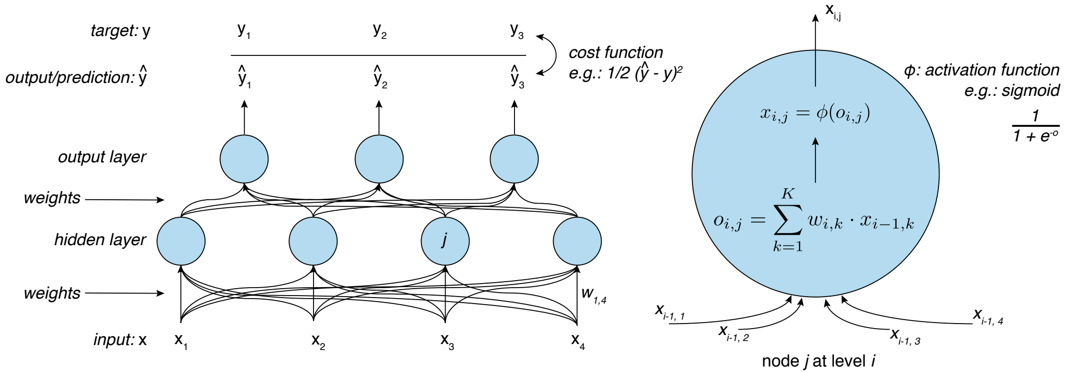
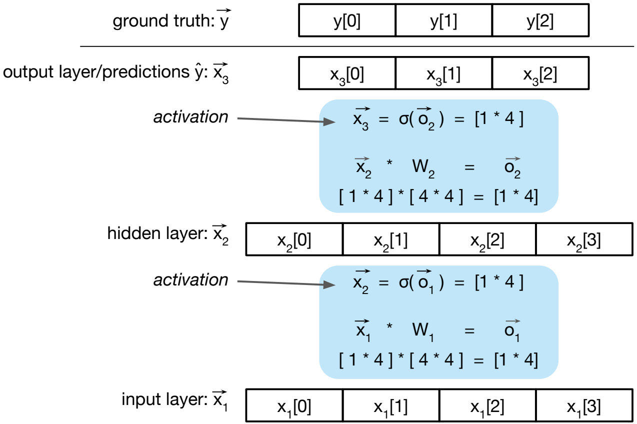
Outline

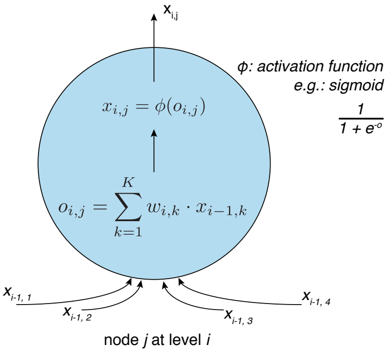
Outline

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



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





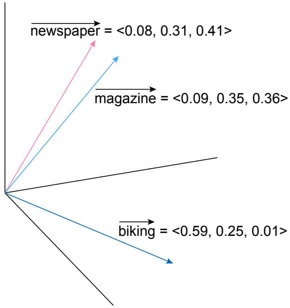
Outline

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:

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



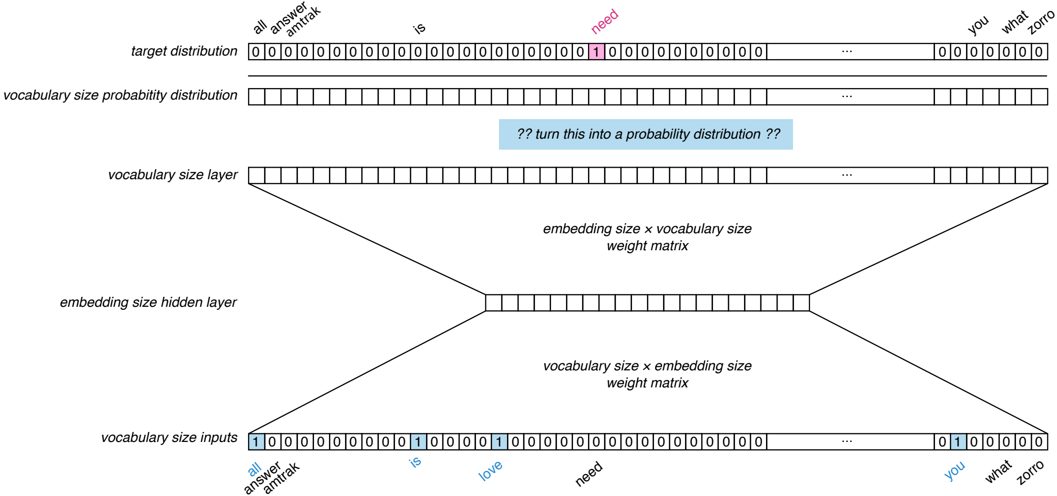
Distributed representations

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

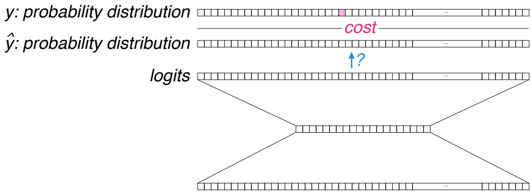
$$\overrightarrow{all}\ \ \overrightarrow{you}\ \ \overrightarrow{need}\ \ \overrightarrow{is}\ \ \overrightarrow{love}$$

Embedding methods



Probability distributions

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

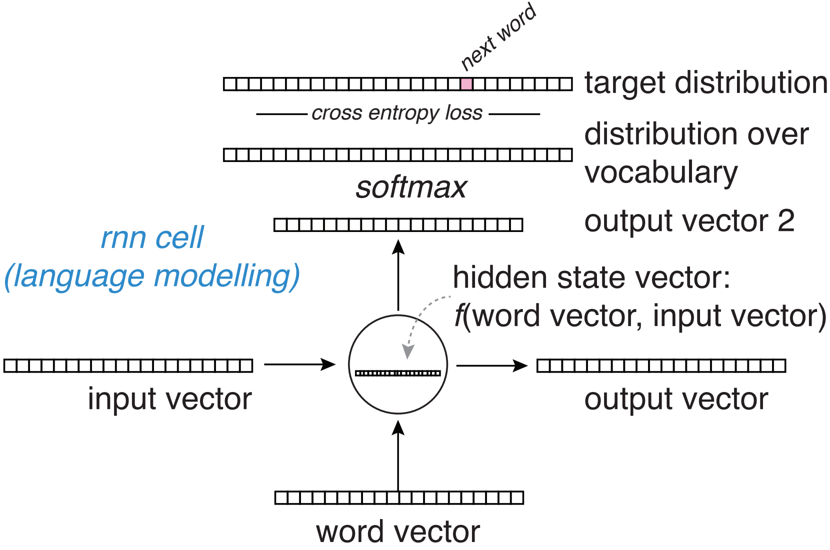


Outline

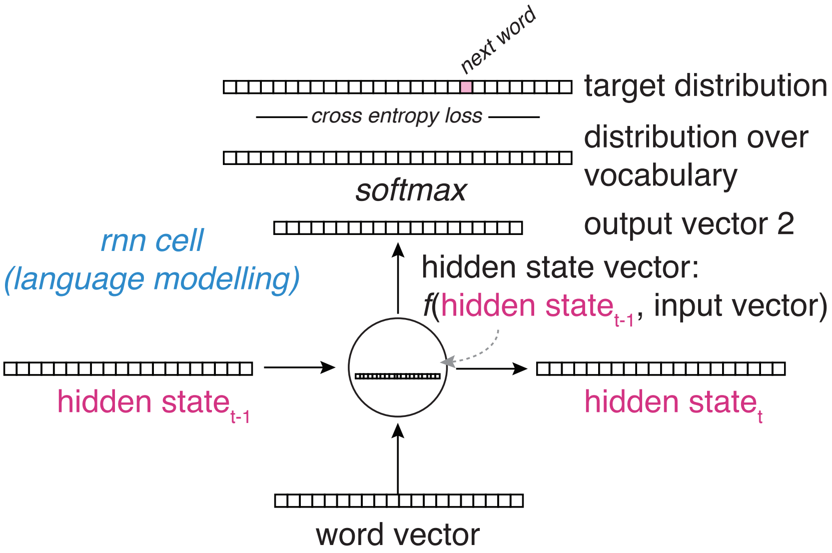
Recurrent neural networks

- Recurrent neural networks (RNNs) are typically used in scenarios where a sequence of inputs and/or outputs is being modelled
- RNNs have memory that captures information about what has been computed so far
- RNNs are called recurrent because they perform same task for every element of sequence, with output dependent on previous computations
- RNNs can, in theory, make use of information in arbitrarily long sequences – in practice, however, they are limited to looking back only few steps

Recurrent cell

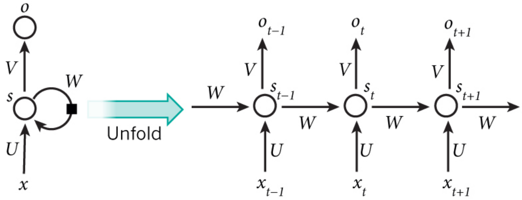


Recurrent cell



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
  - input at the current step
  - previous hidden state

$$s_t = f(Ux_t + Ws_{t-1})$$

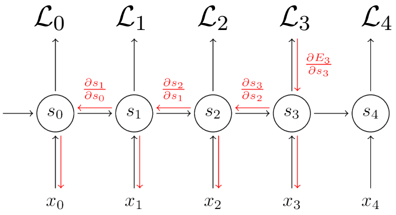
- f usually nonlinearity, e.g., $\tanh$ or ReLU
  f can also be LSTM or GRU.

Image credits: Nature

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
- Cross-entropy loss as loss function
- 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

Vanishing and exploding gradients



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

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

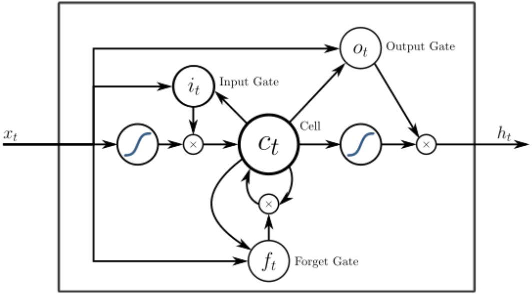
- More generally: $\frac{\partial \mathcal{L}}{\partial s_t} = \frac{\partial \mathcal{L}}{\partial s_m} \cdot \underbrace{\frac{\partial s_m}{\partial s_{m-1}}}_{<1} \cdot \underbrace{\frac{\partial s_{m-1}}{\partial s_{m-2}}}_{<1} \cdot \dots \cdot \underbrace{\frac{\partial s_{t+1}}{\partial s_t}}_{<1} \Rightarrow \ll 1$
- 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`

Long Short Term Memory [Hochreiter and Schmidhuber, 1997]

LSTMs designed to combat vanishing gradients through gating mechanism

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



($\circ$ is elementwise multiplication)

Image credits:
https://commons.wikimedia.org/wiki/File:Peephole_Long_Short-Term_Memory.svg

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

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

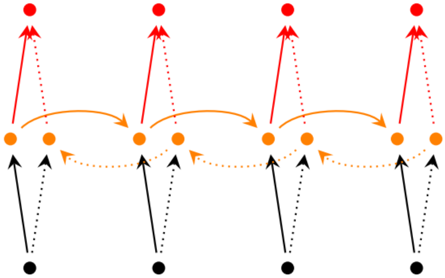


Image credits: `http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/`

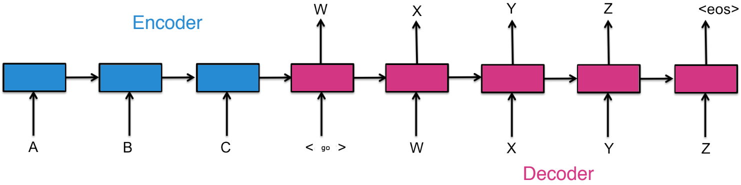
Outline

Sequence-to-sequence models

Increasingly important: not just retrieval but also generation

- Machine translation, spoken results, chatbots, conversational interfaces, . . . , but also snippets, 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:



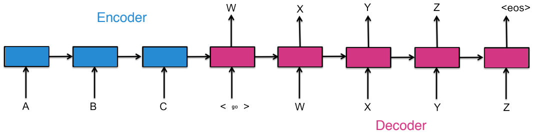
Each box represents cell of RNN (often GRU cell or LSTM cell). Encoder and decoder can share weights or, as is more common, use a different set of parameters

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

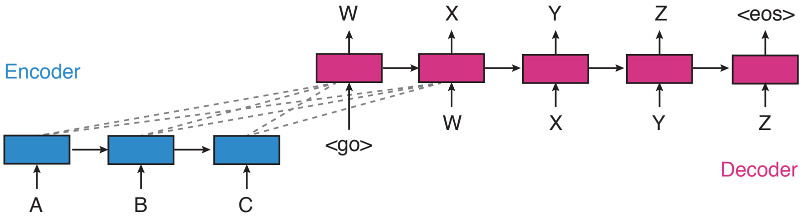
Attention mechanism Bahdanau et al. [2014]

Normal decoder model:

$$\mathbf{h}_t = f(\mathbf{x}, \mathbf{h}_{t-1}; \theta),$$

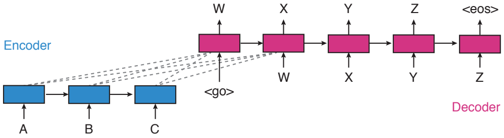


Decoder with attention:



$$\mathbf{h}_t^{dec} = g(\mathbf{x}^{dec}, \mathbf{H}^{encoder}, \mathbf{h}_{t-1}^{dec})$$

Attention mechanism



Decoder with attention Luong and Manning [2016], Vinyals et al. [2015]:

$$\mathbf{h}_t^{dec} = g(\mathbf{x}^{dec}, \mathbf{H}^{encoder}, \mathbf{h}_{t-1}^{dec})$$
$$= \mathbf{W}_{proj} \cdot \mathbf{d}_t || \hat{\mathbf{h}}_t^{dec},$$

$||$ is the concatenation operator, and $\hat{\mathbf{h}}_t^{dec} = f(\mathbf{x}^{dec}, \mathbf{h}_{t-1}^{dec}; \theta^{dec})$ from above.

$\mathbf{d}_t$ is calculated from $\mathbf{H}^{encoder}$ by:

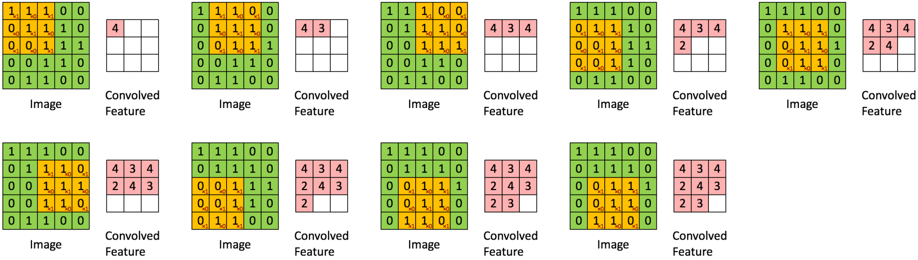
$$\mathbf{d}_t = \sum_{i=1}^{n} a_{t,i} \mathbf{h}_i^{encoder}$$
$$\mathbf{a}_t = \mathit{softmax}(\mathbf{u}_t)$$
$$\mathbf{u}_{t,i} = v^T \tanh(\mathbf{W}_1 \mathbf{h}_i^{encoder} + \mathbf{W}_2 \mathbf{h}_t^{dec}),$$

Outline

Convolutional neural networks

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

Example: convolution with 3×3 filter

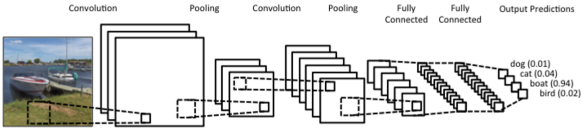


Multiply values element-wise with original matrix, then sum. Slide over whole matrix.

Image credits:

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

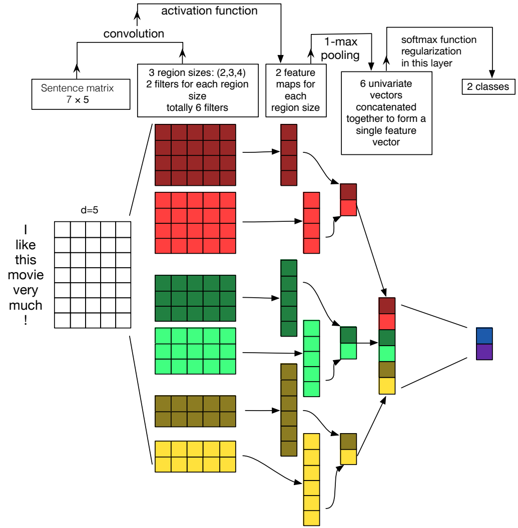
Convolutional neural networks



- Each layer applies different filters and combines results
- Pooling (subsampling) layers
- During training, CNN learns values of filters

- First layer may learn to detect edges from raw pixels in first layer
- Use edges to detect simple shapes in second layer
- Use shapes to detect higher-level features, such as facial shapes in higher layers
- Last layer: classifier using high-level features

CNNs in text



Architecture for sentiment analysis Zhang and Wallace [2015]

- Instead of image pixels, inputs typically are word embeddings.
- 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 1-dimensional convolutions are used. I.e. filters slide over rows of the matrix (words).

CNNs in text

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
- Modeling interestingness with deep neural networks Gao et al. [2014]
  A latent semantic model with convolutional-pooling structure for information retrieval [Shen et al., 2014]

Take aways

1. Information Retrieval (IR) systems help people find the right (most useful) information in the right (most convenient) format at the right time (when they need it).
2. Neural Network (NN) is a function $F(x; \Theta)$ with (a large number of) parameters Θ that maps an input object x (which can be text, image or arbitrary vector of features) to an output object y (class label, sequence of class labels, text, image).
3. There three main architectures (classes of $F(x; \Theta)$): (i) feed-forward NN (FFNN), (ii) recurrent NN (RNN), (iii) convolutional NN (CNN).
4. Embeddings are vector representations of objects in a high-dimensional space that are learned during training. In many practical applications, these vectors reflect similarities between objects that are important for solving the task.
5. Other stuff, such as seq2seq, vanishing and exploding gradients, LSTM and GRU, attention mechanism, softmax, cross-entropy.