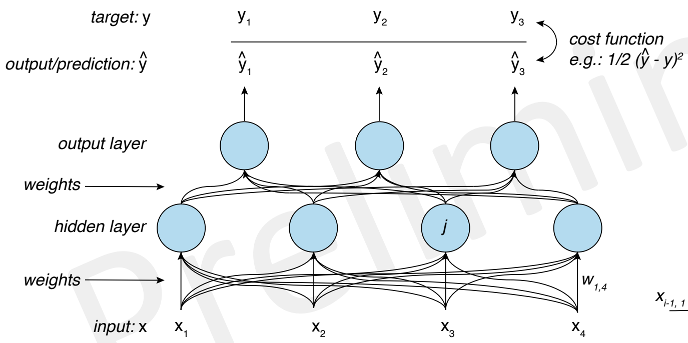
Outline

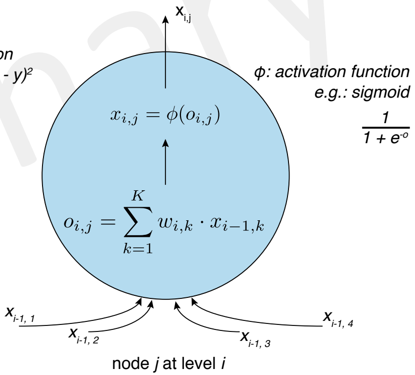
Preliminary

Outline

Preliminary

Multi-layer perceptron a.k.a. feedforward neural network



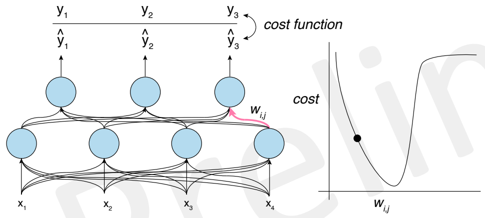


Preliminary

Outline

Preliminary

Back propagation



until convergence:
- do a forward pass
- compute the cost/error
- adjust weights $\leftarrow$ how??

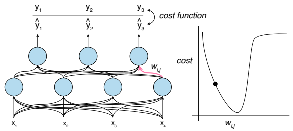
Adjust every weight $w_{i,j}$ by:

$$\Delta w_{i,j} = -\alpha \frac{\partial cost}{\partial w_{i,j}}$$

α is the learning rate.

Preliminary

Back propagation



$$cost(\hat{y}, y) = \frac{1}{2}(y - \hat{y})^2$$

$$\hat{y}_j = x_{i,j} = \phi(o_{i,j})$$

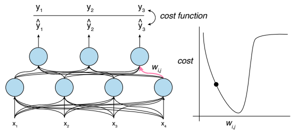
$$\Delta w_{i,j} = -\alpha \frac{\partial cost}{\partial w_{i,j}}$$

$$= -\alpha \frac{\partial cost}{\partial x_{i,j}} \frac{\partial x_{i,j}}{\partial w_{i,j}}$$

$\leftarrow$ chain rule

Preliminary

Back propagation



$$cost(\hat{y}, y) = \frac{1}{2}(y - \hat{y})^2$$

$$\hat{y_j} = x_{i,j} = \phi(o_{i,j}), \text{e.g. } \sigma(o_{i,j})$$

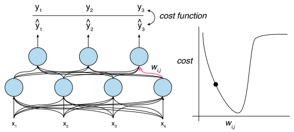
$$x_{i,j} = \sigma(o) = \frac{1}{1 + e^{-o}}$$

$$o_{i,j} = \sum_{k=1}^{K} w_{i,k} \cdot x_{i-1,k}$$

$$\begin{aligned} \Delta w_{i,j} &= -\alpha \; \frac{\partial cost}{\partial w_{i,j}} \\ &= -\alpha \; \frac{\partial cost}{\partial x_{i,j}} \; \frac{\partial x_{i,j}}{\partial w_{i,j}} \\ &= -\alpha \; \frac{\partial cost}{\partial x_{i,j}} \; \frac{\partial x_{i,j}}{\partial o_{i,j}} \; \frac{\partial o_{i,j}}{\partial w_{i,j}} \quad \leftarrow \text{chain rule} \end{aligned}$$

Preliminary

Back propagation



$$cost(\hat{y}, y) = \frac{1}{2}(y - \hat{y})^2$$

$$\hat{y_j} = x_{i,j} = \phi(o_{i,j}), \text{e.g. } \sigma(o_{i,j})$$

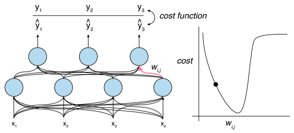
$$x_{i,j} = \sigma(o) = \frac{1}{1 + e^{-o}}$$

$$o_{i,j} = \sum_{k=1}^{K} w_{i,k} \cdot x_{i-1,k}$$

$$\begin{aligned}\Delta w_{i,j} &= -\alpha \frac{\partial cost}{\partial w_{i,j}} \\ &= -\alpha \frac{\partial cost}{\partial x_{i,j}} \frac{\partial x_{i,j}}{\partial w_{i,j}} \\ &= -\alpha \frac{\partial cost}{\partial x_{i,j}} \frac{\partial x_{i,j}}{\partial o_{i,j}} \frac{\partial o_{i,j}}{\partial w_{i,j}}\end{aligned}$$

Preliminary

Back propagation



$$cost(\hat{y}, y) = \frac{1}{2}(y - \hat{y})^2$$

$$\hat{y_j} = x_{i,j} = \phi(o_{i,j}), \text{e.g. } \sigma(o_{i,j})$$

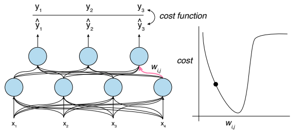
$$x_{i,j} = \sigma(o) = \frac{1}{1 + e^{-o}}$$

$$o_{i,j} = \sum_{k=1}^{K} w_{i,k} \cdot x_{i-1,k}$$

$$\begin{aligned} \Delta w_{i,j} &= -\alpha \frac{\partial cost}{\partial w_{i,j}} \\ &= -\alpha \frac{\partial cost}{\partial x_{i,j}} \frac{\partial x_{i,j}}{\partial w_{i,j}} \\ &= -\alpha \frac{\partial cost}{\partial x_{i,j}} \frac{\partial x_{i,j}}{\partial o_{i,j}} \frac{\partial o_{i,j}}{\partial w_{i,j}} \\ &= -\alpha \frac{\partial cost}{\partial x_{i,j}} \frac{\partial x_{i,j}}{\partial o_{i,j}} x_{i-1,j} \end{aligned}$$

Preliminary

Back propagation



$$cost(\hat{y}, y) = \frac{1}{2}(y - \hat{y})^2$$

$$\hat{y_j} = x_{i,j} = \phi(o_{i,j}), \text{e.g. } \sigma(o_{i,j})$$

$$x_{i,j} = \sigma(o) = \frac{1}{1+e^{-o}}$$

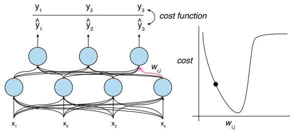
$$\sigma'(o) = \sigma(o)(1 - \sigma(o))$$

$$o_{i,j} = \sum_{k=1}^{K} w_{i,k} \cdot x_{i-1,k}$$

$$\begin{aligned}\Delta w_{i,j} &= -\alpha \frac{\partial cost}{\partial w_{i,j}} \\ &= -\alpha \frac{\partial cost}{\partial x_{i,j}} \frac{\partial x_{i,j}}{\partial w_{i,j}} \\ &= -\alpha \frac{\partial cost}{\partial x_{i,j}} \frac{\partial x_{i,j}}{\partial o_{i,j}} \frac{\partial o_{i,j}}{\partial w_{i,j}} \\ &= -\alpha \frac{\partial cost}{\partial x_{i,j}} x_{i,j}(1 - x_{i,j}) \; x_{i-1,j}\end{aligned}$$

Preliminary

Back propagation



$$cost(\hat{y}, y) = \frac{1}{2}(y - \hat{y})^2$$

$$\hat{y}_j = x_{i,j} = \phi(o_{i,j}), \text{e.g. } \sigma(o_{i,j})$$

$$x_{i,j} = \sigma(o) = \frac{1}{1 + e^{-o}}$$

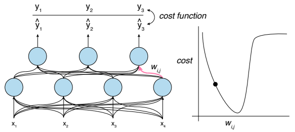
$$\sigma'(o) = \sigma(o)(1 - \sigma(o))$$

$$o_{i,j} = \sum_{k=1}^{K} w_{i,k} \cdot x_{i-1,k}$$

$$\begin{aligned}
\Delta w_{i,j} &= -\alpha \frac{\partial cost}{\partial w_{i,j}} \\
&= -\alpha \frac{\partial cost}{\partial x_{i,j}} \frac{\partial x_{i,j}}{\partial w_{i,j}} \\
&= -\alpha \frac{\partial cost}{\partial x_{i,j}} \frac{\partial x_{i,j}}{\partial o_{i,j}} \frac{\partial o_{i,j}}{\partial w_{i,j}} \\
&= -\alpha \; y_j - x_{i,j} \;\; x_{i,j}(1 - x_{i,j}) \;\; x_{i-1,j}
\end{aligned}$$

Preliminary

Back propagation



$$cost(\hat{y}, y) = \frac{1}{2}(y - \hat{y})^2$$

$$\hat{y}_j = x_{i,j} = \phi(o_{i,j}), \text{e.g. } \sigma(o_{i,j})$$

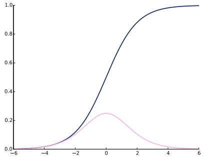
$$x_{i,j} = \sigma(o) = \frac{1}{1 + e^{-o}}$$

$$\sigma'(o) = \sigma(o)(1 - \sigma(o))$$

$$o_{i,j} = \sum_{k=1}^{K} w_{i,k} \cdot x_{i-1,k}$$

$$\begin{aligned}
\Delta w_{i,j} &= -\alpha \frac{\partial cost}{\partial w_{i,j}} \\
&= -\alpha \frac{\partial cost}{\partial x_{i,j}} \frac{\partial x_{i,j}}{\partial w_{i,j}} \\
&= -\alpha \frac{\partial cost}{\partial x_{i,j}} \frac{\partial x_{i,j}}{\partial o_{i,j}} \frac{\partial o_{i,j}}{\partial w_{i,j}} \\
&= -\alpha \quad y_j - x_{i,j} \quad x_{i,j}(1 - x_{i,j}) \quad x_{i-1,j} \\
&= \text{l.rate} \quad cost \quad activation \quad input
\end{aligned}$$

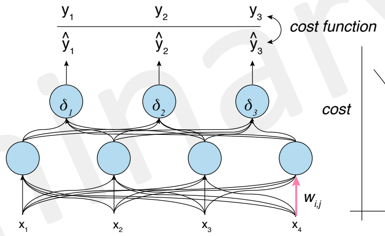
$\sigma(o)$ $\sigma'(o)$

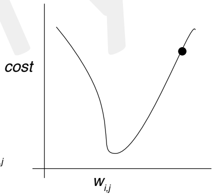


Preliminary

Back propagation

$$\begin{aligned}\Delta w_{i,j} &= -\alpha \frac{\partial cost}{\partial w_{i,j}} \\ &= -\alpha \frac{\partial cost}{\partial x_{i,j}} \frac{\partial x_{i,j}}{\partial o_{i,j}} \frac{\partial o_{i,j}}{\partial w_{i,j}} \\ &= \text{l.rate} \quad cost \; activation \; input \\ &= -\alpha \frac{\partial cost}{\partial x_{i,j}} \frac{\partial x_{i,j}}{\partial o_{i,j}} \frac{\partial o_{i,j}}{\partial w_{i,j}} \\ &= -\alpha \; \delta \; x_{i-1,j}\end{aligned}$$



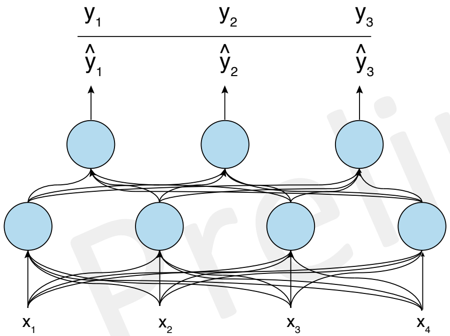


$$\delta_{output} = (y_j - x_{i,j}) \; x_{i,j}(1 - x_{i,j}) \qquad \leftarrow \text{previous slide}$$

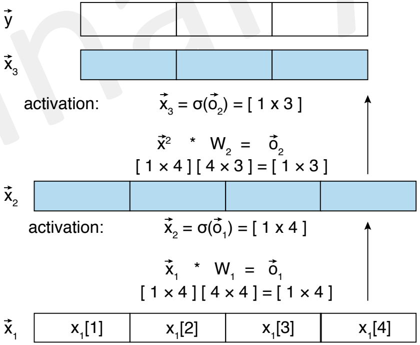
$$\delta_{hidden} = \left(\sum_{n \in nodes} \delta_n w_{n,j} \right) \; x_{i,j}(1 - x_{i,j})$$

Preliminary

Network representation



$\leftrightarrow$



$\vec{y}$
$\vec{x}_3$
activation: $\vec{x}_3 = \sigma(\vec{o}_2) = [\ 1 \times 3\ ]$
$\vec{x}^2 \ * \ W_2 \ = \ \vec{o}_2$
$[\ 1 \times 4\ ]\,[\ 4 \times 3\ ] = [\ 1 \times 3\ ]$
$\vec{x}_2$
activation: $\vec{x}_2 = \sigma(\vec{o}_1) = [\ 1 \times 4\ ]$
$\vec{x}_1 \ * \ W_1 \ = \ \vec{o}_1$
$[\ 1 \times 4\ ]\,[\ 4 \times 4\ ] = [\ 1 \times 4\ ]$
$\vec{x}_1$
$x_1[1]$ $x_1[2]$ $x_1[3]$ $x_1[4]$

Preliminary

Outline

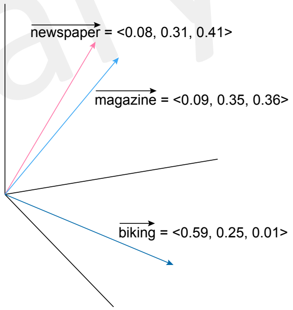
Preliminary

Distributed representations

- Represent units, e.g., words, as vectors
- Goal: words that are similar, e.g., in terms of meaning, should get similar embeddings

Cosine similarity to determine how similar two vectors are:

$$cosine(\vec{v}, \vec{w}) = \frac{\vec{v}^\top \cdot \vec{w}}{||\vec{v}||_2 ||\vec{w}||_2} = \frac{\sum_{i=1}^{|v|} v_i * w_i}{\sqrt{\sum_{i=1}^{|v|} v_i^2}\sqrt{\sum_{i=1}^{|w|} w_i^2}}$$


$\overrightarrow{newspaper}$ = <0.08, 0.31, 0.41>
$\overrightarrow{magazine}$ = <0.09, 0.35, 0.36>
$\overrightarrow{biking}$ = <0.59, 0.25, 0.01>

Preliminary

Distributed representations

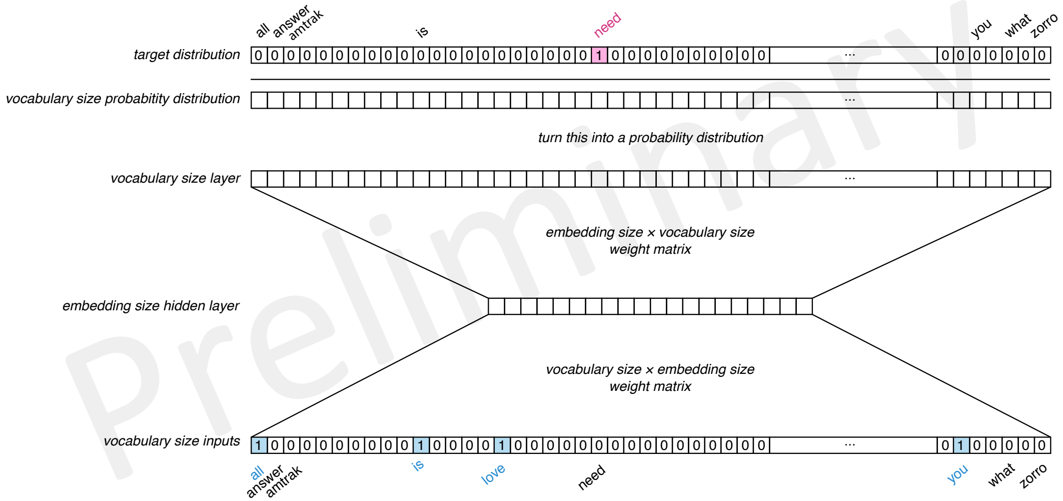
How do we get these vectors?

- You shall know a word by the company it keeps [Firth, 1957]
- The vector of a word should be similar to the vectors of the words surrounding it

$$\overrightarrow{all}\ \overrightarrow{you}\ \overrightarrow{need}\ \overrightarrow{is}\ \overrightarrow{love}$$

Preliminary

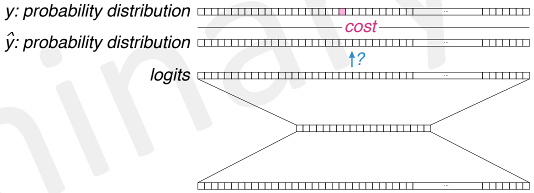
Embedding methods



Preliminary

Probability distributions

$$\text{softmax} = \text{normalize the logits}$$
$$= \frac{e^{logits[i]}}{\sum_{j=1}^{|logits|} e^{logits[j]}}$$
$$cost = \text{cross entropy loss}$$
$$= -\sum_x p(x) \log \hat{p}(x)$$
$$= -\sum_i p_{\text{ground truth}}(word = vocabulary[i]) \log p_{\text{predictions}}(word = vocabulary[i])$$
$$= -\sum_i y_i \log \hat{y}_i$$



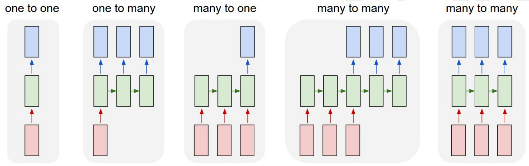
Preliminary

Outline

Preliminary

Recurrent neural networks

- Lots of information is sequential and requires a memory for successful processing
- Sequences as input, sequences as output



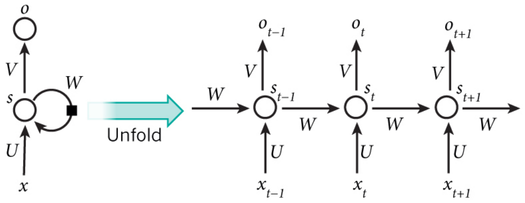
- Recurrent neural networks (RNNs) are called recurrent because they perform same task for every element of sequence, with output dependent on previous computations
- RNNs have memory that captures information about what has been computed so far
- RNNs can make use of information in arbitrarily long sequences – in practice they limited to looking back only few steps

Image credits: `http://karpathy.github.io/assets/rnn/diags.jpeg`

Preliminary

Recurrent neural networks

- RNN being unrolled (or unfolded) into full network
- Unrolling: write out network for complete sequence



- Formulas governing computation:
  - x_t input at time step t
  - s_t hidden state at time step t – memory of the network, calculated based on previous hidden state and input at the current step: $s_t = f(Ux_t + Ws_{t-1})$; f usually nonlinearity, e.g., $\tanh$ or ReLU; s_{-1} typically initialized to all zeroes
  - o_t output at step t. E.g.,, if we want to predict next word in sentence, a vector of probabilities across vocabulary: $o_t = \mathrm{softmax}(Vs_t)$

Image credits: Nature

Prelim.

Language modeling using RNNs

- Language model allows us to predict probability of observing sentence (in a given dataset) as: $P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1})$
- In RNN, set $o_t = x_{t+1}$: we want output at step t to be actual next word
- Input x a sequence of words; each x_t is a single word; we represent each word as a one-hot vector of size `vocabulary_size`
- Initialize parameters U, V, W to small random values around 0

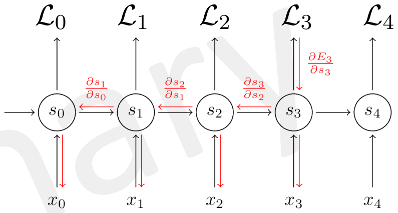
Preliminary

Language modeling using RNNs

- Language model allows us to predict probability of observing sentence (in a given dataset) as: $P(w_1, \ldots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \ldots, w_{i-1})$
- In RNN, set $o_t = x_{t+1}$: we want output at step t to be actual next word
- Input x a sequence of words; each x_t is a single word; we represent each word as a one-hot vector of size `vocabulary_size`
- Initialize parameters U, V, W to small random values around 0
- Cross-entropy loss as loss function
- For N training examples (words in text) and C classes (the size of our vocabulary), loss with respect to predictions o and true labels y is: $\mathcal{L}(y, o) = -\frac{1}{N} \sum_{n \in N} y_n \log o_n$
- Training RNN similar to training a traditional NN: backpropagation algorithm, but with small twist
- Parameters shared by all time steps, so gradient at each output depends on calculations of previous time steps: Backpropagation Through Time

Preliminary

Vanishing and exploding gradients



$\mathcal{L}_0$
$\mathcal{L}_1$
$\mathcal{L}_2$
$\mathcal{L}_3$
$\mathcal{L}_4$
$\frac{\partial E_3}{\partial s_3}$
$\frac{\partial s_1}{\partial s_0}$
$\frac{\partial s_2}{\partial s_1}$
$\frac{\partial s_3}{\partial s_2}$
s_0
s_1
s_2
s_3
s_4
x_0
x_1
x_2
x_3
x_4

- For training RNNs, calculate gradients for U, V, W – ok for V but for W and U ...
- Gradients for W:

$$\frac{\partial \mathcal{L}_3}{\partial W} = \frac{\partial \mathcal{L}_3}{\partial o_3}\frac{\partial o_3}{\partial s_3}\frac{\partial s_3}{\partial W} \quad = \sum_{k=0}^{3} \frac{\partial \mathcal{L}_3}{\partial o_3}\frac{\partial o_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial W}$$

- More generally: $\frac{\partial \mathcal{L}}{\partial s_t} = \frac{\partial \mathcal{L}}{\partial s_m} \cdot \underbrace{\frac{\partial s_m}{\partial s_{m-1}}}_{<1} \cdot \underbrace{\frac{\partial s_{m-1}}{\partial s_{m-2}}}_{<1} \cdot \cdots \cdot \underbrace{\frac{\partial s_{t+1}}{\partial s_t}}_{<1} \Rightarrow \ll 1$
- Gradient contributions from far away steps become zero: state at those steps doesn't contribute to what you are learning

Image credits: `http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-grad`

Preliminary

Long Short Term Memory [Hochreiter and Schmidhuber, 1997]

LSTMs designed to combat vanishing gradients through gating mechanism

- How LSTM calculates hidden state s_t

$$\begin{aligned} i &= \sigma(x_t U^i + s_{t-1} W^i) \\ f &= \sigma(x_t U^f + s_{t-1} W^f) \\ o &= \sigma(x_t U^o + s_{t-1} W^o) \\ g &= \tanh(x_t U^g + s_{t-1} W^g) \\ c_t &= c_{t-1} \circ f + g \circ i \\ s_t &= \tanh(c_t) \circ o \end{aligned}$$

  ($\circ$ is elementwise multiplication)

- RNN computes hidden state as $s_t = \tanh(U x_t + W s_{t-1})$ – an LSTM unit does exact same thing

Preliminary

Long Short Term Memory [Hochreiter and Schmidhuber, 1997]

LSTMs designed to combat vanishing gradients through gating mechanism

- How LSTM calculates hidden state s_t

$$
\begin{aligned}
i &= \sigma(x_t U^i + s_{t-1} W^i) \\
f &= \sigma(x_t U^f + s_{t-1} W^f) \\
o &= \sigma(x_t U^o + s_{t-1} W^o) \\
g &= \tanh(x_t U^g + s_{t-1} W^g) \\
c_t &= c_{t-1} \circ f + g \circ i \\
s_t &= \tanh(c_t) \circ o
\end{aligned}
$$

($\circ$ is elementwise multiplication)

- RNN computes hidden state as $s_t = \tanh(U x_t + W s_{t-1})$ – an LSTM unit does exact same thing

- i, f, o: input, forget and output gates
- Gates optionally let information through: composed out of sigmoid neural net layer and pointwise multiplication operation
- g is a candidate hidden state computed based on current input and previous hidden state
- c_t is internal memory of LSTM unit: combines previous memory c_{t-1} multiplied by forget gate, and newly computed hidden state g, multiplied by input gate

Preliminary

Long Short Term Memory [Hochreiter and Schmidhuber, 1997]

LSTMs designed to combat vanishing gradients through gating mechanism

- How LSTM calculates hidden state s_t

$$
\begin{aligned}
i &= \sigma(x_t U^i + s_{t-1} W^i) \\
f &= \sigma(x_t U^f + s_{t-1} W^f) \\
o &= \sigma(x_t U^o + s_{t-1} W^o) \\
g &= \tanh(x_t U^g + s_{t-1} W^g) \\
c_t &= c_{t-1} \circ f + g \circ i \\
s_t &= \tanh(c_t) \circ o
\end{aligned}
$$

  ($\circ$ is elementwise multiplication)

- RNN computes hidden state as $s_t = \tanh(U x_t + W s_{t-1})$ – an LSTM unit does exact same thing
- . . .
- Compute output hidden state s_t by multiplying memory with output gate
- Plain RNNs a special case of LSTMs:
  - Fix input gate to all 1's
  - Fix forget gate to all 0's (always forget the previous memory)
  - Fix output gate to all 1's (expose the whole memory)
  - Additional $\tanh$ squashes output
- Gating mechanism allows LSTMs to model long-term dependencies
- Learn parameters for gates, to learn how memory should behave

Preliminary

Gated Recurrent Units

- GRU layer quite similar to that of LSTM layer, as are the equations:

$$\begin{aligned} z &= \sigma(x_t U^z + s_{t-1} W^z) \\ r &= \sigma(x_t U^r + s_{t-1} W^r) \\ h &= \tanh(x_t U^h + (s_{t-1} \circ r) W^h) \\ s_t &= (1 - z) \circ h + z \circ s_{t-1} \end{aligned}$$

- GRU has two gates: reset gate r and update gate z.
  - Reset gate determines how to combine new input with previous memory; update gate defines how much of the previous memory to keep around
  - Set reset to all 1's and update gate to all 0's to get plain RNN model
- On many tasks, LSTMs and GRUs perform similarly

Preliminary

Bidirectional RNNs

- Bidirectional RNNs based on idea that output at time t may depend on previous and future elements in sequence
  - Example: predict missing word in a sequence
- Bidirectional RNNs are two RNNs stacked on top of each other
- Output is computed based on hidden state of both RNNs

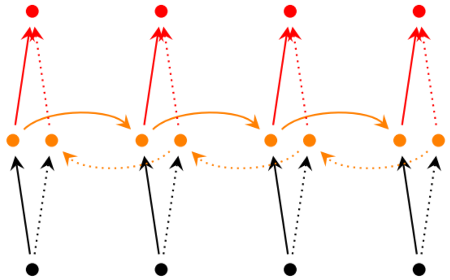


Image credits: `http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/`

Prelim

Attention

- Attention mechanisms come from visual analysis — suppose you want to locate a specific object in a large image; like people, focus on specific areas
- First applied to text and NLP in 2015
- Basic mechanism behind every attention mechanism
  1. Read operator: read a “patch” from the input
  2. Glimpse sensor: extract information from “patch”
  3. Locator: predict the next location of read operator
  4. RNN: combine the previous and current responses from glimpse sensor

Assume we are at time step t. From $t-1$ we get next location to which we should pay attention (produced by locator). Move sensor there and extract information, which the RNN combines with previous outputs. After several iterations, we produce final response, e.g., classification or label.

Preliminary

Attention

Hard attention

- Read operator: fixed size, but there may be several of them
- Glimpse sensor can be any NN
- Locator predicts x and y location of sensor (images) or words ahead/previous (text)

Not differentiable, so Reinforcement Learning is used

Soft attention

- Read operator: it only has fixed aspect ratio; it can zoom into the image and blur if needed
- Glimpse sensor can be any NN
- Locator predict more than x and y parameter (like sigma for blur .etc)

Differentiable

Preliminary

Outline

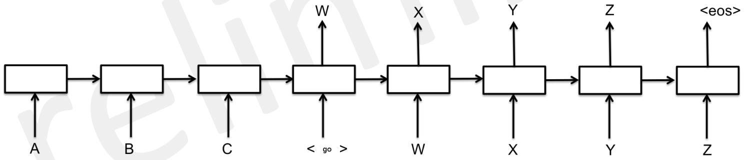
Preliminary

Sequence-to-sequence models

Increasingly important: not just retrieval but also generation

- Snippets, summaries, small screen versions of search results, spoken results, chatbots, conversational interfaces, . . . , but also query suggestion, query correction, . . .

Basic sequence-to-sequence (seq2seq) model consists of two RNNs: an encoder that processes input and a decoder that generates output:



Each box represents cell of RNN (often GRU cell or LSTM cell). Encoder and decoder can share weights or, as is more common, use a different set of parameters

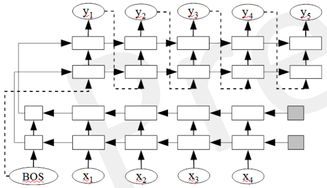
Image credits: `https://www.tensorflow.org/tutorials/seq2seq`

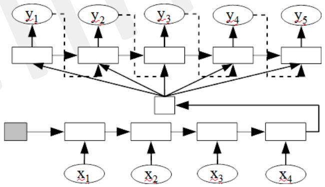
Preliminary

Sequence-to-sequence models

Preliminary

- seq2seq models build on top of language models
  - Encoder step: a model converts input sequence into a fixed representation
  - Decoder step: a language model is trained on both the output sequence (e.g., translated sentence) as well as fixed representation from the encoder
  - Since decoder model sees encoded representation of input sequence as well as output sequence, it can make more intelligent predictions about future words based on current word





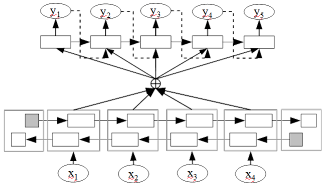


Image credits: [Sutskever et al., 2014; Cho et al., 2014; Bahdanou et al., 2014]

Sequence-to-sequence models

Used for a "traditional information retrieval task"

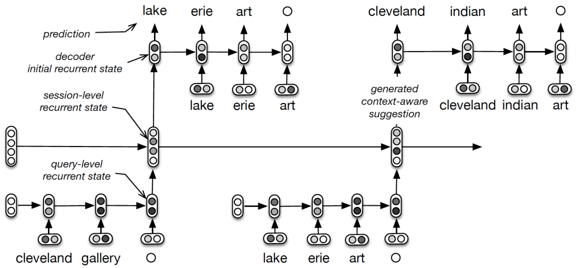

prediction
decoder initial recurrent state
session-level recurrent state
query-level recurrent state
generated context-aware suggestion
lake
erie
art
O
cleveland
indian
art
O
lake
erie
art
cleveland
indian
art
cleveland
gallery
O
lake
erie
art
O

Figure 3: The hierarchical recurrent encoder-decoder (HRED) for query suggestion. Each arrow is a non-linear transformation. The user types *cleveland gallery* → *lake erie art*. During training, the model encodes *cleveland gallery*, updates the session-level recurrent state and maximize the probability of seeing the following query *lake erie art*. The process is repeated for all queries in the session. During testing, a contextual suggestion is generated by encoding the previous queries, by updating the session-level recurrent states accordingly and by sampling a new query from the last obtained session-level recurrent state. In the example, the generated contextual suggestion is *cleveland indian art*.

Image credits: Sordoni et al. [2015a]

Outline

Preliminary

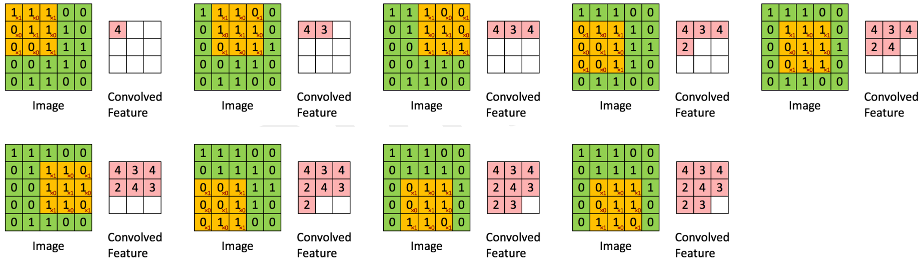
Convolutional neural networks

Major breakthroughs in image classification – at core of many computer visions systems

Some initial applications of CNNs to problems in text and information retrieval

What is a convolution? Intuition: sliding window function applied to a matrix

Example: convolution with 3×3 filter



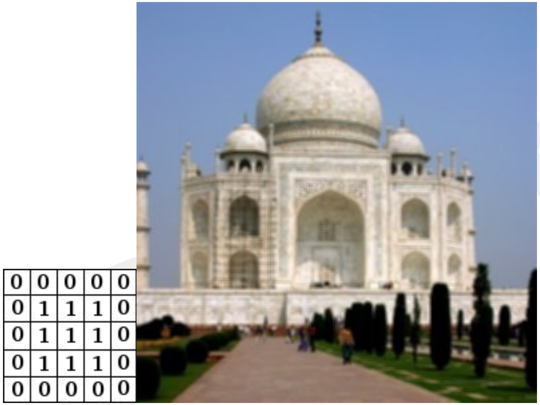
Multiply values element-wise with original matrix, then sum. Slide over whole matrix.

Image credits:

`http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution`

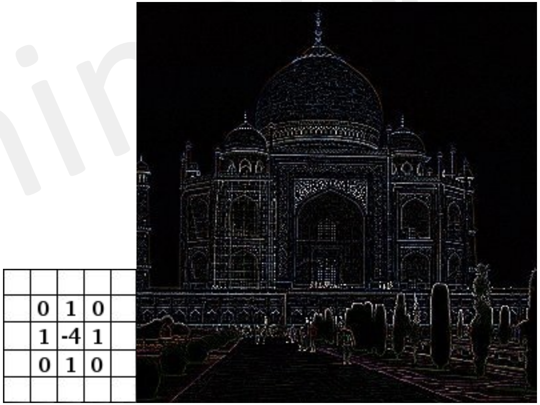
Visual examples of CNNs

Averaging each pixel with neighboring values blurs image:



0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

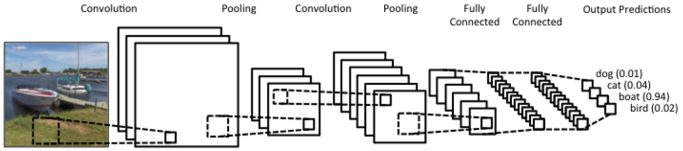
Taking difference between pixel and its neighbors detects edges:



	0	1	0	
	1	-4	1	
	0	1	0	

Image credits: `https://docs.gimp.org/en/plug-in-convmatrix.html`

Convolutional neural networks



- Use convolutions over input layer to compute output
- Yields local connections: each region of input connected to a neuron in output
- Each layer applies different filters and combines results
- Pooling (subsampling) layers
- During training, CNN learns values of filters

- Image classification a CNN may learn to detect edges from raw pixels in first layer
- Then use edges to detect simple shapes in second layer
- Then use shapes to detect higher-level features, such as facial shapes in higher layers
- Last layer is then a classifier that uses high-level features

Preliminary

CNNs in text

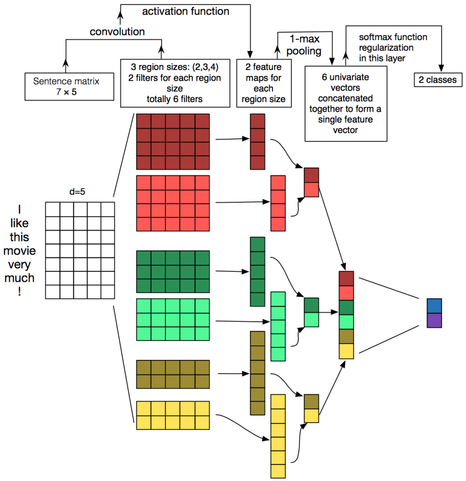
Basic intution

- Instead of image pixels, input to most NLP tasks are sentences or documents represented as a matrix. Each row of matrix corresponds to one token, typically a word, but could be a character. That is, each row is vector that represents word.
- Typically, these vectors are word embeddings (low-dimensional representations) like word2vec or GloVe, but they could also be one-hot vectors that index the word into a vocabulary.
- For a 10 word sentence using a 100-dimensional embedding we would have a 10×100 matrix as our input.
- That's our "image"
- Typically use filters that slide over full rows of the matrix (words): the "width" of our filters is usually the same as the width of the input matrix. The height, or region size, may vary, but sliding windows over 2-5 words at a time is typical.

Preliminary

CNNs in text

Example architecture (Zhang and Wallace, 2015; Sentence classification)


activation function
convolution
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
1-max pooling
6 univariate vectors concatenated together to form a single feature vector
softmax function regularization in this layer
2 classes
d=5
I
like
this
movie
very
much
!

Preliminary

CNNs in text

preliminary

Example uses in IR

- MSR: how to learn semantically meaningful representations of sentences that can be used for Information Retrieval
- Recommending potentially interesting documents to users based on what they are currently reading
- Sentence representations are trained based on search engine log data
- Gao et al. Modeling Interestingness with Deep Neural Networks. EMNLP 2014; Shen et al. A Latent Semantic Model with Convolutional-Pooling Structure for Information Retrieval. CIKM 2014.