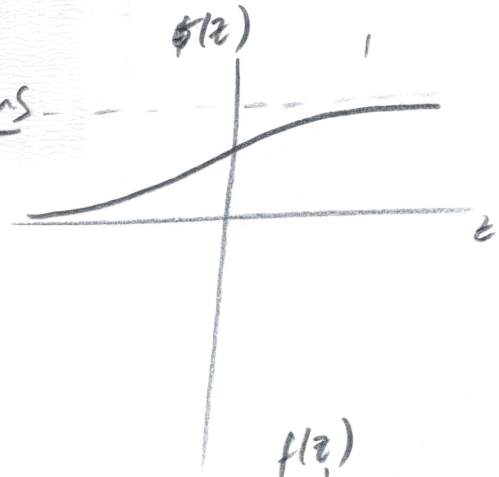
11-785

09/05/2020

L4 - Backpropagation - review

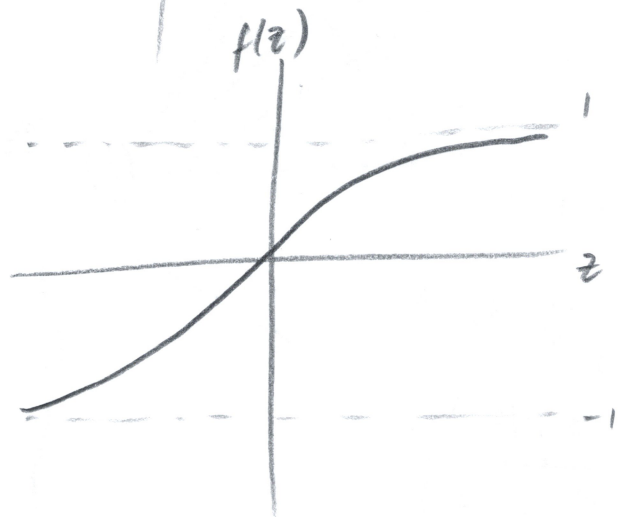
(*) Activation functions

(*) Will go on activation functions



Sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$

$\sigma'(z) = \sigma(z)(1-\sigma(z))$

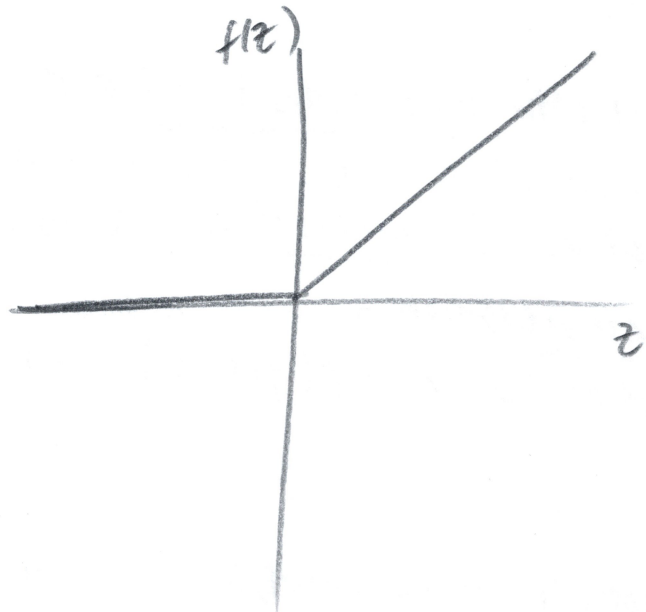


Tanh $f(z) = \tanh(z)$

$f'(z) = 1 - f(z)^2 = 1 - \tanh^2(z)$

$= \max(0, z)$

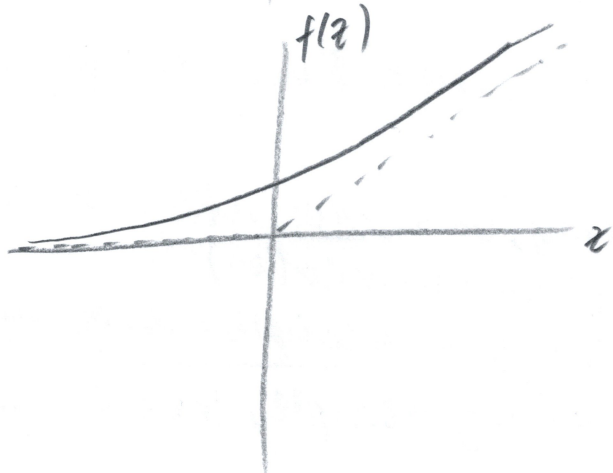
ReLU $f(z) = \begin{cases} 0 & z < 0 \\ z & z \geq 0 \end{cases}$



$f'(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$

Softplus $f(z) = \ln(1+e^z)$

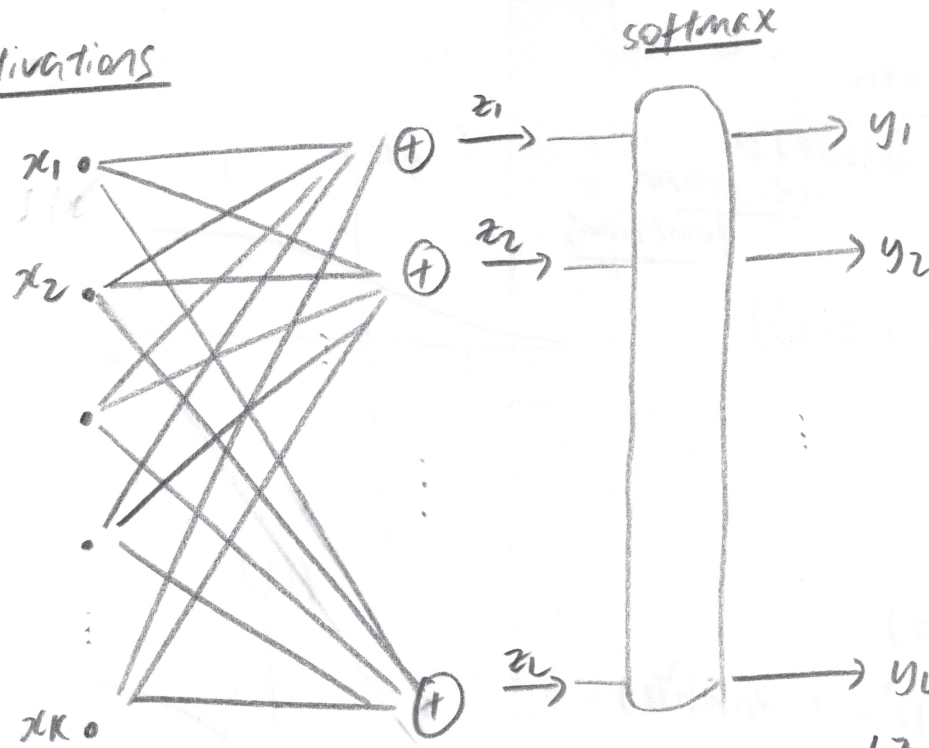
$f'(z) = \frac{1}{1+e^{-z}}$



(smooth approximation to ReLU)

(*) vector activations

- softmax:



(example)

$$z_i = \sum_j w_{ji} x_j + b_i$$

$$y_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

these can be stacked

$$\underline{z} = \begin{pmatrix} z_1 \\ \vdots \\ z_L \end{pmatrix} \qquad \underline{y} = \begin{pmatrix} \frac{\exp(z_1)}{\sum_j \exp(z_j)} \\ \vdots \\ \frac{\exp(z_L)}{\sum_j \exp(z_j)} \end{pmatrix}$$

(*) Natural affinity of output unit and individual error/loss/divergence

- Recall you can choose whether to endow with prob. interp. / est. principle
- And that you can instead choose to frame in terms of minimisation
- prob. interpretation:-
  - conditional distn:- $p(y|\underline{x};\underline{\theta}) \longrightarrow$ ML

(*) BR doesn't use consistent, nor memorable, nor standard notation

↳ look elsewhere if you want formalism

(*) Essentially, BR's 'error' refers to the average training error (empirical risk minimisation)

(*) Also; this is a sample average of error/divergence/loss over each training example; that is there is additive structure

- For the relationship between error minimisation, functional form, probabilistic interpretation (ML estimation) and info-theoretic relations; see Bishop for former; and Goodfellow for latter.

(*) Real-valued output (vectors)

- notationally, corresponds to Bishop (2006) - S.5.2
- Tree is a summation over training instances; and over each component of the target vector in this form:- (multiple target variables)

- BR: "$Div(\underline{Y}, \underline{d}) = \frac{1}{2}\|\underline{Y} - \underline{d}\|_2^2 = \frac{1}{2}\sum_i (y_i - d_i)^2$"

$\underline{Y}$ - vector valued
$\underline{d}$ - vector

- over each component of target vector

AND "$Err(W) = \frac{1}{T}\sum_{i=1}^{T} Div(\underline{f}(\underline{X}_i; \underline{W}), \underline{d}_i)$"

- over all training instances

(*) Also fails to make clear that div and error are also a function of parameters entirely explicitly (or at least does not emphasise)

$$\frac{dDiv(\underline{Y}, \underline{d})}{dy_i} = (y_i - d_i) \quad \checkmark$$

$$\nabla_{\underline{Y}} Div(\underline{Y}, \underline{d}) = \begin{bmatrix} y_1 - d_1 \\ y_2 - d_2 \\ \vdots \end{bmatrix} \quad \checkmark$$

Bishop ref.: $E(\underline{w}) = \frac{1}{2}\sum_{n=1}^{N} \|\underline{y}(\underline{x}_n, \underline{w}) - \underline{t}_n\|_2^2 \quad \frac{dE}{dy_k} = y_k - t_k \quad \checkmark$

(*) Binary class.

- scalar output $Y \in (0,1)$ $d \in (0,1)$

$$div(Y,d) = -d \log Y - (1-d)\log(1-Y)$$ (cross entropy)

(*) Not explicit about probabilistic interp. → see Goodfellow, Bishop

→ see Appendix of HW1

- info-theoretic interp.
- Bernoulli

$$\frac{dDiv(Y,d)}{dY} = \begin{cases} -\frac{1}{Y} & \text{if } d=1 \\ \frac{1}{1-Y} & \text{if } d=0 \end{cases}$$

- sigmoid activation used in combination with binary class. (for outputs) (see Bishop)

(*) Multi-class classification

- output: $\begin{bmatrix} y_1 \\ y_2 \\ \vdots \end{bmatrix}$ target output: $\begin{bmatrix} d_1 \\ d_2 \\ \vdots \end{bmatrix}$
- one-hot encoding
- with c^{th} position $d_c = 1$ if output is class c.

$$div(\underline{Y}, \underline{d}) = -\sum_i d_i \log y_i = -\log y_c$$

→ a summation over components of vector y and $\underline{d}$

$$\frac{dDiv(\underline{Y}, \underline{d})}{dY_i} = \begin{cases} -\frac{1}{y_c} & \text{for } c^{th} \text{ component} \\ 0 & \text{for remaining components} \end{cases}$$

$$\nabla_{\underline{Y}} div(\underline{Y}, \underline{d}) = \begin{bmatrix} 0 \\ \vdots \\ -\frac{1}{y_c} \\ \vdots \\ 0 \end{bmatrix}$$

$$E(\underline{W}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln y_k(\underline{x}_n; \underline{W})$$

Bishop ref: full summed error over training instances and components of vector outputs

- If considering i^{th} training instance:

$$E(\underline{W}) = \sum_{i=1}^{N} E_i(\underline{W})$$

i.e. suppress training instance indexing

$$E_i(\underline{W}) = -\sum_{k=1}^{K} t_k \ln y_k$$

(*) Gradient descent algorithm (scalar formulation)

- Initialise weights $\{w_{ij}^{(k)}\}$
- DO:-
  - $\forall i,j,k$ (i.e. initialise all weights in all layers), initialise $\frac{dErr}{dw_{ij}^{(k)}} = 0$
  - $\forall t = 1, \ldots, T$ (i.e. all training instances)
    - $\forall$ layer k and $\forall i,j$:-
      - compute $\frac{dDiv(Y_t, d_t)}{dw_{ij}^{(k)}}$
      - $\frac{dErr}{dw_{ij}^{(k)}} \mathrel{+}= \frac{dDiv(Y_t, d_t)}{dw_{ij}^{(k)}}$
  - For every layer k, and $\forall i,j$ =

$$w_{ij}^{(k)} = w_{ij}^{(k)} - \frac{\eta}{T} \frac{dErr}{dw_{ij}^{(k)}}$$

- until Err has converged

(*) Require efficient algorithm for computing $\frac{dDiv(Y_t, d_t)}{dw_{ij}^{(k)}}$ for all weights

- This is the essence of backpropagation
- Trick is to find a way of calculating how much individual per training instance loss/error/divergence is responding to changes in your parameters

(*) chain rule of calculus + influence diagram

Backprop → forward pass
↘ backward pass

- Forward pass → fairly clear
- propagate info through network

Forward pass (pseudocode) (scalar)

- Input: D dim vector $\underline{x} = [x_1, x_j, x_D]$ $j = 1, \dots, D$

- Set:-
  - $D_0 = D$ (width of 0^{th} layer (input))
  - $y_j^{(0)} = x_j$ $j = 1, \dots, D$ $y_0^{(k=1,\dots,N)} = x_0 = 1$ (local 0^{th} layer (inputs) and initialise biases)

- For layer $k = 1, \dots, N$
  - For $j = 1, \dots, D_k$ (D_k is size of k^{th} layer)
    - $z_j^{(k)} = \sum_{i=0}^{D_{k-1}} w_{i,j}^{(k)} y_i^{(k-1)}$
    - $y_j^{(k)} = f_k(z_j^{(k)})$

Output: $\underline{Y} = y_j^{(N)}$ $j = 1, \dots, D_N$ (final layer outputs)

(*) not entirely clear what intermediate forward value comput. are stored - needs clarity

→ notes + slides cover already

(*) Backpropagation

(*) compute derivatives layer by layer

- (i) $\frac{\partial Div(\underline{Y}, \underline{d})}{\partial y_i} = \frac{\partial Div(\underline{Y}, \underline{d})}{\partial y_i^{(N)}}$ $\forall i$ (derivative of div wrt output)

(ii) $\frac{\partial Div(\underline{Y}, \underline{d})}{\partial z_i^{(N)}}$ (iii) $\frac{\partial Div(\underline{Y}, \underline{d})}{\partial w_{ij}^{(N)}}$ iv) $\frac{\partial Div(\underline{Y}, \underline{d})}{\partial y_i^{(N-1)}}$ - bias included in weight

do this for each layer $N, (N-1), \dots$

(*)1

(*) Key is to note
derivative of each layer is made up of
i) an element computed in an earlier iteration of algorithm (i.e. a later layer) (div)
ii) an element computed during the forward pass, which has been 'cached' (divv)

BR: does not make clear
what values are cached from forward pass

(*)(1) - Backpropagation
(*) one for one (final layer) - 4 characteristic/archetypal derivatives (see diagrams so you can visualise after)

(*) Do for $(N-1)^{th}$ layer to illustrate chain rule summation

$$\frac{\partial Div}{\partial y_i^{(N-1)}} = \sum_j \frac{\partial z_j^{(N)}}{\partial y_i^{(N-1)}} \cdot \frac{\partial Div}{\partial z_j^{(N)}}$$

$\frac{\partial z_j^{(N)}}{\partial y_i^{(N-1)}} = w_{ij}^{(N)}$ — already comput.

as $z_j^{(N)} = w_{ij}^{(N)} y_i^{(N-1)}$ + other terms (forward pass ?)

$$\Rightarrow \frac{\partial Div}{\partial y_i^{(N-1)}} = \sum_j w_{ij}^{(N)} \frac{\partial Div}{\partial z_j^{(N)}}$$

$$\frac{\partial Div}{\partial z_i^{(N-1)}} = f_{N-1}'(z_i^{(N-1)}) \frac{\partial Div}{\partial y_i^{(N-1)}}$$

$f_{N-1}'(z_i^{(N-1)})$ — already comput in forward pass.

$\frac{\partial Div}{\partial y_i^{(N-1)}}$ — already comp.

$$\frac{\partial Div}{\partial w_{ij}^{(N-1)}} = \frac{\partial z_j^{(N-1)}}{\partial w_{ij}^{(N-1)}} \cdot \frac{\partial Div}{\partial z_j^{(N-1)}}$$

$$= y_i^{(N-2)} \frac{\partial Div}{\partial z_j^{(N-1)}}$$

$y_0^{(N-2)}$ bias = 1

Backward pass pseudocode (scalar form)

- Output layer (N):
- For $i = 1, \ldots, D_N$ (iterate over size (width) of N^{th} layer from 1 - D_N)

$$\frac{\partial Div}{\partial y_i} = \frac{\partial Div(Y, d)}{\partial y_i^{(N)}}$$

$$\frac{\partial Div}{\partial z_i^{(N)}} = \frac{\partial Div}{\partial y_i^{(N)}} \frac{\partial y_i^{(N)}}{\partial z_i^{(N)}}$$

- For layer $k = N-1$ to 0: (iterate over layers starting from $N-1$, in rev.)
- For $i = 1, \ldots, D_k$ (iterate over every unit in layer)

$$\frac{\partial Div}{\partial y_i^{(k)}} = \sum_j w_{ij}^{(k+1)} \frac{\partial Div}{\partial z_j^{(k+1)}}$$

$$\frac{\partial Div}{\partial z_i^{(k)}} = \frac{\partial Div}{\partial y_i^{(k)}} f_k'\left(z_i^{(k)}\right)$$

$$\frac{\partial Div}{\partial w_{ji}^{(k+1)}} = y_j^{(k)} \frac{\partial Div}{\partial z_i^{(k+1)}} \quad \text{for } j = 1, \ldots, D_{k+1}$$

(*) (w) (#) - still can't entirely see the analogy/similarity between backwards and forwards pass

(*) Backpropagation → derivative of error (divergence) is backpropagated recursively through network

"already computed"