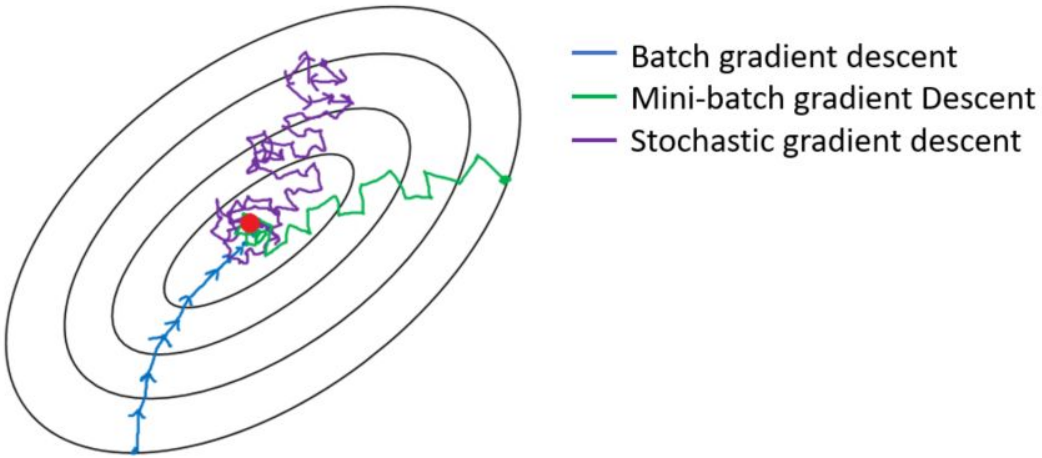


Stochastic Gradient Descent





Stochastic Gradient Descent

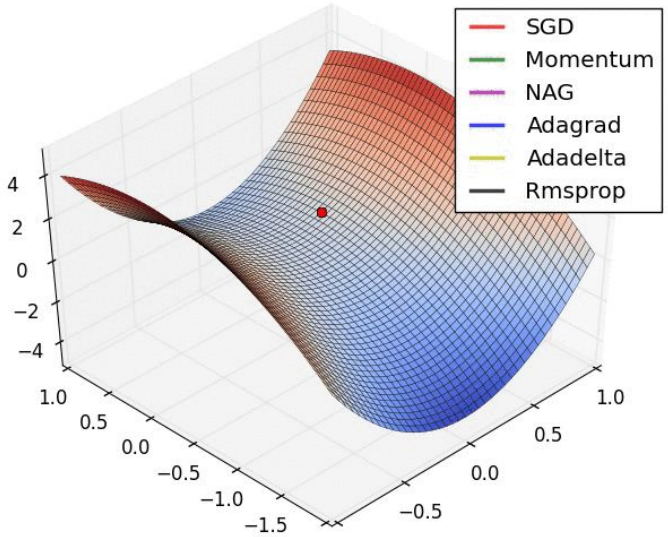


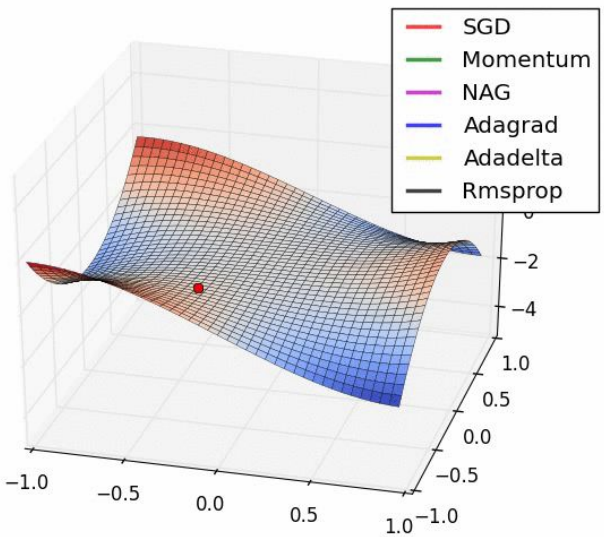
Stochastic Gradient Descent: Variants

- Many variants have been developed to address the shortcomings of SGD

Stochastic Gradient Descent: Variants

- Many variants have been developed to address the shortcomings of SGD





Alec Radford

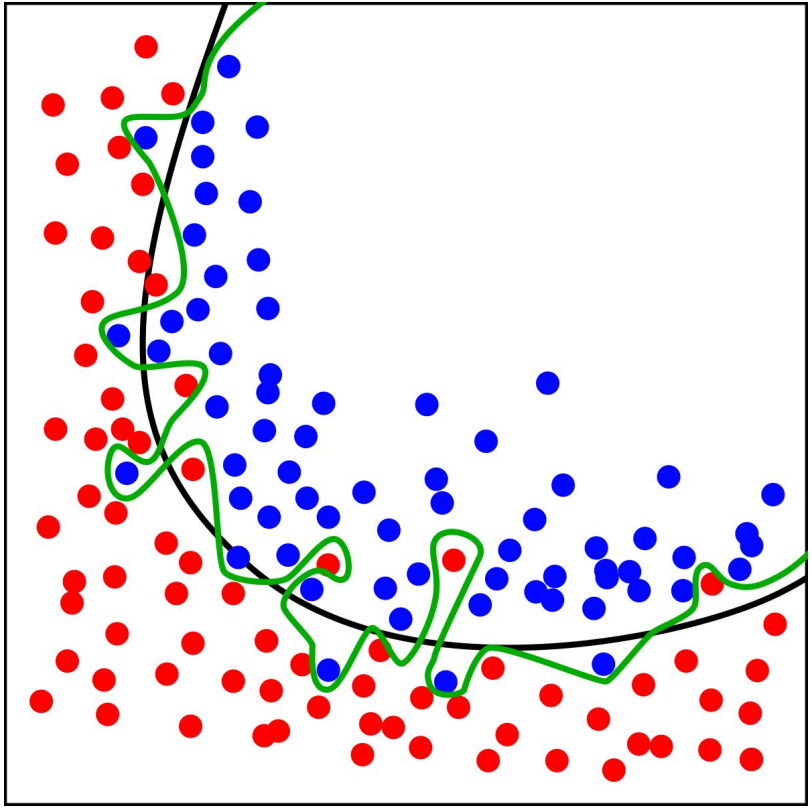
Difficulties of Training Neural Networks

- 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!

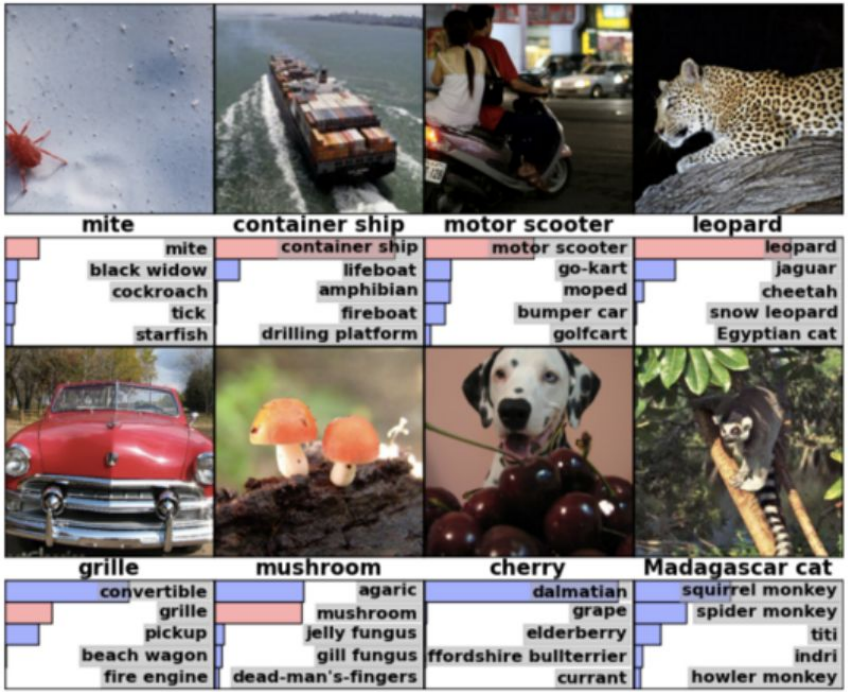
Regularization

- 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!

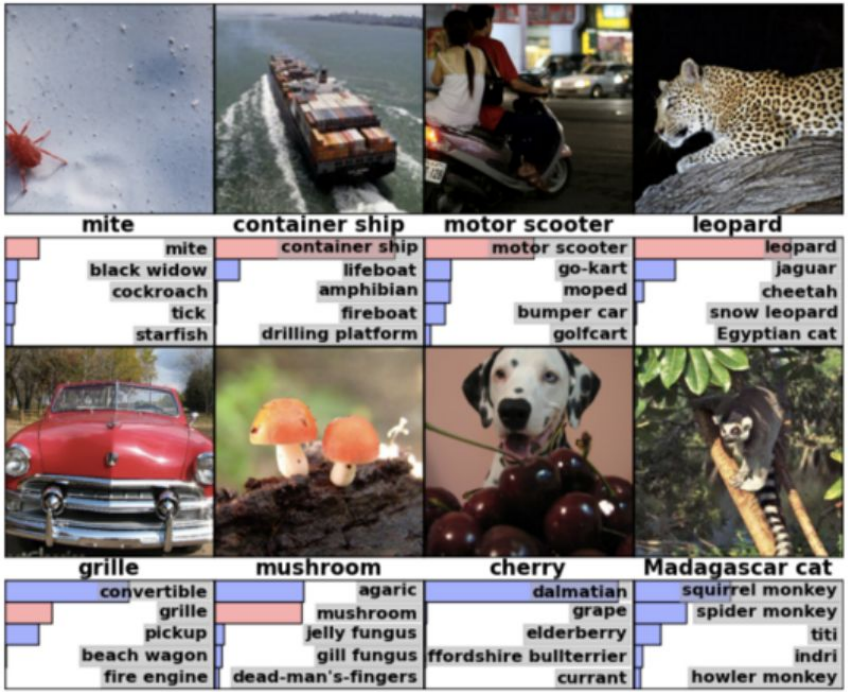


Krizhevsky et al., 2012

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



Krizhevsky et al., 2012

2011: Language Modeling

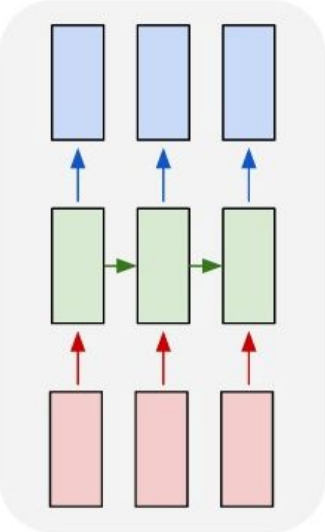
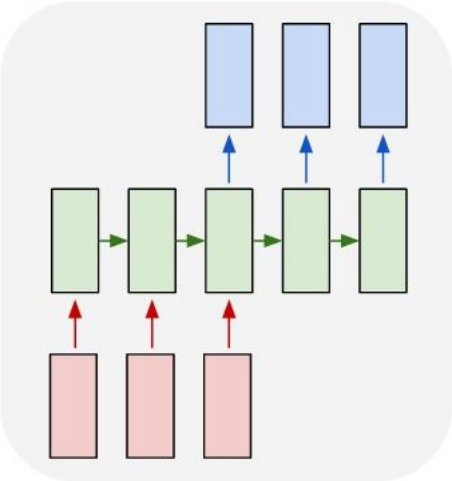


Table 2: *Comparison of various configurations of RNN LMs and combinations with backoff models while using 6.4M words in training data (WSJ DEV).*

	PPL		WER	
Model	RNN	RNN+KN	RNN	RNN+KN
KN5 - baseline	-	221	-	13.5
RNN 60/20	229	186	13.2	12.6
RNN 90/10	202	173	12.8	12.2
RNN 250/5	173	155	12.3	11.7
RNN 250/2	176	156	12.0	11.9
RNN 400/10	171	152	12.5	12.1
3xRNN static	151	143	11.6	11.3
3xRNN dynamic	128	121	11.3	11.1

Mikolov et al., 2011

2014: Machine Translation



Method	test BLEU score (ntst14)
Bahdanau et al. [2]	28.45
Baseline System [29]	33.30
Single forward LSTM, beam size 12	26.17
Single reversed LSTM, beam size 12	30.59
Ensemble of 5 reversed LSTMs, beam size 1	33.00
Ensemble of 2 reversed LSTMs, beam size 12	33.27
Ensemble of 5 reversed LSTMs, beam size 2	34.50
Ensemble of 5 reversed LSTMs, beam size 12	**34.81**

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.

Sutskever et al., 2014

2014: Dependency Parsing

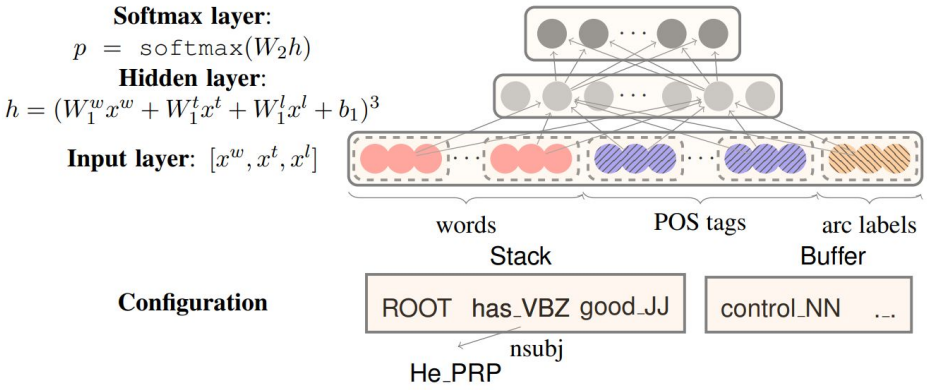


Figure 2: Our neural network architecture.

Parser	Dev UAS	Dev LAS	Test UAS	Test LAS	Speed (sent/s)
standard	90.2	87.8	89.4	87.3	26
eager	89.8	87.4	89.6	87.4	34
Malt:sp	89.8	87.2	89.3	86.9	469
Malt:eager	89.6	86.9	89.4	86.8	448
MSTParser	91.4	88.1	90.7	87.6	10
Our parser	**92.0**	**89.7**	**91.8**	**89.6**	**654**

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

Parser	Dev UAS	Dev LAS	Test UAS	Test LAS	Speed (sent/s)
standard	82.4	80.9	82.7	81.2	72
eager	81.1	79.7	80.3	78.7	80
Malt:sp	82.4	80.5	82.4	80.6	420
Malt:eager	81.2	79.3	80.2	78.4	393
MSTParser	**84.0**	82.1	83.0	81.2	6
Our parser	**84.0**	**82.4**	**83.9**	**82.4**	**936**

Table 6: Accuracy and parsing speed on CTB.

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

Pretrain & Finetune

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?

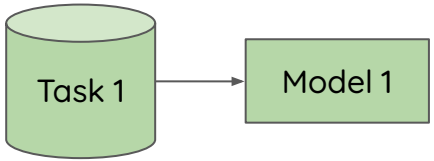
The “Pretrain & Finetune” Paradigm

- 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

Standard

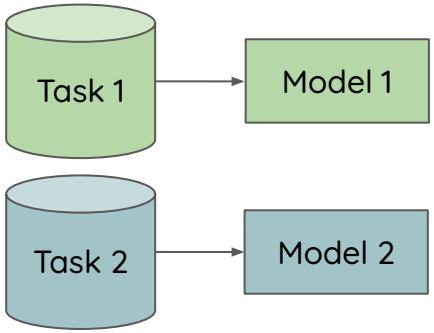


Pretrain & finetune

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

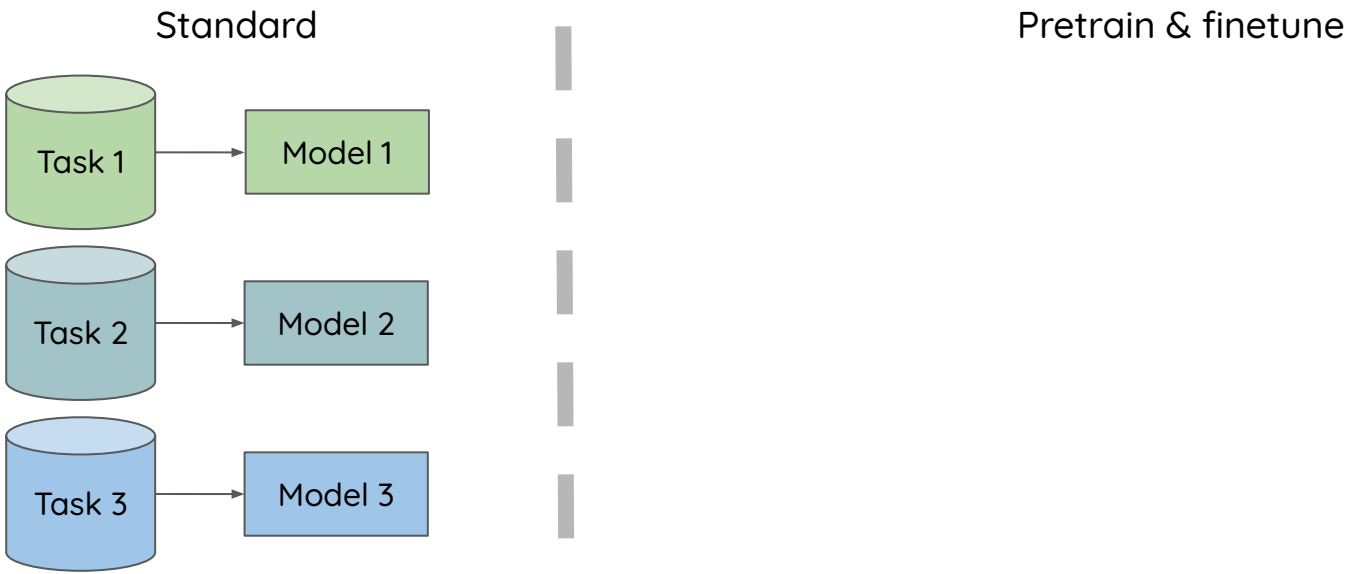
Standard



Pretrain & finetune

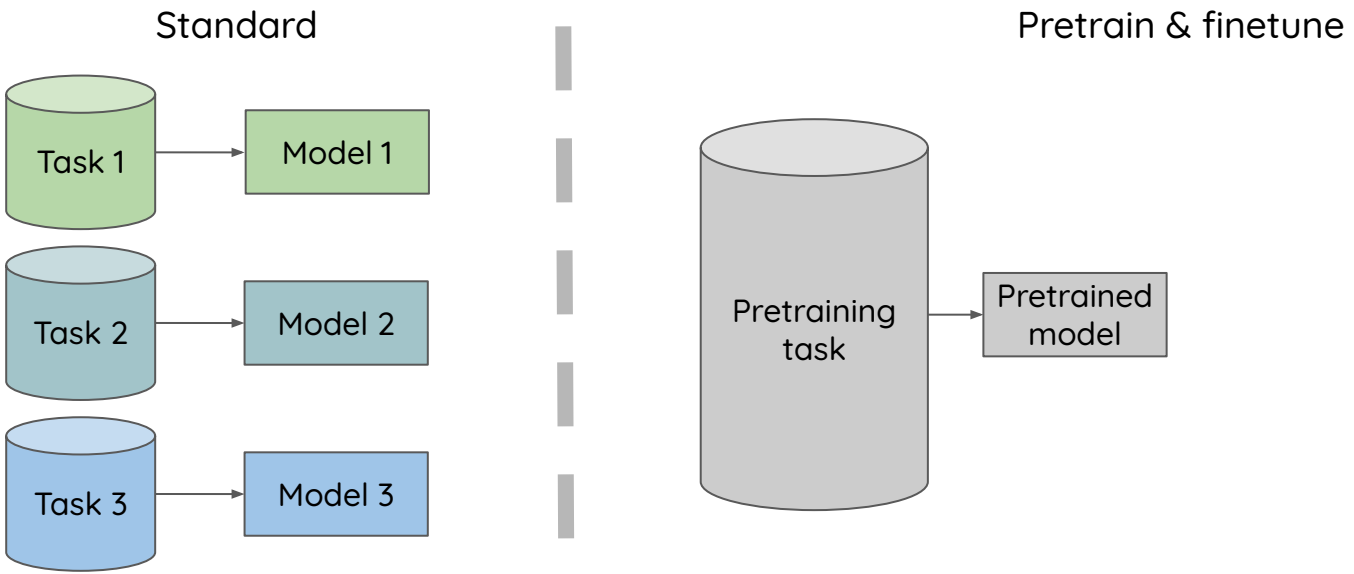
The “Pretrain & Finetune” Paradigm

- Just the standard pipeline with extra steps?
- NO: using a pre-trained model makes it possible to use much less data during fine tuning



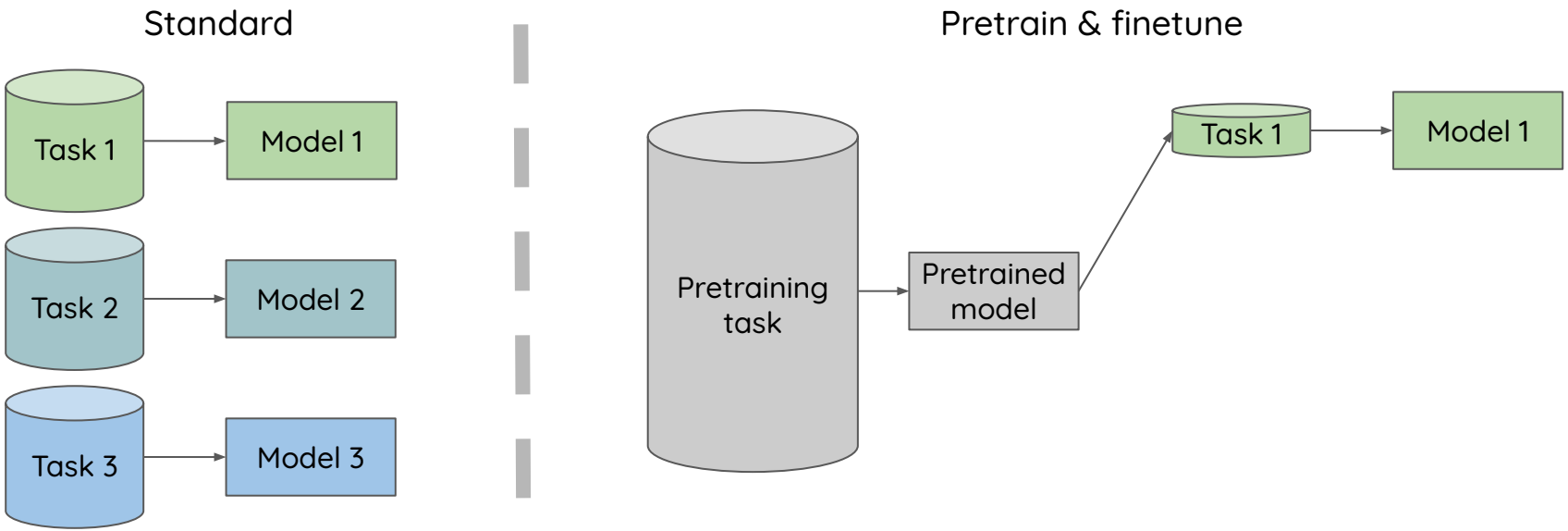
The “Pretrain & Finetune” Paradigm

- Just the standard pipeline with extra steps?
- NO: using a pre-trained model makes it possible to use much less data during fine tuning



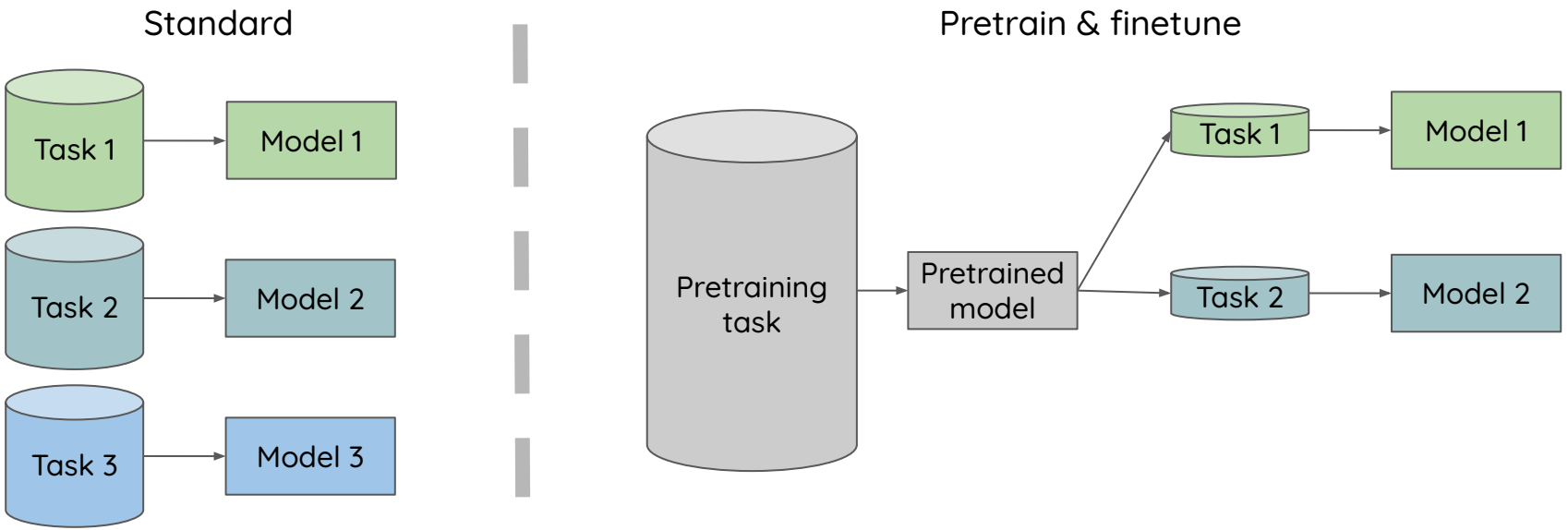
The “Pretrain & Finetune” Paradigm

- Just the standard pipeline with extra steps?
- NO: using a pre-trained model makes it possible to use much less data during fine tuning



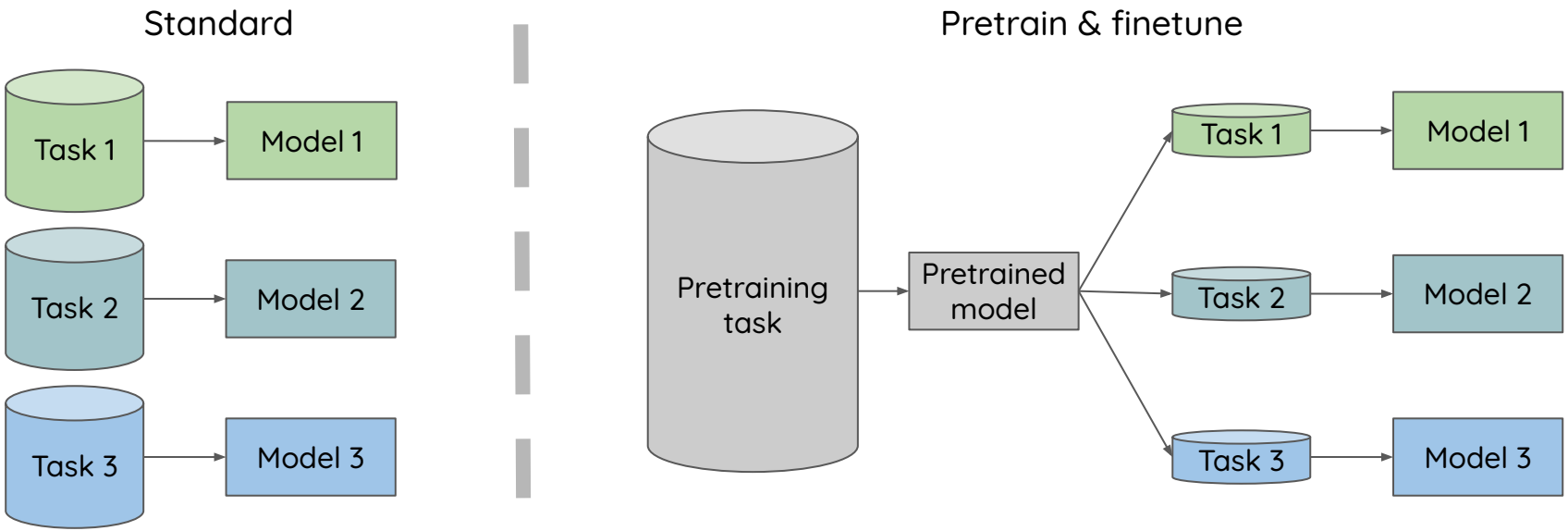
The “Pretrain & Finetune” Paradigm

- 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

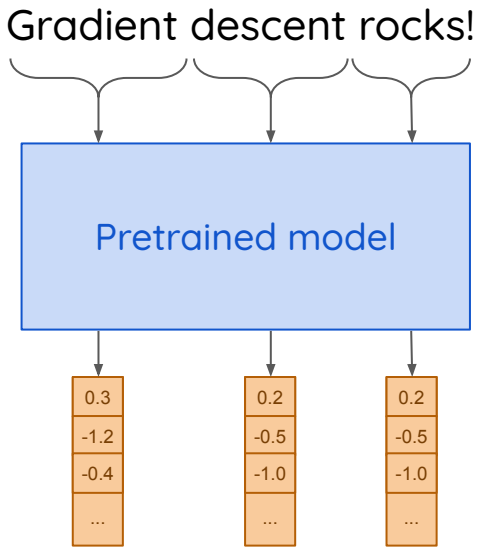


The “Pretrain & Finetune” Paradigm

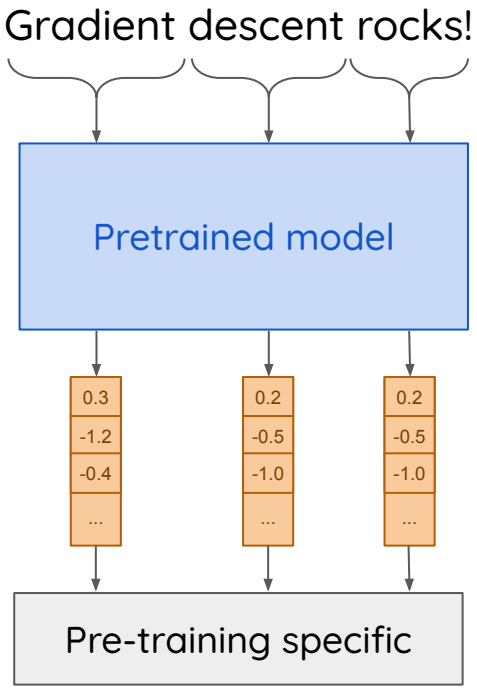
System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1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

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.[8] 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.

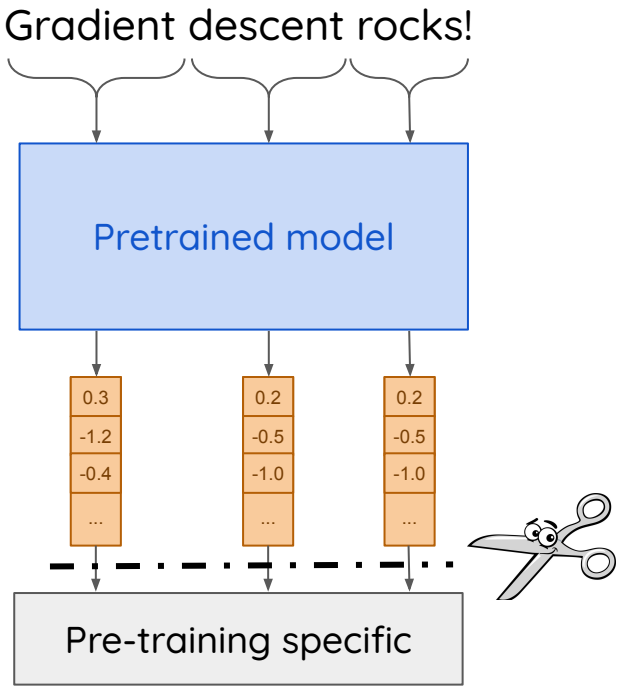
Pre-training: Architectures



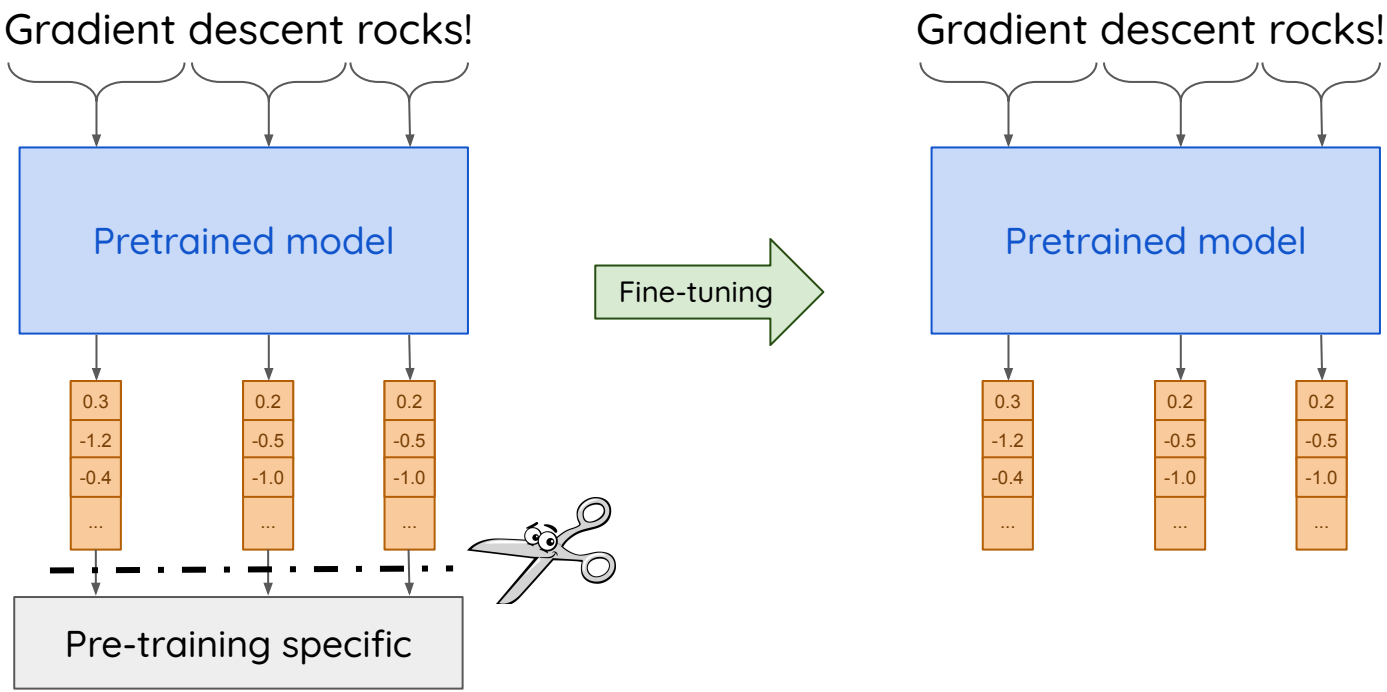
Pre-training: Architectures



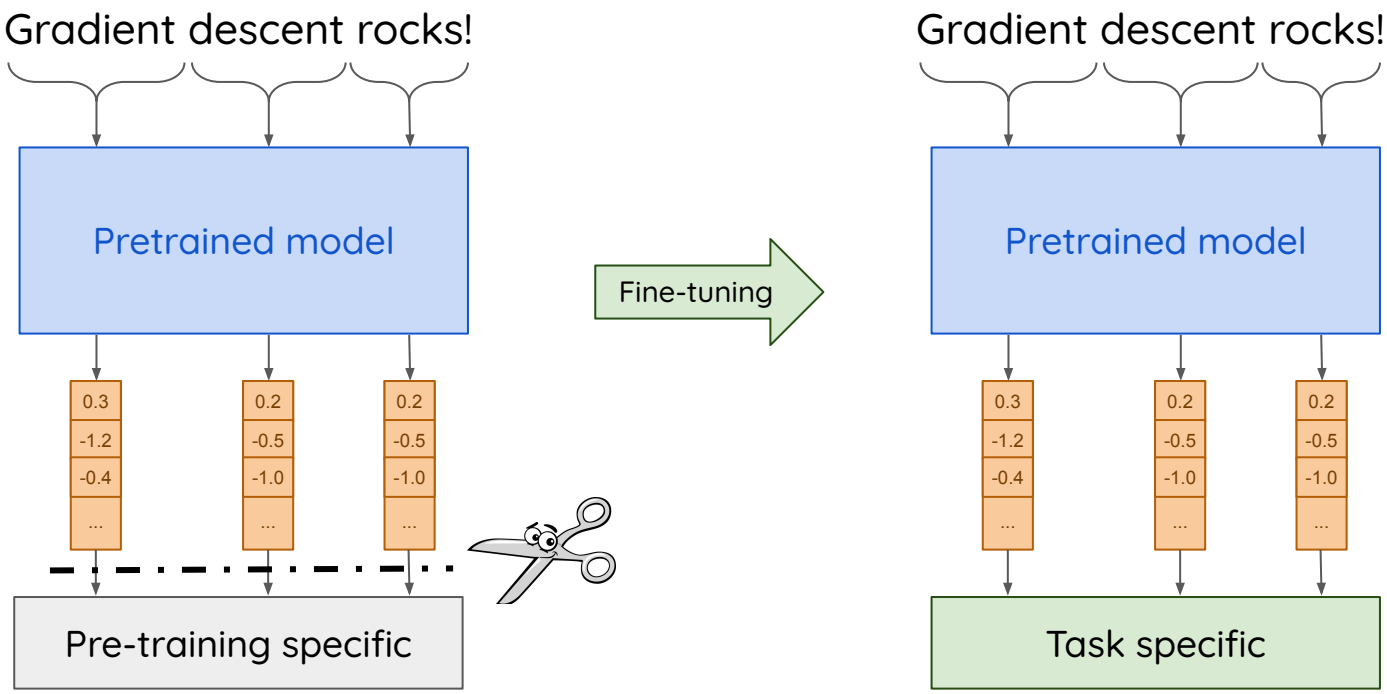
Pre-training: Architectures



Pre-training: Architectures



Pre-training: Architectures



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-frequencey 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-frequencey 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-frequencey 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

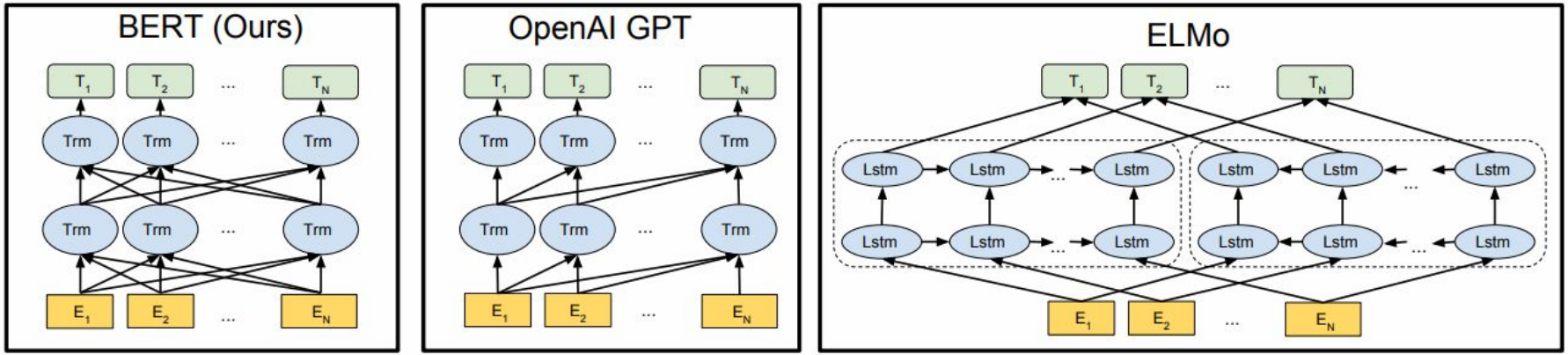


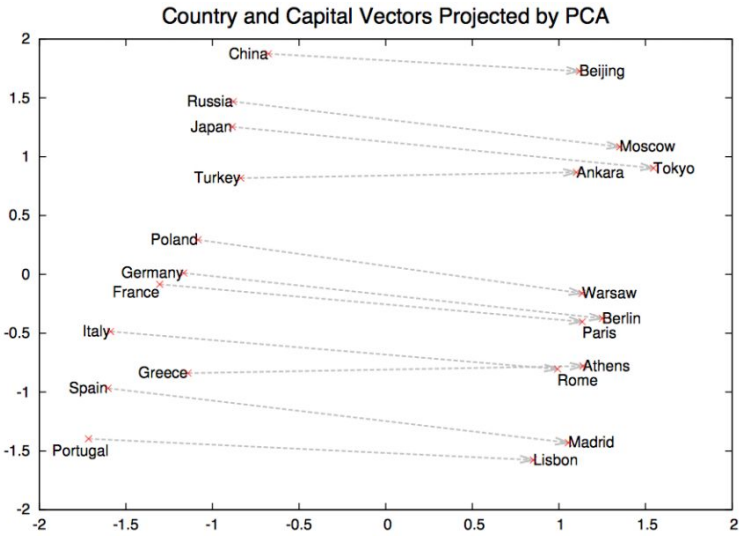
Figure 3: Differences in pre-training model architectures. BERT uses a bidirectional Transformer. OpenAI GPT uses a left-to-right Transformer. ELMo uses the concatenation of independently trained left-to-right and right-to-left LSTMs to generate features for downstream tasks. Among the three, only BERT 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

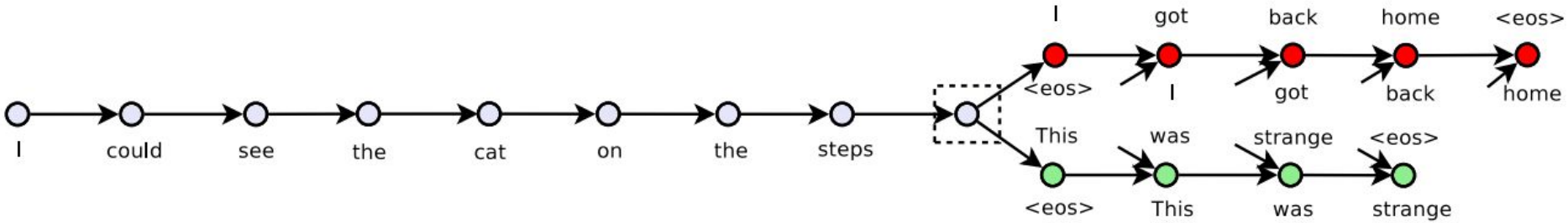
- What objective to use during pre-training?
  - Shouldn't require annotation
  - Should transfer well ("the model should learn something about language")

- 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

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

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

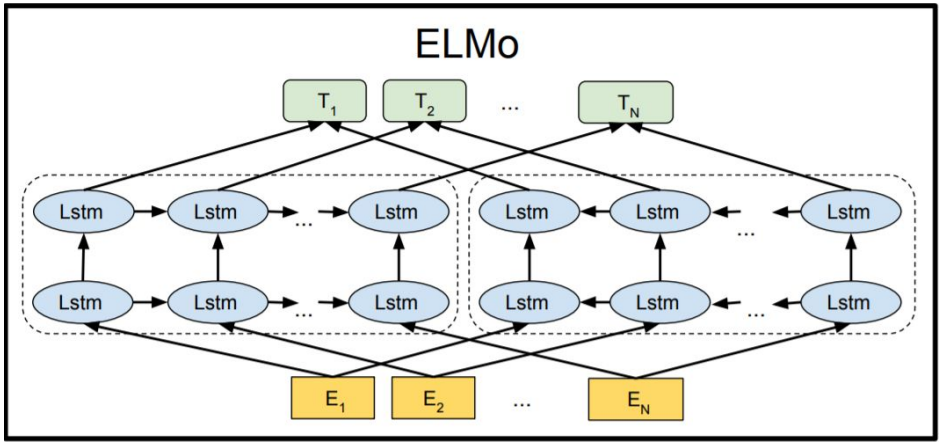
- What objective to use during pre-training?
  - Shouldn't require annotation
  - Should transfer well ("the model should learn something about language")

- 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

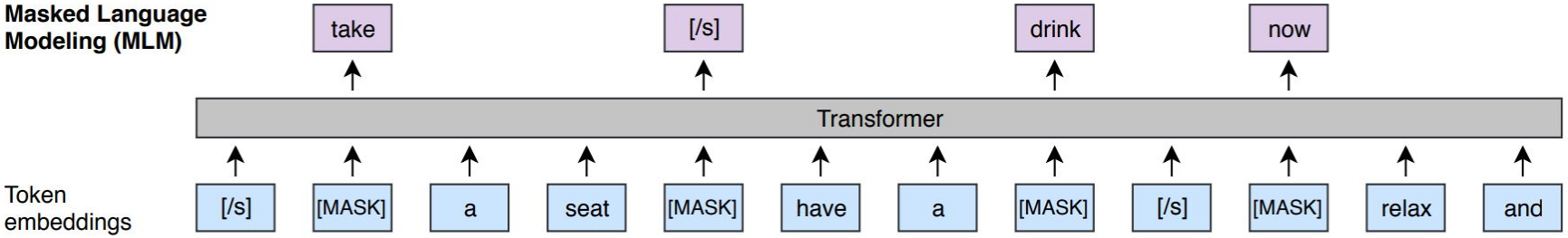
- What objective to use during pre-training?
  - Shouldn’t require annotation
  - Should transfer well (“the model should learn something about language”)

- 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

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

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.[8] 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)

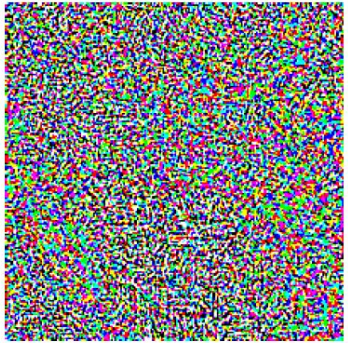
- This can be a problem!
  - For critical decision making, having a rationale can be necessary (eg. recidivism prediction, loan applications...) to ensure that there is no bias (eg. race, gender...)
  - Idiosyncratic behaviour (adversarial attacks)

Shortcomings: Adversarial Attacks

$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$



$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

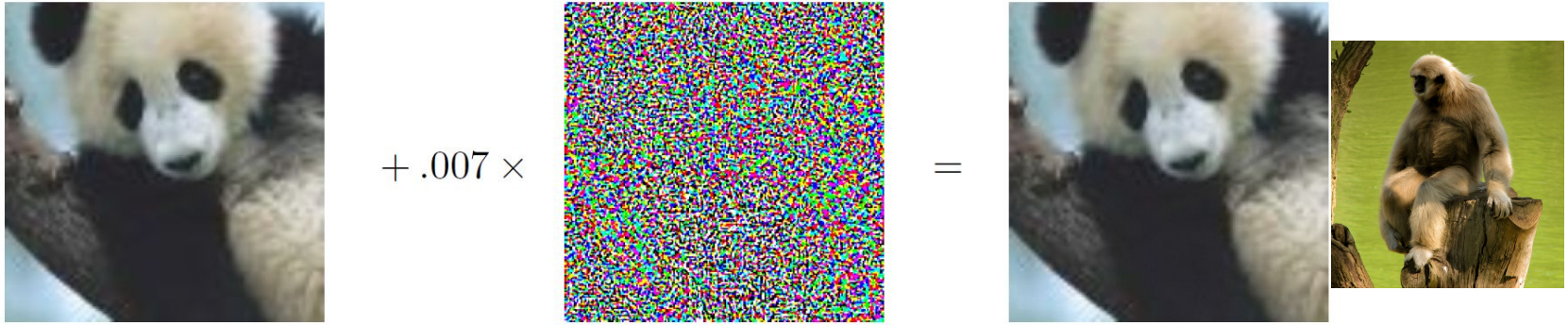
$=$

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"gibbon"
99.3 % confidence

Figure from Goodfellow et al. (2014)

Shortcomings: Adversarial Attacks



$\boldsymbol{x}$

"panda"
57.7% confidence

$+ .007 \times$

$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"nematode"
8.2% confidence

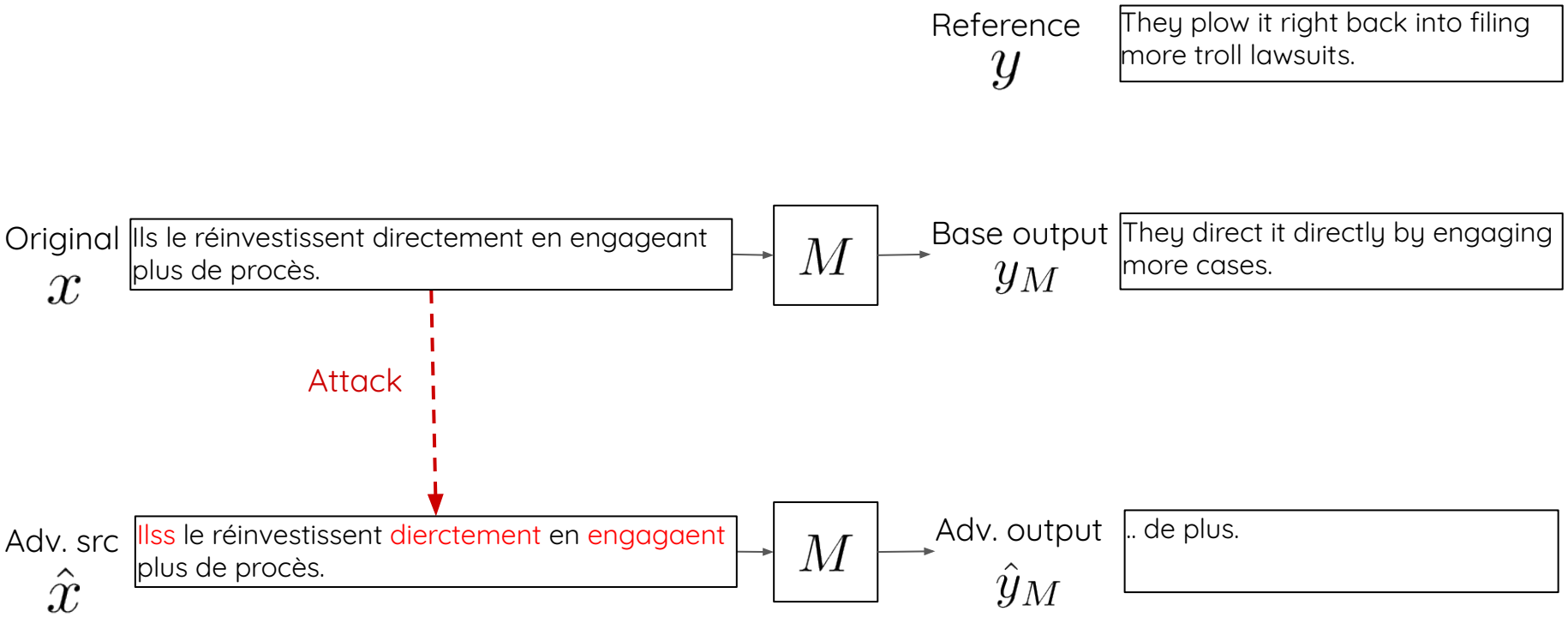
$=$

$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$

"gibbon"
99.3 % confidence

Figure from Goodfellow et al. (2014)

Shortcomings: Adversarial Attacks



"On Evaluation of Adversarial Perturbations for Sequence-to-Sequence Models" Michel et al. 2019

Understanding Deep Learning: Probing

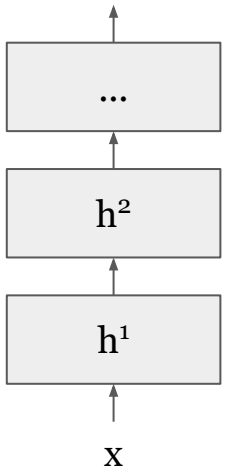
- What does my model learn?
  - Does it learn syntax/POS? Does it learn gender/number/tense?

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

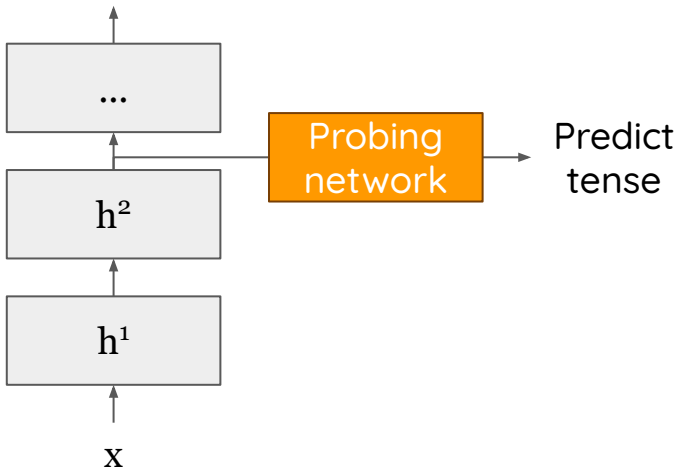
Understanding Deep Learning: Probing

- 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

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

Figure from Conneau et al. (2018)

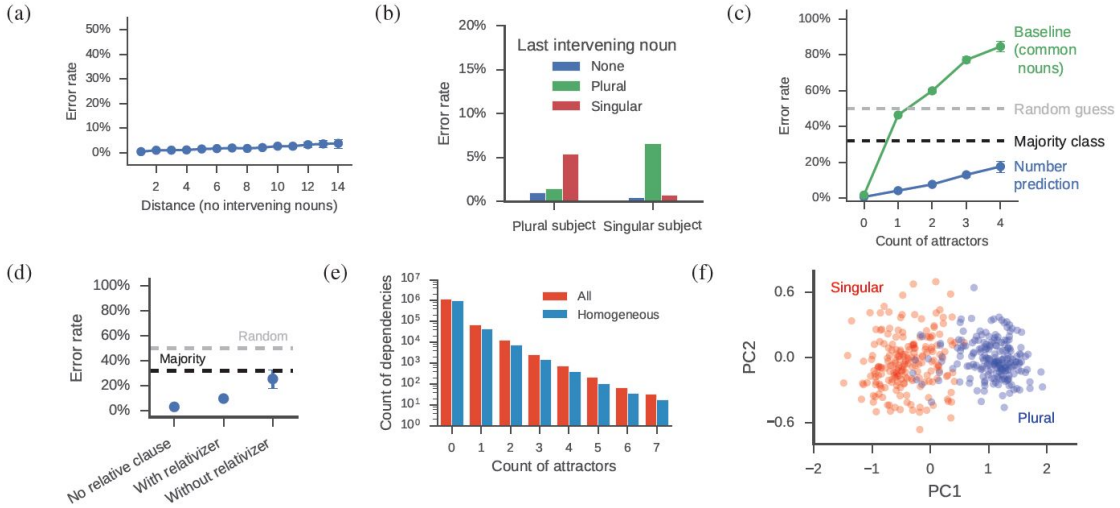
Understanding Deep Learning: Probing

- 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)

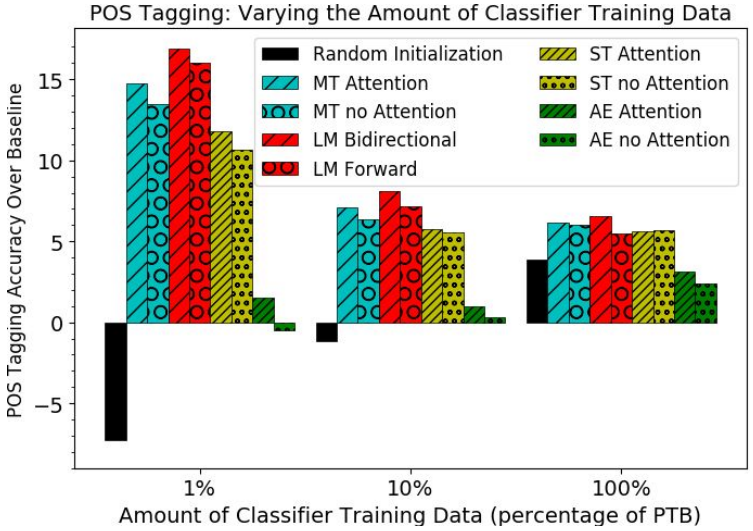
Understanding Deep Learning: Probing

- 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

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



Zhang & Bowman (2018)

Understanding Deep Learning: Explaining

- Why does my model return this prediction?
  - > “Easy” in a linear model: look at the active features and their weights

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

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 ~= 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… — Pretrained models help
  - > Problem for low resource languages — Cross/multilingual training
  - > Or tasks where annotation is expensive — Pretrained models help
- Compute intensive
  - > Training can take many GPU hours — Transformer, hardware…
  - > Large number of parameters — Distillation, pruning…
- Hard to interpret
  - > Hard to explain why the model is making a prediction — Word importance scores, etc…
  - > Not clear what the model is learning — Probing, etc…
- Robustness
  - > Can be brittle in the face of (well chosen) noise — Robustness
  - > Doesn’t work as well on different domains — Domain adaptation

Would you like to know more?

CMU CS 11-747, Spring 2020

Neural Networks for NLP

Questions?