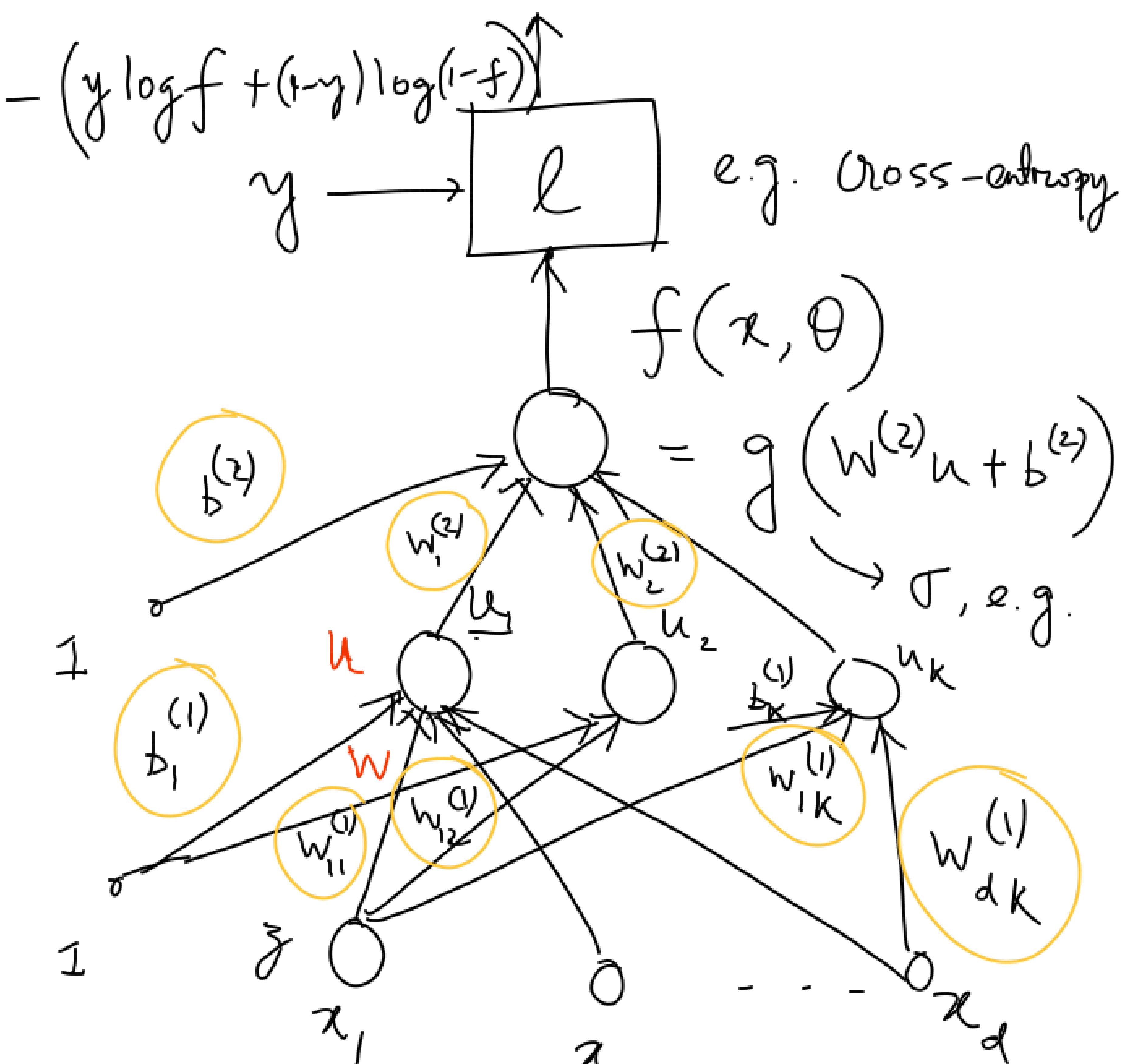
Lec 12: Neural Network Training Algorithm (Feed forward)

Input: $NN(x;\theta)$, training data $(x_i, y_i)_{i=1}^{n}$

loss function ℓ

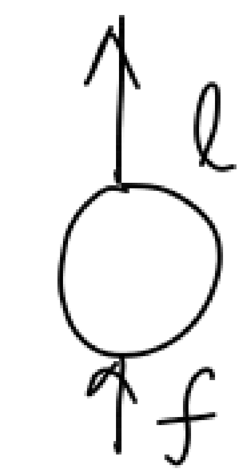
- Randomly initialize θ
- Repeat until stopping criteria is met:
  - Pick randomly a batch of examples $(x_i, y_i)_{i\in B}$, $B \subset N$, $|B| = T < n$ (↑ hyperparameter)
  - Compute the gradient of the loss function $\nabla_\theta \ell$ at θ
  - update $\theta \leftarrow \theta - \eta \nabla_\theta \ell$
- Return θ



$u_i = g(\ldots)$

$$g\left(W^{(1)}_{K\times d}\, x + b^{(1)}\right) = u$$

Goal: learn weights W, b that minimize ℓ



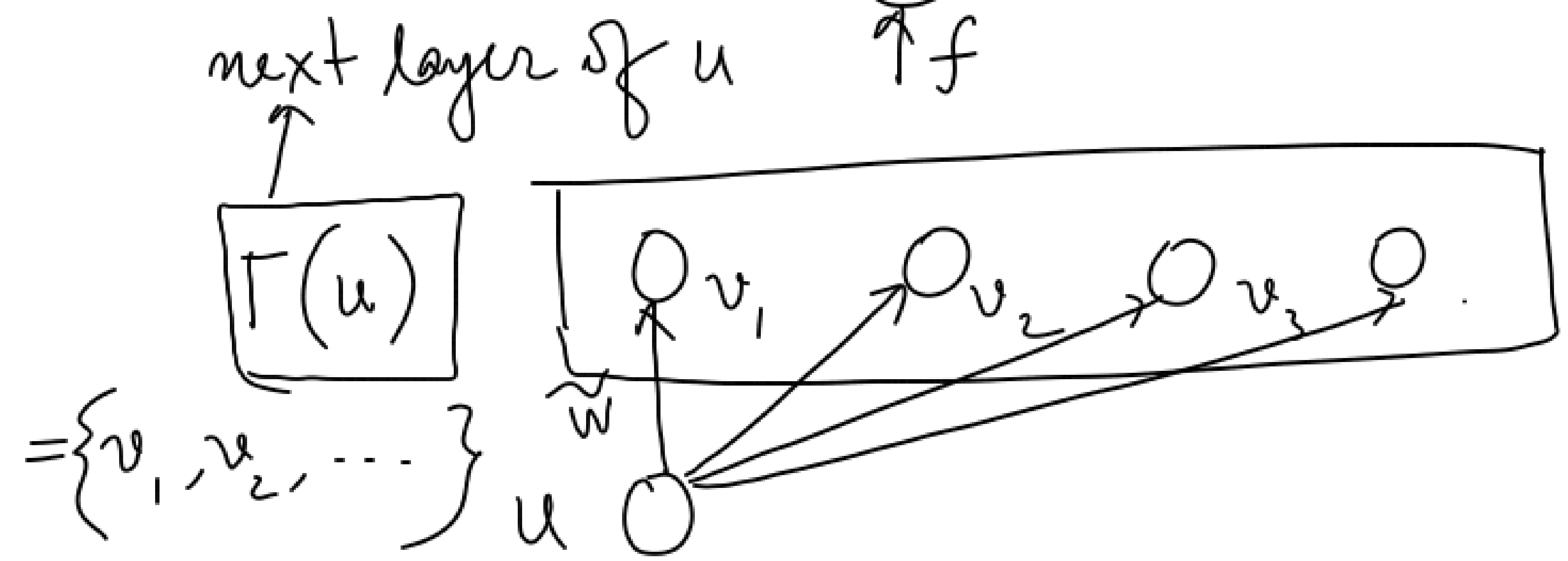
$$\frac{\partial \ell}{\partial \ell} = 1$$

Gradient descent: $\dfrac{\partial \ell}{\partial W_{ij}^{(k)}}$ ← layer

j ← output node

i ← input node of that layer

$$W_{ij}^{(k)} \leftarrow W_{ij}^{(k)} - \eta \frac{\partial \ell}{\partial W_{ij}^{(k)}}$$



Q: How to efficiently compute the gradients?

① Compute $\left.\dfrac{\partial \ell}{\partial u}\right|_{\theta \leftarrow \text{current } \theta}$ for every node u in the NN $\quad \theta = (W, b)$

② Compute $\dfrac{\partial \ell}{\partial w} = \dfrac{\partial \ell}{\partial u} \cdot \dfrac{\partial u}{\partial w}$

$u = g(W z + \dots)$

$\dfrac{\partial u}{\partial w} = g' \cdot z$

$v = g(\tilde{W} u + \dots)$

$$\frac{\partial \ell}{\partial u} = \sum_{v \in \Gamma(u)} \frac{\partial \ell}{\partial v} \cdot \frac{\partial v}{\partial u}$$

Backpropagation: two stage algorithm

Forward pass: Calculate the value of each node given an input (x_i, y_i) at θ

$$u_1 = g\left(W_{11}^{(k)} z_1 + W_{21} z_2 + \cdots + b_1^{(k)}\right)$$

Backward pass: Base case $\frac{\partial \ell}{\partial \ell} = 1$

For each u in a given layer:

For each $v \in T(u)$

- use already computed $\frac{\partial \ell}{\partial v}$
- compute $\left.\frac{\partial v}{\partial u}\right|_{x_i, y_i, \theta}$
- $\frac{\partial \ell}{\partial v} \cdot \frac{\partial v}{\partial u}$

get $\frac{\partial \ell}{\partial u}$, compute $\frac{\partial \ell}{\partial W} = \frac{\partial \ell}{\partial u} \cdot \frac{\partial u}{\partial W}$, $\frac{\partial \ell}{\partial b}$

One example in the webpage:

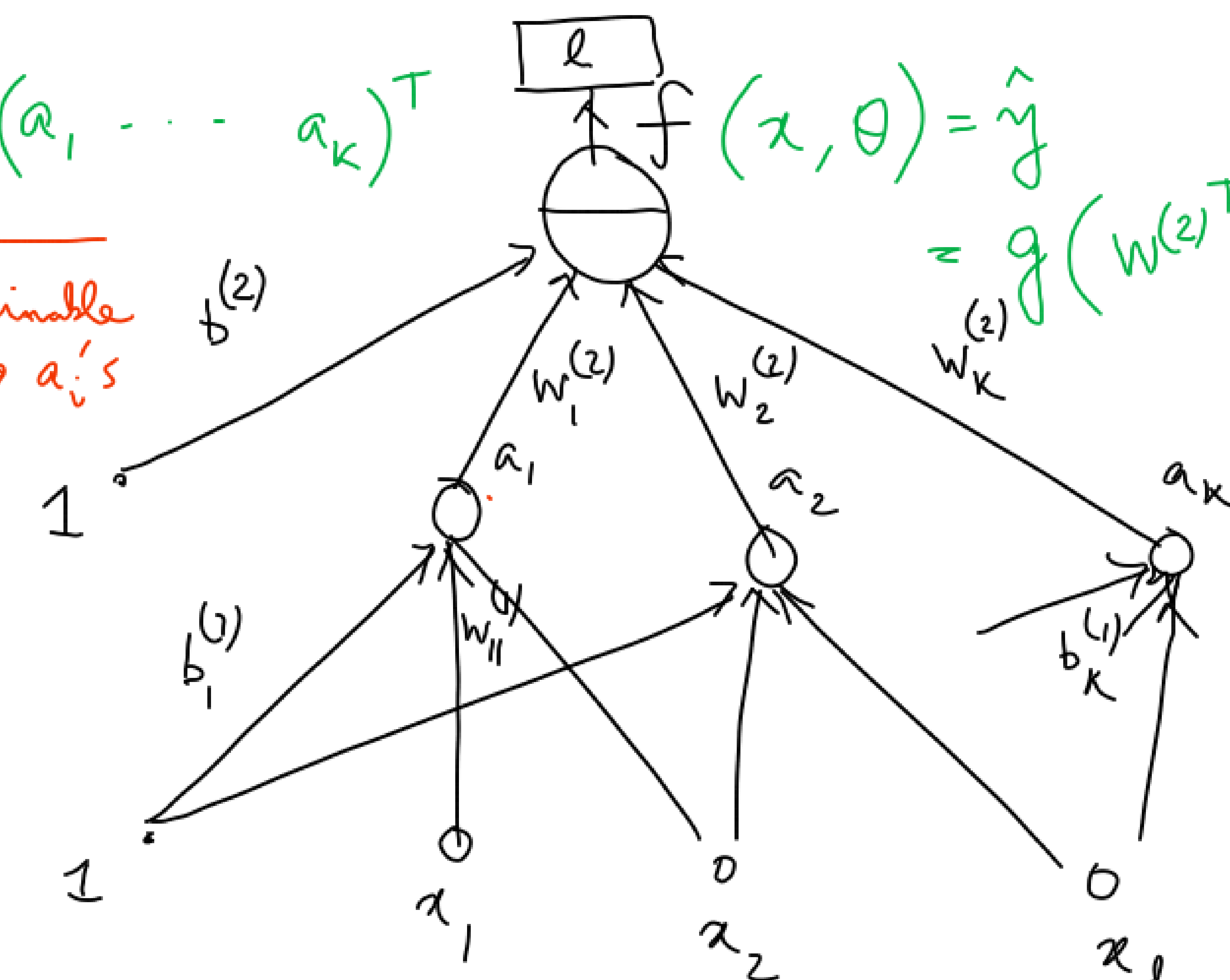
Typical these derivatives are more efficient to compute in vector form.

$\boxed{\text{Computational graph}}$

$\underline{a} = (a_1 \cdots a_K)^T$

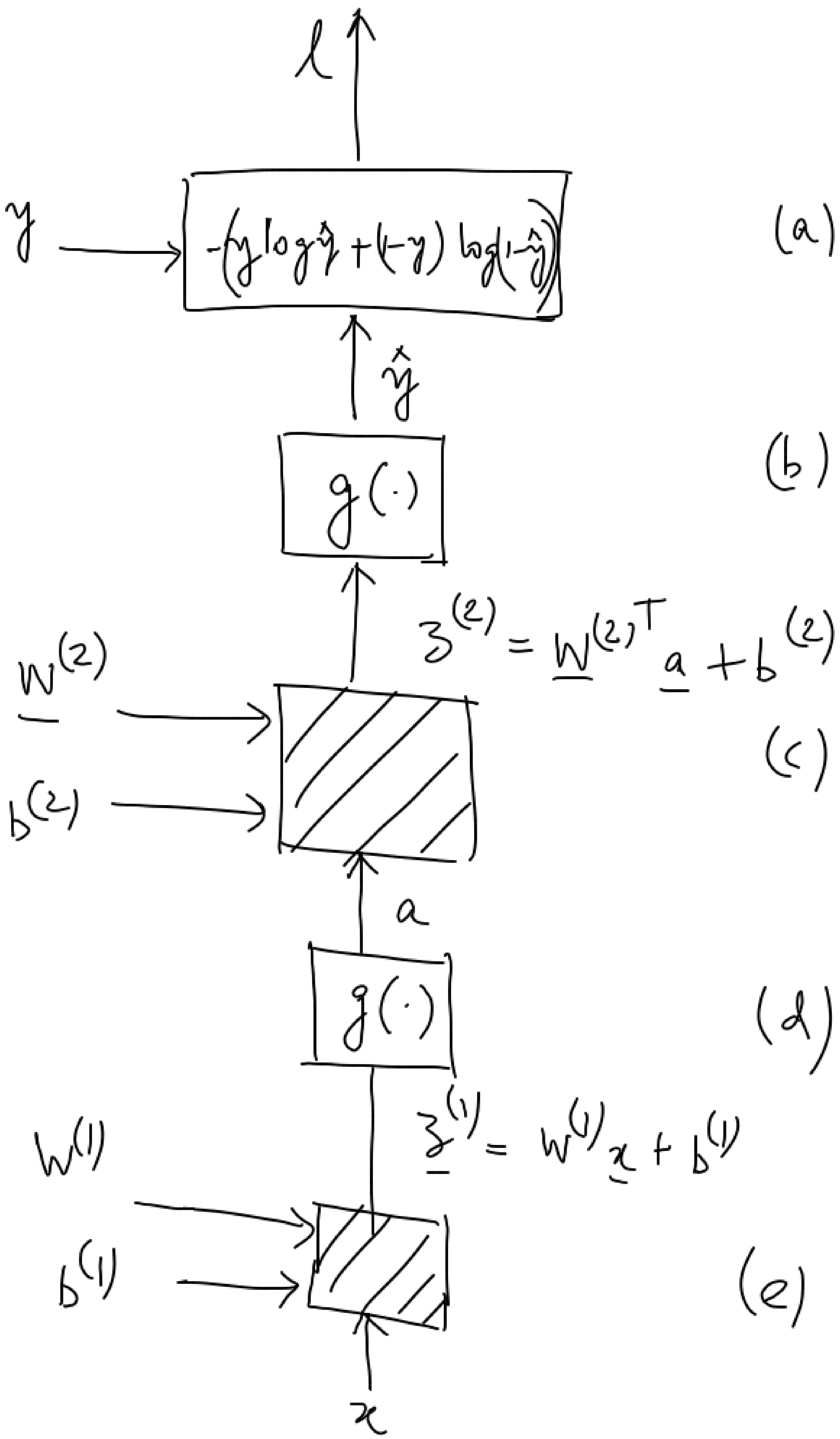
$f(x, \theta) = \hat{y} = g\left(W^{(2)T} \underline{a} + b^{(2)}\right)$

Explainable AI → a_i's



$$\underline{a} = g\left(\underset{K \times d}{W^{(1)}} \underline{x} + \underline{b}^{(1)}\right)$$

$$\theta = \left\{ W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)} \right\}$$



(a) $\longrightarrow$

$$\frac{\partial \ell}{\partial \hat{y}} = -\frac{\partial}{\partial \hat{y}}\left[y \log \hat{y} + (1-y)\log(1-\hat{y})\right] = -\frac{y}{\hat{y}} + \frac{1-y}{1-\hat{y}}$$

$$\delta_{\hat{y}} = \left.\frac{\partial \ell}{\partial \hat{y}}\right|_{x,y,\theta}$$

(b) $\longrightarrow$

$$\frac{\partial \hat{y}}{\partial z^{(2)}} = g'(z^{(2)}), \quad \delta_{z^{(2)}} = \frac{\partial \ell}{\partial z^{(2)}} = \frac{\partial \ell}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z^{(2)}}$$

$$= \delta_{\hat{y}} \cdot \frac{\partial \hat{y}}{\partial z^{(2)}}$$

(c) $\longrightarrow$

$$\frac{\partial z^{(2)}}{\partial \underline{W}^{(2)}} = \frac{\partial}{\partial \underline{W}^{(2)}}\left[W^{(2)T}\underline{a} + b^{(2)}\right] = \underline{a}$$

$$\delta_{W^{(2)}} = \frac{\partial \ell}{\partial \underline{W}^{(2)}}$$

use chain rule.

(d) $\longrightarrow$

$\rightarrow$ HW

(e) $\longrightarrow$

$$\boxed{\frac{\partial z^{(1)}}{\partial W^{(1)}}} = \frac{\partial}{\partial W^{(1)}}\left[W^{(1)}x + b^{(1)}\right] = \boxed{x}$$

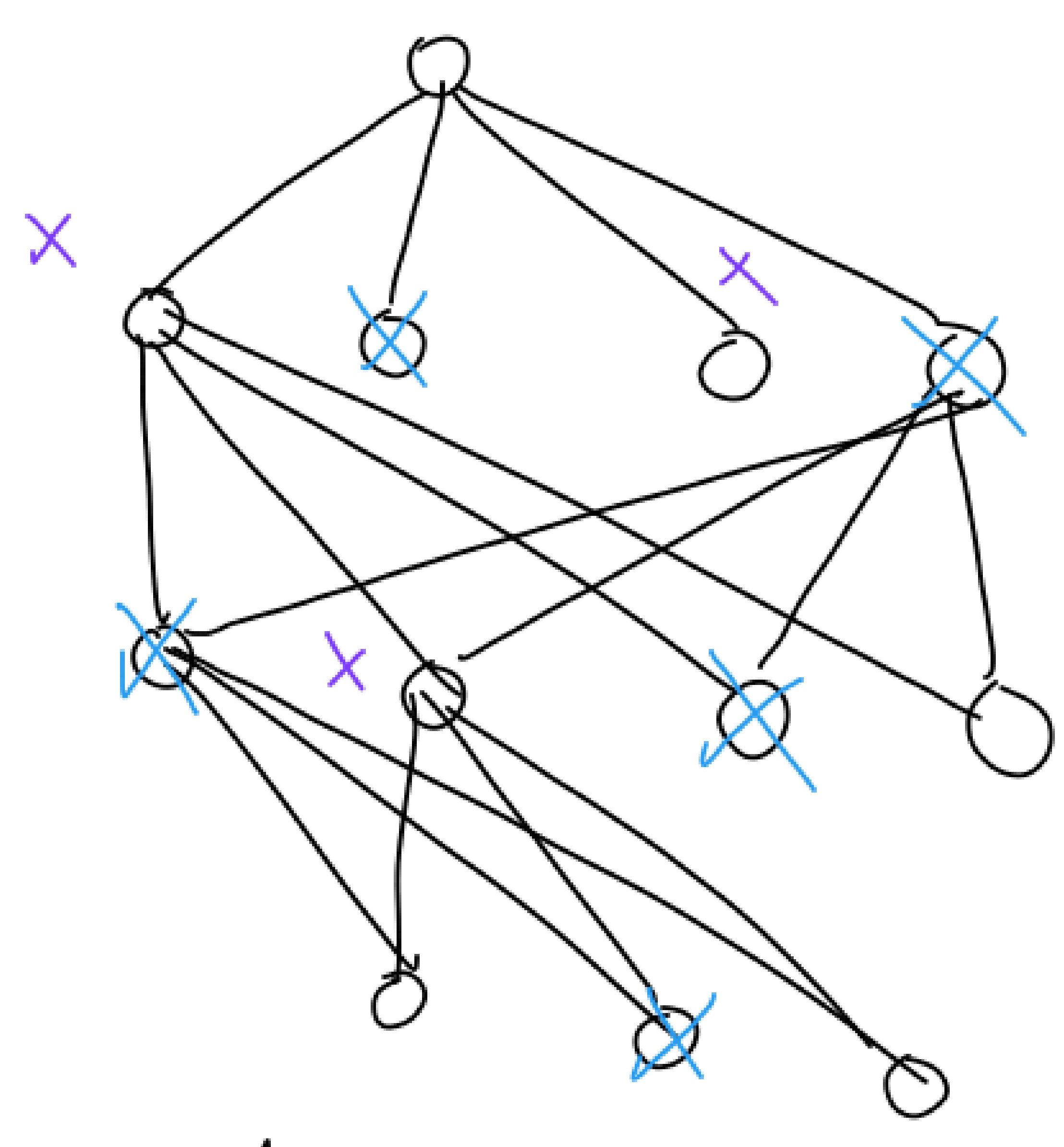
$\delta_{W^{(1)}} \rightarrow$

Regularization:

- L_2 regularization: $\ell_{Reg} = \ell_{CE} + \lambda \|W\|_2^2 \longrightarrow \sum_{ijk} \left(W_{ij}^{(k)}\right)^2$

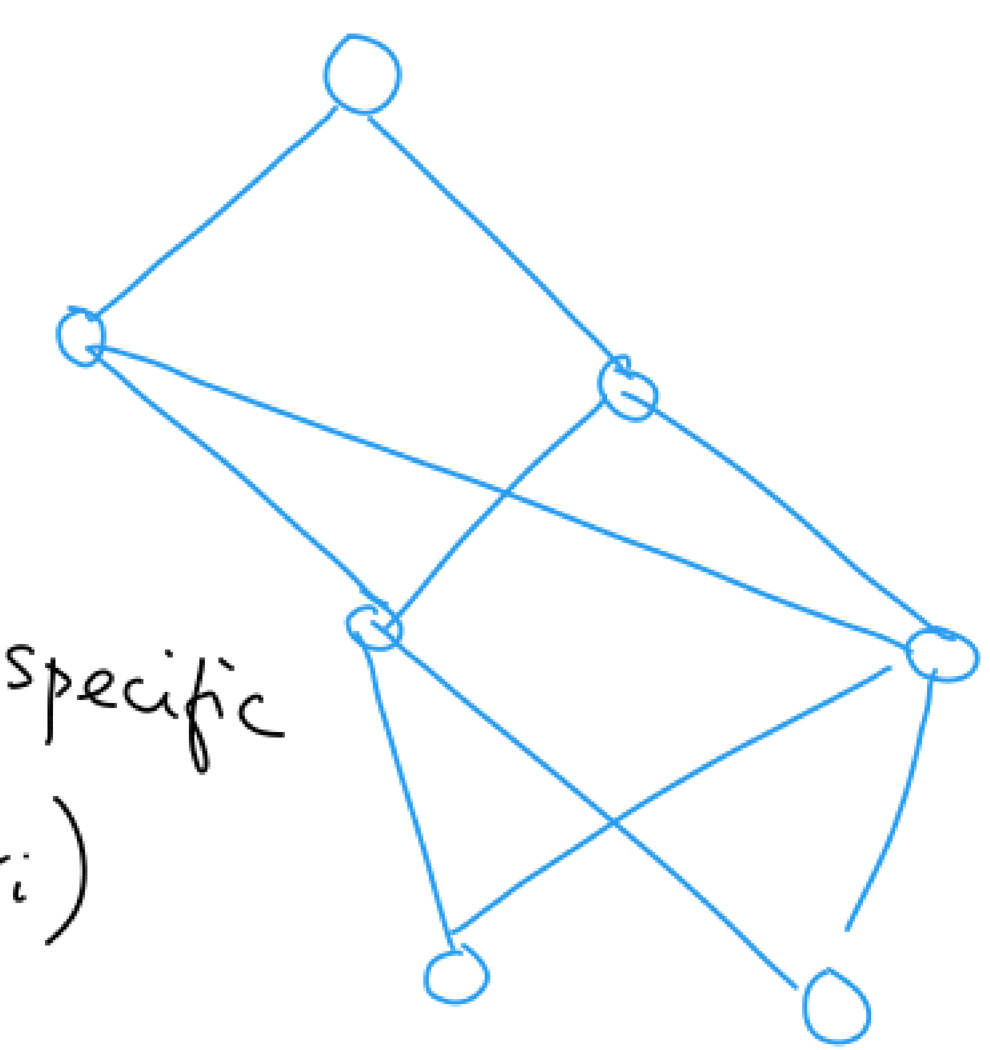
- Dropout regularization:

during training, keep a neuron active w.p. p, ← hyperparameter
delete otherwise



before dropout

for a specific (x_i, y_i)



after dropout

expected weights as an output

at test time → use the entire network with expected weights.

- Early stopping: Stopping when performance on validation set stops improving.