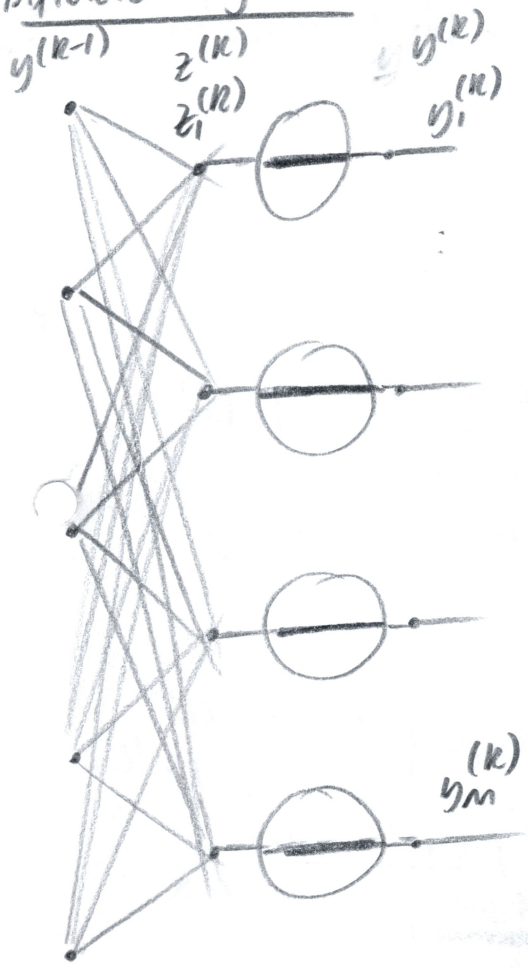
4.5 Backprop calc - review

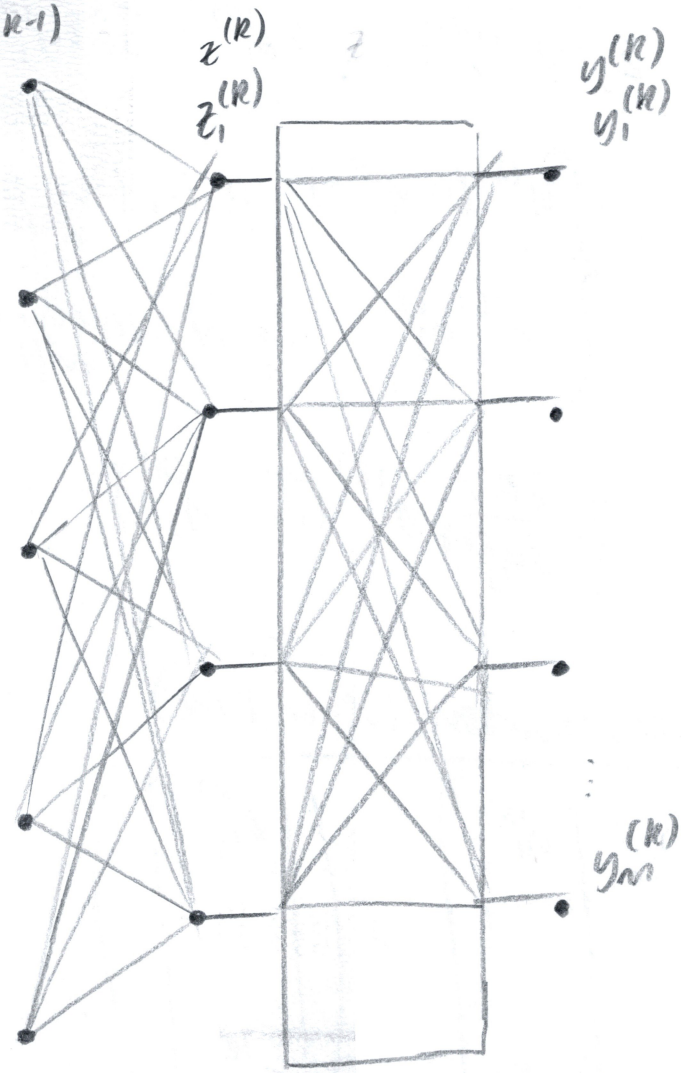
(*) clarify distinction between scalar and vector act.

Influence diagrams (*)



scalar activation

- each z_i influences one y_i
- so in diagram $z_1^{(k)}$ only influences $y_1^{(k)}$



vector activation

- each z_i influences (multiple)/all $y_1, y_2, \cdots, y_m$
- in diagram, each $z_1^{(k)}$ influences (multiple)/all $y_1^{(k)}, y_2^{(k)}, \cdots, y_m^{(k)}$

(*) no. outputs in layer $y^{(k)}$ does not have to equal no. inputs in layer $z^{(k)}$

scalar activation :- $\frac{\partial Div}{\partial z_i^{(k)}} = \frac{\partial Div}{\partial y_i^{(k)}} \frac{dy_i^{(k)}}{dz_i^{(k)}}$

- derivative of error wrt input is product

vector activation :- $\frac{\partial Div}{\partial z_i^{(k)}} = \sum_j \frac{\partial Div}{\partial y_j^{(k)}} \frac{dy_j^{(k)}}{\partial z_i^{(k)}}$

- derivative of error wrt input - sum of partial derivs regardless of no. outputs ($y_j^{(k)}$).

(*) softmax → found HW1 pt 1 very confusing (this will provide clarity)

$$y_i^{(k)} = \frac{\exp(z_i^{(k)})}{\sum_j \exp(z_j^{(k)})} \qquad \frac{\partial Div}{\partial z_i^{(k)}} = \sum_j \frac{\partial Div}{\partial y_j^{(k)}} \frac{\partial y_j^{(k)}}{\partial z_i^{(k)}}$$

→ see L4.5 s~~e~~f notes for derivatn.

$$\frac{\partial y_j^{(k)}}{\partial z_i^{(k)}} = \begin{cases} y_i^{(k)}(1 - y_i^{(k)}) & \text{if } i=j \text{ (non-dicog influence path)} \\ -y_i^{(k)} y_j^{(k)} & \text{if } i \neq j \text{ (dicgonal infl. path)} \end{cases}$$

$$\frac{\partial Div}{\partial z_i^{(k)}} = \sum_j \frac{\partial Div}{\partial y_j^{(k)}} y_i^{(k)} (\delta_{ij} - y_j^{(k)}) \quad \text{[using compression of not via Kronecker deltas)} \quad (I)$$

$\delta_{ij} = 1$ if $i=j$; $\delta_{ij} = 0$ if $i \neq j$

(*) vector activations (vector form)

$$\underline{y}^{(k)} = \begin{bmatrix} y_1^{(k)} \\ \vdots \\ y_M^{(k)} \end{bmatrix} = f\left(\begin{bmatrix} z_1^{(k)} \\ \vdots \\ z_D^{(k)} \end{bmatrix} \right)$$

- M - no. of output units in layer k (width of layer k)
- D - no. of pre-activation affine combos in layer k

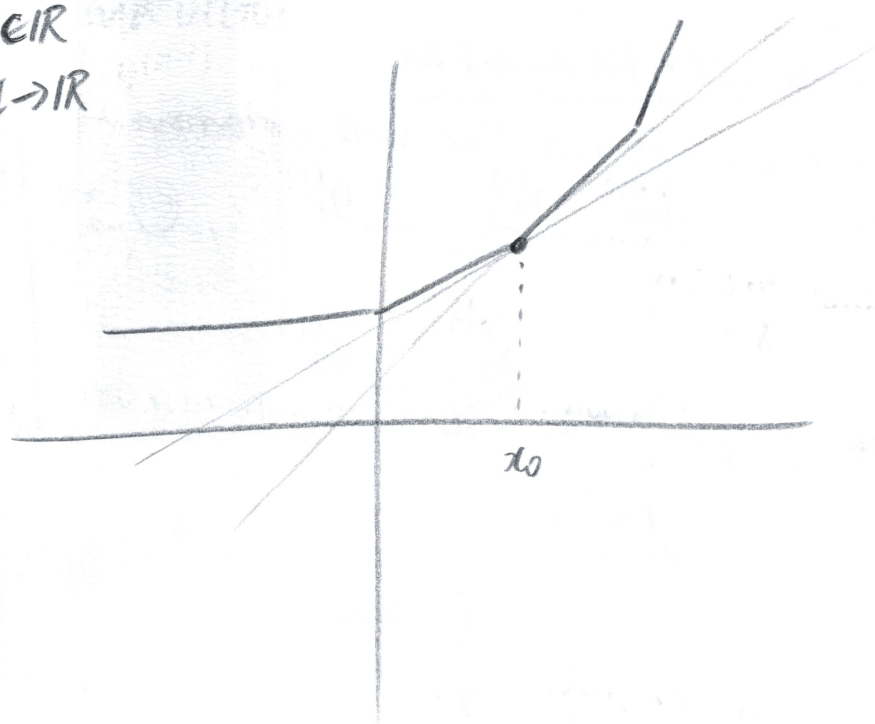
(*) examine how using vector activation for softmax differs from stand backprop scalar form pseudocode (only change in output act. fcn)
i.e. use (I) in output layer part of pseudocode

(*) subgradients

- Multiple def.
- If $f: \underline{U} \to \mathbb{R}$ is a real-valued convex function defined on a 'convex open set' in the Euclidean space $\mathbb{R}^n$, a vector $\underline{v}$ is called a subgradient at a point $\underline{x}_0$ in $\underline{U}$ if for any $\underline{x}$ in $\underline{U}$ one has

$$f(\underline{x}) - f(\underline{x}_0) \geq \underline{v}^T (\underline{x} - \underline{x}_0)$$

- Intuitive exp. (in 2D space) $I \subseteq \mathbb{R}$
- At kinks, function not differentiable $f: I \to \mathbb{R}$
- For any x_0 in the domain of f, one can draw a line which goes through point $(x_0, f(x_0))$



- This point is everywhere either touching or below the graph of f.
- The slope of such a line (there are/may be many) is the subderivative
- A subgradient is a generalisation to functions of multiple variables

---

(*) Zeiler's I can't fully understand:-
i) subgradient is direction in which function is guaranteed to increase
ii) If function differentiable at x_0, subgradient is gradient

---

(*) subgradients used for RELU proof

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

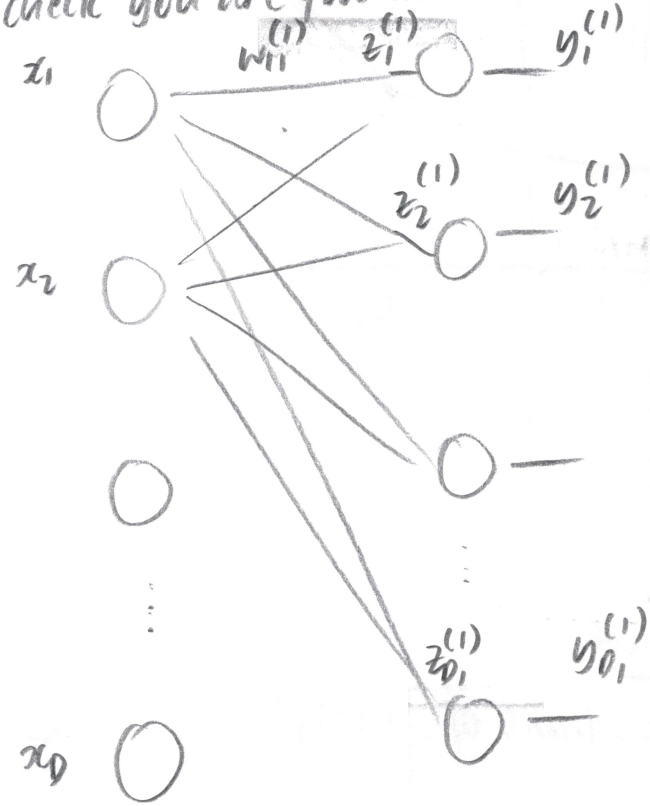
- (*) Subgradients + max → resist with CNNs

- ~~not~~ (*) Note where the use of vector activations for non output layers (c.f. output layers) affects the code for backprop

---

(*)(*) NN vector formulation

- check you are familiar with notation:-



(01) $\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix}$ $\underline{z}_k = \begin{bmatrix} z_1^{(k)} \\ z_2^{(k)} \\ \vdots \\ z_{D_k}^{(k)} \end{bmatrix}$

$\underline{y}_k = \begin{bmatrix} y_1^{(k)} \\ y_2^{(k)} \\ \vdots \\ y_{D_k}^{(k)} \end{bmatrix}$ $\underline{b}_k = \begin{bmatrix} b_1^{(k)} \\ b_2^{(k)} \\ \vdots \\ b_{D_{k+1}}^{(k)} \end{bmatrix}$

$\underline{W}_k = \begin{bmatrix} w_{11}^{(k)} & w_{21}^{(k)} & \cdots & w_{D_{k-1}1}^{(k)} \\ w_{12}^{(k)} & w_{22}^{(k)} & \cdots & w_{D_{k-1}2}^{(k)} \\ \vdots & & \ddots & \\ w_{1D_k}^{(k)} & w_{2D_k}^{(k)} & \cdots & w_{D_{k-1}D_k}^{(k)} \end{bmatrix}$

(*) each column vector in $\underline{W}_k$ contains the weights from all units in $(k-1)^{th}$ layer to 1st unit of k^{th} layer

- $\underline{W}_k$ - weights matrix from $(k-1)$ to k^{th} layer

(*) note f is a vector function of a vector input

(*) vector notation for relevant transformations in k^{th} layer of a NN.

(01)

- set $\underline{y}_0 = \underline{x}$

$\underline{z}_k = \underline{W}_k \underline{y}_{k-1} + \underline{b}_k$

$\underline{y}_k = \underline{f}_k(\underline{z}_k)$

(*) Forward pass pseudocode (vector form)

- set $\underline{y}_0 = \underline{x}$
- For layer $k = 1, \ldots, N$
  - $\underline{z}_k = \underline{W}_k \underline{y}_{k-1} + \underline{b}_k$
  - $\underline{y}_k = \underline{f}_k(\underline{z}_k)$
- $\underline{\hat{y}} = \underline{y}_N$

- note output is a composition of affine, activations layer by layer.
- see 2 representations in L4.5 notes

(*) Jacobians matrices of vector-valued functions are matrices containing all first-order partial derivatives (*)

- vector-valued fn: $f: \mathbb{R}^D \to \mathbb{R}^M$ (in context of L4.5 notes not.)

$$\underline{z} = \begin{pmatrix} z_1 \\ \vdots \\ z_D \end{pmatrix} \quad \underline{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix} = \underline{f}\left(\begin{bmatrix} z_1 \\ \vdots \\ z_D \end{bmatrix}\right)$$

- BR defines counterintuitively (see L4.5 notes you made)

$\underline{J}_{\underline{y}}(\underline{z})$ - Jacobian of $\underline{z}$ wrt $\underline{x}$ (!) (does he change this late?)

(*) Note Jacobians can describe

i) scalar activations (diagonal Jacobian)

ii) vector activations (fully populated Jacobian)

(*) vector calc. notation using gradients, Jacobians

(*) - affine fn: $\underline{z} = \underline{W}\underline{y} + \underline{b}$

$\underline{J}_{\underline{z}}(\underline{y}) = \underline{W}$ ✓

(*) chain rule our vector functions of vector inputs

$\underline{y} = \underline{f}(\underline{g}(\underline{x}))$ $\underline{z} = \underline{g}(\underline{x})$ $\underline{y} = \underline{f}(\underline{z})$

$\underline{J}_{\underline{y}}(\underline{x}) = \underline{J}_{\underline{y}}(\underline{z})\underline{J}_{\underline{z}}(\underline{x})$ ✓

(*) chain rule combining Jacobians and gradients

- Scalar fns of vector inputs

$D = f(\underline{g}(\underline{x}))$ $\underline{z} = \underline{g}(\underline{x})$ $D = f(\underline{z})$

$\nabla_{\underline{x}} D = \nabla_{\underline{z}} D \, \underline{J}_{\underline{z}}(\underline{x})$ ✓

(*) scalar fns of affine fns:- $D = f(\underline{W}\underline{y} + \underline{b})$ $\underline{z} = \underline{W}\underline{y} + \underline{b}$ $D = f(\underline{z})$

$$\nabla_{\underline{y}} D = \nabla_{\underline{z}}(D) J_{\underline{z}}(\underline{y}) = \nabla_{\underline{z}} D \, \underline{W}$$

(*) $\nabla_{\underline{b}} D = \nabla_{\underline{z}} D$, $\nabla_{\underline{W}} D = \underline{y} \nabla_{\underline{z}}(D)$

(*) Gradient descent algorithm pseudocode (vector form)

- Initialise weights matrices $\underline{W}_1, \underline{W}_2, \ldots, \underline{W}_R$
- Do:
  - For every layer k, compute:- → (compute gradient of error as average of gradients or individual divergences over all instances)
    - $\nabla_{\underline{W}_k} Err = \frac{1}{T} \sum_t \nabla_{\underline{W}_k} Div(\underline{Y}_t, \underline{d}_t)$
    - $\underline{W}_k = \underline{W}_k - \eta \nabla_{\underline{W}_k} Err^T$ - update param.
- until Err has converged

(*) Dimensionality clarity → next suppl. notes

(*) NN training algo pseudocode (vector form)

- Initialise all weights, biases $(\underline{W}_1, \underline{b}_1, \underline{W}_2, \underline{b}_2, \ldots, \underline{W}_R, \underline{b}_R)$
- Do:-
  - $Err = 0$
  - $\forall k$ layers, initialise $\nabla_{\underline{W}_k} Err = 0$, $\nabla_{\underline{b}_k} Err = 0$
  - $\forall t = 1, \ldots, T$ instances, comp. (FORWARD)
    - divergence $Div(\underline{Y}_t, \underline{d}_t)$
    - $Err \mathrel{+}= Div(\underline{Y}_t, \underline{d}_t)$
  - $\forall k$ layers, comp. (BACKWARD)

(i) - $\nabla_{\underline{y}_k} Div = \nabla_{\underline{z}_{k+1}} Div \, \underline{W}_k$

(ii) - $\nabla_{\underline{z}_k} Div = \nabla_{\underline{y}_k} Div \, J_{\underline{y}_k}(\underline{z}_k)$

iii) $\nabla_{W_R} Div(Y_t, d_t)$; $\nabla_{b_R} Div(Y_t, d_t)$

iv) $\nabla_{W_R} Err \mathrel{+}= \nabla_{W_R} Div(Y_t, d_t)$; $\nabla_{b_R} Err \mathrel{+}= \nabla_{b_R} Div(Y_t, d_t)$

- $\forall k$, update:-

$W_k = W_k - \frac{\eta}{T}(\nabla_{W_k} Err)^T$; $b_k = b_k - \frac{\eta}{T}(\nabla_{W_k} Err)^T$

- until Err has converged

(*) do you understand comput. flow

(*) An individual look at each step of backprop. (view in combo with slides)

- $\nabla_Y Div$ - dimensionality varies on divergence/problem setting ? : ∇ as a generic way of specifying deriv, grad, Jacob

(i)

- $\nabla_{z_N} Div = \nabla_Y Div \cdot \nabla_{z_N} Y = \nabla_Y Div \, J_Y(z_N)$

- $\nabla_{y_{N-1}} Div = \nabla_{z_N} Div \cdot \nabla_{y_{N-1}} z_N = \nabla_{z_N} Div \, W_N$ → (m) intermediate step; store what we're interested in

(non output layers)

- $\nabla_{z_{N-1}} Div = \nabla_{y_{N-1}} Div \cdot \nabla_{z_{N-1}} y_{N-1}$

$= \nabla_{y_{N-1}} Div \cdot J_{y_{N-1}}(z_{N-1})$ (ii)

- $\nabla_{y_{N-2}} = \nabla_{z_{N-1}} Div \cdot \nabla_{y_{N-2}} z_{N-1}$

$= \nabla_{z_{N-1}} Div \, W_{N-1}$

(up to inputs)

$\nabla_{z_1} Div = \nabla_{y_1} Div \, J_{y_1}(z_1)$

(*) $\nabla_{W_N} Div = y_{N-1} \nabla_{z_N} Div$

$\nabla_{b_N} Div = \nabla_{z_N} Div$

(m)

(*) store $\nabla_{W_{N-1}} Div = y_{N-2} \nabla_{z_{N-1}} Div$

$\nabla_{b_{N-1}} Div = \nabla_{z_{N-1}} Div$

(*) store $\nabla_{W_1} Div = x \nabla_{z_1} Div$ iii)

$\nabla_{b_1} Div = \nabla_{z_1} Div$

---

i) Note $\nabla_{z_N} Y = J_Y(z_N)$

(gradient of vector fn wrt vector is same as Jacobian)

ii) Jacobian diagonal for scalar act.

iii) $y_0 = x$