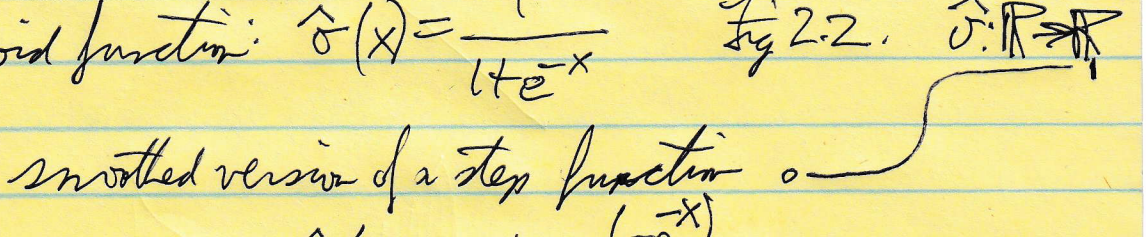
DL1

Lecture 14

following Higham + Higham, SIREV 2020

Deep Learning + the Stochastic Gradient Method

Sigmoid function: $\hat{\sigma}(x) = \frac{1}{1+e^{-x}}$ Fig 2.2. $\hat{\sigma}: \mathbb{R} \to \mathbb{R}$



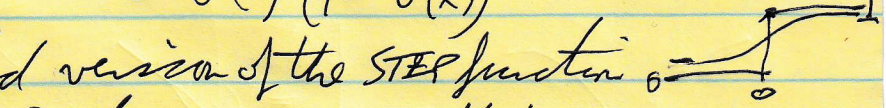
smoothed version of a step function

$$\hat{\sigma}'(x) = \frac{-1}{(1+e^{-x})^2}\left(-e^{-x}\right) =$$

$$= \hat{\sigma}(x)\left(\frac{+e^{-x}}{1+e^{-x}}\right) = \hat{\sigma}(x)\left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)$$

$$= \hat{\sigma}(x)\left(1-\hat{\sigma}(x)\right)$$

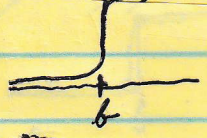
$\hat{\sigma}(x)$ is a smoothed version of the STEP function



which mimics a single neuron in the brain:
firing, or not firing.

Scale + shift (weight + bias in NN language)

$$\hat{\sigma}(w(x+b))$$



Vector version: $\sigma: \mathbb{R}^m \to \mathbb{R}^m$

$$(\sigma(z))_i := \hat{\sigma}(z_i) \qquad z \in \mathbb{R}^m$$

We want layers of neurons:

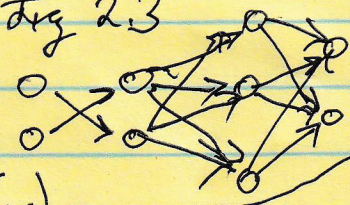
output from one layer = a = input to next layer

output from next layer $= \sigma(Wa+b)$

$W \in \mathbb{R}^{m \times m}$ $b \in \mathbb{R}^m$

square iff # neurons in input layer = # " " output from layer.

Example Fig 2.3



For $x \in \mathbb{R}^2$, $F(x)$ → Output:

$$\sigma\left(W^{[4]}\sigma\left(W^{[3]}\sigma\left(W^{[2]}\sigma\left(W^{[1]}x + b^{[1]}\right) + b^{[2]}\right) + b^{[3]}\right) + b^{[4]}\right)$$

(now $W^{[1]}, b^{[1]}$)

DL2

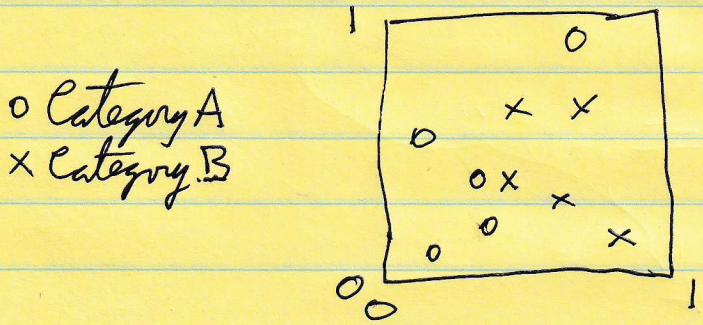
$W^{[2]} \in \mathbb{R}^{2\times 2}$ $b^{[2]} \in \mathbb{R}^2$ # pars = 6

$W^{[3]} \in \mathbb{R}^{3\times 2}$ $b^{[3]} \in \mathbb{R}^3$ # pars = 9

$W^{[4]} \in \mathbb{R}^{2\times 3}$ $b^{[4]} \in \mathbb{R}^2$ # pars = 8

total 23

We'd like to build a "classifier" based on data



o Category A
× Category B

We have 10 points $x^{(i)} \in \mathbb{R}^2$

We'll arbitrarily say we'd like the output $F(x)$ to be close to $\begin{bmatrix}1\\0\end{bmatrix}$ for points x in cat. A and $\begin{bmatrix}0\\1\end{bmatrix}$ for points x in cat. B. (These vectors $\begin{bmatrix}1\\0\end{bmatrix}$ and $\begin{bmatrix}0\\1\end{bmatrix}$ have no relationship to the x-picture.) We write

$$y(x^{(i)}) = \begin{cases} \begin{bmatrix}1\\0\end{bmatrix} & \text{if } x^{(i)} \text{ is cat A} \\ \begin{bmatrix}0\\1\end{bmatrix} & \text{if } x^{(i)} \text{ is cat B.} \end{cases}$$

We want to choose the 23 parameter to

minimize $\text{Cost}(W^{[2]}, W^{[3]}, W^{[4]}, b^{[2]}, b^{[3]}, b^{[4]})$

$$= \frac{1}{10}\sum_{i=1}^{10} \frac{1}{2}\| y(x^{(i)}) - F(x^{(i)}) \|_2^2$$

$F(x^{(i)})$ defined on p. DL1

This is a nonlinear, nonconvex opt problem — called a nonlinear least squares problem.

Possible Methods to Minimize Cost

BFGS
Newton
Gauss-Newton
Levenberg-Marquardt
Trust region method

} implemented in lsqnonlin (Matlab opt toolbx).

DL3

Using lsqnonlin to choose the 23 parameters — TRAINING THE NETWORK — & then plotting the points in the x square where $F_1(x) > F_2(x)$ (meaning: best choice for x is cat. A) gives Fig 2.4
Adding one more cat. B point, & re-optimizing (re-training), get Fig. 2.5.

However, there is no guarantee any method will find the globally optimal choice of parameters — especially if there are a lot of them.

Deep Learning ≡ Many layers in NN.

So far, we are not exploiting the property that the output from one layer is a simple function of the input to it.

Assume there are L layers labelled $l = 1, \dots, L$, with n_l neurons in layer l. (input ↑, output ↑)
Let $a_j^{(l)}$ = output, or activation, from neuron j at layer l. We have

$$a^{[1]} = x = \text{input}$$
$$a^{[l]} = \sigma\left(W^{[l]} a^{[l-1]} + b^{[l]}\right) \in \mathbb{R}^{n_l}, \quad l = 2, 3, \dots, L$$

"feed forward". Now want to minimize

$$\underbrace{\text{Cost}}_{C}(W^{[2]}, \dots, W^{[L]}, b^{[2]}, \dots, b^{[L]}) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} \left\| y(x^{(i)}) - a^{[L]}(x^{(i)}) \right\|_2^2.$$

Write $p^T = \left((\text{vec}(W^{[2]}))^T, \dots (\text{vec}(W^{[L]}))^T, (b^{[2]})^T, \dots (b^{[L]})^T \right)$
vector of parameters $p \in \mathbb{R}^S$

DL4

Stochastic Gradient Method (SG"D")

Two key differences from the gradient method we used before.

(1) Replace $\nabla \text{Cost}(p)$ by ∇(only one (or a few) terms in Cost()) ~~that~~ these are chosen randomly

(2) No line search. Update $p \leftarrow p + \eta \nabla(\;)$ where η is small (stepsize or "learning rate" in ML)

ONLY MAKES SENSE IF WE BELIEVE THE TERMS $C_{x^{(i)}}$ ARE ALL RELATED SOMEHOW.

Let $C_{x^{(i)}} = \frac{1}{2} \| y(x^{(i)}) - a^{[L]}(x^{(i)}) \|_2^2 \equiv C_{x^{(i)}}(p)$

Then $C = \frac{1}{N} \sum_{i=1}^{N} C_{x^{(i)}}$ and

depends on p.

$$\nabla C(p) = \frac{1}{N} \sum_{i=1}^{N} \nabla C_{x^{(i)}}(p)$$

Stochastic Gradient Method

Initialize p. For $k = 1, 2, \ldots$

[Choose i randomly from $\{1, 2, 3, \ldots, N\}$

[Update $p \leftarrow p - \eta \nabla C_{x[i]}(p)$

called "sampling with replacement" because the same index might be sampled multiple times.

Sampling without replacement: pick a permutation of $\{1, 2, \ldots, N\}$ randomly & then run through all indices once. This is called an epoch, & may be repeated multiple times.

More common: instead of one index at a time, choose m integers $k_1, \ldots, k_m$ randomly from $\{1, \ldots, N\}$ and use

$$p \leftarrow p - \eta \frac{1}{m} \sum_{i=1}^{m} \nabla C_{x[k_i]}(p).$$

This is a MINIBATCH.

DL5

In SG for FIXED i → We need $\frac{\partial C}{\partial w_{jk}^{[l]}}$ and $\frac{\partial C}{\partial b_j^{[l]}}$ as these make up $\nabla C(\rho)$.

Gradients via Back Propagation (Automatic Differentiation)

Key point: the training data $x^{(i)}$ and $y(x^{(i)})$ are fixed.

(A) So write $C = \frac{1}{2}\|y - a^{[L]}\|_2^2$ — $a^{[L]}(x^{(i)})$

$C_{x^{(i)}}$, $y(x^{(i)})$; depends on ALL $w_{jk}^{[l]}$, $b_j^{[l]}$ (weights & biases)

Need to refer back to all of these.

(B) Let $z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]} \in \mathbb{R}^{n_l}$ $\quad l = 2, 3, \dots, L$

$z_j^{[l]}$ = weighted input for neuron j at layer l.

(C) Then $a^{[l]} = \sigma(z^{[l]})$

(D) Let $\delta_j^{(l)} = \frac{\partial C}{\partial z_j^{(l)}}$

Notation Let $p, q \in \mathbb{R}^n$, then $p \circ q \in \mathbb{R}^n$ is defined by $(p \circ q)_i = p_i q_i$ componentwise product. NOT composition as in Lec 12.

Recall i is fixed. $j = 1, 2, \dots$ $\left(\frac{\partial C}{\partial z_j^{[L]}}, \frac{\partial C}{\partial z_j^{(l)}}\right)$

Lemma

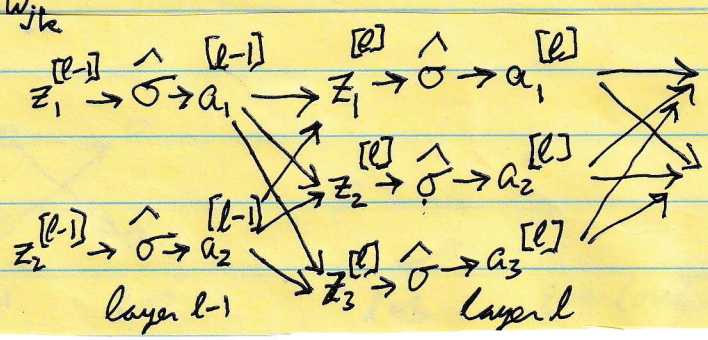
(i) $\delta^{[L]} = \sigma'(z^{[L]}) \circ (a^{[L]} - y)$

(ii) $\delta^{[l]} = \sigma'(z^{[l]}) \circ ((w^{[l+1]})^T \delta^{[l+1]})$ $\quad 2 \le l \le L-1$

(iii) $\frac{\partial C}{\partial b_j^{[l]}} = \delta_j^{[l]}$ $\quad 2 \le l \le L$

(iv) $\frac{\partial C}{\partial w_{jk}^{[l]}} = \delta_j^{[l]} a_k^{[l-1]}$ $\quad 2 \le l \le L$

WHAT WE NEED FOR $\nabla C(\rho)$



$z_l = W^{[l]} a^{[l-1]} + b^{[l]}$

DL6

Recall: i is fixed

Proof

(i) We have $a^{[L]} = \sigma(z^{[L]})$ (C) so $\frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} = \hat{\sigma}'(z_j^{[L]})$

and $\frac{\partial C}{\partial a_j^{[L]}} = \frac{\partial}{\partial a_j^{[L]}} \frac{1}{2} \sum_{k=1}^{n_L} (y_k - a_k^{[L]})^2 = -(y_j - a_j^{[L]})$ (A)

so by the ordinary chain rule

$$\delta_j^{[L]} = \frac{\partial C}{\partial z_j^{[L]}} = \frac{\partial C}{\partial a_j^{[L]}} \frac{\partial a_j^{[L]}}{\partial z_j^{[L]}} = (a_j^{[L]} - y_j)\hat{\sigma}'(z_j^{[L]}) \quad (D)$$

which is equivalent to (i).

(ii) $\delta_j^{[l]} = \frac{\partial C}{\partial z_j^{[l]}} = \sum_{k=1}^{n_{l+1}} \frac{\partial C}{\partial z_k^{[l+1]}} \frac{\partial z_k^{[l+1]}}{\partial z_j^{[l]}} = \sum_{k=1}^{n_{l+1}} \delta_k^{[l+1]} \frac{\partial z_k^{[l+1]}}{\partial z_j^{[l]}}$ (D)

again by chain rule, for $l < L$

Need to consider how all $z_k^{[l+1]}$ depend on the particular $z_j^{[l]}$ that we are differentiating C with respect to.

The z_k at layer $l+1$ are connected to z_j at layer l by

$$z_k^{[l+1]} = \sum_{s=1}^{n_l} w_{ks}^{[l+1]} \sigma(z_s^{[l]}) + b_k^{[l+1]}.$$

so $\frac{\partial z_k^{[l+1]}}{\partial z_j^{[l]}} = w_{kj}^{[l+1]} \hat{\sigma}'(z_j^{[l]})$

so

becomes $\delta_j^{[l]} = \hat{\sigma}'(z_j^{[l]}) \left((W^{[l+1]})^T \delta^{[l+1]} \right)_j \equiv$ (ii)

(iii) We have $z_j^{[l]} \underset{(B)}{=} \left(W^{[l]} a^{[l-1]} + b^{[l]} \right)_j \underset{(E)}{=} \left(W^{[l]} \sigma(z^{[l-1]}) \right)_j + b_j^{[l]}$

since $z^{[l-1]}$ does not depend on $b_j^{[l]}$ this says

$$\frac{\partial z_j^{[l]}}{\partial b_j^{[l]}} = 1$$

DL 7

so $\frac{\partial C}{\partial b_j^{[l]}} = \frac{\partial C}{\partial z_j^{[l]}} \frac{\partial z_j^{[l]}}{\partial b_j^{[l]}} \underset{(b)}{=} \delta_j^{[l]}$ ✓

(iv) From (b), $z_j^{[l]} = \left(\sum_{k=1}^{n_{l-1}} w_{jk}^{[l]} a_k^{[l-1]} \right) + b_j^{[l]}$

so $\frac{\partial z_j^{[l]}}{\partial w_{jk}^{[l]}} = a_k^{[l-1]}$ $\forall j$ (jth neuron at layer l uses weights only in j^{th} row of $W^{[l]}$ & applies them linearly)

and $\frac{\partial z_s^{[l]}}{\partial w_{jk}^{[l]}} = 0$ for $s \neq j$.

so $\frac{\partial C}{\partial w_{jk}^{[l]}} = \sum_{s=1}^{n_l} \frac{\partial C}{\partial z_s^{[l]}} \frac{\partial z_s^{[l]}}{\partial w_{jk}^{[l]}} = \left(\frac{\partial C}{\partial z_j^{[l]}} a_k^{[l-1]} + 0 \right)$ ✓

$\frac{\partial C}{\partial z_j^{[l]}} = \delta_j^{[l]}$ (b)

Key Points of the Lemma.

- $a^{[L]}$ is computed by forward pass through the network, computing $a^{[1]}, a^{[2]}, \ldots, a^{[L]}$ one at a time.
- then (i) gives us $\delta^{[L]}$
- then (ii) gives us $\delta^{[L-1]}, \delta^{[L-2]}, \ldots, \delta^{[2]}$ in a backwards pass through the network. This is called back propagation. It is a special case of the REVERSE MODE of AUTOMATIC DIFFERENTIATION (see N+W ch. 8.2) (see also Griewank+Walther). (idea goes back many decades)

Can replace Hadamard notation by diagonal matrices.

Let $D^{[l]} = \mathrm{Diag}\left(\sigma'(z_1^{[l]}), \ldots, \sigma'(z_{n_l}^{[l]}) \right)$. Then $\delta^{[l]} \underset{(ii)}{=} D^{[l]} \left(W^{[l+1]} \right)^T \delta^{[l+1]}$ $2 \leq l \leq L-1$

DL8

Pseudocode : SG with BP

hence SG

for counter = 1, 2, ... :

choose k randomly from $\{1, 2, 3, \ldots, N\}$ (a.k.a. i)

$a^{[1]} = x^{(k)}$ ← CURRENT TRAINING POINT

forward pass:

for $l = 2$ to L

$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$$

$$a^{[l]} = \sigma(z^{[l]})$$

$$D^{[l]} = \mathrm{Diag}\left(\sigma'(z^{[l]})\right)$$

end

$$\delta^{[L]} = D^{[L]}\left(a^{[L]} - y(x^{(k)})\right)$$

back prop.:

for $l = L-1$ down to 2

$$\delta^{[l]} = D^{[l]}\left(W^{[l+1]}\right)^T \delta^{[l+1]}$$

end

for $l = \{2 \text{ to } L, \ L \text{ down to } 2\}$ ← order doesn't matter

gradient step:

$$W^{[l]} \leftarrow W^{[l]} - \eta\left(\delta^{[l]}\left(a^{[l-1]}\right)^T\right)$$

outer product with j,k entry $\delta_j^{[l]} a_k^{[l-1]}$

$$b^{[l]} \leftarrow b^{[l]} - \eta \delta^{[l]}$$

end

Efficiency is very dependent on step size/learning rate η.

Other Extensions Discussed in HBH

- convolutional neural networks
- avoiding overfitting using "validation data"
- other activation functions

  e.g. RELU $\sigma(x) = \begin{cases} 0 & x \le 0 \\ x & x \ge 0 \end{cases}$
- other cost functions

  e.g. softmax instead of $\| \ \|_2^2$ (or $\| \ \|_\infty$).