Neural Networks for Natural Language Processing

Jun.-Prof. Sophie Fellenz

Week 4 – Language Modeling and Neural Networks

11 Nov 2024

Questions

- Is Skipgram based on neural networks?

- What is the difference between a neural network word embedding and Skipgram? (question from last exam)

Answers

- Difference:
  - Skipgram embeddings are used inside neural networks (first layer)
  - NN embeddings are layers inside the neural network (usually the last layer(s))
- Therefore:
  - For Skipgram you don‘t need a neural network (skipgram itself is a log-linear model. The parameters of this model **are** the embeddings)
  - For NN embeddings, you need the network to get the embeddings, the parameters of the NN are **not** the embeddings

Language Models

Are these sentences ok?

- Jane went to the store.
- store to Jane went the.
- Jane went store.
- Jane goed to the store.
- The store went to Jane.
- The food truck went to Jane.

Are these sentences ok?

- Jane went to the store.
- store to Jane went the.
- Jane went store.
- Jane goed to the store.
- The store went to Jane.
- The food truck went to Jane.

Create a grammar of the language

Consider morphology and exceptions

Semantic categories, preferences

And their exceptions

Probabilistic language models

$$P(X) = \prod_{i=1}^{I} P(\underbrace{x_i}_{\text{Next word}} | \underbrace{x_1, \ldots, x_{i-1}}_{\text{Context}})$$

The big problem: How do we predict

$$P(x_i | x_1, \ldots, x_{i-1})$$

??

What can we do with LMs?

Score sentences:

- Jane went to the store . -> high
- Store to Jane went the . -> low
- (same as calculating loss for training)

Generate sentences:

while didn't choose end-of-sentence symbol:
  calculate probability
  sample a new word from the probability distribution

Count-based Language Models

Count-based unigram model

Independence assumption: $P(x_i|x_1, \dots, x_{i-1}) \approx P(x_i)$

Count-based maximum-likelihood estimation:

$$P_{MLE}(x_i) = \frac{c_{train}(x_i)}{\sum_{\tilde{x}} c_{train}(\tilde{x})}$$

Interpolation with UNK model:

$$P(x_i) = (1 - \lambda_{unk}) * P_{MLE}(x_i) + \lambda_{unk} * P_{unk}(x_i)$$

Higher-order n-gram models

Limit context length to n, count, and divide

$$P_{ML}(x_i|x_{i-n+1},\dots,x_{i-1}) \coloneqq \frac{c(x_{i-n+1},\dots,x_i)}{c(x_{i-n+1},\dots,x_{i-1})}$$

$$P(example|this\ is\ an) = \frac{c(this\ is\ an\ example)}{c(this\ is\ an)}$$

Add smoothing, to deal with zero counts:

$$P(x_i|x_{i-n+1},\dots,x_{i-1})$$
$$= \lambda P_{ML}(x_i|x_{i-n+1},\dots,x_{i-1}) + (1-\lambda)P(x_i|x_{i-n+2},\dots,x_{i-1})$$

Smoothing methods

Additive/Dirichlet:

Fallback distribution

$$P(x_i|x_{i-n+1},\dots,x_{i-1}) \coloneqq \frac{c(x_{i-n+1},\dots,x_i) + \alpha P(x_i|x_{i-n+2},\dots,x_{i-1})}{c(x_{i-n+1},\dots,x_{i-1}) + \alpha}$$

Interpolation hyperparameter

Discounting:

$$P(x_i|x_{i-n+1},\dots,x_{i-1}) \coloneqq \frac{c(x_{i-n+1},\dots,x_i) - d + \alpha P(x_i|x_{i-n+2},\dots,x_{i-1})}{c(x_{i-n+1},\dots,x_{i-1})}$$

discount hyperparameter

Interpolation calculated by sum of discounts $\alpha = \sum_{\{\tilde{x};c(x_{i-n+1},\dots,\tilde{x})>0\}} d$

Problems and solutions?

Cannot share strength among similar words

she bought a car	she bought a bicycle
she purchased a car	she purchased a bicycle

solution: class based language models

Cannot condition on context with intervening words

Dr. Jane Smith	Dr. Gertrude Smith

solution: skip-gram language models

Problems and solutions?

Cannot handle long-distance dependencies

For tennis class he wanted to buy his own raquet for programming class he wanted to buy his own computer

solution: cache, trigger, topic, syntactic models, etc.

When to use n-gram models

- Neural language models (next) achieve better performance, but
- n-gram models are extremely fast to estimate/apply
- n-gram models can be better at modeling low-frequency phenomena

LM Evaluation

Evaluation of LMs

Log-likelihood:

$$LL(\mathcal{E}_{test}) = \sum_{E \in \mathcal{E}_{test}} \log P(E)$$

Per-word Log Likelihood:

$$WLL(\mathcal{E}_{test}) = \frac{1}{\sum_{E \in \mathcal{E}_{test}} |E|} \sum_{E \in \mathcal{E}_{test}} \log P(E)$$

Per-word Entropy:

$$\mathrm{H}(\mathcal{E}_{test}) = \frac{1}{\sum_{E \in \mathcal{E}_{test}} |E|} \sum_{E \in \mathcal{E}_{test}} -\log_2 P(E)$$

Perplexity:

$$\mathrm{ppl}(\mathcal{E}_{test}) = 2^{H(\mathcal{E}_{test})} = e^{-WLL(\mathcal{E}_{test})}$$

Evaluation and Vocabulary

- Important: the vocabulary must be the same over models you compare
- Or more accurately, all models must be able to generate the test set
- E.g.: A model that has a vocabulary with only the unknown word is trivial and it would be unfair to compare with a model that has all different words in the vocabulary

Log-linear models

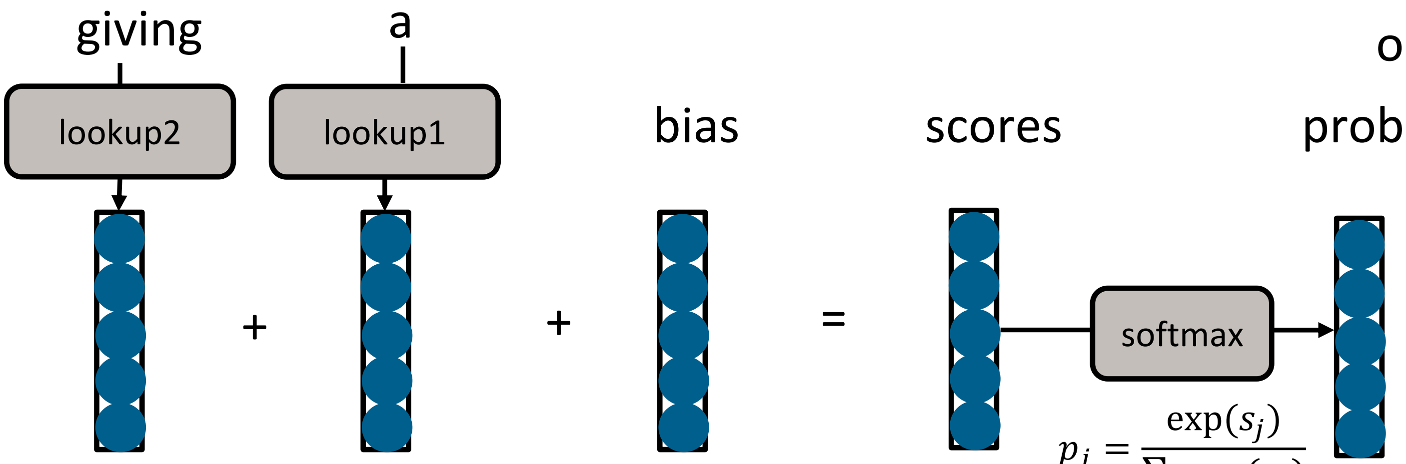
An Alternative: Featurized Models

- Calculate features of the context
- Based on the features, calculate probabilities
- Optimize feature weights using gradient descent, etc.

An Alternative: Featurized Models

Calculate features of the context, calculate probabilities

Each vector is size of output vocabulary



$$p_j = \frac{\exp(s_j)}{\sum_j \exp(s_j)}$$

Feature weights optimized by SGD, etc

Example:

Previous words: “giving a"

Words we‘re predicting		How likely are they?		How likely are they given prev. word is “a”?		How likely are they given 2nd prev. word is “giving”?		Total score
a the talk gift hat …	$b = $	$\begin{pmatrix} 3.0 \\ 2.5 \\ -0.2 \\ 0.1 \\ 1.2 \\ \dots \end{pmatrix}$	$\mathrm{W}_{1,a} =$	$\begin{pmatrix} -6.0 \\ -5.1 \\ 0.2 \\ 0.1 \\ 0.5 \\ \dots \end{pmatrix}$	$W_{2,giving} =$	$\begin{pmatrix} -0.2 \\ -0.3 \\ 1.0 \\ 2.0 \\ -1.2 \\ \dots \end{pmatrix}$	$s =$	$\begin{pmatrix} -3.2 \\ -2.9 \\ 1.0 \\ 2.2 \\ 0.6 \\ \dots \end{pmatrix}$

Training Algorithm

- Calculate the gradient of the loss function with respect to the parameters
- How? Use the chain rule / back-propagation. More in a second
- Update to move in a direction that decreases the loss

What Problems are Handled?

Cannot share strength among similar words

she bought a car
she purchased a car

she bought a bicycle
she purchased a bicycle

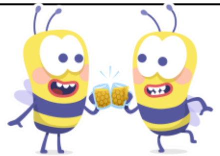
not solved yet

Cannot condition on context with intervening words

Dr. Jane Smith

Dr. Gertrude Smith

Solved!



Problems and solutions?

Cannot handle long-distance dependencies

For tennis class he wanted to buy his own raquet
for programming class he wanted to buy his own computer

Not solved yet

Beyond linear models

Linear Models can't Learn Feature Combinations

Students take tests → high

Students write tests → low

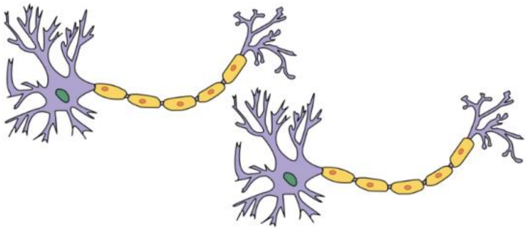
Teachers take tests → low

Teachers write tests → high

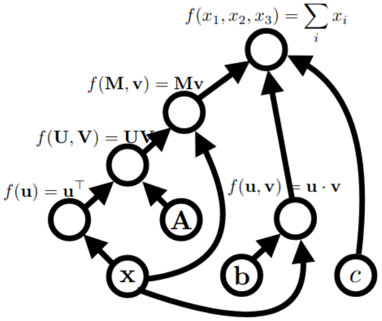
- These can't be expressed by linear features
- What can we do?
  - Remember combinations as features (individual
  - scores for "students take", "teachers write")
  - → Feature space explosion!
- Neural networks!

“Neural” Nets

Original Motivation: Neurons in the Brain



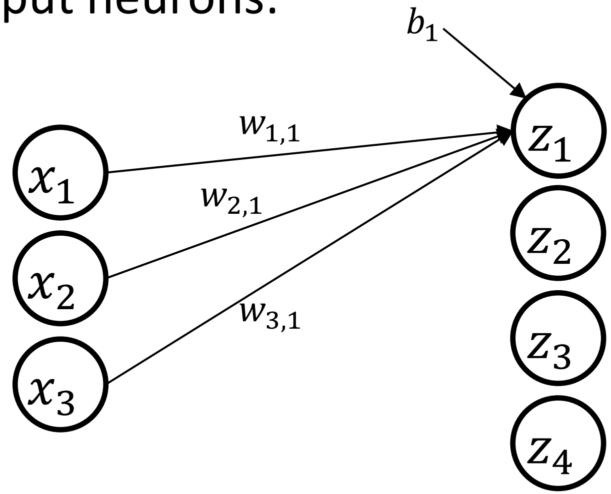
Current Conception: Computation Graphs



RPTU

Input neurons:

Output neurons:



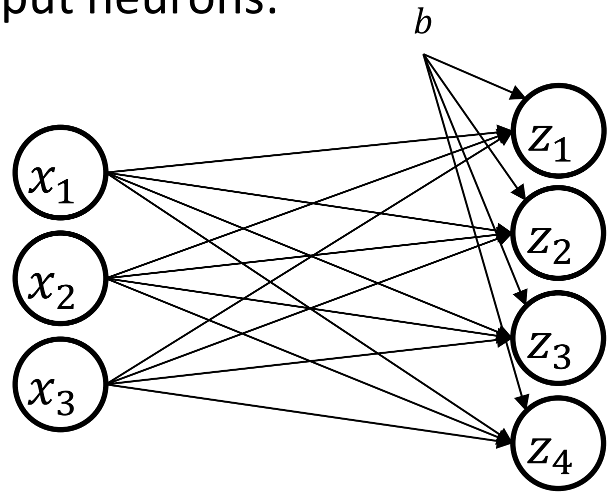
$x_1 w_{1,1} + x_2 w_{2,1} + x_3 w_{3,1} + b_1$

$x_1 w_{1,2} + x_2 w_{2,2} + x_3 w_{3,2} + b_2$

$x_1 w_{1,3} + x_2 w_{2,3} + x_3 w_{3,3} + b_3$

$x_1 w_{1,4} + x_2 w_{2,4} + x_3 w_{3,4} + b_4$

Input neurons:



Output neurons:

$x_1 w_{1,1} + x_2 w_{2,1} + x_3 w_{3,1} + b_1$

$x_1 w_{1,2} + x_2 w_{2,2} + x_3 w_{3,2} + b_2$

$x_1 w_{1,3} + x_2 w_{2,3} + x_3 w_{3,3} + b_3$

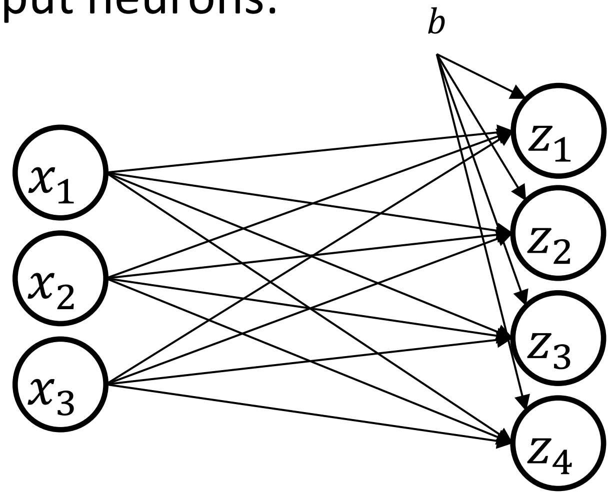
$x_1 w_{1,4} + x_2 w_{2,4} + x_3 w_{3,4} + b_4$

$$\boldsymbol{z^T} = \boldsymbol{x^T W} + \boldsymbol{b^T}$$

$$z^T = (z_1, z_2, z_3, z_4), x^T = (x_1, x_2, x_3),$$
$$W = \left(w_{i,j}\right)_{i,j}, b^T = (b_1, b_2, b_3, b_4)$$

Input neurons:

Output neurons:



$x_1 w_{1,1} + x_2 w_{2,1} + x_3 w_{3,1} + b_1$

$x_1 w_{1,2} + x_2 w_{2,2} + x_3 w_{3,2} + b_2$

$x_1 w_{1,3} + x_2 w_{2,3} + x_3 w_{3,3} + b_3$

$x_1 w_{1,4} + x_2 w_{2,4} + x_3 w_{3,4} + b_4$

Neurons in the brain only activate after a certain threshold is overcome

Simulate with activation function $\sigma(x) = \begin{cases} x \text{ } if \text{ } x > 0 \\ 0 \text{ } otherwise \end{cases}$

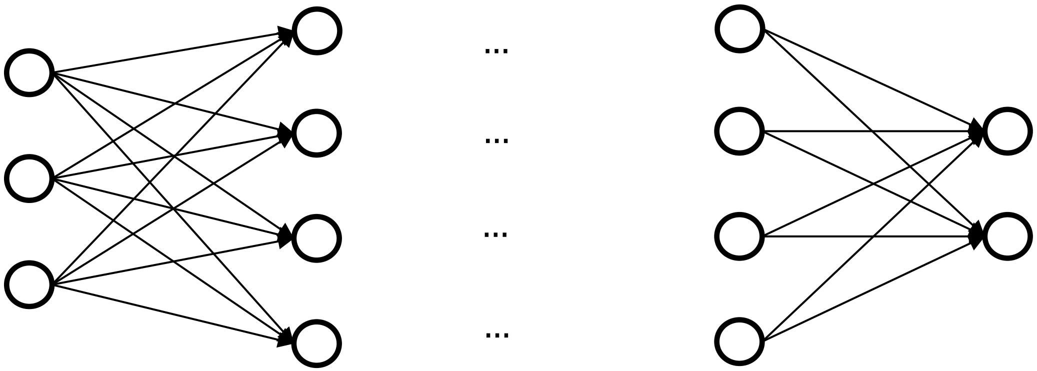
$$\boldsymbol{Output} = \boldsymbol{\sigma}(\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{W} + \boldsymbol{b}^{\boldsymbol{T}})$$

Input layer Hidden layers Output

Usually we use multiple layers

$$\boldsymbol{Output} = \boldsymbol{\sigma}(\ldots\boldsymbol{\sigma}\left(\boldsymbol{x}^{\boldsymbol{T}}\boldsymbol{W}_{\mathbf{1}} + \boldsymbol{b}_{\mathbf{1}}^{\boldsymbol{T}}\right)\ldots\boldsymbol{W}_{\boldsymbol{n}} + \boldsymbol{b}_{\boldsymbol{n}}^{\boldsymbol{T}})$$

Input layer Hidden layers Output



How does this work specifically?

Example: Given cat or dog image.

Task: Find out if it is a cat or a dog. Output $(1,0)$ if cat and $(0,1)$ if dog.

Input layer Hidden layers Output

Why does this work?

- One can prove that any function, i.e. the cat-dog recognition function can be approximated by a neural network!
- Network needs to be large enough and have correct weights W and b

Input layer Hidden layers Output

How do we find out how large it needs to be?

- Experimentation!

How do we find the weights?

- We initiate randomly and then train on train data!

Training

- Given:
  Training data, i.e. input data x where desired output y is known.
  Neural Network $n_{W,b}$
- Output of Neural Network: $n_{W,b}(x)$
  Desired output: y
- Loss: $\left\| n_{W,b}(x) - y \right\|^2$
- Goal: Minimize loss by optimizing weights W and b

expression:

$$x$$

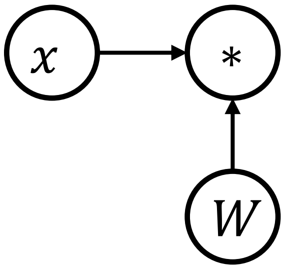
graph:

A **node** is a {tensor, matrix, vector, scalar} value

x

- An **edge** represents a function argument (and also a data dependency). They are just pointers to nodes.
- A **node** with an incoming **edge** is a **function** of that edge's tail node.
- A **node** knows how to compute its value and the value of its derivative w.r.t each argument (edge) times a derivative of an arbitrary input $\frac{\partial f}{\partial u}$.

$f(x, W) = x^T W$



expression:

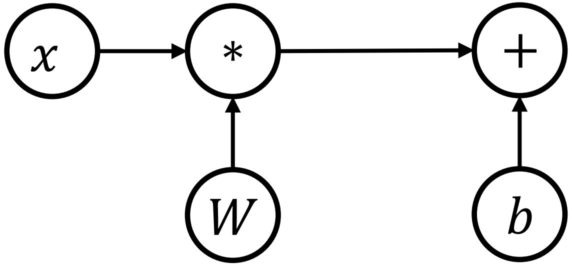
$$x^T W + b$$

graph:

Functions can be nullary, unary, binary, ... n-ary.

Often they are unary or binary

$$f_1(x, W) = x^T W \quad f_2(u, b) = u + b$$

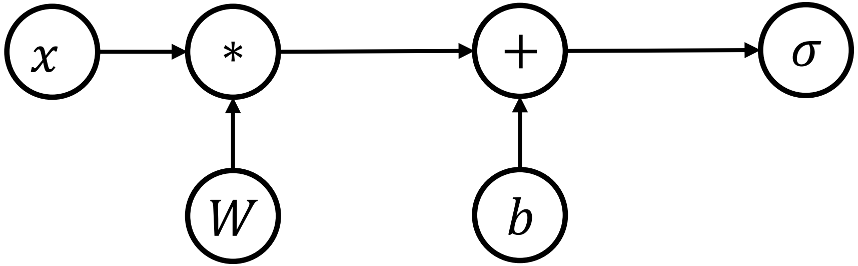


expression:

$$\sigma(x^T W + b^T)$$

graph:

$$f_1(x, W) = x^T W \quad f_2(u, b) = u + b^T \quad f_3(u) = \sigma(u)$$

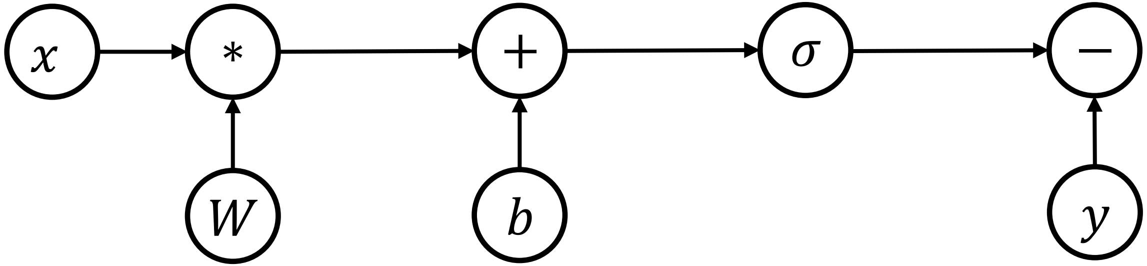


expression:

$$\sigma(x^T W + b^T) - y^T$$

graph:

$f_1(x, W) = x^T W$ $\quad f_2(u, b) = u + b^T$ $\quad f_3(u) = \sigma(u)$ $\quad f_4(u, y) = u - y$

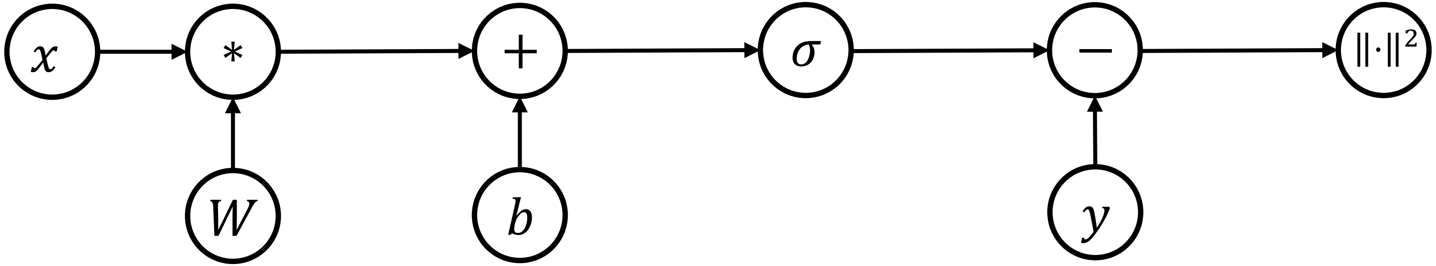


expression:

$$\text{Loss} = \|\sigma(x^T W + b^T) - y^T\|^2 = f_5(f_4(f_3(f_2(f_1(x, W), b)), y))$$

graph:

$f_1(x, W) = x^T W$ $\quad f_2(u, b) = u + b^T$ $\quad f_3(u) = \sigma(u)$ $\quad f_4(u, y) = u - y$ $\quad f_5(u) = \|u\|^2$



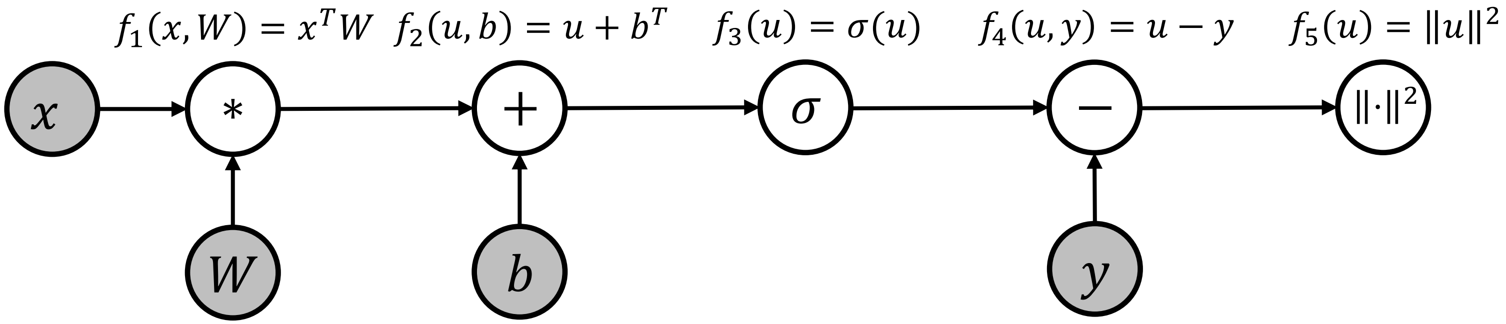
Algorithms (1)

- Graph construction

- Forward propagation

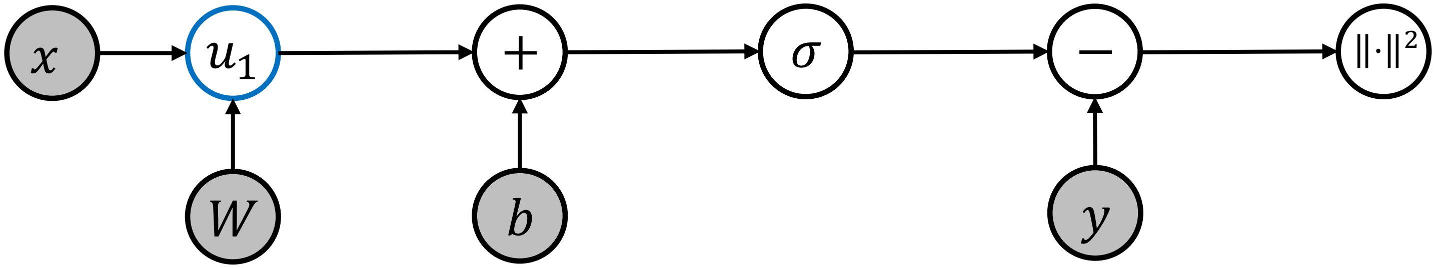
  In topological order, compute the **value** of the node given its inputs

Forward Propagation

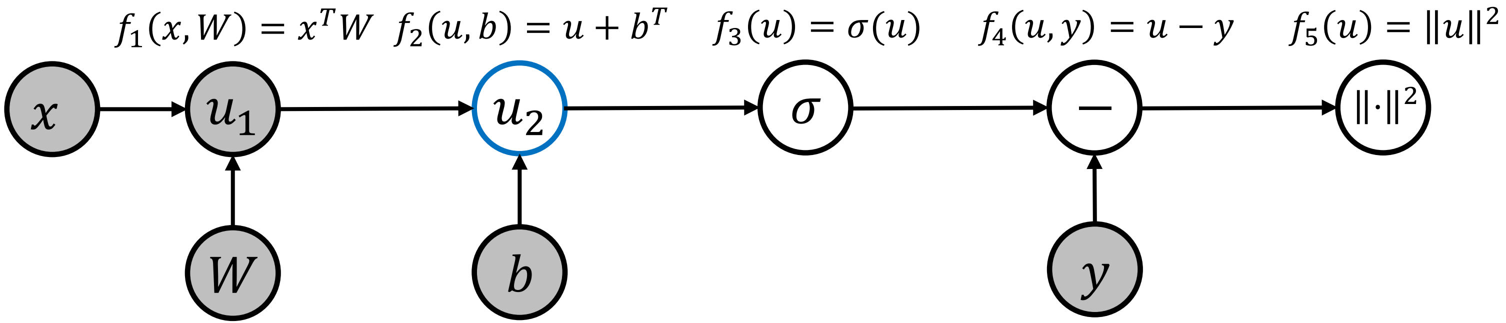


Forward Propagation

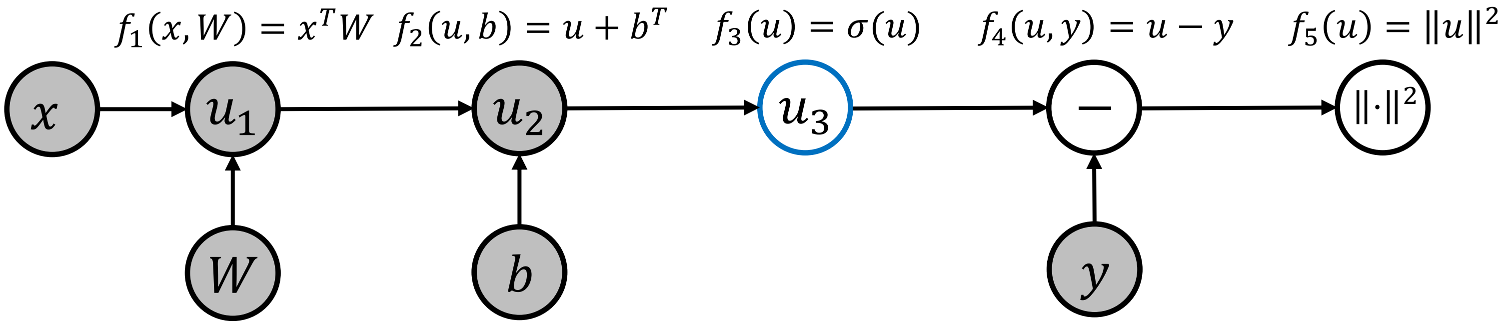
$f_1(x, W) = x^T W \quad f_2(u, b) = u + b^T \quad f_3(u) = \sigma(u) \quad f_4(u, y) = u - y \quad f_5(u) = \|u\|^2$



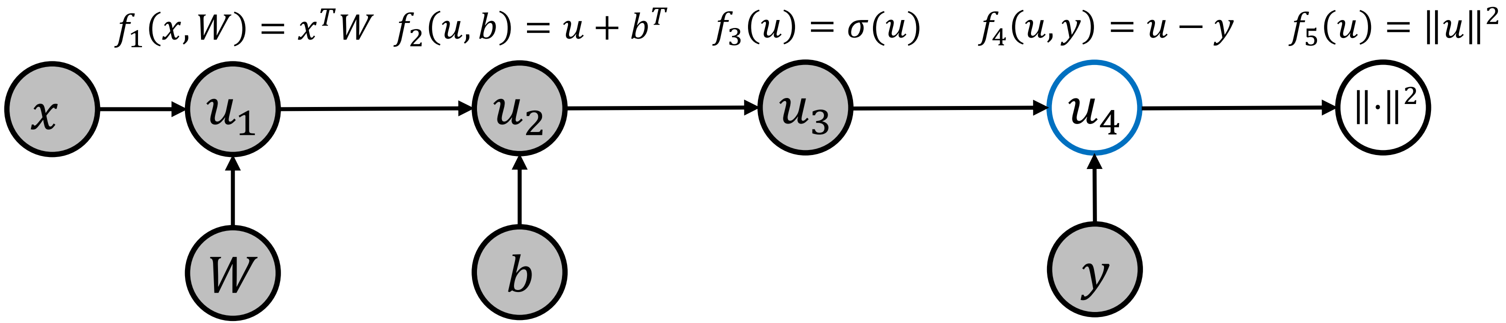
Forward Propagation



Forward Propagation

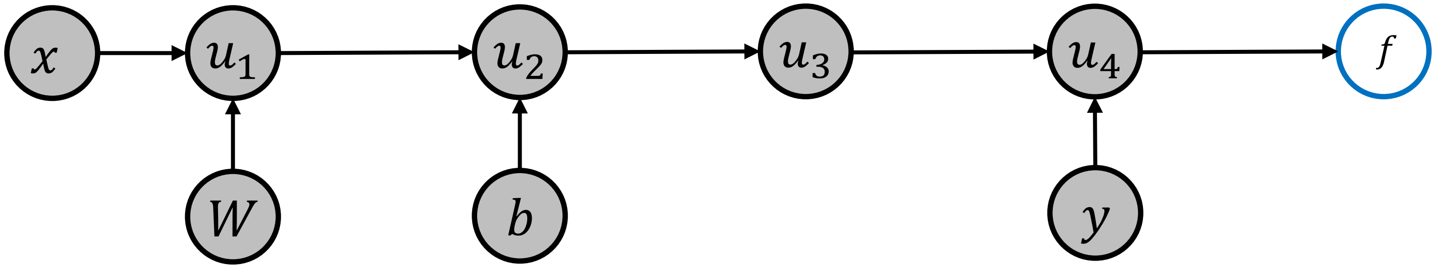


Forward Propagation



Forward Propagation

$f_1(x, W) = x^T W$ $f_2(u, b) = u + b^T$ $f_3(u) = \sigma(u)$ $f_4(u, y) = u - y$ $f_5(u) = \|u\|^2$



Algorithms (2)

- Aim: Minimize loss $f(x, W, b) = \|\sigma(x^T W + b^T) - y^T\|^2$ w.r.t. weights W, b
- Idea: Gradient = Direction of highest increase
  Calculate its gradients $\frac{\partial f}{\partial W}$ and $\frac{\partial f}{\partial b}$ and move against it
- Parameter update:
  - Move the parameters against the direction of this derivative
  - $W = W - \alpha * \frac{\partial f}{\partial W}$, $b = b - \alpha * \frac{\partial f}{\partial b}$
- $\alpha > 0$ is learning rate

Algorithms (2)

Back-propagation:

- Process examples in reverse topological order
- Calculate the derivatives of the parameters with respect to the final value

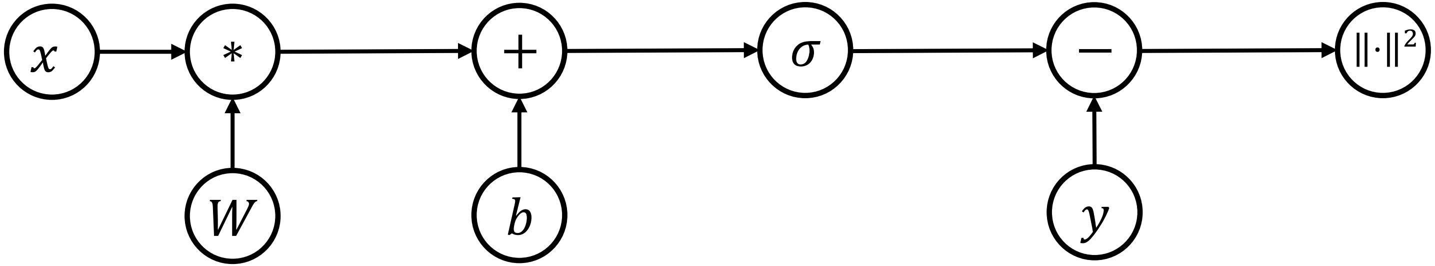
Back Propagation

Expression:

$$\text{Loss} = \|\sigma(x^T W + b^T) - y^T\|^2$$

Aim: Minimize loss by optimizing weights W, b

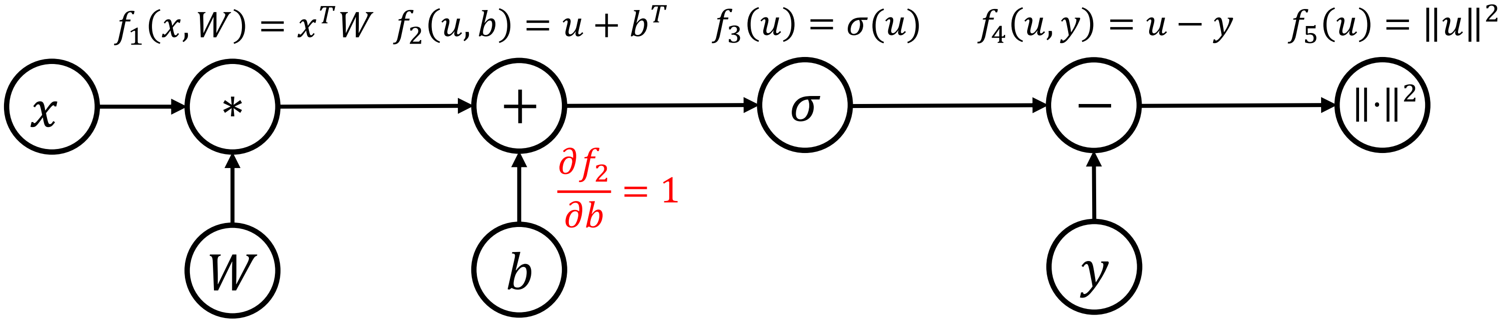
$f_1(x, W) = x^T W$ $\quad f_2(u, b) = u + b^T$ $\quad f_3(u) = \sigma(u)$ $\quad f_4(u, y) = u - y$ $\quad f_5(u) = \|u\|^2$



Back Propagation

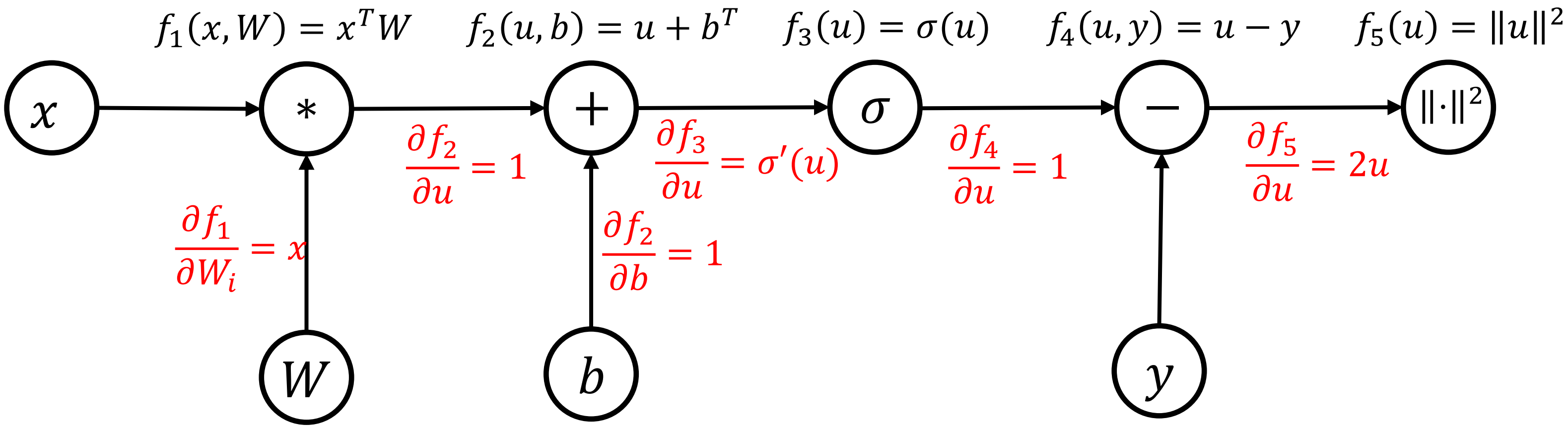
Step 1: Compute all derivatives at every node wrt. the inputs (at relevant edges)

Example: $\frac{\partial f_2}{\partial b} = 1$



Back Propagation

Step 1: Compute all derivatives at every node wrt. the inputs.



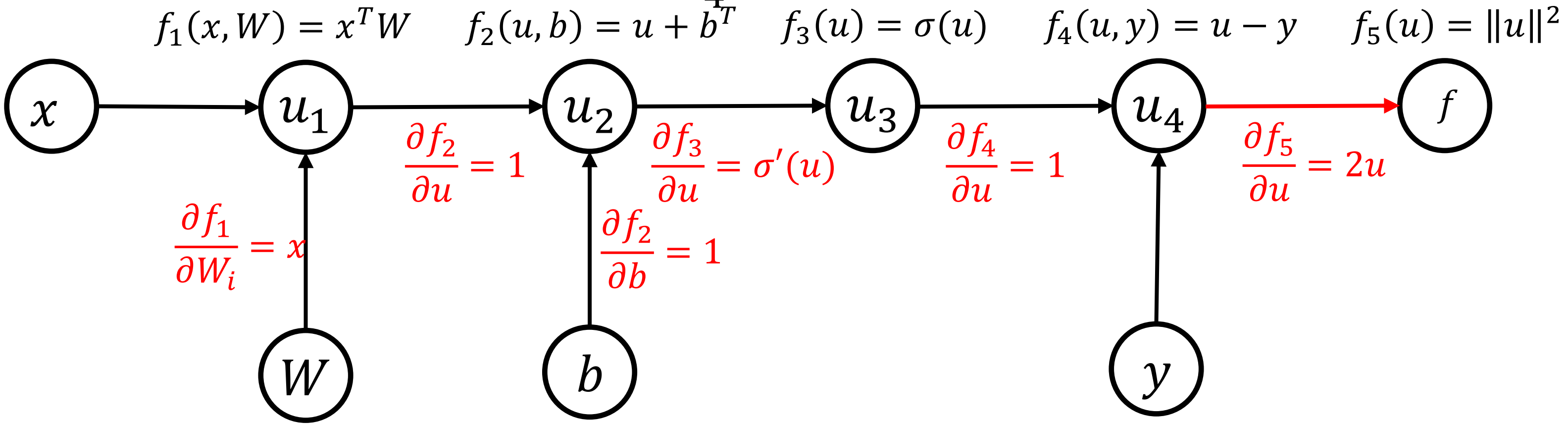
Back Propagation

Step 2: Use the chain rule.

Example: $f(x, W, b) = \|\sigma(x^T W + b^T) - y^T\|^2$

$= f_5(f_4(f_3(f_2(f_1(x, W), b)), y))$

$$\frac{\partial f(x, W, b)}{\partial b} = \frac{\partial f_5}{\partial u_4}(u_4)\frac{\partial f_4}{\partial b}(u_3, y) = 2u_4\frac{\partial f_4}{\partial b}(u_3, y)$$



Back Propagation

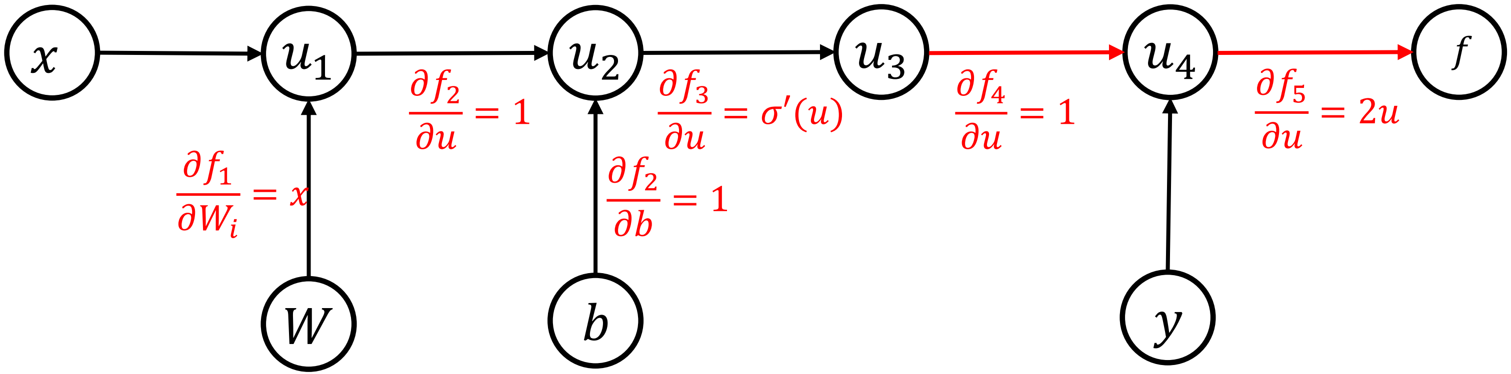
Step 2: Use the chain rule.

Example: $f(x, W, b) = \|\sigma(x^T W + b^T) - y^T\|^2$

$= f_5(f_4(f_3(f_2(f_1(x, W), b)), y))$

$$\frac{\partial f(x, W, b)}{\partial b} = \frac{\partial f_5}{\partial u_4}(u_4) \frac{\partial f_4}{\partial u_3}(u_3, y) \frac{\partial f_3}{\partial b}(u_2) = 2u_4 \frac{\partial f_3}{\partial b}(u_2)$$

$f_1(x, W) = x^T W$ $\quad f_2(u, b) = u + b^T$ $\quad f_3(u) = \sigma(u)$ $\quad f_4(u, y) = u - y$ $\quad f_5(u) = \|u\|^2$

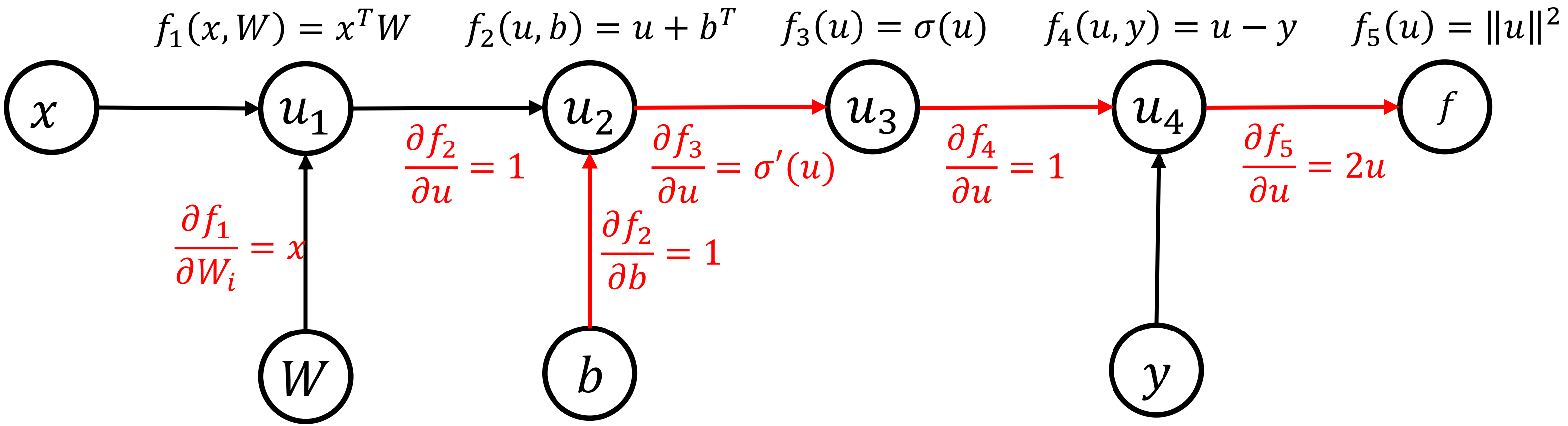


Back Propagation

Step 2: Use the chain rule.

Example: $f(x, W, b) = \|\sigma(x^T W + b^T) - y^T\|^2 = f_5(f_4(f_3(f_2(f_1(x, W), b)), y))$

$$\frac{\partial f(x, W, b)}{\partial b} = \frac{\partial f_5}{\partial u_4}(u_4)\frac{\partial f_4}{\partial u_3}(u_3, y)\frac{\partial f_3}{\partial u_2}(u_2)\frac{\partial f_2}{\partial b}(u_1, b) = 2u_4\sigma'(u_2)$$

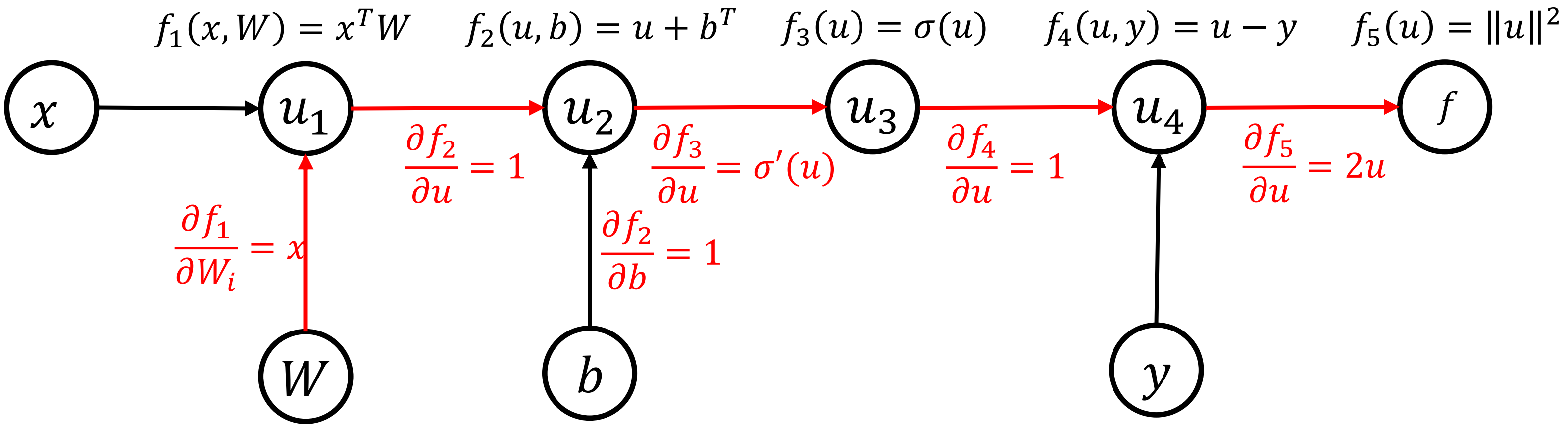


Back Propagation

Step 2: Use the chain rule.

Similarly, we calculate $\frac{\partial f}{\partial W}$ .Note that we already have calculated $\frac{\partial f}{\partial u_2}$ previously.

In summary: Multiply derivatives starting from end



Back Propagation

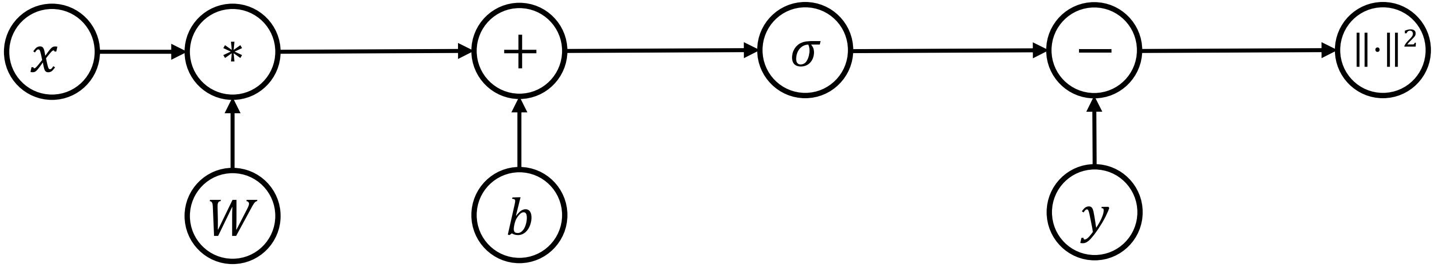
Step 3: Apply Gradient descent

Update: $\alpha > 0$ learning rate

$$W_{new} = W_{old} - \alpha \frac{\partial f}{\partial W}$$

$$b_{new} = b_{old} - \alpha \frac{\partial f}{\partial b}$$

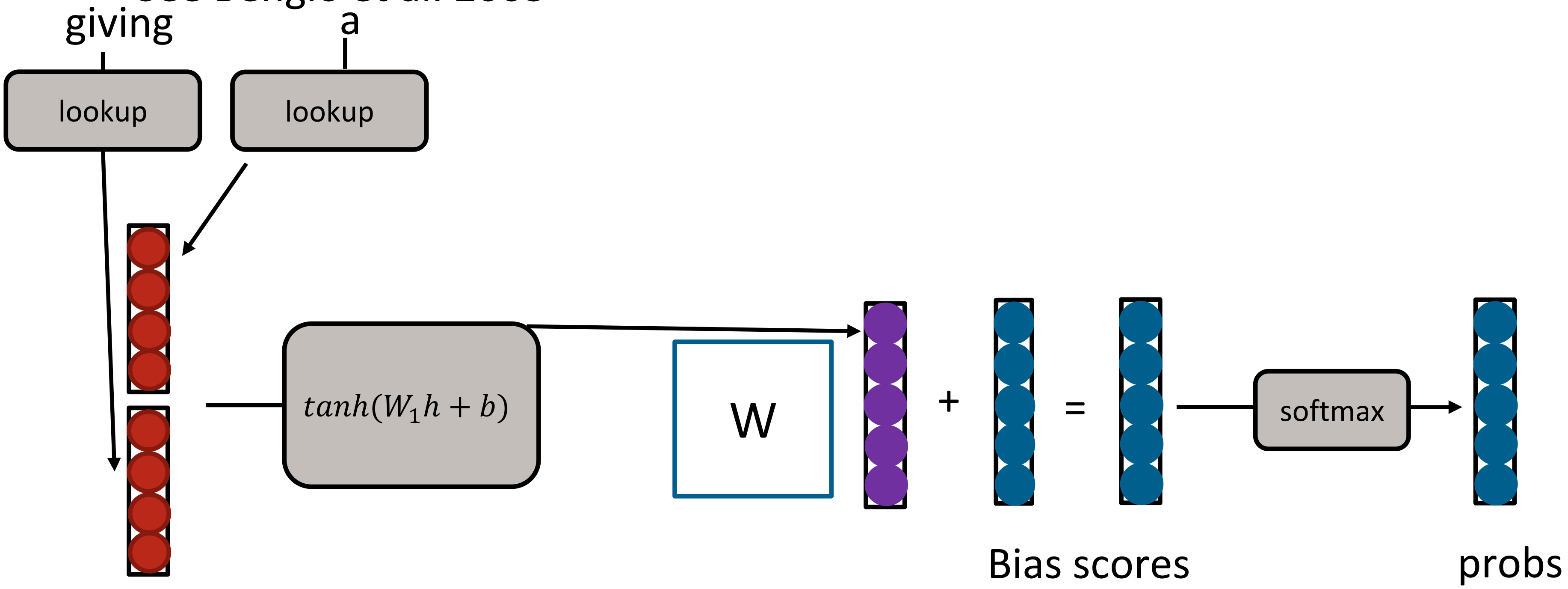
$f_1(x, W) = x^T W$ $\quad f_2(u, b) = u + b^T$ $\quad f_3(u) = \sigma(u)$ $\quad f_4(u, y) = u - y$ $\quad f_5(u) = \|u\|^2$



Back to language modeling

Feed-forward Neural Language Models

- See Bengio et al. 2003



Example of Combination Features

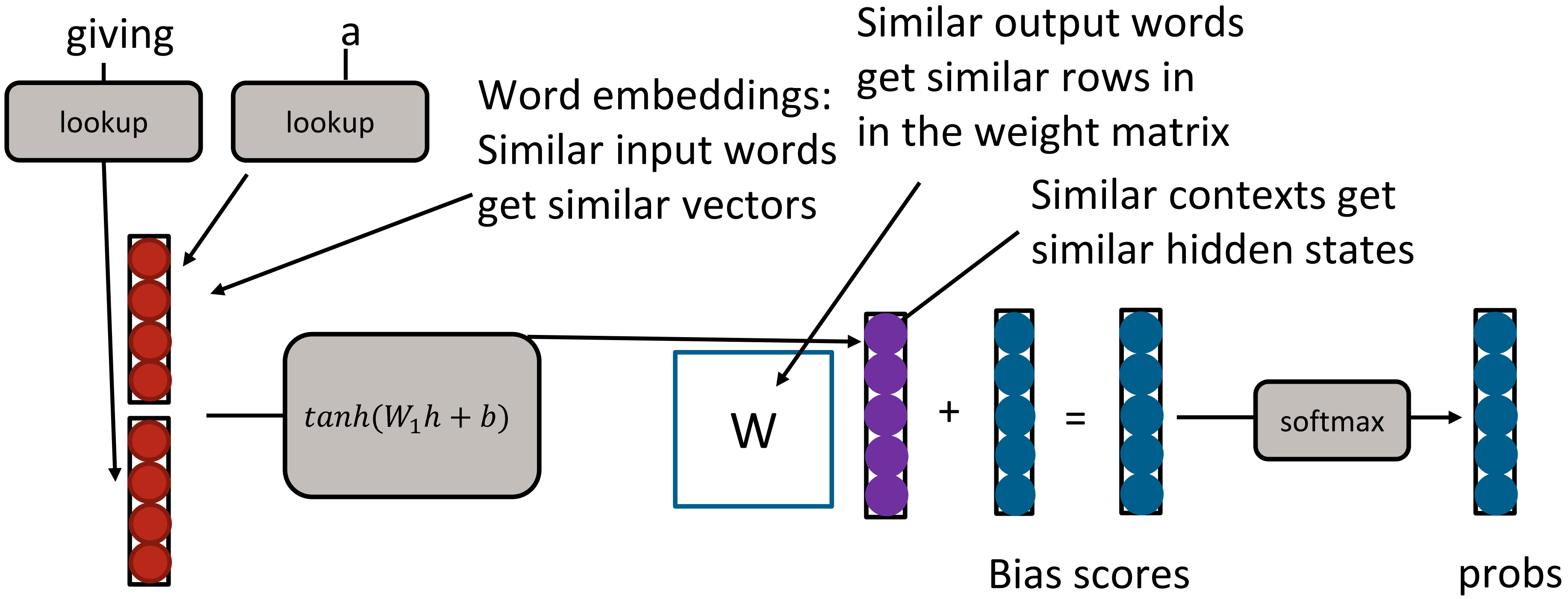
RPTU

- Word embeddings capture features of words
- e.g. feature 1 indicates verbs, feature 2 indicates determiners
- A row in the weight matrix (together with the bias) can capture particular combinations of these features
- e.g. the 34th row in the weight matrix looks at feature 1 in the second to-previous word, and feature 2 in the previous word

$$\begin{array}{ll} \text{giving} & \begin{bmatrix} 1.2 \\ -0.1 \\ 0.7 \\ -2.1 \\ 0.5 \end{bmatrix} \\ \text{a} & \begin{bmatrix} -0.3 \\ 2.0 \\ 0.6 \\ -0.8 \\ -0.4 \end{bmatrix} \end{array} \; * \; \overset{w_{34}}{\begin{bmatrix} 1.5 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1.3 \\ 0 \\ 0 \\ 0 \end{bmatrix}} \; + \; \overset{b_{34}}{-2}$$

positive number if the previous word is a determiner and second-to-previous word is a verb

Where is Strength Shared?



What Problems are Handled?

Cannot share strength among similar words

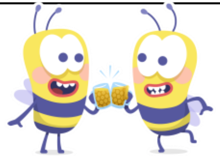
she bought a car	she bought a bicycle
she purchased a car	she purchased a bicycle

solved, and similar contexts as well!

Cannot condition on context with intervening words

Dr. Jane Smith	Dr. Gertrude Smith

Solved!



Problems and solutions?

Cannot handle long-distance dependencies

For tennis class he wanted to buy his own raquet
for programming class he wanted to buy his own computer

Not solved yet

Many Other Potential Designs!

- Neural networks allow design of arbitrarily complex functions!
- In future classes:
  - Recurrent neural network LMs
  - Transformer LMs

Next lecture
Recurrent Neural Networks

Acknowledgements

- CMU Advanced NLP Course:
- https://phontron.com/class/anlp2022/schedule.html
- Sören Laue
- Feibai Huang

References

- Video on Backprop by Andrej Karpathy:
- https://youtu.be/VMj-3S1tku0
- Video on Language modeling by Andrej Karpathy:
- https://youtu.be/PaCmpygFfXo