ONLINE MASTERS IN **DATA SCIENCE**

DSC 255 - MACHINE LEARNING FUNDAMENTALS

TRAINING NEURAL NETS BY BACKPROPAGATION

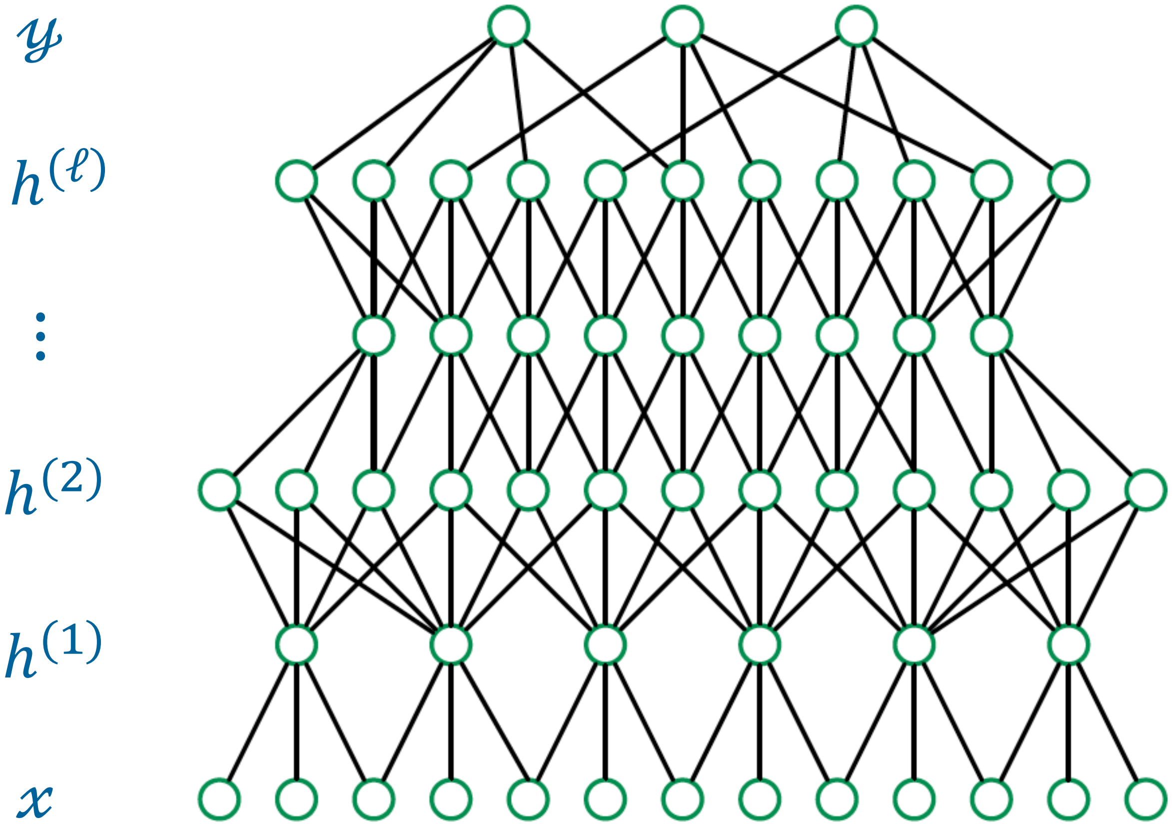
SANJOY DASGUPTA, PROFESSOR

UC San Diego

COMPUTER SCIENCE & ENGINEERING

HALICIOĞLU DATA SCIENCE INSTITUTE

Feedforward Neural Nets



Learning a Net: the Loss Function

Classification problem with k labels.

- Parameters of entire net: W
- For any input x, net computes probabilities of labels:

$$Pr_W(label = j|x)$$

Learning a Net: the Loss Function

Classification problem with k labels.

- Parameters of entire net: W
- For any input x, net computes probabilities of labels:

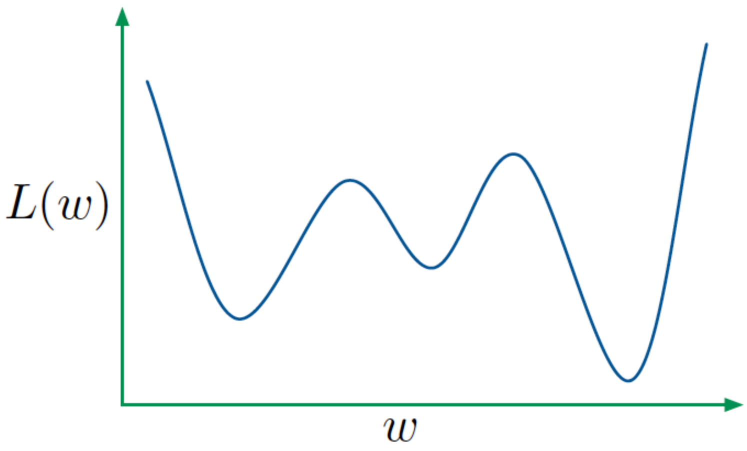
$$Pr_W(label = j|x)$$

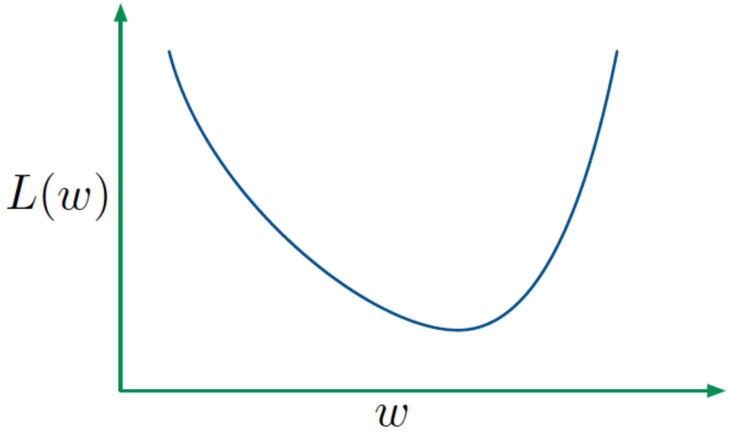
- Given data set $\left(x^{(1)}, y^{(1)}\right), \ldots, \left(x^{(n)}, y^{(n)}\right)$, loss function:

$$L(W) = -\sum_{i=1}^{n} In\, Pr_w\left(y^{(i)} \middle| x^{(i)}\right)$$

(also called **cross-entropy**).

Nature of the Loss Function





Variants of Gradient Descent

Initialize W and then repeatedly update.

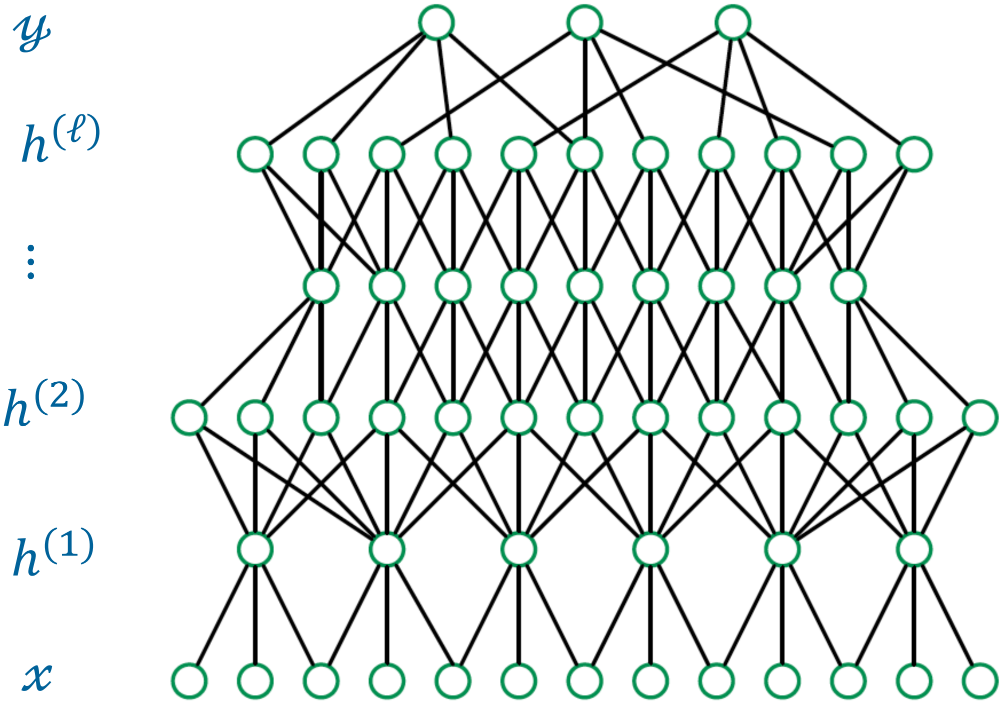
1. Gradient descent
   Each update involves the entire training set.

2. Stochastic gradient descent
   Each update involves a single data point.

3. Mini-batch stochastic gradient descent
   Each update involves a modest, fixed number of data points.

Derivative of the Loss Function

Update for a specific parameter: derivative of loss function wrt that parameter.



Chain Rule

1. Suppose $h(x) = g\big(f(x)\big)$, where $x \in \mathbb{R}$ and $f, g: \mathbb{R} \longrightarrow \mathbb{R}$.
   Then: $h'(x) = g'\big(f(x)\big)f'(x)$

Chain Rule

1. Suppose $h(x) = g\big(f(x)\big)$, where $x \in \mathbb{R}$ and $f, g\colon \mathbb{R} \longrightarrow \mathbb{R}$.
   Then: $h'(x) = g'\big(f(x)\big)f'(x)$

2. Suppose z is a function of y, which is a function of x.

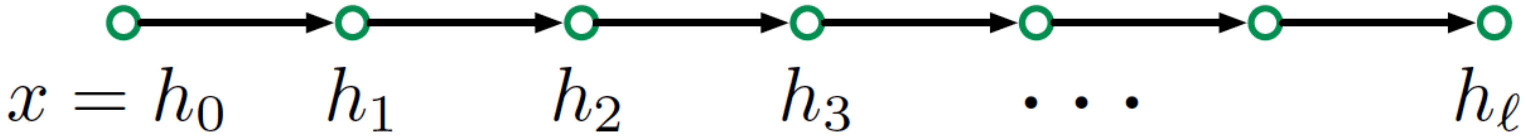
$$x \longrightarrow y \longrightarrow z$$

Then:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

A Single Chain of Nodes

A neural net with one node per hidden layer:

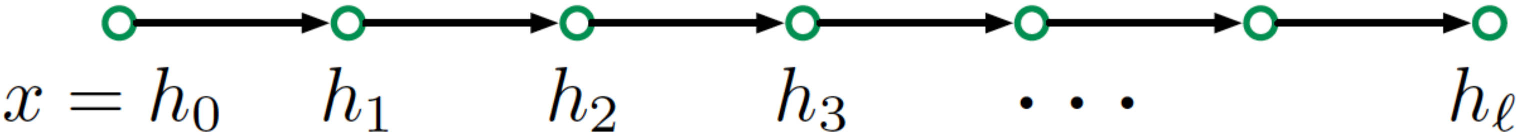


For a specific input x,

- $h_i = \sigma(w_i h_{i-1} + b_i)$
- The loss L can be gleaned from h_ℓ

A Single Chain of Nodes

A neural net with one node per hidden layer:



For a specific input x,

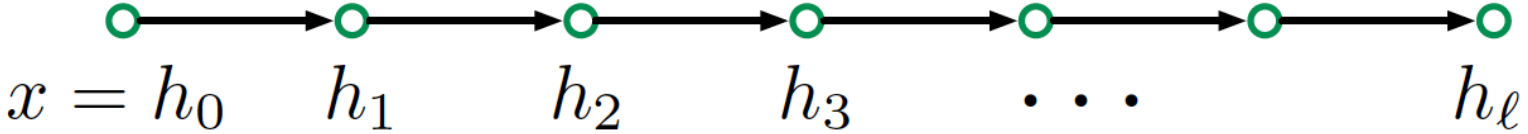
- $h_i = \sigma(w_i h_{i-1} + b_i)$
- The loss L can be gleaned from h_ℓ

To compute dL/dw_i we just need dl/dh_i:

$$\frac{dL}{dw_i} = \frac{dL}{dh_i}\frac{dh_i}{dw_i} = \frac{dL}{dh_i}\sigma'(w_i h_{i-1} + b_i)h_{i-1}$$

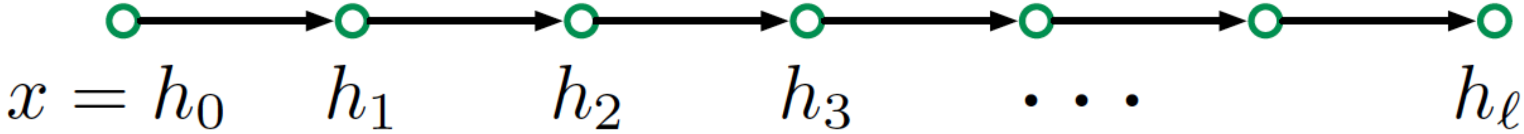
Backpropagation

- On a single forward pass, compute all the h_i.
- On a single backward pass, compute dL/dh_ℓ, …, dL/dh_1



Backpropagation

- On a single forward pass, compute all the h_i.
- On a single backward pass, compute dL/dh_ℓ, …, dL/dh_1



From $h_{i+1} = \sigma(w_{i+1}h_i + b_{i+1})$, we have:

$$\frac{dL}{dh_i} = \frac{dL}{dh_{i+1}}\frac{dh_{i+1}}{dh_i} = \frac{dL}{dh_{i+1}}\sigma'(w_{i+1}h_i + b_{i+1})w_{i+1}$$

It is good to know how backprop works, but in practice you will probably not need to implement it yourself!

In PyTorch and similar systems, you define

- the structure of the net
- the weights
- the loss function

and the system automatically figures out the rules for updating the weights.