Neural Language Models

CMSC 473/673 - NATURAL LANGUAGE PROCESSING

Slides modified from Dr. Frank Ferraro

Learning Objectives

Define the basic architecture of a neural network

Distinguish between count-based, logistic regression, and neural LMs

Review: Add-λ estimation

Other names: Laplace smoothing, Lidstone smoothing

Pretend we saw each word λ more times than we did

Add λ to all the counts

$$p(\mathrm{z}) \cong count(\mathrm{z}) + \lambda$$

$$= \frac{count(\mathrm{z}) + \lambda}{\sum_v (count(\mathrm{v}) + \lambda)}$$

Review: An Extended Trigram Example

The film got a great opening and the film went on to become a hit .

Context: x y	Word (Type): z	Raw Count	Add-1 count	Norm.	Probability p(z | x y)
The film	The	0	1	17 (=1+16*1)	1/17
The film	film	0	1		1/17
The film	got	1	2		2/17
The film	went	0	1		1/17
…					…
The film	OOV	0	1		1/17
The film	EOS	0	1		1/17
…					
a great	great	0	1	17	1/17
a great	opening	1	2		2/17
a great	and	0	1		1/17
a great	the	0	1		1/17
…					

Review: Language Model with Maxent n-grams

$$p_n(\text{🖹}|y) = \prod_{i=1}^{M} \text{maxent}(\overset{\text{label}}{\overset{\downarrow}{y}}, \underbrace{x_{i-n+1:i-1}, x_i}_{\text{n-gram}})$$

$$= \prod_{i=1}^{M} \frac{\exp(\theta_{x_i}^T f(y, x_{i-n+1:i-1}))}{\sum_{x'} \exp(\theta_{x'}^T f(y, x_{i-n+1:i-1}))}$$

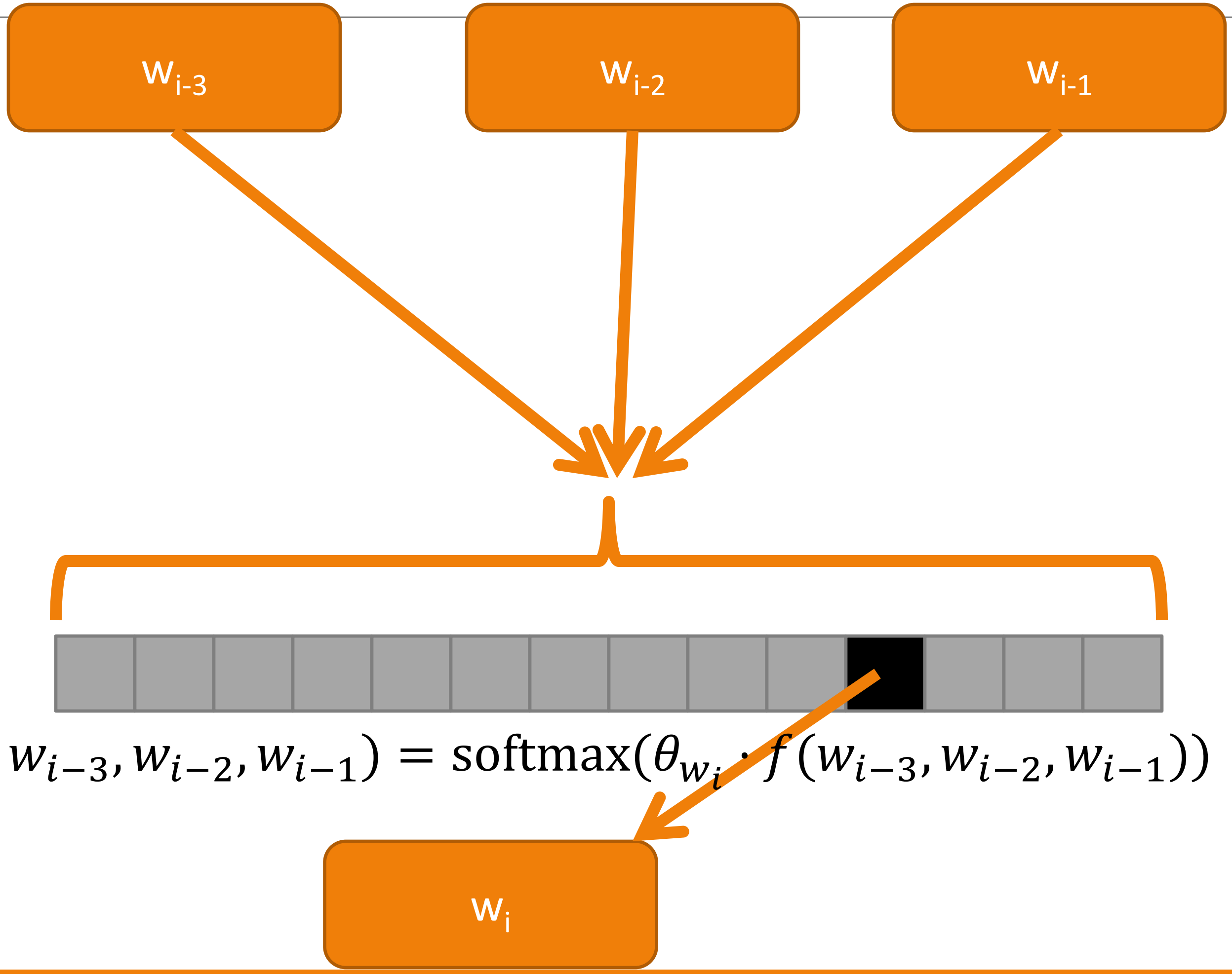
Iterate through all possible output vocab types x'---just like in count-based LMs

Review: Maxent Language Models

given some context…

compute beliefs about what is likely…

predict the next word



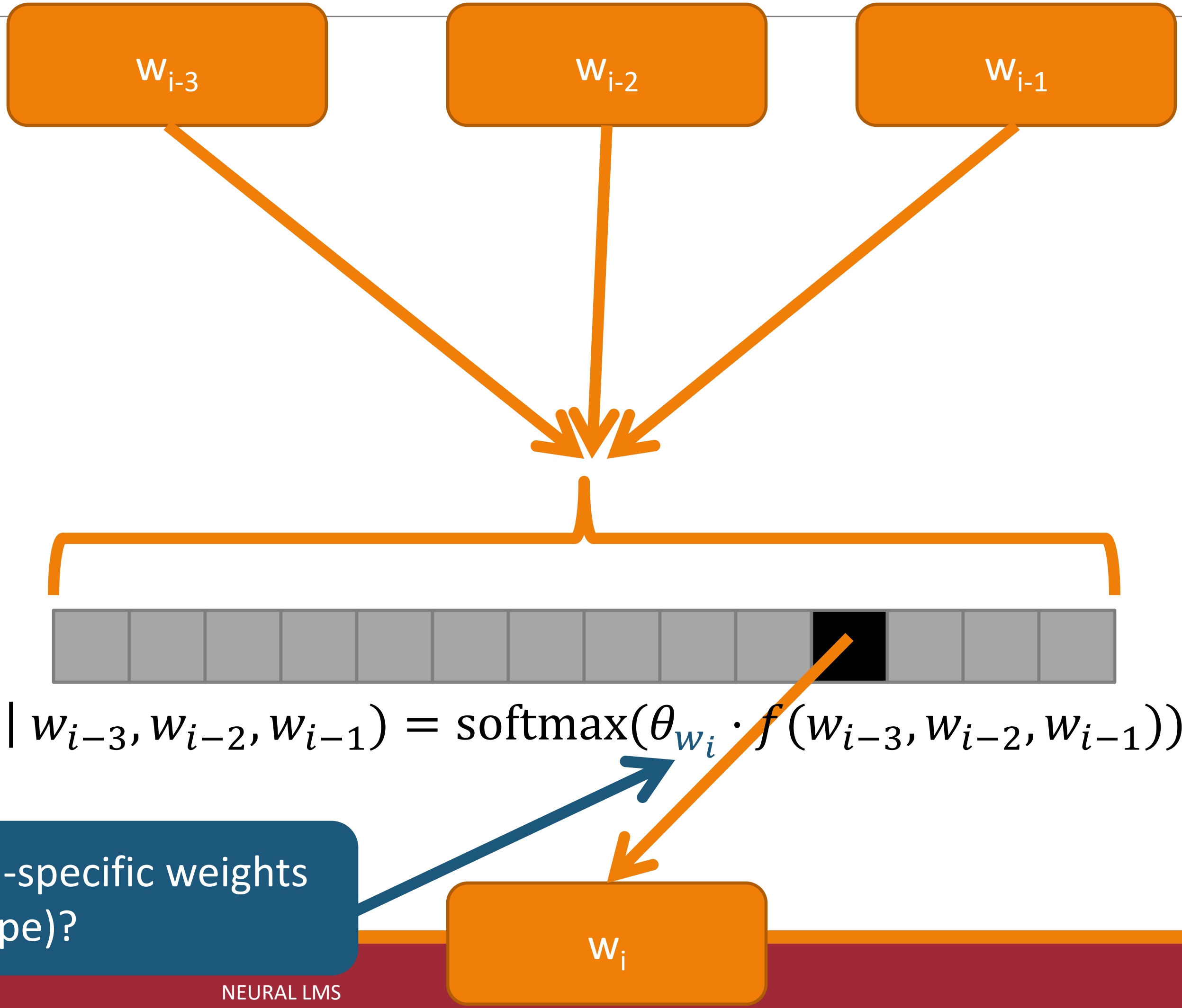
$$p(w_i \mid w_{i-3}, w_{i-2}, w_{i-1}) = \text{softmax}(\theta_{w_i} \cdot f(w_{i-3}, w_{i-2}, w_{i-1}))$$

Maxent Language Models

given some context…

compute beliefs about what is likely…

predict the next word



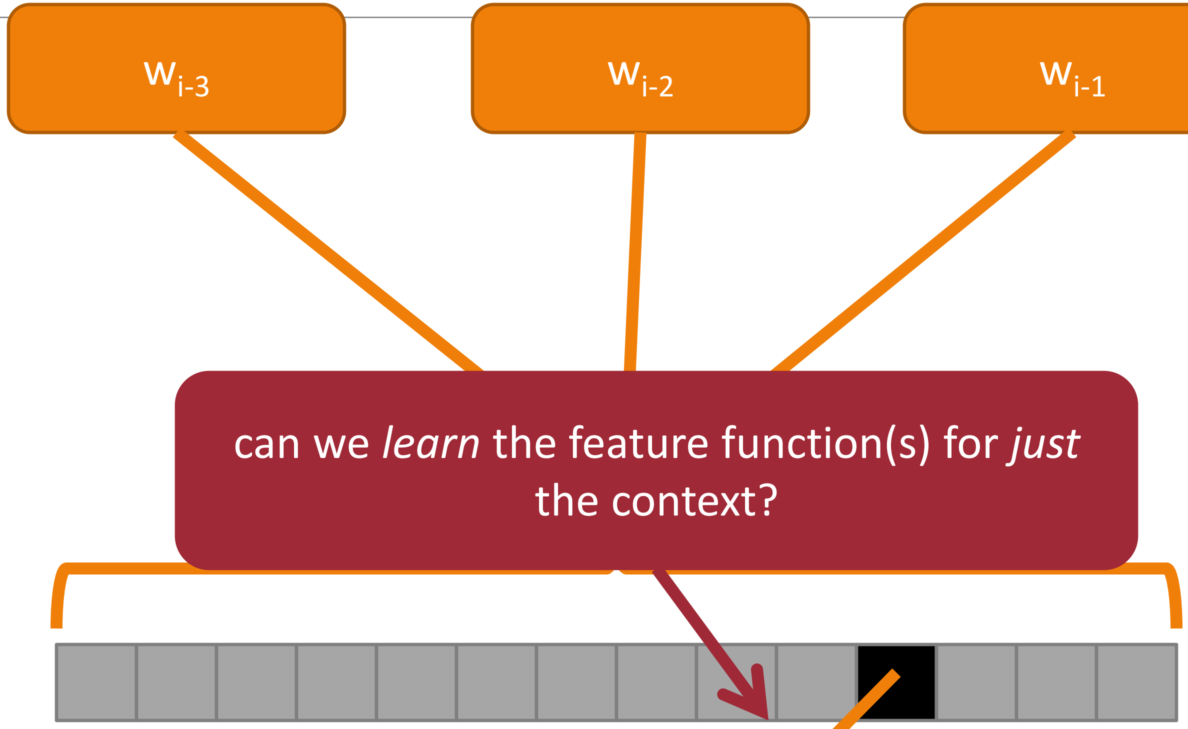
$$p(w_i \mid w_{i-3}, w_{i-2}, w_{i-1}) = \text{softmax}(\theta_{w_i} \cdot f(w_{i-3}, w_{i-2}, w_{i-1}))$$

Neural Language Models

given some context…

compute beliefs about what is likely…

predict the next word



$$p(w_i \mid w_{i-3}, w_{i-2}, w_{i-1}) = \text{softmax}(\theta_{w_i} \cdot f(w_{i-3}, w_{i-2}, w_{i-1}))$$

can we learn word-specific weights (by type)?

w_i

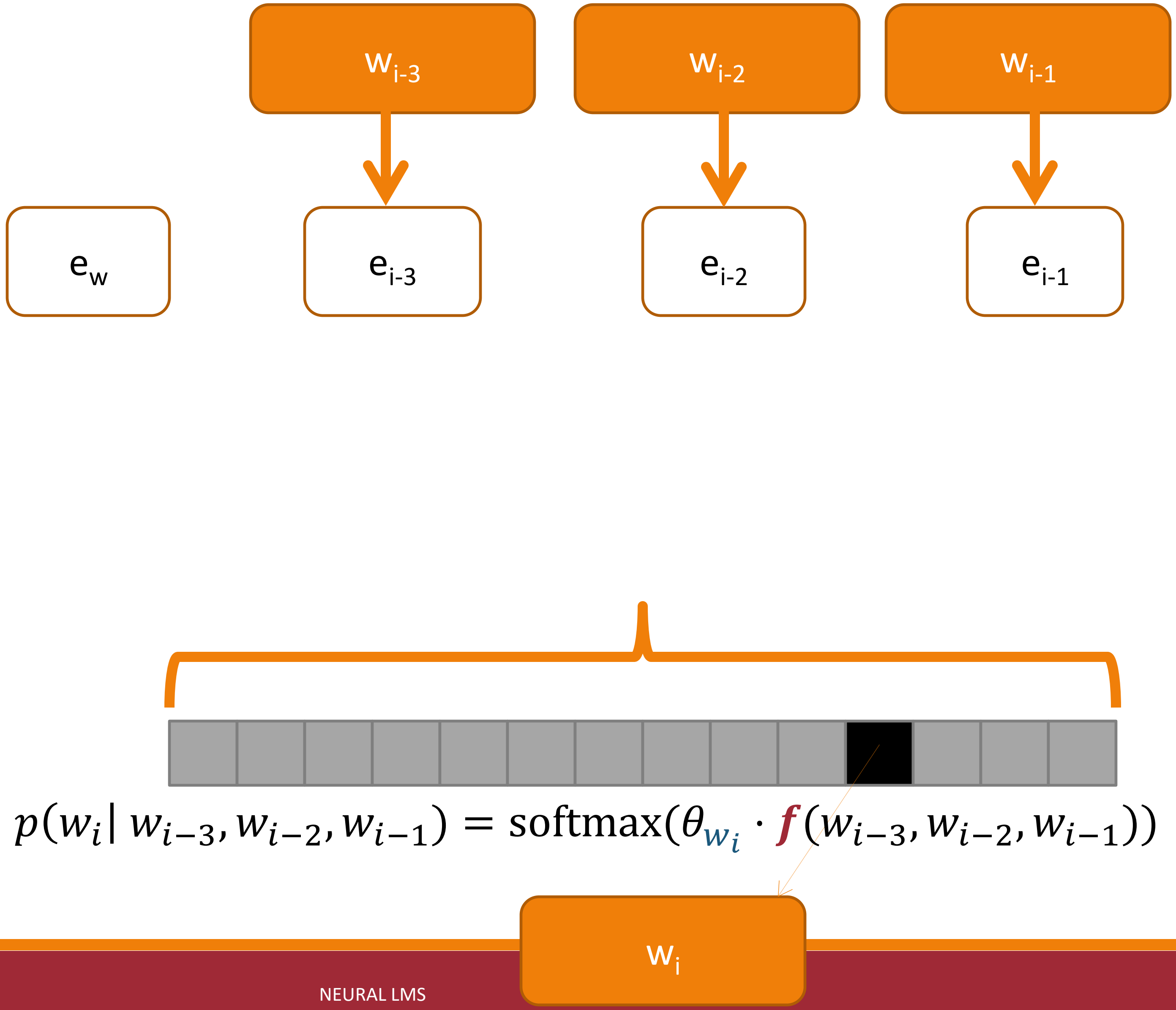
Neural Language Models

given some context…

create/use "distributed representations"…

compute beliefs about what is likely…

predict the next word



$$p(w_i \mid w_{i-3}, w_{i-2}, w_{i-1}) = \text{softmax}(\theta_{w_i} \cdot \boldsymbol{f}(w_{i-3}, w_{i-2}, w_{i-1}))$$

Neural Language Models

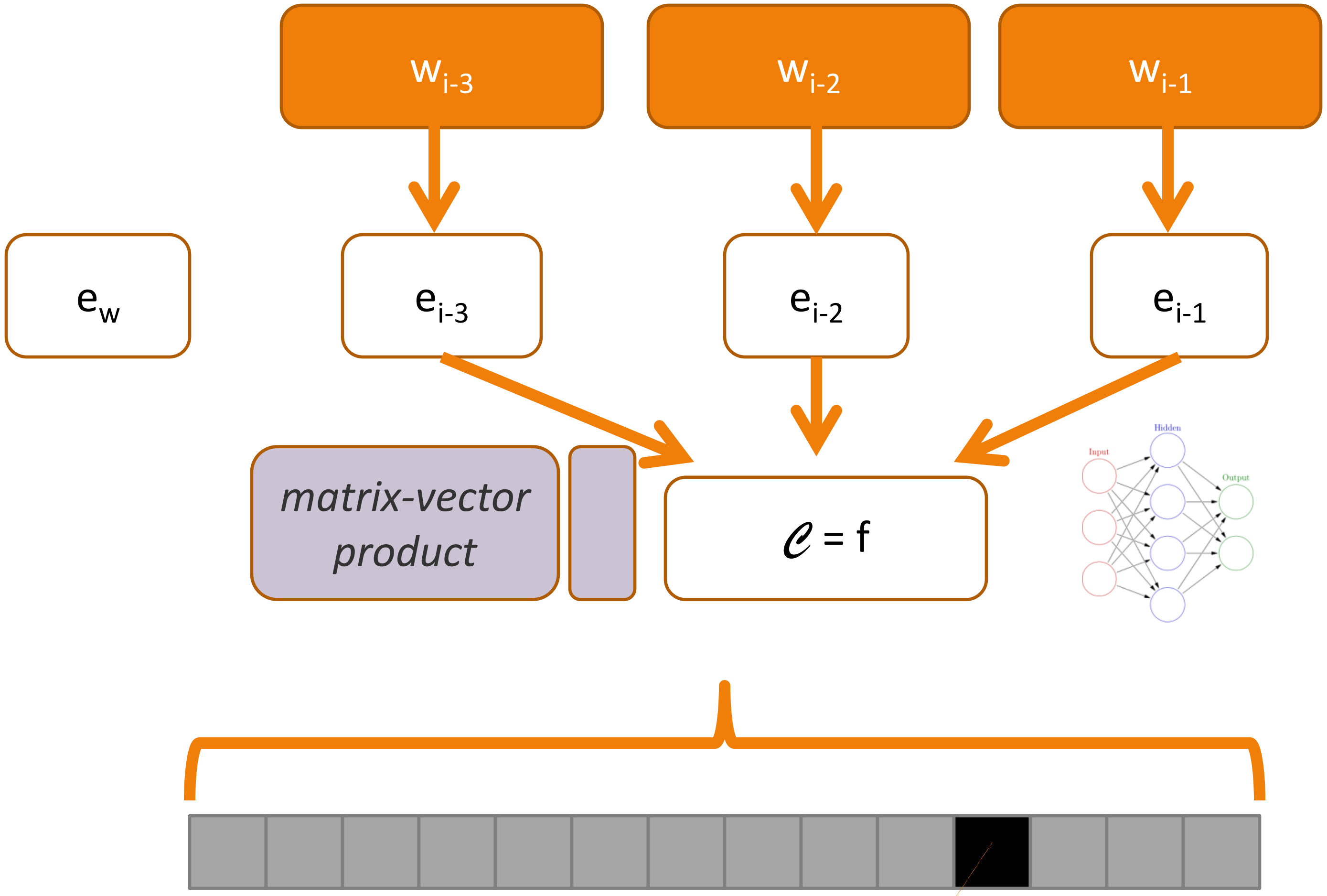
given some context…

create/use "distributed representations"…

combine these representations…

compute beliefs about what is likely…

predict the next word



$$p(w_i \mid w_{i-3}, w_{i-2}, w_{i-1}) = \text{softmax}(\theta_{w_i} \cdot \boldsymbol{f}(w_{i-3}, w_{i-2}, w_{i-1}))$$

Neural Language Models

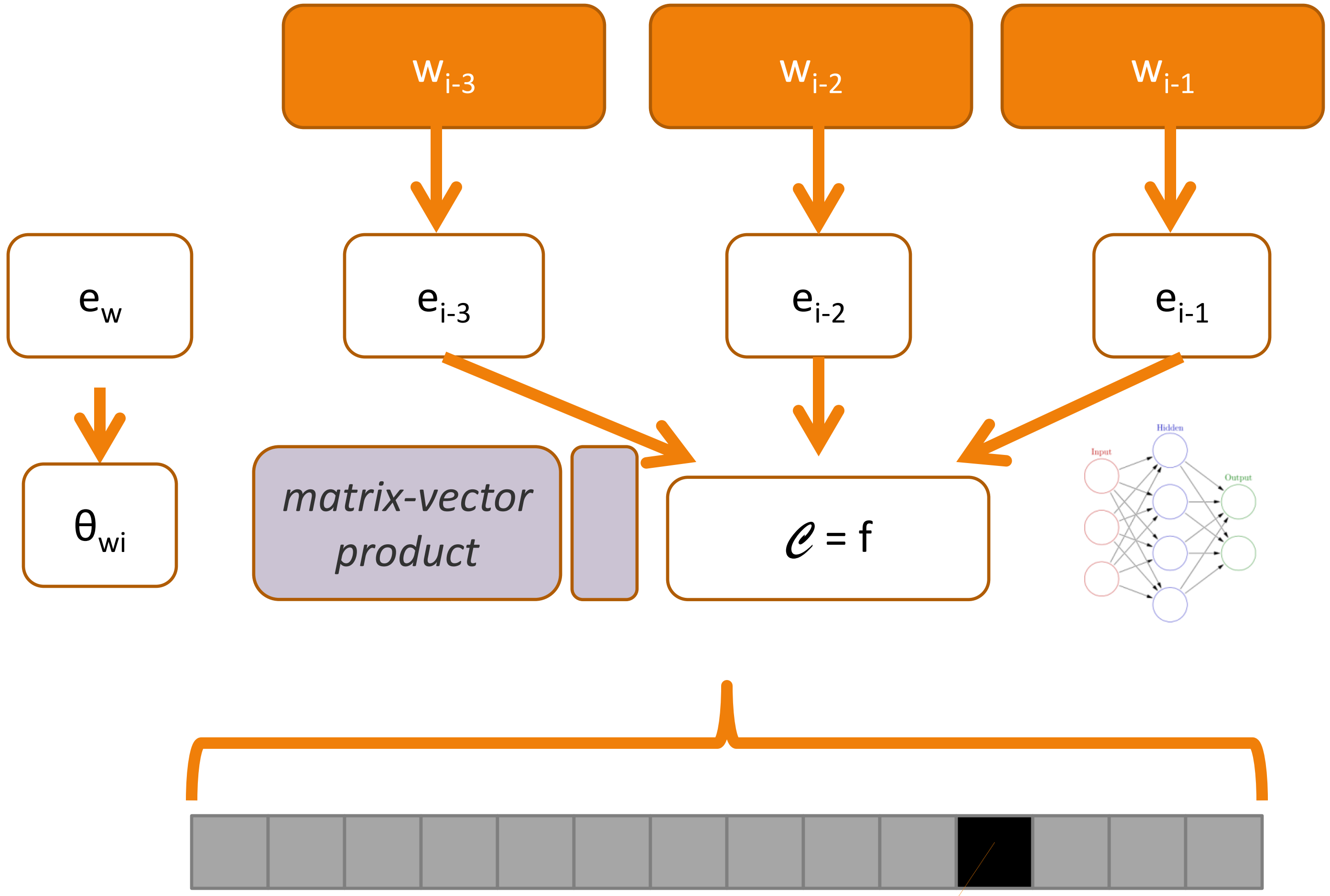
given some context…

create/use “distributed representations”…

combine these representations…

compute beliefs about what is likely…

predict the next word



$$p(w_i |\, w_{i-3}, w_{i-2}, w_{i-1}) = \text{softmax}(\theta_{w_i} \cdot \boldsymbol{f}(w_{i-3}, w_{i-2}, w_{i-1}))$$

w_i

Neural Language Models

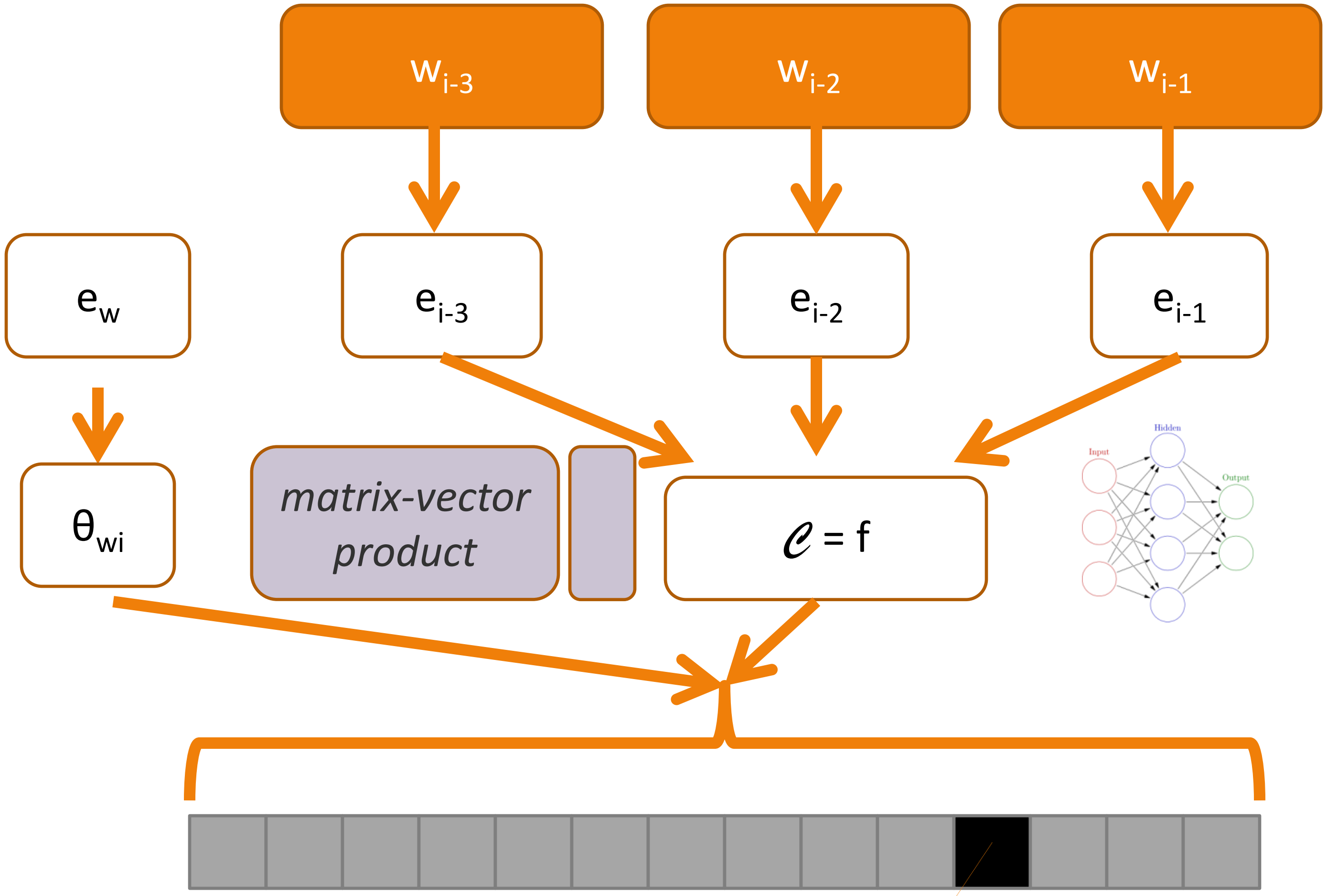
given some context…

create/use “distributed representations”…

combine these representations…

compute beliefs about what is likely…

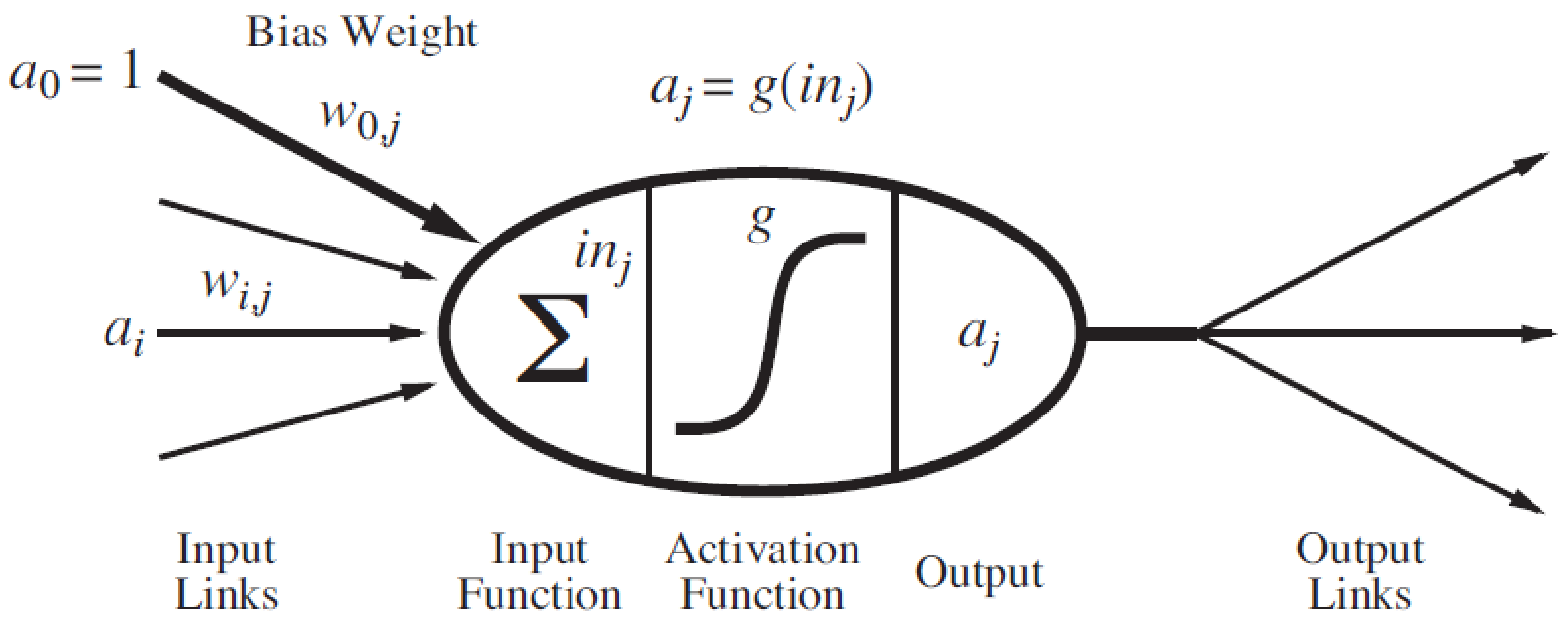
predict the next word



$$p(w_i | w_{i-3}, w_{i-2}, w_{i-1}) = \text{softmax}(\theta_{w_i} \cdot \boldsymbol{f}(w_{i-3}, w_{i-2}, w_{i-1}))$$

Biologically-Inspired Learning Models: Neuron Unit

$$in_j = w_{0j} + w_{1j}a_1 + w_{2j}a_2 + \cdots + w_{ij}a_i$$

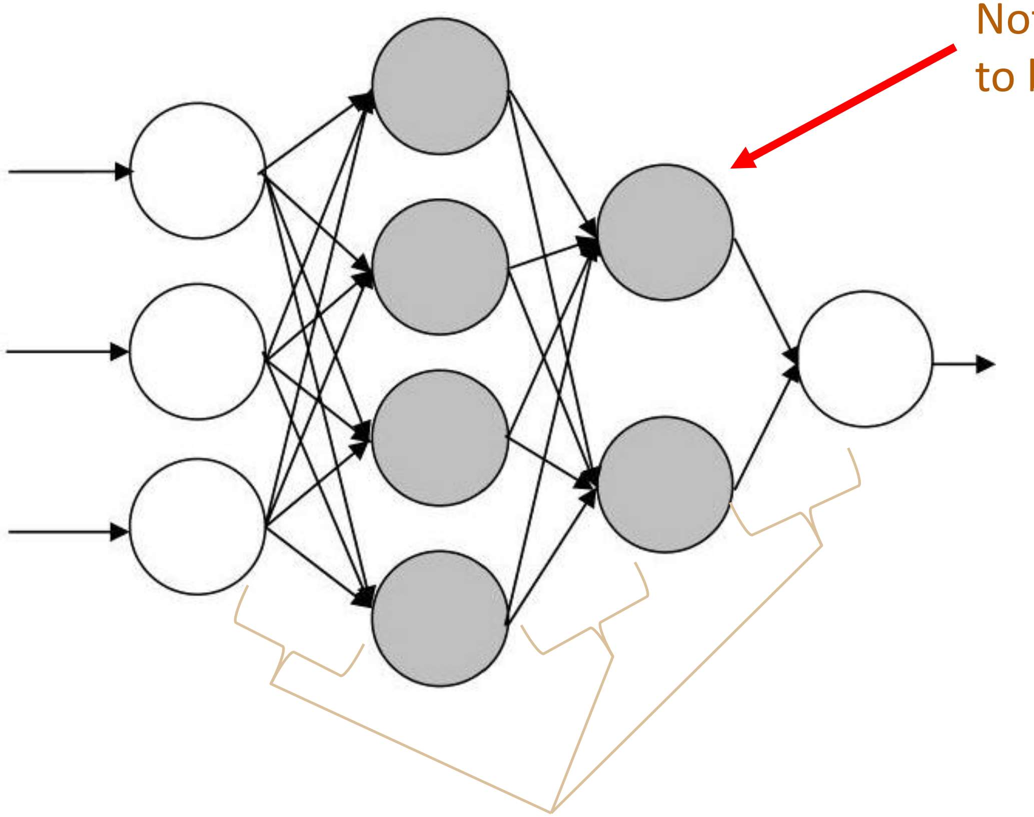


activations
$0 \leq a_i \leq 1$

weights
$-\infty < w_{ij} < \infty$

Multi-layer Networks: General Structure Example

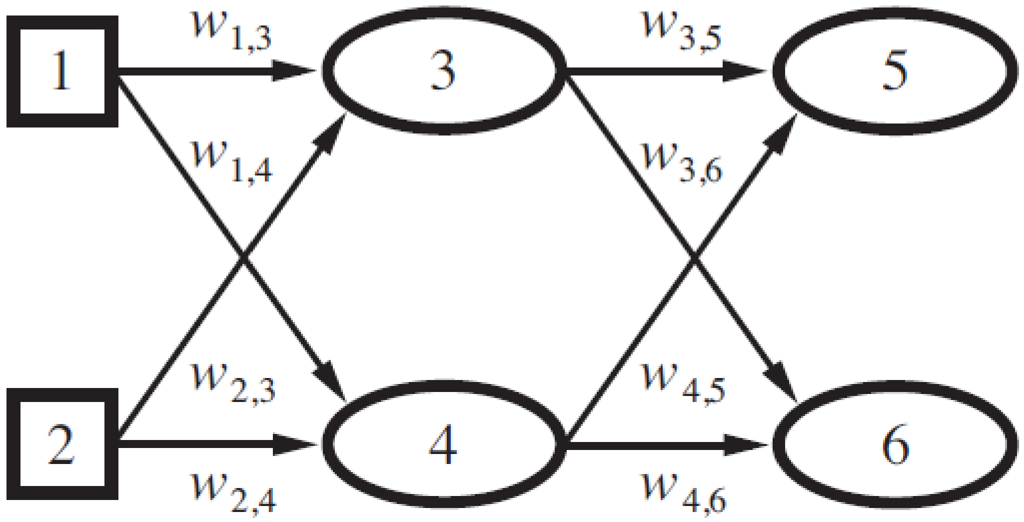
Note that the layers don't have to be the same size



Fully connected

Multi-layer Networks: General Structure

Multi-layer perceptrons (aka neural networks) will have **inputs**, one or more **hidden layers**, and an **output layer:**



Multi-layer Networks: General Structure

Multi-layer perceptrons (aka neural networks) will have **inputs**, one or more **hidden layers**, and an **output layer:**

Number of inputs, outputs, and number and size of hidden layers can vary

Combination of **different weights** and **different structures** represent different **functions**

We will treat each layer as **fully-connected**

- Each unit in one layer connects to every unit in the next layer

Computing Values: Forward Propagation

Forward propagation calculates the output values for a given set of input values

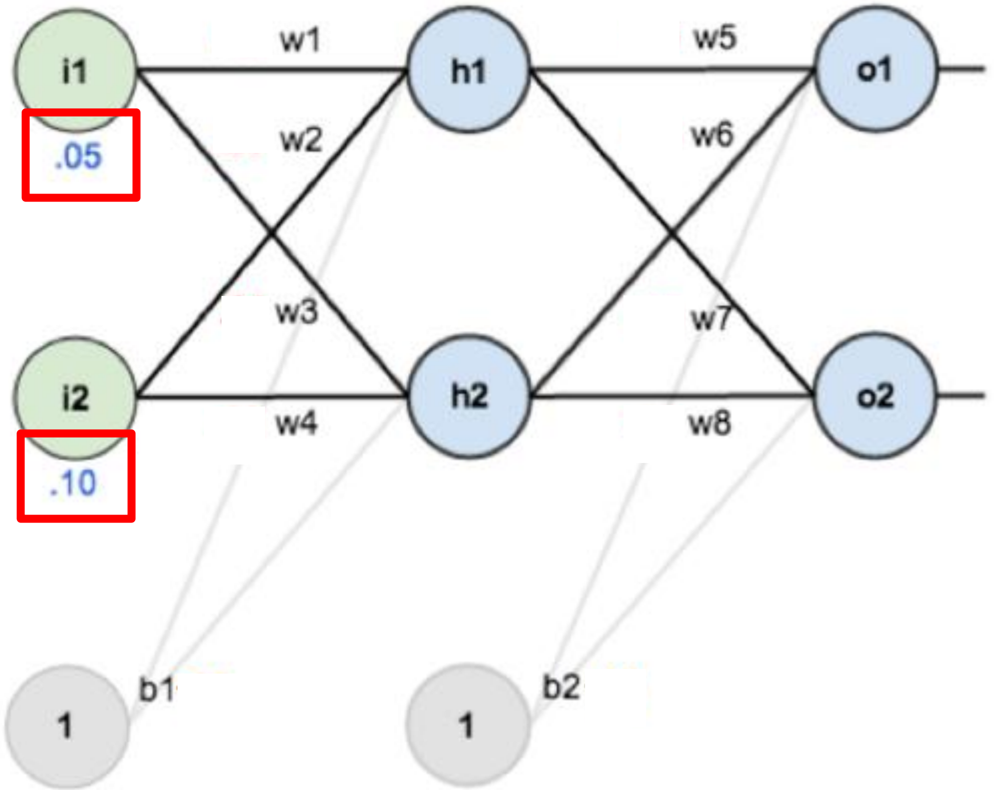
Algorithm

For each layer:

1. Calculate the weighted sum of inputs to each neuron unit
2. Evaluate the activation function to determine the output of each neuron unit
3. Use outputs as inputs for the next layer

Forward Propagation Example

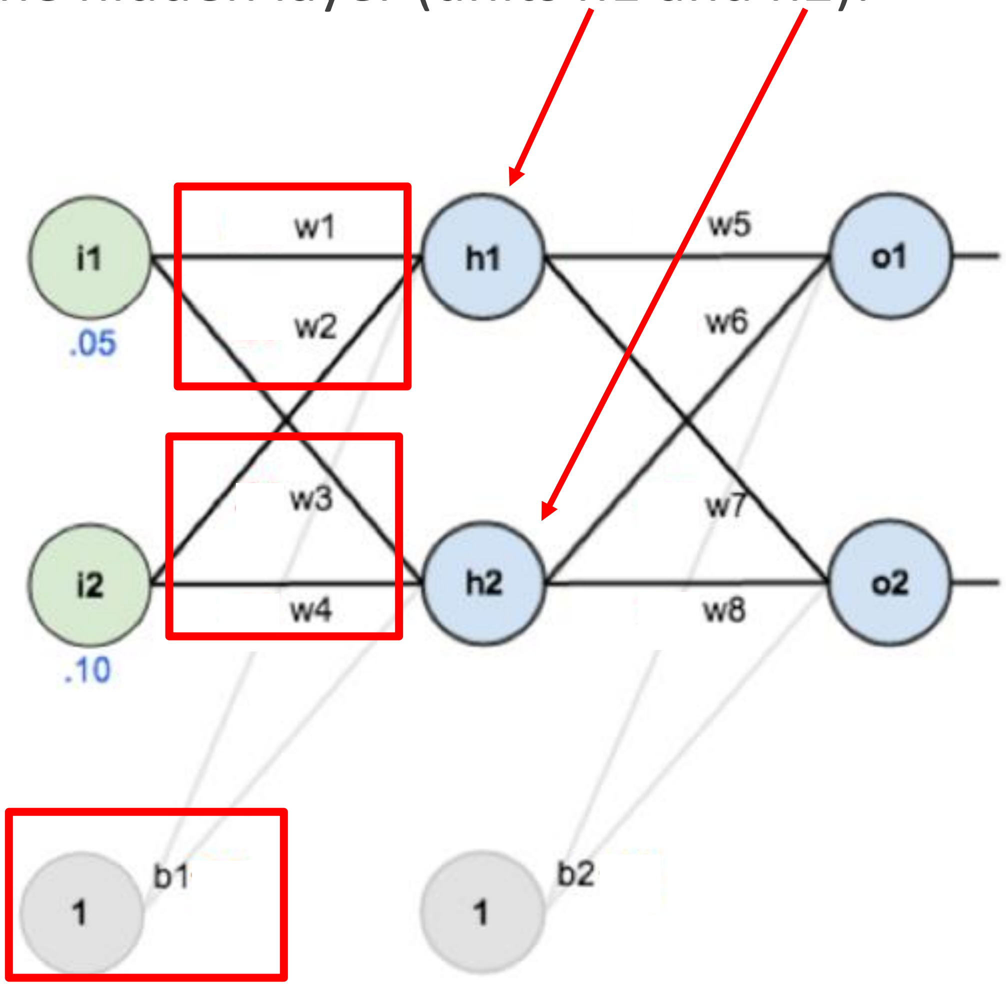
Calculate the output of the network below, assuming each neuron uses a sigmoid activation function, given 0.05 and 0.1 as inputs.



For each layer:
1. Calculate the weighted sum of inputs to each neuron unit
2. Evaluate the activation function to determine the output of each neuron unit
3. Use outputs as inputs for the next layer

Forward Propagation Example

Calculate inputs to the hidden layer (units h1 and h2):



$$\begin{aligned} in_{h1} &= w_1 i_1 + w_2 i_2 + b_1 \\ &= .15(.05)+.2(.1)-.35 \\ &= .0075+.02-.35 \\ &= -.3225 \end{aligned}$$

$$\begin{aligned} in_{h2} &= w_3 i_1 + w_4 i_2 + b_2 \\ &= .25(.05)+.3(.1)-.35 \\ &= .0125+.03-.35 \\ &= -.3075 \end{aligned}$$

For each layer:

1. Calculate the weighted sum of inputs to each neuron unit
2. Evaluate the activation function to determine the output of each neuron unit
3. Use outputs as inputs for the next layer

Forward Propagation Example

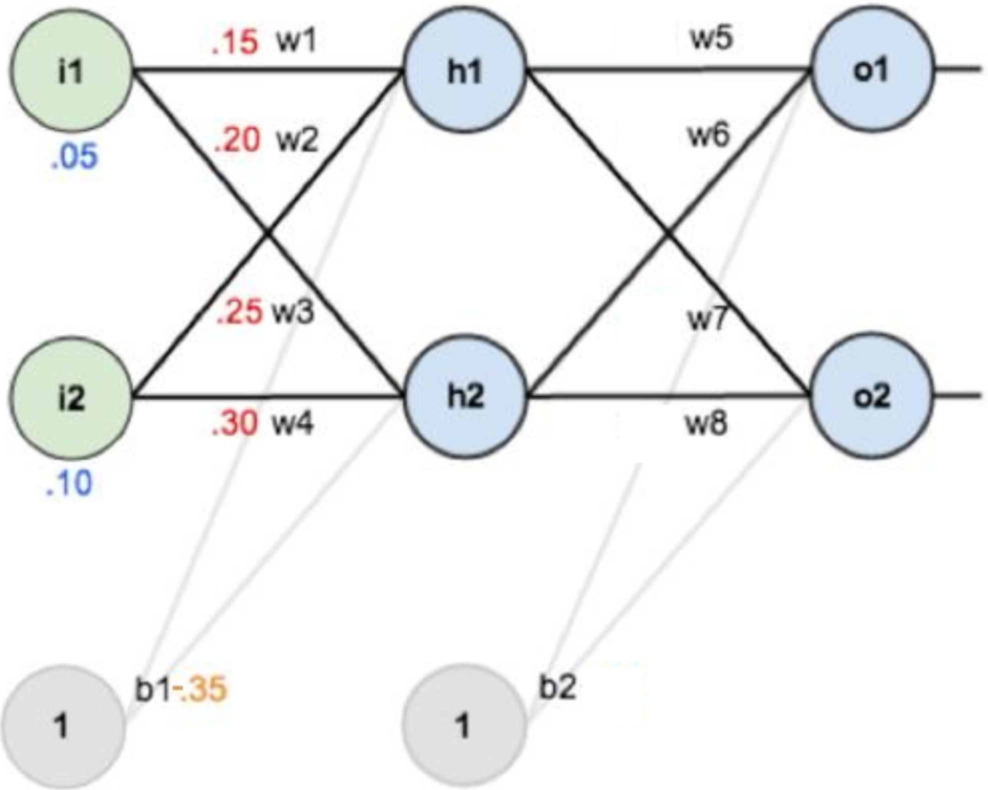
Calculate <u>outputs</u> to the hidden layer (units h1 and h2):

How do we do this?
Use our activation function!

$$g(x) = \frac{1}{1 + e^{-x}}$$

What will be our *x*?

in_{h1} =–.3225
in_{h2} = –.3075



$out_{h1} = g(in_{h1})$
$= \frac{1}{1+e^{-in_{h1}}}$
$= \frac{1}{1+e^{-(-.3275)}}$
$= .4188$

$out_{h2} = g(in_{h2})$
$= \frac{1}{1+e^{-in_{h2}}}$
$= \frac{1}{1+e^{-(-.3075)}}$
$= .4237$

For each layer:
1. Calculate the weighted sum of inputs to each neuron unit
2. Evaluate the activation function to determine the output of each neuron unit
3. Use outputs as inputs for the next layer

How are Neural Networks used?

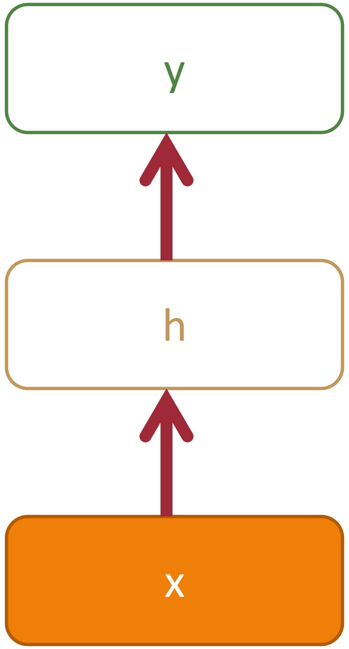
Are neural networks supervised or unsupervised learning?

- Inputs to the network are features of our data set
- Outputs to the network are our labels

Can they be used for classification or regression?

- Either!

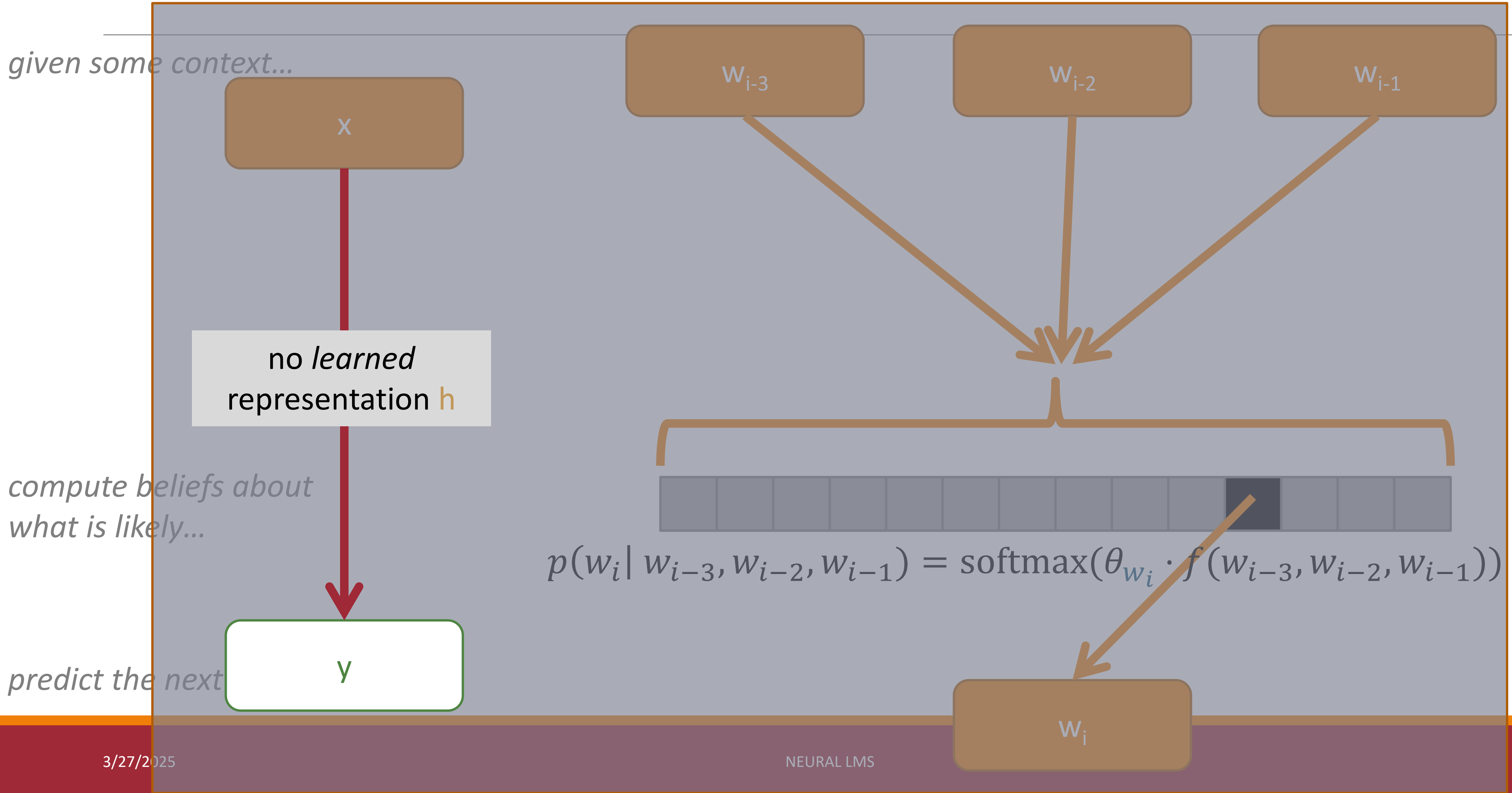
Network Types: Flat Input, Flat Output



1. Feed forward

Linearizable feature input
Bag-of-items classification/regression
Basic non-linear model

Maxent Language Models



$$p(w_i|\, w_{i-3}, w_{i-2}, w_{i-1}) = \text{softmax}(\theta_{w_i} \cdot f(w_{i-3}, w_{i-2}, w_{i-1}))$$

Neural Language Models

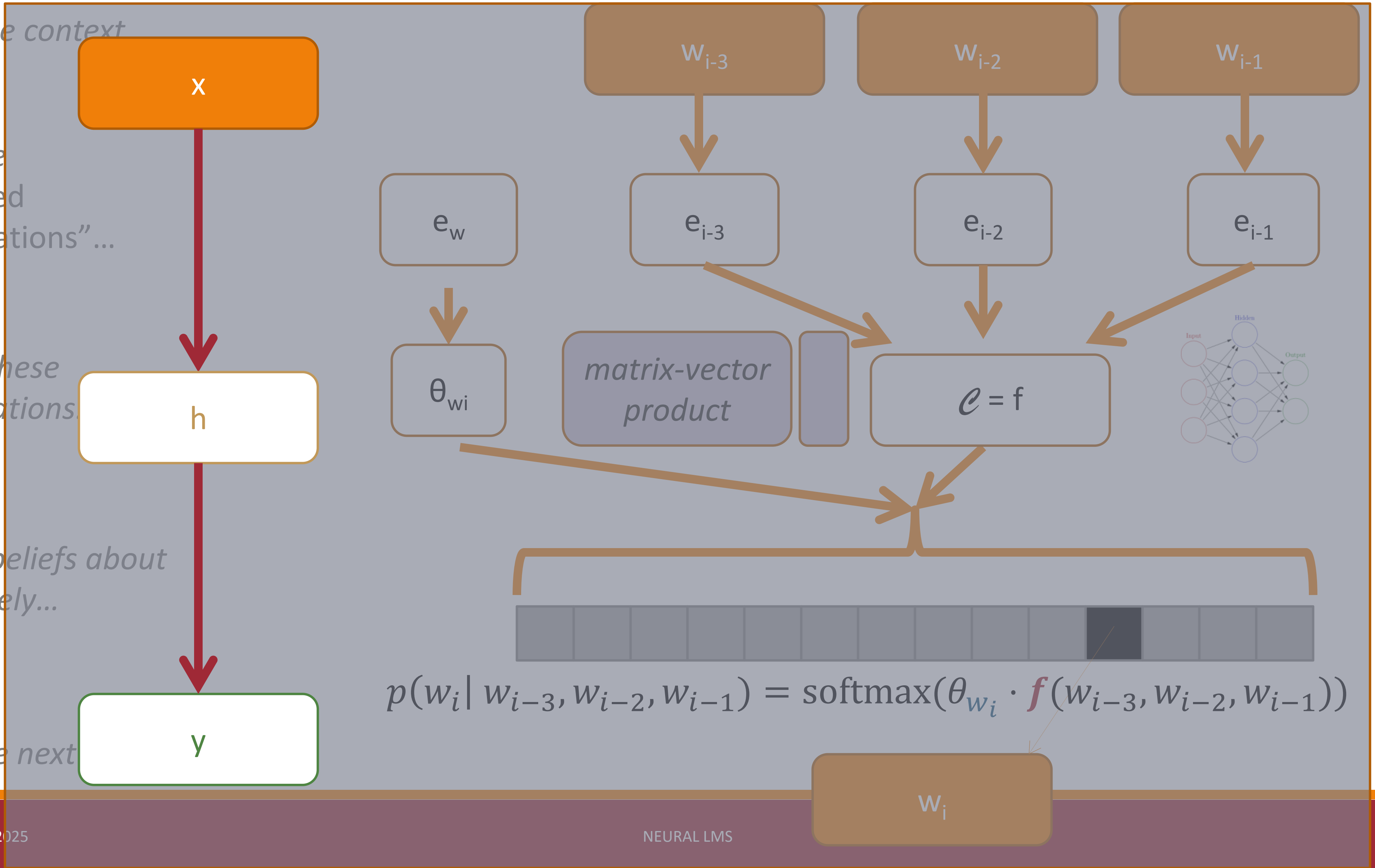
given some context…

create/use "distributed representations"…

combine these representations…

compute beliefs about what is likely…

predict the next…



Common Types of Flat Input, Flat Output

Feed forward networks

Multilayer perceptrons (MLPs)

General Formulation:

Input: x
Compute:

$h_0 = x$
for layer l = 1 to L:
$h_l = f_l(W_l h_{l-1} + b_l)$ linear layer

hidden state at layer l

(non-linear) activation function at l

return $\underset{y}{\operatorname{argmax}} \operatorname{softmax}(\theta h_L)$

In Pytorch (torch.nn):

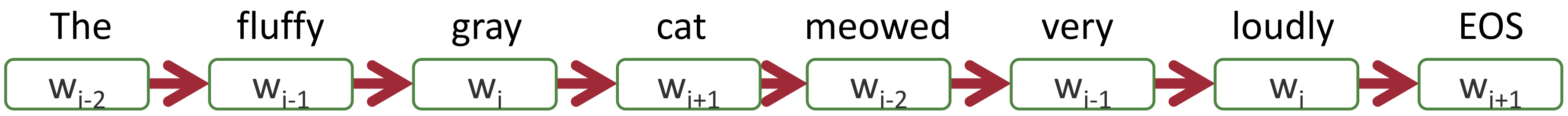
Activation functions:
https://pytorch.org/docs/stable/nn.html?highlight=activation#non-linear-activations-weighted-sum-nonlinearity

Linear layer:
https://pytorch.org/docs/stable/nn.html#linear-layers

```
torch.nn.Linear(
    in_features=<dim of h_{l-1} >,
    out_features=<dim of h_l >,
    bias=<Boolean: include bias b_l >)
```

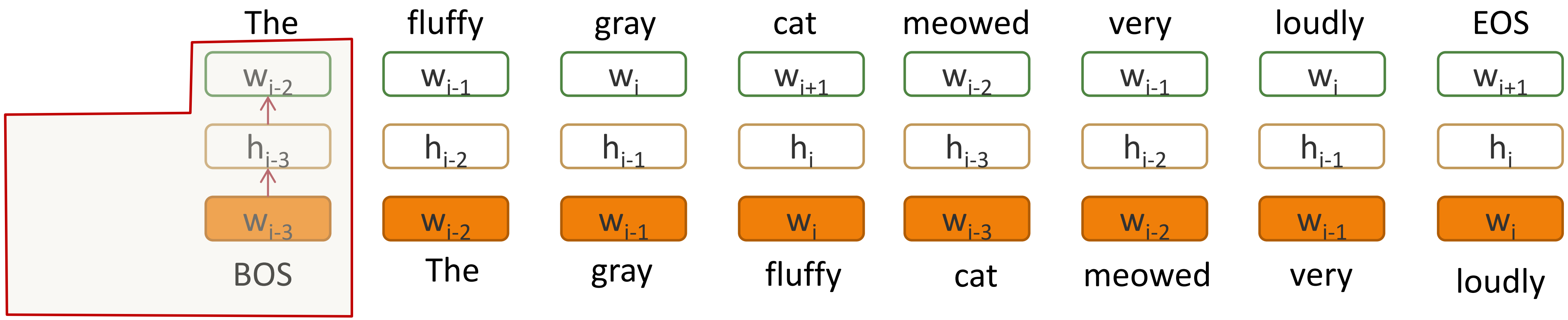
A Neural N-Gram Model

The fluffy gray cat meowed very loudly



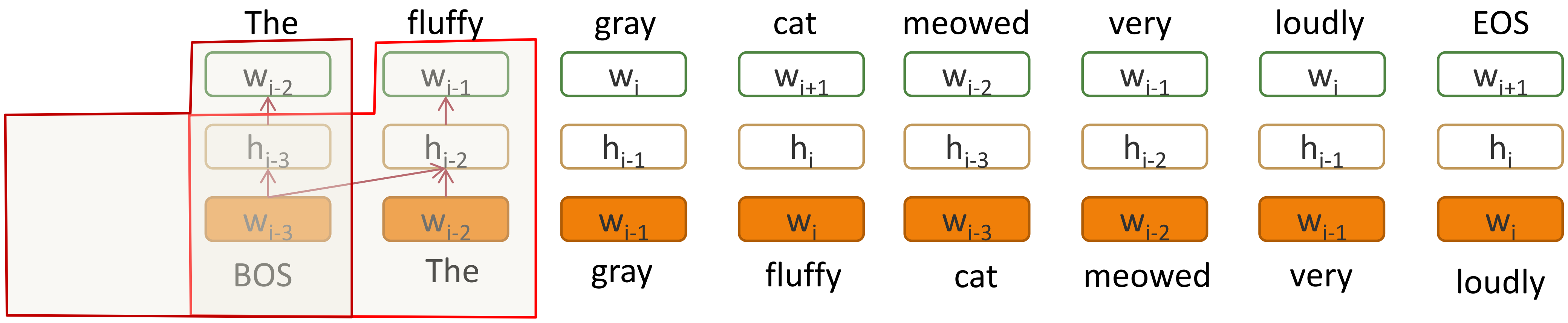
A Neural N-Gram Model (N=3)

The fluffy gray cat meowed very loudly



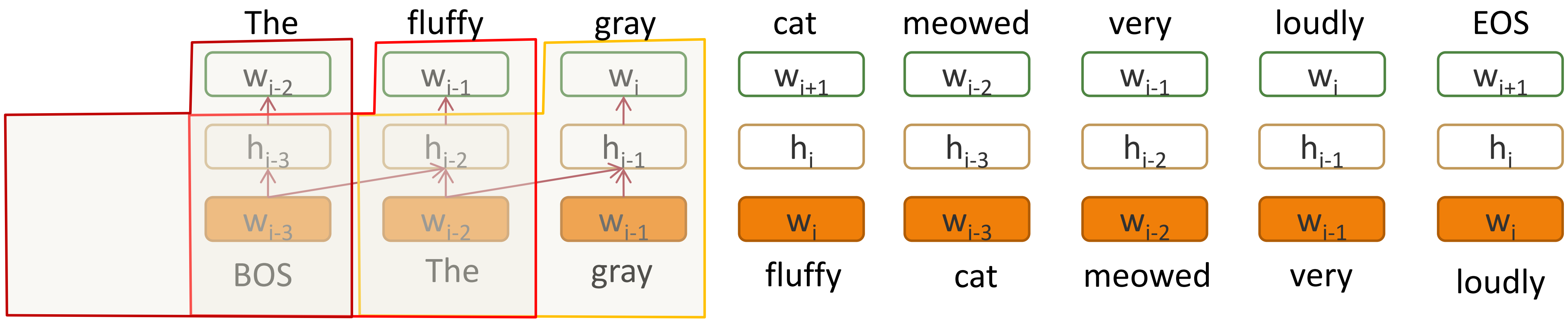
A Neural N-Gram Model (N=3)

The fluffy gray cat meowed very loudly



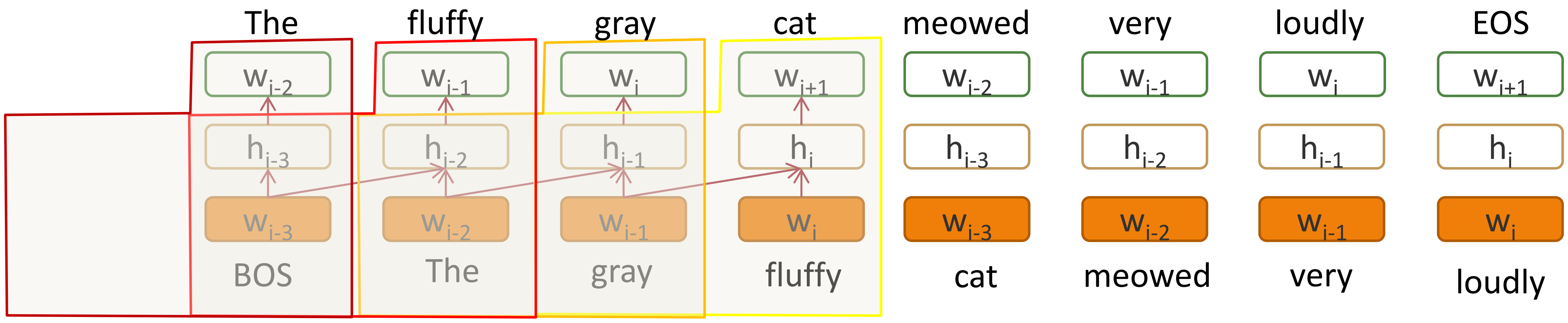
A Neural N-Gram Model (N=3)

The fluffy gray cat meowed very loudly



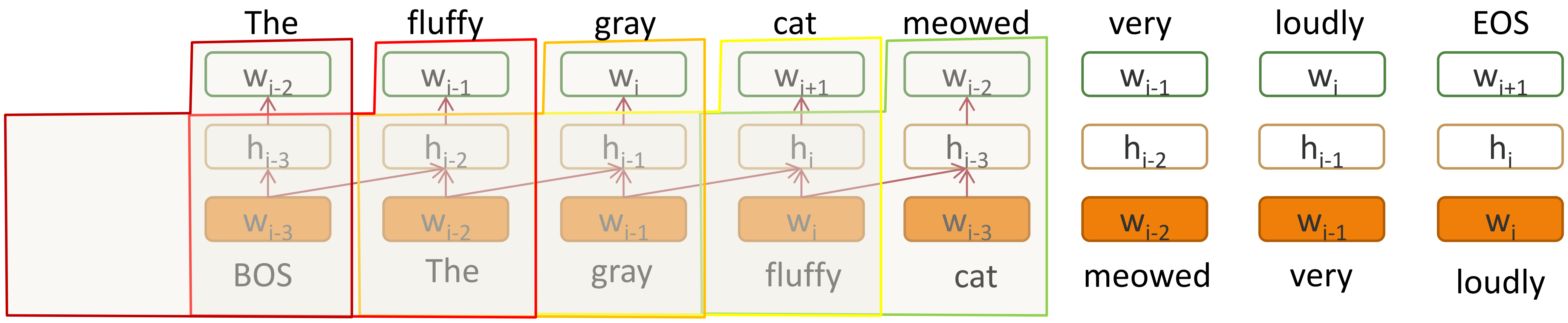
A Neural N-Gram Model (N=3)

The fluffy gray cat meowed very loudly



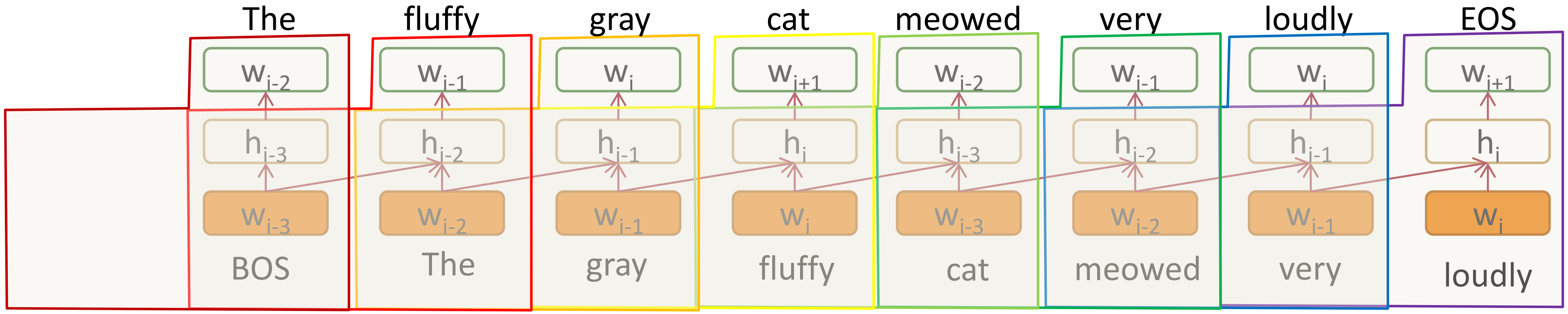
A Neural N-Gram Model (N=3)

The fluffy gray cat meowed very loudly

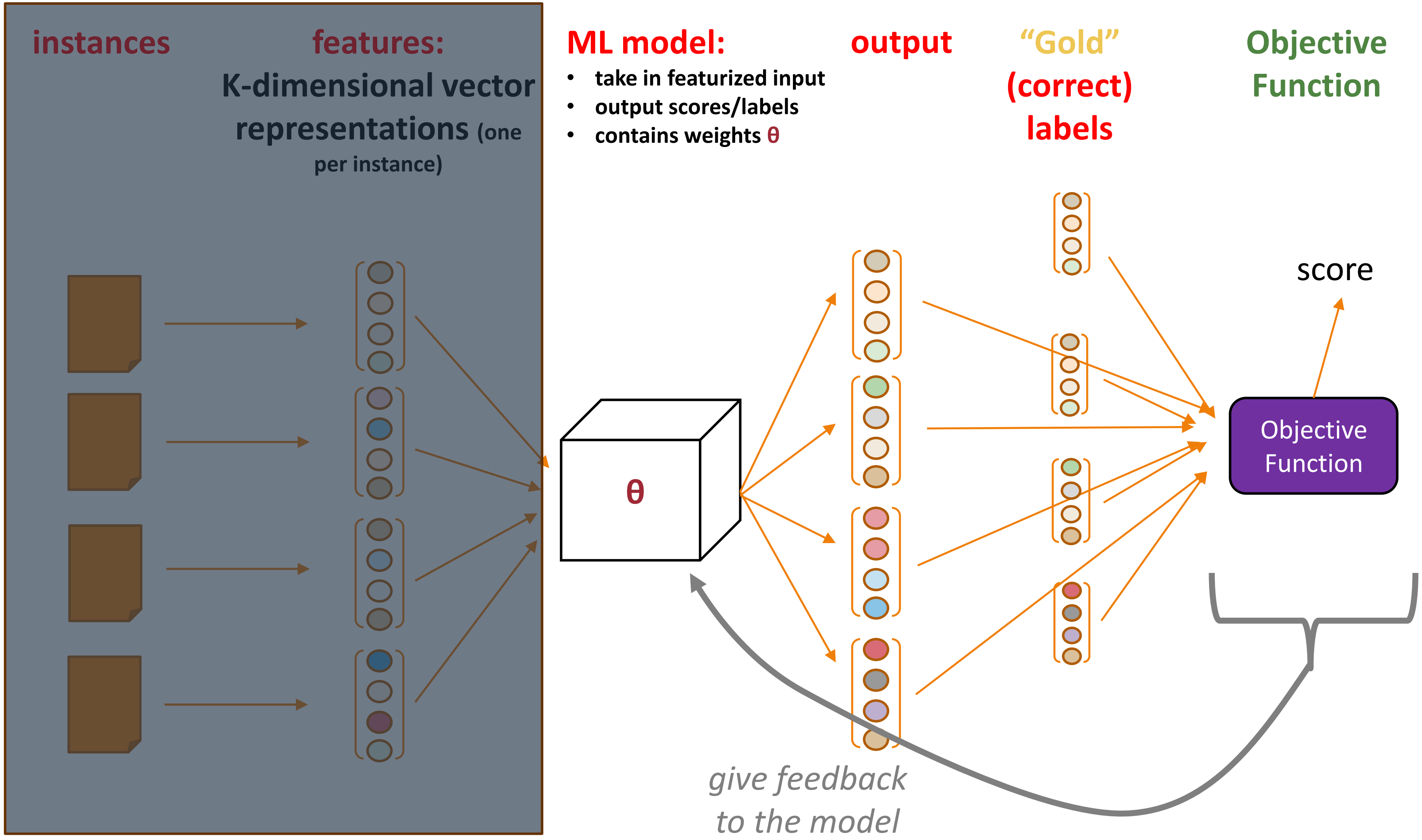


A Neural N-Gram Model (N=3)

The fluffy gray cat meowed very loudly



ML/NLP Framework for Learning



Review:
Maximize Log-Likelihood (Classification)

Original maxent equation

$$\frac{\exp(\theta_y^T f(x))}{\sum_{y'} \exp(\theta_{y'}^T f(x))}$$

Differentiating this becomes nicer (even though Z depends on θ)

$$\log \prod_i p_\theta(y_i|x_i) = \sum_i \log p_\theta(y_i|x_i)$$

Inverse of exp

log(exp(x)) = x

$$= \sum_i \overbrace{\theta_{y_i}^T f(x_i) - \log Z(x_i)}$$

$$= F(\theta)$$

Review:
Minimize Cross Entropy Loss

Cross entropy:
How much $\hat{y}$ differs from the true y

Classifier output

True probability (i.e., correct output)

$$L^{\text{xent}}(\hat{y}, y) =$$

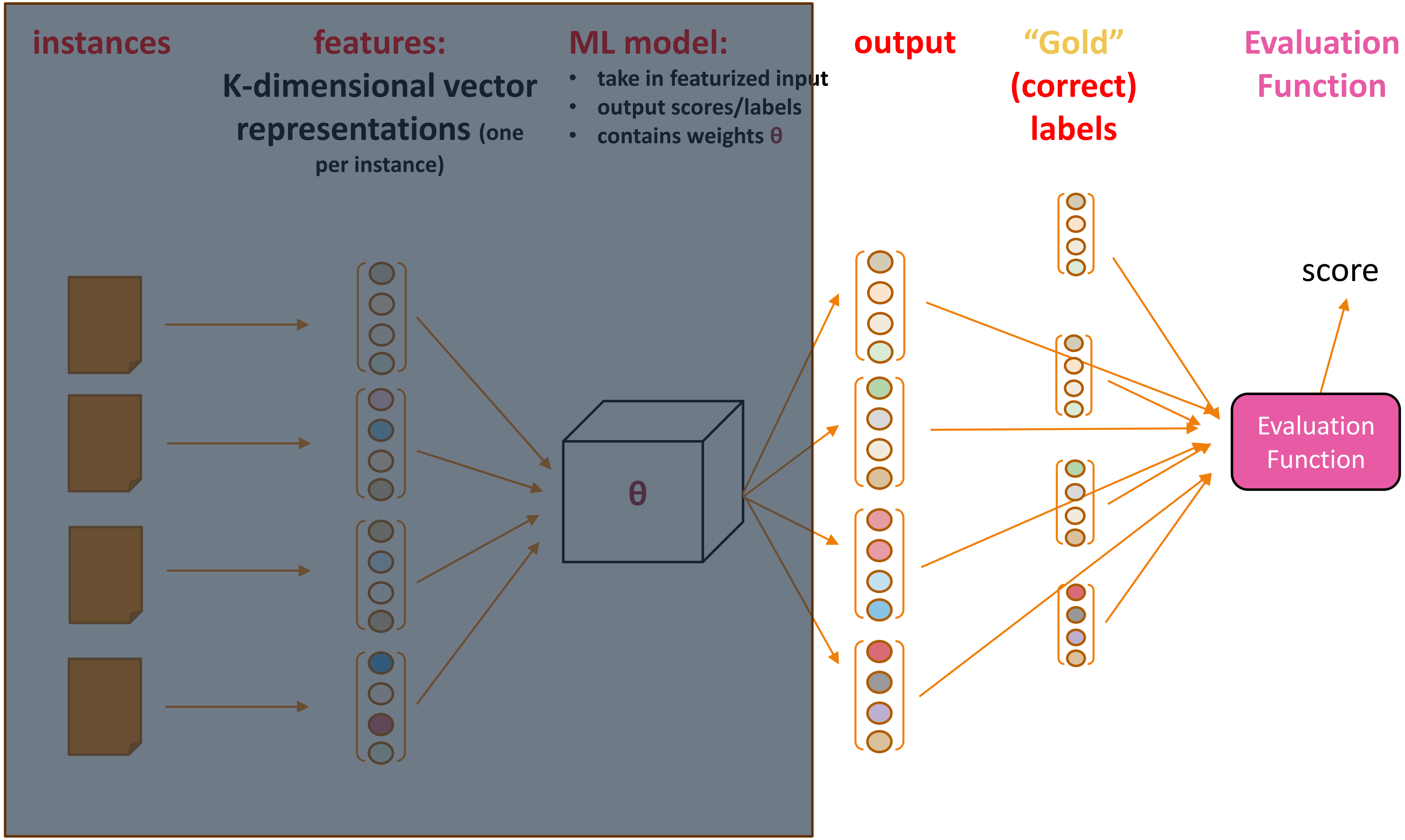
index of "1" indicates correct value

$$\begin{pmatrix} 0 \\ 0 \\ \dots \\ 1 \\ \dots \\ 0 \end{pmatrix}$$

one-hot vector

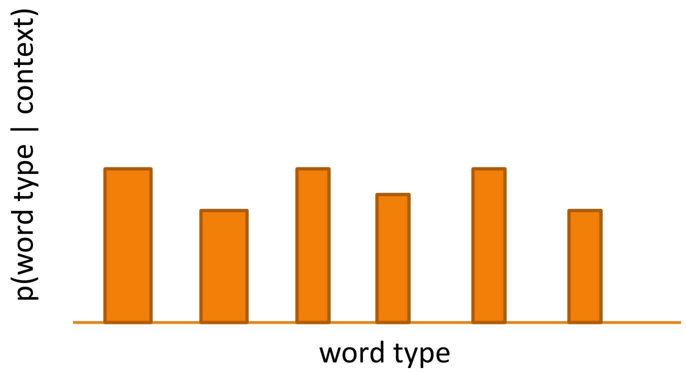
objective is convex (when f(x) is not learned)

ML/NLP Framework for Prediction

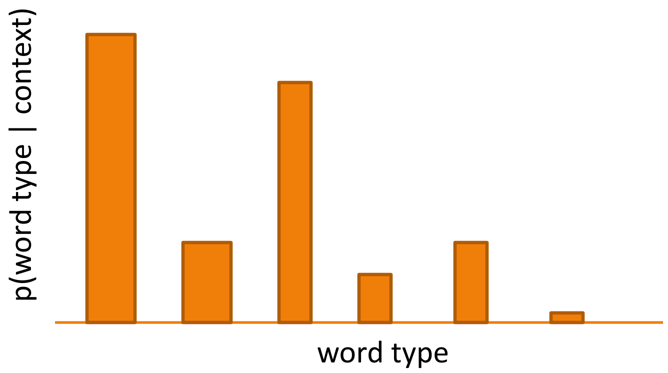


Perplexity: Average "Surprisal"

Lower is better : lower perplexity ➔ less surprised



Less certain ➔
More surprised ➔
Higher perplexity



More certain ➔
Less surprised ➔
Lower perplexity

"A Neural Probabilistic Language Model," Bengio et al. (2003)

BASELINES

LM Name	N-gram	Params.	Test PPL
Interpolation	3	---	336
Kneser-Ney backoff	3	---	323
Kneser-Ney backoff	5	---	321
Class-based backoff	3	500 classes	312
Class-based backoff	5	500 classes	312

“A Neural Probabilistic Language Model,” Bengio et al. (2003)

BASELINES

LM Name	N-gram	Params.	Test PPL
Interpolation	3	---	336
Kneser-Ney backoff	3	---	323
Kneser-Ney backoff	5	---	321
Class-based backoff	3	500 classes	312
Class-based backoff	5	500 classes	312

NPLM

N-gram	Word Vector Dim.	Hidden Dim.	Mix with non-neural LM	PPL
5	60	50	No	268
5	60	50	Yes	257
5	30	100	No	276
5	30	100	Yes	252

“we were not able to see signs of over- fitting (on the validation set), possibly because we ran only 5 epochs (over 3 weeks using 40 CPUs)” (Sect. 4.2)

A Closer Look at Neural $p($ Won’t you please donate? $|$ Primary $)$

This is a *class-based* language model, but incorporate the label into the *embedding representation*

To learn $p($ Won’t you please donate? $|$ Class $)$:

Define an embedding method that makes use of the specific label Class

Unlike count-based models, you don’t *need* “separate” models here

LM Comparison for $p($ Won't you please donate? $|$ Primary $)$

N-GRAM/COUNT-BASED	MAXENT/LR	NEURAL
Class-specific	Class-based	Class-based
	Uses features	Uses *embedded* features