FFNNs for Classification and Language Modeling

LING 575K Deep Learning for NLP

Shane Steinert-Threlkeld

April 13 2022

Announcements

- Running time:
  - Many factors influence this, including the load on nodes on patas
  - So don’t worry too much about your raw numbers!
  - Do: run in advance; it may take several hours
  - HW2: ~1-2hrs for our ref
- HW2 written: be detailed, e.g. explain your steps (can help with partial credit)
- HW1: good grades overall, well done!
  - Data statements: hard to do :) b/c not well-documented, tracing all the way back to data curation reveals that all the reviews are from only 4 critics O_O

Today's Plan

- Deep Averaging Networks for text classification
- Neural Probabilistic Language Model
- Additional Training Notes
  - Regularization
  - Early stopping
  - Hyper-parameter searching
- HW3 / edugrad / PyTorch

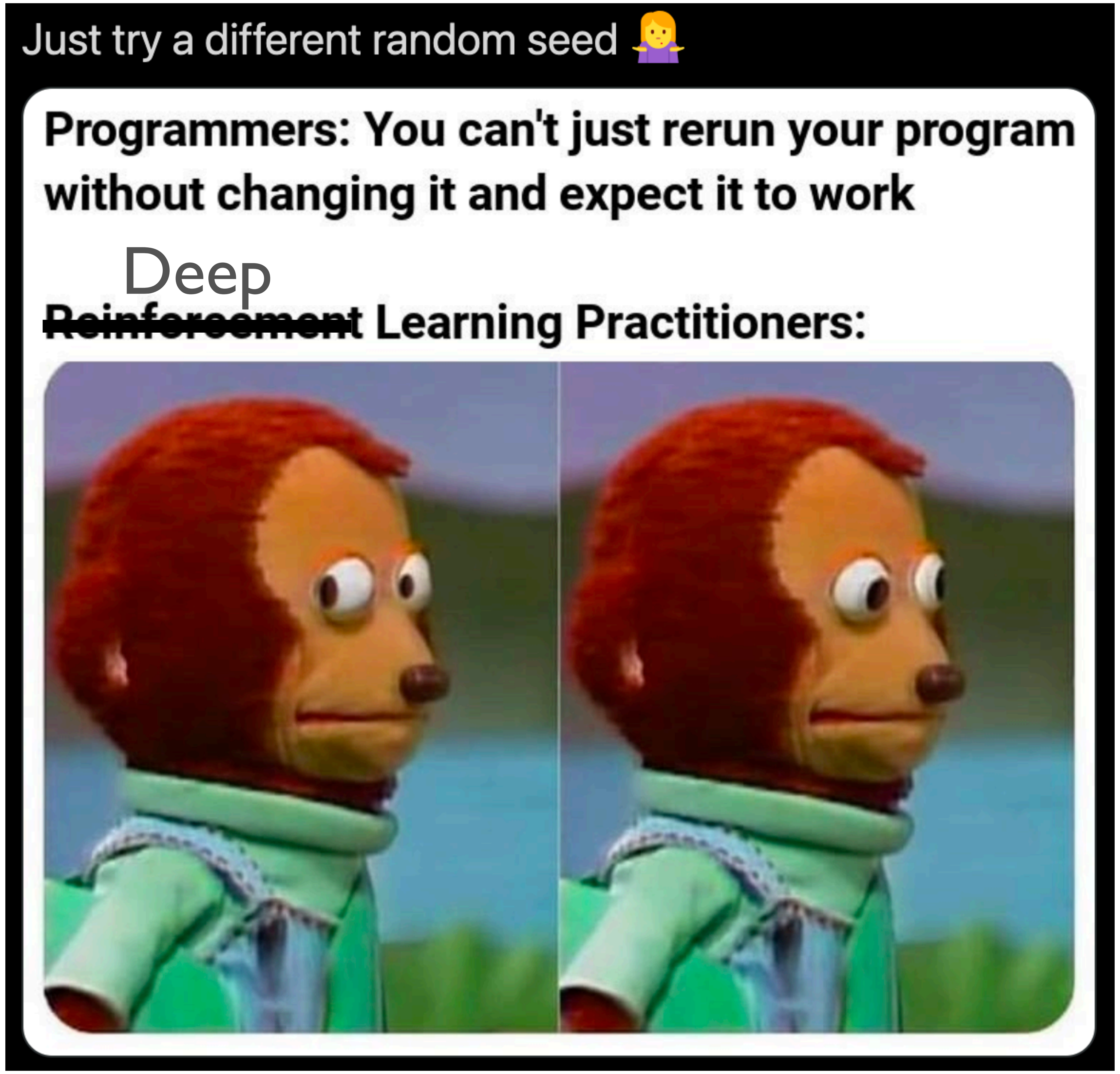
Note on Random Seeds

- In word2vec.py / util.py:
- Random seed:
  - Behavior of pseudo-random number generators is determined by their “seed” value
  - If not specified, determined by e.g. # of seconds since 1970
  - Same seed —> same (non-random) behavior
- Sources of randomness in DL: shuffling the data each epoch, weight initialization, negative *sampling*, …
- Very important for reproducibility!
  - In general, run on several seeds and report means / std’s

```
# set random seed
util.set_seed(args.seed)
```

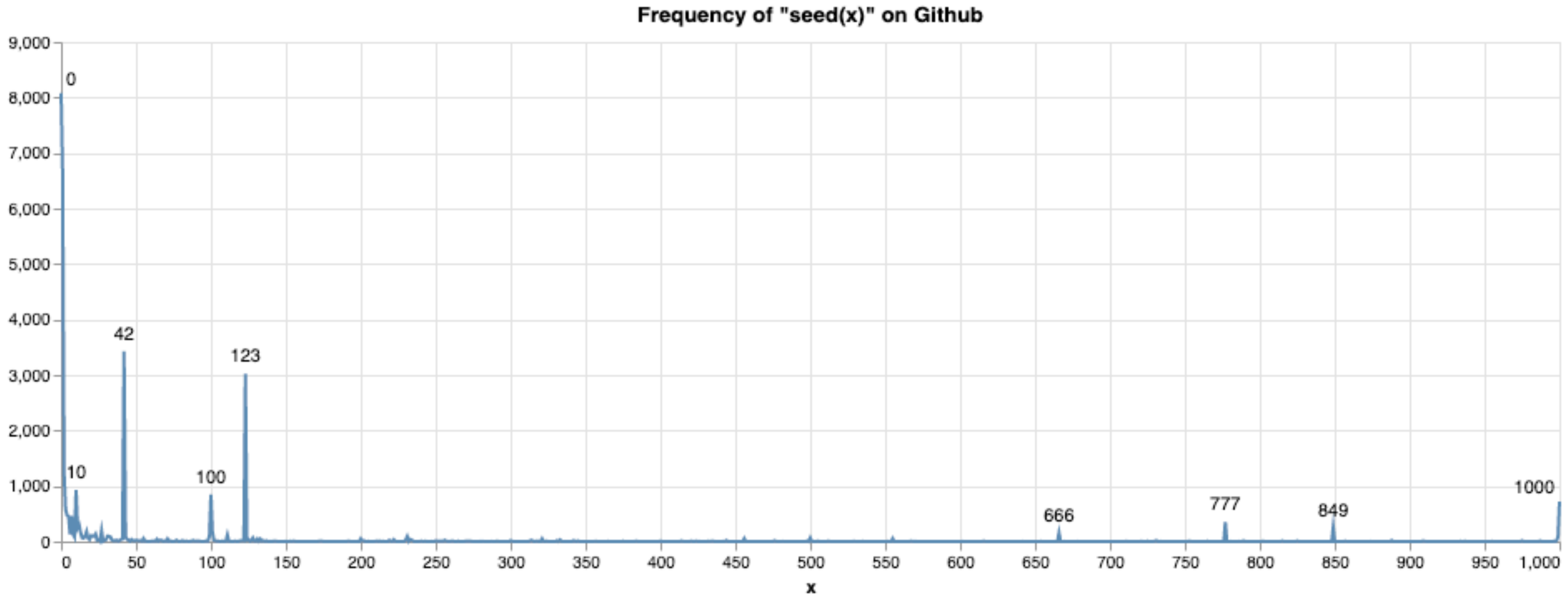
```
def set_seed(seed: int) -> None:
    """Sets various random seeds. """
    random.seed(seed)
    np.random.seed(seed)
```

Random Seeds and Reproducibility



Random Seeds, cont

- Ideally: “randomly generate” seeds, but save/store them!
- Random seed is not a hyper-parameter! (Some discussions in [these threads](#).)



[source](#)

Deep Averaging Networks

Deep Unordered Composition Rivals Syntactic Methods for Text Classification

Mohit Iyyer,[1] **Varun Manjunatha,**[1] **Jordan Boyd-Graber,**[2] **Hal Daumé III**[1]
[1]University of Maryland, Department of Computer Science and UMIACS
[2]University of Colorado, Department of Computer Science
{miyyer,varunm,hal}@umiacs.umd.edu, Jordan.Boyd.Graber@colorado.edu

Abstract

Many existing deep learning models for natural language processing tasks focus on learning the *compositionality* of their inputs, which requires many expensive computations. We present a simple deep neural network that competes with and, in some cases, outperforms such models on sen-

results have shown that syntactic functions outperform unordered functions on many tasks (Socher et al., 2013b; Kalchbrenner and Blunsom, 2013).

However, there is a tradeoff: syntactic functions require more training time than unordered composition functions and are prohibitively expensive in the case of huge datasets or limited computing resources. For example, the recursive neural network (Section 2) computes costly matrix/tensor products

Deep, Unordered, Classification

Deep, Unordered, Classification

- Deep:
  - One or more hidden layers in a neural network

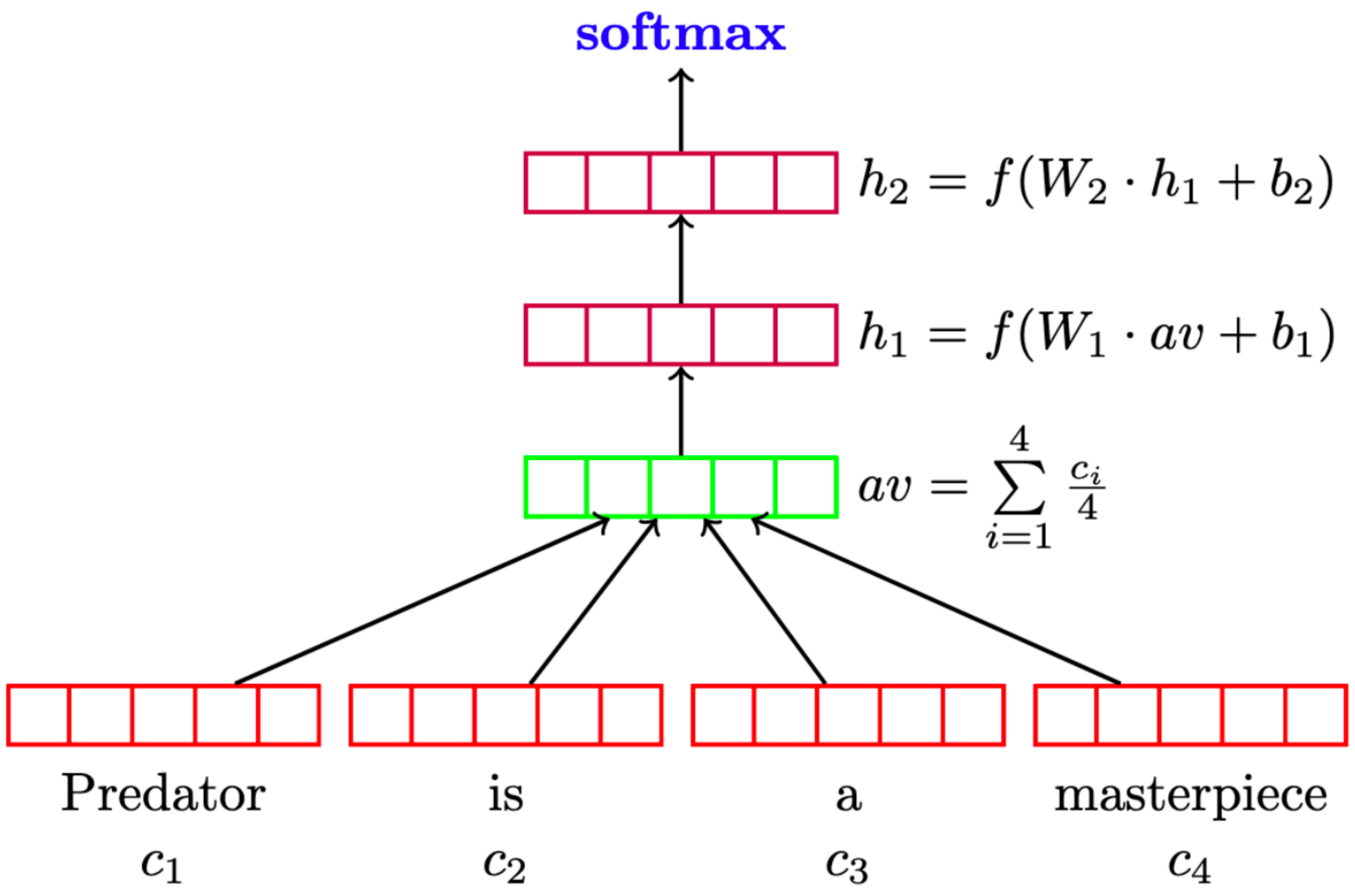
Deep, Unordered, Classification

- Deep:
  - One or more hidden layers in a neural network
- Unordered:
  - Text is represented as a “bag of words”
  - No notion of syntactic order

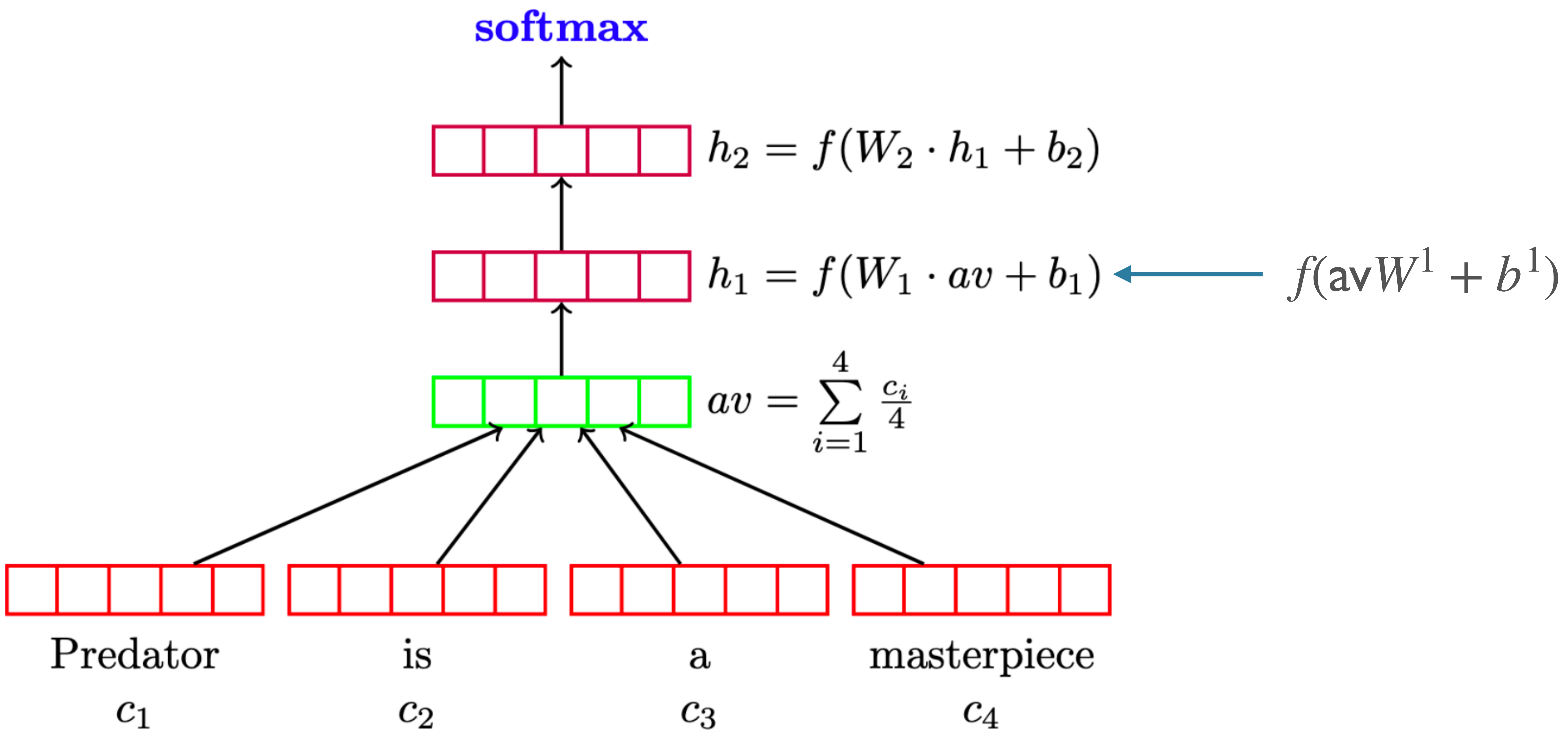
Deep, Unordered, Classification

- Deep:
  - One or more hidden layers in a neural network
- Unordered:
  - Text is represented as a “bag of words”
  - No notion of syntactic order
- Classification:
  - Applied to several classification tasks, including SST
  - Via softmax layer

Model Architecture, One Input

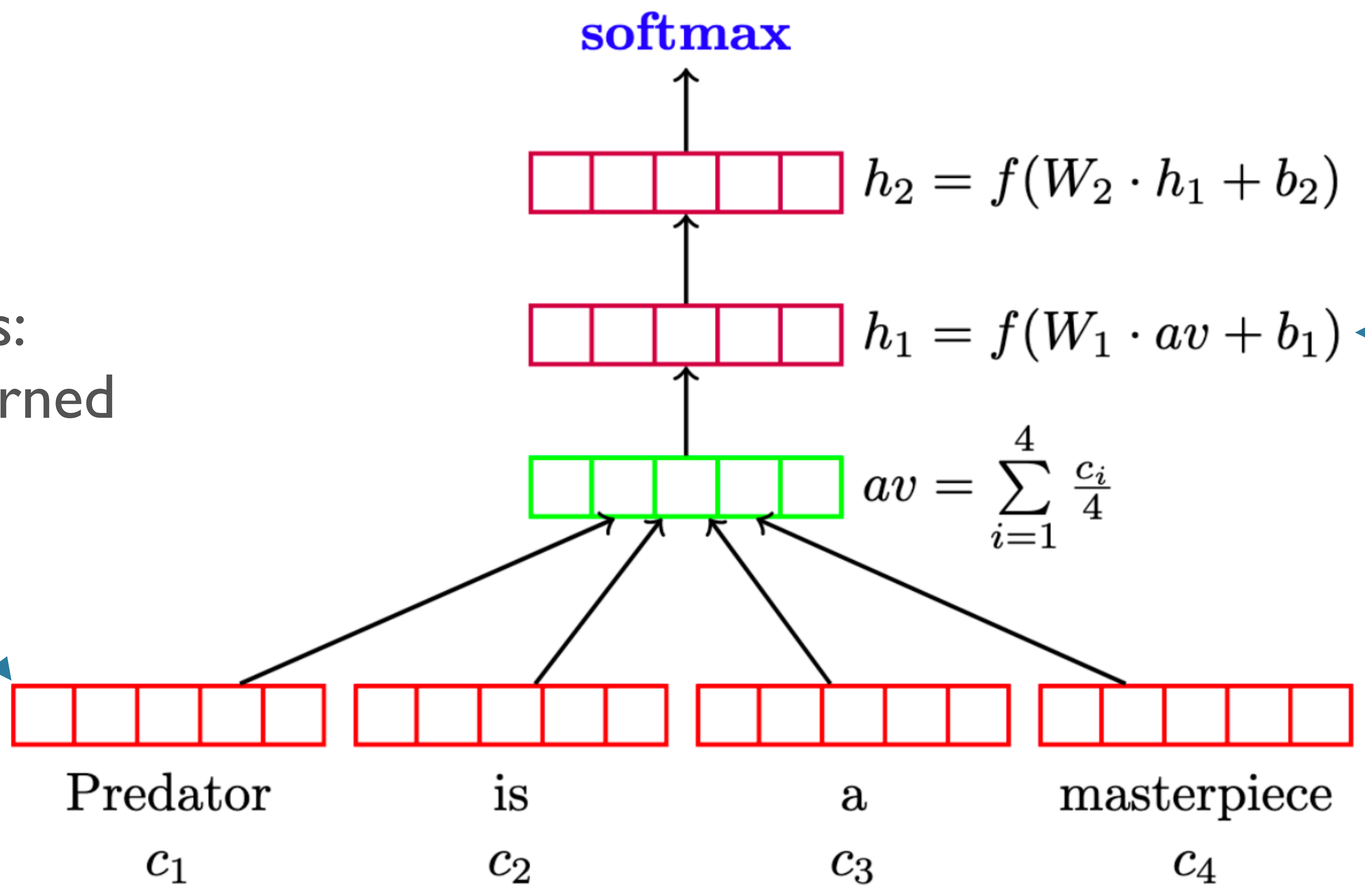


Model Architecture, One Input



Model Architecture, One Input

Word embeddings:
Pre-trained or learned



$f(\mathrm{av}W^1 + b^1)$

Hyper-parameters

Hyper-parameters

- Embedding dimension

Hyper-parameters

- Embedding dimension
- Number of hidden layers

Hyper-parameters

- Embedding dimension
- Number of hidden layers
- For each layer:
  - Activation function
  - Hidden dimension size

Hyper-parameters

- Embedding dimension
- Number of hidden layers
- For each layer:
  - Activation function
  - Hidden dimension size
- Exercise: find the values for these hyper-parameters in the paper

Note on Embedding Layer

- Let t be the integer index of word w
- One-hot vector (t=4): $w_t = [0 \quad 0 \quad 0 \quad 1 \quad \cdots \quad 0]$
- For E an embedding matrix of shape [vocab_size, embedding_dimension] and E_t the embedding for t:

$$E_t = w_t E$$

- NB: direct look-up is faster than matrix multiplication, but the latter generalizes in useful ways that we will see soon

Batched Computation in DAN

- We saw how to pass one piece of text through the DAN
- How can we leverage larger batch sizes and their advantages?
  - “Predator is a masterpiece”
  - “Parasite won Best Picture for 2019”
- What issues here?
- Different lengths —> different number of embeddings —> different input size (intuitively)
  - But we need a matrix of shape [batch_size, representation_size] for inputs

Batching with Bag of Words

- Bag of words representation:
  - {word1: 3, word36: 1, word651: 1, …}
  - Let s be a sentence with words t_i occurring count_i times: $\text{bag}_s := \{t_i : \text{count}_i\}$
- Bag of words vector: $\text{vec}_s := [3 \quad 0 \quad \cdots \quad 1 \quad \cdots \quad 1 \quad \cdots]$

$$\text{vec}_s E = \sum_{i=0}^{\text{len}(s)} E_{s_i} = \sum_{t \in s} E_t \cdot \text{count}_t$$

- For every sentence, the vec_s vectors have the same size: [vocab size]
  - So they can be stacked into a matrix, of shape [batch_size, vocab_size]
  - Divide each row by length of that sentence to get average of embeddings

Output and Loss for Classification

$$\text{logits} = \text{hidden}W + b$$

$$\hat{y} = \text{probs} = \text{softmax}(\text{logits})$$

Output and Loss for Classification

$$\text{logits} = \text{hidden}W + b$$

$$\hat{y} = \text{probs} = \text{softmax}(\text{logits})$$

$$\ell_{CE}(\hat{y}, y) = -\sum_{i=0}^{|\text{classes}|} y_i \log \hat{y}_i$$

Output and Loss for Classification

$$\text{logits} = \text{hidden}W + b$$

$$\hat{y} = \text{probs} = \text{softmax}(\text{logits})$$

$$\ell_{CE}(\hat{y}, y) = -\sum_{i=0}^{|\text{classes}|} y_i \log \hat{y}_i$$

One hot for true class label

Results

Model	RT	SST fine
DAN-ROOT	—	46.9
DAN-RAND	77.3	45.4
DAN	80.3	47.7
NBOW-RAND	76.2	42.3
NBOW	79.0	43.6
BiNB	—	41.9
NBSVM-bi	79.4	—
RecNN*	77.7	43.2
RecNTN*	—	45.7
DRecNN	—	49.8
TreeLSTM	—	**50.6**

Results

Model	RT	SST fine
DAN-ROOT	—	46.9
DAN-RAND	77.3	45.4
DAN	80.3	47.7
NBOW-RAND	76.2	42.3
NBOW	79.0	43.6
BiNB	—	41.9
NBSVM-bi	79.4	—
RecNN*	77.7	43.2
RecNTN*	—	45.7
DRecNN	—	49.8
TreeLSTM	—	**50.6**

"Rivals syntactic methods"

Error Analysis

Sentence	DAN	DRecNN	Ground Truth
a lousy movie that's not merely unwatchable, but also unlistenable	negative	negative	negative
if you're not a prepubescent girl, you'll be laughing at britney spears' movie-starring debut whenever it does n't have you impatiently squinting at your watch	negative	negative	negative
blessed with immense physical prowess he may well be, but ahola is simply not an actor	positive	neutral	negative
who knows what exactly godard is on about in this film, but his words and images do n't have to add up to mesmerize you.	positive	positive	positive
it's so good that its relentless, polished wit can withstand not only inept school productions, but even oliver parker's movie adaptation	negative	positive	positive
too bad, but thanks to some lovely comedic moments and several fine performances, it's not a total loss	negative	negative	positive
this movie was not good	negative	negative	negative
this movie was good	positive	positive	positive
this movie was bad	negative	negative	negative
the movie was not bad	negative	negative	positive

Two Additional “Tricks”

- Word dropout
  - A type of *regularization* [more later]
- Adagrad optimizer

Word Dropout

- For each input sequence, flip |V| coins with probability *p*
  - If the i'th coin lands tails, set embedding for w_i to all 0s for this example

Word Dropout

- For each input sequence, flip |V| coins with probability *p*
  - If the i'th coin lands tails, set embedding for w_i to all 0s for this example

$$\begin{aligned} \text{vec}_s &= [20110] \\ \text{mask} &= [01110] \\ \text{vec}_s \odot \text{mask} &= [00110] \end{aligned}$$

Word Dropout

- For each input sequence, flip |V| coins with probability *p*
  - If the i'th coin lands tails, set embedding for w_i to all 0s for this example

$$\text{vec}_s = [20110]$$

$$\text{mask} = [01110]$$

Generated randomly for each sentence

$$\text{vec}_s \odot \text{mask} = [00110]$$

Adagrad

- "Adaptive Gradients"
  - Key idea: *adjust the learning rate* per parameter
  - Frequent features —> more updates
  - Adagrad will make the learning rate smaller for those

Adagrad

- Let $g_{t,i} := \nabla_{\theta_{t,i}} \mathscr{L}$
- SGD: $\theta_{t+1,i} = \theta_{t,i} - \alpha g_{t,i}$
- Adagrad: $$\theta_{t+1,i} = \theta_{t,i} - \frac{\alpha}{\sqrt{G_{t,i} + \epsilon}} g_{t,i}$$

$$G_{t,i} = \sum_{k=0}^{t} g_{k,i}^2$$

Adagrad

- Pros:
  - “Balances” parameter importance
  - Less manual tuning of learning rate needed (0.01 default)
- Cons:
  - $G_{t,i}$ increases monotonically, so step-size always gets smaller
- Newer optimizers try to have the pros without the cons
- Resources:
  - Original paper (veeery math-y): https://jmlr.org/papers/volume12/duchi11a/duchi11a.pdf
  - Overview of optimizers: https://ruder.io/optimizing-gradient-descent/index.html#adagrad

Unordered Models in the Large LM Era

- Last paper: “Deep Unordered Composition Rivals Syntactic Methods for Text Classification” —2015
- From ~April 2021:

Masked Language Modeling and the Distributional Hypothesis: Order Word Matters Pre-training for Little

Koustuv Sinha[†‡] **Robin Jia**[†] **Dieuwke Hupkes**[†] **Joelle Pineau**[†‡]

Adina Williams[†] **Douwe Kiela**[†]

[†] Facebook AI Research; [‡] McGill University / Montreal Institute of Learning Algorithms

{koustuvs,adinawilliams,dkiela}@fb.com

Abstract

A possible explanation for the impressive performance of masked language model (MLM) pre-training is that such models have learned to represent the syntactic structures prevalent in classical NLP pipelines. In this paper, we propose a different explanation: MLMs succeed on downstream tasks almost entirely due to their ability to model higher-order word co-occurrence statistics. To demonstrate this, we pre-train MLMs on sentences with randomly shuffled word order, and show that

NLP pipeline” (Tenney et al., 2019), suggesting that it has learned “the kind of abstractions that we intuitively believe are important for representing natural language” rather than “simply modeling complex co-occurrence statistics” (ibid., p. 1).

In this work, we try to uncover how much of MLM’s success comes from simple distributional information, as opposed to “the types of syntactic and semantic abstractions traditionally believed necessary for language processing” (Tenney et al., 2019; Manning et al., 2020). We disentangle these

Unordered Models in the Large LM Era

- Last paper: “Deep Unordered Composition Rivals Syntactic Methods for Text Classification” —2015
- From ~April 2021:

Masked Language Modeling and the Distributional Hypothesis: Order Word Matters Pre-training for Little

Koustuv Sinha[†‡] Robin Jia[†] Dieuwke Hupkes[†] Joelle Pineau[†‡]

Adina Williams[†] Douwe Kiela[†]

[†] Facebook AI Research; [‡] McGill University / Montreal Institute of Learning Algorithms

{koustuvs,adinawilliams,dkiela}@fb.com

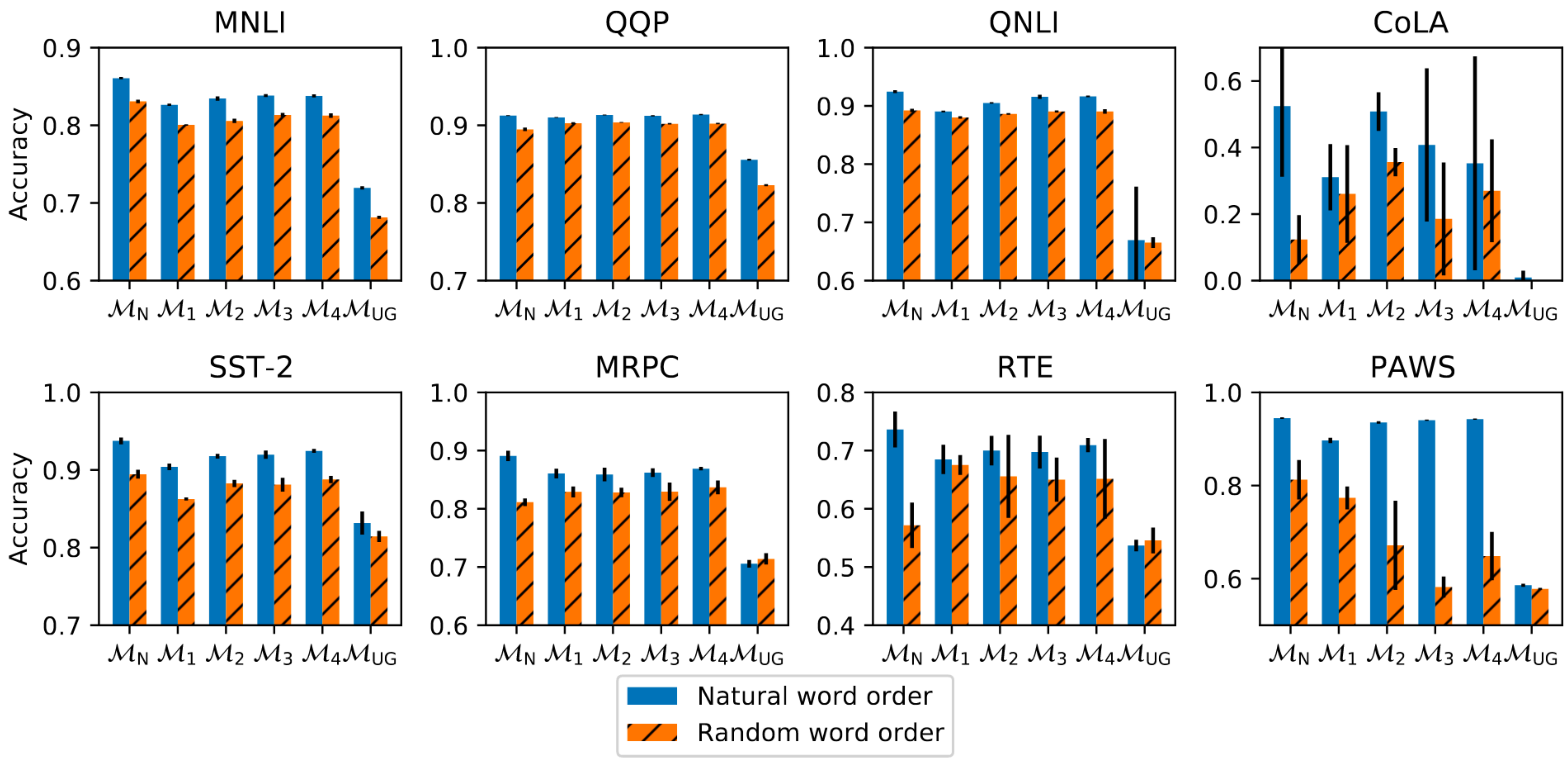
Abstract

A possible explanation for the impressive performance of masked language model (MLM) pre-training is that such models have learned to represent the syntactic structures prevalent in classical NLP pipelines. In this paper, we propose a different explanation: MLMs succeed on downstream tasks almost entirely due to their ability to model higher-order word co-occurrence statistics. To demonstrate this, we pre-train MLMs on sentences with randomly shuffled word order, and show that

NLP pipeline” (Tenney et al., 2019), suggesting that it has learned “the kind of abstractions that we intuitively believe are important for representing natural language” rather than “simply modeling complex co-occurrence statistics” (ibid., p. 1).

In this work, we try to uncover how much of MLM’s success comes from simple distributional information, as opposed to “the types of syntactic and semantic abstractions traditionally believed necessary for language processing” (Tenney et al., 2019; Manning et al., 2020). We disentangle these

Unordered Models in the Large LM Era



Unordered Models in the Large LM Era

- “We observed overwhelmingly that MLM’s success is most likely **not** [emphasis added] due to its ability to discover syntactic and semantic mechanisms necessary for a traditional language processing pipeline. Instead, our experiments suggest that MLM’s success can be mostly explained by it having learned higher-order distributional statistics that make for a useful prior for subsequent fine-tuning.”

Neural Probabilistic Language Model

Language Modeling

- A language model parametrized by θ computes $P_\theta(w_1, \ldots, w_n)$
- Typically (though we'll see variations): $P_\theta(w_1, \ldots, w_n) = \prod_i P_\theta(w_i \mid w_1, \ldots, w_{i-1})$
- E.g. of labeled data: "Today is the sixth day of 575k." —>
  - (<s>, Today)
  - (<s> Today, is)
  - (<s> Today is, the)
  - (<s> Today is the, sixth)

N-gram LMs

- Dominant approach for a long time uses n-grams:

$$P_\theta(w_i \mid w_1, \ldots, w_{i-1}) \approx P_\theta(w_i \mid w_{i-1}, w_{i-2}, \ldots, w_{i-n})$$

- Estimate the probabilities by counting in a corpus
    - Fancy variants (back-off, smoothing, etc)
- Some problems:
    - Huge number of parameters: $\approx |V|^n$
    - Doesn't generalize to unseen n-grams

Neural LM

- Core idea behind the Neural Probabilistic LM
  - Make n-gram assumption
  - But: learn word embeddings
  - “N-gram of word vectors”
  - Probabilities: represented by a neural network, not counts

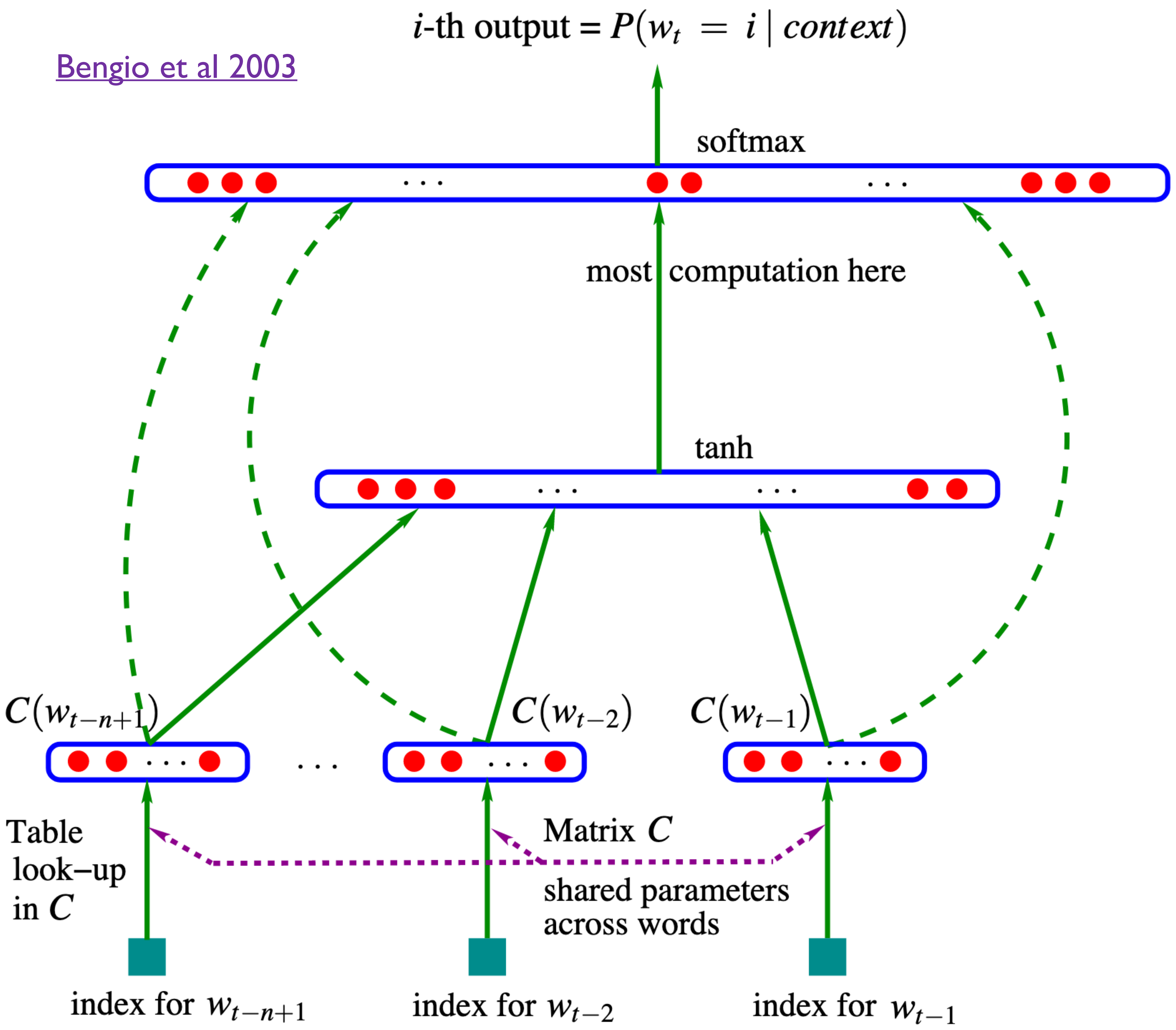
Pros of Neural LM

- Number of parameters:
  - Significantly lower, thanks to “low”-dimensional embeddings
- Generalization: embeddings enable generalizing to similar words

	The cat is walking in the bedroom
to	A dog was running in a room
and likewise to	The cat is running in a room
	A dog is walking in a bedroom
	The dog was walking in the room

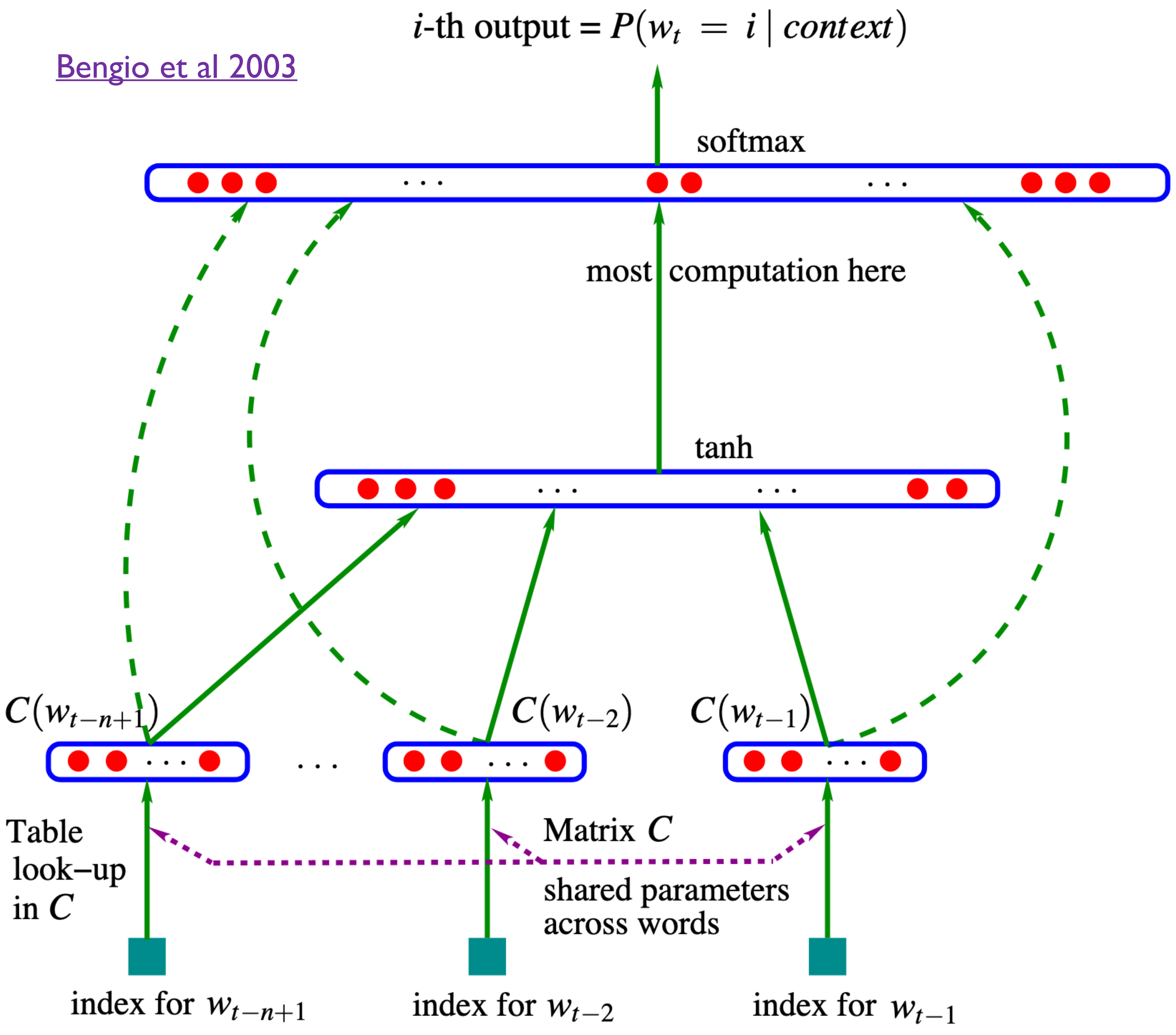
Neural LM Architecture

Bengio et al 2003



Neural LM Architecture

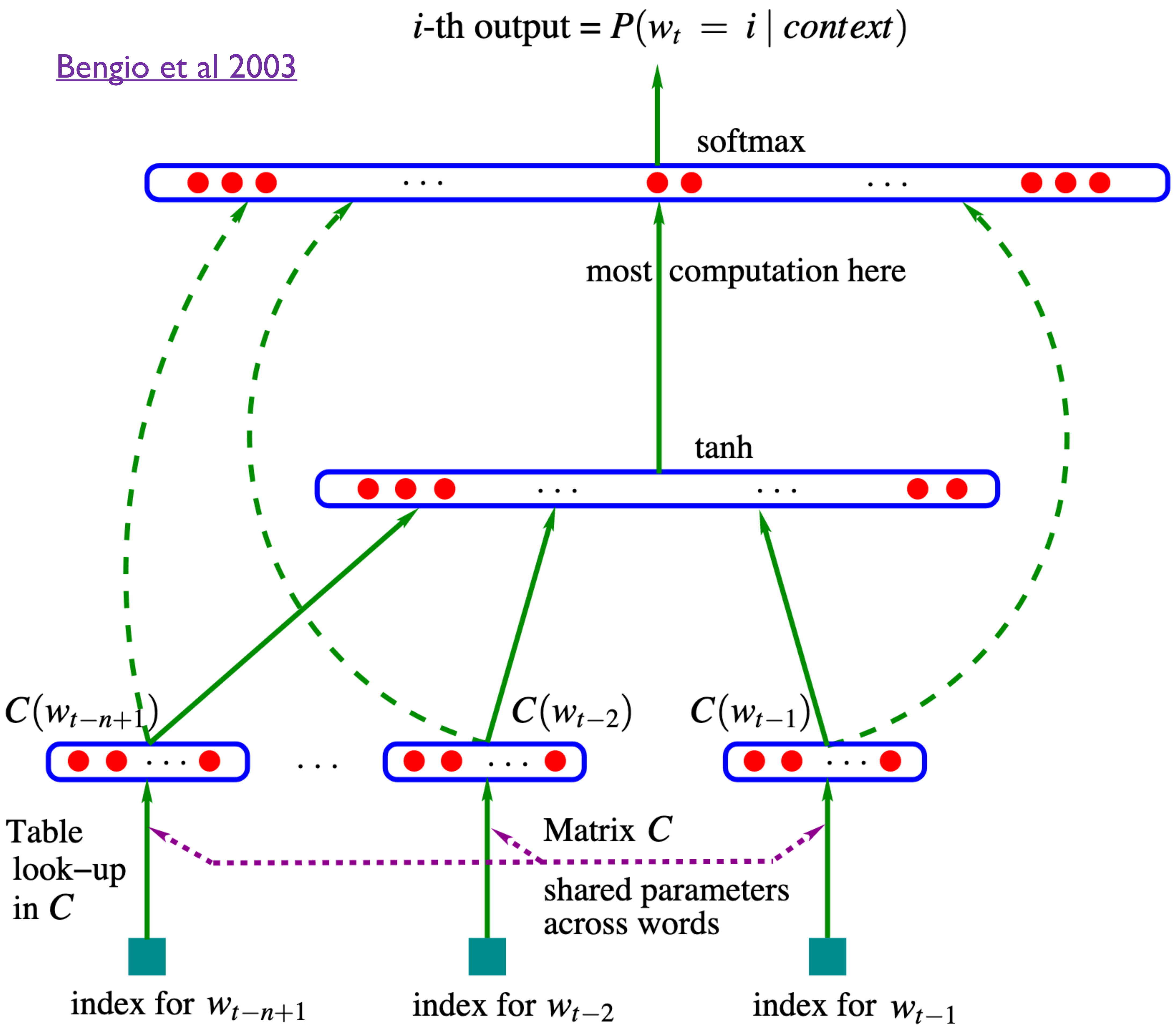
[Bengio et al 2003]



w_t: one-hot vector

Neural LM Architecture

[Bengio et al 2003](#)

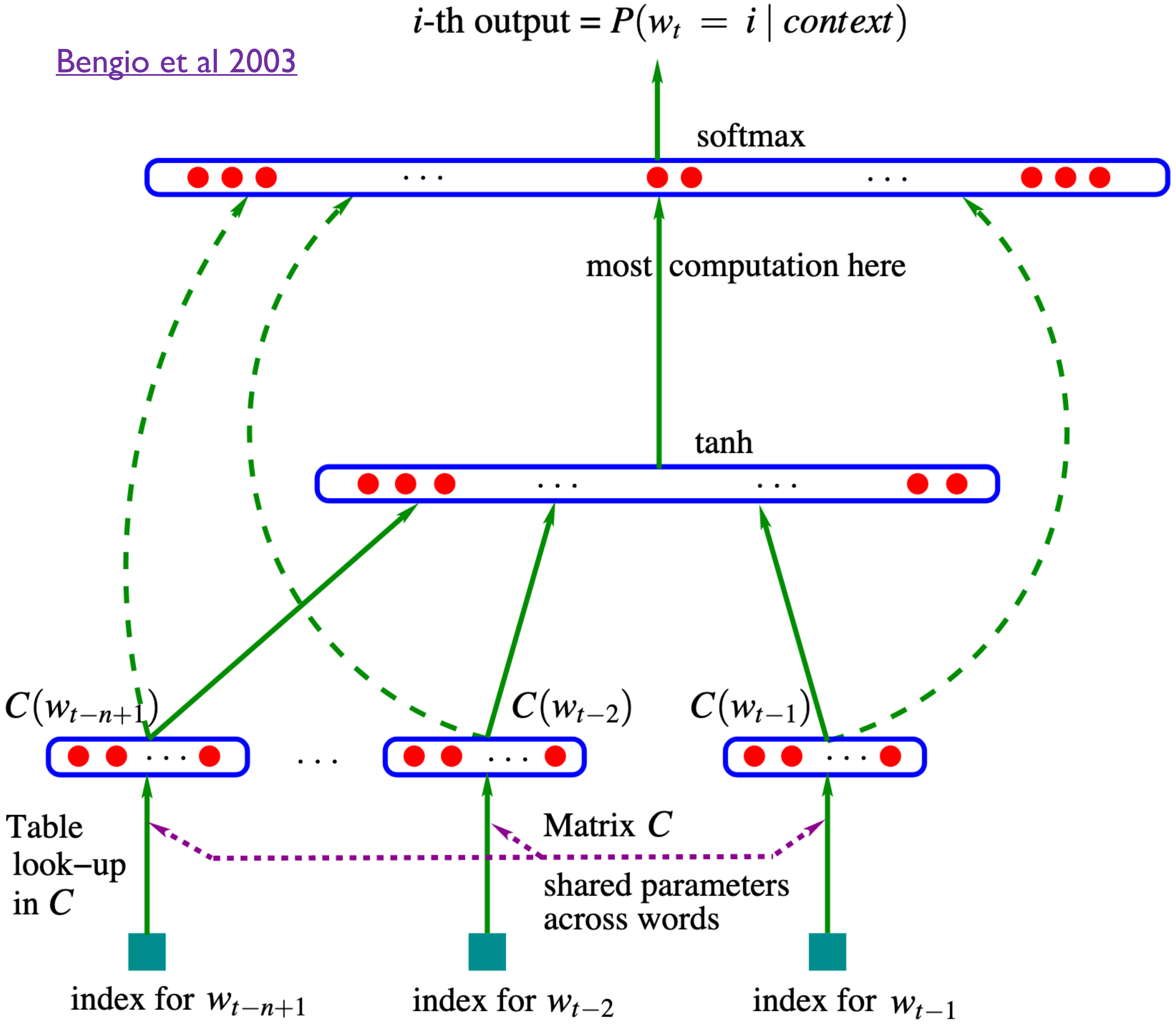


embeddings $= \text{concat}(w_{t-1}C, w_{t-2}C, \ldots, w_{t-(n+1)}C)$

w_t: one-hot vector

Neural LM Architecture

[Bengio et al 2003]



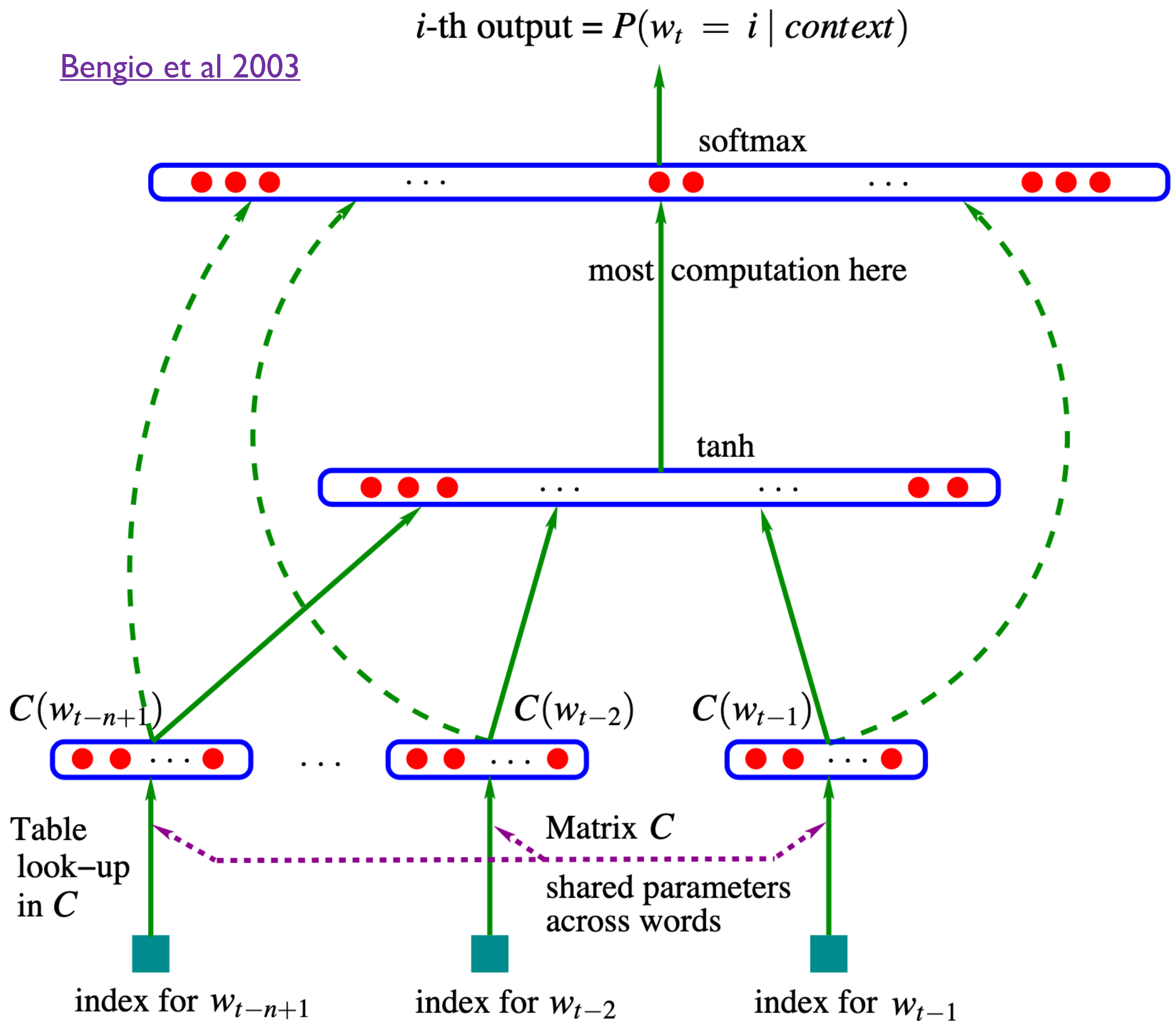
$$\text{hidden} = \tanh(\text{embeddings}\, W^1 + b^1)$$

$$\text{embeddings} = \text{concat}(w_{t-1}C, w_{t-2}C, \ldots, w_{t-(n+1)}C)$$

w_t: one-hot vector

Neural LM Architecture

Bengio et al 2003



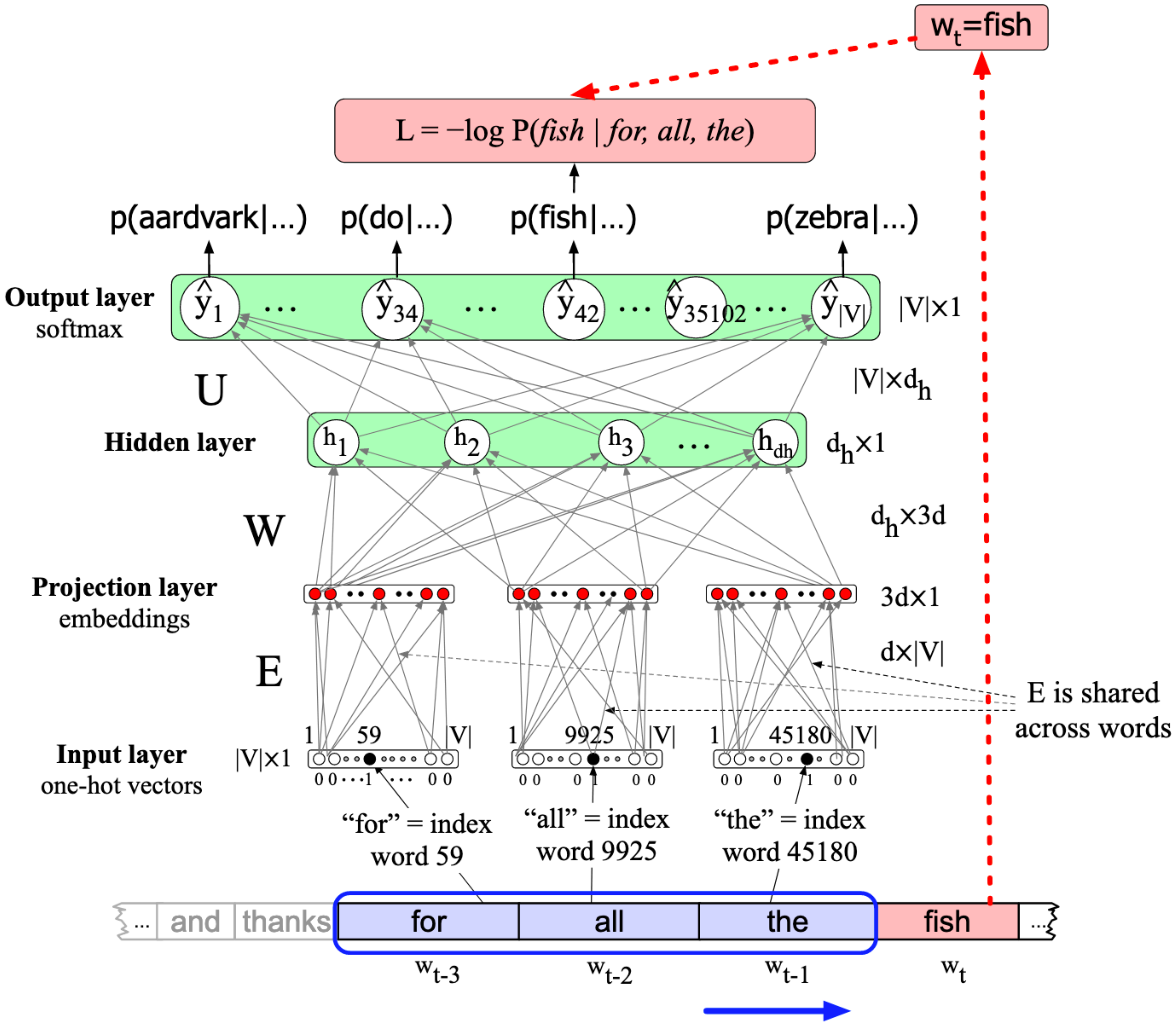
$\text{probabilities} = \text{softmax}(\text{hidden}W^2 + b^2)$

$\text{hidden} = \tanh(\text{embeddings}W^1 + b^1)$

$\text{embeddings} = \text{concat}(w_{t-1}C, w_{t-2}C, \ldots, w_{t-(n+1)}C)$

w_t: one-hot vector

More Detailed Diagram of Architecture



JM sec 7.5

Output and Loss

- Softmax + cross-entropy
  - Essentially, language modeling is |V|-way classification
  - Each word in the vocabulary is a class

Evaluation of LMs

- Extrinsic: use in other NLP systems
- Intrinsic: intuitively, want probability of a test corpus
- Perplexity: inverse probability, weighted by size of corpus
  - NB: lower is better!
  - Only comparable w/ same vocab

$$PP(W) = P(w_1 w_2 \cdots w_N)^{-1/N}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 \cdots w_N)}}$$

$$= \sqrt[N]{\frac{1}{\prod_{i=0}^{N} P(w_i \mid w_1, \ldots, w_{i-1})}}$$

$$= 2^{-\frac{1}{N}\sum_{i=0}^{N} \log P(w_i | w_1, \ldots, w_{i-1})}$$

Results

	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

More Complete Picture of This Model

Revisiting Simple Neural Probabilistic Language Models

Simeng Sun and **Mohit Iyyer**
College of Information and Computer Sciences
University of Massachusetts Amherst
{simengsun, miyyer}@cs.umass.edu

Abstract

Recent progress in language modeling has been driven not only by advances in neural architectures, but also through hardware and optimization improvements. In this paper, we revisit the neural probabilistic language model (NPLM) of Bengio et al. (2003), which simply concatenates word embeddings within a fixed window and passes the result through a feed-forward network to predict the next word. When scaled up to modern hardware, this model (despite its many limitations) performs

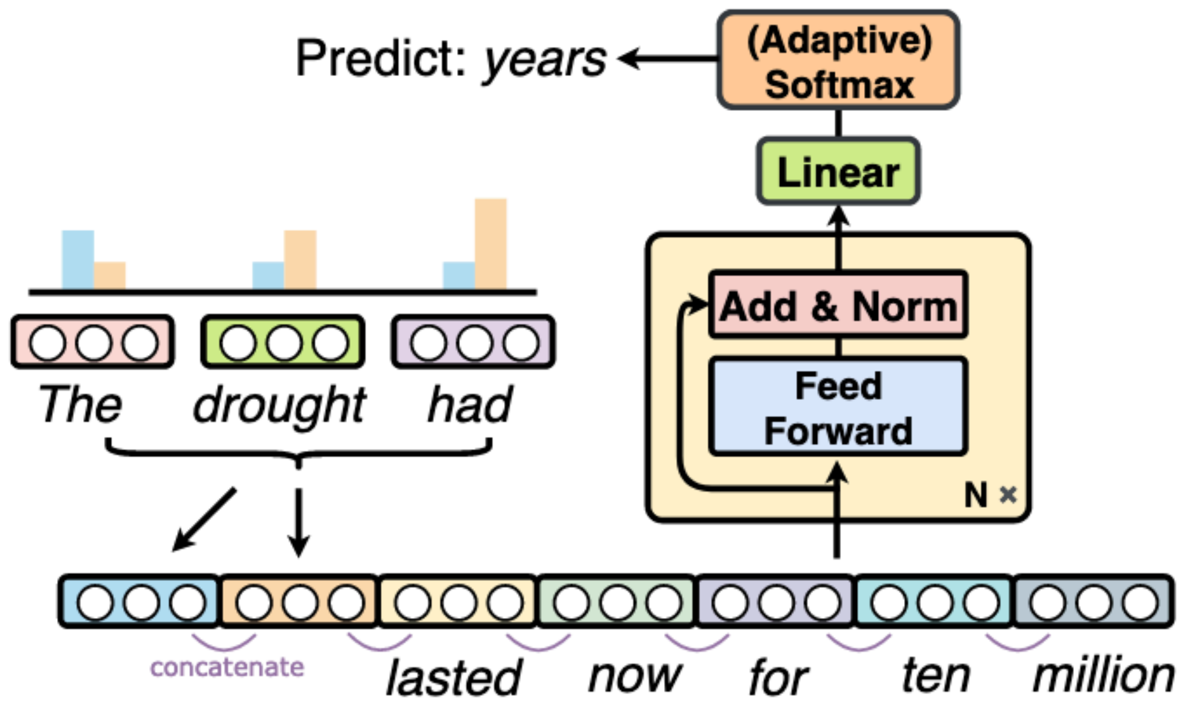


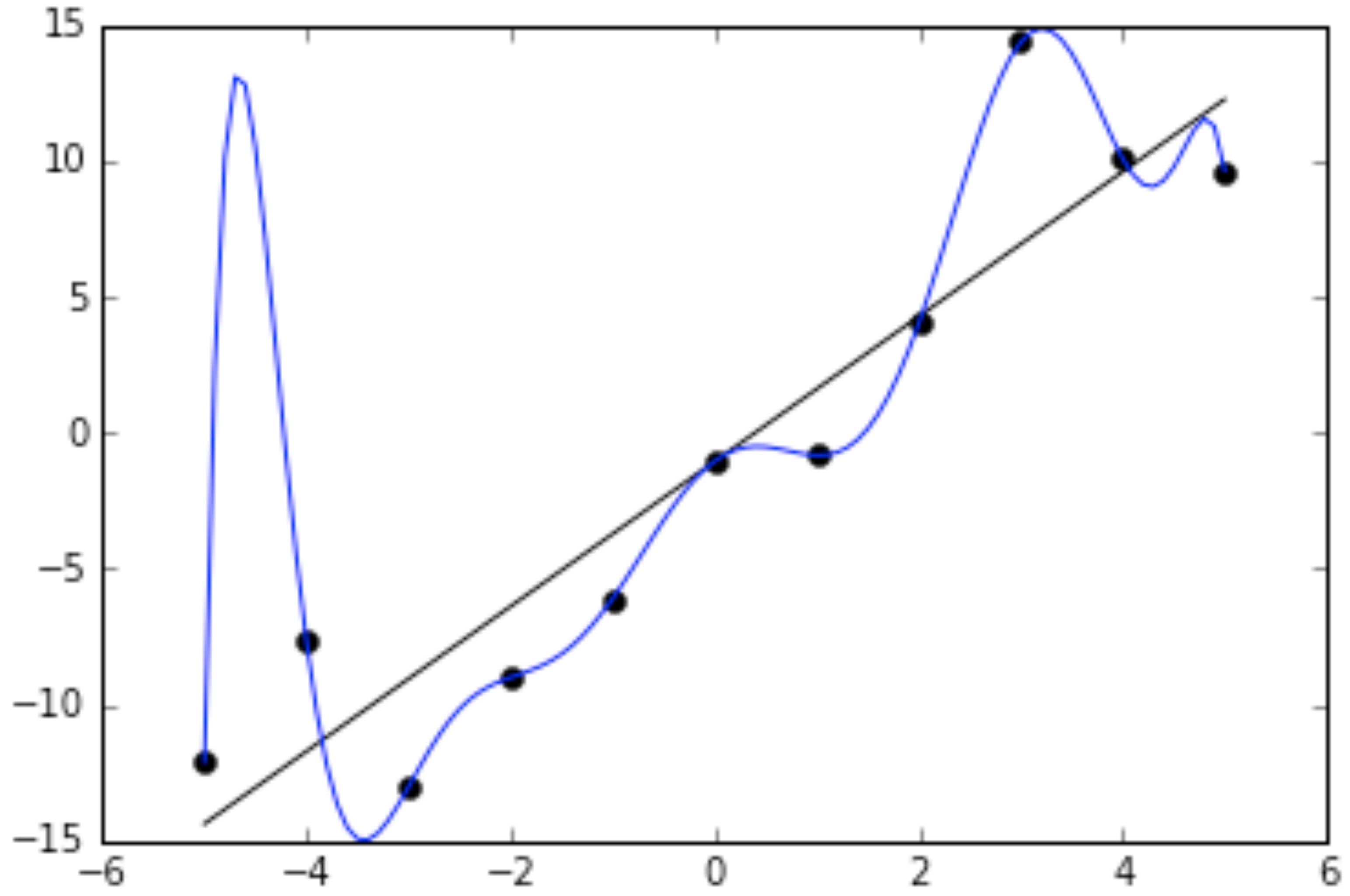
Figure 1: A modernized version of the neural probabilistic language model of Bengio et al. (2003), which

source (NAACL '21)

Additional Training Notes: Regularization and Hyper-Parameters

Overfitting

- Over-fitting: model too closely mimics the training data
  - Therefore, cannot *generalize* well
- Common when models are "over-parameterized"
  - E.g. fitting a high-degree polynomial
  - Neural models are typically over-parameterized
- Key questions:
  - How to detect overfitting?
  - How to prevent it?



Train, Dev, Test Set Splits

- Split total data into three chunks: train, dev (aka valid), test
  - Common: 70/15/15, 80/10/10%
- Train: used for individual model training, as we've seen so far
- Dev/valid:
  - Evaluation during training
  - Hyper-parameter tuning
  - Model selection
- Test:
  - Final evaluation; DO NOT TOUCH otherwise

Early stopping

source

Early stopping

- One: Pick # of epochs, hope for no overfitting

source

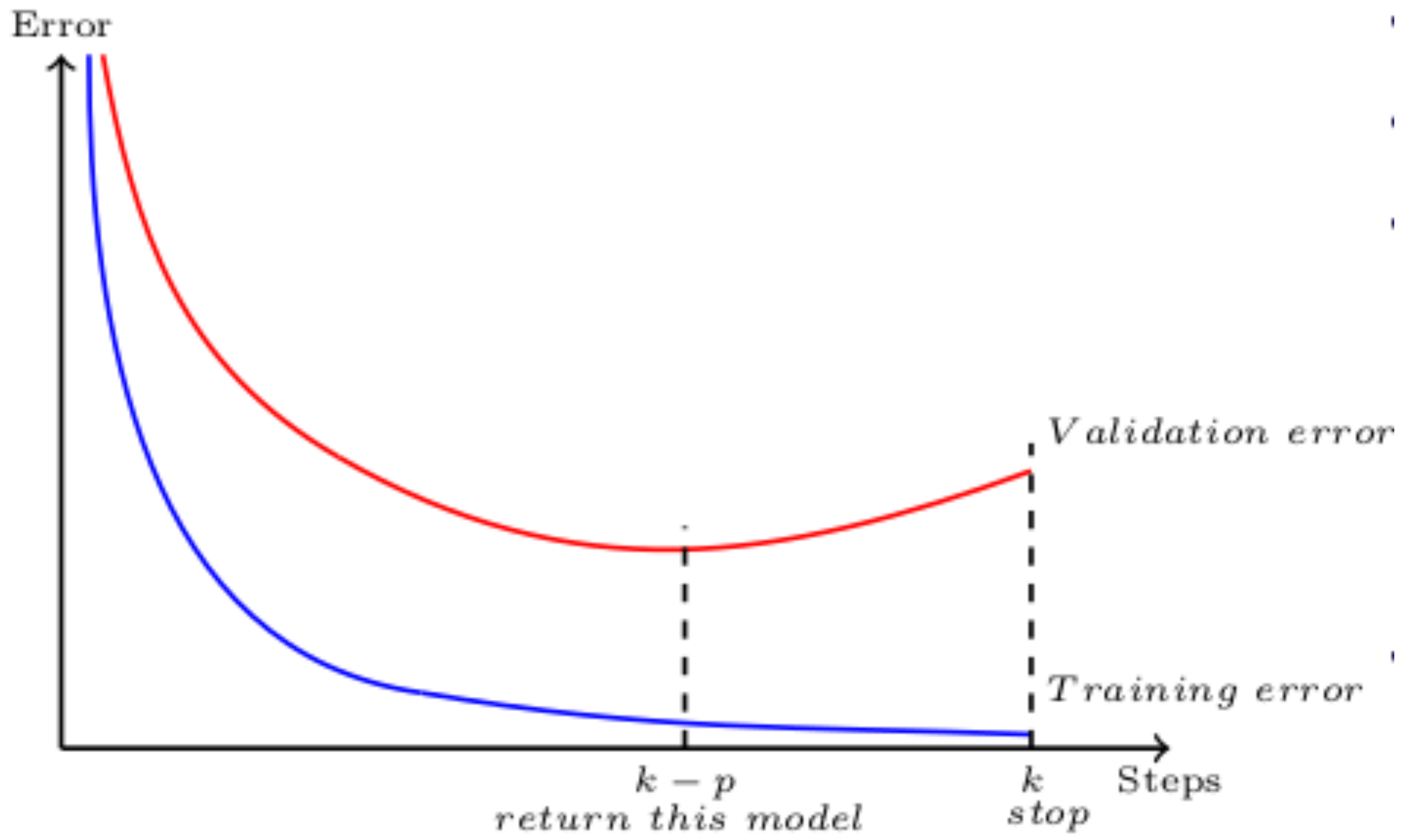
Early stopping

- One: Pick # of epochs, hope for no overfitting
- Better: pick max # of epochs, and "patience"
  - Halt when validation error does not improve over patience-many epochs

source

Early stopping

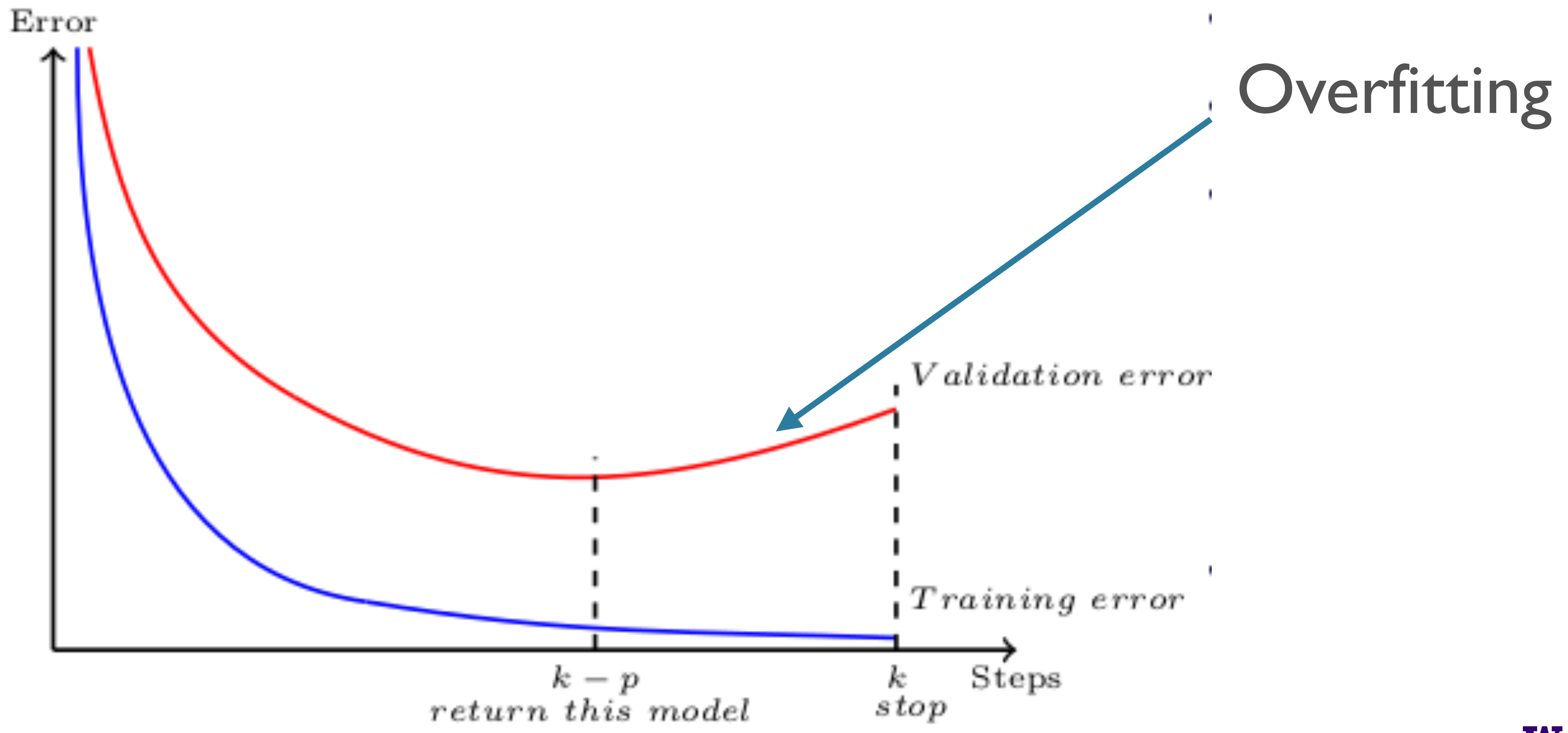
- One: Pick # of epochs, hope for no overfitting
- Better: pick max # of epochs, and "patience"
  - Halt when validation error does not improve over patience-many epochs



[source]

Early stopping

- One: Pick # of epochs, hope for no overfitting
- Better: pick max # of epochs, and "patience"
  - Halt when validation error does not improve over patience-many epochs



source

Regularization

- NNs are often *overparameterized*, so regularization helps
- L1/L2: $\mathscr{L}'(\theta, y) = \mathscr{L}(\theta, y) + \lambda \|\theta\|^2$
- Dropout:
  - *During training*, randomly turn off X% of neurons in each layer
  - (Don't do this during testing/predicting)
- Batch Normalization / Layer Norm
- NB: batch size 🤯

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Hyper-parameters

- In addition to the model architecture ones mentioned earlier
- Optimizer: SGD, Adam, Adagrad, RMSProp, ….
  - Optimizer-specific hyper-parameters: learning rate, alpha, beta, …
  - NB: backprop computes gradients; optimizer uses them to update parameters
- Regularization: L1/L2, Dropout, BN, …
  - regularizer-specific ones: e.g. dropout rate
- Batch size
- Number of epochs to train for
  - Early stopping criterion (e.g. patience)

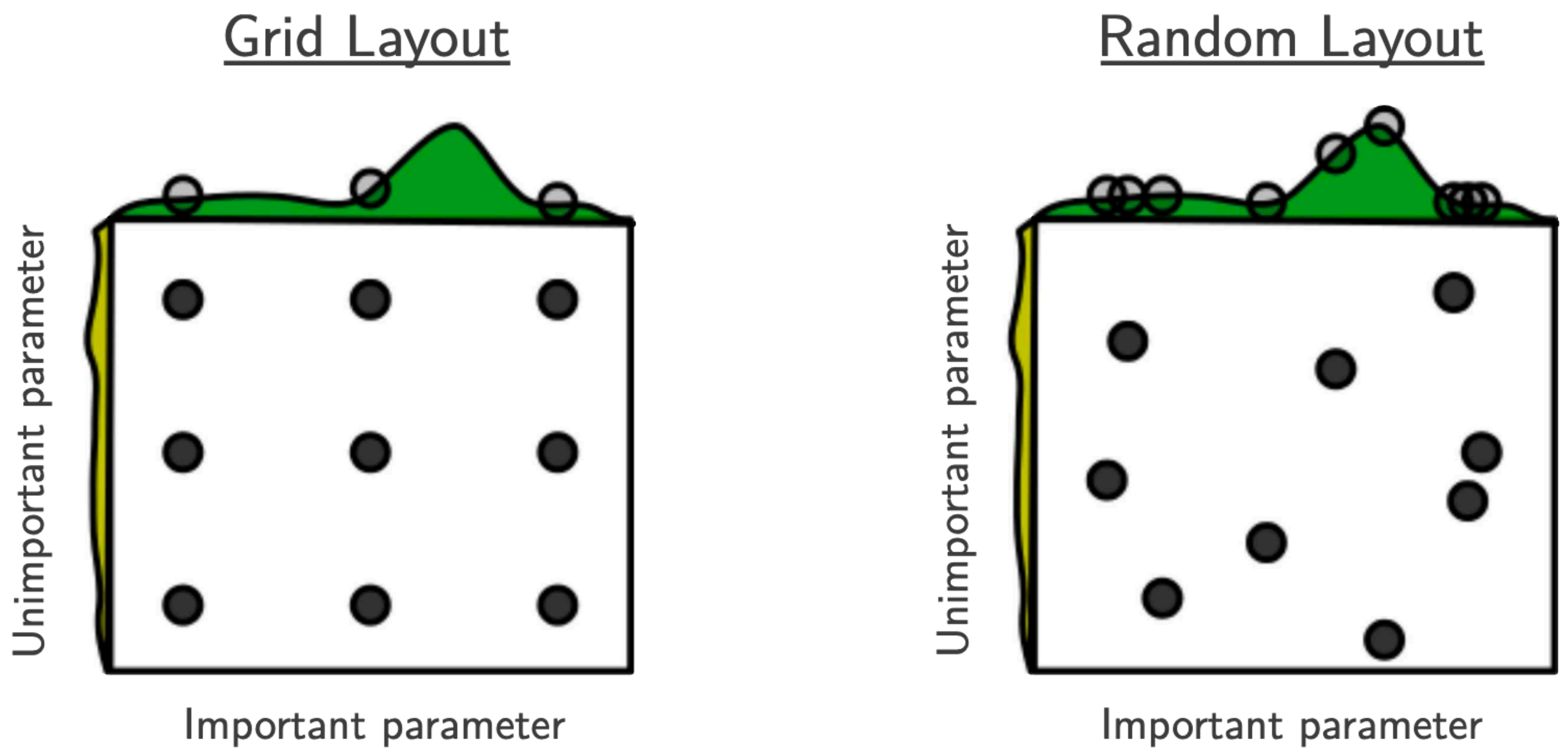
A note on hyper-parameter tuning

- Grid search: specify range of values for each hyper-parameter, try all possible combinations thereof
- Random search: specify possible values for all parameters, randomly sample values for each, stop when some criterion is met

[Bergstra and Bengio 2012]

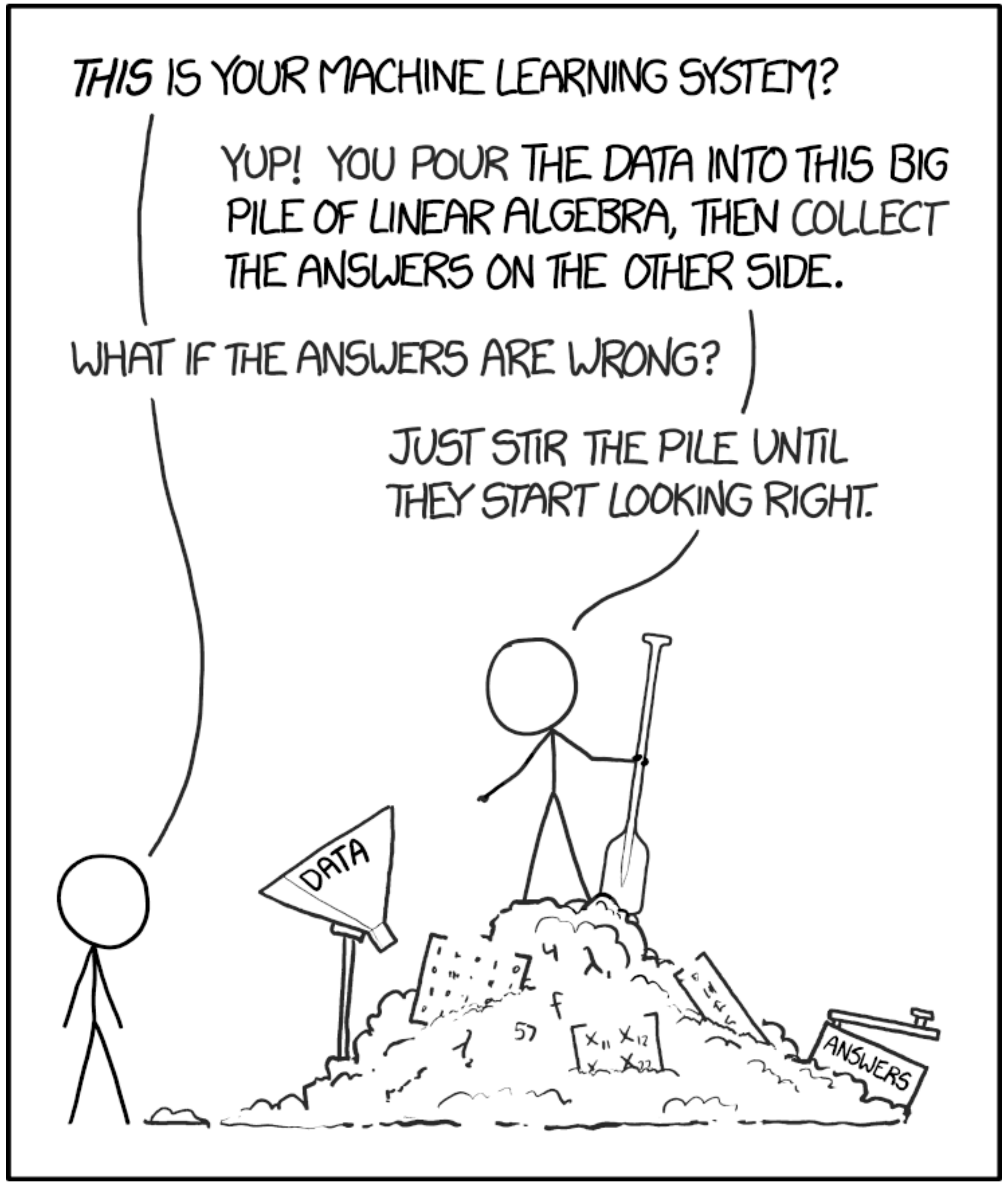
A note on hyper-parameter tuning

- Grid search: specify range of values for each hyper-parameter, try all possible combinations thereof
- Random search: specify possible values for all parameters, randomly sample values for each, stop when some criterion is met



Bergstra and Bengio 2012

Craft/Art of Deep Learning



https://xkcd.com/1838/

Some Practical Pointers

- Hyper-parameter tuning and the like are not the focus of this course
- For some helpful hand-on advice about training NNs from scratch, debugging under “silent failures”, etc:
  - http://karpathy.github.io/2019/04/25/recipe/

Hyper-parameter Tuning

h/t CM Downey