Language Models

Joost Bastings

http://joost.ninja

you know nothing, jon ___

life is like a box of ___

the ___

the quick ___

the quick brown ___

the quick brown fox ___

the quick brown fox jumps ___

the quick brown fox jumps over ___

the quick brown fox jumps over the ___

the quick brown fox jumps over the lazy ___ the quick brown fox jumps over the lazy dog

Predicting the future

Language models give us the probability of a **sentence**.

At any time step, they assign a **probability** to the next word.

Applications

This is useful when we deal with **noisy input**:

speech recognition

handwriting recognition

spelling correction

machine translation

Source: J&M

today:
n-gram based lm
feature based lm
neural network based lm

Probability of a sentence

p(you know nothing jon snow) = **???**

We can use the **chain rule**:

$p(B \mid A) = p(A, B) / p(A)$

p(you know nothing jon snow) =

p(know | you) p(nothing | you know) p(jon | you know nothing) p(snow | you know nothing jon)

n-gram language models

To make (count based) estimation possible, we need a **markov assumption**

bi-gram case:

$$p(x_i \mid x_1\ x_2 \dots x_{i-1}) \approx p(x_i \mid x_{i-1})$$

parameters: $\theta_{x|x'} \quad \forall\ x, x' \in \mathcal{V}$

n-gram case:

$$p(x_i \mid x_1\ x_2 \dots x_{i-1}) \approx p(x_i \mid x_{i-n+1} \dots x_{i-1})$$

parameters: $\theta_{x|\mathbf{h}} \quad \forall\ x \in \mathcal{V}, \mathbf{h} \in \mathcal{V}^{n-1}$

Smoothing

If we didn't observe a certain bigram, then $p(x_i \mid x_{i-1})$ will be 0

This makes the probability of the sentence also 0!

MLE: $p_{MLE}(x_i \mid x_{i-1}) = count(x_{i-1}, x_i) \,/\, count(x_{i-1})$

Laplace / add-one smoothing: $p_{Add1}(x_i \mid x_{i-1}) = count(x_{i-1}, x_i) + 1$

$/ \; count(x_{i-1}) + V$

This doesn't work so well for language modeling.

For more advanced smoothing, cf. Kneser-Ney, Stupid Backoff.

Source: J&M

Perplexity

We would like our models to assign high probability to “real” sentences

Let $\mathbf{x} = x_1\ x_2 \ldots x_N$

$$pp(\mathbf{x}) = \exp -{}^{1}\!/_{N} \textstyle\sum_i \ln P(x_i)$$

$$pp(\mathbf{x}) = P(x_1\ x_2 \ldots x_N)^{-1/N}$$

Log-linear models

$$p_{\mathbf{w}}(Y=y \mid X=x) = \frac{\exp \mathbf{w} \cdot \phi(x, y)}{\sum_{y' \in \mathcal{Y}} \exp \mathbf{w} \cdot \phi(x, y')}$$

Y is the **next word** (and $\mathcal{Y}$ the vocabulary)

X is the **history**

ϕ is a **feature function** which returns a d-dimensional vector

w are the model parameters

Why use log-linear models with features?

With *features* of words and histories we can share **statistical weight**

Mind: with n-grams, there is *no sharing at all!*

We also get **smoothing**

We can add arbitrary features, e.g. we could condition the text on properties of the author (gender, age, political affiliation,..)

We can easily **interpret** these kinds of models

Each feature ϕ_k controls a *factor* to the probability (e^{wk})

if $w_k < 0$, then ϕ_k makes the event *less likely* by a factor of $1/e^{wk}$

if $w_k > 0$, then ϕ_k makes the event *more likely* by a factor of e^{wk}

if $w_k = 0$, then ϕ_k has no effect

Source: Noah Smith / csep 517

What features should we use?

n-gram features: “X_{j-1} = the ^ X_j = puppy”

gappy n-gram features: “X_{j-2} = the ^ X_j = puppy”

spelling features: “X_j’s first character is Capitalized”

class features: “X_j is a member of class 123”

gazetteer features: “X_j is a geographic place name”

Source: Noah Smith / csep 517

Comparison: n-gram vs. log-linear

n-gram

$p_\theta(\mathbf{x}) = \prod_i \theta_{x_i \mid \mathbf{h}_i}$

parameters:

$\theta_{x|\mathbf{h}} \quad \forall x \in \mathcal{V}, \mathbf{h} \in \mathcal{V}^{n-1}$

MLE:

$\hat{\theta}_{x|\mathbf{h}} = \text{count}(\mathbf{h}, x) / \text{count}(\mathbf{h})$

log-linear

$p_\theta(\mathbf{x}) = \prod_i \exp \mathbf{w} \cdot \phi(\mathbf{h}_i, x_i) / Z_\mathbf{w}(h_i)$

parameters:

$w_k \quad \forall k \in \{1, \ldots, d\}$

MLE:

No closed form! → use SGD

training data $\{ (x_i, h_i) \}_{i=1}^N$

MLE: $\max_\mathbf{w} \sum_i \mathbf{w} \cdot \phi(\mathbf{h}_i, x_i) - \log Z_\mathbf{w}(h_i)$

$\nabla_\mathbf{w} f_i = \phi(\mathbf{h}_i, x_i) \quad - \quad \sum_v p(v \mid \mathbf{h}_i) \cdot \phi(\mathbf{h}_i, v)$

Observed features

Expected features

Source: Noah Smith / csep 517

Stochastic Gradient Descent

Goal: minimize $\sum_i^N f_i(\mathbf{w})$ with respect to **w**

Input: initial values **w**, learning rate α

SGD:

while "not converged":

sample training instance i from {1, ... , N}

$w \leftarrow w - \alpha \cdot \nabla_w f_i$

Source: Noah Smith / csep 517

Regularization

Problem:

if a certain feature is often positive, we can increase the objective by making its weight slightly larger every time

This can cause **overfitting**!

Solution:

Regularization

We add a term $\lambda \, ||\mathbf{w}||^2$ to our objective (“L2-norm”) to discourage large weights

Source: Noah Smith / csep 517

Summary so far

n-gram

$\mathbf{h}_i$ is n-1 previous words

estimation: count and normalize

think about smoothing

log-linear

featurized representation of ($\mathbf{h}_i$, x_i)

estimation with SGD

think about features

Neural language models

Motivation

n-gram language models have proven to be effective for various tasks 👍

*maybe we do not need the **full** history*

log-linear models allow sharing statistical weight through features 👍

sharing is important for generalization

maybe our history is still too limited (n-1 words) 👎

sometimes larger histories are useful, but hard to estimate parameters

we need to find useful features 👎

and be careful of overfitting with too many features

Motivation (2)

Consider:

1. The cat is walking in the bedroom
2. A dog was running in a room

Observing (1) should help us generalize to make (2) almost as likely.

But with the previous methods this is hard to accomplish!

Bengio et al. (2003)

Feed-forward NN approach

with neural networks we can exploit **distributed representations** to allow statistical weight sharing

how it works:

1. each word is assigned a distributed m-dimensional **feature vector**
2. a **probability function** of word sequences is expressed in terms of those vectors
3. we **jointly** learn the feature vectors and the parameters of that probability function

Bengio et al. (2003)

Why would this work?

Similar words are expected to have a similar feature vector

(dog, cat), (is, was), (running, walking), (room, bedroom), …

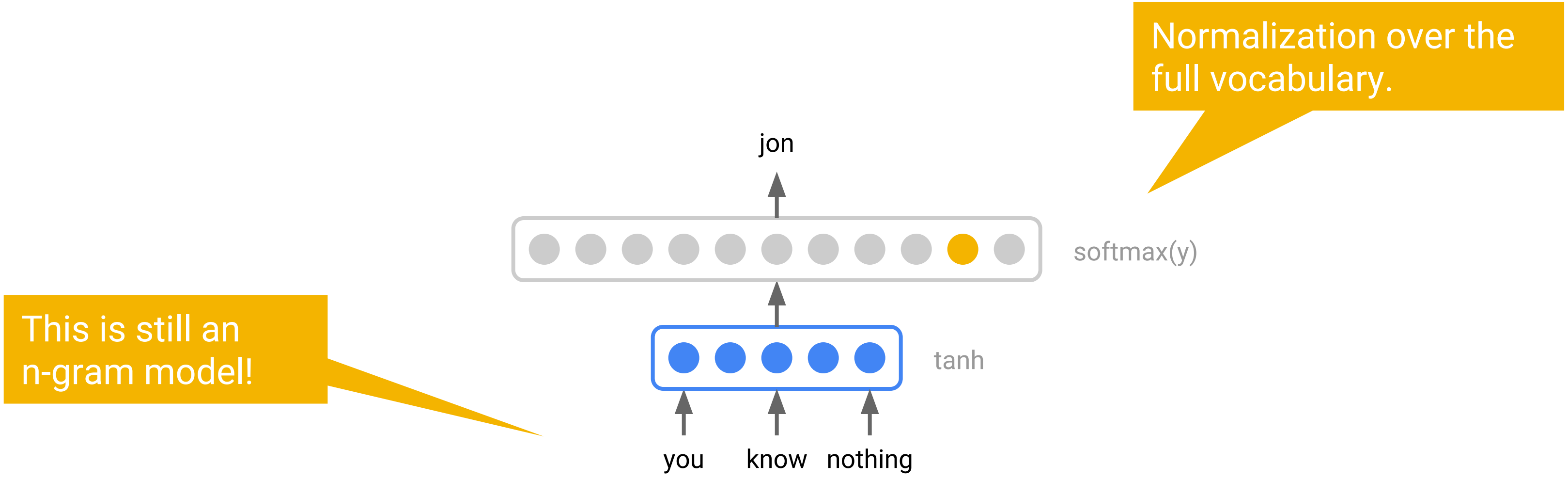
With this, probability mass is naturally transferred from (1) to (2):

1. *The cat is walking in the bedroom*
2. *A dog was running in a room*

and to many, many other similar sentences.

The point: the presence of only 1 sentence in the training data will increase the probability for a *combinatorial number* of “neighbors” in sentence space.

Feed-forward NN language model



$$y = W'' \tanh(Wx + b) + W'x + b'$$

Bengio et al. (2003)

Why does it work?

the nonlinear NN function allows **feature combinations** a linear model cannot get

end-to-end training on next word prediction

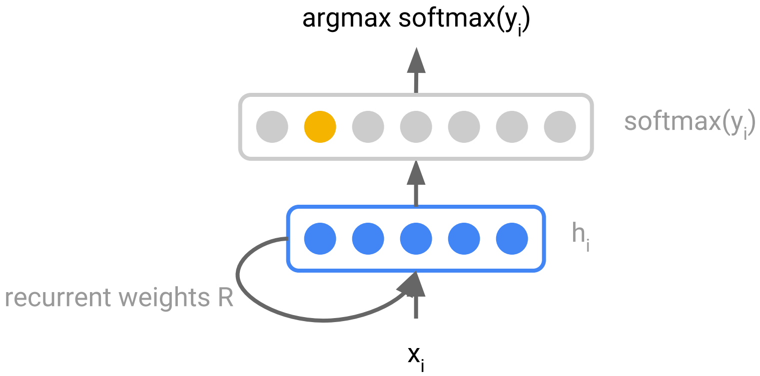
Recurrent NN language model

we now have much better generalization, but still a limited context

recurrent neural networks (RNNs) have an **unlimited context**

Mikolov et al. (2010)

Recurrent NN language model



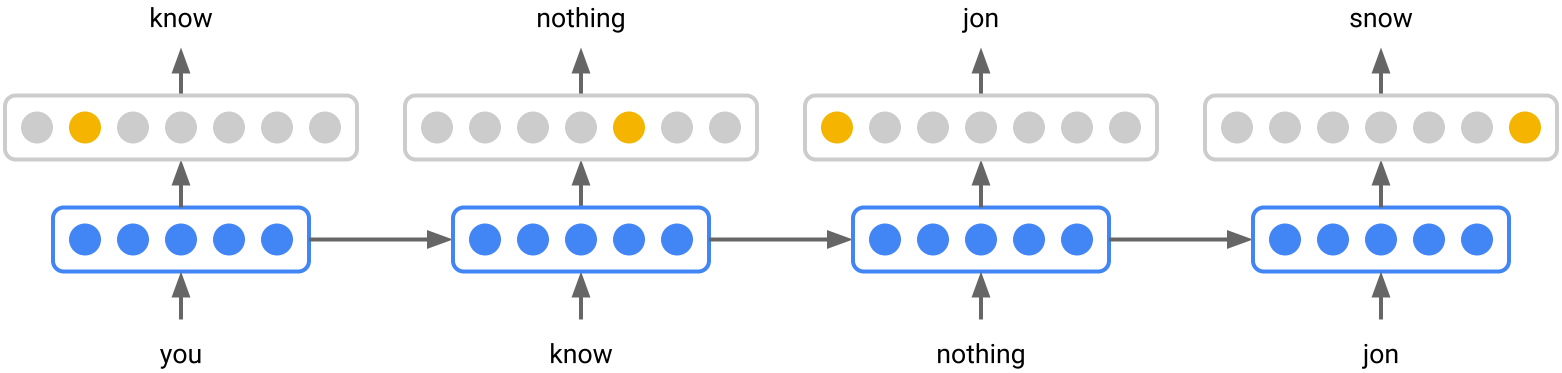
$$h_i = \sigma(\ Wx + Rh_{i-1} + b\ )$$

$$y_i = W'' h_i + b'$$

Mikolov et al. (2010)

Recurrent NN language model

Every prediction here causes a change in the weights



Mikolov et al. (2010)

Final note on neural language models

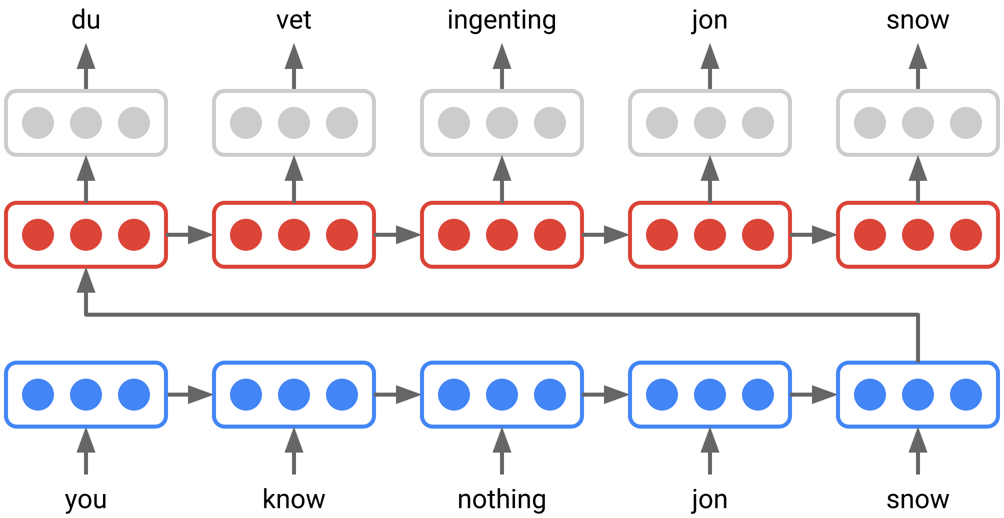
1. the log-likelihood function is **not concave**
2. so when we evaluate a new neural language **model**,

   we also evaluate our **algorithm** to estimate the parameters

   (and its **hyperparameters**)
3. RNNs suffer from the **vanishing gradient problem** (next lecture)
4. many improved language models have been proposed since these ones

   e.g. log-bilinear lm, LSTM-based, character based

Preview: Encoder-decoder



Sutskever et al. (2014), Cho et al. (2014)

References

Bengio, Yoshua, et al. "A neural probabilistic language model." Journal of machine learning research 3.Feb (2003): 1137-1155.

Mikolov, Tomas, et al. "Recurrent neural network based language model." Interspeech. Vol. 2. 2010.

Cho, Kyunghyun, et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." arXiv preprint arXiv:1406.1078 (2014).

Noah Smith, CSEP517: http://courses.cs.washington.edu/courses/csep517/17sp/

Jurafsky & Martin. Speech and Language Processing (3rd ed. draft). https://web.stanford.edu/~jurafsky/slp3/