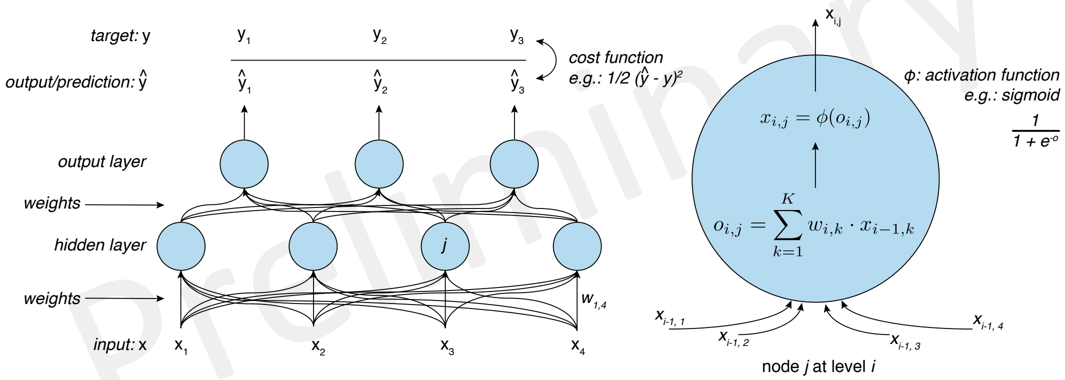
Outline

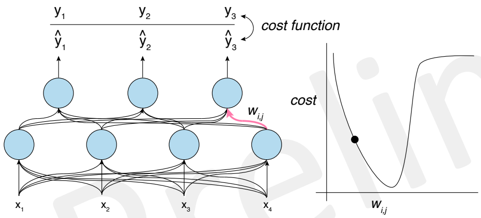
- Morning program
  - Preliminaries
    - Feedforward neural network
    - Back propagation
    - Distributed representations
    - Recurrent neural networks
    - Sequence-to-sequence models
    - Convolutional neural networks
  - Text matching I
  - Text matching II
- Afternoon program
  - Learning to rank
  - Modeling user behavior
  - Generating responses
  - Outlook
  - Wrap up

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



Outline

Back propagation



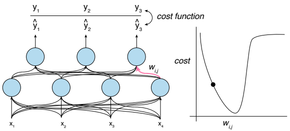
until convergence:
- do a forward pass
- compute the cost/error
- adjust weights ← how??

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

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

α is the learning rate.

Back propagation



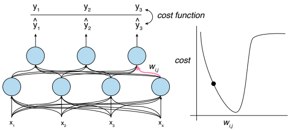
$$cost(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$$

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

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

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

Back propagation



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

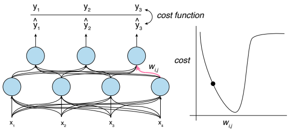
$$\hat{y_j} = x_{i,j} = \phi(o_{i,j}), \text{e.g. } \sigma(o_{i,j})$$

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

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

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

Back propagation



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

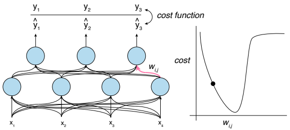
$$\hat{y}_j = x_{i,j} = \phi(o_{i,j}), \text{e.g. } \sigma(o_{i,j})$$

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

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

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

Back propagation



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

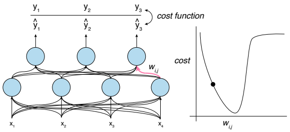
$$\hat{y}_j = x_{i,j} = \phi(o_{i,j}), \text{e.g. } \sigma(o_{i,j})$$

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

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

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

Back propagation



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

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

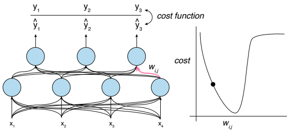
$$x_{i,j} = \sigma(o) = \frac{1}{1 + e^{-o}}$$

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

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

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

Back propagation



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

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

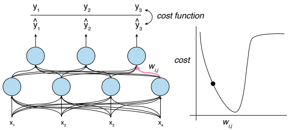
$$x_{i,j} = \sigma(o) = \frac{1}{1 + e^{-o}}$$

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

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

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

Back propagation



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

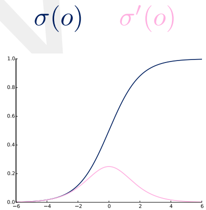
$$\hat{y}_j = x_{i,j} = \phi(o_{i,j}), \text{e.g. } \sigma(o_{i,j})$$

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

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

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

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



Back propagation

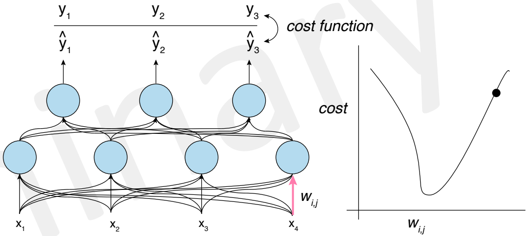
$$\Delta w_{i,j} = -\alpha \frac{\partial cost}{\partial w_{i,j}}$$

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

$$= \text{l.rate} \quad cost \quad activation \quad input$$

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

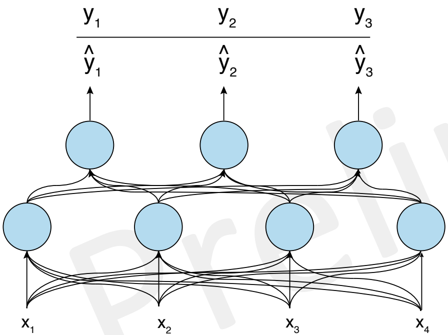
$$= -\alpha \; \delta \; x_{i-1,j}$$



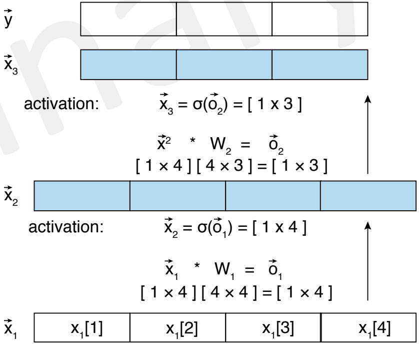
$$\delta_{output} = (y_j - x_{i,j}) \; x_{i,j}(1 - x_{i,j})$$ ← previous slide

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

Network representation



$\leftrightarrow$



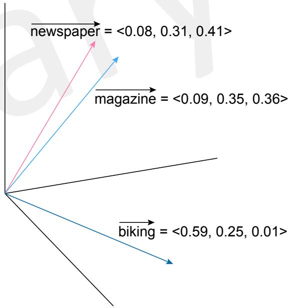
Outline

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:

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



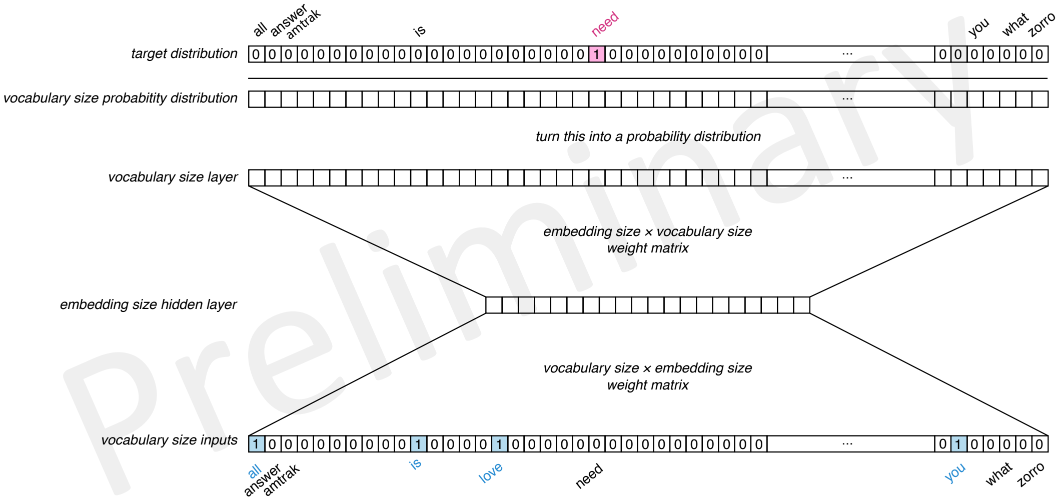
Distributed representations

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

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

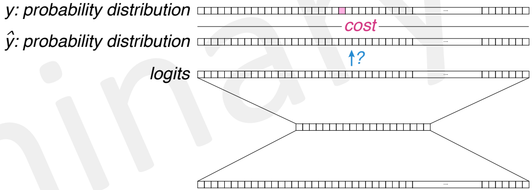
Embedding methods



Probability distributions

$$\mathrm{softmax} = \text{normalize the logits}$$
$$= \frac{e^{logits[i]}}{\sum_{j=1}^{|logits|} e^{logits[j]}}$$

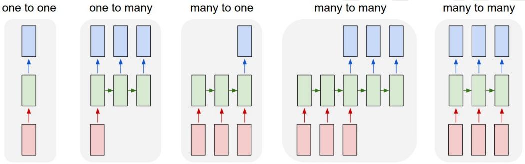
$$cost = \text{cross entropy loss}$$
$$= -\sum_x p(x) log\, \hat{p}(x)$$
$$= -\sum_i p_{\text{ground truth}}(word = vocabulary[i]) \log p_{\text{predictions}}(word = vocabulary[i])$$
$$= -\sum_i y_i \log \hat{y}_i$$



Outline

Recurrent neural networks

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

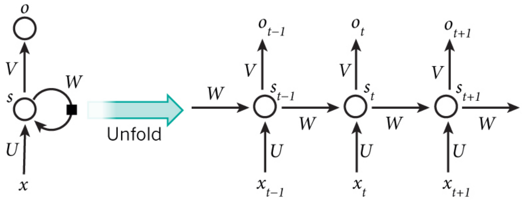


- Recurrent neural networks (RNNs) are called recurrent because they perform same task for every element of sequence, with output dependent on previous computations
- RNNs have memory that captures information about what has been computed so far
- RNNs can make use of information in arbitrarily long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

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

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 wanted to predict next word in sentence, a vector of probabilities across vocabulary: $o_t = \mathrm{softmax}(Vs_t)$

Image credits: Nature

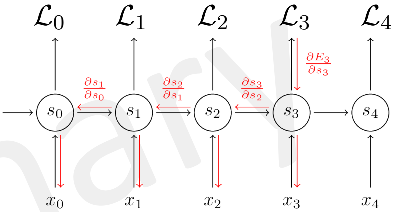
Language modeling using RNNs

- 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
- 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

Language modeling using RNNs

- Language model allows us to predict probability of observing sentence (in a given dataset) as: $P(w_1, \dots, w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, \dots, w_{i-1})$
- 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 (BPTT)

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

- How an LSTM calculates a 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 are called input, forget and output gates

-

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

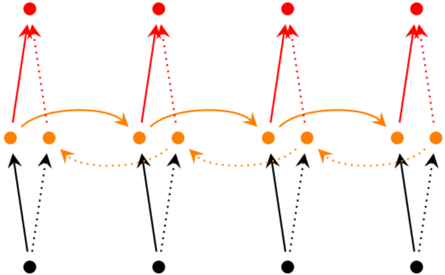


Image credits: http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

Outline

Morning program

Preliminaries

- Feedforward neural network
- Back propagation
- Distributed representations
- Recurrent neural networks
- Sequence-to-sequence models
- Convolutional neural networks

Text matching I

Text matching II

Afternoon program

Learning to rank

Modeling user behavior

Generating responses

Outlook

Wrap up

Sequence-to-sequence models

Important feature: we are now *generating* stuff.

- What does a sequence-to-sequence model do?
- It is basically two different RNNs
- Originally in translation setting, but more problems can be seen as seq2seq:
  - question answering
  - chatbots
  - IR...?
- Important technique to explain: attention mechanism