# Language Modeling

# +

# Feed-Forward Networks 3

CS 287

# Review: LM ML 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)

# Review: 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

$$0 \leq \lambda \leq 1$$

How do you learn conjunction combinations?

# Quiz

Assume we have seen the following training sentences,

- a tractor drove slow
- the red tractor drove fast
- the parrot flew fast
- the parrot flew slow
- the tractor slowed down

Compute $p_{ML}$ for bigrams and use them to estimate whether *parrot* or *tractor* fit better in the following contexts.

1. the red ___ ?
2. the ___ ?
3. the ___ drove?

# Answer I

| | | |
|---|---|---|
| a | tractor | 1 |
| a | red | $\frac{1}{2}$ |
| the | red | $\frac{1}{4}$ |
| the | parrot | $\frac{1}{2}$ |
| the | tractor | $\frac{1}{4}$ |
| red | tractor | 1 |
| tractor | drove | $\frac{2}{3}$ |
| tractor | slowed | $\frac{1}{3}$ |
| parrot | flew | 1 |
| ... | | |

# Answer II

- the red tractor
- the parrot
- the tractor drove

# 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 log-bilinear.
- Efficient methods for approximated estimation.

# Intuition: NGram Issues

In training we might 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 might 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 might 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.

# Goal

- Learn representations that share properties between words.
- Particularly helpful for unseen contexts.
- Not a silver bullet, e.g. proper nouns

  the eagles play the arizona diamondbacks

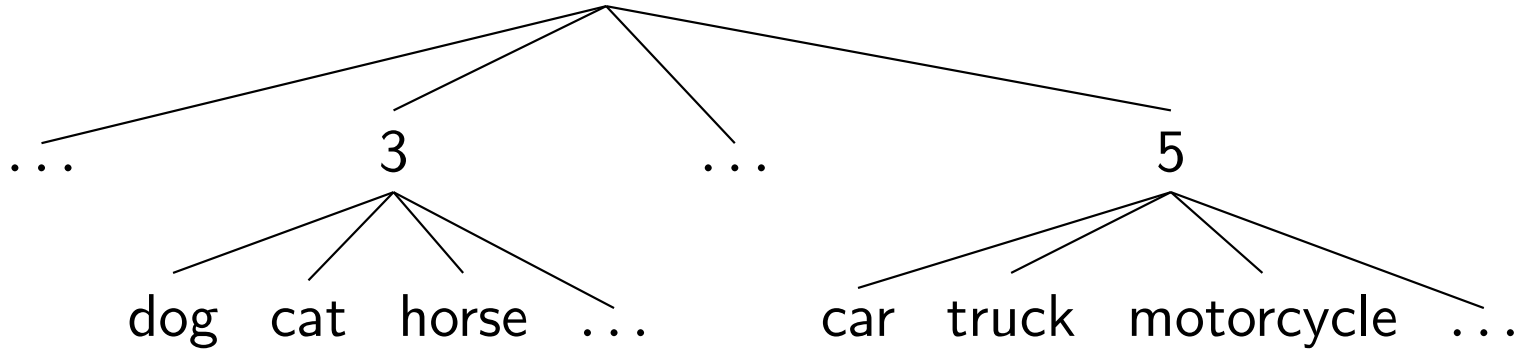
  Whereas at test we might see,

  the eagles play the colorado ___

  (We will discuss this issue more for in MT)

# Baseline: Class-Based Language Models

- Groups words into classes based on word-context.




- Various factorization methods for estimating with count-based approaches.

- However, assumes a hard-clustering, often estimated separately.

# Contents

# Recall: Word Embeddings

- Embeddings give multi-dimensional representation of words.
- Ex: Closest by cosine similarity

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

- Gives a multi-clustering over words.

# Feed-Forward Neural NNLM (Bengio, 2003)

- $w_{i-n+1}, \ldots w_{i-1}$ are input embedding representations
- $w_i$ is an output embedded representation
- Model simultaneously learns,
  - input word relations
  - output word relations
  - conjunctions of input words (through NLM, no n-gram features)

# Feed-Forward Neural Representation

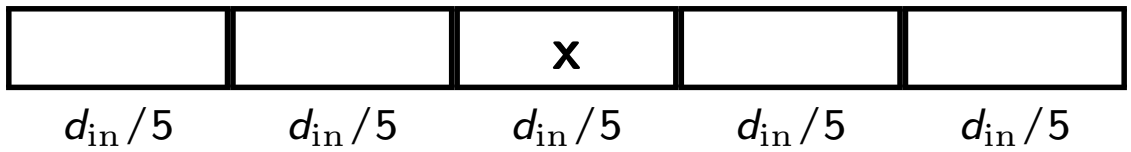
- $p(w_i | w_{i-n+1}, \ldots w_{i-1}; \theta)$
- $f_1, \ldots, f_{d_{\text{win}}}$ are words in window
- Input representation is the concatenation of embeddings

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

Example: NNLM ($d_{\text{win}} = 5$)

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

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



# A Neural Probabilistic Language Model (Bengio, 2003)

One hidden layer multi-layer perceptron architecture,

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

Neural network architecture on top of concat.

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

Best model uses $d_{\text{in}} = 30$, $d_{\text{hid}} = 100$.

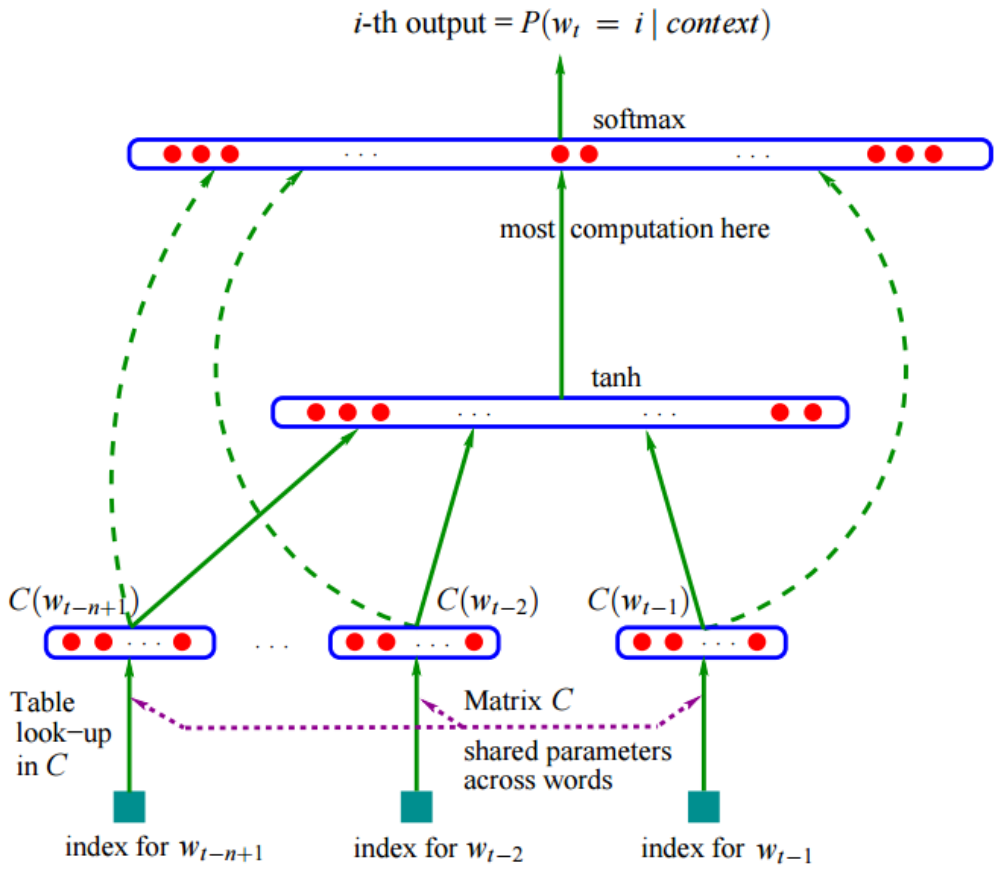
# A Neural Probabilistic Language Model

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

| | 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 |

Dashed-lines show the optional direct connections, $C = v$.

# Parameters

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

# Historical Note

- Bengio et al notes that many of these aspects predate the work
- Furthermore proposes many of the ideas that Collobert et al. and word2vec implemnt and scale
- Around this time, very few NLP papers on NN, most-cited papers are about conditional random fields (CRFs).

# Log-Bilinear Language Model

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

- Remove the tanh layer.
- Dense $\mathbf{x}$ concatenated word-embeddings
- Can be faster to use, and in some cases simpler.

# Comparison

Both count-based models and feed-forward NNLMs are Markovian language models,

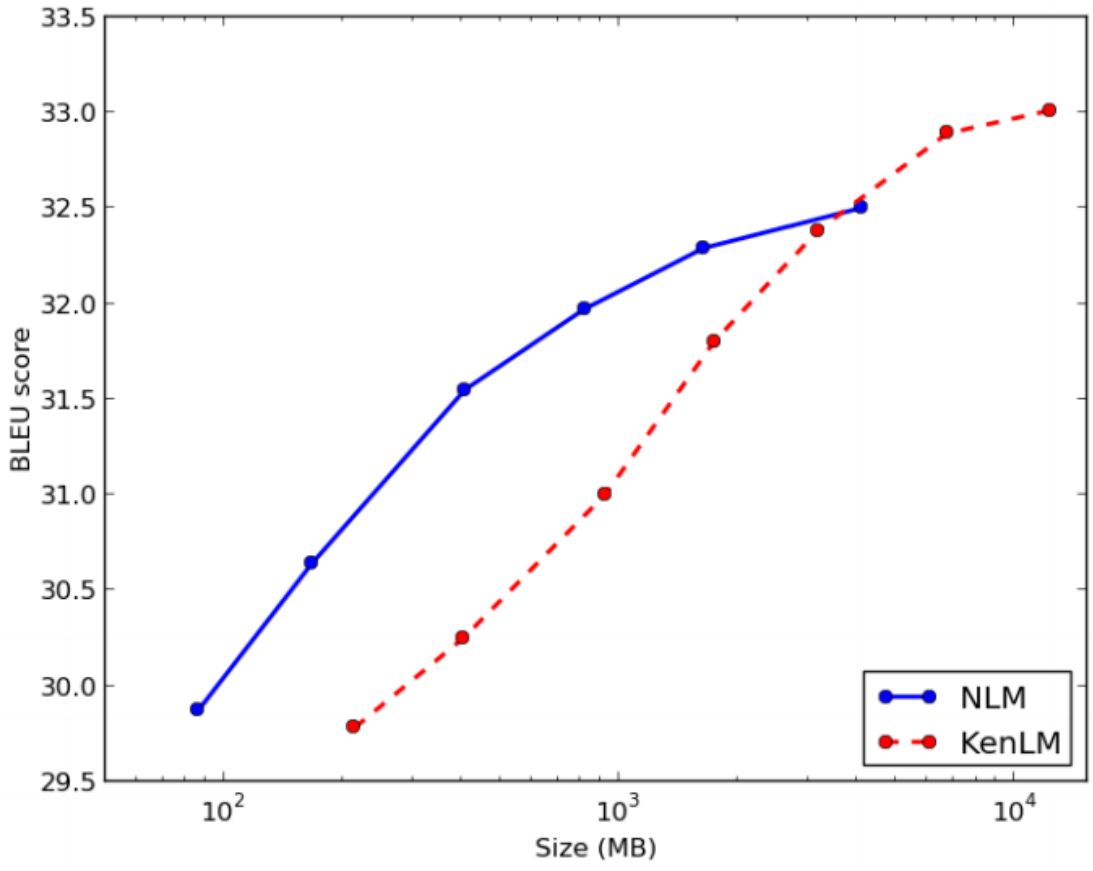
Comparison:

- Training Speed: ngrams are much faster (more coming)
- Usage Speed: ngrams very fast, NN can be fast with some tricks.
- Memory: NN models can be much smaller (but there are big ones)
- Accuracy: Comparable for small data, NN does better with more.

Advantages of NN model

- Can be trained end-to-end.
- Does not require smoothing methods.

# Translation Performance ( and Blunsom, 2015)



# 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 Scores

Recall the score defined as (dropping bias)

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

Unnormalized score of each word before soft-max,

$$z_j = \tanh(\mathbf{x}\mathbf{W}^1)\mathbf{W}^2_{*,j}$$

for any $j \in \{1, \ldots d_{\text{out}}\}$

Note: can be computed efficiently $O(1)$ versus $O(d_{\text{out}})$.

# Coherence

- Saw similar idea earlier for ranking embedding.
- **Idea:** Learn to distinguish coherent n-grams from corruption.
- Want to discriminate correct next words from other choices.

[ the dog walks ]

[ the dog house ]

[ the dog cats ]

[ the dog skips ]

# Warm-Up

Imagine we have a new dataset,

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

- $\mathbf{x}$; representation of context $w_{i-n+1}, \ldots w_{i-1}$
- $\mathbf{y}$; a possible $w_i$
- $d$; 1 if $\mathbf{y}$ is correct, 0 otherwise

Objective is based on predicted :

$$\mathcal{L}(\theta) = \sum_i L_{crossentropy}(d_i, \hat{d}_i)$$

# Warm-Up: Binary Classification

How do we score $(\mathbf{x}_i, \mathbf{y}_i = \delta(c))$?

Could use unnormalized score,

$$z_c = \tanh(\mathbf{x}\mathbf{W}^1)\mathbf{W}^2_{*,c}$$

Becomes softmax regression/non-linear logistic regression,

$$\hat{d} = \sigma(z_c)$$

- Much faster
- But does not help us train LM.

# Implementation

Standard MLP language model, (only takes in $\mathbf{x}$)

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

Computing binary (takes in $\mathbf{x}$ and $\mathbf{y}$)

$$\hat{d} = \sigma(z_c)$$

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

# Noise Contrastive Estimation 1

Probabilistic model,

- Introduce random variable $D$
- If $D = 1$ produce true sample
- If $D = 0$ produce sample from a noise distribution.
- Hyperparameter $K$ is ratio of noise

$$p(D = 1) = \frac{1}{K + 1}$$

$$p(D = 0) = \frac{K}{K + 1}$$

# Noise Contrastive Estimation 2

For a given $\mathbf{x}, \mathbf{y}$,

$$\begin{aligned} p(D=1|\mathbf{x},\mathbf{y}) &= \frac{p(\mathbf{y}|D=1,\mathbf{x})p(D=1|\mathbf{x})}{\sum_d p(\mathbf{y}|D=d,\mathbf{x})p(D=d|\mathbf{x})} \\ &= \frac{p(\mathbf{y}|D=1,\mathbf{x})p(D=1|\mathbf{x})}{p(\mathbf{x}|D=0)p(D=0|\mathbf{x}) + p(\mathbf{y}|D=1,\mathbf{x})p(D=1|\mathbf{x})} \end{aligned}$$

Plug-in the noise distribution and hyperparameters,

$$\begin{aligned} p(D=1|\mathbf{x},\mathbf{y}) &= \frac{\frac{1}{K+1}p(\mathbf{y}|D=1,\mathbf{x})}{\frac{1}{K+1}p(\mathbf{y}|D=1,\mathbf{x}) + \frac{K}{K+1}p(\mathbf{y}|D=0,\mathbf{x})} \\ &= \frac{p(\mathbf{y}|D=1,\mathbf{x})}{p(\mathbf{y}|D=1,\mathbf{x}) + Kp(\mathbf{y}|D=1,\mathbf{x})} \\ &= \sigma(\log p(\mathbf{y}|D=1,\mathbf{x}) - \log(Kp(\mathbf{y}|D=0,\mathbf{x}))) \end{aligned}$$

# Noise Contrastive Estimation 3

$$p(D = 1|\mathbf{x}, \mathbf{y}) = \sigma(\log p(\mathbf{y}|D = 1, \mathbf{x}) - \log(Kp(\mathbf{y}|D = 0, \mathbf{x})))$$

- $\log p(\mathbf{y} = c|D = 1, \mathbf{x})$; becomes $z_c$ (as above)
- $\log p(\mathbf{y} = c|D = 0, \mathbf{x})$; noise distribution
- In practice, people use unigram for noise
- (Can precompute samples and noise scores)

# Noise Contrastive Estimation 4

Full objective,

- $s_{i,k}$ are $K$ samples for each position $i$

$$
\begin{aligned}
\mathcal{L}(\theta) &= \sum_i \log p(D = 1|\mathbf{x}_i, \mathbf{y}_i) + \sum_{k=1}^{K} \log p(D = 0|\mathbf{x}_i, Y = s_{i,k}) \\
&= \sum_i \log \sigma(\hat{z}_c - \log(K p_{ML}(c))) \\
&+ \sum_{k=1}^{K} \log(1 - \sigma(\hat{z}_{s_{i,k}} - \log(K p_{ML}(s_{i,k}))))
\end{aligned}
$$

- Paper shows that this yields a consistent estimation of the true LM distribution

# Implementation

- How do you efficiently compute $z_c$?

  Need a lookup table for output embeddings! (not linear) and dot product.
- How do you efficiently handle $p_{ML}(c)$

  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 $\sigma(z_c - \log(Kp_{ML}(c)))$,

$$\mathbf{x} \Rightarrow \mathbf{W}^1 \Rightarrow \tanh \Rightarrow \overset{\cdot}{\mathbf{W}^2_{*,w}(\text{Lookup})} \Rightarrow \overset{-}{Kp_{ML}(c)(\text{Lookup})} \Rightarrow \sigma$$

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

# Using in Practice

Several options for test time,

- Use full softmatx with learned parameters.
- Compute subset of scores and renormalize (homework) .
- Can sometimes just use treat unormalized params as being normalized (self-normalization)