Lecture 12: Neural Network Training Algorithm

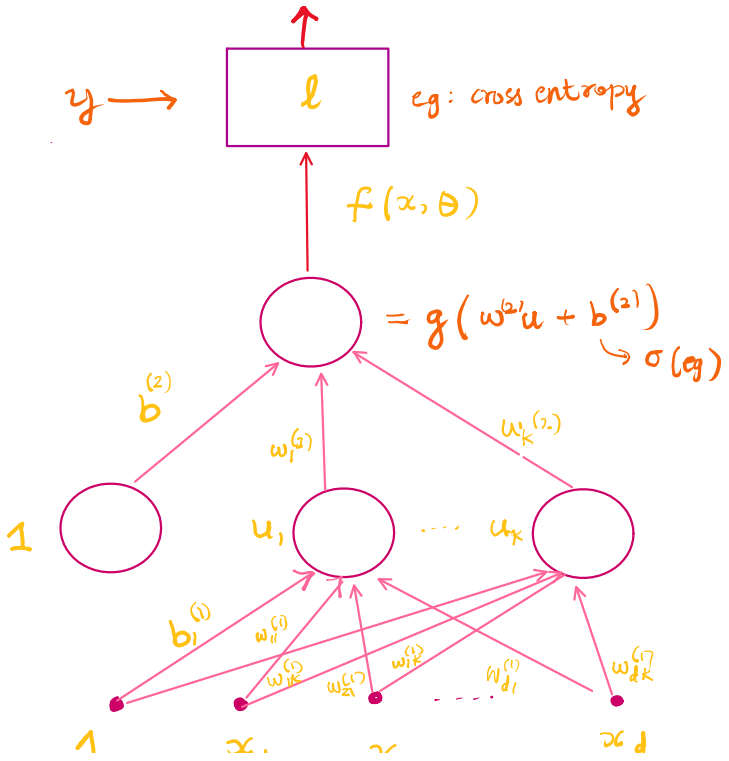
21 February 2024 11:03

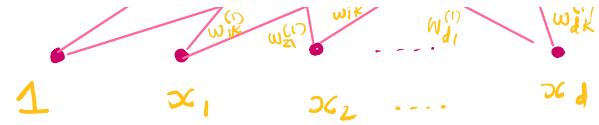
The neural networks we are addressing so far is "fit forward"

NN training algorithm

Input: $NN(x;\theta)$, training data $(x_i, y_i)_{i=1}^n$, loss function, ℓ.

- Randomly initialize θ
- Repeat until stopping criteria is met:
  - Pick randomly a batch of examples $(x_i, y_i)_{i \in B}$, $B \subset N$, $|B| = T < n$ → hyper parameter
  - Compute the gradient of the loss function, $\nabla_\theta \ell$ at θ
  - Update $\theta \leftarrow \theta - \eta \nabla_\theta \ell$
- Return θ





$$g\left(\underset{k\times d}{w^{(1)}}\, x + b^{(1)}\right) = u \longrightarrow u_i = g\left(w_i^{(1)} x + b_i^{(1)}\right)$$

Goal: learn the weights w, b that minimises ℓ.

Gradient descent:

$\frac{\partial \ell}{\partial w_{ij}^{(k)}}$ ← large

w_{ij} ← output node

input node of that layer

$$w_{ij}^{(k)} \leftarrow w_{ij}^{(k)} - \eta \frac{\partial \ell}{\partial w_{ij}^{(k)}}$$

How will we compute these gradient values efficiently?

Back Propagation:

1. Compute $\left.\frac{\partial \ell}{\partial u}\right|_{\theta \leftarrow \text{current } \theta}$ for every node u in the NN, $\theta = (w, b)$

2. Compute $\frac{\partial \ell}{\partial w} = \frac{\partial \ell}{\partial u} \cdot \frac{\partial u}{\partial w}$

$$\frac{\partial \ell}{\partial u} = \sum_{v \in \Gamma(u)} \frac{\partial \ell}{\partial v} \cdot \frac{\partial v}{\partial u}$$

$u = g(w z + \cdots)$ (z in previous layer)

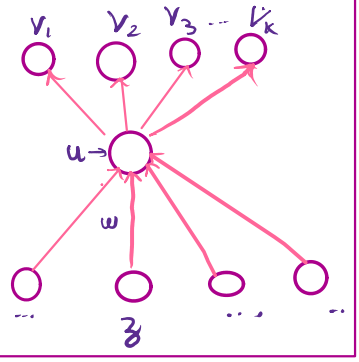
$\frac{\partial u}{\partial w} = g' \cdot z$

Next layer of u,

$\Gamma(u) = \{v_1, v_2, \ldots v_k\}$

$v = g(\tilde{w} u + \cdots)$

for $v \in \Gamma(u)$



Backpropagation algorithm

Two stages:

(a) Forward pass: Calculate the value of each node given an input (x_i, y_i) at θ

$$u_1 = g\left(w_{11}^{(k)} z_1 + w_{21}^{(k)} z_2 + \dots + b_1^{(k)}\right)$$

(b) Backward Pass: Base case: $\frac{\partial \ell}{\partial \ell} = 1$

For each u in a given layer:

For each $v \in \Gamma(u)$:

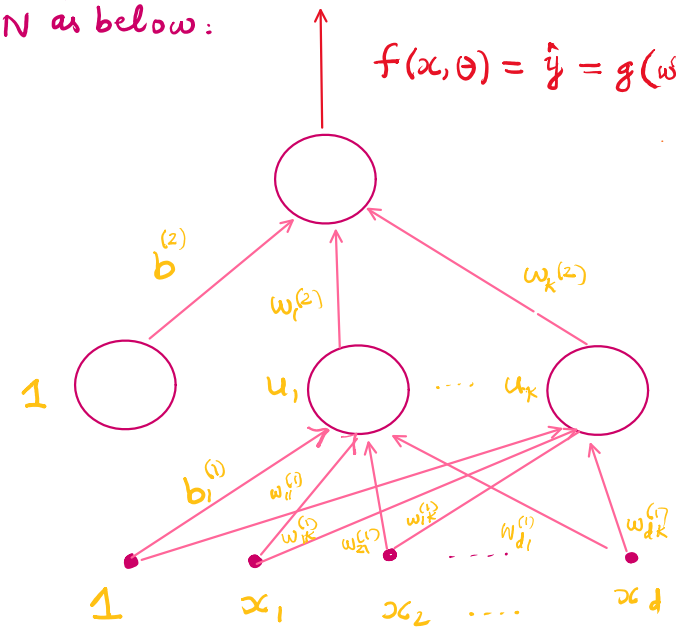
- Use already computed $\frac{\partial \ell}{\partial v}$ (known by induction hypothesis)
- Compute $\frac{\partial v}{\partial u}$
- $\frac{\partial \ell}{\partial v} \cdot \frac{\partial v}{\partial u}$ (sum this)

get $\frac{\partial \ell}{\partial u}$ $\left(\sum_{v \in \Gamma(u)} \frac{\partial \ell}{\partial v} \cdot \frac{\partial v}{\partial u}\right)$

Compute $\frac{\partial \ell}{\partial w} = \frac{\partial \ell}{\partial u} \cdot \frac{\partial u}{\partial w}$, $\frac{\partial \ell}{\partial b}$

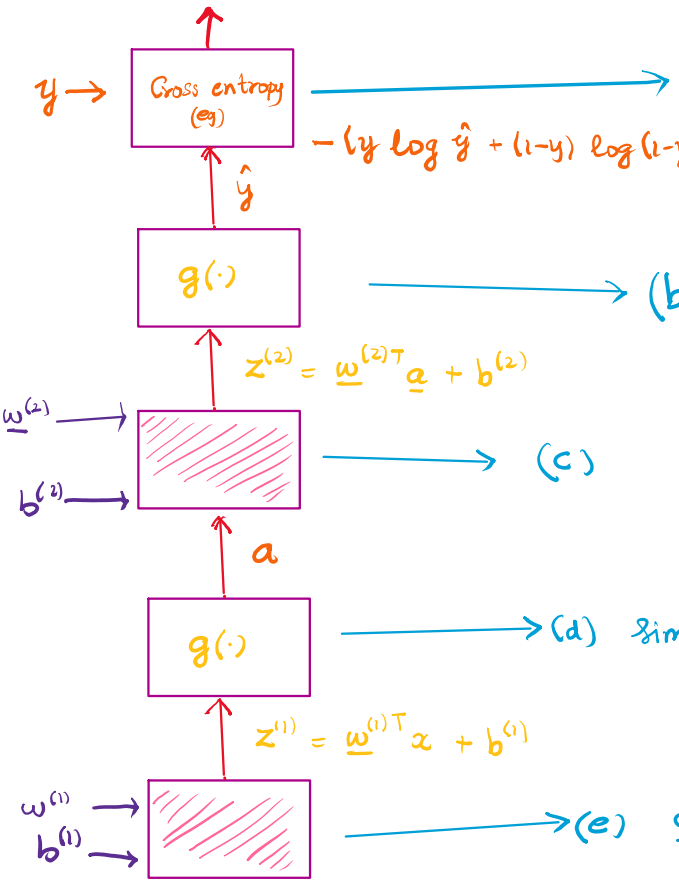
Typically these derivatives are more efficient to compute in vector form. We use a Computational graph for this.

Consider an NN as below:



The computational

The computational graph for above NN is as below:



$\rightarrow$ (a) $\frac{\partial \ell}{\partial \hat{y}} = -\frac{\partial}{\partial \hat{y}}\left(y \log \hat{y} + (1-y)\log(1-\hat{y})\right) = \frac{-y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$

$\delta_{\hat{y}} = \frac{\partial \ell}{\partial \hat{y}}\Big|_{x,y,\theta}$

$\rightarrow$ (b) $\frac{\partial \hat{y}}{\partial z^{(2)}} = g'(z^{(2)})$, $\delta_{z^{(2)}} = \frac{\partial \ell}{\partial z^{(2)}} = \frac{\partial \ell}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{(2)}} = \delta_{\hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{(2)}}$

$\rightarrow$ (c) $\frac{\partial z^{(2)}}{\partial w^{(2)}} = \frac{\partial}{\partial w^{(2)}}(w^{(2)T} a + b^{(2)}) = a$, $\delta_{w^{(2)}} = \frac{\partial \ell}{\partial w^{(2)}}$ (use chain rule)

$\rightarrow$ (d) Similar as above

$\rightarrow$ (e) $\frac{\partial z^{(1)}}{\partial w^{(1)}} = \frac{\partial}{\partial w^{(1)}}(w^{(1)} x + b^{(1)}) = x$

↘ Refer NOTES posted by prof.

Explanable AI: Tries to understant what the weights, a_i's in indermediate layers, mean.

Regularization in NN:

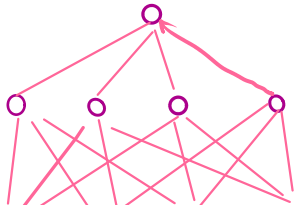
To prevent overfitting in NN.

The commonly used regularization functions:

i) L_2 regularization: $\ell_{Reg} = \ell_{CE} + \lambda \|w\|_2^2$, where $\sum_{i,j,k} (w_{ij}^{(k)})^2$

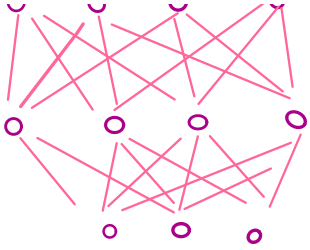
ii) Drop out regularization:

During training keep a neuron active with propability p, delete otherwise



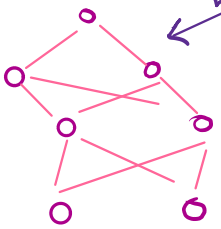
During training keep a neuron active with propability p, delete otherwise

hyper parameter



Drop out for a specific (x_i, y_i)

Expected weight as an output



During test time, use the entire network with expected weights

(iii) Early stopping:

Stopping when performance on validation set stops improving.