Sistemas Inteligentes

Word Embeddings II

José Eduardo Ochoa Luna
Dr. Ciencias - Universidade de São Paulo

Maestría C.C. Universidad Católica San Pablo
Sistemas Inteligentes

6 de Diciembre 2017

Recall

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m\\ j\neq 0}} \log p(w_{t+j}|w_t)$$

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V}\exp(u_w^T v_c)}$$

$$\frac{\partial}{\partial v_c}\log p(o|c) = \frac{\partial}{\partial v_c}\log\frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V}\exp(u_w^T v_c)}$$

$$\frac{\partial}{\partial v_c}\log p(o|c) = \frac{\partial}{\partial v_c}\log\exp(u_o^T v_c) - \log\sum_{w=1}^{V}\exp(u_w^T v_c)$$

Gradients

1)

$$\frac{\partial}{\partial v_c} \log \exp(u_o^T v_c) = \frac{\partial}{\partial v_c} u_o^T v_c = u_o$$

2)

$$\frac{\partial}{\partial v_c} \log \sum_{w=1}^{V} \exp(u_w^T v_c) = ?\ \text{(chain rule)}$$

Second term - Chain Rule

$$
\begin{aligned}
\frac{\partial}{\partial v_c} \log \sum_{w=1}^{V} \exp(u_w^T v_c) &= \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \frac{\partial}{\partial v_c} \sum_{x=1}^{V} \exp(u_x^T v_c) \\
&= \ldots [\sum_{x=1}^{V} \frac{\partial}{\partial v_c} \exp(u_x^T v_c)] \\
&= \ldots [\sum_{x=1}^{V} exp(u_x^T v_c) \frac{\partial}{\partial v_c} u_x^T v_c] \\
&= \ldots [\sum_{x=1}^{V} exp(u_x^T v_c) u_x] \\
&= \frac{1}{\sum_{w=1}^{V} \exp(u_w^T v_c)} [\sum_{x=1}^{V} exp(u_x^T v_c) u_x]
\end{aligned}
$$

Final Formula

$$\begin{aligned}
\frac{\partial}{\partial v_c} \log \textstyle\sum_{w=1}^{V} \exp(u_w^T v_c) &= \textstyle\sum_{x=1}^{V} \frac{\exp(u_x^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)} u_x \\
&= \textstyle\sum_{x=1}^{V} p(x|c) u_x
\end{aligned}$$

Final formula

$$\frac{\partial}{\partial v_c} \log p(o|c) = \underbrace{u_o}_{\text{Observed}} - \underbrace{\sum_{x=1}^{V} p(x|c) u_x}_{\text{Expectation}}$$

Gradient Descent

We will optimize (maximize or minimize) our objective / cost functions
Updates would be for each element of θ :

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta)$$

In matrix notation for all parameters:

$$\theta^{new} = \theta^{old} - \alpha \frac{\partial}{\partial \theta^{old}} J(\theta)$$

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

Stochastic Gradient Descent

- But corpus may have 40B tokens and windows
- You would wait a very long time before making a single update
- instead: we will update parameters after each window t: stochastic gradient descent (SGD)

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J_t(\theta)$$

Classification Setup

- We have a training dataset consisting of N samples

$$\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$$

- $x^{(i)}$ inputs, e.g. words (indices or vectors), context windows, sentences, documents, etc.
- $y^{(i)}$, labels we try to predict
  - class: sentiment, named entities

Classification Setup

- We have a training dataset consisting of N samples

$$\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$$

- $x^{(i)}$ inputs, e.g. words (indices or vectors), context windows, sentences, documents, etc.
- $y^{(i)}$, labels we try to predict
  - class: sentiment, named entities
  - other words

Classification Setup

- We have a training dataset consisting of N samples

$$\{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$$

- $x^{(i)}$ inputs, e.g. words (indices or vectors), context windows, sentences, documents, etc.
- $y^{(i)}$, labels we try to predict
  - class: sentiment, named entities
  - other words
  - multi-word sequences (Machine Translation)

General Setup

- General ML: assume x is fixed, train logistic regression weights w, $\rightarrow$ only modify the decision boundary

General Setup

- General ML: assume x is fixed, train logistic regression weights w, $\rightarrow$ only modify the decision boundary
- Goal: predict for each x: $p(y|x) = \frac{\exp(W_y x)}{\sum_{c=1}^{C} \exp(W_c x)}$

Window classification

- Classifying single words is rarely done

Window classification

- Classifying single words is rarely done
- Interesting problems arise in context

Window classification

- Classifying single words is rarely done
- Interesting problems arise in context
- Example auto-antonymous, “to sanction” can mean “to permit” or “to punish”

Window classification

- Classifying single words is rarely done
- Interesting problems arise in context
- Example auto-antonymous, "to sanction" can mean "to permit" or "to punish"
- Example ambiguous named entities: Paris$\rightarrow$ Paris, France vs Paris Hilton

Window classification

- idea: classify a word in its context window of neighboring words

Window classification

- idea: classify a word in its context window of neighboring words
- For example named entity recognition into 4 classes: person, location, organization, none

Window classification

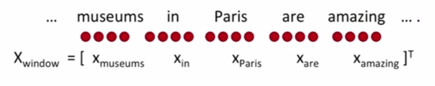
- idea: classify a word in its context window of neighboring words
- For example named entity recognition into 4 classes: person, location, organization, none
- Many possibilities exist for classifying one word in context, e.g. averaging all the words in a window but that looses position information

Window classification

- Train Softmax classifier by assigning a label to a center word and containing all word vectors surrounding it

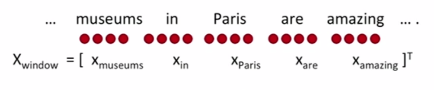
Window classification

- Train Softmax classifier by assigning a label to a center word and containing all word vectors surrounding it
- Example. classify Paris in the context of this sentence with length 2:



Window classification

- Train Softmax classifier by assigning a label to a center word and containing all word vectors surrounding it
- Example. classify Paris in the context of this sentence with length 2:



- Resulting vector $x_{window} = x \in \mathbb{R}^{5d}$, a column vector

Simplest Window Classifier: Softmax

- With $x = x_{window}$

$$\hat{y}_y = p(y|x) = \frac{\exp(W_y x)}{\sum_{c=1}^{C} \exp(W_c x)}$$

Simplest Window Classifier: Softmax

- With $x = x_{window}$

$$\hat{y}_y = p(y|x) = \frac{\exp(W_y x)}{\sum_{c=1}^{C} \exp(W_c x)}$$

- With cross entropy error:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} -\log\left(\frac{e^{f_{y_i}}}{\sum_{c=1}^{C} e^{f_c}}\right)$$

Neural Net Window Classifier

- $x_{window} = [x_{museums} x_{in} x_{Paris} x_{are} x_{amazing}]$

Neural Net Window Classifier

- $x_{window} = [x_{museums} x_{in} x_{Paris} x_{are} x_{amazing}]$
- Assume we want to classify whether the center word is a location or not

A Single Hidden Layer Neural Network

- Hidden layer is a combination of a linear layer and a nonlinearity

$$\begin{aligned} z &= Wx + b \\ a &= f(z) \end{aligned}$$

A Single Hidden Layer Neural Network

- Hidden layer is a combination of a linear layer and a nonlinearity

$$\begin{aligned} z &= Wx + b \\ a &= f(z) \end{aligned}$$

- The neural activations a can be used to compute some output

A Single Hidden Layer Neural Network

- Hidden layer is a combination of a linear layer and a nonlinearity

$$
\begin{aligned}
z &= Wx + b \\
a &= f(z)
\end{aligned}
$$

- The neural activations *a* can be used to compute some output
- For instance, a probability via softmax

$$p(y|x) = softmax(Wa)$$

A Single Hidden Layer Neural Network

- Hidden layer is a combination of a linear layer and a nonlinearity

$$
\begin{aligned}
z &= Wx + b \\
a &= f(z)
\end{aligned}
$$

- The neural activations a can be used to compute some output
- For instance, a probability via softmax

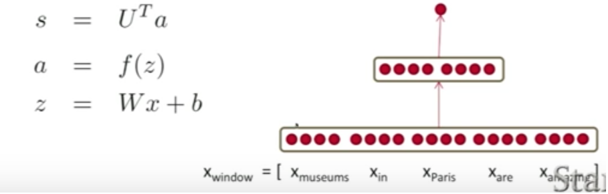
$$p(y|x) = softmax(Wa)$$

- Or an unnormalized score (simpler)

$$score(x) = U^T a \in \mathbb{R}$$

Feedforward computation

Computing a window's score with a 3-layer neural net: $s = score$ (museums in Paris are amazing)



$$s = U^T f(Wx + b),\ x \in \mathbb{R}^{20\times 1},\ W \in \mathbb{R}^{8\times 20},\ U \in \mathbb{R}^{8\times 1}$$

The max-margin loss

- $s = score$ (museums in Paris are amazing)
- $s_c = score$ (Not all museums in Paris)
- Idea for training objective: make score of true window larger and corrupt window's score lower (until they are good enough): minimize

$$J = max(0, 1 - s + s_c)$$

- This is continuous, we can use SGD

Training with Backpropagation

- $$J = max(0, 1 - s + s_c)$$

Training with Backpropagation

- $$J = max(0, 1 - s + s_c)$$
- $s = U^T f(Wx + b)$, $s_c = U^T f(Wx_c + b)$

Training with Backpropagation

- $$J = max(0, 1 - s + s_c)$$
- $s = U^T f(Wx + b)$, $s_c = U^T f(Wx_c + b)$
- Assuming cost J is > 0

Training with Backpropagation

- $$J = max(0, 1 - s + s_c)$$
- $s = U^T f(Wx + b)$, $s_c = U^T f(Wx_c + b)$
- Assuming cost J is > 0
- compute the derivatives of s and s_c wrt the involved variables: U, W, b, x

Training with Backpropagation

- $$J = max(0, 1 - s + s_c)$$
- $s = U^T f(Wx + b)$, $s_c = U^T f(Wx_c + b)$
- Assuming cost J is > 0
- compute the derivatives of s and s_c wrt the involved variables: U, W, b, x
- $\frac{\partial s}{\partial U} = \frac{\partial}{\partial U} U^T a = a$

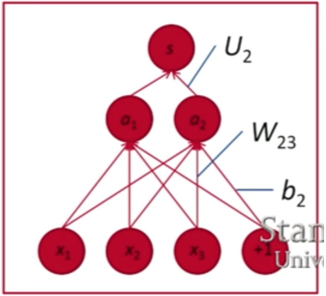
Training

Let's consider the derivative of a single weight W_{ij}

$$\frac{\partial s}{\partial W} = \frac{\partial}{\partial W} U^T a = \frac{\partial s}{\partial W} U^T f(z) = \frac{\partial s}{\partial W} U^T f(Wx + b)$$

This only appears inside a_i

For example: W_{23} is only used to compute a_2



Backpropagation

Derivative of weight W_{ij}

$$\frac{\partial}{\partial W_{ij}} U^T a \rightarrow \frac{\partial}{\partial W_{ij}} U_i a_i$$

$$\begin{aligned}
U_i \frac{\partial}{\partial W_{ij}} a_i &= U_i \frac{\partial a_i}{\partial z_i} \frac{\partial z_i}{\partial W_{ij}} \\
&= U_i \frac{\partial f(z_i)}{\partial z_i} \frac{\partial z_i}{\partial W_{ij}} \\
&= U_i f'(z_i) \frac{\partial z_i}{\partial W_{ij}} \\
&= U_i f'(z_i) \frac{\partial W_i x + b_i}{\partial W_{ij}}
\end{aligned}$$

Backpropagation

Derivative of weight W_{ij}

$$
\begin{aligned}
U_i \frac{\partial}{\partial W_{ij}} a_i &= U_i f'(z_i) \frac{\partial W_i x + b_i}{\partial W_{ij}} \\
&= U_i f'(z_i) \frac{\partial}{\partial W_{ij}} \sum_k W_{ik} x_k \\
&= \underbrace{U_i f'(z_i)} x_j \\
&= \delta_i x_j \\
&= \text{local signal error - local input signal}
\end{aligned}
$$

$f'(z) = f(z)(1 - f(z))$ for logistic f

Backpropagation

From single weight W_{ij} to full W:

$$\begin{aligned} \frac{\partial s}{\partial W_{ij}} &= \underbrace{U_i f'(z_i)} x_j \\ &= \delta_i x_j \end{aligned}$$

solution: outer product

$$\frac{\partial s}{\partial W} = \delta x^T$$

Backpropagation

For biases b, we get:

$$\begin{aligned} & U_i \frac{\partial}{\partial b_i} a_i \\ = \;& \underbrace{U_i f'(z_i)}\frac{\partial W_i x + b_i}{\partial b_i} \\ = \;& \delta_i \end{aligned}$$

Backpropagation

- That's almost backpropagation:
  it's taking derivatives and using the chain rule

Backpropagation

- That's almost backpropagation:
  it's taking derivatives and using the chain rule
- We can re-use derivatives computed for higher layers in computing derivatives for lower layers

Backpropagation

- That's almost backpropagation:
  it's taking derivatives and using the chain rule
- We can re-use derivatives computed for higher layers in computing derivatives for lower layers
- last derivatives of model, the word vectors in x

Backpropagation

- Take derivative of score with respect to single element of word vector

Backpropagation

- Take derivative of score with respect to single element of word vector
- Now, we cannot just take into consideration one a_i because each x_j is connected to all neurons above and hence x_j influences the overall score trough all these:

Backpropagation

$$
\begin{aligned}
\frac{\partial s}{\partial x_j} &= \sum_{i=1}^{2} \frac{\partial s}{\partial a_i}\frac{\partial a_i}{\partial x_j} \\
&= \sum_{i=1}^{2} \frac{\partial U^T a}{\partial a_i}\frac{\partial a_i}{\partial x_j} \\
&= \sum_{i=1}^{2} U_i \frac{\partial f(W_i x + b)}{\partial x_j} \\
&= \sum_{i=1}^{2} \underbrace{U_i f'(W_i x + b)} \frac{\partial W_i x}{\partial x_j} \\
&= \sum_{i=1}^{2} \delta_i W_{ij} \\
&= W_j^T \delta
\end{aligned}
$$

Backpropagation

With $\frac{\partial s}{\partial x_j} = W_j^T \delta$, what is the full gradient?

$$\frac{\partial s}{\partial x} = W^T \delta$$