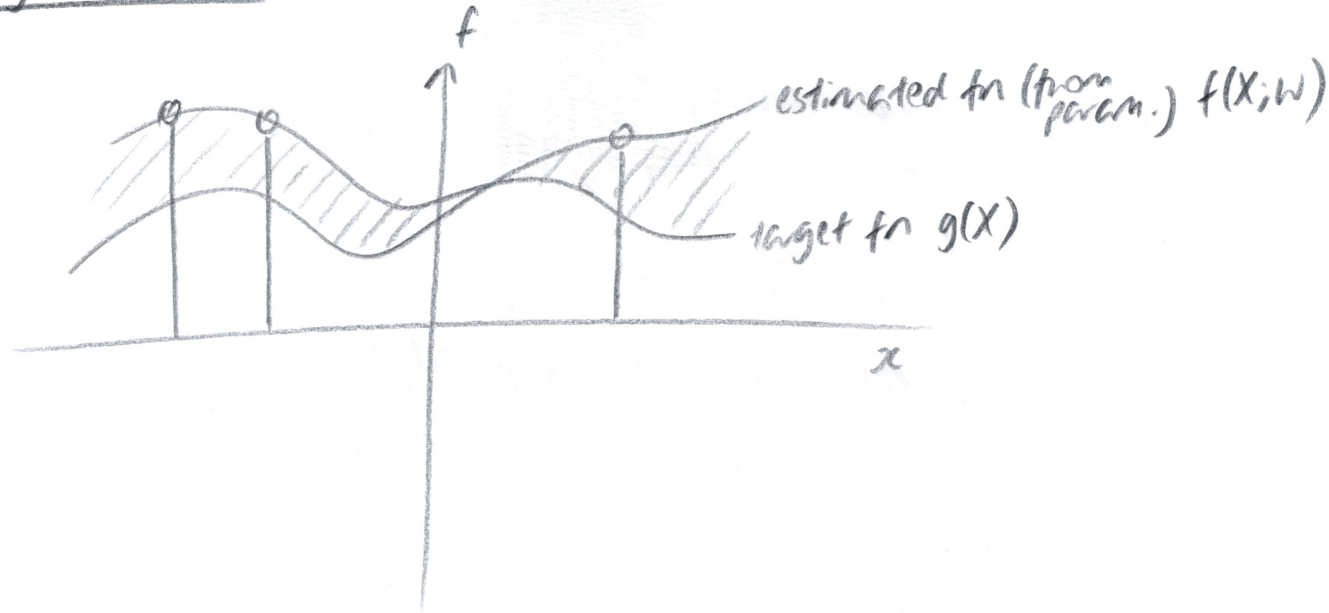
11-785 04/05/2020

L4 - Backpropagation

BR: continue training from L3 (will later deconvolute)

L3 - Training (continued)

- BR:



BR:
- we want to estimate parametes W to minimise the area between f(X;W) and g(X) (or some function of it)
- we require entire function g(X) to compute the integral
$\int_X div(f(X;W), g(X)) P(X) dX$
- g(X) unknown, therefore obtain samples (x_i, d_i) (i.e. values of $g(x_i)$)
- Rather than minimise area, minimise average of 'errors' as a proxy
- Quantified by divergence function (~~either~~ (≥ 0)
- Require divergence to be differentiable
- minimise empirical average error over training data

- optimisation
- High school calculus refresher
- distinguish minimum, maximum, turning points
- $f''(x)$ -ve at maxima
$f''(x)$ +ve at minima

- none of this accounts for concavity/convexity
- critical point → local minima, maxima, inflection points.
- $f''(x) = 0$ at inflection points → require further investigation

---

multivariate optimisation

- find a location where derivative is 0
- small perturbations ~~will not~~ in any direction will not change the value of function.
- gradient: vector of partial derivatives
- function increases fastest in direction of gradient
- —— " decreases —"— in opposite direction of gradient

(*) gradient 2nd property:

- slice the function $f(x)$ at any height
- edge is a level set/contour
- gradient is orthogonal to the level set (remember Bishop! :))
- (W)(A1) → clarity on this ↘ especially when using lagrange multipliers

- recall orthogonal - perpendicular

(*) Hessian - matrix of 2nd derivatives of multivariate input

- generalisations of 2nd deriv for scalar fn
- (*) Hessian eigenvectors give directions of greatest increase/decrease of function
  eigenvalues give direction of curvature (W)(A2) → clarify

- eigenvalue - 0 - inflection
  +ve - minimum
  -ve - maximum
  } analog to 2nd deriv

(*) unconstrained max of multivariate scalar fn

1. solve $\nabla_x f(\underline{x}) = 0$
2. compute $\nabla^2_x f(x)$ at candidate soln.; evaluate

- If Hessian ∇^2 is SPD (positive def), all eigenvalues are +ve
  ⇒ local minima.

(W)(13) → clarify

- If Hessian ∇^2 is SND (negative def), all eigenvalues are negative and
  ⇒ local maxima

(*) Multiple dimensions → maximum in one-direction, minimum in another → horseshoe → 0 derivative → some ∇^2 eigenvalues +ve, some -ve

- often non-convexity forbids such solutions analytically
- use iterative solutions

(*) or is it that local minima are not necessarily global minima (W)(14)

- Iterative solutions:- (optimisation of $f(\underline{x})$)
  i) initial guess $\underline{x}_0$
  ii) update guess
  iii) stop when $f(\underline{x})$ no longer decreases → $\underline{x}^*$

  - direction to step in
  - How big steps must be

- gradient descent

- for maximum:- $\underline{x}^{k+1} = \underline{x}^k + \eta^k \nabla f(\underline{x}^k)^T$

  minimum:- $\underline{x}^{k+1} = \underline{x}^k - \eta^k \nabla f(\underline{x}^k)^T$

(W)(15) - clarify logic, check you completely understand

- (*) You must make sure you understood the logic of gradient descent unimpeachably (W)(15)(16)(17) - (611)
- (*) Relation between step size, derivative (magnitude)

INCOMPLETE IF YOU DO NOT REFRESH

- convergence of gradient descent:-
- Results for convex / non-convex functions → L4 - Backprop

- Back to DL context: Given $(\underline{X}_1, \underline{d}_1), \ldots, (\underline{X}_T, d_T)$

$$\underset{W}{\text{argmin}}\ \text{Loss}(W) = \frac{1}{T}\sum_i^T \text{div}(f(X_i; W), d_i)$$

BR kept this deliberately vague notationally:-
Allows explicit consideration of following issues

Q1) what are input-output pairs $(\underline{x}_i, \underline{d}_i)$

Q2) what is $f()$ and parametes W

Q3) what is divergence $div()$

- $f()$ - MLP directed / NN
  - directed → no feedback loops (1-way info flow)
  - layered → NN (subseq.)
    - Each layer only gets input from previous; outputs to late layers
  - some disagreement about terminology (remember Bishop)

- generally; an individual neron:-
  - A diff. activation applied to an affine input combo

    $y = f\left(\sum_i w_i x_i + b\right)$

    (*) Assume this otherwise specified in course i.e. always affine
  - Any differentiable fn $f(x_1, \ldots, x_N; W)$
- ↳ examine effects of parmete perturbation on output (for gradient descent)
  - G.D. - If I modify param; how much does div/error change (itself fn of output)

(*) However, can use different activations

W(A7): Examine / familiarise with activation functions and derivatives (*)

- neru peculiarity - not differentiable at 0

BR: vector activations

(*) specific activations

(W)(A8): clarity on what he means here

- softmax activation (as vector activation

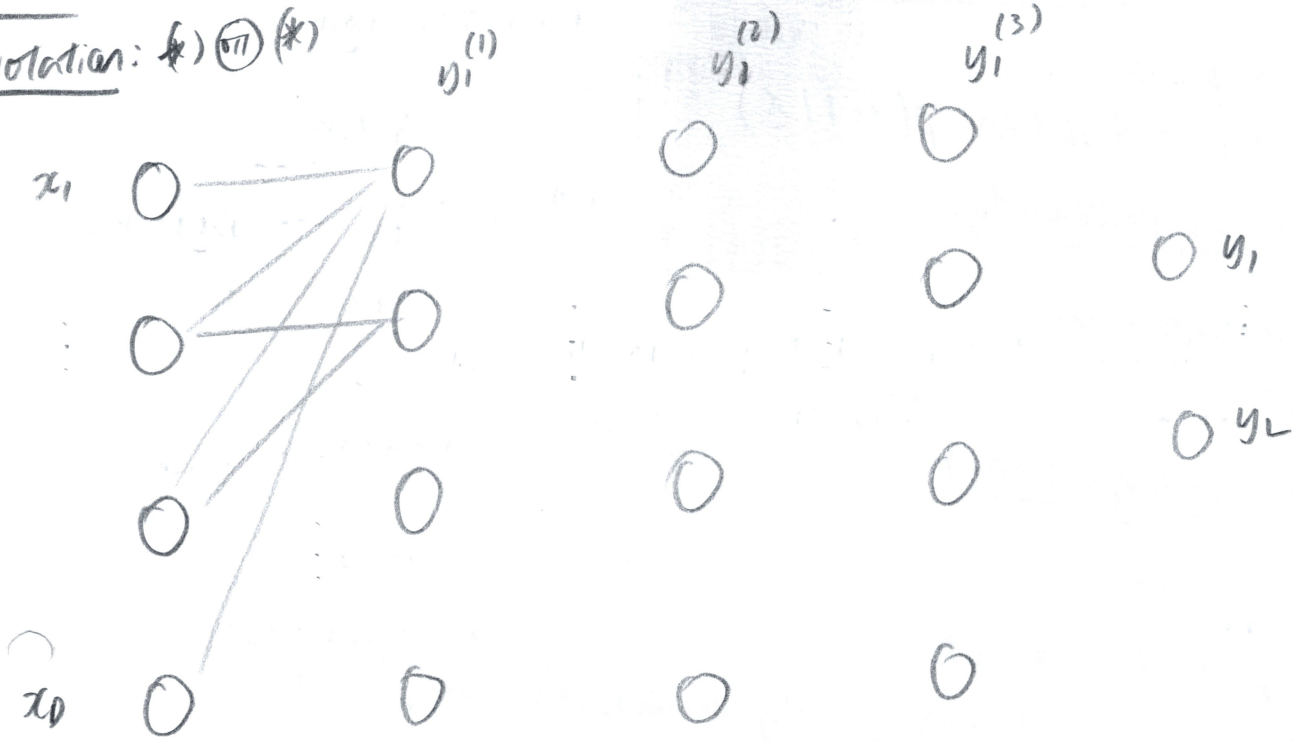
$$z_i = \sum_j w_{ji} x_j + b_i$$

(*) Natue of activ - chnge one param ⇒ all layr outputs will chnge

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

- vector activation is generalisation of scalar activations
(layer) (individual neurons)

- Notation: (*) (**) (***)



- output of i^{th} perceptron of k^{th} layer - $y_i^{(k)}$
- input to network
  $y_i^{(0)} = x_i$
  $y_i^{(N)} = y_i$
- output of network
- (*) weight of connection bet i^{th} unit of $(k-1)^{th}$ layer and j^{th} unit of k^{th} layer
  $w_{ij}^{(k)}$
- Bias to j^{th} unit of k^{th} layer $b_j^{(k)}$

- input-output pairs $(\underline{x}_i, \underline{d}_i)$ $\underline{x}_n$ - n^{th} training input vector
- for i^{th} training pair : $\underline{d}_n$ - n^{th} target output vector $\underline{y}_n$ - n^{th} NN output

$$\underline{x}_i = \begin{bmatrix} x_{n1} \\ \vdots \\ x_{nD} \end{bmatrix} \quad \underline{d}_n = \begin{bmatrix} d_{n1} \\ \vdots \\ d_{nL} \end{bmatrix} \quad \underline{y}_n = \begin{bmatrix} y_{n1} \\ \vdots \\ y_{nL} \end{bmatrix}$$

- Input layer - hidden layer - output layer
- standard ways of representing inputs

- Binary classification
- sigmoid output activation $p(Y=1|\underline{X})$
- i.e. posterior class probability

- multiclass classification

e.g. 5 classes
- use one hot encoding (vector)

---

multiclass network
- target output $\underline{d}$ - one hot
- NN output $\underline{y}$ - one hot → ideally binary, but in practice probability

- softmax vector activ.

$$z_i = \sum_j w_{ji}^{(n)} y_j^{(n-1)}$$

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

(W) (B): check that these are vectors (dimensionality) (*)

→ not vectors but can be stacked (see diagram)

- exponentiation, then normalisation
- like probability vector

Problem statement
- Binary classification
- well balanced training set
- multiclass classification
- one hot representation (10d vector)
- Given $(\underline{X}_1, \underline{d}_1) \dots (\underline{X}_n, \underline{d}_n)$; estimate $\underline{W}$, $\underline{b}$ (weights, biases)

- divergence function div( ) · $\mathbb{R}$ valued $\underline{d}$

- for loss to be differentiable, divergence must be diff. wrt $\underline{W}$

- L_2 divergence $Div(\underline{Y}, \underline{d}) = \frac{1}{2}\|\underline{Y} - \underline{d}\|^2 = \frac{1}{2}\sum_i (y_i - d_i)^2$ $\quad \nabla_{\underline{Y}} \frac{1}{2}\|\underline{Y} - \underline{d}\|^2 = \begin{bmatrix} y_1 - d_1 \\ y_2 - d_2 \\ \vdots \end{bmatrix}$

- squared Euclidean distance

Divergence function

- Binary class

- Based on KL divergence; use cross entropy

(W)(A9) - review + fill in graphs

$Div(Y, d) =$

- check against Bishop

$\nabla_Y Div(Y, d) =$ { }

- All filled in and clarified in supplement notes

- convergence (comparison of X-entropy, L_2 divergence)

✓ (W): use gaps to fill in notes / or supplement

---

divergence function

- multiclass class.

(*) for N classes

$d_1, \ldots, d_N \rightarrow$ 1 hot

$y_1, \ldots, y_N \rightarrow$ probability sum to 1

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

$\nabla_{\underline{Y}} Div(\underline{Y}, d) =$

- sometimes instead of one-hot; $\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ use $\begin{bmatrix} \epsilon \\ \epsilon \\ 1-(K-1)\epsilon \\ \epsilon \\ \epsilon \end{bmatrix}$

W(A10) - label smoothing - supplement

- training neural nets through gradient descent

- Total training error $= E = \frac{1}{T}\sum_{t=1}^{T} Div(\underline{Y}_t, \underline{d}_t)$

Pseudocode

- init. $\{w_{ij}^{(k)}\}$

- Do:

  For every layer k, for all i,j update:-

$$w_{ij}^{(k)} = w_{ij}^{(k)} - \eta \frac{\partial E}{\partial w_{ij}^{(k)}}$$

- until E has converged

$$Loss = \frac{1}{T}\sum_{t=1}^{T} Div(\underline{Y}_t, \underline{d}_t)$$

$$\frac{\partial Loss}{\partial w_{ij}^{(k)}} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial Div(\underline{Y}_t, \underline{d}_t)}{\partial w_{ij}^{(k)}}$$

---

scalar diff fn:-

$y = f(x)$ $\quad \frac{dy}{dx}$ $\quad \Delta y \approx \frac{dy}{dx}\Delta x$ small Δx

multiv.

$y = f(x_1, x_2, \ldots, x_M)$

$\Delta y \approx \frac{\partial y}{\partial x_1}\Delta x_1 + \frac{\partial y}{\partial x_2}\Delta x_2 + \ldots$ sufficiently small $\Delta x_1, \Delta x_2, \ldots, \Delta_M$

$$+ \frac{\partial y}{\partial x_M}\Delta x_M$$

---

engineering def. of chain rule (use above)

$y = f(g(x))$ $\quad z = g(x) \Rightarrow \Delta z = \frac{dg(x)}{dx}\Delta x$

$\frac{dy}{dx} = \frac{\partial f}{\partial g}\frac{dg}{dx}$ $\quad y = f(z) \Rightarrow \Delta y = \frac{df(z)}{dz}\Delta z = \frac{df}{dz}\frac{dg(x)}{dx}\Delta x$

- continue with his presentation; can harmonise rather than recontent.

(w): go our Bishop ✓

(x)

- uncomfortable with notation - see supp. notes

- Adopt

- (w): notation ✓

- calculus refresher

$y = f(g_1(x), g_2(x), \ldots, g_m(x))$

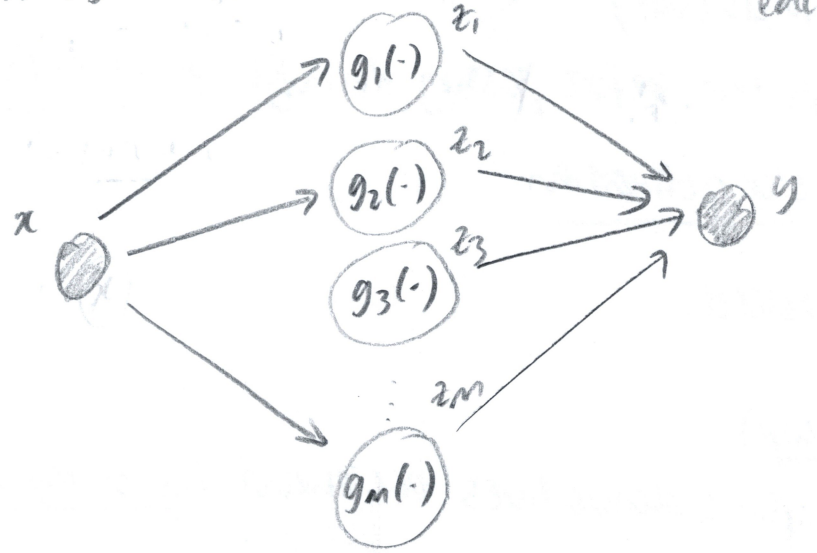
$$\frac{dy}{dx} = \frac{\partial f}{\partial g_1(x)} \frac{dg_1(x)}{dx} + \frac{\partial f}{\partial g_2(x)} \frac{dg_2(x)}{dx} + \ldots + \frac{\partial f}{\partial g_m(x)} \frac{dg_m(x)}{dx}$$

- BR: similar illustration

- influence diagram

$y = f(g_1(x), \ldots, g_m(x))$

(*) x affects y through each of $g_1, \ldots, g_m$



(*) $\frac{dy}{dx} = \frac{dy}{dz_1}\frac{dz_1}{dx} + \ldots + \frac{dy}{dz_m}\frac{dz_m}{dx}$

- source → destination
  - all paths
- influence diagrams give topological way of envisaging chain rule

(*)

$\frac{d\text{Div}(\underline{y}, \underline{d})}{dw_{ij}^{(k)}}$ - compute

(*) BR: each neuron is 2 computations:- affine → activation ✓

- computing derivative for a single input

(*) some notational inconsistencies across slides

- z - affine combination
  y - output after activation applied to affine combination

(w): tracing effect of $\frac{d\text{Div}(\underline{y}, \underline{d})}{dw_{ij}^{(k)}}$: imagine a route, from wrt you tweak parameter to divergence through the topology

(influence)

(*) Derivative comput.
requires intermediate and final output values of network in response to input.

Scalar formulation → (✓)

- Forward propagation - very clear

y_N $y^{(N)} = f_N(\underline{z}^{(N)})$ - vector activation e.g. softmax

(*) Issue with
algo on slides (purposely included)
- All $z_j^{(k)}$ need to be computed before passing though final activation
- As $\underline{y}^{(N)} = f_N(\underline{z}^{(N)})$ is a vector activation

---

Forward pass pseudocode checked.

Computing derivatives (backprop)
start from final layer, compute derivatives backward layer by layer

- Backpropagation → see slides

Backpropagation

-High level
- Illustration for a single training instance (actual - multiple training instances)

(*) Compute $\frac{\partial Div(Y,d)}{\partial y_i} = \frac{\partial Div(Y,d)}{\partial y_i^{(N)}} \quad \forall i$

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

$\frac{\partial Div}{\partial y_i^{(N)}}$ already computed

$\frac{\partial y_i^{(N)}}{\partial z_i^{(N)}} \rightarrow f_N'(z_i^{(N)})$ - derivative of activation function computed in forward pass.

$\Rightarrow \frac{\partial Div}{\partial z_i^{(N)}} = f_N'(z_i^{(N)}) \frac{\partial Div}{\partial y_i^{(N)}}$ (*)

$$\frac{\partial Div}{\partial w_{11}^{(N)}} = \frac{\partial z_1^{(N)}}{\partial w_{11}^{(N)}} \frac{\partial Div}{\partial z_1^{(N)}} = y_1^{(N-1)} \frac{\partial Div}{\partial z_1^{(N)}} \quad (*)$$

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

$\frac{\partial Div}{\partial z_1^{(N)}}$ - already computed on previous backprop step

$= y_1^{(N-1)}$ as $z_1^{(N)} = w_{11}^{(N)} y_1^{(N-1)}$ + (other terms) (computed on forward pass)

$$\frac{\partial Div}{\partial y_1^{(N-1)}} = \sum_j \frac{\partial z_j^{(N)}}{\partial y_1^{(N-1)}} \frac{\partial Div}{\partial z_j^{(N)}} \quad (*)$$

(*) Remember topological ordering; now it relates here to the chain rule

$\frac{\partial Div}{\partial z_j^{(N)}}$ - already computed

$= w_{1j}^{(N)}$ because $z_j^{(N)} = w_{ij}^{(N)} y_1^{(N-1)} + \ldots$ other terms not containing $y_1^{(N-1)}$

$$= \sum_j w_{1j}^{(N)} \frac{\partial Div}{\partial z_j^{(N)}}$$

- W A10: Backpropagation clarity
  - The three (*) types of derivatives summarise the process
  - All are instances of the same principle (A8) (A9)
    (i.e. chain rule, efficient use of computation)

- Backward pass pseudocode → W A11 check you understand (*) similarity of forward and backward pass algos.