DL2: Linear regression model

Linear model: $Y = X\beta + \varepsilon$, $\varepsilon \sim N(0, \sigma^2)$

Y = dependent var / outcome var / response var

X = p-dim independent var, predictor var, explanatory var

β = p-dim regression coefficients: β_i: the effect on the dependent var when increasing the ith independent var by 1 unit, holding all other (p-1) predictors constant.

ε = random variable (error) assumed to be normal.

Assume restricted (linear) form

$$Y = f(x_1, \cdots, x_p) + \varepsilon \rightarrow Y = \underbrace{x_1\beta_1 + x_2\beta_2 + \cdots + x_p\beta_p}_{\text{fixed part}} + \underbrace{\varepsilon}_{\text{random part}}$$

- $E[Y] = X\beta$,
  but for fixed x^*, Y differs from $E[Y]$ by a random ε

Parameter estimation → obtained by the principle of least squares.

- minimize error function: $\{ \underset{n\times1}{Y} = \underset{n\times p}{X}\,\underset{p\times1}{\beta} + \underset{n\times1}{\varepsilon} \}$

$$J(\beta) = (y - X\beta)'(y - X\beta) = y'y - 2\beta'X'y + \beta'X'X\beta$$

normal eq $\nabla J(\beta) = -2X'y + 2X'X\beta = 0$

$$\hat{\beta} = (X'X)^{-1}X'y$$

- $\text{cov}(\hat{\beta}) = \text{cov}((X'X)^{-1}X'y) = (X'X)^{-1}X'\text{cov}(y) = (X'X)^{-1}X'\sigma^2((X'X)^{-1}X')' = \sigma^2(X'X)^{-1}$

$\Rightarrow \sigma^2(X'X)^{-1} \approx \hat{\sigma}^2(X'X)^{-1}$

- standard error of $\hat{\beta}_i$:

$$\sqrt{[\hat{\sigma}^2(X'X)^{-1}]_{i,i}} \leftarrow \text{diagonal elements}$$

- Error sum of squares (SSE):

$$SSE = (y - \hat{y})'(y - \hat{y}) = \hat{\varepsilon}'\underbrace{\hat{\varepsilon}}_{\text{residual}}$$

- Coefficient of determination

$$r^2 = 1 - \frac{SSE}{SST}, \quad SST = (y - \bar{y})'(y - \bar{y})$$

- Estimated σ^2 (variance of error):

$$\hat{\sigma}^2 = \frac{(y - \hat{y})'(y - \hat{y})}{n - p - 1} = \frac{SSE}{n - p - 1}$$

- categorical Independent variable:
  → set up k-1 dummy variables for k levels

- Hypothesis testing: omnibus test

$H_0: \beta_1 = \beta_2 = \cdots = \beta_p = 0$

H_1: at least one $\beta_k \neq 0$

Source	df	Sum of Squares	Mean Squares
Regression	p	$SSR = (\hat{y} - \bar{y})'(\hat{y} - \bar{y})$	SSR/p
Error	$n-p-1$	$SSE = (y - \hat{y})'(y - \hat{y})$	$SSE/(n-p-1)$
Total	$n-1$	$SST = (y - \bar{y})'(y - \bar{y})$	

Table: ANOVA table

$$F = \frac{SSR/p}{SSE/(n-p-1)} = \frac{MSR}{MSE} \sim F_{\alpha,p,n-p-1}$$

significance level.

- F-test for subsets of independent variables

Full: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \varepsilon$

Reduced: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon$

Hypothesis:

H_0: reduced model with I predictors ($=2$)

H_1: full model with p-predictors ($=4$)

$\xrightarrow{\text{ANOVA}}$ $F = \dfrac{(SSE_R - SSE_F)/(p - I)}{SSE_F/(n-p-1)} \sim F_{\alpha, p-I, n-p-1}$

Hypothesis tests for individual regression coefficients

$H_0: \hat{\beta}_i = 0$

$H_1: \hat{\beta}_i \neq 0$

$$T = \frac{\hat{\beta}_i}{se(\hat{\beta}_i)} \sim T_{\alpha/2, n-p-1}$$

standard error of $\hat{\beta}_i$

$$\sqrt{[\hat{\sigma}^2(X'X)^{-1}]_{i,i}} \Leftarrow \sqrt{var(\hat{\beta})}$$

DL4: Regularization: Ridge regression & Lasso

assessing whether $\hat{f}(x) = X\hat{\beta}$ is a good model:

1) $\hat{\beta}$ close to β? $\Rightarrow \hat{\beta} - \beta$ = bias
2) will $\hat{f}(x)$ fit new observations well? (prediction)

1) Bias

- MSE for $\hat{\beta}$

$$MSE(\hat{\beta}) = E[\|\hat{\beta}-\beta\|^2] = E[(\hat{\beta}-\beta)'(\hat{\beta}-\beta)]$$

2) Prediction

- good fitting for current data doesn't guarantee good prediction for new data

small prediction errors (MSPE)

$$MSPE(X_0) = E[(\hat{f}(X_0) - Y)'(\hat{f}(X_0) - Y)]$$
$$= Bias^2(\hat{f}(X_0)) + Var(\hat{f}(X_0)) + \sigma^2 \quad , Bias(\hat{f}(X_0)) = E[\hat{f}(X_0)] - E[Y]$$

bias- variance trade off.

- complex model (more terms included)
  - Bias ↓ (Bias($\hat{f}(X_0)$)↓)
  - variance ↑ (more sensitive to new data)
  - ⇒ Introducing small bias can lead to substantial derease in variane (MSPE↓)

example

case ①: $y \leftarrow (X_1, \cdots, X_{1000})$

Bias ↓ ⇒ more independent var → ↓ SSE

Variance ↑ ⇒ adding too many var → variance ↑ for new, unobserved point.

→ less sensitive to new data

case ②: $y \leftarrow (X_1, X_2, X_3)$

Bias ↑

variance ↓ ⇒ more robust as only using 3 variables.

By Ridge or Lasso ⇒ Bias ↑ , variance ↓

ΔBias < Δvariance ⇒ ↓ MSPE

1. Ridge regression & regularization

- unconstrained β can lead to high variance ⇒ regularize β to control variance
- obtain ridge coefficients with constraints:

objective functions: $\min (y - X\beta)'(y - X\beta)$ s.t. $\sum_{j=1}^{p} \beta_j^2 \le t$

- equivalent to minimizing l_2 loss function:

$$J_2(\beta) = (y - X\beta)'(y - X\beta) + \lambda\|\beta\|_2$$
$$\frac{d}{d\beta} J_2(\beta) = -2X'y + 2X'X\beta + 2\lambda\beta = 0$$
$$\hat{\beta}^{\lambda}_{ridge} = (X'X + \lambda I_p)^{-1} X'y$$

Neural Network (NN)

1) Forward propagation :

- Start with initial weights , calculate output
- Error(actual output − predicted output) calculated

2) Backward 〃 :

- weights are updated by reducing the error term in each iteration.
- gradient descent is used

1)

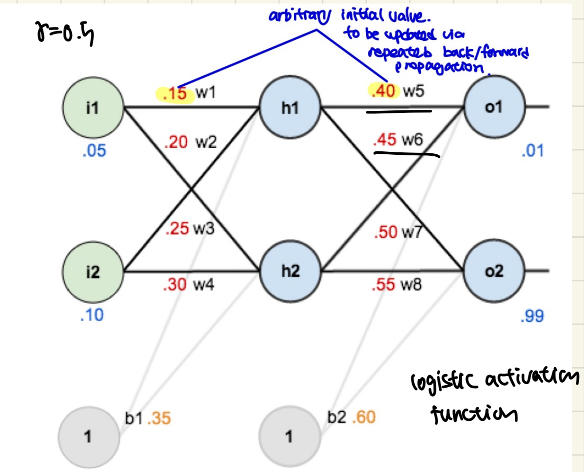
n inputs — weight param w — k hidden layers — weight param w^*

input $\longrightarrow$ hidden $\longrightarrow$ output

$net_{h_1} = \sum_{i=1}^{n} w_i input_i + b_1$, $\ldots$, net_{h_k} $\Rightarrow$ $f(net_{h_1}) = out_{h_1}$, $f(net_{h_2}) = out_{h_2}$, $\ldots$, $f(net_{h_k}) = out_{h_k}$ $\Rightarrow$ $\boxed{net_o = \sum_{i=1}^{5} w_i^* out_{h_i}}$ $\Rightarrow$ $out_o = f(net_o)$

linear comb of weights.

Total error : $E = \frac{1}{2}(target - out_o)^2$



① calculation of input → hidden layer

$w_1 = 0.15$ $w_2 = 0.20$ $w_3 = 0.25$ $w_4 = 0.30$
$i_1 = 0.05$ $i_2 = 0.10$ $b_1 = 0.35$ $b_2 = 0.60$

$net_{h_1} = w_1 i_1 + w_2 i_2 + b_1 = 0.3775$
$net_{h_2} = w_3 i_1 + w_4 i_2 + b_1 = 0.3925$
$out_{h_1} = \frac{1}{1+e^{-net_{h_1}}} = 0.59327$
$out_{h_2} = \frac{1}{1+e^{-net_{h_2}}} = 0.59688$

② calculation of hidden layer → output

$net_{o_1} = out_{h_1} \cdot w_5 + out_{h_2} \cdot w_6 + b_2 = 1.1059$
$net_{o_2} = out_{h_1} \cdot w_7 + out_{h_2} \cdot w_8 + b_2 = 1.2249$
$out_{o_1} = \frac{1}{1+e^{-net_{o_1}}} = 0.75136$
$out_{o_2} = \frac{1}{1+e^{-net_{o_2}}} = 0.77292$

③ Calculation of total error

$\Rightarrow$ $out_{o_1} = 0.75136$, $out_{o_2} = 0.77292$
$o_1 = 0.01$, $o_2 = 0.99$
$E = \frac{1}{2}(o_1 - out_{o_1})^2 + \frac{1}{2}(o_2 - out_{o_2})^2$
$= 0.29837$
⇒ back propagation to update w & b to minimize E.

2)

- Investigate how much a change in w_i can affect the total error E :

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial out_o} \frac{\partial out_o}{\partial net_o} \frac{\partial net_o}{\partial w_i}$$

- Update weights in the network :

$$w_i^+ = w_i - \gamma \frac{\partial E}{\partial w_i}$$

① Backward Pass : output layer. (w_5, w_6, w_7, w_8)

ex) w_5

$$\frac{\partial E}{\partial w_5} = \underbrace{\frac{\partial E}{\partial out_{o_1}}}_{(1)} \underbrace{\frac{\partial out_{o_1}}{\partial net_{o_1}}}_{(2)} \underbrace{\frac{\partial net_{o_1}}{\partial w_5}}_{(3)}$$

(1) $\frac{\partial E}{\partial out_{o_1}}$. $E = E_{o_1} + E_{o_2} = \frac{1}{2}(target_{o_1} - out_{o_1})^2 + \frac{1}{2}(target_{o_2} - out_{o_2})^2$

$\frac{\partial E}{\partial out_{o_1}} = -(target_{o_1} - out_{o_1}) = -(0.01 - 0.7531) = +0.74136$

(2) $\frac{\partial out_{o_1}}{\partial net_{o_1}}$ $out_{o_1} = \frac{1}{1+e^{-net_{o_1}}}$

$$\frac{\partial out_{o_1}}{\partial net_{o_1}} = \frac{e^{-net_{o_1}}}{(1+e^{-net_{o_1}})^2} = \frac{1+e^{-net_{o_1}} - 1}{(1+e^{-net_{o_1}})^2} = \frac{1+e^{-net_{o_1}}}{(1+e^{-net_{o_1}})^2} - \frac{1}{(1+e^{-net_{o_1}})^2}$$

$$= \frac{1}{1+e^{-net_{o_1}}} - \frac{1}{(1+e^{-net_{o_1}})^2} = out_{o_1} - (out_{o_1})^2 = out_{o_1}(1 - out_{o_1})$$

$$= 0.7513(1 - 0.7513) = 0.1868$$

(3) $\frac{\partial net_{o_1}}{\partial w_5}$ $net_{o_1} = out_{h_1} \cdot w_5 + out_{h_2} \cdot w_6 + b_2$

$\frac{\partial net_{o_1}}{\partial w_5} = out_{h_1} = 0.59327$

$\therefore \frac{\partial E}{\partial w_5} = 0.74136 \times 0.1868 \times 0.59327 = 0.08216704.$

② Backward pass : update w_5

$$w_5^+ = w_5 - \gamma \frac{\partial E}{\partial w_5} = 0.4 - 0.5 \times 0.082167 = 0.3589.$$

:

repeat for w_6^+, w_7^+, w_8^+ [w_5, w_6) → net_{o_1} → out_{o_1} → E ; w_7, w_8) → net_{o_2} → out_{o_2} → E

③ Backward pass . Hidden layer (w_1, w_2, w_3, w_4)

ex) $w_1 \to net_{h1} \to out_{h1} \to net_{o1} \to out_{o1} \to E_{o1}$
$\qquad\qquad\qquad\qquad\qquad \to net_{o2} \to out_{o2} \to E_{o2}$ $\Rightarrow E$) more complicated.

$$\frac{\partial E}{\partial w_1} = \underbrace{\frac{\partial E}{\partial out_{h1}}}_{(1)} \cdot \underbrace{\frac{\partial out_{h1}}{\partial net_{h1}}}_{(2)} \cdot \underbrace{\frac{\partial net_{h1}}{\partial w_1}}_{(3)}$$

(1) $\frac{\partial E}{\partial out_{h1}} = \frac{\partial E_{o1}}{out_{h1}} + \frac{\partial E_{o2}}{out_{h1}}$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{o1}} \cdot \frac{\partial out_{o1}}{\partial net_{o1}} \cdot \frac{\partial net_{o1}}{\partial out_{h1}}$$

$net_{o1} = out_{h1} \cdot w_5 + out_{h2} \cdot w_6 + b_2$

$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5$

$= 0.74136 \times 0.1868 \times 0.4$

$= 0.05539$

$$\frac{\partial E_{o2}}{\partial out_{h1}} = \frac{\partial E_{o2}}{\partial out_{o2}} \cdot \frac{\partial out_{o2}}{\partial net_{o2}} \cdot \frac{\partial net_{o2}}{\partial out_{h1}}$$

$=$

(2) $\frac{\partial out_{h1}}{\partial net_{h1}}$ $\qquad out_{h1} = \frac{1}{1+e^{-net_{h1}}}$

$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1-out_{h1}) = 0.59327(1-0.59327) = 0.2413$

(3) $\frac{\partial net_{h1}}{\partial w_1}$ $\qquad net_{h1} = w_1 i_1 + w_2 i_2 + b_1$

$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$

Multilayer Neural Networks.

↳ work with complex non-linear representations of data

→ adding more layers & neurons ⇒ ↑ specialization to train data, ↓ performance on test data

- perceptron
  - simple perceptron : outputs are binary ⇒ $y = \sum w_i x_i$. If $y>0$, output=1; $y<0$, output=−1
  - multi-class perceptron : many possible labels.

- linear separable classifier
  - If $f(b+\sum w_i x_i) >$ threshold ⇒ output = 1 (active)
  - $f(b+\sum w_i x_i) <$ threshold ⇒ output = −1 (inactive.

  If classification is incorrect, change w. else, don't change w.

- weight updates
  1) get x and output label y
  2) Initialize w, b for f(x)
  3) If f(x)=y, mark as completed; else, fix it
  4) adjust score based on error.
     - $f(x) = \text{sign}(b+\sum w_i x_i)$
     - if y=+1 and f(x)=−1, $\sum w_i x_i$ is too small, make it bigger
       y=−1 and f(x)=+1, $\sum w_i x_i$ is too large, make it smaller.
  5) w=w if f(x)=y, else w=w+yx
  6) b=b if f(x)=y, else $b=b+yR^2$, $R=\max(\|x_i\|)$
  7) repeat 3~6 until f(x)=y

CNN

- Input(image) : 20×20×3 array of numbers, n=1000 ⇒ # of data samples
  (20×20: resolution, ×3: RGB values; 20×20×3: dim of input datasets)

- output : class[cat, dog, …] → probability of classes [0.25, 0.45, 0.1, 0.05, 0.15]

- Structure

Input → Conv → ReLU → ConV → ReLU → Pool → ReLU → Conv → Relu → Pool → FC → output

Conv: filter the image, extract important info from images through dim reduc.

Pool: shrink image size

FC: voting & classification. generates vector of prob

1) convolutional layer.

32×32×3 $\xrightarrow[\text{one 5×5×3 filter, stride=1}]{}$ 28×28×1 conv

32×32×3 $\xrightarrow[\text{two 5×5×3, stride=1}]{}$ 28×28×2 conv

⇒ multiplications between the filter & pixel values.

↑ stride ⇒ overlaps less ⇒ more dimension reduction

⇒ as we keep applying conv layers, the dim of the volume decreases.

⇒ we want to preserve original input volume in the early layers of NN so that we can extract some info.

⇒ Padding to prevent fast volume reduction

※ Padding

32×32×3 → three 5×5×3 filter, but want to keep output volume as 32×32×3

⇒ apply zero padding size 2. → 36×36×3 input volume.

★ $P = \frac{K-1}{2}$ (to keep input & output dim the same)

(P: padding size, K: filter size)

$O = \frac{(W-K+2P)}{S} + 1$

(W: input height/length, S: stride size)

2) ReLU activation function

$y = x\,I(x>0)$

3) Pooling

ex) maxpooling of 2x2 filter with strid 2

$$\left.\begin{array}{cc|cc} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{array}\right. \xrightarrow{\text{take max}} \begin{array}{cc} 6 & 8 \\ 14 & 16 \end{array} \Rightarrow \text{reduce dim}$$

4) FC Layer.

→ at the end of CNN

→ takes an input volume → outputs an N-dim vector of prob.

- optimization of CNN
  - · initial values : randomize fter & filter values (weights)
  - · tuning param : stride, padding, filter size
  - · minimize error : $E = \frac{1}{2}(\text{target}-\text{out})^2$

Back propagation : CNN

Assume we have

$$\frac{\partial E}{\partial net} = \frac{\partial E}{\partial out}\frac{\partial out}{\partial net}$$

Backpropagation

· $\frac{\partial E}{\partial b}$: how the change in b can affect E

$$b^+ = b - r\frac{\partial E}{\partial b}$$

· $\frac{\partial E}{\partial w}$: how the change in w " " E

$$w^+ = w - r\frac{\partial E}{\partial w}$$

· $\frac{\partial E}{\partial x}$: how the change in pixel x_{ij} can affec E

1-Dim Input Example

if stride = 1

$$(\ \boxed{x_1}\ \boxed{x_2}\ x_3\ x_4\ )$$
$w_1 \quad w_2$

· Input : $x = (x_1, x_2, x_3, x_4)$

$w = (w_1, w_2), b$

· Net : $net = (net_1, net_2, net_3)$

$$x_1w_1 + x_2w_2 + b = net_1$$
$$x_2w_1 + x_3w_2 + b = net_2$$
$$x_3w_1 + x_4w_2 + b = net_3$$

applying 1 kernal, output volume of 3x1 dim

$$\frac{\partial E}{\partial b} = \frac{\partial E}{\partial net}\frac{\partial net}{\partial b}$$

$$= \begin{bmatrix} \frac{\partial E}{\partial net_1} & \frac{\partial E}{\partial net_2} & \frac{\partial E}{net_3} \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \frac{\partial E}{\partial net_1} + \frac{\partial E}{\partial net_2} + \frac{\partial E}{\partial net_3}$$

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial net}\boxed{\frac{\partial net}{\partial w}} = \begin{bmatrix} x_1 & x_2 \\ x_2 & x_3 \\ x_3 & x_4 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial E}{\partial net_1} & \frac{\partial E}{\partial net_2} & \frac{\partial E}{\partial net_3} \end{bmatrix} \begin{bmatrix} x_1 & x_2 \\ x_2 & x_3 \\ x_3 & x_4 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{\partial E}{\partial net_1}x_1 + \frac{\partial E}{\partial net_2}x_2 + \frac{\partial E}{\partial net_3}x_3 \\ \frac{\partial E}{\partial net_1}x_2 + \frac{\partial E}{\partial net_2}x_3 + \frac{\partial E}{\partial net_3}x_4 \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial w_1} \\ \frac{\partial E}{\partial w_2} \end{bmatrix}$$

$$\frac{\partial E}{\partial x} = \begin{bmatrix} \frac{\partial E}{\partial net_1} & \frac{\partial E}{\partial net_2} & \frac{\partial E}{\partial net_3} \end{bmatrix} \begin{bmatrix} w_1 & w_2 & 0 & 0 \\ 0 & w_1 & w_2 & 0 \\ 0 & 0 & w_1 & w_2 \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} \frac{\partial E}{\partial x_1} \\ \frac{\partial E}{\partial x_2} \\ \frac{\partial E}{\partial x_3} \\ \frac{\partial E}{\partial x_4} \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial net_1}w_1 \\ \frac{\partial E}{\partial net_1}w_2 + \frac{\partial E}{\partial net_2}w_1 \\ \frac{\partial E}{\partial net_2}w_2 + \frac{\partial E}{\partial net_3}w_1 \\ \frac{\partial E}{\partial net_3}w_2 \end{bmatrix}$$

2dim input example.

Input:

- $X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & x_{14} \\ x_{21} & & & \\ x_{31} & & \cdots & \\ x_{41} & & & \end{bmatrix}$

- $W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}$

- b

Output

- $net = \begin{bmatrix} net_{11} & net_{12} & net_{13} \\ net_{21} & net_{22} & net_{23} \\ net_{31} & net_{32} & net_{33} \end{bmatrix}$

<conv with one filter w, stride=1, padding=0>

$net_{11} = w_{11}x_{11} + w_{12}x_{12} + w_{21}x_{21} + w_{22}x_{22} + b$

$net_{12} = w_{11}x_{12} + w_{12}x_{13} + w_{21}x_{22} + w_{22}x_{23} + b$

$\vdots$

$\Rightarrow$ 3x3 output volume

$* \; net_{ij} = \left(\sum_{k=1}^{2}\sum_{l=1}^{2} w_{kl}\, x_{i+k-1,\, j+l-1}\right) + b \;, \quad \forall i, j \in \{1,2,3\}^2$

RNN

↳ unlike feed forward networks, RNNs can use internal memory for their processing.

↳ RNN takes the previous output or hiddenstates as inputs.

The input at time t has some historical information at time $T<t$.

↳ Intermediate values (state) of RNNs can save info about past inputs

Equations

num of hidden nodes — dim of input vector x

- Hidden nodes : $\underset{}{\overset{d\times 1}{a_t}} = \overset{d\times d}{W_H}\,\overset{d\times 1}{h_{t-1}} + \overset{d\times k}{W_x}\,\overset{k\times 1}{x_t}$

$2dk+d^2$ → total # of weight param that you have to estimate.

- Output from hidden state : $\overset{d\times 1}{h_t} = \tanh(a_t)$
- Prediction at time t : $\overset{k\times 1}{y_t} = \mathrm{softmax}(\overset{k\times d}{W_y}\,\overset{d\times 1}{h_t})$

↳ we only train RNN one time from each randomized weights W_x, W_y, W_H.

Repeat optimization via backpropagation. ⇒ W_x, W_y, W_H will change.

RNN updates

$$W_x^+ = W_x - \gamma\frac{\partial L}{\partial W_x}$$

$$W_y^+ = W_y - \gamma\frac{\partial L}{\partial W_y}$$

$$W_H^+ = W_H - \gamma\frac{\partial L}{\partial W_H}$$

multi-class cross entropy Loss function.

$$L(y,\hat{y}) = -y'\log(\hat{y})$$

- Binary ex)
  - $y = (0.99, 0.01)$ $\hat{y}_1 = (0.99, 0.01)$ $\hat{y}_2 = (0.01, 0.99)$
  - loss for $\hat{y}_1$: $L(y,\hat{y}_1) = -y'\log(\hat{y})$

$$L_{total}(y,\hat{y}) = L_1(y_1,\hat{y}_1) + L_2(y_2,\hat{y}_2) + L_3(y_3,\hat{y}_3)$$

- chainrule for updating W_y.

$$z_t = W_y h_t \rightarrow \hat{y}_t = \mathrm{softmax}(z_t) \rightarrow L_t = -y_t'\log(\hat{y}_t)$$

chain rule : $\dfrac{\partial L_t}{\partial W_y} = \dfrac{\partial L_t}{\partial \hat{y}_t}\cdot\dfrac{\partial \hat{y}_t}{\partial z_t}\cdot\dfrac{\partial z_t}{\partial W_y}$

Gradient : $\dfrac{\partial L_{total}}{\partial W_y} = \sum_t \dfrac{\partial L_t}{\partial W_y}$

update : $W_y^+ = W_y - \gamma\dfrac{\partial L_{total}}{\partial W_y}$

Chain rule for updating W_x

$$h_t = \tanh(W_H h_{t-1} + W_x x_t) \rightarrow z_t = W_y h_t \rightarrow \hat{y}_t = \mathrm{softmax}(z_t) \rightarrow L_t = -y_t'\log(\hat{y}_t)$$

Chainrule :

$$\frac{\partial L_t}{\partial W_x} = \frac{\partial L_t}{\partial \hat{y}_t}\cdot\frac{\partial \hat{y}_t}{\partial z_t}\cdot\frac{\partial z_t}{\partial h_t}\frac{\partial h_t}{\partial W_x} + \frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial z_t}\frac{\partial z_t}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial W_x} + \cdots$$

$$\frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial z_t}\cdot\frac{\partial z_t}{\partial h_t}\frac{\partial h_t}{\partial h_{t-n}}\frac{\partial h_{t-n}}{\partial W_x} = \sum_{k=0}^{t}\frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial z_t}\frac{\partial z_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W_y}$$

gradient : $\dfrac{\partial L_{total}}{\partial W_x} = \sum_{t=1}^{n}\sum_{k=0}^{t}\dfrac{\partial L_t}{\partial \hat{y}_t}\dfrac{\partial \hat{y}_t}{\partial z_t}\dfrac{\partial z_t}{\partial h_t}\dfrac{\partial h_t}{\partial h_k}\dfrac{\partial h_k}{\partial W_y}$

update : $W_x^+ = W_x - \gamma\dfrac{\partial L_{total}}{\partial W_x}$

chain rule for updating W_H

$$h_t = \tanh(W_H h_{t-1} + W_x x_t) \rightarrow z_t = W_y h_t \rightarrow \hat{y}_t = \mathrm{softmax}(z_t) \rightarrow L_t = -y_t'\log(\hat{y}_t)$$

chain rule :

$$\frac{\partial L_t}{\partial W_H} = \frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial z_t}\frac{\partial z_t}{\partial h_t}\frac{\partial h_t}{\partial W_H} + \frac{\partial L_t}{\partial \hat{y}_t}\frac{\partial \hat{y}_t}{\partial z_t}\frac{\partial z_t}{\partial h_t}\frac{\partial h_t}{\partial h_{t-1}}\frac{\partial h_{t-1}}{\partial W_x} + \cdots$$

RNN issue : vanishing gradient (+ deep NNs)

∵ weight update requires chain rule which has lots of components.

so If each component close to 0, gradient becomes 0 ⇒ no update.

↳ happens because we assume all the precious informations $(h_1, \cdots, h_n)$ gives impact on h_{t}.

↳ Solution : LSTM.

LSTM : have gates that make the long/short term memory to be stored.

forget gate & inputgate decides which information to be abandoned & stored.