neural networks for natural language processing (nn4nlp)

Chris Kedzie
kedzie@cs.columbia.edu

September 18, 2019

who am i?

Chris Kedzie
kedzie@cs.columbia.edu

Ph.D. Candidate
(Advisor: Kathy McKeown)

Interested in text summarization, compression, and generation.

Also neural networks and deep learning.

Lesson Plan

linear models and feature design

multi-layer perceptron

optimization

feed-forward language model

Lesson Plan

linear models and feature design

multi-layer perceptron

optimization

feed-forward language model

Example: Movie Review Classification

x	y
2001 is a really great movie	positive
Ed Wood was a great waste of time	negative
The Room is so bad it's good	positive
$\vdots$	$\vdots$

Feature Functions

Mine your dataset for frequently occurring words and phrases:

Feature Functions

Mine your dataset for frequently occurring words and phrases:

"really great movie"
"so bad"
"waste of time"
⋮

Feature Functions

Feature Functions

$$\phi(x)_i = \mathbb{1}\left\{\text{``really great movie''} \in x\right\}$$

Feature Functions

$$\phi(x)_i = \mathbb{1}\left\{\text{“really great movie”} \in x\right\}$$

$$\phi(x)_{i+1} = \mathbb{1}\left\{\text{“really great”} \in x\right\}$$

Feature Functions

$$\phi(x)_i = \mathbb{1}\left\{\text{``really great movie''} \in x\right\}$$

$$\phi(x)_{i+1} = \mathbb{1}\left\{\text{``really great''} \in x\right\}$$

$$\phi(x)_{i+2} = \mathbb{1}\left\{\text{``great movie''} \in x\right\}$$

Feature Functions

$$\phi(x)_i = \mathbb{1}\left\{\text{``really great movie''} \in x\right\}$$

$$\phi(x)_{i+1} = \mathbb{1}\left\{\text{``really great''} \in x\right\}$$

$$\phi(x)_{i+2} = \mathbb{1}\left\{\text{``great movie''} \in x\right\}$$

$$\phi(x)_{i+3} = \mathbb{1}\left\{really \in x\right\}$$

$$\phi(x)_{i+4} = \mathbb{1}\left\{great \in x\right\}$$

$$\phi(x)_{i+5} = \mathbb{1}\left\{movie \in x\right\}$$

Linear Classifier

$$f(x;w) = \begin{cases} 1 & \text{if } \sum_i \phi_i(x) \cdot w_i > 0 \\ -1 & \text{otherwise} \end{cases}$$

Linear Classifier

$$f(x;w) = \begin{cases} 1 & \text{if } \sum_i \phi_i(x) \cdot w_i > 0 \\ -1 & \text{otherwise} \end{cases}$$

If we want a probabilistic model we can simply wrap the decision function in a sigmoid function:

$$P(Y=1|x;w) = \sigma\left(\sum_i \phi_i(x) \cdot w_i\right) = \frac{1}{1+\exp\left\{-\sum_i \phi_i(x) \cdot w_i\right\}}$$

$$P(Y=-1|x;w) = 1 - P(Y=1|x;w)$$

Linear Classifier

- Designing features can be challenging in certain domains. E.g. speech recognition or image classification.

Linear Classifier

- Designing features can be challenging in certain domains. E.g. speech recognition or image classification.
- Ineffecient parameter sharing!

Learning about

$$\phi(x)_i = \mathbb{1}\left\{\text{``really great movie''} \in x\right\}$$

doesn't tell us anything about

$$\phi(x)_j = \mathbb{1}\left\{\text{``really awesome movie''} \in x\right\}$$

$$\phi(x)_k = \mathbb{1}\left\{\text{``really great film''} \in x\right\}$$

$$\phi(x)_l = \mathbb{1}\left\{\text{``really great book''} \in x\right\}$$

even though they may occur in similar contexts.

By comparison, neural network models will allow us to efficiently share parameters and learn useful representations.

They also have their own particular shortcomings as well!

Lesson Plan

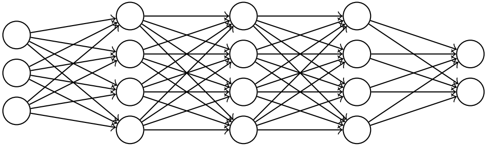
linear models and feature design

multi-layer perceptron

optimization

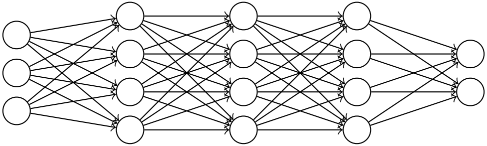
feed-forward language model

Multi-Layer Perceptron (Feed-forward Neural Network)



Multi-Layer Perceptron (Feed-forward Neural Network)

- Input is introduced to the first layer neurons.

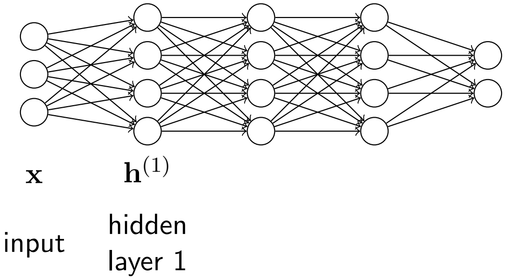


$\mathbf{x}$

input

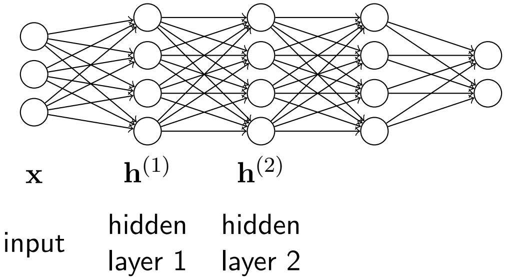
Multi-Layer Perceptron (Feed-forward Neural Network)

- Input is introduced to the first layer neurons.
- Each successive layer activates the next layer,



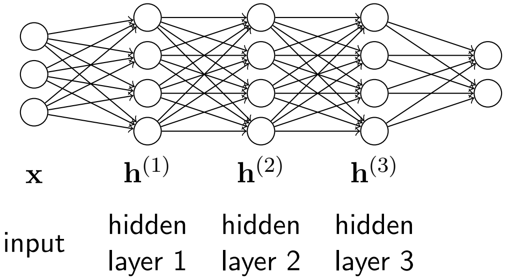
Multi-Layer Perceptron (Feed-forward Neural Network)

- Input is introduced to the first layer neurons.
- Each successive layer activates the next layer,



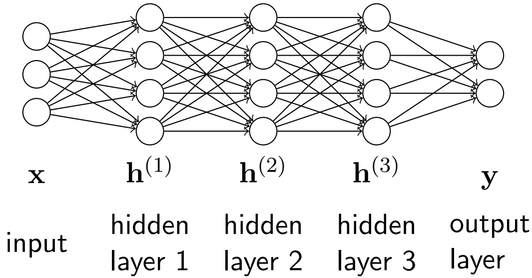
Multi-Layer Perceptron (Feed-forward Neural Network)

- Input is introduced to the first layer neurons.
- Each successive layer activates the next layer,



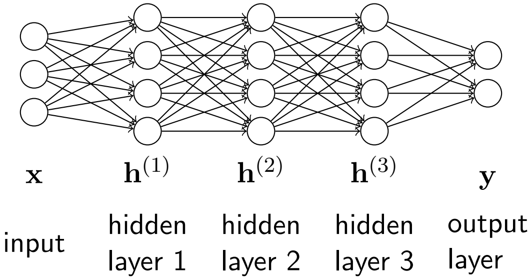
Multi-Layer Perceptron (Feed-forward Neural Network)

- Input is introduced to the first layer neurons.
- Each successive layer activates the next layer,
- finally, producing activations at the output neurons.



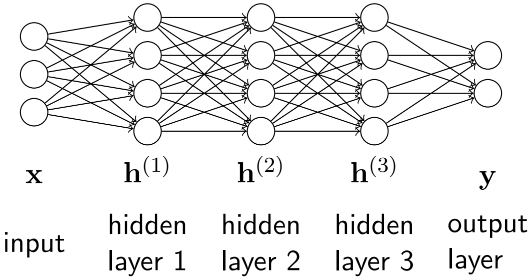
Multi-Layer Perceptron (Feed-forward Neural Network)

- Input is introduced to the first layer neurons.
- Each successive layer activates the next layer,
- finally, producing activations at the output neurons.
- Fully connected: each neuron in layer i connects to every neuron in layer $i+1$.



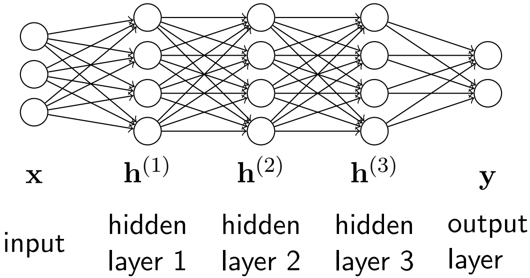
Multi-Layer Perceptron (Feed-forward Neural Network)

- Input is introduced to the first layer neurons.
- Each successive layer activates the next layer,
- finally, producing activations at the output neurons.
- Fully connected: each neuron in layer i connects to every neuron in layer $i + 1$.
- No feedback/cycles (network is a directed acyclic graph).

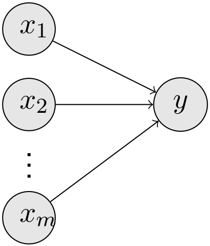


Multi-Layer Perceptron (Feed-forward Neural Network)

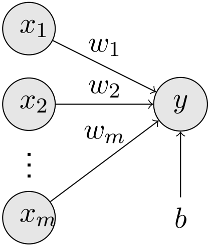
- Input is introduced to the first layer neurons.
- Each successive layer activates the next layer,
- finally, producing activations at the output neurons.
- Fully connected: each neuron in layer i connects to every neuron in layer $i+1$.
- No feedback/cycles (network is a directed acyclic graph).
- Not a generative model of the input (discriminative).



Single Layer Perceptron
(m input neurons, 1 output neuron)



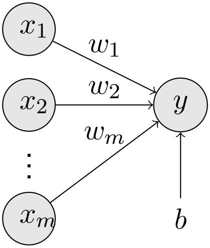
Single Layer Perceptron
(m input neurons, 1 output neuron)



$$\text{activation } y = f\left(\underbrace{\sum_{i=1}^{m} w_i \cdot x_i + b}_{\text{preactivation}}\right)$$

- w_i indicates the strength of the connection between the input activation x_i and the output activation y.

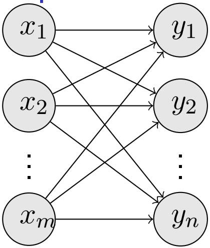
Single Layer Perceptron
(m input neurons, 1 output neuron)



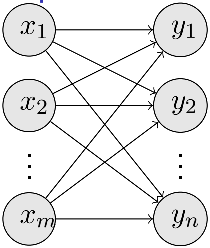
$$\text{activation } y = f\left(\underbrace{\sum_{i=1}^{m} w_i \cdot x_i + b}_{\text{preactivation}}\right)$$

- w_i indicates the strength of the connection between the input activation x_i and the output activation y.
- $f : \mathbb{R} \to \mathbb{R}$ is a nonlinear function.
  Typically, tanh, relu, sigmoid, or softmax.

Single Layer Perceptron
(m input neurons, n output neurons)

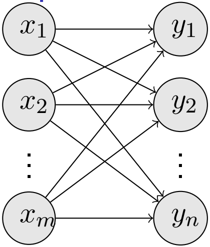


Single Layer Perceptron
(m input neurons, n output neurons)



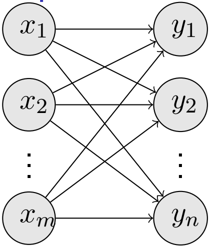
- $y_i = f\left(\sum_{j=1}^{m} w_{i,j} \cdot x_j + b_i\right)$

Single Layer Perceptron
(m input neurons, n output neurons)



- $y_i = f\left(\sum_{j=1}^{m} w_{i,j} \cdot x_j + b_i\right)$
- Equivalently, $y = f(W^\intercal x + b)$
  where $W \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^m$, and $b \in \mathbb{R}^n$

Single Layer Perceptron (m input neurons, n output neurons)



- $y_i = f\left(\sum_{j=1}^{m} w_{i,j} \cdot x_j + b_i\right)$
- Equivalently, $y = f(W^{\intercal} x + b)$
  where $W \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^m$, and $b \in \mathbb{R}^n$
- f is applied elementwise to a vector $v \in \mathbb{R}^n$:

$$f(v) = \begin{bmatrix} f(v_1) & f(v_2) & \ldots & f(v_n) \end{bmatrix}$$

$$f(v)_i = f(v_i)$$

Limitations of a single layer perceptron

- Can only learn functions where input is linearly separable.

Limitations of a single layer perceptron

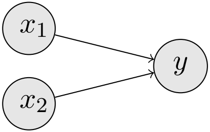
- Can only learn functions where input is linearly separable.
- E.g. Can't learn xor function.

Limitations of a single layer perceptron

- Can only learn functions where input is linearly separable.
- E.g. Can't learn xor function.
- We could design a different kernel/feature representation.

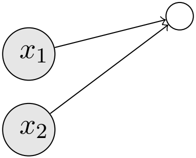
Limitations of a single layer perceptron

- Can only learn functions where input is linearly separable.
- E.g. Can't learn xor function.
- We could design a different kernel/feature representation.
- Or simply add more layers...



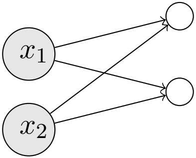
Limitations of a single layer perceptron

- Can only learn functions where input is linearly separable.
- E.g. Can't learn xor function.
- We could design a different kernel/feature representation.
- Or simply add more layers...



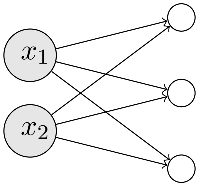
Limitations of a single layer perceptron

- Can only learn functions where input is linearly separable.
- E.g. Can't learn xor function.
- We could design a different kernel/feature representation.
- Or simply add more layers...



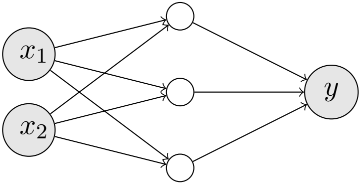
Limitations of a single layer perceptron

- Can only learn functions where input is linearly separable.
- E.g. Can't learn xor function.
- We could design a different kernel/feature representation.
- Or simply add more layers...

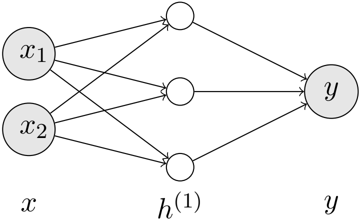


Limitations of a single layer perceptron

- Can only learn functions where input is linearly separable.
- E.g. Can't learn xor function.
- We could design a different kernel/feature representation.
- Or simply add more layers...

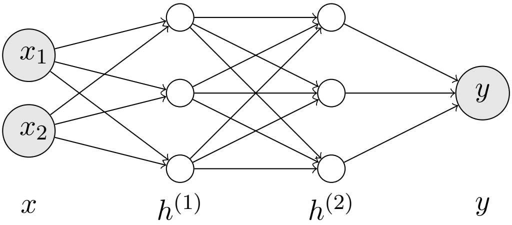


Multi-Layer Perceptron



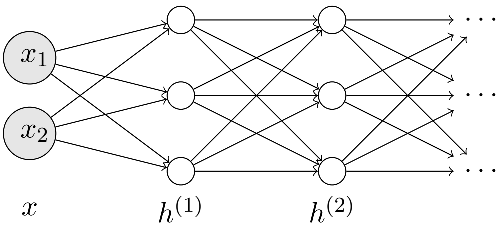
$$
\begin{aligned}
h^{(1)} &= f\left(W^{(1)} \cdot x + b^{(1)}\right) \\
y &= f\left(W^{(2)} \cdot h^{(1)} + b^{(2)}\right)
\end{aligned}
$$

Multi-Layer Perceptron



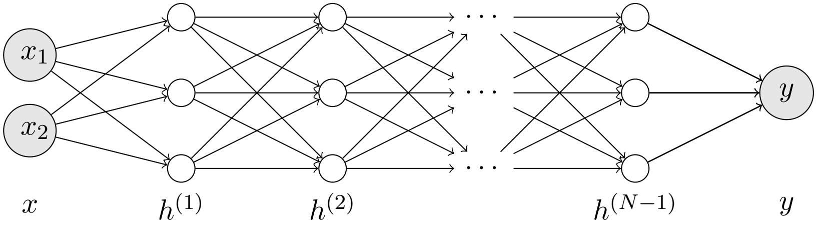
$$\begin{aligned} h^{(1)} &= f\left(W^{(1)} \cdot x + b^{(1)}\right) \\ h^{(2)} &= f\left(W^{(2)} \cdot h^{(1)} + b^{(2)}\right) \\ y &= f\left(W^{(3)} \cdot h^{(2)} + b^{(3)}\right) \end{aligned}$$

Multi-Layer Perceptron



$$
\begin{aligned}
h^{(1)} &= f\left(W^{(1)} \cdot x + b^{(1)}\right) \\
h^{(2)} &= f\left(W^{(2)} \cdot h^{(1)} + b^{(2)}\right) \\
\vdots \quad & \vdots \quad \vdots
\end{aligned}
$$

Multi-Layer Perceptron



$$
\begin{array}{rcl}
h^{(1)} & = & f\left(W^{(1)} \cdot x + b^{(1)}\right) \\
h^{(2)} & = & f\left(W^{(2)} \cdot h^{(1)} + b^{(2)}\right) \\
\vdots & \vdots & \vdots \\
y & = & f\left(W^{(N)} \cdot h^{(N-1)} + b^{(N)}\right)
\end{array}
$$

Activation Functions

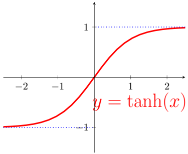
- tanh
- ReLU
- sigmoid
- softmax

There are many variants/alternative functions with different properties.

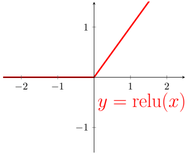
Must be continuous and differentiable (almost everywhere)

Activation Functions (hidden layers)

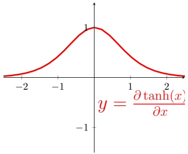
$$\tanh(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)}$$



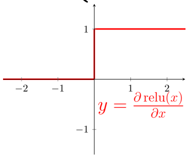
$$\mathrm{relu}(x) = \max(0, x)$$



$$\frac{\partial \tanh(x)}{\partial x} = 1 - \tanh^2(x)$$



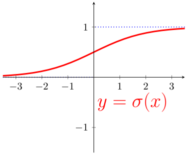
$$\frac{\partial\, \mathrm{relu}(x)}{\partial x} = \begin{cases} 1 & \text{if } \; x > 0 \\ 0 & \text{otherwise} \end{cases}$$



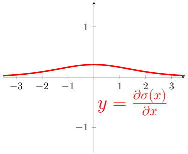
Activation Functions (hidden layers/output layers)

Logistic Sigmoid

$\sigma(x) = \frac{1}{1+\exp(-x)}$



$\frac{\partial \sigma(x)}{\partial x} = \sigma(x) \cdot (1 - \sigma(x))$



Softmax

$\sigma(x)_i = \frac{\exp(x_i)}{\sum_{i'=1}^{d} \exp(x_{i'})}$ where $x \in \mathbb{R}^d$

$$\frac{\partial \sigma(x)_i}{\partial x_j} = \begin{cases} \sigma(x)_i \cdot (1 - \sigma(x)_i) & \text{if } i = j \\ -\sigma(x)_i \cdot \sigma(x)_j & \text{if } i \neq j \end{cases}$$

Activation Functions (output layers)

- Output layer typically a sigmoid or softmax
- $a = W^{(N)} \cdot h^{(N-1)} + b^{(N)}$
- sigmoid:

$$p(Y = 1|X = x; \theta) = \sigma(a) = \frac{1}{1 + \exp(-a)}$$

$$p(Y = 0|X = x; \theta) = 1 - p(Y = 1|X = x; \theta)$$

- softmax:

$$p(Y = i|X = x; \theta) = \sigma(a)_i = \frac{\exp(a_i)}{\sum_{i'} \exp(a_{i'})}$$

Lesson Plan

linear models and feature design

multi-layer perceptron

optimization

feed-forward language model

Loss Functions/Objective Functions

Define the network, e.g.:

$$p(y|x;\theta) = \sigma(W^{(N)} \cdot \mathrm{relu}(W^{(N-1)} \cdot (\cdots) + b^{(N-1)}) + b^{N2)})$$

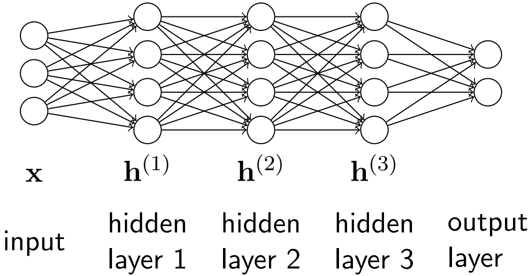
$$\theta = \left\{W^{(1)}, \ldots, W^{(N)}, b^{(1)}, \ldots, b^{(N)}\right\}$$

Cross Entropy

- Given a training dataset $\mathcal{D} = (x^{(i)}, y^{(i)})|_{i=1}^{N}$
- Multi-Class Cross Entropy loss:

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{i} \ln p(y^{(i)}|x^{(i)};\theta)$$

- Also referred to as the negative log likelihood.


$\mathbf{x}$
$\mathbf{h}^{(1)}$
$\mathbf{h}^{(2)}$
$\mathbf{h}^{(3)}$
input
hidden layer 1
hidden layer 2
hidden layer 3
output layer

Optimization

Learning of the network parameters θ is done by minimizing the loss function with respect to θ.

$$\min_\theta \mathcal{L}(\theta)$$

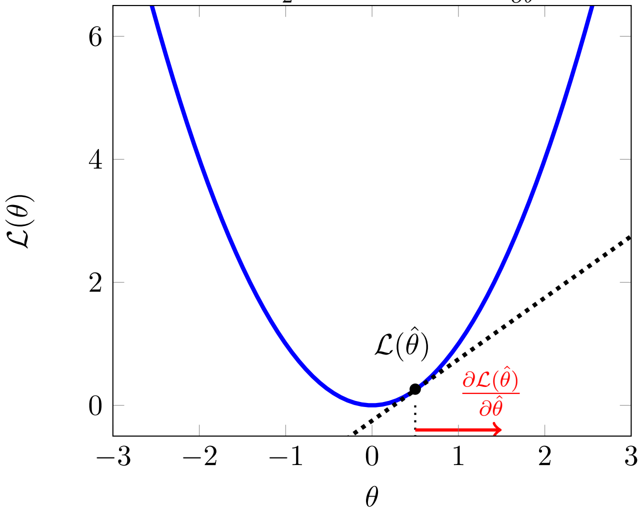
Typically, this is done by performing some variant of **stochastic gradient descent** (SGD).

Algorithm 1 Stochastic Gradient Descent

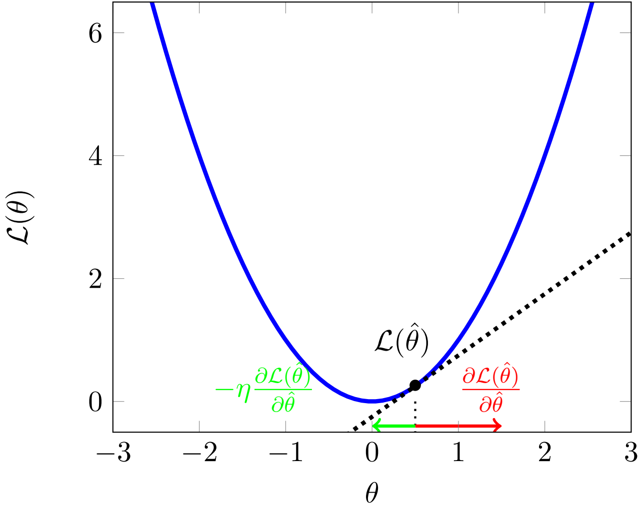
Randomly initialize θ.
for $\text{EPOCH} = 1$ to $\textsc{MaxEpochs}$ **do**
    Shuffle dataset $\mathcal{D} = (x^{(i)}, y^{(i)})|_{i=1}^{N}$
    for $i = 1$ to N **do**
        $\theta \leftarrow \theta - \eta \frac{\partial \mathcal{L}_i(\theta)}{\partial \theta}$
    end for
end for

η is the learning rate, typically a small value e.g. 10^{-3}

A simple case: $\mathcal{L}(\theta) = \frac{1}{2}(y_1 - \theta \cdot x_1)^2 \quad \frac{\partial \mathcal{L}(\theta)}{\partial \theta} = -x_1(y_1 - \theta \cdot x_1)$



A simple case: $\mathcal{L}(\theta) = \frac{1}{2}(y_1 - \theta \cdot x_1)^2 \quad \frac{\partial \mathcal{L}(\theta)}{\partial \theta} = -x_1(y_1 - \theta \cdot x_1)$



Backpropagation

To perform SGD, we need to efficiently compute $\frac{\partial \mathcal{L}_i(\theta)}{\partial \theta}$.

- Forward pass — compute the $\mathcal{L}_i(\theta)$ (e.g. the probability of y_i) given input x_i with the current θ.
  (Store intermediate outputs for backward pass)
- Backward pass — propagate the gradient of the loss backwards through the network, collecting the parameter gradients $\nabla\theta$

Chain Rule of Calculus

We want to compute the derivative of nested function $f(g(x))$ with respect to x.

By the chain rule:

$$\frac{df(g(x))}{dx} = \frac{df(g(x))}{dg(x)} \cdot \frac{dg(x)}{dx}$$

Chain Rule of Calculus

We want to compute the derivative of nested function $f(g(x))$ with respect to x.

By the chain rule:

$$\frac{df(g(x))}{dx} = \frac{df(g(x))}{dg(x)} \cdot \frac{dg(x)}{dx}$$

A concrete example:

$$\begin{aligned} f(z) &= \ln z \qquad & \tfrac{df(z)}{dz} &= \tfrac{1}{z} \\ g(x) &= 2x \qquad & \tfrac{dg(x)}{dx} &= 2 \end{aligned}$$

Chain Rule of Calculus

We want to compute the derivative of nested function $f(g(x))$ with respect to x.

By the chain rule:

$$\frac{df(g(x))}{dx} = \frac{df(g(x))}{dg(x)} \cdot \frac{dg(x)}{dx}$$

A concrete example:

$$\begin{aligned} f(z) &= \ln z \qquad & \tfrac{df(z)}{dz} &= \tfrac{1}{z} \\ g(x) &= 2x \qquad & \tfrac{dg(x)}{dx} &= 2 \end{aligned}$$

$$\frac{df(g(x))}{dx} = \frac{df(g(x))}{dg(x)} \cdot \frac{dg(x)}{dx}$$

Chain Rule of Calculus

We want to compute the derivative of nested function $f(g(x))$ with respect to x.

By the chain rule:

$$\frac{df(g(x))}{dx} = \frac{df(g(x))}{dg(x)} \cdot \frac{dg(x)}{dx}$$

A concrete example:

$$\begin{aligned} f(z) &= \ln z & \qquad \tfrac{df(z)}{dz} &= \tfrac{1}{z} \\ g(x) &= 2x & \qquad \tfrac{dg(x)}{dx} &= 2 \end{aligned}$$

$$\begin{aligned} \frac{df(g(x))}{dx} &= \frac{df(g(x))}{dg(x)} \cdot \frac{dg(x)}{dx} \\ &= \frac{1}{2x} \cdot 2 \end{aligned}$$

Chain Rule of Calculus

We want to compute the derivative of nested function $f(g(x))$ with respect to x.

By the chain rule:

$$\frac{df(g(x))}{dx} = \frac{df(g(x))}{dg(x)} \cdot \frac{dg(x)}{dx}$$

A concrete example:

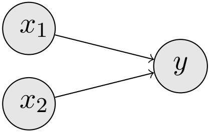
$$\begin{aligned} f(z) &= \ln z & \qquad \frac{df(z)}{dz} &= \frac{1}{z} \\ g(x) &= 2x & \qquad \frac{dg(x)}{dx} &= 2 \end{aligned}$$

$$\begin{aligned} \frac{df(g(x))}{dx} &= \frac{df(g(x))}{dg(x)} \cdot \frac{dg(x)}{dx} \\ &= \frac{1}{2x} \cdot 2 = \frac{1}{x} \end{aligned}$$

Backpropagation (Forward Pass)

Simple, 1-layer sigmoid network

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $o = p(Y = 1|x; w_1, w_2, b) = \sigma(a)$
- $\mathcal{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)})\}$
  where $x^{(1)}, x^{(2)} \in \mathbb{R}^2$ and $y^{(1)}, y^{(2)} \in \{0, 1\}$
- $y^{(1)} = 1$, $y^{(2)} = 0$



Backpropagation (Forward Pass)

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $o = p(Y = 1|x; w_1, w_2, b) = \sigma(a)$
- $\mathcal{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)})\}$
  where $x^{(1)}, x^{(2)} \in \mathbb{R}^2$ and $y^{(1)}, y^{(2)} \in \{0, 1\}$
- $y^{(1)} = 1$, $y^{(2)} = 0$

Forward Pass

Backpropagation (Forward Pass)

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $o = p(Y = 1|x; w_1, w_2, b) = \sigma(a)$
- $\mathcal{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)})\}$
  where $x^{(1)}, x^{(2)} \in \mathbb{R}^2$ and $y^{(1)}, y^{(2)} \in \{0, 1\}$
- $y^{(1)} = 1$, $y^{(2)} = 0$

Forward Pass

1. $a^{(1)} = w_1 \cdot x_1^{(1)} + w_2 \cdot x_2^{(1)} + b$

Backpropagation (Forward Pass)

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $o = p(Y = 1|x; w_1, w_2, b) = \sigma(a)$
- $\mathcal{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)})\}$
  where $x^{(1)}, x^{(2)} \in \mathbb{R}^2$ and $y^{(1)}, y^{(2)} \in \{0, 1\}$
- $y^{(1)} = 1$, $y^{(2)} = 0$

Forward Pass

1. $a^{(1)} = w_1 \cdot x_1^{(1)} + w_2 \cdot x_2^{(1)} + b$
2. $p(Y = 1|x^{(1)}) = \sigma(a^{(1)})$

Backpropagation (Forward Pass)

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $o = p(Y = 1 | x; w_1, w_2, b) = \sigma(a)$
- $\mathcal{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)})\}$
  where $x^{(1)}, x^{(2)} \in \mathbb{R}^2$ and $y^{(1)}, y^{(2)} \in \{0, 1\}$
- $y^{(1)} = 1$, $y^{(2)} = 0$

Forward Pass

1. $a^{(1)} = w_1 \cdot x_1^{(1)} + w_2 \cdot x_2^{(1)} + b$
2. $p(Y = 1 | x^{(1)}) = \sigma(a^{(1)})$
3. $p(y^{(1)} | x^{(1)}) = p(Y = 1 | x^{(1)})$

Backpropagation (Forward Pass)

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $o = p(Y = 1|x; w_1, w_2, b) = \sigma(a)$
- $\mathcal{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)})\}$
  where $x^{(1)}, x^{(2)} \in \mathbb{R}^2$ and $y^{(1)}, y^{(2)} \in \{0, 1\}$
- $y^{(1)} = 1$, $y^{(2)} = 0$

Forward Pass

1. $a^{(1)} = w_1 \cdot x_1^{(1)} + w_2 \cdot x_2^{(1)} + b$
2. $p(Y = 1|x^{(1)}) = \sigma(a^{(1)})$
3. $p(y^{(1)}|x^{(1)}) = p(Y = 1|x^{(1)})$
4. $a^{(2)} = w_1 \cdot x_1^{(2)} + w_2 \cdot x_2^{(2)} + b$

Backpropagation (Forward Pass)

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $o = p(Y = 1|x; w_1, w_2, b) = \sigma(a)$
- $\mathcal{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)})\}$
  where $x^{(1)}, x^{(2)} \in \mathbb{R}^2$ and $y^{(1)}, y^{(2)} \in \{0, 1\}$
- $y^{(1)} = 1$, $y^{(2)} = 0$

Forward Pass

1. $a^{(1)} = w_1 \cdot x_1^{(1)} + w_2 \cdot x_2^{(1)} + b$
2. $p(Y = 1|x^{(1)}) = \sigma(a^{(1)})$
3. $p(y^{(1)}|x^{(1)}) = p(Y = 1|x^{(1)})$
4. $a^{(2)} = w_1 \cdot x_1^{(2)} + w_2 \cdot x_2^{(2)} + b$
5. $p(Y = 1|x^{(2)}) = \sigma(a^{(2)})$

Backpropagation (Forward Pass)

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $o = p(Y = 1|x; w_1, w_2, b) = \sigma(a)$
- $\mathcal{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)})\}$
  where $x^{(1)}, x^{(2)} \in \mathbb{R}^2$ and $y^{(1)}, y^{(2)} \in \{0, 1\}$
- $y^{(1)} = 1,\ y^{(2)} = 0$

Forward Pass

1. $a^{(1)} = w_1 \cdot x_1^{(1)} + w_2 \cdot x_2^{(1)} + b$
2. $p(Y = 1|x^{(1)}) = \sigma(a^{(1)})$
3. $p(y^{(1)}|x^{(1)}) = p(Y = 1|x^{(1)})$
4. $a^{(2)} = w_1 \cdot x_1^{(2)} + w_2 \cdot x_2^{(2)} + b$
5. $p(Y = 1|x^{(2)}) = \sigma(a^{(2)})$
6. $p(y^{(2)}|x^{(2)}) = 1 - p(Y = 1|x^{(2)})$

Backpropagation (Forward Pass)

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $o = p(Y = 1|x; w_1, w_2, b) = \sigma(a)$
- $\mathcal{D} = \{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)})\}$
  where $x^{(1)}, x^{(2)} \in \mathbb{R}^2$ and $y^{(1)}, y^{(2)} \in \{0, 1\}$
- $y^{(1)} = 1,\ y^{(2)} = 0$

Forward Pass

1. $a^{(1)} = w_1 \cdot x_1^{(1)} + w_2 \cdot x_2^{(1)} + b$
2. $p(Y = 1|x^{(1)}) = \sigma(a^{(1)})$
3. $p(y^{(1)}|x^{(1)}) = p(Y = 1|x^{(1)})$
4. $a^{(2)} = w_1 \cdot x_1^{(2)} + w_2 \cdot x_2^{(2)} + b$
5. $p(Y = 1|x^{(2)}) = \sigma(a^{(2)})$
6. $p(y^{(2)}|x^{(2)}) = 1 - p(Y = 1|x^{(2)})$
7. $\mathcal{L} = -\frac{1}{2}\left[\ln p(y^{(1)}|x^{(1)}) + \ln p(y^{(2)}|x^{(2)})\right]$

Backpropagation (Backward Pass)

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $o = p(Y = 1|x; w_1, w_2, b) = \sigma(a)$

$$\frac{\partial \mathcal{L}}{\partial w_1} = - \frac{1}{2} \left[\frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial w_1} + \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial w_1} \right]$$

Backpropagation (Backward Pass)

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $o = p(Y = 1|x; w_1, w_2, b) = \sigma(a)$

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_1} &= -\frac{1}{2}\left[\frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial w_1} + \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial w_1}\right] \\
&= -\frac{1}{2}\left[\frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial p(y^{(1)}|x^{(1)})} \cdot \frac{\partial p(y^{(1)}|x^{(1)})}{\partial w_1} + \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial p(y^{(2)}|x^{(2)})} \cdot \frac{\partial p(y^{(2)}|x^{(2)})}{\partial w_1}\right]
\end{aligned}
$$

Backpropagation (Backward Pass)

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $o = p(Y = 1|x; w_1, w_2, b) = \sigma(a)$

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_1} &= -\frac{1}{2}\left[\frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial w_1} + \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial w_1}\right] \\
&= -\frac{1}{2}\left[\frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial p(y^{(1)}|x^{(1)})} \cdot \frac{\partial p(y^{(1)}|x^{(1)})}{\partial w_1} + \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial p(y^{(2)}|x^{(2)})} \cdot \frac{\partial p(y^{(2)}|x^{(2)})}{\partial w_1}\right] \\
&= -\frac{1}{2}\left[\begin{array}{c} \frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial p(y^{(1)}|x^{(1)})} \cdot \frac{\partial p(y^{(1)}|x^{(1)})}{\partial a^{(1)}} \cdot \frac{\partial a^{(1)}}{\partial w_1} \\ + \\ \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial p(y^{(2)}|x^{(2)})} \cdot \frac{\partial p(y^{(2)}|x^{(2)})}{\partial a^{(2)}} \cdot \frac{\partial a^{(2)}}{\partial w_1} \end{array}\right]
\end{aligned}
$$

Backpropagation (Backward Pass)

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $o = p(Y = 1|x; w_1, w_2, b) = \sigma(a)$

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_1} &= -\frac{1}{2}\left[\frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial w_1} + \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial w_1}\right] \\
&= -\frac{1}{2}\left[\frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial p(y^{(1)}|x^{(1)})} \cdot \frac{\partial p(y^{(1)}|x^{(1)})}{\partial w_1} + \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial p(y^{(2)}|x^{(2)})} \cdot \frac{\partial p(y^{(2)}|x^{(2)})}{\partial w_1}\right] \\
&= -\frac{1}{2}\left[\begin{array}{c} \frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial p(y^{(1)}|x^{(1)})} \cdot \frac{\partial p(y^{(1)}|x^{(1)})}{\partial a^{(1)}} \cdot \color{red}{\frac{\partial a^{(1)}}{\partial w_1}} \\ + \\ \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial p(y^{(2)}|x^{(2)})} \cdot \frac{\partial p(y^{(2)}|x^{(2)})}{\partial a^{(2)}} \cdot \color{red}{\frac{\partial a^{(2)}}{\partial w_1}} \end{array}\right]
\end{aligned}
$$

Backpropagation (Backward Pass)

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $o = p(Y = 1 | x; w_1, w_2, b) = \sigma(a)$

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_1} &= -\frac{1}{2}\left[\frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial w_1} + \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial w_1}\right] \\
&= -\frac{1}{2}\left[\frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial p(y^{(1)}|x^{(1)})} \cdot \frac{\partial p(y^{(1)}|x^{(1)})}{\partial w_1} + \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial p(y^{(2)}|x^{(2)})} \cdot \frac{\partial p(y^{(2)}|x^{(2)})}{\partial w_1}\right] \\
&= -\frac{1}{2}\left[\begin{array}{c} \frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial p(y^{(1)}|x^{(1)})} \cdot \frac{\partial p(y^{(1)}|x^{(1)})}{\partial a^{(1)}} \cdot \color{red}{x_1} \\ + \\ \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial p(y^{(2)}|x^{(2)})} \cdot \frac{\partial p(y^{(2)}|x^{(2)})}{\partial a^{(2)}} \cdot \color{red}{x_1} \end{array}\right]
\end{aligned}
$$

Backpropagation (Backward Pass)

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $o = p(Y = 1|x; w_1, w_2, b) = \sigma(a)$

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_1} &= -\frac{1}{2}\left[\frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial w_1} + \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial w_1}\right] \\
&= -\frac{1}{2}\left[\frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial p(y^{(1)}|x^{(1)})} \cdot \frac{\partial p(y^{(1)}|x^{(1)})}{\partial w_1} + \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial p(y^{(2)}|x^{(2)})} \cdot \frac{\partial p(y^{(2)}|x^{(2)})}{\partial w_1}\right] \\
&= -\frac{1}{2}\left[\begin{array}{c} \frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial p(y^{(1)}|x^{(1)})} \cdot \frac{\partial p(y^{(1)}|x^{(1)})}{\partial a^{(1)}} \cdot x_1 \\ + \\ \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial p(y^{(2)}|x^{(2)})} \cdot \frac{\partial p(y^{(2)}|x^{(2)})}{\partial a^{(2)}} \cdot x_1 \end{array}\right]
\end{aligned}
$$

Backpropagation (Backward Pass)

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $o = p(Y = 1|x; w_1, w_2, b) = \sigma(a)$

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_1} &= -\frac{1}{2}\left[\frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial w_1} + \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial w_1}\right] \\
&= -\frac{1}{2}\left[\frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial p(y^{(1)}|x^{(1)})} \cdot \frac{\partial p(y^{(1)}|x^{(1)})}{\partial w_1} + \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial p(y^{(2)}|x^{(2)})} \cdot \frac{\partial p(y^{(2)}|x^{(2)})}{\partial w_1}\right] \\
&= -\frac{1}{2}\left[\begin{array}{c} \frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial p(y^{(1)}|x^{(1)})} \cdot \sigma(a^{(1)})(1-\sigma(a^{(1)})) \cdot x_1 \\ + \\ \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial p(y^{(2)}|x^{(2)})} \cdot -\sigma(a^{(2)})(1-\sigma(a^{(2)})) \cdot x_1 \end{array}\right]
\end{aligned}
$$

Backpropagation (Backward Pass)

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $o = p(Y = 1|x; w_1, w_2, b) = \sigma(a)$

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_1} &= -\frac{1}{2}\left[\frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial w_1} + \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial w_1}\right] \\
&= -\frac{1}{2}\left[\frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial p(y^{(1)}|x^{(1)})} \cdot \frac{\partial p(y^{(1)}|x^{(1)})}{\partial w_1} + \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial p(y^{(2)}|x^{(2)})} \cdot \frac{\partial p(y^{(2)}|x^{(2)})}{\partial w_1}\right] \\
&= -\frac{1}{2}\left[\begin{array}{c} \frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial p(y^{(1)}|x^{(1)})} \cdot \sigma(a^{(1)})(1-\sigma(a^{(1)})) \cdot x_1 \\ + \\ \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial p(y^{(2)}|x^{(2)})} \cdot -\sigma(a^{(2)})(1-\sigma(a^{(2)})) \cdot x_1 \end{array}\right]
\end{aligned}
$$

Backpropagation (Backward Pass)

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $o = p(Y = 1|x; w_1, w_2, b) = \sigma(a)$

$$\frac{\partial \mathcal{L}}{\partial w_1} = -\frac{1}{2}\left[\frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial w_1} + \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial w_1}\right]$$

$$= -\frac{1}{2}\left[\frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial p(y^{(1)}|x^{(1)})} \cdot \frac{\partial p(y^{(1)}|x^{(1)})}{\partial w_1} + \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial p(y^{(2)}|x^{(2)})} \cdot \frac{\partial p(y^{(2)}|x^{(2)})}{\partial w_1}\right]$$

$$= -\frac{1}{2}\left[\begin{array}{c} \frac{1}{\sigma(a^{(1)})} \cdot \sigma(a^{(1)})(1 - \sigma(a^{(1)})) \cdot x_1 \\ + \\ \frac{1}{1-\sigma(a^{(2)})} \cdot -\sigma(a^{(2)})(1 - \sigma(a^{(2)})) \cdot x_1 \end{array}\right]$$

Backpropagation (Backward Pass)

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $o = p(Y = 1|x; w_1, w_2, b) = \sigma(a)$

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w_1} &= -\frac{1}{2}\left[\frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial w_1} + \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial w_1}\right] \\
&= -\frac{1}{2}\left[\frac{\partial \ln p(y^{(1)}|x^{(1)})}{\partial p(y^{(1)}|x^{(1)})} \cdot \frac{\partial p(y^{(1)}|x^{(1)})}{\partial w_1} + \frac{\partial \ln p(y^{(2)}|x^{(2)})}{\partial p(y^{(2)}|x^{(2)})} \cdot \frac{\partial p(y^{(2)}|x^{(2)})}{\partial w_1}\right] \\
&= -\frac{1}{2}\left[\begin{array}{c} \frac{1}{\sigma(a^{(1)})} \cdot \sigma(a^{(1)})(1-\sigma(a^{(1)})) \cdot x_1 \\ + \\ \frac{1}{1-\sigma(a^{(2)})} \cdot -\sigma(a^{(2)})(1-\sigma(a^{(2)})) \cdot x_1 \end{array}\right] \\
&= -\frac{1}{2}\left[(1-\sigma(a^{(1)})) \cdot x_1 - \sigma(a^{(2)}) \cdot x_1\right]
\end{aligned}
$$

Backpropagation (Backward Pass)

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $o = p(Y = 1|x; w_1, w_2, b) = \sigma(a)$

$$\frac{\partial \mathcal{L}}{\partial w_1} = -\frac{1}{2}\left[(1 - \sigma(a^{(1)})) \cdot x_1 - \sigma(a^{(2)}) \cdot x_1\right]$$

Backpropagation (Backward Pass)

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $o = p(Y = 1|x; w_1, w_2, b) = \sigma(a)$

$$\frac{\partial \mathcal{L}}{\partial w_1} = -\frac{1}{2}\left[(1 - \sigma(a^{(1)})) \cdot x_1 - \sigma(a^{(2)}) \cdot x_1\right]$$

$w_1 \leftarrow w_1 - \eta \frac{\partial \mathcal{L}}{\partial w_1}$

Backpropagation (Backward Pass)

- $a = w_1 \cdot x_1 + w_2 \cdot x_2 + b$
- $o = p(Y = 1|x; w_1, w_2, b) = \sigma(a)$

$$\frac{\partial \mathcal{L}}{\partial w_1} = -\frac{1}{2}\left[(1 - \sigma(a^{(1)})) \cdot x_1 - \sigma(a^{(2)}) \cdot x_1\right]$$

$w_1 \leftarrow w_1 - \eta \frac{\partial \mathcal{L}}{\partial w_1}$

Repeat for w_2 and b

Optimization

Algorithm 2 Minibatch Stochastic Gradient Descent

```
Randomly initialize θ.
Set batch size s (e.g., s = 64)
for EPOCH = 1 to MAXEPOCHS do
    Shuffle dataset D = (x^(i), y^(i))|_{i=1}^N
    for i = 1 to N/s do
        Sample s datapoints (x^(i), y^(i)) ~ D without replacement
        θ ← θ − η (1/s) (Σ_{j=1}^s ∂L_j(θ)/∂θ)
    end for
end for
```

η is the learning rate, typically a small value e.g. 10^{-3}

Weight Initialization

How to initialize weights?

$$W \sim \mathrm{Normal}(0, 1)$$

$$W \sim \mathrm{Uniform}(-1, 1)$$

Weight Initialization

How to initialize weights?

$$W \sim \mathrm{Normal}(0, 1)$$

$$W \sim \mathrm{Uniform}(-1, 1)$$

Xavier Initialization (Glorot and Bengio, 2010)

$$W \sim \mathrm{Normal}(0, \frac{2}{\mathrm{fan\_in} + \mathrm{fan\_out}})$$

$$W \sim \mathrm{Uniform}(\pm\sqrt{\frac{6}{\mathrm{fan\_in} + \mathrm{fan\_out}}})$$

Weight Initialization

How to initialize weights?

$$W \sim \mathrm{Normal}(0, 1)$$

$$W \sim \mathrm{Uniform}(-1, 1)$$

Xavier Initialization (Glorot and Bengio, 2010)

$$W \sim \mathrm{Normal}(0, \frac{2}{\mathrm{fan\_in} + \mathrm{fan\_out}})$$

$$W \sim \mathrm{Uniform}(\pm\sqrt{\frac{6}{\mathrm{fan\_in} + \mathrm{fan\_out}}})$$

Set biases to 0.

Optimization tricks

- When implementing a model, try to fit to 100% accuracy on 1 or 2 data points.
- Decrease the learning rate with each epoch, or when the loss stops decreasing on validation data.
- Find a good initial learning rate before adjusting other hyperparameters.
- Train with dropout. (Great for image classification; YMMV for NLP)
- Even better use a different optimizer:
  - SGD with momentum or Nesterov accelerated gradient
  - rmsprop
  - adagrad
  - adadelta
  - adam

All of these take the parameter gradient as input.
For a good overview of these methods, see
`http://ruder.io/optimizing-gradient-descent/`

Lesson Plan

linear models and feature design

multi-layer perceptron

optimization

feed-forward language model

Language Modeling and you

A *language model* assigns a probability to an arbitrary sequence of word tokens.

Often used in speech recognition and machine translation.

Typically, lm's make a low-order Markov assumption.

$$p(the,\ \ werewolf,\ \ howled,\ \ at,\ \ the,\ \ moon) =$$

$$\begin{aligned} & p(the) \\ \times\ & p(werewolf|the) \\ \times\ & p(howled|the, werewolf) \\ \times\ & p(at|the,\ \ werewolf,\ \ howled) \\ \times\ & p(the|werewolf,\ \ howled,\ \ at) \\ \times\ & p(moon|howled,\ \ at,\ \ the) \end{aligned}$$

Language Modeling and you

Traditionally, the design of *ngram language models* focused on estimating terms like $p(moon|howled, \ \ at, \ \ the)$ by:

- counting occurrence of ngrams
  $(howled, \ \ at, \ \ the, \ \ moon)$,
  i$(howled, \ \ at, \ \ the, \ \ sky)$,
  i$(howled, \ \ at, \ \ the, \ \ *)$,
- Naive approach:
  $p(moon|howled, at, the) = \frac{\text{count}(howled,at,the,moon)}{\text{count}(howled,at,the,*)}$
- interpolating lower order models built on these counts

Unfortunately, these counts are sparse (especially beyond trigrams)

Observing $(barked, \ \ at, \ \ the, \ \ moon)$ doesn't tell us much about $(howled, \ \ at, \ \ the, \ \ moon)$

A Feedforward Language Model

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

The main ideas (copied verbatim from the paper):

1. associate with each word in the vocabulary a distributed *word feature vector* (a real-valued vector in $\mathbb{R}^m$),
2. express the joint *probability function* of word sequences in terms of the feature vectors of these words in the sequence, and
3. learn simultaneously the *word feature vectors* and the parameters of that *probability function*.

loss $\mathcal{L} = -\ln o_y$

Softmax Layer (output)

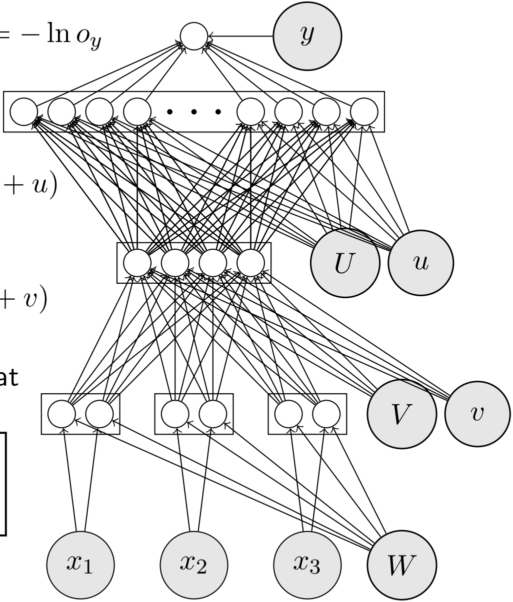
$o = \mathrm{softmax}(Uh^{(2)} + u)$

Hidden Layer

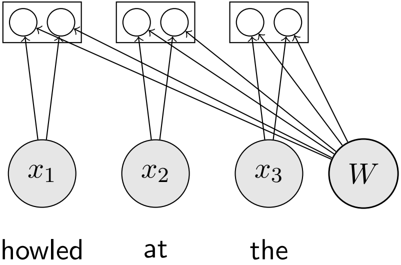
$h^{(2)} = \mathrm{relu}(Vh^{(1)} + v)$

Lookup + Concat Layer

$$h^{(1)} = \begin{bmatrix} Wx_1 \\ Wx_2 \\ Wx_3 \end{bmatrix}$$



Lookup and Concat Layer (View 1)



$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{bmatrix} \quad \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \end{bmatrix} \quad \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix}$$

Each word is encoded as a one-hot vector $x_i \in \mathbb{R}^V$.

Word embeddings $W \in \mathbb{R}^{m \times V}$

Lookup and Concat Layer (View 1)

$$\begin{array}{ccccc} \begin{bmatrix} W_{1,1} & W_{1,2} & \dots & W_{1,V} \\ W_{2,1} & W_{2,2} & \dots & W_{2,V} \\ \vdots & \vdots & \ddots & \vdots \\ W_{m-1,1} & W_{m-1,2} & \dots & W_{m-1,V} \\ W_{m,1} & W_{m,2} & \dots & W_{m,V} \end{bmatrix} & \times & \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix} & = & \begin{bmatrix} W_{1,2} \\ W_{2,2} \\ \vdots \\ W_{m-1,2} \\ W_{m,2} \end{bmatrix} \\ W & \cdot & x_1 & = & W_{:,2} \end{array}$$

Lookup and Concat Layer (View 1)

$$\begin{bmatrix} W_{1,1} & \color{red}{W_{1,2}} & \cdots & W_{1,V} \\ W_{2,1} & \color{red}{W_{2,2}} & \cdots & W_{2,V} \\ \vdots & \color{red}{\vdots} & \ddots & \vdots \\ W_{m-1,1} & \color{red}{W_{m-1,2}} & \cdots & W_{m-1,V} \\ W_{m,1} & \color{red}{W_{m,2}} & \cdots & W_{m,V} \end{bmatrix} \times \begin{bmatrix} 0 \\ \color{red}{1} \\ \vdots \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} W_{1,2} \\ W_{2,2} \\ \vdots \\ W_{m-1,2} \\ W_{m,2} \end{bmatrix}$$

$$W \quad \cdot \quad x_1 \quad = \quad W_{:,2}$$

Lookup and Concat Layer (View 1)

Each individual embedding is then concatenated into a larger single vector.

$$h^{(1)} = \begin{bmatrix} W \cdot x_1 \\ W \cdot x_2 \\ W \cdot x_3 \end{bmatrix}$$

Lookup and Concat Layer (View 1)

Each individual embedding is then concatenated into a larger single vector.

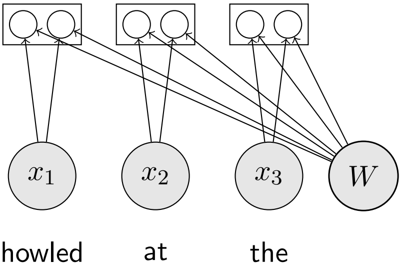
$$h^{(1)} = \begin{bmatrix} W \cdot x_1 \\ W \cdot x_2 \\ W \cdot x_3 \end{bmatrix} = \begin{bmatrix} W_{:,2} \\ W_{:,3} \\ W_{:,1} \end{bmatrix}$$

Lookup and Concat Layer (View 1)

Each individual embedding is then concatenated into a larger single vector.

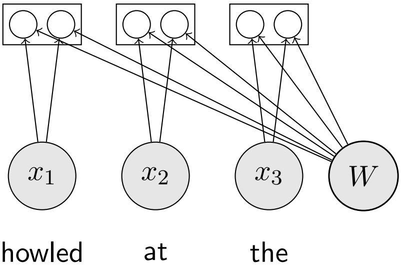
$$h^{(1)} = \begin{bmatrix} W \cdot x_1 \\ W \cdot x_2 \\ W \cdot x_3 \end{bmatrix} = \begin{bmatrix} W_{:,2} \\ W_{:,3} \\ W_{:,1} \end{bmatrix} = \begin{bmatrix} W_{1,2} \\ \vdots \\ W_{m,2} \\ W_{1,3} \\ \vdots \\ W_{m,3} \\ W_{1,1} \\ \vdots \\ W_{m,1} \end{bmatrix}$$

Lookup and Concat Layer (View 2)



$$x_1 = \text{index}(howled) = 2$$
$$x_2 = \text{index}(at) = 3$$
$$x_3 = \text{index}(the) = 1$$

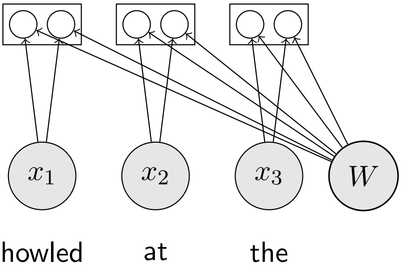
Lookup and Concat Layer (View 2)



$$x_1 = \text{index}(howled) = 2$$
$$x_2 = \text{index}(at) = 3$$
$$x_3 = \text{index}(the) = 1$$

$$\text{lookup}(W, i) = W_{:,i}$$

Lookup and Concat Layer (View 2)



$$x_1 = \text{index}(\mathit{howled}) = 2$$
$$x_2 = \text{index}(\mathit{at}) = 3$$
$$x_3 = \text{index}(\mathit{the}) = 1$$

$$\text{lookup}(W, i) = W_{:,i}$$

$$h^{(1)} = \begin{bmatrix} \text{lookup}(W, x_1) \\ \text{lookup}(W, x_2) \\ \text{lookup}(W, x_3) \end{bmatrix} = \begin{bmatrix} W_{:,2} \\ W_{:,3} \\ W_{:,1} \end{bmatrix}$$

loss $\mathcal{L} = -\ln o_y$

Softmax Layer (output)

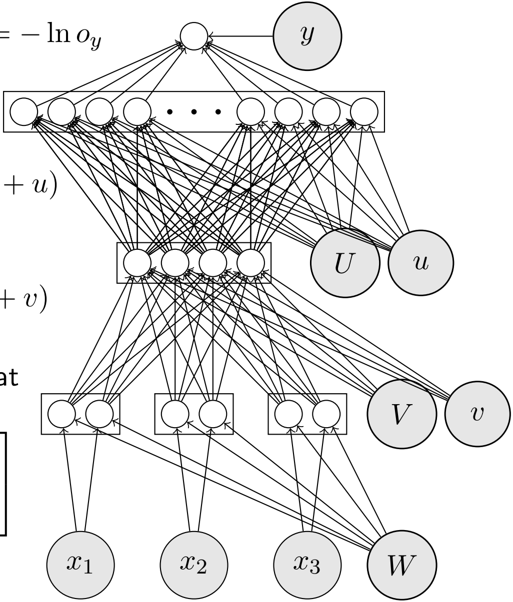
$o = \mathrm{softmax}(Uh^{(2)} + u)$

Hidden Layer

$h^{(2)} = \mathrm{relu}(Vh^{(1)} + v)$

Lookup + Concat Layer

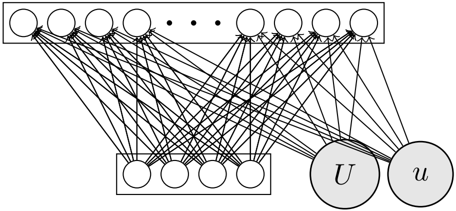
$$h^{(1)} = \begin{bmatrix} Wx_1 \\ Wx_2 \\ Wx_3 \end{bmatrix}$$



Softmax Layer

Softmax Layer
(output)
$o = \text{softmax}(Uh^{(2)} + u)$

Hidden Layer $h^{(2)}$



$h^{(2)} \in \mathbb{R}^d$, encoding of input word prefix into a vector space

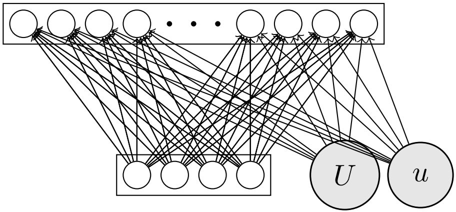
The ouput layer $o \in (0,1)^V$ contains one neuron (unit) for every word in the vocabulary.

o_i represents the probability of the i-th word in the vocabulary occurring after the word prefix represented by (x_1, x_2, x_3).

Softmax Layer

Softmax Layer
(output)
$o = \mathrm{softmax}(Uh^{(2)} + u)$

Hidden Layer $h^{(2)}$

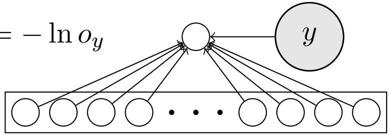


$U \in \mathbb{R}^{V \times d}$ is also a matrix of word embeddings.

Loss Layer

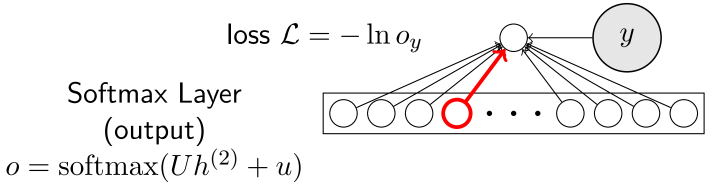
loss $\mathcal{L} = -\ln o_y$

Softmax Layer
(output)
$o = \text{softmax}(Uh^{(2)} + u)$



To compute the negative log likelihood (cross entropy loss), simply pick out the y-th element of o and take the negative log.

Loss Layer



To compute the negative log likelihood (cross entropy loss), simply pick out the y-th element of o and take the negative log.

E.g. $y = 4$, $-\ln o_y = -\ln o_4$

loss $\mathcal{L} = -\ln o_y$

Softmax Layer (output)

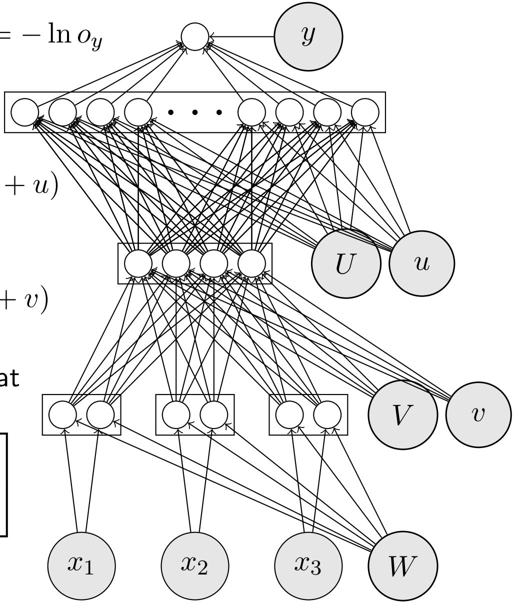
$$o = \mathrm{softmax}(Uh^{(2)} + u)$$

Hidden Layer

$$h^{(2)} = \mathrm{relu}(Vh^{(1)} + v)$$

Lookup + Concat Layer

$$h^{(1)} = \begin{bmatrix} Wx_1 \\ Wx_2 \\ Wx_3 \end{bmatrix}$$



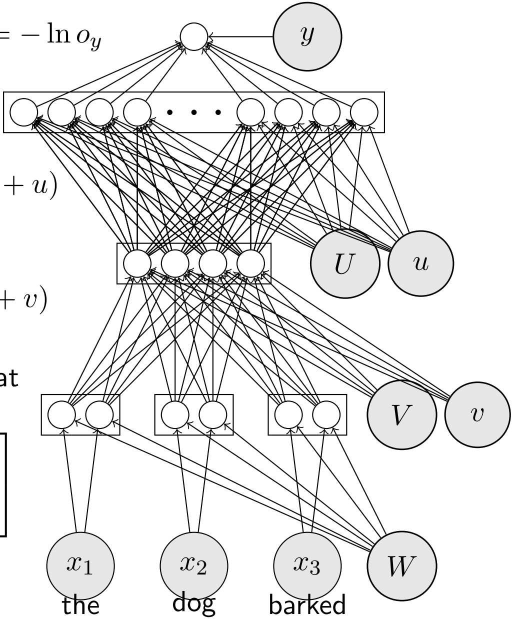
loss $\mathcal{L} = -\ln o_y$

Softmax Layer (output)
$o = \mathrm{softmax}(Uh^{(2)} + u)$

Hidden Layer
$h^{(2)} = \mathrm{relu}(Vh^{(1)} + v)$

Lookup + Concat Layer

$$h^{(1)} = \begin{bmatrix} Wx_1 \\ Wx_2 \\ Wx_3 \end{bmatrix}$$



loss $\mathcal{L} = -\ln o_y$

Softmax Layer (output)

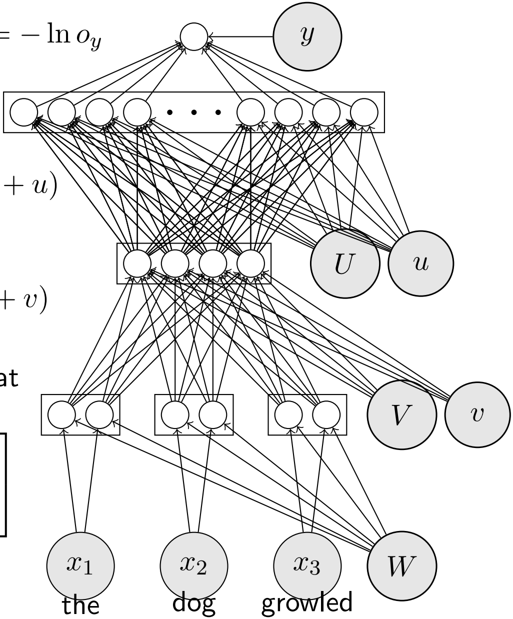
$o = \mathrm{softmax}(Uh^{(2)} + u)$

Hidden Layer

$h^{(2)} = \mathrm{relu}(Vh^{(1)} + v)$

Lookup + Concat Layer

$$h^{(1)} = \begin{bmatrix} Wx_1 \\ Wx_2 \\ Wx_3 \end{bmatrix}$$



loss $\mathcal{L} = -\ln o_y$

Softmax Layer (output)

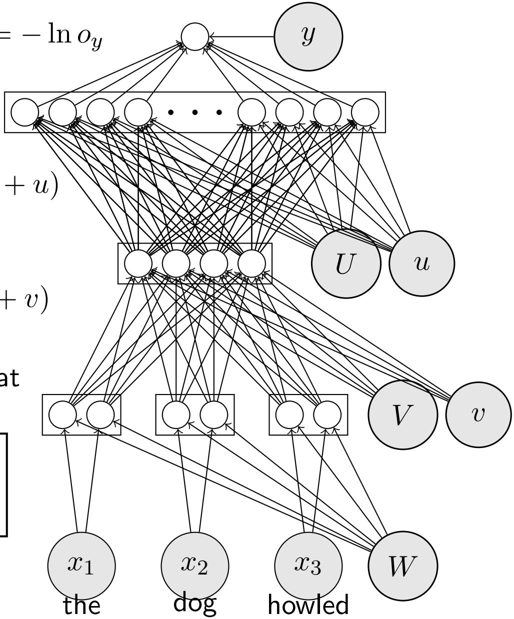
$$o = \mathrm{softmax}(Uh^{(2)} + u)$$

Hidden Layer

$$h^{(2)} = \mathrm{relu}(Vh^{(1)} + v)$$

Lookup + Concat Layer

$$h^{(1)} = \begin{bmatrix} Wx_1 \\ Wx_2 \\ Wx_3 \end{bmatrix}$$

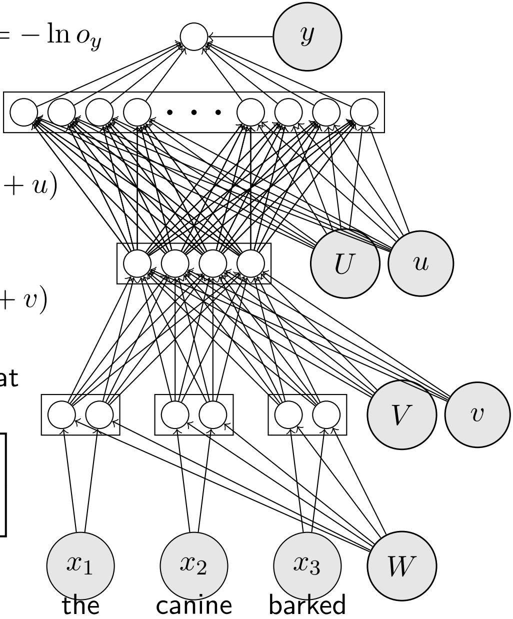


loss $\mathcal{L} = -\ln o_y$

Softmax Layer
(output)
$o = \mathrm{softmax}(Uh^{(2)} + u)$

Hidden Layer
$h^{(2)} = \mathrm{relu}(Vh^{(1)} + v)$

Lookup + Concat
Layer
$$h^{(1)} = \begin{bmatrix} Wx_1 \\ Wx_2 \\ Wx_3 \end{bmatrix}$$



loss $\mathcal{L} = -\ln o_y$

Softmax Layer (output)

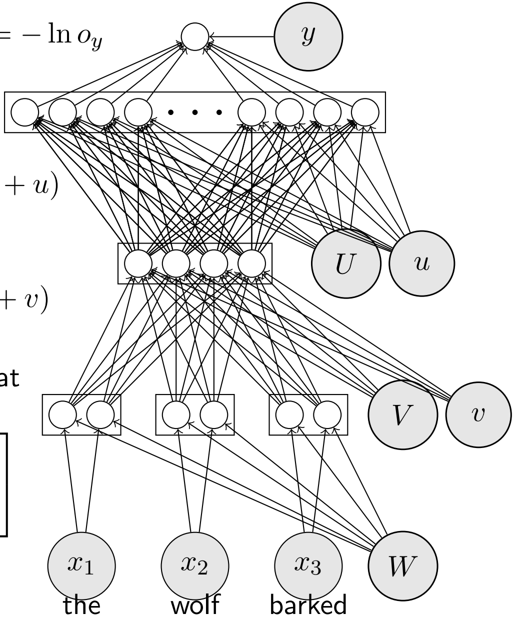
$$o = \text{softmax}(Uh^{(2)} + u)$$

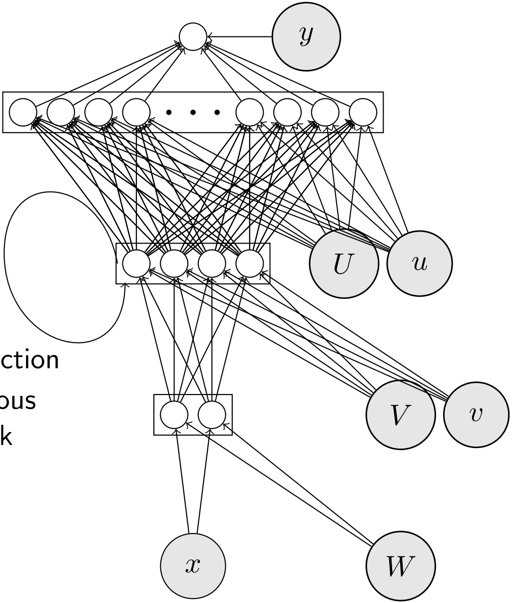
Hidden Layer

$$h^{(2)} = \text{relu}(Vh^{(1)} + v)$$

Lookup + Concat Layer

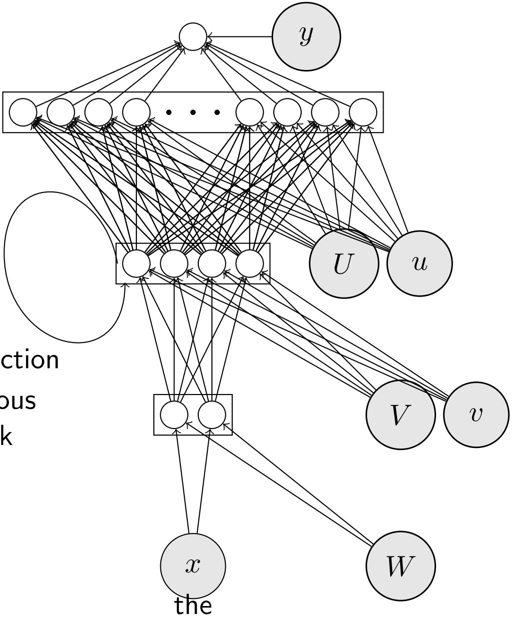
$$h^{(1)} = \begin{bmatrix} Wx_1 \\ Wx_2 \\ Wx_3 \end{bmatrix}$$





Recurrent Connection

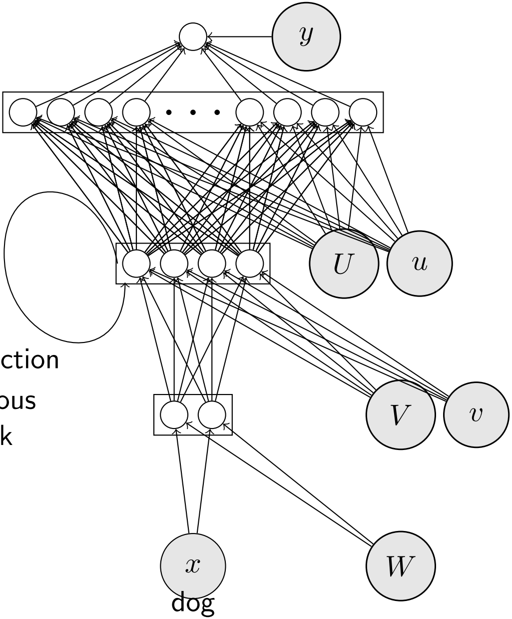
$h^{(2)}$ from previous word is fed back into the hidden layer

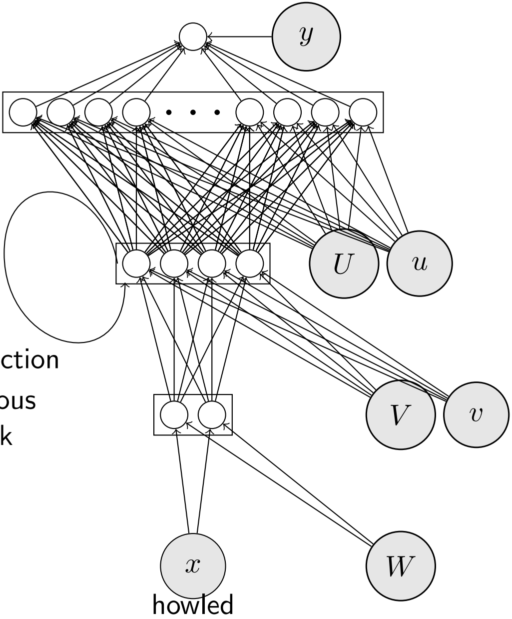


Recurrent Connection
$h^{(2)}$ from previous word is fed back into the hidden layer

Recurrent Connection

$h^{(2)}$ from previous word is fed back into the hidden layer



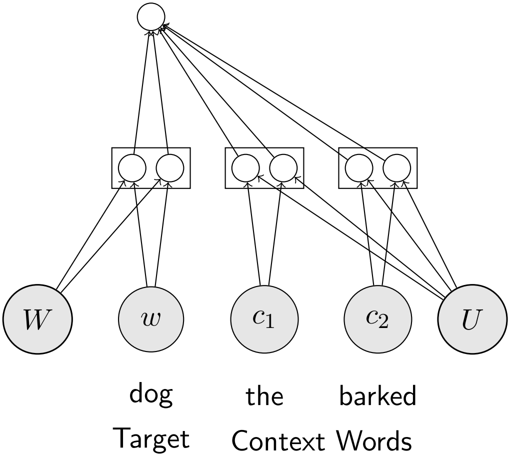


Recurrent Connection

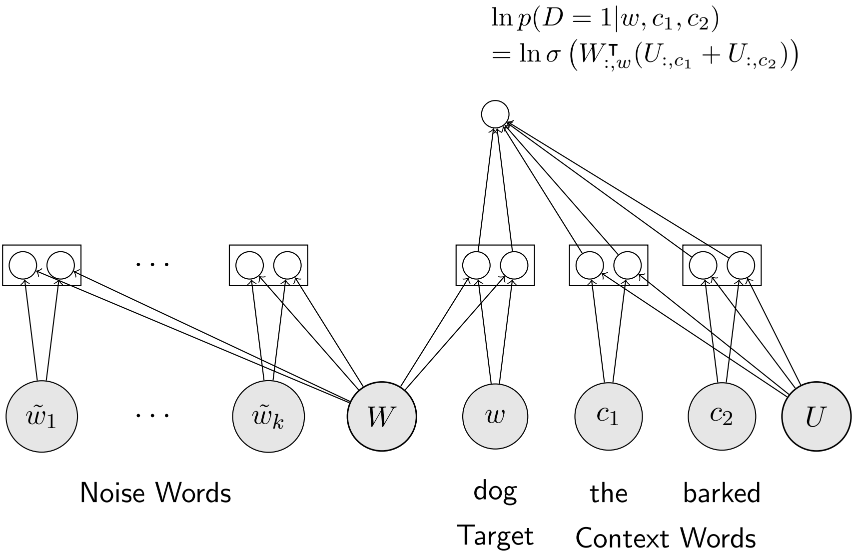
$h^{(2)}$ from previous word is fed back into the hidden layer

CBOW

$$\ln p(D=1|w,c_1,c_2)$$
$$= \ln \sigma \left(W_{:,w}^{\intercal}(U_{:,c_1} + U_{:,c_2}) \right)$$



CBOW

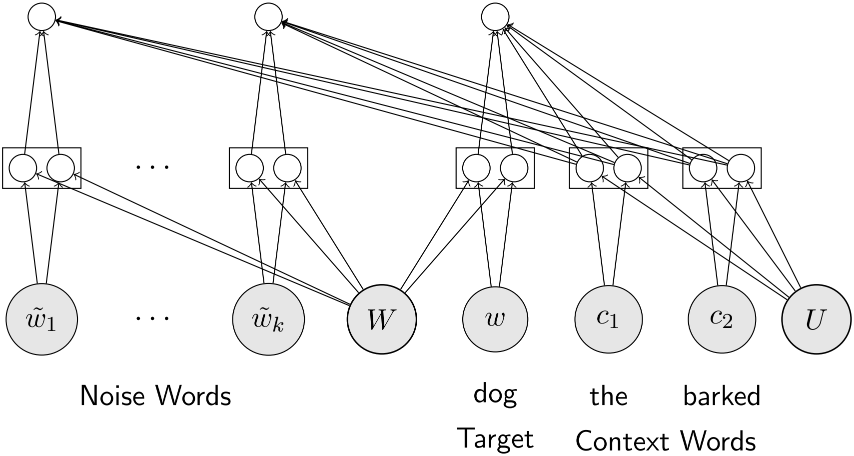


$\ln p(D = 1|w, c_1, c_2)$
$= \ln \sigma \left(W_{:,w}^\intercal (U_{:,c_1} + U_{:,c_2})\right)$
$\tilde{w}_1$
$\cdots$
$\tilde{w}_k$
W
w
c_1
c_2
U
Noise Words
dog
the
barked
Target
Context Words

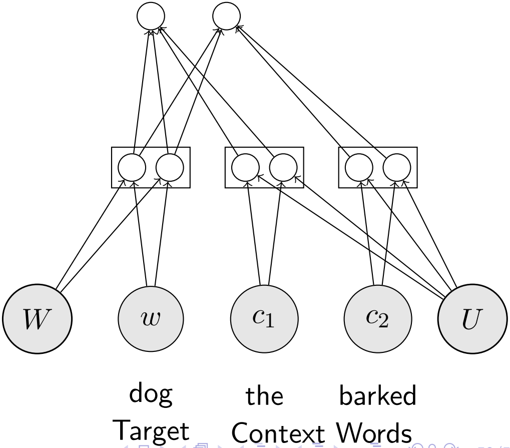
CBOW

$\sum_{i=1}^{k} \ln p(D = 0 | \tilde{w}_i, c_1, c_2)$
$= \sum_{i=1}^{k} \ln \left(1 - \sigma \left(W_{:,\tilde{w}_i}^{\intercal} \left(U_{:,c_1} + U_{:,c_2}\right)\right)\right)$

$\ln p(D = 1 | w, c_1, c_2)$
$= \ln \sigma \left(W_{:,w}^{\intercal}(U_{:,c_1} + U_{:,c_2})\right)$

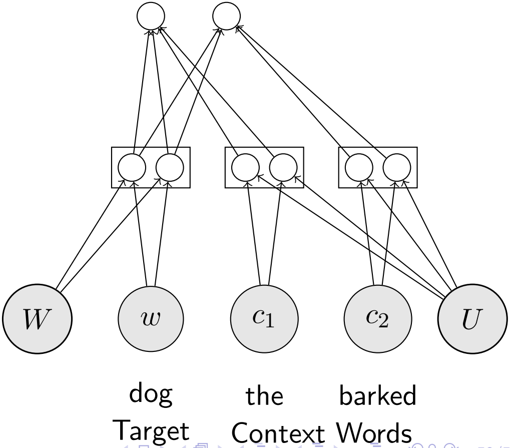


Skip-Gram



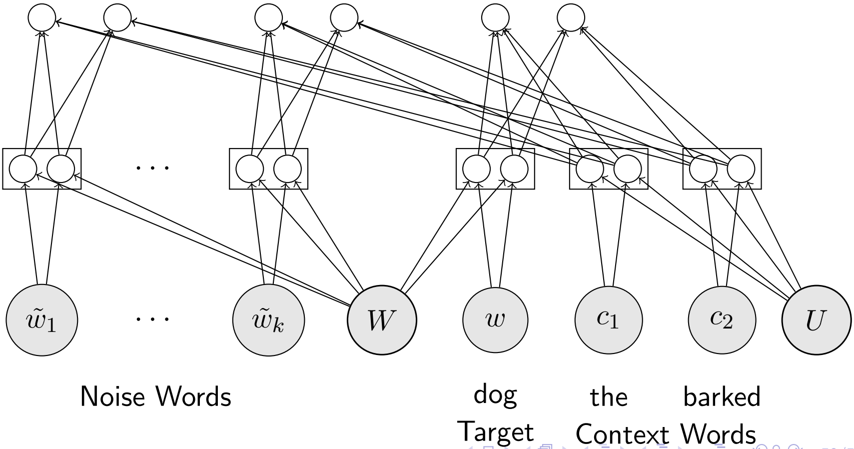
Skip-Gram

$$\ln p(D = 1|w, c_1, c_2)$$
$$= \ln p(D = 1|w, c_1) + \ln p(D = 1|w, c_2)$$
$$= \sum_{i=1}^{2} \ln \sigma \left(W_{:,w}^{\intercal} U_{:,c_i} \right)$$



Skip-Gram

$$\ln p(D = 1|w, c_1, c_2)$$
$$= \ln p(D = 1|w, c_1) + \ln p(D = 1|w, c_2)$$
$$= \sum_{i=1}^{2} \ln \sigma \left(W_{:,w}^{\intercal} U_{:,c_i}\right)$$



Skip-Gram

$\sum_{i=1}^{k} \ln p(D=0|\tilde{w}_i, c_1, c_2)$

$= \sum_{i=1}^{k} \ln p(D=0|\tilde{w}_i, c_1) + \ln p(D=0|\tilde{w}_i, c_2)$

$= \sum_{i=1}^{k} \sum_{j=1}^{2} \ln \left(1 - \sigma\left(W_{:,\tilde{w}_i}^{\intercal} U_{:,c_j}\right)\right)$

$\ln p(D=1|w, c_1, c_2)$

$= \ln p(D=1|w, c_1) + \ln p(D=1|w, c_2)$

$= \sum_{i=1}^{2} \ln \sigma\left(W_{:,w}^{\intercal} U_{:,c_i}\right)$