Deep Learning for Natural Language processing

Jindřich Libovický

March 1, 2017

Charles University
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

Outline

Deep Learning in NLP

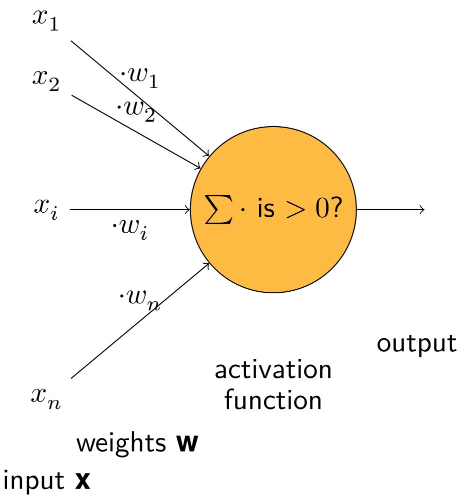
- NLP tasks learn end-to-end using deep learning — the number-one approach in current research
- State of the art in POS tagging, parsing, named-entity recognition, machine translation, ...
- Good news: training without almost any linguistic insight
- Bad news: requires enormous amount of training data and really big computational power

What is deep learning?

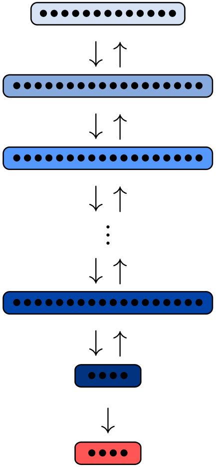
- Buzzword for machine learning using neural networks with many layers using back-propagation
- Learning of a real-valued function with millions of parameters that solves a particular problem
- Learning more and more abstract representation of the input data until we reach such a suitable representation for our problem

Neural Networks Basics

Neural Networks Basics


x_1
x_2
x_i
x_n
$\cdot w_1$
$\cdot w_2$
$\cdot w_i$
$\cdot w_n$
$\sum \cdot$ is > 0?
output
activation function
weights **w**
input **x**

Neural Network



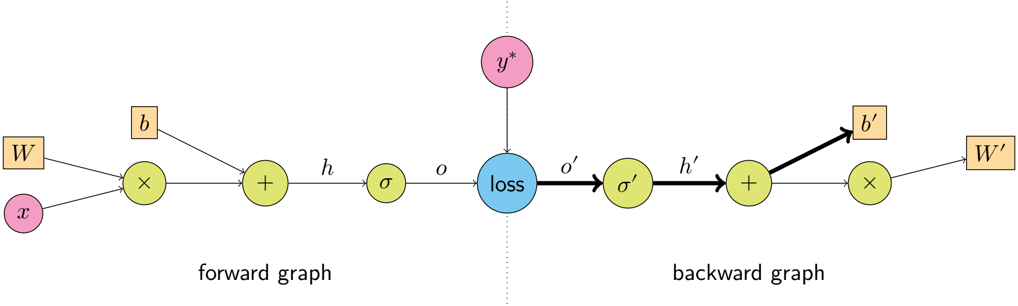
$$x$$
$$\downarrow$$
$$h_1 = f(W_1 x + b_1)$$
$$\downarrow$$
$$h_2 = f(W_2 h_1 + b_2)$$
$$\downarrow$$
$$\vdots$$
$$\downarrow$$
$$h_n = f(W_n h_{n-1} + b_n)$$
$$\downarrow$$
$$o = g(W_o h_n + b_o) \qquad \frac{\partial E}{\partial W_o} = \frac{\partial E}{\partial o} \cdot \frac{\partial o}{\partial W_o}$$
$$\downarrow \qquad\qquad \uparrow$$
$$E = e(o, t) \rightarrow \frac{\partial E}{\partial o}$$

Implementation

Logistic regression:

$$y = \sigma\left(Wx + b\right) \tag{1}$$

Computation graph:



Frameworks for Deep Learning

research and prototyping in Python

- graph statically constructed, symbolic computation
- computation happens in a session
- allows graph export and running as a binary

- computations written dynamically as normal procedural code
- easy debugging: inspecting variables at any time of the computation

Representing Words

Language Modeling

- estimate probability of a next word in a text

$$\mathsf{P}(w_i | w_{i-1}, w_{i-2}, \dots, w_1)$$

- standard approach: n-gram models with Markov assumption

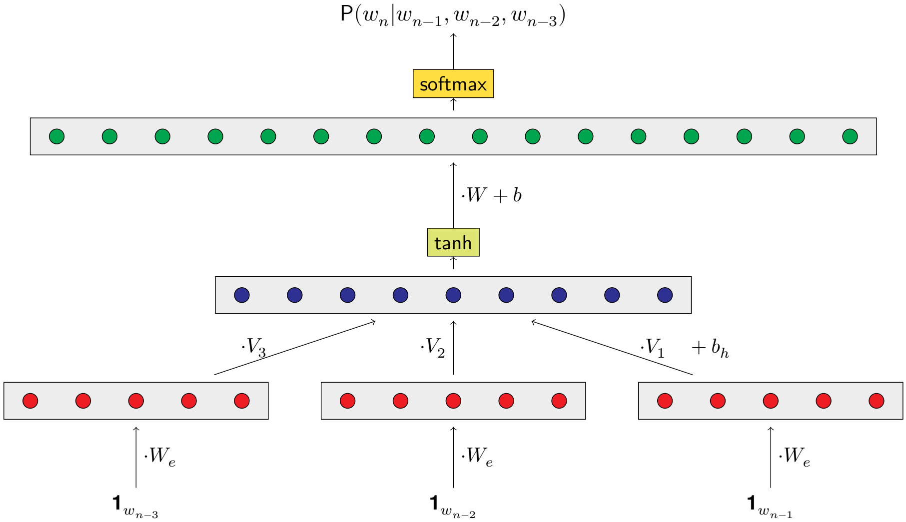
$$\approx \mathsf{P}(w_i | w_{i-1}, w_{i-2}, \dots, w_{i-n}) \approx \sum_{j=0}^{n} \lambda_j \frac{c(w_i | w_{i-1}, \dots, w_{i-j})}{c(w_i | w_{i-1}, \dots, w_{i-j+1})}$$

- Let's simulate it with a neural network:

$$\dots \approx F(w_{i-1}, \dots, w_{i-n} | \theta)$$

θ is a set of trainable parameters.

Simple Neural Language Model



Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *The Journal of Machine Learning Research*, 3 (Feb):1137–1155, 2003. ISSN 1532-4435

Neural LM: Word Representation

- limited vocabulary (hundred thousands words): indexed set of words
- words are initially represented as one-hot-vectors $\mathbf{1}_w = (0, \dots, 0, 1, 0, \dots 0)$
- projection $\mathbf{1}_w \cdot V$ corresponds to selecting one row from matrix V
- V: is a table of learned word vector representations
  so-called *word embeddings*
- dimension typically 100 — 300

The first hidden layer is then:

$$h_1 = V_{w_{i-n}} \oplus V_{w_{i-n+1}} \oplus \dots \oplus V_{w_{i-1}}$$

Matrix V is shared for all words.

Neural LM: Next Word Estimation

- optionally add extra hidden layer:

$$h_2 = f(h_1 W_1 + b_1)$$

- last layer: probability distribution over vocabulary

$$y = \mathsf{softmax}(h_2 W_2 + b_2) = \frac{\mathsf{exp}(h_2 W_2 + b_2)}{\sum \mathsf{exp}(h_2 W_2 + b_2)}$$

- training objective: cross-entropy between the true (i.e., one-hot) distribution and estimated distribution

$$E = -\sum_i p_{\mathsf{true}}(w_i) \log y(w_i) = \sum_i -\log y(w_i)$$

- learned by error back-propagation

Learned Representations

- word embeddings from LMs have interesting properties
- cluster according to POS & meaning similarity

FRANCE 454	JESUS 1973	XBOX 6909	REDDISH 11724	SCRATCHED 29869	MEGABITS 87025
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

Table 7: Word embeddings in the word lookup table of the language model neural network LM1 trained with a dictionary of size 100,000. For each column the queried word is followed by its index in the dictionary (higher means more rare) and its 10 nearest neighbors (using the Euclidean metric, which was chosen arbitrarily).

Table taken from Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011. ISSN 1533-7928

- in IR: query expansion by nearest neighbors
- in deep learning models: embeddings initialization speeds up training / allows complex model with less data

Implementation in PyTorch I

```
import torch
import torch.nn as nn

class LanguageModel(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim):
        super().__init__()

        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.hidden_layer = nn.Linear(3 * embedding_dim, hidden_dim)
        self.output_layer = nn.Linear(hidden_dim, vocab_size)
        self.loss_function = nn.CrossEntropyLoss()

    def forward(self, word_1, word_2, word_3, target=None):
        embedded_1 = self.embedding(word_1)
        embedded_2 = self.embedding(word_2)
        embedded_3 = self.embedding(word_3)
```

Implementation in PyTorch II

```
hidden = torch.tanh(self.hidden_layer(
    torch.cat(embedded_1, embedded_2, embedded_3)))
logits = self.output_layer(hidden)

loss = None
if target is not None:
    loss = self.loss_function(logits, targets)

return logits, loss
```

Implementation in TensorFlow I

```
import tensorfow as tf

input_words = [tf.placeholder(tf.int32, shape=[None]) for _ in range(3)]
target_word = tf.placeholder(tf.int32, shape[None])

embeddings = tf.get_variable(tf.float32, shape=[vocab_size, emb_dim])
embedded_words = tf.concat([tf.nn.embedding_lookup(w) for w in input_words])

hidden_layer = tf.layers.dense(embedded_words, hidden_size, activation=tf.tanh)
output_layer = tf.layers.dense(hidden_layer, vocab_size, activation=None)
output_probabilities = tf.nn.softmax(output_layer)

loss = tf.nn.cross_entropy_with_logits(output_layer, target_words)

optimizer = tf.optimizers.AdamOptimizers()
train_op = optimizer.minimize(loss)
```

Implementation in TensorFlow II

```
session = tf.Session()
# initialize variables
```

Training given batch

```
_, loss_value = session.run([train_op, loss], feed_dict={
    input_words[0]: ..., input_words[1]: ..., input_words[2]: ...,
    target_word: ...
})
```

Inference given batch

```
probs = session.run(output_probabilities, feed_dict={
    input_words[0]: ..., input_words[1]: ..., input_words[2]: ...,
})
```

Representing Sequences

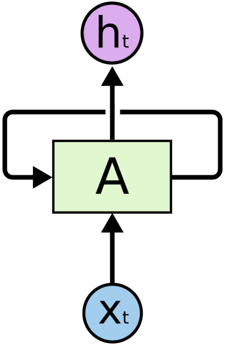
Representing Sequences

Representing Sequences

Recurrent Networks

Recurrent Networks (RNNs)

…the default choice for sequence labeling

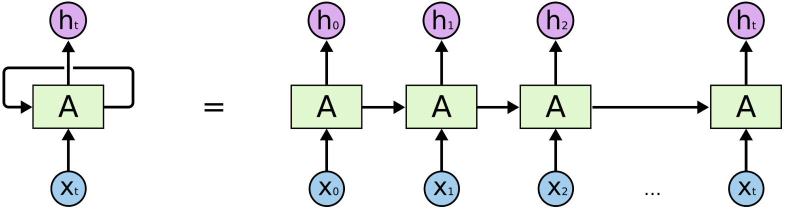


- inputs: $x_{,} \dots , x_T$
- initial state $h_0 = \mathbf{0}$, a result of previous computation, trainable parameter
- recurrent computation: $h_t = A(h_{t-1}, x_t)$

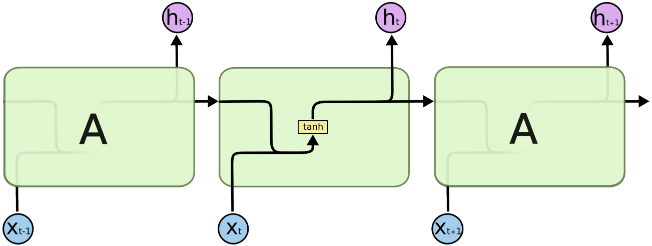
RNN as Imperative Code

```python
def rnn(initial_state, inputs):
  prev_state = initial_state
  for x in inputs:
    new_state, output = rnn_cell(x, prev_state)
    prev_state = new_state
    yield output
```

RNN as a Fancy Image



Vanilla RNN



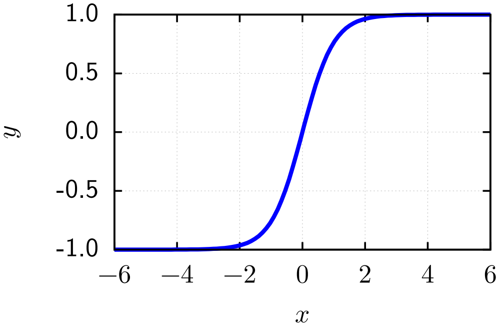
$$h_t = \text{tanh}\left(W[h_{t-1}; x_t] + b\right)$$

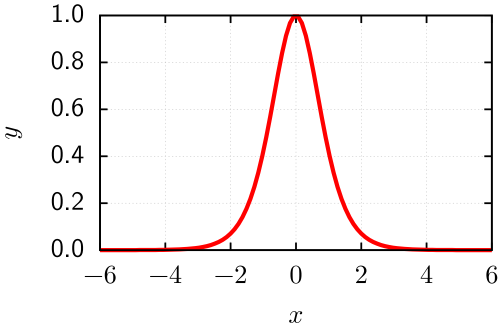
- cannot propagate long-distance relations
- vanishing gradient problem

Vanishing Gradient Problem (1)

$$\tanh x = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

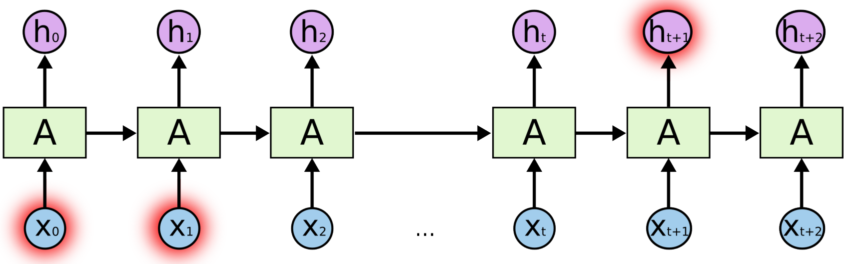
$$\frac{\mathrm{d}\tanh x}{\mathrm{d}x} = 1 - \tanh^2 x \in (0, 1]$$





Weight initialized $\sim \mathcal{N}(0, 1)$ to have gradients further from zero.

Vanishing Gradient Problem (2)



$$\frac{\partial E_{t+1}}{\partial b} = \frac{\partial E_{t+1}}{\partial h_{t+1}} \cdot \frac{\partial h_{t+1}}{\partial b} \quad \text{(chain rule)}$$

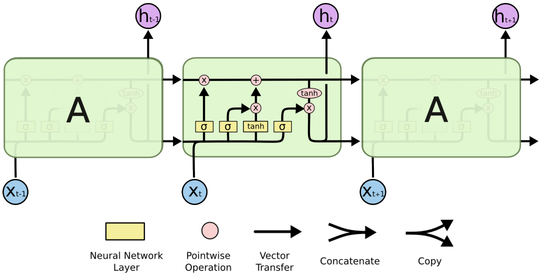
Vanishing Gradient Problem (3)

$$
\begin{aligned}
\frac{\partial h_t}{\partial b} &= \frac{\partial \tanh \overbrace{(W_h h_{t-1} + W_x x_t + b)}^{= z_t \text{ (activation)}}}{\partial b} \qquad \text{(tanh}' \text{ is derivative of tanh)} \\
&= \tanh'(z_t) \cdot \left(\frac{\partial W_h h_{t-1}}{\partial b} + \underbrace{\frac{\partial W_x x_t}{\partial b}}_{=0} + \underbrace{\frac{\partial b}{\partial b}}_{=1} \right) \\
&= \underbrace{W_h}_{\sim \mathcal{N}(0,1)} \underbrace{\tanh'(z_t)}_{\in (0;1]} \frac{\partial h_{t-1}}{\partial b} + \tanh'(z_t)
\end{aligned}
$$

Long Short-Term Memory Networks

LSTM = Long short-term memory

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. ISSN 0899-7667

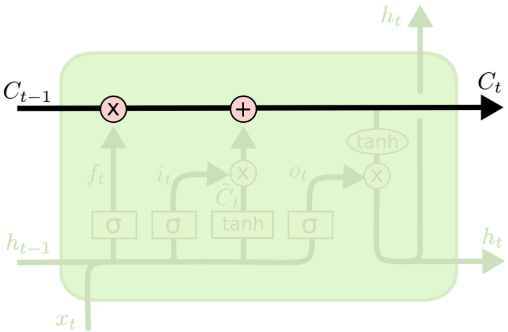


Control the gradient flow by explicitly gating:

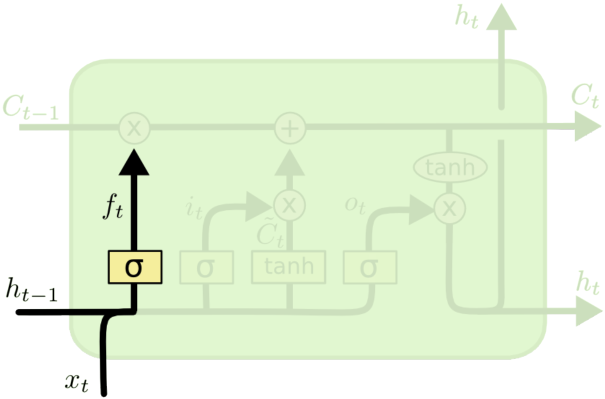
- what to use from input,
- what to use from hidden state,
- what to put on output

LMST: Hidden State

- two types of hidden states
- h_t — "public" hidden state, used an output
- c_t — "private" memory, no non-linearities on the way
- direct flow of gradients (without multiplying by ≤ 1 derivatives)



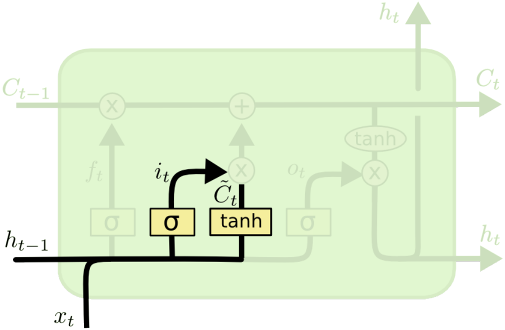
LSTM: Forget Gate



$$f_t = \sigma\left(W_f[h_{t-1}; x_t] + b_f\right)$$

- based on input and previous state, decide what to forget from the memory

LSTM: Input Gate

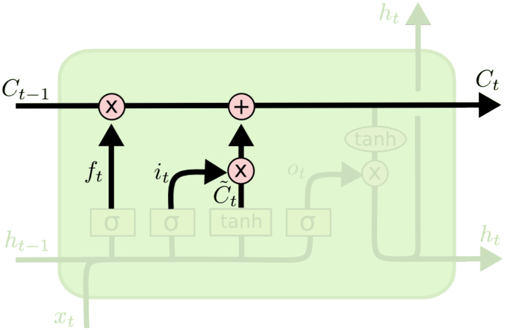


$$i_t = \sigma\left(W_i \cdot [h_{t-1}; x_t] + b_i\right)$$

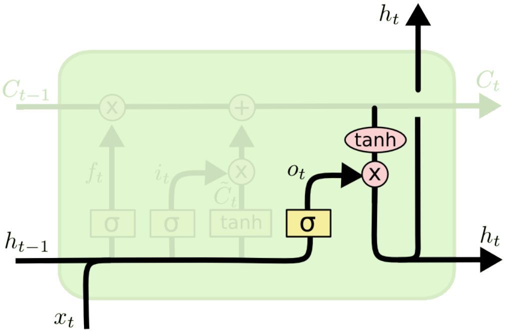
$$\tilde{C}_t = \tanh\left(W_c \cdot [h_{t-1}; x_t] + b_C\right)$$

- $\tilde{C}$ — candidate what may want to add to the memory
- i_t — decide how much of the information we want to store

LMST: Cell State Update



$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$$



$$o_t = \sigma\left(W_o \cdot [h_{t-1}; x_t] + b_o\right)$$

$$h_t = o_t \odot \tanh C_t$$

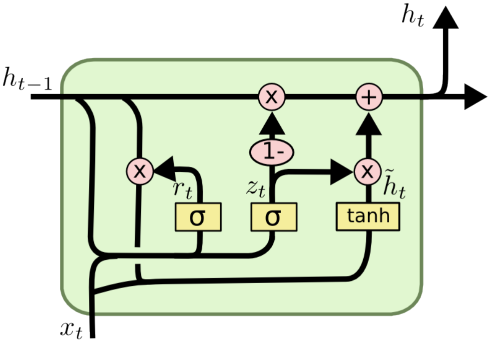
Here we are, LSTM!

$$
\begin{aligned}
f_t &= \sigma\left(W_f[h_{t-1};x_t] + b_f\right)\\
i_t &= \sigma\left(W_i \cdot [h_{t-1};x_t] + b_i\right)\\
o_t &= \sigma\left(W_o \cdot [h_{t-1};x_t] + b_o\right)\\
\tilde{C}_t &= \mathsf{tanh}\left(W_c \cdot [h_{t-1};x_t] + b_C\right)\\
C_t &= f_t \odot C_{t-1} + i_t \odot \tilde{C}_t\\
h_t &= o_t \odot \mathsf{tanh}\, C_t
\end{aligned}
$$

Question How would you implement it efficiently?
Compute all gates in a single matrix multiplication.

Gated Recurrent Units

update gate	$z_t = \sigma(x_t W_z + h_{t-1} U_z + b_z) \in (0,1)$
remember gate	$r_t = \sigma(x_t W_r + h_{t-1} U_r + b_r) \in (0,1)$
candidate hidden state	$\tilde{h}_t = \mathsf{tanh}\,(x_t W_h + (r_t \odot h_{t-1}) U_h) \in (-1,1)$
hidden state	$h_t = (1 - z_t) \odot h_{t-1} + z_t \cdot \tilde{h}_t$



LSTM vs. GRU

- GRU is smaller and therefore faster
- performance similar, task dependent
- theoretical limitation: GRU accepts regular languages, LSTM can simulate counter machine

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. ISSN 2331-8422; Gail Weiss, Yoav Goldberg, and Eran Yahav. On the practical computational power of finite precision rnns for language recognition. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 740–745, Melbourne, Australia, July 2018. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P18-2117`

RNN in PyTorch

```
rnn = nn.LSTM(input_dim, hidden_dim=512, num_layers=1,
    bidirectional=True, dropout=0.8)
output, (hidden, cell) = self.rnn(x)
```

https://pytorch.org/docs/stable/nn.html?highlight=lstm#torch.nn.LSTM

RNN in TensorFlow

```
inputs = ... # float tf.Tensor of shape [batch, length, dim]
lengths = ... # int tf.Tensor of shape [batch]

# Cell objects are templates
fw_cell = tf.nn.rnn_cell.LSTMCell(512, name="fw_cell")
bw_cell = tf.nn.rnn_cell.LSTMCell(512, name="bw_cell")

outputs, states = tf.nn.bidirectional_dynamic_rnn(
    cell_fw, cell_bw, inputs, sequence_length=lengths)
```

https://www.tensorflow.org/api_docs/python/tf/nn/bidirectional_dynamic_rnn

Bidirectional Networks

- simple trick to improve performance
- run one RNN forward, second one backward and concatenate outputs

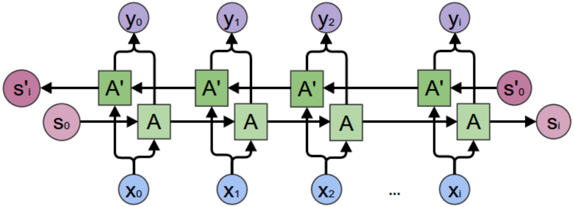


Image from: `http://colah.github.io/posts/2015-09-NN-Types-FP/`

- state of the art in tagging, crucial for neural machine translation

Representing Sequences

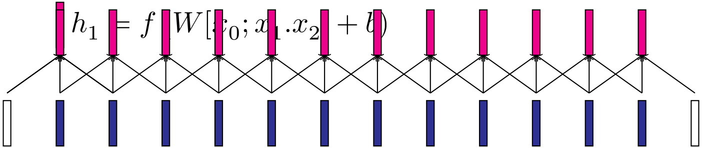
Convolutional Networks

1-D Convolution

≈ sliding window over the sequence

$$h_i = f\left(W\left[x_{i-1}; x_i; x_{i+1}\right] + b\right)$$

$$h_1 = f\left(W\left[x_0; x_1; x_2\right] + b\right)$$



$x_0 = \vec{0}$ embeddings $\mathbf{x} = (x_1, \dots, x_N)$ $x_N = \vec{0}$

pad with 0s if we want to keep sequence length

1-D Convolution: Pseudocode

```
xs = ... # input sequnce

kernel_size = 3 # window size
filters = 300 # output dimensions
strides=1     # step size

W = trained_parameter(xs.shape[2] * kernel_size, filters)
b = trained_parameter(filters)
window = kernel_size // 2

outputs = []
for i in range(window, xs.shape[1] - window):
    h = np.mul(W, xs[i - window:i + window]) + b
    outputs.append(h)
return np.array(h)
```

1-D Convolution: Frameworks

TensorFlow

```
h = tf.layers.conv1d(x, filters=300 kernel_size=3,
                     strides=1, padding='same')
```

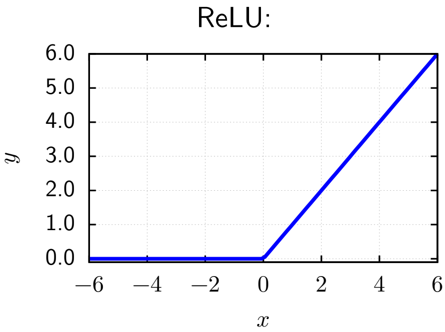
https://www.tensorflow.org/api_docs/python/tf/layers/conv1d

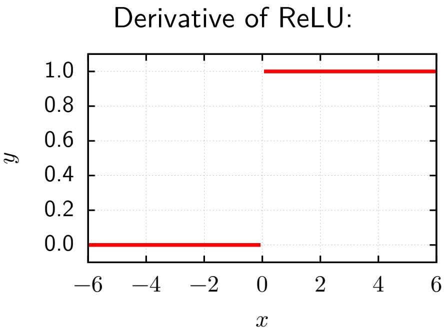
PyTorch

```
conv = nn.Conv1d(in_channels, out_channels=300, kernel_size=3, stride=1,
                 padding=0, dilation=1, groups=1, bias=True)
h = conv(x)
```

https://pytorch.org/docs/stable/nn.html#torch.nn.Conv1d

Rectified Linear Units



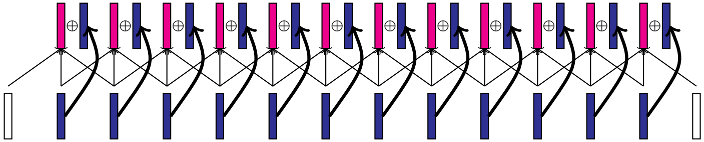


faster, suffer less with vanishing gradient

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, TODO, TODO 2010. TODO

Residual Connections

$$h_i = f\left(W\left[x_{i-1}; x_i; x_{i+1}\right] + b\right) + x_i$$



$x_0 = \vec{0}$ embeddings $\mathbf{x} = (x_1, \ldots, x_N)$ $x_N = \vec{0}$

Allows training deeper networks.

Why do you it helps?

Better gradient flow – the same as in RNNs.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, TODO, TODO 2016. IEEE Computer Society

Residual Connections: Numerical Stability

Numerically unstable, we need activation to be in similar scale ⇒ layer normalization.
Activation before non-linearity is normalized:

$$\overline{a}_i = \frac{g_i}{\sigma_i} \left(a_i - \mu_i \right)$$

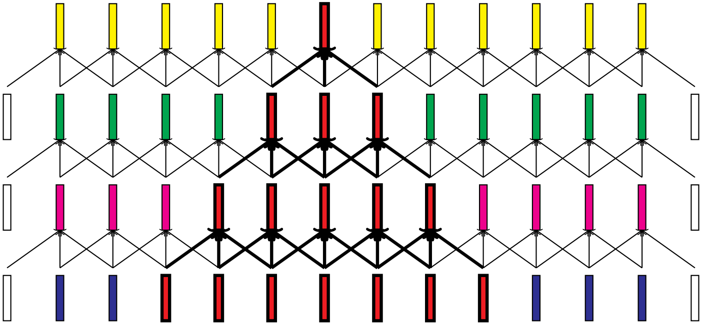
...g is a trainable parameter, μ, σ estimated from data.

$$\mu = \frac{1}{H} \sum_{i=1}^{H} a_i$$

$$\sigma = \sqrt{\frac{1}{H} \sum_{i=1}^{H} (a_i - \mu)^2}$$

Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. ISSN 2331-8422

Receptive Field



$x_0 = \vec{0}$ embeddings $\mathbf{x} = (x_1, \ldots, x_N)$ $x_N = \vec{0}$

Can be enlarged by dilated convolutions.

Convolutional architectures

$+$

- extremely computationally efficient

$-$

- limited context
- by default no aware of n-gram order

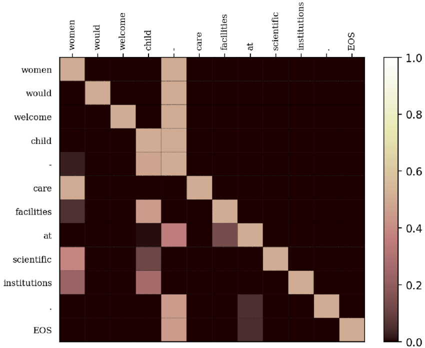
- max-pooling over the hidden states $=$ element-wise maximum over sequence
- can be understood as an $\exists$ operator over the feature extractors

Representing Sequences

Self-attentive Networks

Self-attentive Networks

- In some layers: states are linear combination of previous layer states
- Originally for the Transformer model for machine translation



- similarity matrix between all pairs of states
- $O(n^2)$ memory, $O(1)$ time (when paralelized)
- next layer: sum by rows

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 6000–6010, Long Beach, CA, USA, December 2017. Curran Associates, Inc

Single-head setup

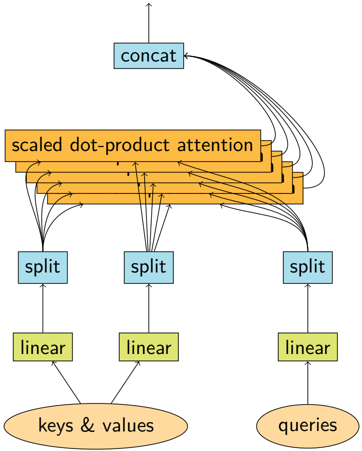
$$\mathrm{Attn}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V$$

$$h_{i+1} = \sum \mathrm{softmax}\left(\frac{h_i h_i^\top}{\sqrt{d}}\right)$$

Multihead-head setup

$$\mathrm{Multihead}(Q, V) = (H_1 \oplus \cdots \oplus H_h)W^O$$

$$H_i = \mathrm{Attn}(QW_i^Q, VW_i^K, VW_i^V)$$



Dot-Product Attention in PyTorch

```python
def attention(query, key, value, mask=None):
    d_k = query.size(-1)
    scores = torch.matmul(query, key.transpose(-2, -1)) \
            / math.sqrt(d_k)
    p_attn = F.softmax(scores, dim = -1)
    return torch.matmul(p_attn, value), p_attn
```

Dot-Product Attention in TensorFlow

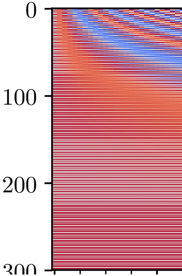
```python
def scaled_dot_product(self, queries, keys, values):
    o1 = tf.matmul(queries, keys, transpose_b=True)
    o2 = o1 / (dim**0.5)

    o3 = tf.nn.softmax(o2)
    return tf.matmul(o3, values)
```

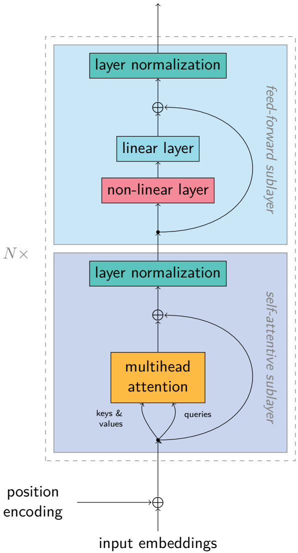
Position Encoding

Model cannot be aware of the position in the sequence.

$$\text{pos}(i) = \begin{cases} \sin\left(\frac{t}{10^4}^{\frac{i}{d}}\right), & \text{if } i \mod 2 = 0 \\ \cos\left(\frac{t}{10^4}^{\frac{i-1}{d}}\right), & \text{otherwise} \end{cases}$$



Stacking self-attentive Layers



- several layers (original paper 6)
- each layer: 2 sub-layers: self-attention and feed-forward layer
- everything inter-connected with residual connections

Architectures Comparison

	computation	sequential operations	memory
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n \cdot d)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(n \cdot d)$
Self-attentive	$O(n^2 \cdot d)$	$O(1)$	$O(n^2 \cdot d)$

d model dimension, n sequence length, k convolutional kernel

Classification and Labeling

Sequence Clasification

- tasks like sentiment analysis, genre classification
- need to get one vector from sequence $\rightarrow$ average or max pooling
- optionally hidden layers, at the and softmax for probability distribution over classes

Softmax & Cross-Entropy

Output layer with softmax (with parameters W, b):

$$P_y = \text{softmax}(\mathbf{x}) = \mathrm{P}(y = j \mid \mathbf{x}) = \frac{\exp \mathbf{x}^\top W + b}{\sum \exp \mathbf{x}^\top W + b}$$

Network error = cross-entropy between estimated distribution and one-hot ground-truth distribution $T = \mathbf{1}(y^*)$:

$$\begin{aligned} L(P_y, y^*) = H(P,T) &= -\mathbb{E}_{i\sim T} \log P(i) \\ &= -\sum_i T(i) \log P(i) \\ &= -\log P(y^*) \end{aligned}$$

Derivative of Cross-Entropy

Let $l = \mathbf{x}^\top W + b$, l_{y^*} corresponds to the correct one.

$$\begin{aligned}
\frac{\partial L(P_y, y^*)}{\partial l} &= -\frac{\partial}{\partial l} \log \frac{\exp l_{y^*}}{\sum_j \exp l_j} = -\frac{\partial}{\partial l} l_{y^*} - \log \sum \exp l \\
&= \mathbf{1}_{y^*} + \frac{\partial}{\partial l} - \log \sum \exp l = \mathbf{1}_{y^*} - \frac{\sum \mathbf{1}_{y^*} \exp l}{\sum \exp l} = \\
&= \mathbf{1}_{y^*} - P_y(y^*)
\end{aligned}$$

Interpretation: Reinforce the correct logit, supress the rest.

Sequence Labeling

- assign value / probability distribution to every token in a sequence
- morphological tagging, named-entity recognition, LM with unlimited history, answer span selection
- every state is classified independently with a classifier
- during training, error babckpropagate form all classifiers

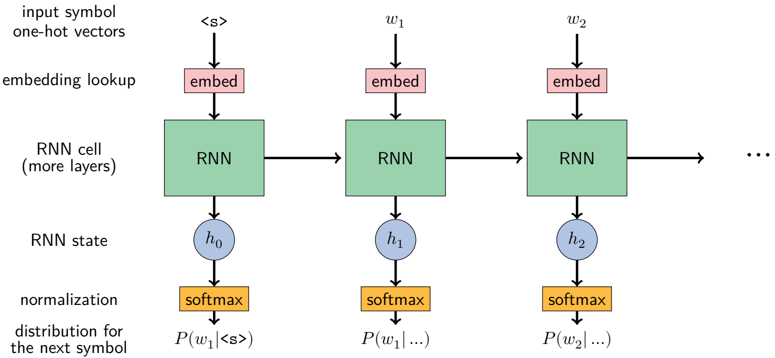
Lab next time: i/y spelling as sequence labeling

Generating Sequences

Sequence-to-sequence Learning

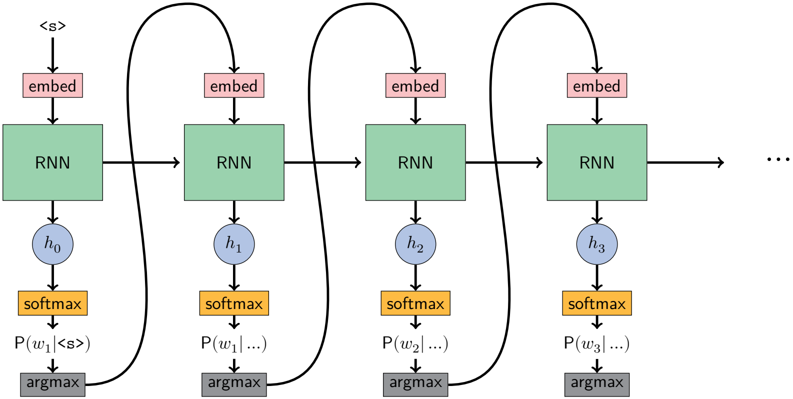
- target sequence is of different lenght tahn source
- no-trivial (= not monotonic) correspondence of source and target
- taks like: machine translation, text summarization, image captioning

Neural Language Model



- estimate probability of a sentence using the chain rule
- output distributions can be used for sampling

Sampling from a LM



when conditioned on input $\rightarrow$ autoregressive decoder

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112, Montreal, Canada, December 2014. Curran Associates, Inc

Autoregressive Decoding: Pseudocode

```
last_w = "<s>"
while last_w != "</s>":
    last_w_embeding = target_embeddings[last_w]
    state, dec_output = dec_cell(state,
                                 last_w_embeding)
    logits = output_projection(dec_output)
    last_w = np.argmax(logits)
    yield last_w
```

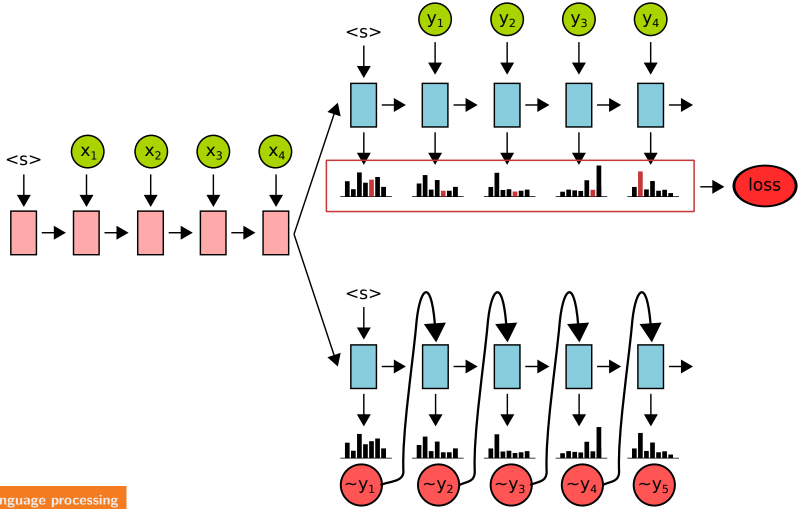
Architectures in the Decoder

- RNN – original sequence-to-sequence learning (2015)
  - principle known since 2014 (University of Montreal)
  - made usable in 2016 (University of Edinburgh)
- CNN – convolution sequence-to-sequence by Facebook (2017)
- Self-attention (so called Transformer) by Google (2017)

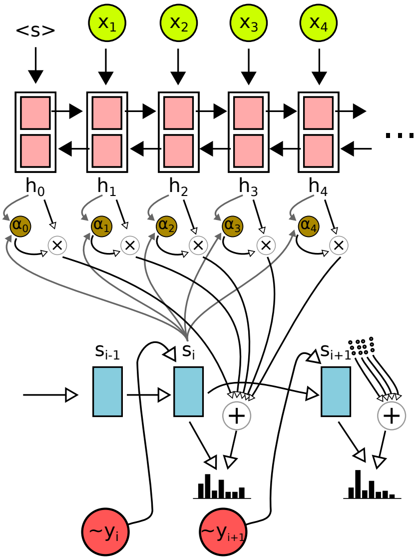
More on the topic in the MT class.

Implementation: Runtime vs. training

runtime: $\hat{y}_j$ (decoded) $\times$ **training:** y_j (ground truth)



Attention Model



Attention Model in Equations (1)

Inputs:

decoder state s_i

encoder states $h_j = \left[\overrightarrow{h_j}; \overleftarrow{h_j}\right] \quad \forall i = 1 \ldots T_x$

Attention energies:

$$e_{ij} = v_a^\top \tanh\left(W_a s_{i-1} + U_a h_j + b_a\right)$$

Attention distribution:

$$\alpha_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{k=1}^{T_x} \exp\left(e_{ik}\right)}$$

Context vector:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. ISSN 2331-8422

Attention Model in Equations (2)

Output projection:

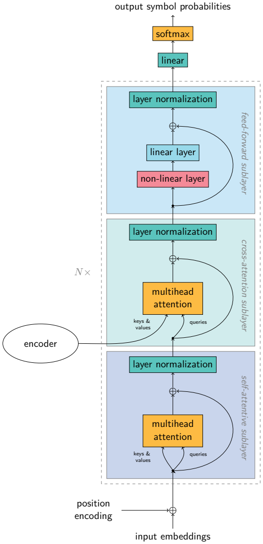
$$t_i = \mathsf{MLP}\left(U_o s_{i-1} + V_o E y_{i-1} + C_o c_i + b_o\right)$$

...attention is mixed with the hidden state

Output distribution:

$$p\left(y_i = k | s_i, y_{i-1}, c_i\right) \propto \mathsf{exp}\left(W_o t_i\right)_k + b_k$$

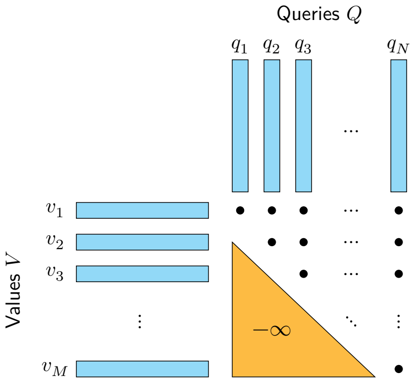
Transformer Decoder



- similar to encoder, additional layer with attention to the encoder
- in every steps self-attention over complete history $\Rightarrow O(n^2)$ complexity

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 6000–6010, Long Beach, CA, USA, December 2017. Curran Associates, Inc

Transfomer Decoder: Non-autoregressive training



- analogical to encoder
- target is known at training: don't need to wait until it's generated
- self attention can be parallelized via matrix multiplication
- prevent attentding the future using a mask

Question 1: What if the matrix was diagonal?
Question 2: How such a matrix look like for convolutional architecture?

Pre-training Representations

Pre-training Representations

Pre-trained Representations

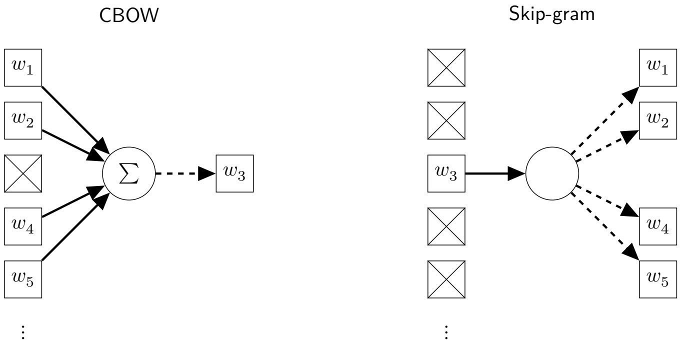
- representations that emerge in models seem to be carry a lot of information about the language
- representations pre-trained on large data can be re-used on tasks with smaller training data

Pre-training Representations

Word2Vec

Word2Vec

- way to learn word embeddings without training the complete LM



- CBOW: minimize cross-entropy of the middle word of a sliding windows
- skip-gram: minimize cross-entropy of a bag of words around a word (LM other way round)

Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia, jun 2013. Association for Computational Linguistics

Word2Vec: sampling

1.	**[All human beings]** are born free and equal in dignity ...	$\rightarrow$	(All, humans) (All, beings)
2.	**[All human beings are]** born free and equal in dignity ...	$\rightarrow$	(human, All) (human, beings) (human, are)
3.	**[All human beings are born]** free and equal in dignity ...	$\rightarrow$	(beings, All) (beings, human) (beings, are) (beings, born)
4.	All **[human beings are born free]** and equal in dignity ...	$\rightarrow$	(are, human) (are, beings) (are, born) (are, free)

Word2Vec: Formulas

- Training objective:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{j\sim(-c,c)}\log p(w_{t+c}|w_t)$$

- Probability estimation:

$$p(w_O|w_I) = \frac{\exp\left({V'}_{w_O}^{\top} V_{w_I}\right)}{\sum_w \exp\left({V'}_{w}^{\top} V_{w_i}\right)}$$

where V is input (embedding) matrix, V' output matrix

Equations 1, 2. Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia, jun 2013. Association for Computational Linguistics

Word2Vec: Training using Negative Sampling

The summation in denominator is slow, use noise contrastive estimation:

$$\log \sigma \left({V'}_{w_O}^{\top} V_{w_I} \right) + \sum_{i=1}^{k} E_{w_i \sim P_n(w)} \left[\log \sigma \left(-{V'}_{w_i}^{\top} V_{w_I} \right) \right]$$

Main idea: classify independently by logistic regression the positive and few sampled negative examples.

Equations 1, 3. Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia, jun 2013. Association for Computational Linguistics

Word2Vec: Vector Arithmetics

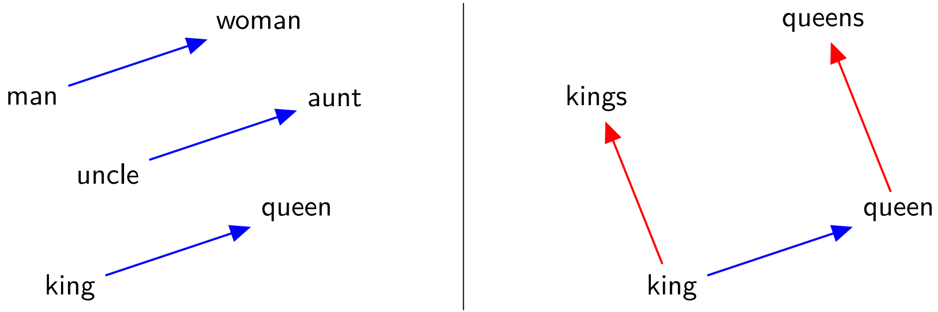


Image originally from Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia, jun 2013. Association for Computational Linguistics

Few More Notes on Embeddings

- many method for pre-trained words embeddings (most popluar GloVe)
- embeddings capturing character-level properties
- multilingual embeddings

Training models

FastText – Word2Vec model implementation by Facebook

`https://github.com/facebookresearch/fastText`

```
./fasttext skipgram -input data.txt -output model
```

Pre-training Representations

ELMo

What is ELMo?

- pre-trained large language model
- "nothing special" – combines all known tricks, trained on extremely large data
- improves almost all NLP tasks
- published in June 2018

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/N18-1202`

ELMo Architecture: Input

linear projection to 512 dimensions

↑

2× highway layer (2,048 dimensions)

↑

1D-convolution to 2,048 dimensions
+ max-pool

window	filters
1	32
2	32
3	64
4	128
5	256
6	512
7	1024

↑

character embeddings of size 16

- input tokenized, treated on character level
- 2,048 n-gram filters + max-pooling ($\sim$ soft search for learned n-grams)
- 2 highway layers:

$$
\begin{aligned}
g^{l+1} &= \sigma\left(W_g h^l + b_g\right) \\
h^{l+1} &= (1 - g^{l+1}) \odot h^l + \\
&\quad g^{l+1} \odot \mathrm{ReLu}\left(W h^l + b\right)
\end{aligned}
$$

contain gates that contol if projection is needed

ELMo Architecture: Language Models

- token representations input for 2 language models: forward and backward
- both LMs 2 layers with 4,096 dimensions wiht layer normalization and residual connections
- output classifier shared (only used in training, does hot have to be good)

Learned layer combination for downstream tasks:

$$\text{ELMo}_k^{\text{task}} = \gamma^{\text{task}} \sum_{\text{layer} L} s_L^{\text{task}} h_k^{(L)}$$

γ^{task}, s_L^{task} trainable parameters.

Task where ELMo helps

Answer Span Selection
Find an answer to a question in a unstructured text.

Semantic Role Labeling
Detect *who* did *what* to *whom* in sentences.

Natural Language Inference
Decide whether two sentences are in agreement, contradict each other, or have nothing to do with each other.

Named Entity Recognition
Detect and classify names people, locations, organization, numbers with units, email addresses, URLs, phone numbers ...

Coreference Resolution
Detect what entities pronouns refer to. I

Semantic Similarity
Measure how similar meaning two sentences are. (Think of clustering similar question on StackOverflow or detecting plagiarism.)

Improvements by Elmo

Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

How to use it

AllenNLP

- implemetned in AllenNLP framework (uses PyTorch)
- pre-trained English models available

```python
from allennlp.modules.elmo import Elmo,
    batch_to_ids

options_file = ...
weight_file = ...

elmo = Elmo(options_file, weight_file, 2,
            dropout=0)

sentences = [['First', 'sentence', '.'],
             ['Another', '.']]
character_ids = batch_to_ids(sentences)

embeddings = elmo(character_ids)
```

https://github.com/allenai/allennlp/blob/master/tutorials/how_to/elmo.md

Pre-training Representations

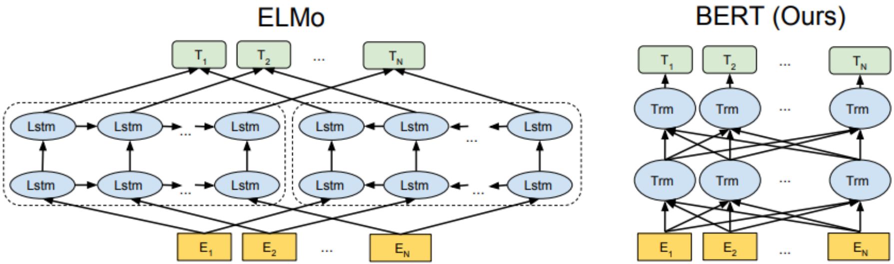
BERT

What is BERT

- another way of pretraining sentence representations
- uses Transformer architecture and slightly different training objective
- even beeter than ELMo
- done by Google, published in November 2018

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *ArXiv e-prints*, October 2018

Achitecture Comparison



Masked Language Model

All human being are born free free MASK hairy free and equal in dignity anc

1. Randomly sample a word → free
2. With 80% change replace with special MASK token.
3. With 10% change replace with random token → hairy
4. With 10% change keep asi is → free

Then a classifier should predict the missing/replaced word free

Additional Objective: Next Sentence Prediction

- trained in the multi-task learning setup
- secondary objective: next sentences prediction
- decide for a pair of consecuitve sentences whether they follow each other

Performance of BERT

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	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**91.1**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**81.9**

Table 1: GLUE Test results, scored by the GLUE evaluation server. 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. OpenAI GPT = (L=12, H=768, A=12); $\text{BERT}_{\text{BASE}}$ = (L=12, H=768, A=12); $\text{BERT}_{\text{LARGE}}$ = (L=24, H=1024, A=16). BERT and OpenAI GPT are single-model, single task. All results obtained from https://gluebenchmark.com/leaderboard and https://blog.openai.com/language-unsupervised/.

System	Dev EM	Dev F1	Test EM	Test F1
Leaderboard (Oct 8th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
#1 Single - nlnet	-	-	83.5	90.1
#2 Single - QANet	-	-	82.5	89.3
Published				
BiDAF+ELMo (Single)	-	85.8	-	-
R.M. Reader (Single)	78.9	86.3	79.5	86.6
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
$\text{BERT}_{\text{BASE}}$ (Single)	80.8	88.5	-	-
$\text{BERT}_{\text{LARGE}}$ (Single)	84.1	90.9	-	-
$\text{BERT}_{\text{LARGE}}$ (Ensemble)	85.8	91.8	-	-
$\text{BERT}_{\text{LARGE}}$ (Sgl.+TriviaQA)	**84.2**	**91.1**	**85.1**	**91.8**
$\text{BERT}_{\text{LARGE}}$ (Ens.+TriviaQA)	**86.2**	**92.2**	**87.4**	**93.2**

Tables 1 and 2. J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *ArXiv e-prints*, October 2018

Summary

1. Discrete symbols $\rightarrow$ continuous representation with trained embeddings
2. Architectures to get suitable representation: recurrent, convolutional, self-attentive
3. Output: classification, sequence labeling, autoregressive decoding
4. Representations pretrained on large data helps on downstream tasks

http://ufal.mff.cuni.cz/~zabokrtsky/fel