# Language Modeling

# +

# Feed-Forward Networks 3

Alexander Rush

# Review: Machine Learning Setup

Multi-class prediction problem,

$$(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)$$

- $\mathbf{y}_i$; the one-hot next word
- $\mathbf{x}_i$; representation of the prefix $(w_1, \ldots, w_{t-1})$

**Challenges:**

- How do you represent input?
- Smoothing is crucially important.
- Output space is very large (next class)

# Review: Machine Learning Setup

Multi-class prediction problem,

$$(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)$$

- $\mathbf{y}_i$; the one-hot next word
- $\mathbf{x}_i$; representation of the prefix $(w_1, \ldots, w_{t-1})$

**Challenges:**

- How do you represent input?
- Smoothing is crucially important.
- Output space is very large (next class)

# Review: Perplexity

Previously, used *accuracy* as a metric.

Language modeling uses of version average negative log-likelihood

- For test data $\bar{w}_1, \ldots, \bar{w}_n$
- $$NLL = -\frac{1}{n} \sum_{i=1}^{n} \log p(w_i | w_1, \ldots, w_{i-1})$$

Actually report *perplexity*,

$$perp = \exp(-\frac{1}{n} \sum_{i=1}^{n} \log p(w_i | w_1, \ldots, w_{i-1}))$$

Requires modeling full distribution as opposed to argmax (hinge-loss)

# Idea 1: Interpolation (Jelinek-Mercer Smoothing)

Can write recursively,

$$p_{interp}(w|c) = \lambda p_{ML}(w|c) + (1-\lambda) p_{interp}(w|c')$$

Ensure that $\lambda$ form convex combination

$$0 \le \lambda \le 1$$

How do you learn conjunction combinations?

# Quiz

We talked briefly last class about using language models for smoothing. It has become a popular task in recent years to utilize language models to predict missing words, for example consider the Microsoft Research Sentence Completion Challenge.

a tractor rode slow

a red tractor rode fast

the parrot flew fast

the parrot flew slow

the tractor slowed down

the red ___ ?

the red ___ flew fast?

# Today's Class

$$p(w_i | w_{i-n+1}, \ldots w_{i-1}; \theta)$$

- Estimate this directly as a neural network.
- Two types of models, neural network and bilinear.
- Efficiency methods for estimation.

# Intuition: NGram Issues

In training we see,

the arizona corporations commission authorized

But at test we see,

the colorado businesses organization ___

- Does this training example help here?
  - Not really. No count overlap.
- Does backoff help here?
  - Maybe, if we have seen organization.
  - Mostly get nothing from the earlier words.

# Intuition: NGram Issues

In training we see,

the arizona corporations commission authorized

But at test we see,

the colorado businesses organization ___

- Does this training example help here?
  - Not really. No count overlap.
- Does backoff help here?
  - Maybe, if we have seen organization.
  - Mostly get nothing from the earlier words.

# Class-Based Language Models

# Contents

# Recall: Word Embeddings

- Embeddings allow us to utilize similar words

| | | |
|---|---|---|
| arizona | texas | 0.932968706025 |
| | florida | 0.932696958878 |
| | kansas | 0.914805968271 |
| | colorado | 0.904197441085 |
| | minnesota | 0.863925347525 |
| | carolina | 0.862697751337 |
| | utah | 0.861915722889 |
| | miami | 0.842350326527 |
| | oregon | 0.842065064748 |

| | |
|---|---|
| firms | 0.894882639809 |
| companies | 0.86738377358 |
| businesses | 0.859315950927 |
| corporate | 0.821590295322 |

# Issues

Although... in training we see,

the eagles play the arizona diamondbacks

And at test we might see,

the eagles play the colorado

# Feed-Forward Neural NNLM (Bengio, 2003)

- $\mathbf{x}$ is an embedded representation

# Feed-Forward Neural Representation

- $p(w_i | w_{i-n+1}, \ldots w_{i-1}; \boldsymbol{\theta})$
- $f_1, \ldots, f_{d_{\mathrm{win}}}$ are words in window
- Input representation is the concatenation of embeddings

$$\mathbf{x} = [v(f_1)\ v(f_2)\ \ldots\ v(f_{d_{\mathrm{win}}})]$$

Example: Tagging

$$[w_3\ w_4\ w_5\ w_6\ w_7]\ w_8$$

$$\mathbf{x} = [v(w_3)\ v(w_4)\ v(w_5)\ v(w_6)\ v(w_7)]$$

| | | | | |
|---|---|---|---|---|
| | | $\mathbf{x}$ | | |
| $d_{\mathrm{in}}/5$ | $d_{\mathrm{in}}/5$ | $d_{\mathrm{in}}/5$ | $d_{\mathrm{in}}/5$ | $d_{\mathrm{in}}/5$ |

# A Neural Probabilistic Language Model (Bengio, 2003)

One hidden layer multi-layer perceptron architecture,

$$NN_{MLP1}(\mathbf{x}) = \tanh(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1)W^2 + \mathbf{b}^2$$

Neural network architecture on top of concat.

$$\hat{\mathbf{y}} = \text{softmax}(NN_{MLP1}(\mathbf{x}))$$

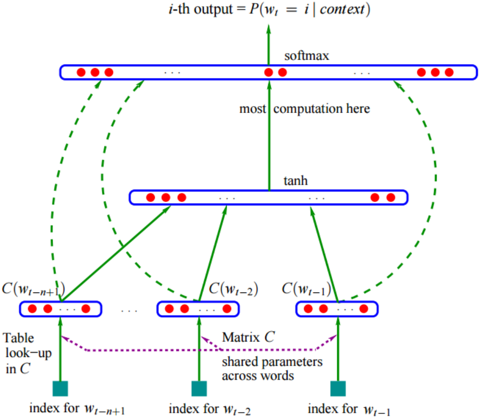
# A Neural Probabilistic Language Model

Optional, direct connection layers,

$$NN_{DMLP1}(\mathbf{x}) = [\tanh(\mathbf{x}\mathbf{W}^1 + \mathbf{b}^1), \mathbf{x}]W^2 + \mathbf{b}^2$$

- $\mathbf{W}^1 \in \mathbb{R}^{d_{\text{in}} \times d_{\text{hid}}}, \mathbf{b}^1 \in \mathbb{R}^{1 \times d_{\text{hid}}}$; first affine transformation
- $\mathbf{W}^2 \in \mathbb{R}^{(d_{\text{hid}} + d_{\text{in}}) \times d_{\text{out}}}, \mathbf{b}^2 \in \mathbb{R}^{1 \times d_{\text{out}}}$; second affine transformation

# A Neural Probabilistic Language Model (Bengio, 2003)



# A Neural Probabilistic Language Model (Bengio, 2003)

| | n | c | h | m | direct | mix | train. | valid. | test. |
|---|---|---|---|---|---|---|---|---|---|
| MLP1 | 5 | | 50 | 60 | yes | no | 182 | 284 | 268 |
| MLP2 | 5 | | 50 | 60 | yes | yes | | 275 | 257 |
| MLP3 | 5 | | 0 | 60 | yes | no | 201 | 327 | 310 |
| MLP4 | 5 | | 0 | 60 | yes | yes | | 286 | 272 |
| MLP5 | 5 | | 50 | 30 | yes | no | 209 | 296 | 279 |
| MLP6 | 5 | | 50 | 30 | yes | yes | | 273 | 259 |
| MLP7 | 3 | | 50 | 30 | yes | no | 210 | 309 | 293 |
| MLP8 | 3 | | 50 | 30 | yes | yes | | 284 | 270 |
| MLP9 | 5 | | 100 | 30 | no | no | 175 | 280 | 276 |
| MLP10 | 5 | | 100 | 30 | no | yes | | 265 | **252** |
| Del. Int. | 3 | | | | | | 31 | 352 | 336 |
| Kneser-Ney back-off | 3 | | | | | | | 334 | 323 |
| Kneser-Ney back-off | 4 | | | | | | | 332 | 321 |
| Kneser-Ney back-off | 5 | | | | | | | 332 | 321 |
| class-based back-off | 3 | 150 | | | | | | 348 | 334 |
| class-based back-off | 3 | 200 | | | | | | 354 | 340 |
| class-based back-off | 3 | 500 | | | | | | 326 | **312** |
| class-based back-off | 3 | 1000 | | | | | | 335 | 319 |
| class-based back-off | 3 | 2000 | | | | | | 343 | 326 |
| class-based back-off | 4 | 500 | | | | | | 327 | 312 |
| class-based back-off | 5 | 500 | | | | | | 327 | 312 |

# Parameters

- Bengio NNLM has $d_{\mathrm{hid}} = 100$, $d_{\mathrm{win}} = 5$, $d_{\mathrm{in}} = 5 \times 50$
- In-Class: How many parameters does it have? How does this compare to Kneser-Ney smoothing?

# Log-Bilinear Language Model ()

$$\hat{\mathbf{y}} = \mathrm{softmax}((\mathbf{x})\mathbf{W}^1 + \mathbf{b})$$

- Remove the tanh layer, but maintain ordering.
- Dense $\mathbf{x}$ concatenated word-embeddings

# Neural Language Modeling

NGram Models

- Fast to train

Neural Models

- Make Markov assumption
- Slower to train
- 

# NGram Models

# Contents

# Review: Softmax Issues

Use a softmax to force a distribution,

$$\text{softmax}(\mathbf{z}) = \frac{\exp(\mathbf{z})}{\sum_{c \in \mathcal{C}} \exp(z_c)}$$

$$\log \text{softmax}(\mathbf{z}) = \mathbf{z} - \log \sum_{c \in \mathcal{C}} \exp(z_c)$$

- **Issue:** class $\mathcal{C}$ is huge.
- For C&W, 100,000, for word2vec 1,000,000 types
- Note largest dataset is 6 billion words

# Unnormalized Network

Recall the score defined as,

$$\mathbf{z} = \tanh(\mathbf{x}\mathbf{W}^1)\mathbf{W}^2 + \mathbf{b}$$

Unnormalized score of each word.

$$z_i = \tanh(\mathbf{x}\mathbf{W}^1)\mathbf{W}^2_{*,i} + b_i$$

Can be computed efficiently $O(1)$ versus $O(|\mathcal{V}|)$.

# Coherence Estimation

- **Idea:** Learn to distinguish coherent n-grams from corruption.
- Similar idea to ranking embedding, but no score function.
- Want to differentiate,

  [ the dog walks ]

  It should score higher than,

  [ the dog house ]

  [ the dog cats ]

  [ the dog skips ]

Imagine we had this dataset

$$(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n),$$

Where $\mathbf{y}$ was

$$\mathcal{L}(\theta) = \sum_i L_{cross}(\mathbf{y}, \hat{\mathbf{y}})$$

# Recall: Binary Classification

$$\mathcal{L}(\theta) = \sum L_{cross}()$$

# NCE 1

Random variable $D$

$D = 1$ with prob $\frac{1}{K+1}$

$D = 0$ with prob $\frac{K}{K+1}$

If $D = 1$ then generate a true word, otherwise if $D = 0$ generate from noise distribution.

$$P(D=1|\mathbf{x},\mathbf{y}) = \frac{P(X|D,Y)P(X|Y)}{\sum_d P(X|D)P(X)} = \frac{P(X|D)P(X)}{P(X|D)P(x)+P(X|D)P(x)}$$

$$\frac{1/k+1p(X)}{1/k+1p(X)+(k)/(k+1)P(n_i)} = \frac{p(X)}{p(X)+(k)P(n_i)}$$

$$\sigma(z_w - \log(kP(w)))$$

# NCE 2

$$\mathcal{L}(\theta) = \sum_i \log P(D = 1 | X_i, Y) + \sum_{k=1}^{K} P(D = 0 | X_i, =)$$

$$\mathcal{L}(\theta) = \sum_i \log \sigma(z_w - \log(kP(w))) + \sum_{k=1}^{K} (1 - \sigma(z_w - \log(kP(w))))$$

# Implementation

How do you efficiently compute $z_w$?

Need a lookup table for output embeddings! (not linear) and dot product.

How do you efficiently handle $p(w)$

Also can be done with lookuptable and add.

How do you handle sampling?

Can precompute large number of samples (not example specific).

How do you handle loss?

Simply BinaryNLL Objective.

# Implementation

Standard MLP language model,

$$\mathbf{x} \Rightarrow \mathbf{W}^1 \Rightarrow \tanh \Rightarrow \mathbf{W}^2 \Rightarrow \text{softmax}$$

Computing $\sigma(z_w - \log(kP(w)))$,

$$\mathbf{x} \Rightarrow \mathbf{W}^1 \Rightarrow \tanh \Rightarrow \underset{\mathbf{W}^2_{*,w}(\text{Lookup})}{\cdot} \Rightarrow \underset{Kp(w)(\text{Lookup})}{-} \Rightarrow \sigma$$

(Efficiency, compute first three layers only once for $K+1$)

# Noise Distribution

What noise distribution should you use.

Unigram estimations.

# Use as an LM

- Unlike HSM learns full $\mathbf{W}^2$
- Can run softmax$(tanh(\mathbf{x}\mathbf{W}^1) + \mathbf{W}^2)$ at test time.
- Instead of 1 multiclass, we do 1+K binary classifications