$$\begin{bmatrix} w_{11} & \cdots & w_{1m} \\ \vdots & \ddots & \vdots \\ w_{n1} & \cdots & w_{nm} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} + \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$$

$n \times m$ matrix

↑ rows ↑ cols

Let $\left\{ \begin{array}{l} V_1 = w_{11}x_1 + w_{12}x_2 + \cdots + w_{1m}x_m + b_1 \\ V_2 = w_{21}x_1 + w_{22}x_2 + \cdots + w_{2m}x_m + b_2 \\ \vdots \\ V_n = w_{n1}x_1 + w_{n2}x_2 + \cdots + w_{nm}x_m + b_n \end{array} \right\}$ be the results of each row

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \begin{pmatrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{pmatrix} \begin{matrix} V_1 \\ V_2 \\ V_3 \end{matrix}$$

softmax formula: $y_i = \text{softmax}(V_i)$ $\sum_j e^{V_j} = V = e^{V_1} + \cdots + e^{V_n}$

$$= \frac{e^{V_i}}{\sum_j e^{V_j}} = \frac{e^{V_i}}{V}$$

Equation To Minimize: $J(y) = -\sum_i y_i' \ln(y_i)$ (actual ↙, prediction ↙)

$$= -\left[y_1' \ln\left(\frac{e^{V_1}}{V}\right) + \cdots + y_n' \ln\left(\frac{e^{V_n}}{V}\right) \right]$$

$$= -\left[y_1'(\ln e^{V_1} - \ln V) + \cdots + y_n'(\ln e^{V_n} - \ln V) \right]$$

$$= -\left[y_1'(V_1 - \ln V) + \cdots + y_n'(V_n - \ln V) \right]$$

$$= -\left[(y_1' V_1 + \cdots + y_n' V_n) - \ln V (y_1' + \cdots + y_n') \right]$$

$$= -\left[y_1' V_1 + \cdots + y_n' V_n - \ln V \left(y_1' + \cdots + y_n' \right) \right]$$

recall $V_i = \omega_{i1} x_1 + \omega_{i2} x_2 + \cdots + \omega_{im} x_m + b_i$

$$\frac{\partial J}{\partial \omega_{ij}} = -\left[y_i' \frac{\partial V_i}{\partial \omega_{ij}} - \left(\frac{\partial}{\partial \omega_{ij}} \ln V \right)\left(y_1' + \cdots + y_n' \right) \right]$$

call it $Y = 1$ b/c "one hot" encoding

all V except V_i become 0 when taking $\frac{\partial}{\partial \omega_{ij}}$

$$\Rightarrow \frac{\partial V_i}{\partial \omega_{ij}} = x_j$$

$$\frac{\partial}{\partial \omega_{ij}} \ln V = \frac{\partial}{\partial \omega_{ij}} \ln \left(e^{V_1} + \cdots + e^{V_n} \right)$$

→ only e^{V_i} remains

$$= \frac{1}{e^{V_1} + \cdots + e^{V_n}} \frac{\partial}{\partial \omega_{ij}} \left[e^{V_1} + \cdots + e^{V_n} \right]$$

$$= \frac{1}{e^{V_1} + \cdots + e^{V_n}} \left(\frac{\partial}{\partial \omega_{ij}} e^{V_i} \right)$$

$$= \frac{1}{e^{V_1} + \cdots + e^{V_n}} e^{V_i} \left(\frac{\partial}{\partial \omega_{ij}} V_i \right) \quad \text{by chain rule}$$

$$= \frac{e^{V_i} x_j}{e^{V_1} + \cdots + e^{V_n}} = \frac{e^{V_i} x_j}{V}$$

$$\Rightarrow \frac{\partial J}{\partial \omega_{ij}} = -\left[y_i' x_j - \frac{e^{V_i} x_j Y}{V} \right]$$

$$= \boxed{-x_j \left[y_i' - \frac{e^{V_i}}{V} Y \right]}$$

Gradient Descent: $\omega_{ij} := \omega_{ij} - \alpha \frac{\partial}{\partial \omega_{ij}} J(\omega)$ formula

for each ω update

where α = learning rate

$$\Rightarrow \boxed{\omega_{ij} := \omega_{ij} + \alpha x_j \left[y_i' - \frac{e^{V_i} Y}{V} \right]}$$

$$\Rightarrow \boxed{\omega_{ij} := \omega_{ij} + \alpha x_j \left[y_i' - \frac{e^{v_i}}{V} \right]}$$

Note: $Y = 1$

$$\Rightarrow \omega_{ij} := \omega_{ij} + \alpha x_j \left[\underbrace{y_i'}_{\text{actual}} - \underbrace{\frac{e^{v_i}}{V}}_{\text{prediction}} \right]$$

for multiple training examples,

$$\omega_{ij} := \omega_{ij} + \alpha \sum_k x_j^{(k)} \left[y_i^{(k)} - \frac{e^{v_i(k)}}{V(k)} \right]$$

Stochastic Method can be used

↳ faster by using estimation

```
for (k=0; k<#training examples; k++) {
    for (each ω_ij) {
        ω_ij := ω_ij + α x_j^(k) [ y_i^(k) - e^(v_i(k)) / V(k) ]
    }
}
```

} repeat until convergence.

⇒ only looks @ 1 training example per update

Back Propogation Speed-Up: Calculate all $e^{v_i(k)}$ & $V(k)$ first to check error rate, save the calculated values & use them again to find new values for ω_{ij}

Note: $\begin{bmatrix} \omega_{11} \cdots \omega_{1m} & b_1 \\ \vdots \ddots \vdots & \vdots \\ \omega_{n1} \cdots \omega_{nm} & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_m \\ 1 \end{bmatrix}$

$$\underbrace{\begin{bmatrix} w_{n1} & \cdots & w_{nm} & b_n \end{bmatrix}}_{n \times (m+1) \text{ matrix}} \begin{bmatrix} x_m \\ 1 \end{bmatrix}$$

should be used for training to account for biases (b) w/o modifying the above algorithm.

Misc. Details: → use OpenCV matrices for storage

→ " " functions to matrix multiply to obtain $V_1 \cdots V_n$

→ Store $e^{V_1} \cdots e^{V_n}$ in private vars

→ Store predictions as well