Deep Learning for NLP

Antonio Valerio Miceli Barone

amiceli@inf.ed.ac.uk

12 May 2020

Overview

Previous part

- Supervised machine learning
  - Linear classifiers
    - Logistic regression
    - Softmax regression
  - Deep neural networks
    - Fully-connected
    - Convolutional
- Training algorithms
  - Stochastic gradient descent
  - Stochastic gradient descent with momentum
  - Backpropagation to compute gradient
- Processing text as input
  - Preprocessing
  - Vectorization

Language modeling

- Language modeling
  - Estimate the probability distribution of pieces of a sentence

Language modeling

- Language modeling
  - Estimate the probability distribution of pieces of a sentence or a document, tweet, etc.

Language modeling

- Language modeling
  - Estimate the probability distribution of pieces of a sentence or a document, tweet, etc.

$$p(T) = p(t_1, \ldots, t_M)$$

  - a sentence T of length M is a sequence of tokens $t_1, \ldots, t_n \in [1, \ldots, V]$

Language modeling

- Language modeling
  - Estimate the probability distribution of pieces of a sentence or a document, tweet, etc.

$$p(T) = p(t_1, \ldots, t_M)$$

  - a sentence T of length M is a sequence of tokens $t_1, \ldots, t_n \in [1, \ldots, V]$
  - Unsupervised machine learning problem

Language modeling

- Language modeling
  - Estimate the probability distribution of pieces of a sentence or a document, tweet, etc.

$$p(T) = p(t_1, \dots, t_M)$$

  - a sentence T of length M is a sequence of tokens $t_1, \dots, t_n \in [1, \dots, V]$
  - Unsupervised machine learning problem or "self-supervised"as they say now

Language modeling

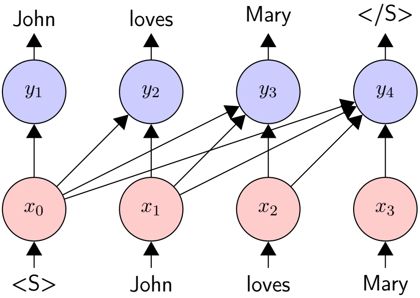
- Language modeling
  - Estimate the probability distribution of pieces of a sentence or a document, tweet, etc.

$$p(T) = p(t_1, \ldots, t_M)$$

  - a sentence T of length M is a sequence of tokens $t_1, \ldots, t_n \in [1, \ldots, V]$
  - Unsupervised machine learning problem or "self-supervised"as they say now
- Autoregressive decomposition: estimate the probability of each token given its prefix

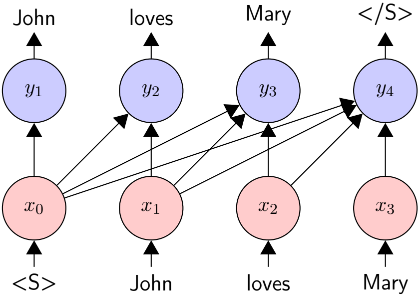
$$p(t_1, \ldots, t_M) = \prod_{i=1}^{M} p(t_i | t_1, \ldots, t_{i-1}) \qquad \text{(chain rule)}$$

Autoregressive Language modeling



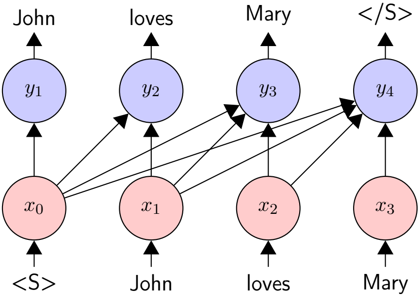
- Similar to a sequence tagging task
  - Each position is tagged with the next token
  - Special start-of-sentence and end-of-sentence tokens
  - Causality
    - each position only depends on the past

Autoregressive Language modeling



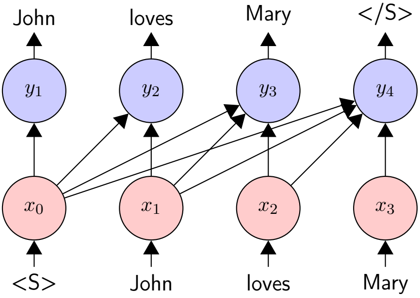
- Applications
  - Text generation

Autoregressive Language modeling



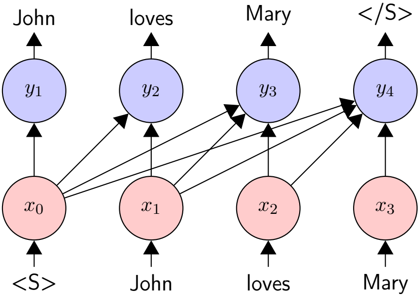
- Applications
  - Text generation usually from a prefix (e.g. autocomplete)

Autoregressive Language modeling



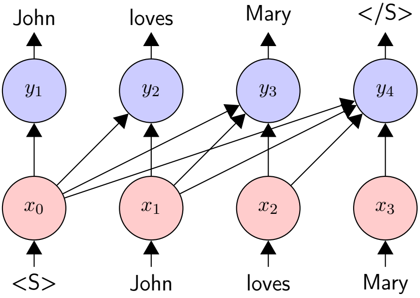
- Applications
  - Text generation usually from a prefix (e.g. autocomplete)
  - Text generation conditional on an input (e.g. image, text in another language)

Autoregressive Language modeling



- Applications
  - Text generation usually from a prefix (e.g. autocomplete)
  - Text generation conditional on an input (e.g. image, text in another language)
  - Scoring

Autoregressive Language modeling



- Applications
  - Text generation usually from a prefix (e.g. autocomplete)
  - Text generation conditional on an input (e.g. image, text in another language)
  - Scoring
  - Representation learning (embeddings)

Language modeling with neural networks

- Estimate probabilities with a neural network

$$p(t_i = k | t_1, \ldots, t_{i-1}) = f_k(t_1, \ldots, t_{i-1}; \theta)$$

- f is a neural network with parameters θ that computes a vector of V probabilities

Language modeling with neural networks

- Estimate probabilities with a neural network

$$p(t_i = k | t_1, \ldots, t_{i-1}) = f_k(t_1, \ldots, t_{i-1}; \theta)$$

- f is a neural network with parameters θ that computes a vector of V probabilities
- general structure:

$$f(t1, \ldots, t_{i-1}) = \text{softmax}(Proj(Seq(Emb(t_1), \ldots, Emb(t_{i-1}))))$$

  - $Emb(k) = W^{Emb}_{:,k}$: word embeddings $W^{Emb} \in \mathcal{R}^{d \times V}$
  - $Seq(x_1, \ldots, x_{i-1})$: sequence combinator
  - $Proj(s) = W^{Out} \cdot s$: output projection $W^{Out} \in \mathcal{R}^{V \times d}$.

Language modeling with neural networks

- Estimate probabilities with a neural network

$$p(t_i = k | t_1, \ldots, t_{i-1}) = f_k(t_1, \ldots, t_{i-1}; \theta)$$

- f is a neural network with parameters θ that computes a vector of V probabilities
- general structure:

$$f(t1, \ldots, t_{i-1}) = \text{softmax}(Proj(Seq(Emb(t_1), \ldots, Emb(t_{i-1}))))$$

  - $Emb(k) = W^{Emb}_{:,k}$: word embeddings $W^{Emb} \in \mathcal{R}^{d \times V}$
  - $Seq(x_1, \ldots, x_{i-1})$: sequence combinator
  - $Proj(s) = W^{Out} \cdot s$: output projection $W^{Out} \in \mathcal{R}^{V \times d}$.
    - usually $W^{Out} = \text{transpose}(W^{Emb})$ [Press and Wolf, 2017]

Training

- Maximum Likelihood Estimation
  - maximize the log-likelihood of the training set $\{T^{(j)}\}$ under the model

$$\underset{\theta}{\operatorname{argmax}} \sum_{j} \sum_{i=1}^{n^{(j)}} \log p(t_i^{(j)} | t_1^{(j)}, \dots, t_{i-1}^{(j)})$$

  - mini-batch stochastic gradient descent
  - adaptive learning rate and momentum (e.g. RMSProp, Adam)

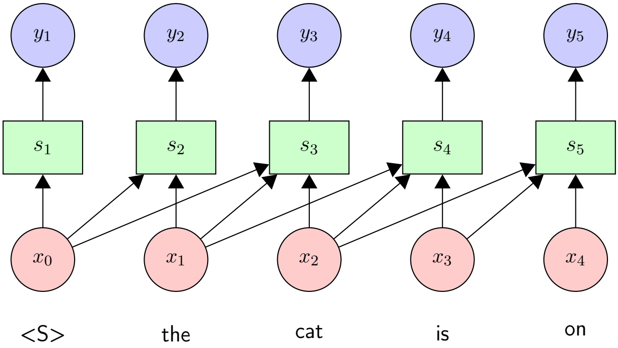
Training

- Maximum Likelihood Estimation
  - maximize the log-likelihood of the training set $\{T^{(j)}\}$ under the model

$$\operatorname*{argmax}_{\theta} \sum_{j} \sum_{i=1}^{n^{(j)}} \log p(t_i^{(j)} | t_1^{(j)}, \ldots, t_{i-1}^{(j)})$$

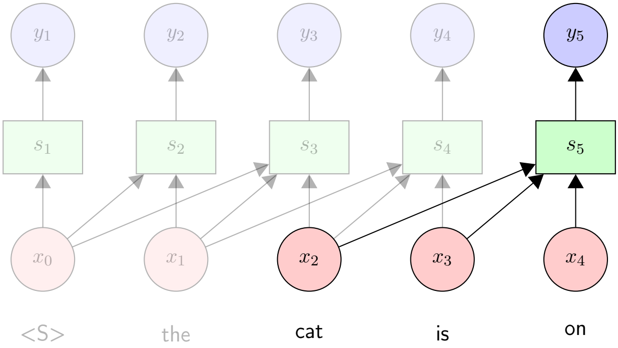
  - mini-batch stochastic gradient descent
  - adaptive learning rate and momentum (e.g. RMSProp, Adam)
  - other training criteria can be used (e.g. reinforcement learning, GANs)
    - in practice it's hard to do better than MLE

Convolutional language model



Fixed width sliding window of lenth L

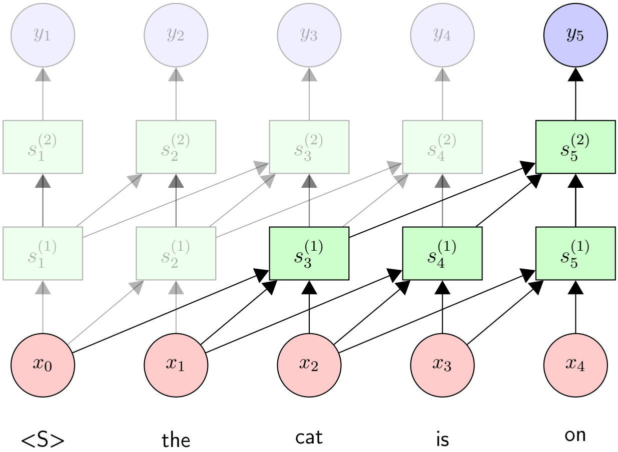
Convolutional language model



Fixed width sliding window of lenth L

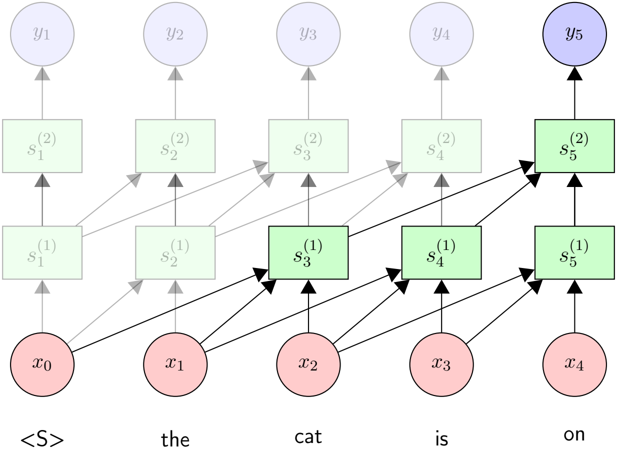
$$\begin{aligned} Seq(x_1, \ldots, x_{i-1}) &= Seq(x_{i-L}, \ldots, x_{i-1}) \\ &= \mathrm{ReLU}(b^{conv} + \sum_{j=1}^{L} W^{conv}_{:,:,j} \cdot x_{i-j}) \end{aligned}$$

Convolutional language model



Multiple layers increase both depth and window size

Convolutional language model



- pro: training can be parallelized over words
- con: strong Markovian independence assumption

Convolutional language model

An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling

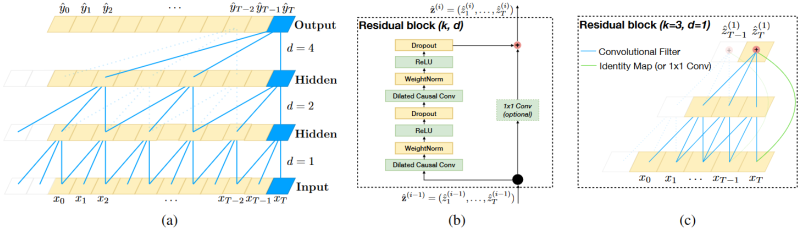


Figure 1. Architectural elements in a TCN. (a) A dilated causal convolution with dilation factors $d = 1, 2, 4$ and filter size $k = 3$. The receptive field is able to cover all values from the input sequence. (b) TCN residual block. An 1x1 convolution is added when residual input and output have different dimensions. (c) An example of residual connection in a TCN. The blue lines are filters in the residual function, and the green lines are identity mappings.

- Extensions [Bai et al., 2018]
  - residual connections
  - dilated convolutions
  - normalization layers
    - weight norm, layer norm, batch norm
  - dropout

Convolutional language model

An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling

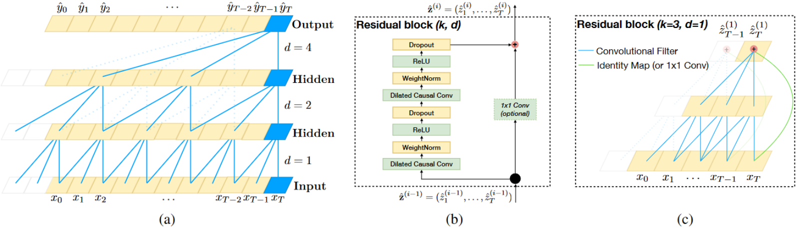


Figure 1. Architectural elements in a TCN. (a) A dilated causal convolution with dilation factors $d = 1, 2, 4$ and filter size $k = 3$. The receptive field is able to cover all values from the input sequence. (b) TCN residual block. An 1x1 convolution is added when residual input and output have different dimensions. (c) An example of residual connection in a TCN. The blue lines are filters in the residual function, and the green lines are identity mappings.

- Dilated convolutions
  - skip over words

$$\sum_{j=1}^{L} W^{conv}_{:,:,j} \cdot x_{i-(j \cdot D)}$$

  - dilation factor D

Convolutional language model

An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling

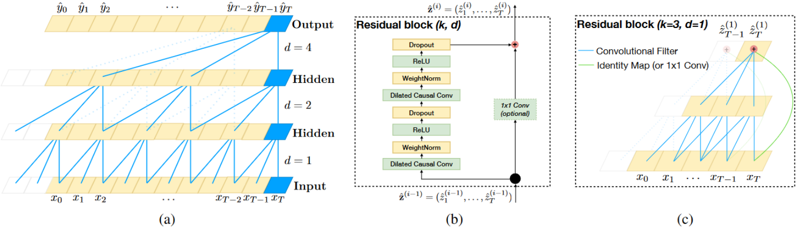


Figure 1. Architectural elements in a TCN. (a) A dilated causal convolution with dilation factors $d = 1, 2, 4$ and filter size $k = 3$. The receptive field is able to cover all values from the input sequence. (b) TCN residual block. An 1x1 convolution is added when residual input and output have different dimensions. (c) An example of residual connection in a TCN. The blue lines are filters in the residual function, and the green lines are identity mappings.

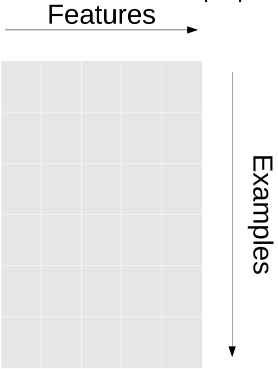
- Dilated convolutions
  - skip over words

$$\sum_{j=1}^{L} W^{conv}_{:,:,j} \cdot x_{i-(j \cdot D)}$$

  - dilation factor D
- Increase dilation exponentially over layers

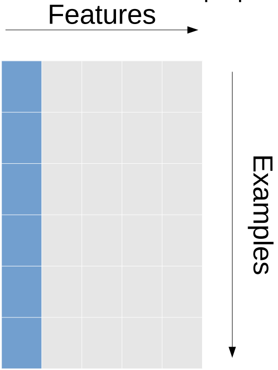
Excursus: normalization

- In a linear model is beneficial to normalize the input
  - Can be done as a preprocessing step



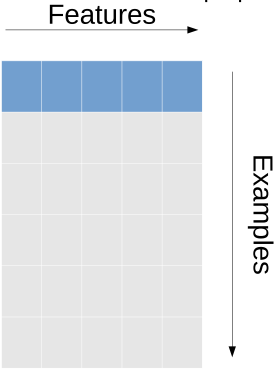
  -

Excursus: normalization

- In a linear model is beneficial to normalize the input
  - Can be done as a preprocessing step
  - 

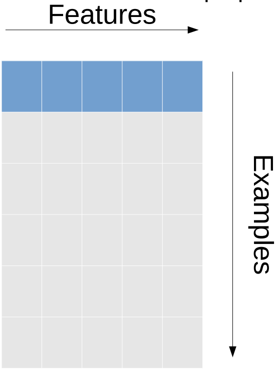
  - E.g. make each feature have mean 0 and sd 1 over the training set

Excursus: normalization

- In a linear model is beneficial to normalize the input
  - Can be done as a preprocessing step
  - 

  - E.g. make each feature have mean 0 and sd 1 over the training set
  - Or make each example have mean 0 and sd 1 over the features

Excursus: normalization

- In a linear model is beneficial to normalize the input
  - Can be done as a preprocessing step
  - 

  - E.g. make each feature have mean 0 and sd 1 over the training set
  - Or make each example have mean 0 and sd 1 over the features
- What about deep models?

Excursus: normalization

- What about deep models?
  - In a deep neural networks the activation and gradient statistics can vary greatly between layers

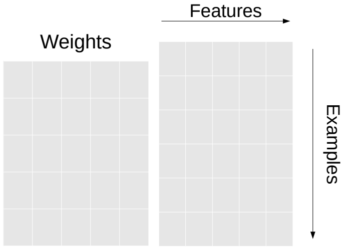
Excursus: normalization

- What about deep models?
  - In a deep neural networks the activation and gradient statistics can vary greatly between layers
  - It can cause numerical instabilities
  - Training is slow of diverges
  - Poor generalization
  - It get worse as training progresses

Excursus: normalization

- What about deep models?
  - In a deep neural networks the activation and gradient statistics can vary greatly between layers
  - It can cause numerical instabilities
  - Training is slow of diverges
  - Poor generalization
  - It get worse as training progresses
- Solution: normalization layers

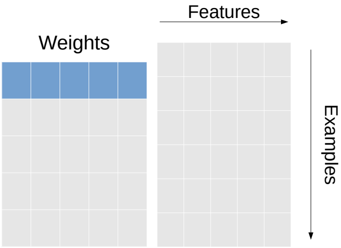
Excursus: normalization



$$z = W \cdot X + b$$

- Base linear layer

Excursus: normalization

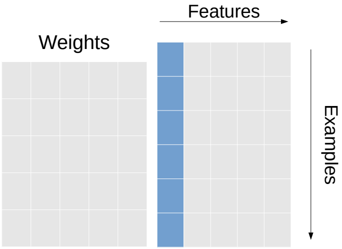


$$z = \hat{W} \cdot X + b$$

$$\hat{W} = g \cdot \text{NormalizeRows}(W)$$

- Weight normalization
  - Normalize the rows of the weight matrix L2 norm equal to 1
  - Multiply by a trainable gain $g \in \mathcal{R}^d$
  - g is initialized to 1

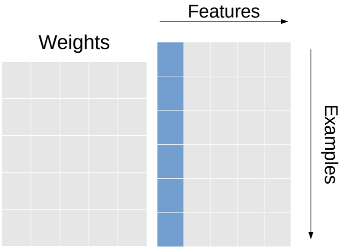
Excursus: normalization



$$z = W \cdot g \cdot \mathsf{NormalizeCols}(X) + b$$

- Batch normalization
  - Normalize the features over a minibatch to mean 0 and sd. 1
  - Multiply by a trainable gain $g \in \mathcal{R}^d$ and add a trainable bias b
  - g is initialized to 1, b is initalized to 0

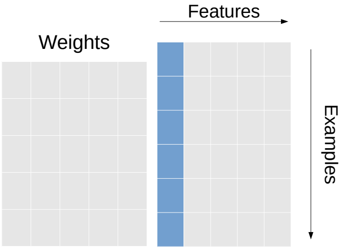
Excursus: normalization



$$z = W \cdot g \cdot \text{NormalizeCols}(X) + b$$

- Batch normalization
  - Normalize the features over a minibatch to mean 0 and sd. 1
  - Multiply by a trainable gain $g \in \mathcal{R}^d$ and add a trainable bias b
  - g is initialized to 1, b is initalized to 0
  - Save running mean and sd for inference

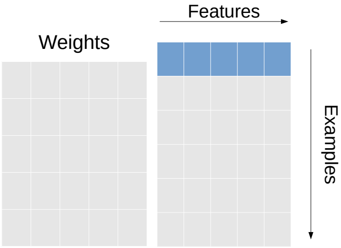
Excursus: normalization



$$z = W \cdot g \cdot \text{NormalizeCols}(X) + b$$

- Batch normalization
  - Normalize the features over a minibatch to mean 0 and sd. 1
  - Multiply by a trainable gain $g \in \mathcal{R}^d$ and add a trainable bias b
  - g is initialized to 1, b is initalized to 0
  - Save running mean and sd for inference
  - Note: in some models W is inside NormalizeCols

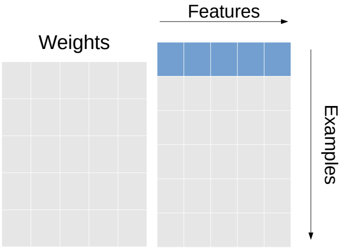
Excursus: normalization



$$z = W \cdot g \cdot \text{NormalizeRows}(X) + b$$

- Layers normalization
  - Normalize the examples over the features to mean 0 and sd. 1
  - Multiply by a trainable gain $g \in \mathcal{R}^d$ and add a trainable bias b
  - g is initialized to 1, b is initalized to 0

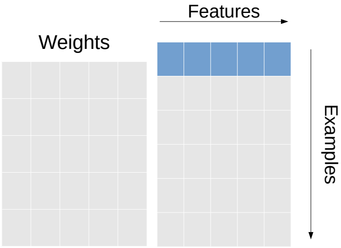
Excursus: normalization



$$z = W \cdot g \cdot \text{NormalizeRows}(X) + b$$

- Layers normalization
  - Normalize the examples over the features to mean 0 and sd. 1
  - Multiply by a trainable gain $g \in \mathcal{R}^d$ and add a trainable bias b
  - g is initialized to 1, b is initalized to 0
  - Training and inference are the same

Excursus: normalization



$$z = W \cdot g \cdot \text{NormalizeRows}(X) + b$$

- Layers normalization
  - Normalize the examples over the features to mean 0 and sd. 1
  - Multiply by a trainable gain $g \in \mathcal{R}^d$ and add a trainable bias b
  - g is initialized to 1, b is initalized to 0
  - Training and inference are the same
  - Note: in some models W is inside NormalizeRows

Excursus: normalization

- Weight normalization
  - Does not guarantee bounded activations
  - Exact reparametrization of the linear layer
  - Train and inference are the same

Excursus: normalization

- Weight normalization
  - Does not guarantee bounded activations
  - Exact reparametrization of the linear layer
  - Train and inference are the same
- Batch normalization
  - Guarantees bounded activations at training time
  - Exact reparametrization of the linear layer
  - Train and inference are the different
  - Common in image processing

Excursus: normalization

- Weight normalization
  - Does not guarantee bounded activations
  - Exact reparametrization of the linear layer
  - Train and inference are the same
- Batch normalization
  - Guarantees bounded activations at training time
  - Exact reparametrization of the linear layer
  - Train and inference are the different
  - Common in image processing
- Layer normalization
  - Guarantees bounded activations always
  - Non-linear function
  - Train and inference are the same
  - Common in NLP

Excursus: Dropout

- Dropout regularization
- During training
  - Randomly choose activation components with probability p and set them to 0
  - Multiply the other by $1/p$ to keep the mean approx. the same

$$\text{Dropout}(z) = \frac{\text{mask}}{p} \cdot z$$

  - where mask $\in \mathcal{R}^d$, $\text{mask}_j \sim \text{Bernoulli}(1 - p)$
  - Independent over components and examples
  - p is a hyperparameter, usually between 0.1 and 0.5
- During inference
  - Do nothing

$$\text{Dropout}(z) = z$$

Excursus: Dropout

- Dropout regularization
- During training
  - Randomly choose activation components with probability p and set them to 0
  - Multiply the other by $1/p$ to keep the mean approx. the same

$$\text{Dropout}(z) = \frac{\text{mask}}{p} \cdot z$$

  - where $\text{mask} \in \mathcal{R}^d$, $\text{mask}_j \sim \text{Bernoulli}(1-p)$
  - Independent over components and examples
  - p is a hyperparameter, usually between 0.1 and 0.5
- During inference
  - Do nothing

$$\text{Dropout}(z) = z$$

- Dropoout is usually the strongest regularization method

Excursus: Dropout

- Dropout regularization
- During training
  - Randomly choose activation components with probability p and set them to 0
  - Multiply the other by $1/p$ to keep the mean approx. the same

$$\text{Dropout}(z) = \frac{\text{mask}}{p} \cdot z$$

  - where $\text{mask} \in \mathcal{R}^d$, $\text{mask}_j \sim \text{Bernoulli}(1-p)$
  - Independent over components and examples
  - p is a hyperparameter, usually between 0.1 and 0.5
- During inference
  - Do nothing

$$\text{Dropout}(z) = z$$

- Dropoout is usually the strongest regularization method
- Can be also be applied to whole tokens (Word dropout)

Excursus: Dropout

- Dropout regularization
- During training
  - Randomly choose activation components with probability p and set them to 0
  - Multiply the other by $1/p$ to keep the mean approx. the same

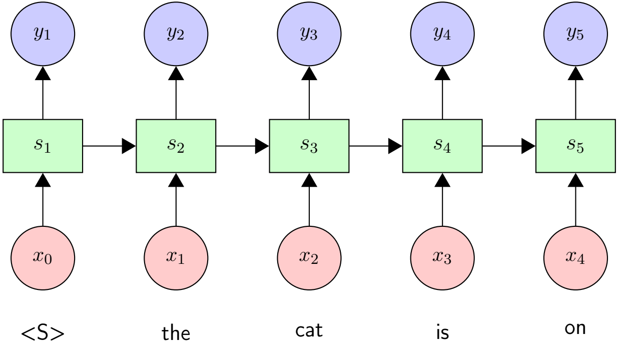
$$\text{Dropout}(z) = \frac{\text{mask}}{p} \cdot z$$

  - where $\text{mask} \in \mathcal{R}^d$, $\text{mask}_j \sim \text{Bernoulli}(1-p)$
  - Independent over components and examples
  - p is a hyperparameter, usually between 0.1 and 0.5
- During inference
  - Do nothing

$$\text{Dropout}(z) = z$$

- Dropoout is usually the strongest regularization method
- Can be also be applied to whole tokens (Word dropout) or weights (Dropconnect)

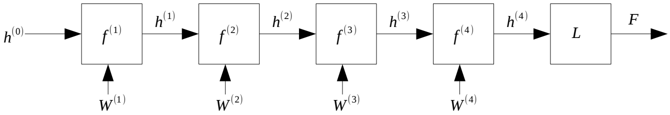
Recurrent language model [Mikolov et al., 2010]



Recurrent decomposition

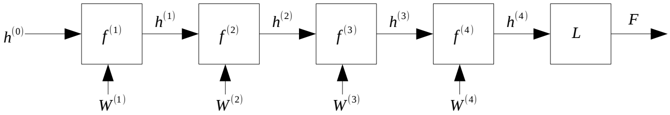
$$
\begin{aligned}
Seq(x_1, \ldots, x_{i-1}) &= \mathsf{RNN}(Seq(x_1, \ldots, x_{i-2}), x_{i-1}) \\
s_0 &= 0 \\
s_i &= \mathsf{RNN}(s_{i-1}, x_{i-1})
\end{aligned}
$$

Backpropagation algorithm



- Forward pass:
  - For k in range $[1, K]$
    - Store in memory $h^{(k-1)}$
    - $h^{(k)} := f^{(k)}(h^{(k-1)}, W^{(k)})$
  - Store in memory $h^{(K)}$
  - Return $L(h^{(K)})$

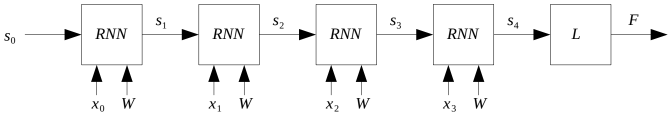
Backpropagation algorithm



- Forward pass:
  - For k in range $[1, K]$
    - Store in memory $h^{(k-1)}$
    - $h^{(k)} := f^{(k)}(h^{(k-1)}, W^{(k)})$
  - Store in memory $h^{(K)}$
  - Return $L(h^{(K)})$

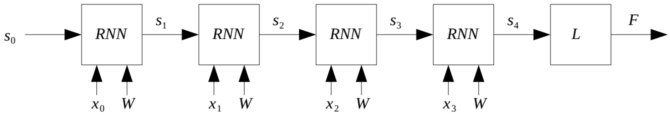
- Backward pass:
  - $h^{(K)} :=$ retrieve from memory
  - adjoint $:= \frac{\mathrm{d}L}{\mathrm{d}h^{(K)}}$
  - For k in range $[K, 1]$
    - $h^{(k-1)} :=$ retrieve from memory
    - $\frac{\mathrm{d}F}{\mathrm{d}W^{(k)}} :=$ adjoint $\cdot \frac{\mathrm{d}}{\mathrm{d}W^{(k)}} f^{(k)}(h^{(k-1)}, W^{(k)})$
    - adjoint $:=$ adjoint $\cdot \frac{\mathrm{d}}{\mathrm{d}h^{(k-1)}} f^{(k)}(h^{(k-1)}, W^{(k)})$
  - Return $\frac{\mathrm{d}F}{\mathrm{d}W^{(k)}}$ for all k

Backpropagation through time



- Forward pass:
  - For k in range $[1, M]$
    - Store in memory s_{k-1}
    - $s_k :=$ RNN(s_k, x_k, W)
  - Store in memory s_M
  - Return $L(s_M)$

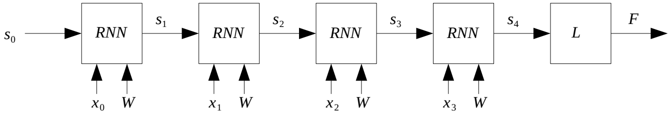
Backpropagation through time



- Forward pass:
  - For k in range $[1, M]$
    - Store in memory s_{k-1}
    - $s_k :=$ RNN(s_k, x_k, W)
  - Store in memory s_M
  - Return $L(s_M)$

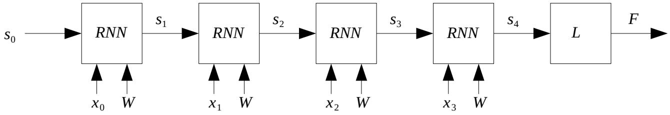
- Backward pass:
  - $\frac{\mathrm{d}F}{\mathrm{d}W} := 0$
  - $s_M :=$ retrieve from memory
  - adjoint $:= \frac{\mathrm{d}L}{\mathrm{d}s_M}$
  - For k in range $[M, 1]$
    - $s_{k-1} :=$ retrieve from memory
    - $\frac{\mathrm{d}F}{\mathrm{d}W} + =$ adjoint $\cdot \frac{\partial}{\partial W}RNN(s_{k-1}, x_k, W)$
    - adjoint $:=$ adjoint $\cdot \frac{\mathrm{d}}{\mathrm{d}s_{k-1}}RNN(s_{k-1}, x_k, W)$
  - Return $\frac{\mathrm{d}F}{\mathrm{d}W}$

Backpropagation through time



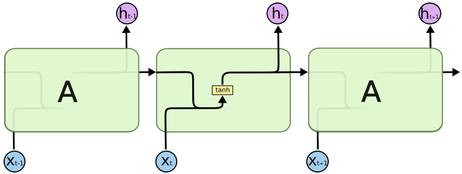
- Deep Learning frameworks handle arbitrary DAGs
- You can freely mix feed-forward and recurrent components
- But beware of memory usage

Backpropagation through time



- Deep Learning frameworks handle arbitrary DAGs
- You can freely mix feed-forward and recurrent components
- But beware of memory usage
- Truncated BPTT
  - Split the sequence up to a given maximum length and run the forward and backward passes on the chunks
  - But keep the intermediate state instead of resetting
  - Heuristic approximation, works well in practice

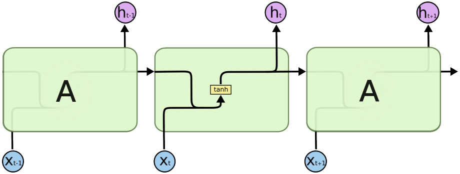
RNN variants



$$\mathsf{RNN}(s_{i-1}, x_{i-1}) = \mathsf{tanh}(W_{\mathsf{state}} \cdot s_{i-1} + W_{\mathsf{in}} \cdot x_{i-1} + b)$$

- Elman's RNN
  - Simple
  - Very similar to a MLP
  - In theory can compute any arbitrary FSA

RNN variants



$$\text{RNN}(s_{i-1}, x_{i-1}) = \tanh(W_{\text{state}} \cdot s_{i-1} + W_{\text{in}} \cdot x_{i-1} + b)$$

- Elman's RNN
  - Simple
  - Very similar to a MLP
  - In theory can compute any arbitrary FSA
  - But it's often hard to train
  - Especially on long sequences with long-term dependencies

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

Why is Elman's RNN hard to train?

- Consider a simplified version
  - No non-linearity
  - Scalar state and input

$$\mathsf{RNN}(s_{i-1}, x_{i-1}) = w \cdot s_{i-1} + x_{i-1}$$

Why is Elman's RNN hard to train?

- Consider a simplified version
  - No non-linearity
  - Scalar state and input

$$\mathsf{RNN}(s_{i-1}, x_{i-1}) = w \cdot s_{i-1} + x_{i-1}$$

- Assuming that $|x| < c$ the long-term behavior depends on the magnitude of w
- If $|w| < 1$
  - The state s is an exponential running average of the input x

Why is Elman's RNN hard to train?

- Consider a simplified version
  - No non-linearity
  - Scalar state and input

$$\text{RNN}(s_{i-1}, x_{i-1}) = w \cdot s_{i-1} + x_{i-1}$$

- Assuming that $|x| < c$ the long-term behavior depends on the magnitude of w
- If $|w| < 1$
  - The state s is an exponential running average of the input x (with reflection if w is negative)
  - Acceptable for short-term dependencies, but works poorly for long-term
  - In the backward pass the adjoints are also exponentially smoothed over time (vanishing gradients)

Why is Elman's RNN hard to train?

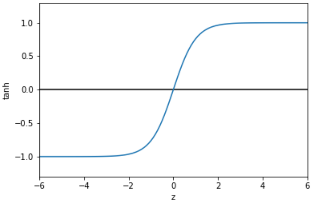
- Consider a simplified version
  - No non-linearity
  - Scalar state and input

$$\text{RNN}(s_{i-1}, x_{i-1}) = w \cdot s_{i-1} + x_{i-1}$$

- Assuming that $|x| < c$ the long-term behavior depends on the magnitude of w
- If $|w| < 1$
  - The state s is an exponential running average of the input x (with reflection if w is negative)
  - Acceptable for short-term dependencies, but works poorly for long-term
  - In the backward pass the adjoints are also exponentially smoothed over time (vanishing gradients)
- If $|w| > 1$
  - Unstable, the state explodes to $+\infty$ or $-\infty$

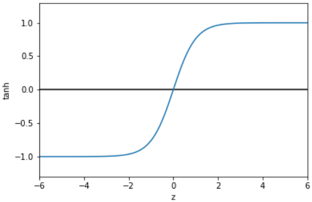
Why is Elman's RNN hard to train?

- We need a bounded activation function
- Can't use ReLU
- We use the hyperbolic tangent $\mathrm{RNN}(s_{i-1}, x_{i-1}) = \tanh(w \cdot s_{i-1} + x_{i-1})$



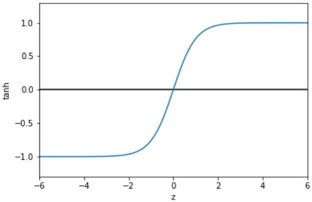
Why is Elman's RNN hard to train?

- We need a bounded activation function
- Can't use ReLU
- We use the hyperbolic tangent $\text{RNN}(s_{i-1}, x_{i-1}) = \tanh(w \cdot s_{i-1} + x_{i-1})$
  - Solves the exploding activations problem



Why is Elman's RNN hard to train?

- We need a bounded activation function
- Can't use ReLU
- We use the hyperbolic tangent $\text{RNN}(s_{i-1}, x_{i-1}) = \tanh(w \cdot s_{i-1} + x_{i-1})$
  - Solves the exploding activations problem
  - But can kill the gradients when saturated



RNN variants

- What about a residual connection?

$$\text{RNN}(s_{i-1}, x_{i-1}) = \tanh(W_{\text{state}} \cdot s_{i-1} + W_{\text{in}} \cdot x_{i-1} + b) + s_{i-1}$$

RNN variants

- What about a residual connection?

$$\text{RNN}(s_{i-1}, x_{i-1}) = \tanh(W_{\text{state}} \cdot s_{i-1} + W_{\text{in}} \cdot x_{i-1} + b) + s_{i-1}$$

  - Solves vaninshing gradients, but brings back exploding activations

Gated RNNs

- Bound the skip connection using another trainable function ("gate")

Gated RNNs

- Bound the skip connection using another trainable function ("gate")
- Tied Recurrent Highway Network

$$\tilde{s} = \mathsf{tanh}(W_{\mathsf{state}} \cdot s_{i-1} + W_{\mathsf{in}} \cdot x_{i-1} + b)$$
$$f = \sigma(W^f_{\mathsf{state}} \cdot s_{i-1} + W^f_{\mathsf{in}} \cdot x_{i-1} + b^f)$$
$$\mathsf{RHN}(s_{i-1}, x_{i-1}) = \tilde{s} \cdot (1 - f) + s_{i-1} \cdot f$$

  - $\tilde{s}$: "proposal"gate
  - f: "forget"gate

Gated RNNs

- Bound the skip connection using another trainable function ("gate")
- Tied Recurrent Highway Network

$$
\begin{aligned}
\tilde{s} &= \tanh(W_{\text{state}} \cdot s_{i-1} + W_{\text{in}} \cdot x_{i-1} + b) \\
f &= \sigma(W^{f}_{\text{state}} \cdot s_{i-1} + W^{f}_{\text{in}} \cdot x_{i-1} + b^{f}) \\
\text{RHN}(s_{i-1}, x_{i-1}) &= \tilde{s} \cdot (1 - f) + s_{i-1} \cdot f
\end{aligned}
$$

- $\tilde{s}$: "proposal"gate
- f: "forget"gate
  - Forget bias b^f is initialized > 0

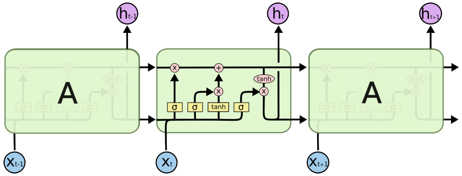
Gated RNNs

- Bound the skip connection using another trainable function ("gate")
- Tied Recurrent Highway Network

$$\tilde{s} = \mathsf{tanh}(W_{\mathsf{state}} \cdot s_{i-1} + W_{\mathsf{in}} \cdot x_{i-1} + b)$$
$$f = \sigma(W^f_{\mathsf{state}} \cdot s_{i-1} + W^f_{\mathsf{in}} \cdot x_{i-1} + b^f)$$
$$\mathsf{RHN}(s_{i-1}, x_{i-1}) = \tilde{s} \cdot (1 - f) + s_{i-1} \cdot f$$

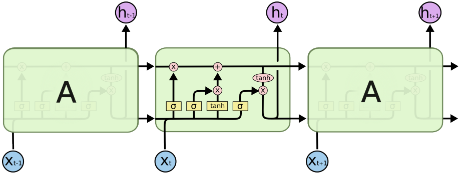
  - $\tilde{s}$: "proposal"gate
  - f: "forget"gate
    - Forget bias b^f is initialized > 0
- Works well, but it is neither the oldest nor the best gated RNN

RNN variants



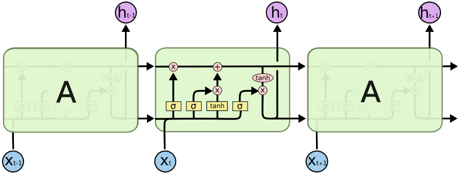
- Gated Recurrent Unit (GRU)
  - Like the RHN, but with an additional "reset"gate to flush unnecessary components of the state

RNN variants



- Gated Recurrent Unit (GRU)
  - Like the RHN, but with an additional "reset"gate to flush unnecessary components of the state
- Long-Short Term Memory (LSTM)
  - Oldest and most complex gated RNN
  - Two state vectors
  - Multiple gates
  - Computes FSA + counters
  - Empirically the strongest
  - But also has the most parameters

RNN variants



- Gated Recurrent Unit (GRU)
  - Like the RHN, but with an additional "reset"gate to flush unnecessary components of the state
- Long-Short Term Memory (LSTM)
  - Oldest and most complex gated RNN
  - Two state vectors
  - Multiple gates
  - Computes FSA + counters
  - Empirically the strongest
  - But also has the most parameters
- TL;DR Use a LSTM, unless you really want a small model

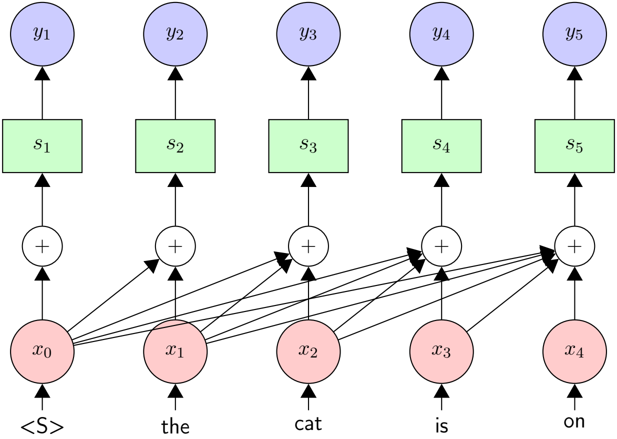
Recurrent language model

- pros & cons
  - pro: can capture long distance dependencies
  - pro: can represent arbitrary FSAs (+ counting)
  - con: inherently sequential even during training
  - con: hidden state can become a bottleneck

Recurrent language model

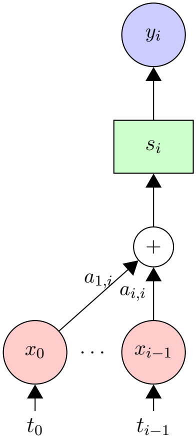
- pros & cons
  - pro: can capture long distance dependencies
  - pro: can represent arbitrary FSAs (+ counting)
  - con: inherently sequential even during training
  - con: hidden state can become a bottleneck
- extensions
  - stacked depth and transition depth [Miceli Barone et al., 2017]
  - residual connections
  - normalization layers (layer norm)
  - dropout
  - etc.

Transformer language model [Vaswani et al., 2017]



Causal self-attention

Transformer language model



causal self-attention

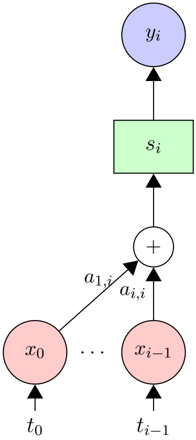
$$e_{j,i} = x_{j-1}^{\dagger} \cdot W^K \cdot W^Q \cdot x_{i-1} \text{ (dot product attention)}$$

$$a_{j,i} = \underset{j \le i}{\text{softmax}}(e_{j,i})$$

$$c_i = \sum_{j=1}^{i} a_{j,i} W^V \cdot x_{j-1}$$

$$s_i = b^{(2)} + W^{(2)} \cdot \text{ReLU}(b^{(1)} + W^{(1)} \cdot c_i)$$

Transformer language model



causal self-attention

$$e_{j,i} = x_{j-1}^{\dagger} \cdot W^K \cdot W^Q \cdot x_{i-1} \text{ (dot product attention)}$$

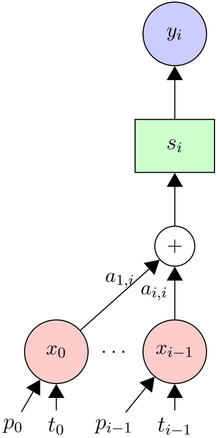
$$a_{j,i} = \operatorname{softmax}_{j \le i}(e_{j,i})$$

$$c_i = \sum_{j=1}^{i} a_{j,i} W^V \cdot x_{j-1}$$

$$s_i = b^{(2)} + W^{(2)} \cdot \mathrm{ReLU}(b^{(1)} + W^{(1)} \cdot c_i)$$

what about word order?

Transformer language model



causal self-attention

$$e_{j,i} = x_{j-1}^{\dagger} \cdot W^K \cdot W^Q \cdot x_{i-1} \text{ (dot product attention)}$$

$$a_{j,i} = \operatorname*{softmax}_{j \leq i}(e_{j,i})$$

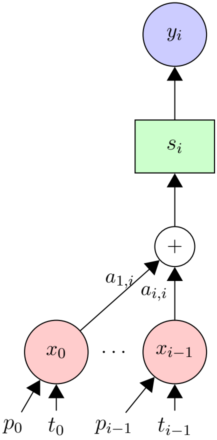
$$c_i = \sum_{j=1}^{i} a_{j,i} W^V \cdot x_{j-1}$$

$$s_i = b^{(2)} + W^{(2)} \cdot \mathrm{ReLU}(b^{(1)} + W^{(1)} \cdot c_i)$$

what about word order?
$x_i = x_i^{word} + x_i^{pos}$

Transformer language model

transformer in practice



$$x_i = x_i^{word} + x_i^{pos} \text{ (position embedding)}$$

$$e_{j,i}^{(h)} = x_{j-1}^{\dagger} \cdot W^{K_h} \cdot W^{Q_h} \cdot x_{i-1}$$

$$a_{j,i}^{(h)} = \operatorname*{softmax}_{j \le i}(e_{j,i}^{(h)})$$

$$c_i^{(h)} = \sum_{j=1}^{i} a_{j,i}^{(h)} W^{V_h} \cdot x_{j-1}$$

$$\widetilde{c}_i = \operatorname*{concat}_{h}(c_i^{(h)}) \text{ (multi-head attention)}$$

$$c_i = \text{layerNorm}(x_i + \widetilde{c}_i) \text{ (residual and layernorm)}$$

$$\widetilde{s}_i = b^{(2)} + W^{(2)} \cdot \text{ReLU}(b^{(1)} + W^{(1)} \cdot c_i)$$

$$s_i = \text{layerNorm}(c_i + \widetilde{s}_i) \text{ (residual and layernorm)}$$

Transformer language model

- pros & cons
  - pro: **SOTA on everything**
  - pro: can capture long distance dependencies
  - pro: training can be parallelized over words
  - con: theoretical complexity increases at each step
    - not an issue for single sentences
  - con: tricky to train
    - requires dropout, learning rate warmup, label smoothing, etc.
    - although modern variants are more stable and easier to train

Transformer language model

- pros & cons
  - pro: **SOTA on everything**
  - pro: can capture long distance dependencies
  - pro: training can be parallelized over words
  - con: theoretical complexity increases at each step
    - not an issue for single sentences
  - con: tricky to train
    - requires dropout, learning rate warmup, label smoothing, etc.
    - although modern variants are more stable and easier to train
- extensions
  - Transformer-XL [Dai et al., 2019]
    - recurrent over sentences
  - dynamic convolutions [Wu et al., 2019]
  - etc.

Sequence-to-sequence

- Generate an output sequence based on an input sequence
  - Machine translation
  - Summarization
  - etc., very flexible framework

Sequence-to-sequence

- Generate an output sequence based on an input sequence
  - Machine translation
  - Summarization
  - etc., very flexible framework
- Suppose that we have:
  - a source sentence S of length m $(x_1, \ldots, x_m)$
  - a target sentence T of length n $(y_1, \ldots, y_n)$
- We can express translation as a probabilistic model

$$T^* = \arg\max_T p(T|S)$$

- Expanding using the chain rule gives

$$\begin{aligned} p(T|S) &= p(y_1, \ldots, y_n | x_1, \ldots, x_m) \\ &= \prod_{i=1}^{n} p(y_i | y_1, \ldots, y_{i-1}, x_1, \ldots, x_m) \end{aligned}$$

Differences Between Translation and Language Model

- Target-side language model:

$$p(T) = \prod_{i=1}^{n} p(y_i | y_1, \ldots, y_{i-1})$$

- Translation model:

$$p(T|S) = \prod_{i=1}^{n} p(y_i | y_1, \ldots, y_{i-1}, x_1, \ldots, x_m)$$

- We could just treat sentence pair as one long sequence, but:
  - We do not care about $p(S)$
  - We may want different vocabulary, network architecture for source text

Differences Between Translation and Language Model

- Target-side language model:

$$p(T) = \prod_{i=1}^{n} p(y_i | y_1, \ldots, y_{i-1})$$

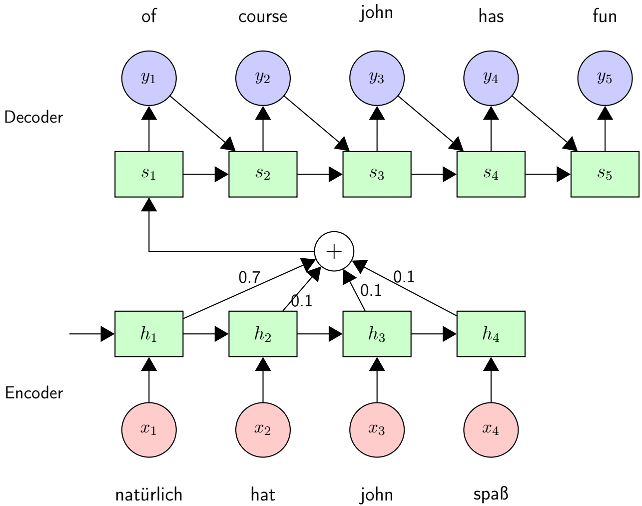
- Translation model:

$$p(T|S) = \prod_{i=1}^{n} p(y_i | y_1, \ldots, y_{i-1}, x_1, \ldots, x_m)$$

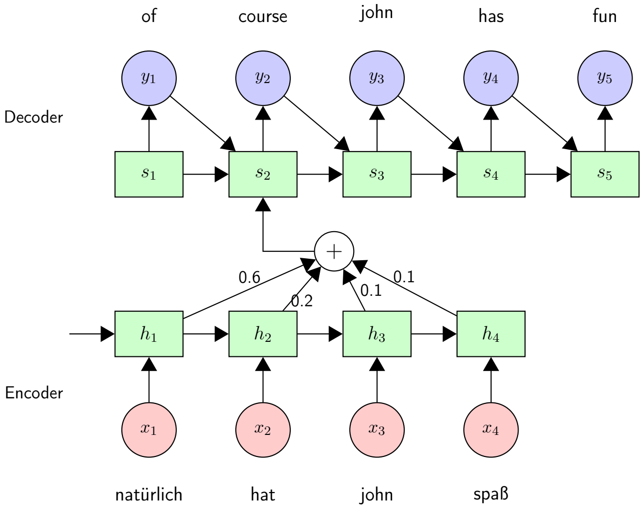
- We could just treat sentence pair as one long sequence, but:
  - We do not care about $p(S)$
  - We may want different vocabulary, network architecture for source text

$\rightarrow$ Use separate neural networks for source and target with an attention mechanism

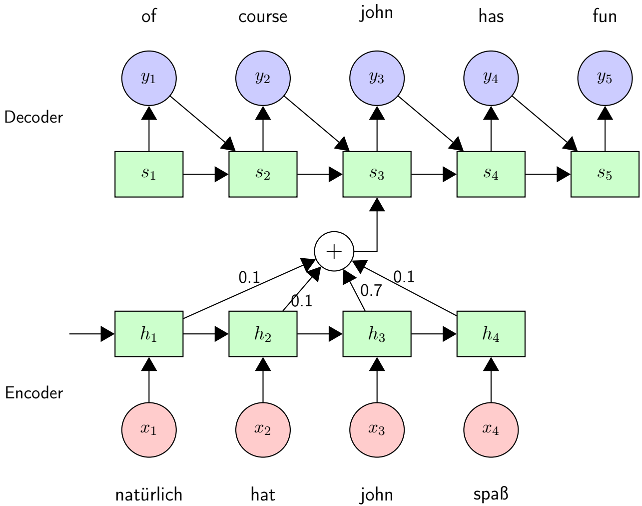
Recurrent Encoder-Decoder with Attention



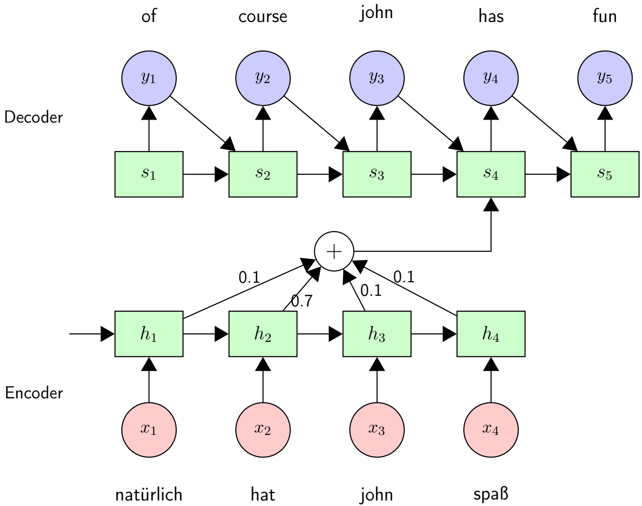
Recurrent Encoder-Decoder with Attention



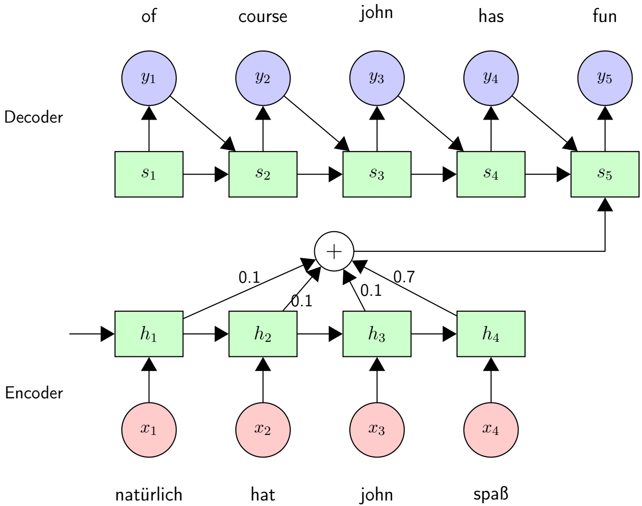
Recurrent Encoder-Decoder with Attention



Recurrent Encoder-Decoder with Attention



Recurrent Encoder-Decoder with Attention



Recurrent Attentional encoder-decoder

encoder

$$\overrightarrow{h}_j = \begin{cases} 0, & \text{, if } j = 0 \\ \mathsf{RNN}(h_{j-1}, x_j) & \text{, if } j > 0 \end{cases}$$

$$\overleftarrow{h}_j = \begin{cases} 0, & \text{, if } j = T_x + 1 \\ \mathsf{RNN}(h_{j+1}, x_j) & \text{, if } j \leq T_x \end{cases}$$

$$h_j = (\overrightarrow{h}_j, \overleftarrow{h}_j)$$

Recurrent Attentional encoder-decoder

decoder

$$s_i = \begin{cases} \mathsf{tanh}(W_s \overleftarrow{h}_i), & \text{, if } i = 0 \\ \mathsf{RNN}(s_{i-1}, y_{i-1}, c_i) & \text{, if } i > 0 \end{cases}$$

$$t_i = \mathsf{tanh}(U_o s_i + W^{out} E_y y_{i-1} + C_o c_i)$$

$$y_i = \mathsf{softmax}(V_o t_i)$$

cross-attention

$$e_{i,j} = h_j^\dagger \cdot W^K \cdot W^Q \cdot s_{i-1}$$

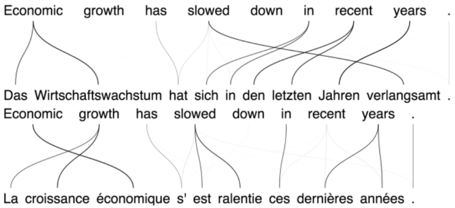
$$a_{i,j} = \underset{j}{\mathsf{softmax}}(e_{i,j})$$

$$c_i = \sum_{j=1}^{T_x} a_{i,j} W^V \cdot h_j$$

Attention model

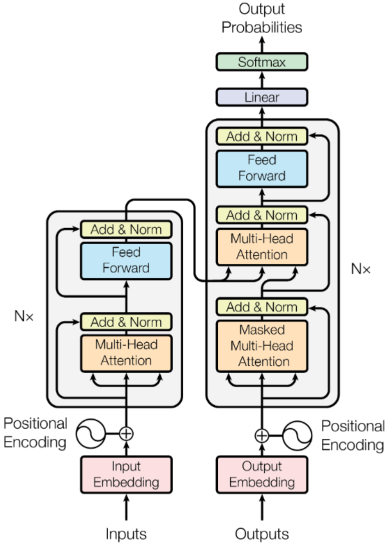
attention model

- side effect: we obtain alignment between source and target sentence
- information can also flow along recurrent connections, so there is no guarantee that attention corresponds to alignment
- applications:
  - visualisation
  - replace unknown words with back-off dictionary [Jean et al., 2015]
  - ...



Kyunghyun Cho http://devblogs.nvidia.com/parallelforall/introduction-neural-machine-translation-gpus-part-3/

Transformer encoder-decoder [Vaswani et al., 2017]



attention is all you need

- acausal self-attention in encoder
- causal self-attention in decoder
- cross-attention between encoder and decoder

Application of Encoder-Decoder Model

Scoring (a translation)

p(La, croissance, économique, s'est, ralentie, ces, dernières, années, . | Economic, growth, has, slowed, down, in, recent, year, .) = ?

Decoding (a source sentence)

Generate the most probable translation of a source sentence

$$y^* = \mathrm{argmax}_y\, p(y|\text{Economic, growth, has, slowed, down, in, recent, year, .})$$

Decoding

exact search

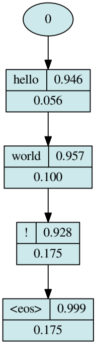
- generate every possible sentence T in target language
- compute score $p(T|S)$ for each
- pick best one

- intractable: $|\text{vocab}|^N$ translations for output length N
  $\rightarrow$ we need approximative search strategy

Decoding

approximative search/1: **greedy search**

- at each time step, compute probability distribution $P(y_i|S, y_{<i})$
- select y_i according to some heuristic:
  - sampling: sample from $P(y_i|S, y_{<i})$
  - greedy search: pick $\mathrm{argmax}_y\, p(y_i|S, y_{<i})$
- continue until we generate *<eos>*



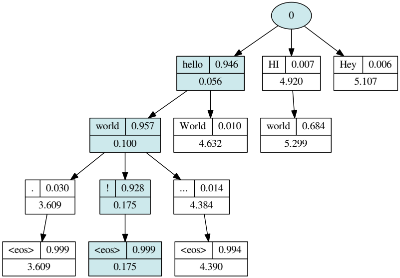
- efficient, but suboptimal

Decoding

approximative search/2: beam search

- maintain list of K hypotheses (beam)
- at each time step, expand each hypothesis k: $p(y_i^k|S, y_{<i}^k)$
- select K hypotheses with highest total probability:

$$\prod_i p(y_i^k|S, y_{<i}^k)$$



$K = 3$

- relatively efficient . . . beam expansion parallelisable
- currently default search strategy in neural machine translation
- small beam ($K \approx 10$) offers good speed-quality trade-off

Subwords for NLP: Motivation

Real text is an open-vocabulary

- compounding and other productive morphological processes
  - they charge a carry-on bag fee.
  - sie erheben eine Hand|gepäck|gebühr.
- names
  - Obama(English; German)
  - Обама (Russian)
  - オバマ (o-ba-ma) (Japanese)
- technical terms, numbers, etc.

... but neural architectures require a small token vocabulary

Subword units

segmentation algorithms: wishlist

- **open-vocabulary seq-to-seq**: encode *all* words through small vocabulary
- encoding generalizes to unseen words
- small text size
- good translation quality

Byte pair encoding for word segmentation [Sennrich et al., 2016]

bottom-up character merging

- starting point: character-level representation
  → computationally expensive
- compress representation based on information theory
  → byte pair encoding [Gage, 1994]
- repeatedly replace most frequent symbol pair ('A','B') with 'AB'
- hyperparameter: when to stop
  → controls vocabulary size

word	freq
'l o w </w>'	5
'l o w e r </w>'	2
'n e w e s t </w>'	6
'w i d e s t </w>'	3

vocabulary:
l o w </w> e r n s t i d

Byte pair encoding for word segmentation [Sennrich et al., 2016]

bottom-up character merging

- starting point: character-level representation
  → computationally expensive
- compress representation based on information theory
  → byte pair encoding [Gage, 1994]
- repeatedly replace most frequent symbol pair ('A','B') with 'AB'
- hyperparameter: when to stop
  → controls vocabulary size

word	freq
'l o w </w>'	5
'l o w e r </w>'	2
'n e w **es** t </w>'	6
'w i d **es** t </w>'	3

vocabulary:
l o w </w> e r n s t i d
es

Byte pair encoding for word segmentation [Sennrich et al., 2016]

bottom-up character merging

- starting point: character-level representation
  → computationally expensive
- compress representation based on information theory
  → byte pair encoding [Gage, 1994]
- repeatedly replace most frequent symbol pair ('A','B') with 'AB'
- hyperparameter: when to stop
  → controls vocabulary size

word	freq
'l o w </w>'	5
'l o w e r </w>'	2
'n e w **est** </w>'	6
'w i d **est** </w>'	3

vocabulary:
l o w </w> e r n s t i d
es **est**

Byte pair encoding for word segmentation [Sennrich et al., 2016]

bottom-up character merging

- starting point: character-level representation
  $\rightarrow$ computationally expensive
- compress representation based on information theory
  $\rightarrow$ byte pair encoding [Gage, 1994]
- repeatedly replace most frequent symbol pair ('A','B') with 'AB'
- hyperparameter: when to stop
  $\rightarrow$ controls vocabulary size

word	freq
'l o w </w>'	5
'l o w e r </w>'	2
'n e w **est</w>**'	6
'w i d **est</w>**'	3

vocabulary:
l o w </w> e r n s t i d
es est **est</w>**

Byte pair encoding for word segmentation [Sennrich et al., 2016]

bottom-up character merging

- starting point: character-level representation
  → computationally expensive
- compress representation based on information theory
  → byte pair encoding [Gage, 1994]
- repeatedly replace most frequent symbol pair ('A','B') with 'AB'
- hyperparameter: when to stop
  → controls vocabulary size

word	freq
'**lo** w </w>'	5
'**lo** w e r </w>'	2
'n e w est</w>'	6
'w i d est</w>'	3

vocabulary:
l o w </w> e r n s t i d
es est est</w> **lo**

Byte pair encoding for word segmentation [Sennrich et al., 2016]

bottom-up character merging

- starting point: character-level representation
  → computationally expensive
- compress representation based on information theory
  → byte pair encoding [Gage, 1994]
- repeatedly replace most frequent symbol pair ('A','B') with 'AB'
- hyperparameter: when to stop
  → controls vocabulary size

word	freq
'**low** </w>'	5
'**low** e r </w>'	2
'n e w est</w>'	6
'w i d est</w>'	3

vocabulary:
l o w </w> e r n s t i d
es est est</w> lo **low**

Byte pair encoding for word segmentation

why BPE?

- open-vocabulary:
  operations learned on training set can be applied to unknown words
- compression of frequent character sequences improves efficiency
  $\rightarrow$ trade-off between text length and vocabulary size

'l o w e s t </w>'

e s	$\rightarrow$	es
es t	$\rightarrow$	est
est </w>	$\rightarrow$	est</w>
l o	$\rightarrow$	lo
lo w	$\rightarrow$	low

Byte pair encoding for word segmentation

why BPE?

- open-vocabulary:
  operations learned on training set can be applied to unknown words
- compression of frequent character sequences improves efficiency
  → trade-off between text length and vocabulary size

'l o w **es** t </w>'

e s	→	**es**
es t	→	est
est </w>	→	est</w>
l o	→	lo
lo w	→	low

Byte pair encoding for word segmentation

why BPE?

- open-vocabulary:
  operations learned on training set can be applied to unknown words
- compression of frequent character sequences improves efficiency
  → trade-off between text length and vocabulary size

'l o w **est** </w>'

e s	→	es
es t	→	**est**
est </w>	→	est</w>
l o	→	lo
lo w	→	low

Byte pair encoding for word segmentation

why BPE?

- open-vocabulary:
  operations learned on training set can be applied to unknown words
- compression of frequent character sequences improves efficiency
  $\rightarrow$ trade-off between text length and vocabulary size

'l o w **est</w>**'

e s	$\rightarrow$	es
es t	$\rightarrow$	est
est </w>	$\rightarrow$	**est</w>**
l o	$\rightarrow$	lo
lo w	$\rightarrow$	low

Byte pair encoding for word segmentation

why BPE?

- open-vocabulary:
  operations learned on training set can be applied to unknown words
- compression of frequent character sequences improves efficiency
  → trade-off between text length and vocabulary size

'**lo** w est</w>'

e s	→	es
es t	→	est
est </w>	→	est</w>
l o	→	**lo**
lo w	→	low

Byte pair encoding for word segmentation

why BPE?

- open-vocabulary:
  operations learned on training set can be applied to unknown words
- compression of frequent character sequences improves efficiency
  → trade-off between text length and vocabulary size

'**low** est</w>'

e s	→	es
es t	→	est
est </w>	→	est</w>
l o	→	lo
lo w	→	**low**

Byte pair encoding for word segmentation

Applications

- Some form of BPE is used in most neural NLP architectures
- Extensions usually add stochasticity
  - Sentencepiece [Kudo, 2018]
  - BPE-Dropout [Provilkov et al., 2019]

Limitations of supervised learning

- For most tasks, annotated data is scarce
  - High quality data annotation is expensive
  - Data annotated by crowdsourcing or scraped from the web is noisy
- Even high quality annotated data can be out-of-domain
  - Sampling bias
  - Distribution shift over time
  - Different topics, styles, dialects, etc.

Limitations of supervised learning

- For most tasks, annotated data is scarce
  - High quality data annotation is expensive
  - Data annotated by crowdsourcing or scraped from the web is noisy
- Even high quality annotated data can be out-of-domain
  - Sampling bias
  - Distribution shift over time
  - Different topics, styles, dialects, etc.
- Non-annotated data
  - Readily available
  - Easier to find in-domain datasets

Semi-supervised learning

- Solution: learn from both unannotated and annotated data

Semi-supervised learning

- Solution: learn from both unannotated and annotated data

"When we're learning to see, nobody's telling us what the right answers are—we just look. Every so often, your mother says 'that's a dog,' but that's very little information. You'd be lucky if you got a few bits of information—even one bit per second—that way. The brain's visual system requires 10^{14} connections. And you only live for 10^9 seconds. So it's no use learning one bit per second. You need more like 10^5 bits per second. And there's only one place you can get that much information—from the input itself." - Geoffrey Hinton

Semi-supervised learning

- Mix unannotated and annotated data
  - Computationally expensive
- Or learn an unsupervised model on unannotated data, then transfer to a supervised task
  - Amortizes the cost of learning the unsupervised model
  - Can be more robust

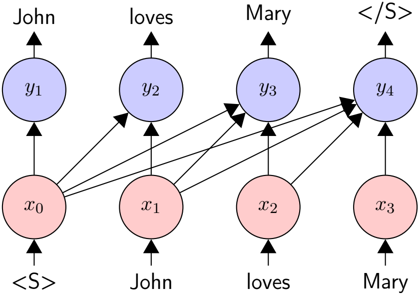
Representation learning

- Learn an encoding $h(x;\theta)$ on unannotated data
- For each supervised task
  - Learn a task model ("head") that takes the encoded data as input $f(h(x;\theta);\phi)$

Representation learning

- Learn an encoding $h(x;\theta)$ on unannotated data
- For each supervised task
  - Learn a task model ("head") that takes the encoded data as input $f(h(x;\theta);\phi)$
  - Either keep the encoder parameters θ fixed
    - Fast
    - Small memory consumption
    - Less risk of overfitting
  - Or finetune θ alongside ϕ
    - Potentially better quality, assuming no overfitting
    - Many forward and backward passes through the encoder

Masked language models

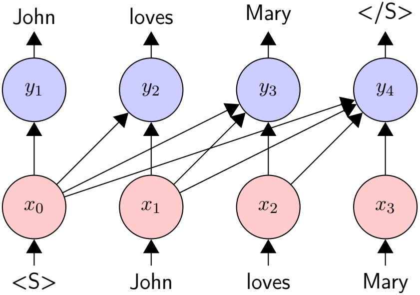


- Recall the autoregressive decomposition

$$p(t_1, \ldots, t_M) = \prod_{i=1}^{M} p(t_i | t_1, \ldots, t_{i-1})$$

  - Based on the chain rule of probability

Masked language models

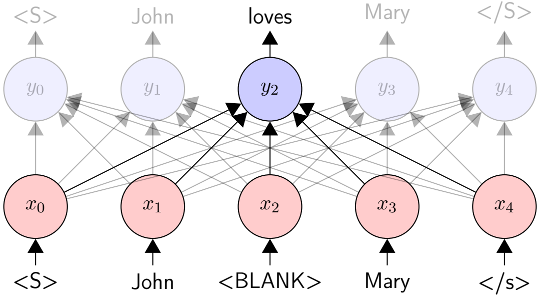


- Recall the autoregressive decomposition

$$p(t_1, \ldots, t_M) = \prod_{i=1}^{M} p(t_i | t_1, \ldots, t_{i-1})$$

  - Based on the chain rule of probability
  - But the joint probability can be decomposed in other ways

Masked language models

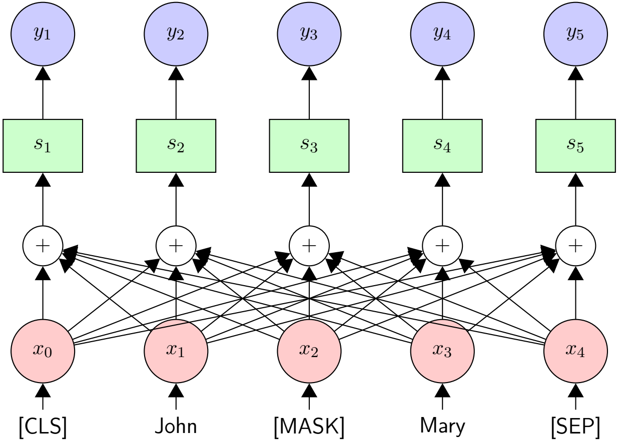


- Gap prediction

$$p(t_1, \dots, t_M) = p(t_i | t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_M) \cdot \\ \cdot\, p(t_1, \dots, t_{i-1}, t_{i+1}, \dots, t_M)$$

  - No closed form for the joint probability
  - But representation learning we don't care

BERT [Devlin et al., 2018]



- Transformer encoder trained on the MLM objective

BERT [Devlin et al., 2018]

- Transformer encoder
  - Very large and deep
  - Sub-word tokenization (wordpiece)
  - Trainable position embeddings

BERT [Devlin et al., 2018]

- Transformer encoder
  - Very large and deep
  - Sub-word tokenization (wordpiece)
  - Trainable position embeddings
- (Approximate) Masked LM training
  - For each example in each training step randomly choose 15% of tokens as target tokens
  - For each target token
    - Mask with 80% prob.
    - Randomly substitute with 10% prob.
    - Leave the same with 10% prob.
  - Training loss
    $-\sum_{i \in \text{targets}} \log p(t_i | t'_1, \ldots, t_M)$

BERT [Devlin et al., 2018]

- BERT is actually trained on sentence pairs
  - Consecutive with 50% prob., randomly sampled otherwise
  - Next Sentence Prediction loss
  - Turns out not to be very important

BERT [Devlin et al., 2018]

- BERT is actually trained on sentence pairs
  - Consecutive with 50% prob., randomly sampled otherwise
  - Next Sentence Prediction loss
  - Turns out not to be very important
- BERT is pretrained on a large English corpus (BookCorpus + Wikipedia)
- Versions of BERT in other languages or multilingual also exist

How to use BERT

- Take the hidden states of the final layers as the input representation

How to use BERT

- Take the hidden states of the final layers as the input representation
- Task
  - Tagging: subword representations
  - Sentence classification: [CLS] representation
  - Sentence pair classification: [CLS] and [SEP] representation

How to use BERT

- Take the hidden states of the final layers as the input representation
- Task
  - Tagging: subword representations
  - Sentence classification: [CLS] representation
  - Sentence pair classification: [CLS] and [SEP] representation
- Use as fixed representations or finetune on the supervised task
- Supervised head: linear model or shallow MLP

Pretrained representation learning models

- BERT-like
  - RoBERTa
  - ALBERT
  - DistilBERT
  - CamemBERT
    - French
  - XLM
    - Cross-lingual

Pretrained representation learning models

- BERT-like
  - RoBERTa
  - ALBERT
  - DistilBERT
  - CamemBERT
    - French
  - XLM
    - Cross-lingual
- Autoregressive
  - ELMo
  - GPT, GPT-2
  - Transformer-XL
  - XLNet
    - Can be used in autoregressive mode or bidirectional mode
    - Trained on permutations

Denoising seq-to-seq models

- MLM and autoregressive LMs are poorly suited to seq-to-seq tasks

Denoising seq-to-seq models

- MLM and autoregressive LMs are poorly suited to seq-to-seq tasks
- Seq-to-seq models trained to reconstruct corrupted text
- BERT-like noise, or variants
  - Mask consecutive spans
  - Learn to reorder sentences in document-order

Denoising seq-to-seq models

- MLM and autoregressive LMs are poorly suited to seq-to-seq tasks
- Seq-to-seq models trained to reconstruct corrupted text
- BERT-like noise, or variants
  - Mask consecutive spans
  - Learn to reorder sentences in document-order
  - **"He <MASK> vase. <SEP> The cat <MASK> table. "→ "The cat jumped on the table. <SEP> He knocked over the vase."**

Denoising seq-to-seq models

- MLM and autoregressive LMs are poorly suited to seq-to-seq tasks
- Seq-to-seq models trained to reconstruct corrupted text
- BERT-like noise, or variants
  - Mask consecutive spans
  - Learn to reorder sentences in document-order
  - "**He <MASK> vase. <SEP> The cat <MASK> table.** "→ "**The cat jumped on the table. <SEP> He knocked over the vase.**"
- The intuition is that the model learns the syntax and semantics of language while maintaining a predominantly copying behavior
- Pretrained models
  - BART, mBART
  - MASS
  - T5

Word embeddings

- Transformer-based LMs produce highly informative **contextual** representations
- The representation of each token depends on how it relates to all the other tokens
- Word embeddings of old (2013) just assign a fixed vector to each word in the vocabulary
- Is there a reason to still use word embeddings?

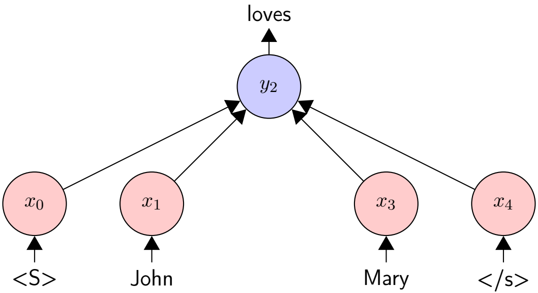
Word embeddings

- Transformer-based LMs produce highly informative **contextual** representations
- The representation of each token depends on how it relates to all the other tokens
- Word embeddings of old (2013) just assign a fixed vector to each word in the vocabulary
- Is there a reason to still use word embeddings?
- For transfer learning
  - Not as accurate as Transformers
  - But way faster to compute (just a table lookup)

Word embeddings

- Transformer-based LMs produce highly informative **contextual** representations
- The representation of each token depends on how it relates to all the other tokens
- Word embeddings of old (2013) just assign a fixed vector to each word in the vocabulary
- Is there a reason to still use word embeddings?
- For transfer learning
  - Not as accurate as Transformers
  - But way faster to compute (just a table lookup)
- For analysis
  - Relations between words
  - How words change over time
  - or over domains

Continuous bag of words [Mikolov et al., 2013]



- Log-bilinear model with fixed window L

$$
\begin{aligned}
p(t_i | t_1, \ldots, t_{i-1}, t_{i+1}, \ldots, t_M) &\approx p(t_i | t_{i-L}, \ldots, t_{i-1}, t_{i+1}, \ldots, t_{i+L}) \\
&= \mathsf{softmax}_i(W^{Out} \cdot s)
\end{aligned}
$$

$$
s = \sum_{j \neq i}^{[-L, L]} W^{Emb}_{:, t_j}
$$

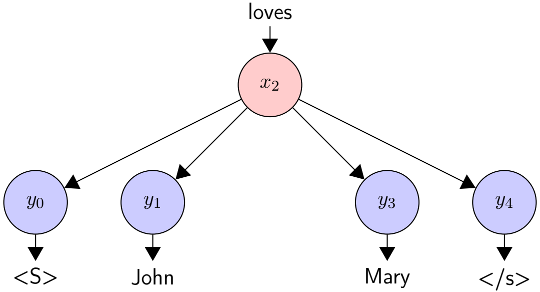
Continuous bag of words [Mikolov et al., 2013]

- Log-bilinear model with fixed window L

$$
\begin{aligned}
p(t_i|t_1,\ldots,t_{i-1},t_{i+1},\ldots,t_M) &\approx p(t_i|t_{i-L},\ldots,t_{i-1},t_{i+1},\ldots,t_{i+L}) \\
&= \mathsf{softmax}_i(W^{Out} \cdot s) \\
s &= \sum_{j \neq i}^{[-L,L]} W^{Emb}_{:,t_j}
\end{aligned}
$$

  - Output is $W^{Emb}_{:,t_i}$, not s
  - W^{Emb} and W^{Out} are not tied
  - $-\log \mathsf{softmax}_i(W^{Out} \cdot s)$ is approximated by Negative Sampling
    - Contrastive loss similar to NCE

Skipgram [Mikolov et al., 2013]



- Predict context from central word in a fixed window L

$$p(t_j|t_i) = \text{softmax}_j(W^{Out} \cdot W^{Emb}_{:,t_i})$$
$$|i - j| \leq L$$

- Also with approximated by Negative Sampling

Word embeddings

- CBOW and Skipgram are implemented in word2vec
- fastText [Joulin et al., 2016] is also based on Skipgram, extended with character n-gram features
- GloVe [Pennington et al., 2014] is based on matrix factorization
  - In practice, similar to Skipgram [Levy and Goldberg, 2014]
- Word embeddings can be also extracted from the parameters of recurrent language models or machine translation models

What do word embeddings encode?

- Syntactic similarity
  "to" $\approx$ **"from"**, **"in"**, **"on"**, ...
- Semantic similarity
  "one" $\approx$ **"two"**, **"three"**, **"four"**, ...
  "Paris" $\approx$ **"London"**, **"Rome"**, **"Berlin"**, ...
- Analogies
  "Paris" - "France" $+$ **"UK"** $\approx$ **"London"**
  "king" - "man" $+$ **"woman"** $\approx$ **"queen"**

What do word embeddings encode?

- Syntactic similarity
  "to" $\approx$ **"from"**, **"in"**, **"on"**, ...
- Semantic similarity
  "one" $\approx$ **"two"**, **"three"**, **"four"**, ...
  "Paris" $\approx$ **"London"**, **"Rome"**, **"Berlin"**, ...
- Analogies
  "Paris" - **"France"** + **"UK"** $\approx$ **"London"**
  "king" - **"man"** + **"woman"** $\approx$ **"queen"**
  "doctor" - **"man"** + **"woman"** $\approx$ **"nurse"**
  - Word Embeddings learn stereotypes

Aligning word embeddings

- Find a mapping between embeddings trained on different corpora
  - Same language or different languages
- Why?

Aligning word embeddings

- Find a mapping between embeddings trained on different corpora
  - Same language or different languages
- Why?
  - Initialize multilingual models
  - Study semantic or syntactic differences over space, time, political belief, etc. [Shoemark et al., 2019]

Aligning word embeddings

- Find a mapping between embeddings trained on different corpora
  - Same language or different languages
- Why?
  - Initialize multilingual models
  - Study semantic or syntactic differences over space, time, political belief, etc. [Shoemark et al., 2019]
- Given word embedding matrices W^X and W^Y

$$\text{argmax}_{P,\theta} \sum_{i<V} ||W^Y_{:,i} - F(W^X_{:,P(i)}; \theta)||$$

- Find a permutation over the indices P and the parameter θ of a mapping function F that minimize the distance

Aligning word embeddings

- Given word embedding matrices W^X and W^Y

$$\text{argmax}_{P,\theta} \sum_{i<V} ||W^Y_{:,i} - F(W^X_{:,P(i)};\theta)||$$

- Find a permutation over the indices P and the parameter θ of a mapping function F that minimize the distance
  - Usually $F(\cdot;\theta)$ is parametrized as an orthogonal matrix Θ s.t. $\Theta^T \cdot \Theta = I$
  - Isometry assumption
    "king" - "man" + "woman" $\approx$ "queen"
    "König" - "Mann" + "Frau " $\approx$ "Königin"
  - Orthogonal transformations preserve angles

Aligning word embeddings [Ruder et al., 2019]

$$\text{argmax}_{P,\Theta} \sum_{i<V} ||W^Y_{:,i} - \Theta \cdot W^X_{:,P(i)};||$$

1. Initialize permutation P from a seed dictionary
2. Repeat until convergence
   2.1 Find best orthogonal mapping Θ using the Orthogonal Procrustes method
   2.2 Find best permutation P

Orthogonal Procrustes method

$$\text{argmax}_{\Theta} ||\Theta \cdot A - B||$$

- Solution: $\Theta = U \cdot V^T$
- where $B \cdot A^T = U \cdot S \cdot V^T$
  - Singular value decomposition

Open problems in Deep Learning

- Deep Learning revolutionized NLP and other fields
- Nevertheless, Deep Learning is no silver bullet
  - Out-of-distribution fragility
  - Adversarial examples
  - Fairness
  - Explainability

Out-of-distribution fragility

- Deep Learning models are very sensitive to distribution shift between training and inference
- In NLP a common form of distribution shift is word usage
  - E.g. Wikipedia vs. Twitter
  - Europarl vs. Opensubtitles

Out-of-distribution fragility

- Deep Learning models are very sensitive to distribution shift between training and inference
- In NLP a common form of distribution shift is word usage
  - E.g. Wikipedia vs. Twitter
  - Europarl vs. Opensubtitles
- But even when word usage is similar deep learning models can struggle due to lack of compositionality

Compositionality

- Natural language is generally assumed to be compositional
  "*The meaning of a whole is a function of the meanings of the parts and of the way they are syntactically combined*"[Partee, 1995]

Compositionality

- Natural language is generally assumed to be compositional
  "*The meaning of a whole is a function of the meanings of the parts and of the way they are syntactically combined*"[Partee, 1995]
- Generalization without Systematicity: On the Compositional Skills of Sequence-to-Sequence Recurrent Networks [Lake and Baroni, 2017]

Compositionality

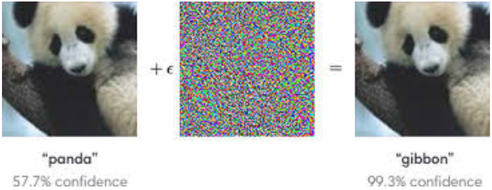
- Natural language is generally assumed to be compositional "*The meaning of a whole is a function of the meanings of the parts and of the way they are syntactically combined*"[Partee, 1995]
- Generalization without Systematicity: On the Compositional Skills of Sequence-to-Sequence Recurrent Networks [Lake and Baroni, 2017]
- The compositionality of neural networks: integrating symbolism and connectionism [Hupkes et al., 2019]

Compositionality

- Natural language is generally assumed to be compositional
  "*The meaning of a whole is a function of the meanings of the parts and of the way they are syntactically combined*"[Partee, 1995]
- Generalization without Systematicity: On the Compositional Skills of Sequence-to-Sequence Recurrent Networks [Lake and Baroni, 2017]
- The compositionality of neural networks: integrating symbolism and connectionism [Hupkes et al., 2019]
- Measuring abstract reasoning in neural networks [Barrett et al., 2018]

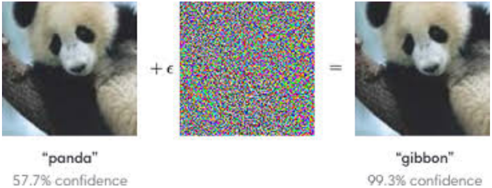
Adversarial examples

- Deep Learning models can be fooled to make very incorrect predictions when the inputs are perturbed adversarially



Adversarial examples

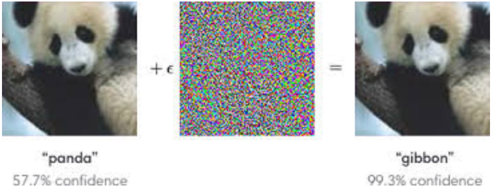
- Deep Learning models can be fooled to make very incorrect predictions when the inputs are perturbed adversarially



- Adversarial perturbations don't need to be precise
  - Adversary transfer across models
  - Physical adversaries

Adversarial examples

- Deep Learning models can be fooled to make very incorrect predictions when the inputs are perturbed adversarially



- Adversarial perturbations don't need to be precise
  - Adversary transfer across models
  - Physical adversaries
    - [Eykholt et al., 2018]

Figure 1: The left image shows real graffiti on a Stop sign, something that most humans would not think is suspicious. The right image shows our a physical perturbation applied to a Stop sign. We design our perturbations to mimic graffiti,

Adversarial examples

- Image adversaries are usually generated by gradient descent in pixel space
- Is NLP safe from adversaries?

Adversarial examples

- Image adversaries are usually generated by gradient descent in pixel space
- Is NLP safe from adversaries?
  - HotFlip: White-Box Adversarial Examples for Text Classification [Ebrahimi et al., 2017]
  - Universal Adversarial Triggers for Attacking and Analyzing NLP [Wallace et al., 2019]
  - Is BERT Really Robust? A Strong Baseline for Natural Language Attackon Text Classification and Entailment [Jin et al., 2019]

Fairness

- Machine learning models are used to automate decision that significantly affect people's lives
  - Credit scoring
  - Parole decisions
- What if they discriminate against certain groups of people?

Fairness

- Machine learning models are used to automate decision that significantly affect people's lives
  - Credit scoring
  - Parole decisions
- What if they discriminate against certain groups of people?
- COMPAS
  - ML model to assess criminal recidivism
  - Used by judges in sentencing
  - ProPublica study: COMPAS flags black people more often

Fairness

- Machine learning models are used to automate decision that significantly affect people's lives
  - Credit scoring
  - Parole decisions
- What if they discriminate against certain groups of people?
- COMPAS
  - ML model to assess criminal recidivism
  - Used by judges in sentencing
  - ProPublica study: COMPAS flags black people more often and has a higher false positive on black people

Fairness

- Machine learning models are used to automate decision that significantly affect people's lives
  - Credit scoring
  - Parole decisions
- What if they discriminate against certain groups of people?
- COMPAS
  - ML model to assess criminal recidivism
  - Used by judges in sentencing
  - ProPublica study: COMPAS flags black people more often and has a higher false positive on black people
  - Rebuttal by the model manufacturer: black people have higher base rate

Fairness

- Machine learning models are used to automate decision that significantly affect people's lives
  - Credit scoring
  - Parole decisions
- What if they discriminate against certain groups of people?
- COMPAS
  - ML model to assess criminal recidivism
  - Used by judges in sentencing
  - ProPublica study: COMPAS flags black people more often and has a higher false positive on black people
  - Rebuttal by the model manufacturer: black people have higher base rate according to statistics. Is the data accurate?

Fairness

- Machine learning models are used to automate decision that significantly affect people's lives
  - Credit scoring
  - Parole decisions
- What if they discriminate against certain groups of people?
- COMPAS
  - ML model to assess criminal recidivism
  - Used by judges in sentencing
  - ProPublica study: COMPAS flags black people more often and has a higher false positive on black people
  - Rebuttal by the model manufacturer: black people have higher base rate according to statistics. Is the data accurate?
  - Even if the data is accurate, if the model is not perfect it will make errors. What does it mean to err fairly?
- Different definition of fairness are mutually exclusive [Kleinberg et al., 2016]

Explainability

- Neural networks are black-boxes
- In many applications the model decisions must be "explainable"

Explainability

- Neural networks are black-boxes
- In many applications the model decisions must be "explainable"
  - Medical domain
  - Credit
  - Sentencing

Explainability

- Neural networks are black-boxes
- In many applications the model decisions must be "explainable"
  - Medical domain
  - Credit
  - Sentencing
- The European Union GDPR defines a "right to explanation and appeal"for any automated decision that significantly affects the interests of an EU citizen.
- Similar laws exist in the US for credit scoring

Bibliography I

Bai, S., Kolter, J. Z., and Koltun, V. (2018).
An empirical evaluation of generic convolutional and recurrent networks for sequence modeling.
arXiv preprint arXiv:1803.01271.

Barrett, D. G., Hill, F., Santoro, A., Morcos, A. S., and Lillicrap, T. (2018).
Measuring abstract reasoning in neural networks.
arXiv preprint arXiv:1807.04225.

Dai, Z., Yang, Z., Yang, Y., Cohen, W. W., Carbonell, J., Le, Q. V., and Salakhutdinov, R. (2019).
Transformer-xl: Attentive language models beyond a fixed-length context.
arXiv preprint arXiv:1901.02860.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018).
Bert: Pre-training of deep bidirectional transformers for language understanding.

Ebrahimi, J., Rao, A., Lowd, D., and Dou, D. (2017).
Hotflip: White-box adversarial examples for text classification.

Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., and Song, D. (2018).
Robust physical-world attacks on deep learning visual classification.
In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1625–1634.

Gage, P. (1994).
A New Algorithm for Data Compression.
C Users J., 12(2):23–38.

Bibliography II

Hupkes, D., Dankers, V., Mul, M., and Bruni, E. (2019).
The compositionality of neural networks: integrating symbolism and connectionism.
arXiv preprint arXiv:1908.08351.

Jean, S., Cho, K., Memisevic, R., and Bengio, Y. (2015).
On Using Very Large Target Vocabulary for Neural Machine Translation.
In
Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th Internation
pages 1–10, Beijing, China. Association for Computational Linguistics.

Jin, D., Jin, Z., Zhou, J. T., and Szolovits, P. (2019).
Is bert really robust? a strong baseline for natural language attack on text classification and entailment.

Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., and Mikolov, T. (2016).
Fasttext.zip: Compressing text classification models.
arXiv preprint arXiv:1612.03651.

Kleinberg, J., Mullainathan, S., and Raghavan, M. (2016).
Inherent trade-offs in the fair determination of risk scores.
arXiv preprint arXiv:1609.05807.

Kudo, T. (2018).
Subword regularization: Improving neural network translation models with multiple subword candidates.

Lake, B. M. and Baroni, M. (2017).
Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks.

Bibliography III

Levy, O. and Goldberg, Y. (2014).
Neural word embedding as implicit matrix factorization.
In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, Advances in Neural Information Processing Systems 27, pages 2177–2185. Curran Associates, Inc.

Miceli Barone, A. V., Helcl, J., Sennrich, R., Haddow, B., and Birch, A. (2017).
Deep architectures for neural machine translation.
In Proceedings of the Second Conference on Machine Translation, WMT 2017, Copenhagen, Denmark, September 7-8, 2017, pages 99–107.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013).
Efficient estimation of word representations in vector space.

Mikolov, T., Karafiát, M., Burget, L., Cernocký, J., and Khudanpur, S. (2010).
Recurrent neural network based language model.
In
INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhar
pages 1045–1048.

Partee, B. (1995).
Lexical semantics and compositionality.
An invitation to cognitive science: Language, 1:311–360.

Pennington, J., Socher, R., and Manning, C. D. (2014).
Glove: Global vectors for word representation.
In Empirical Methods in Natural Language Processing (EMNLP), pages 1532–1543.

Bibliography IV

Press, O. and Wolf, L. (2017).
Using the Output Embedding to Improve Language Models.
In
Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (E
Valencia, Spain.

Provilkov, I., Emelianenko, D., and Voita, E. (2019).
Bpe-dropout: Simple and effective subword regularization.

Ruder, S., Vulić, I., and Søgaard, A. (2019).
A survey of cross-lingual word embedding models.
Journal of Artificial Intelligence Research, 65:569–631.

Sennrich, R., Haddow, B., and Birch, A. (2016).
Neural Machine Translation of Rare Words with Subword Units.
In
Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Paper
pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Shoemark, P., Liza, F. F., Nguyen, D., Hale, S., and McGillivray, B. (2019).
Room to Glo: A systematic comparison of semantic change detection approaches with word embeddings.
In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 66–76, Hong Kong, China. Association for Computational Linguistics.

Bibliography V

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017).
Attention is all you need.
In Advances in neural information processing systems, pages 5998–6008.

Wallace, E., Feng, S., Kandpal, N., Gardner, M., and Singh, S. (2019).
Universal adversarial triggers for attacking and analyzing nlp.

Wu, F., Fan, A., Baevski, A., Dauphin, Y., and Auli, M. (2019).
Pay less attention with lightweight and dynamic convolutions.
In International Conference on Learning Representations.