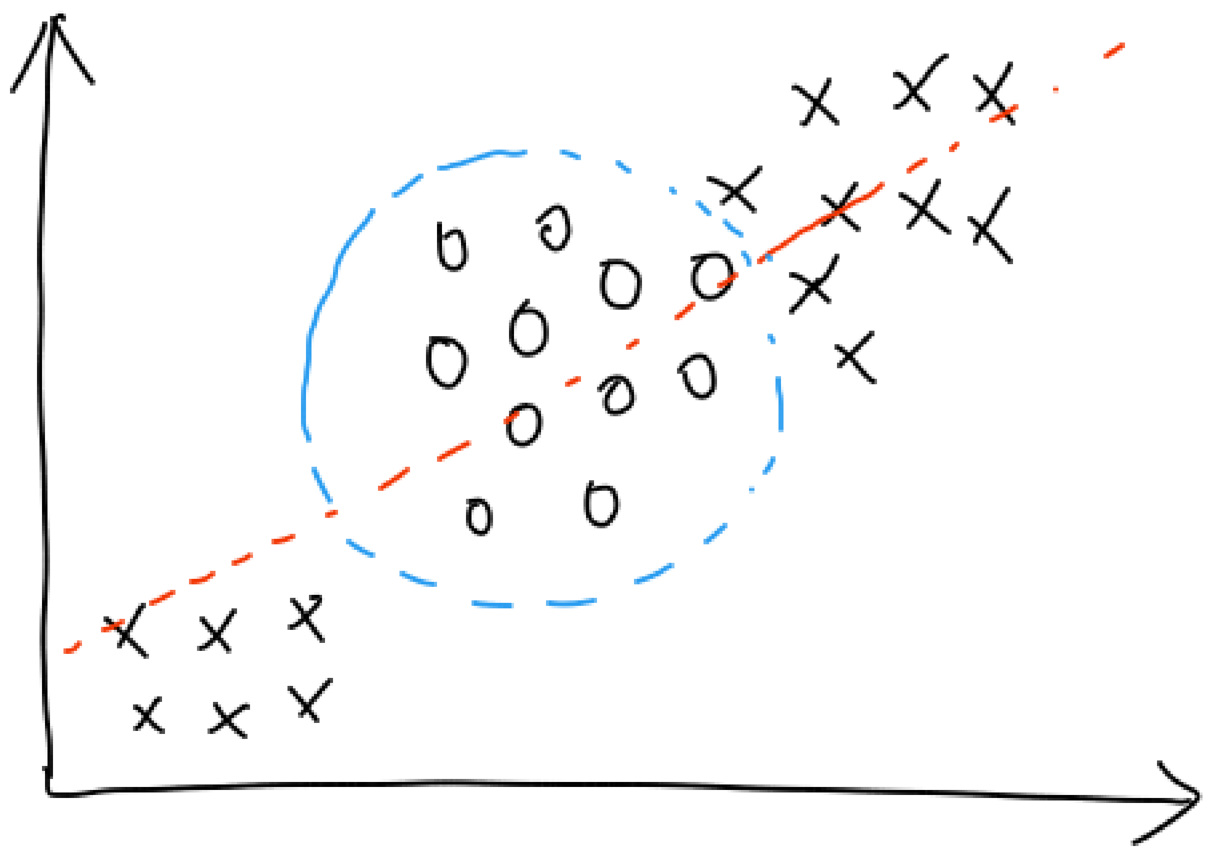
Lec 11: Neural Networks

Ex. 1



Vanilla LR

$$f(x, w) = \frac{1}{1+e^{-w^T x}}$$

$$f_B(x, w) = \frac{1}{1+e^{-w^T \phi(x)}}$$

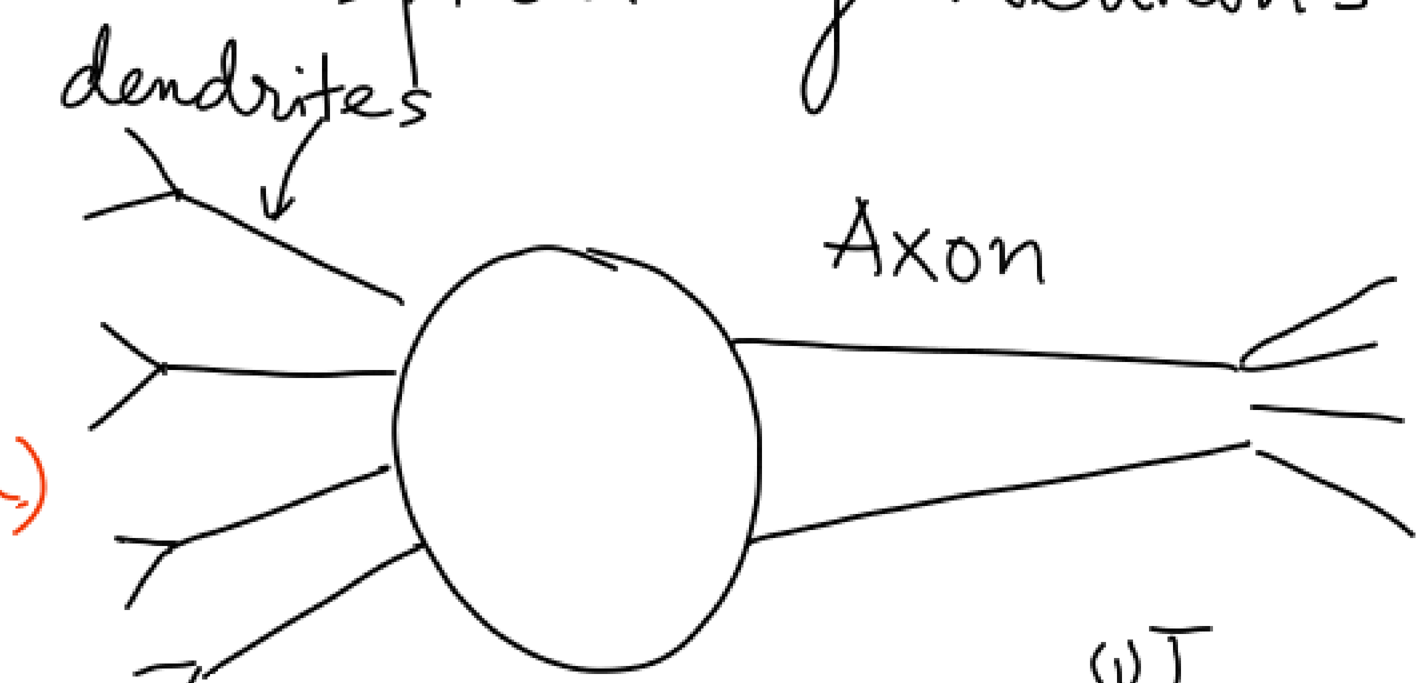
$$\phi(x) = \begin{pmatrix} 1 & x_1 & x_2 & x_1^2 & x_2^2 \end{pmatrix}^T$$

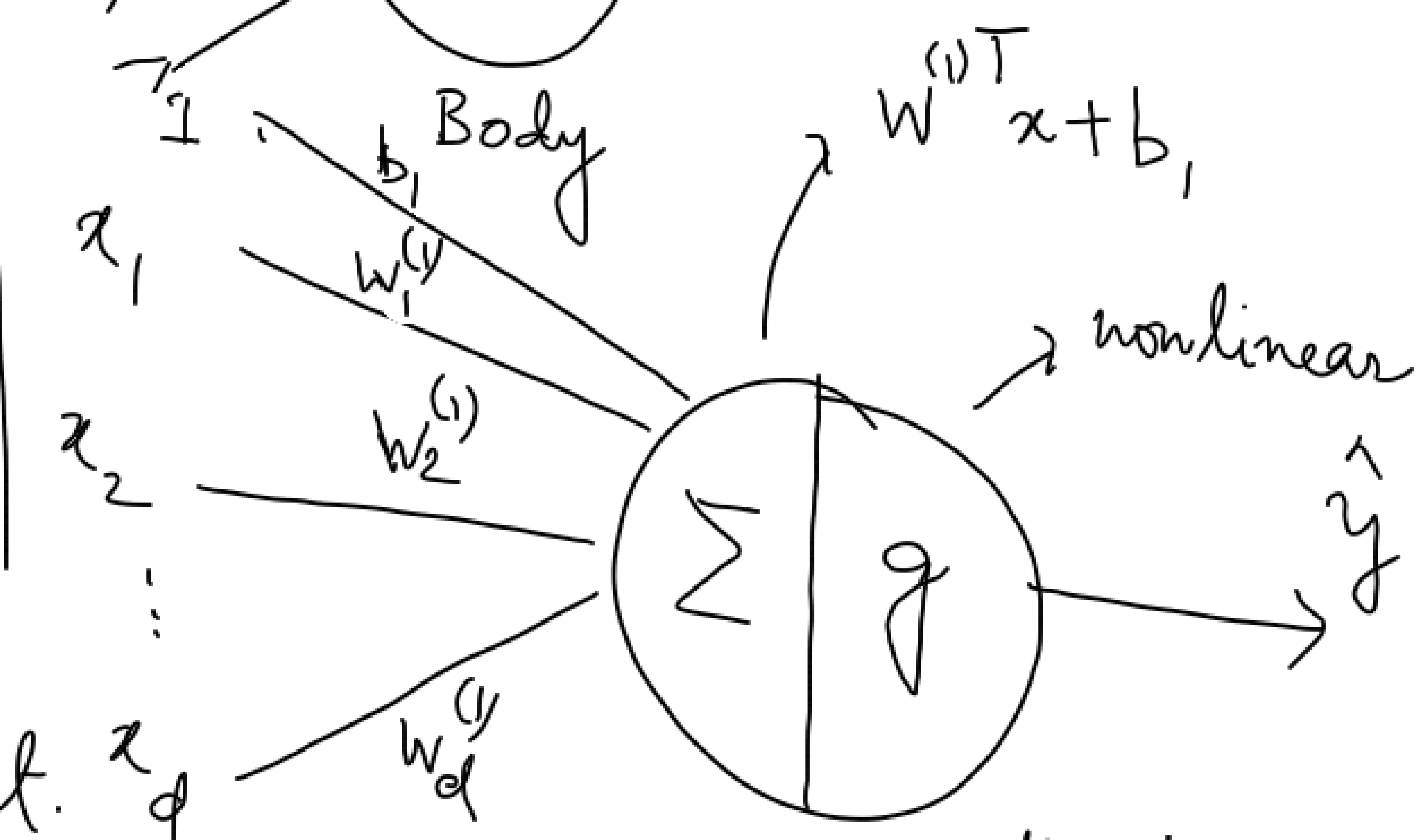
Philosophy of NN: Come up with some $\phi(x)$ without explicitly programming it.

Origin

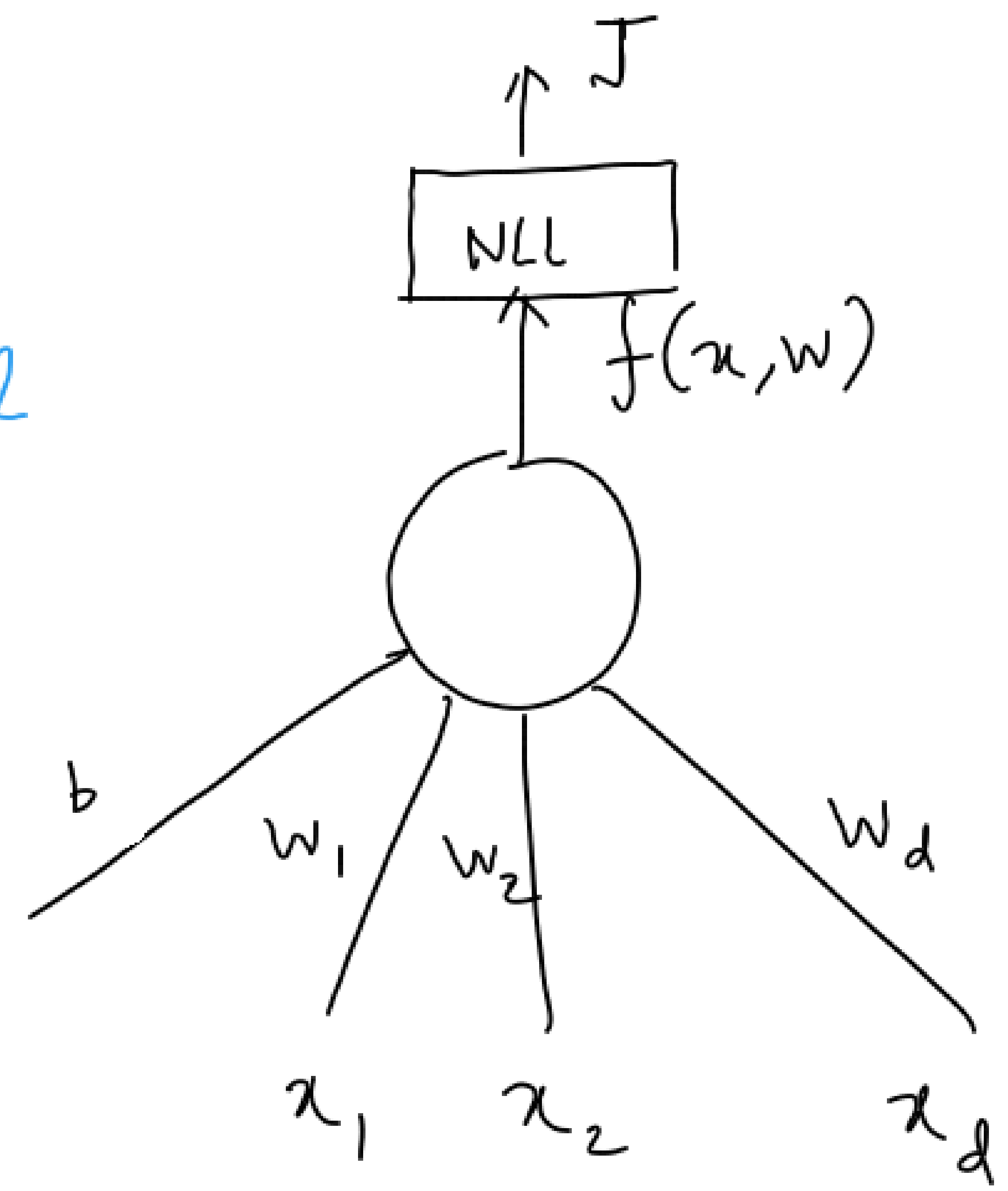
1943 → 1960 → 1980's

Inspired by Neurons





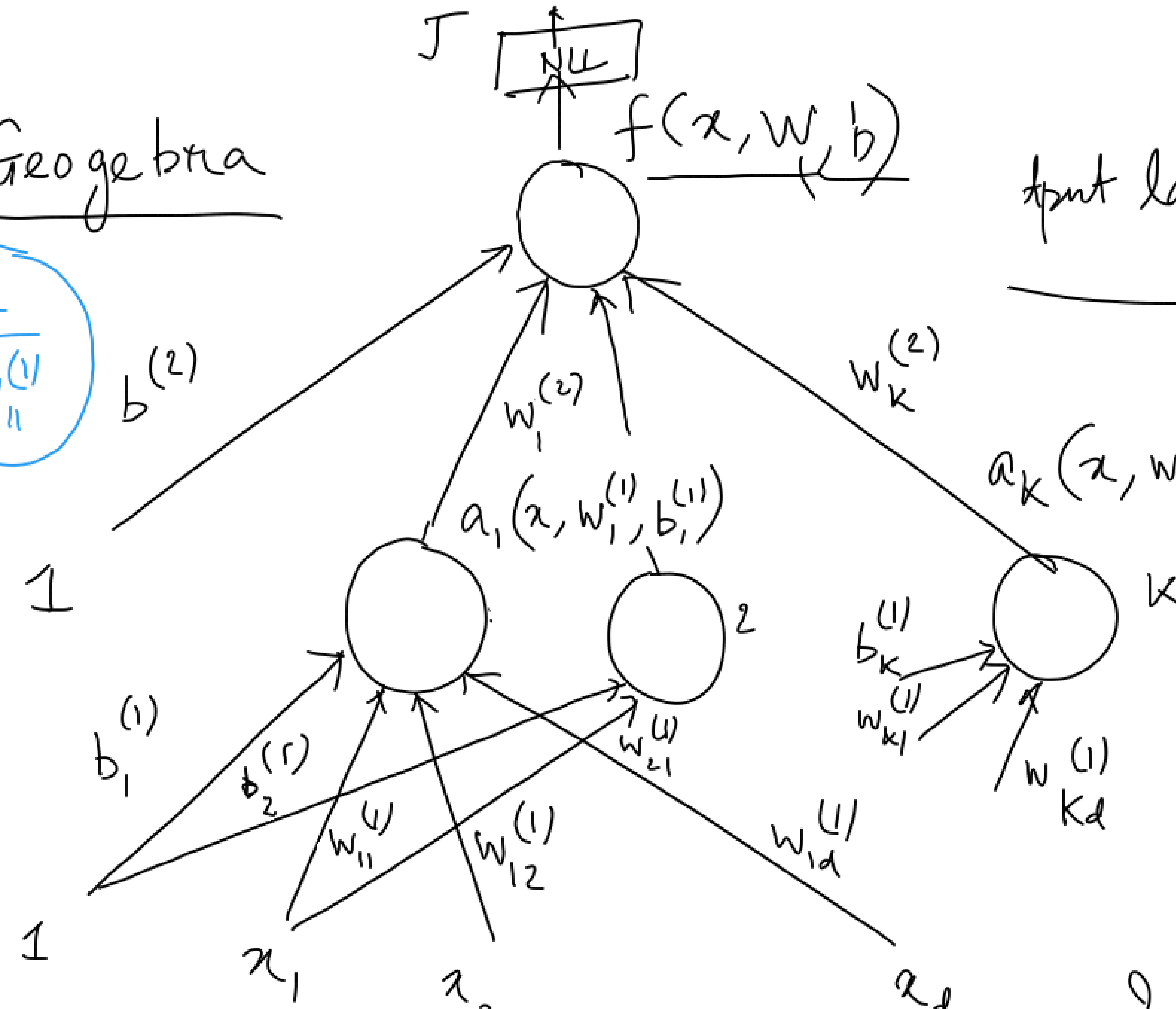
$\nabla_\theta \ell$



NN representation of Binary LR

Geogebra

$$\frac{\partial \ell}{\partial w_{11}^{(1)}}$$



Output layer

hidden layer

Input layer

$$f(x,w) = g\left(w^T x + b\right)$$

$$g = \sigma \ (\text{sigmoid})$$

$$NLL_i(w) = -\left[y_i \log f(x_i,w) + (1-y_i)\log\left(1-f(x_i,w)\right)\right]$$

cross-entropy

Goal: to learn the w's and b's of every layer to minimize the loss.

Popular activation functions

Sigmoid: $\sigma(x) = \dfrac{1}{1+e^{-x}}$

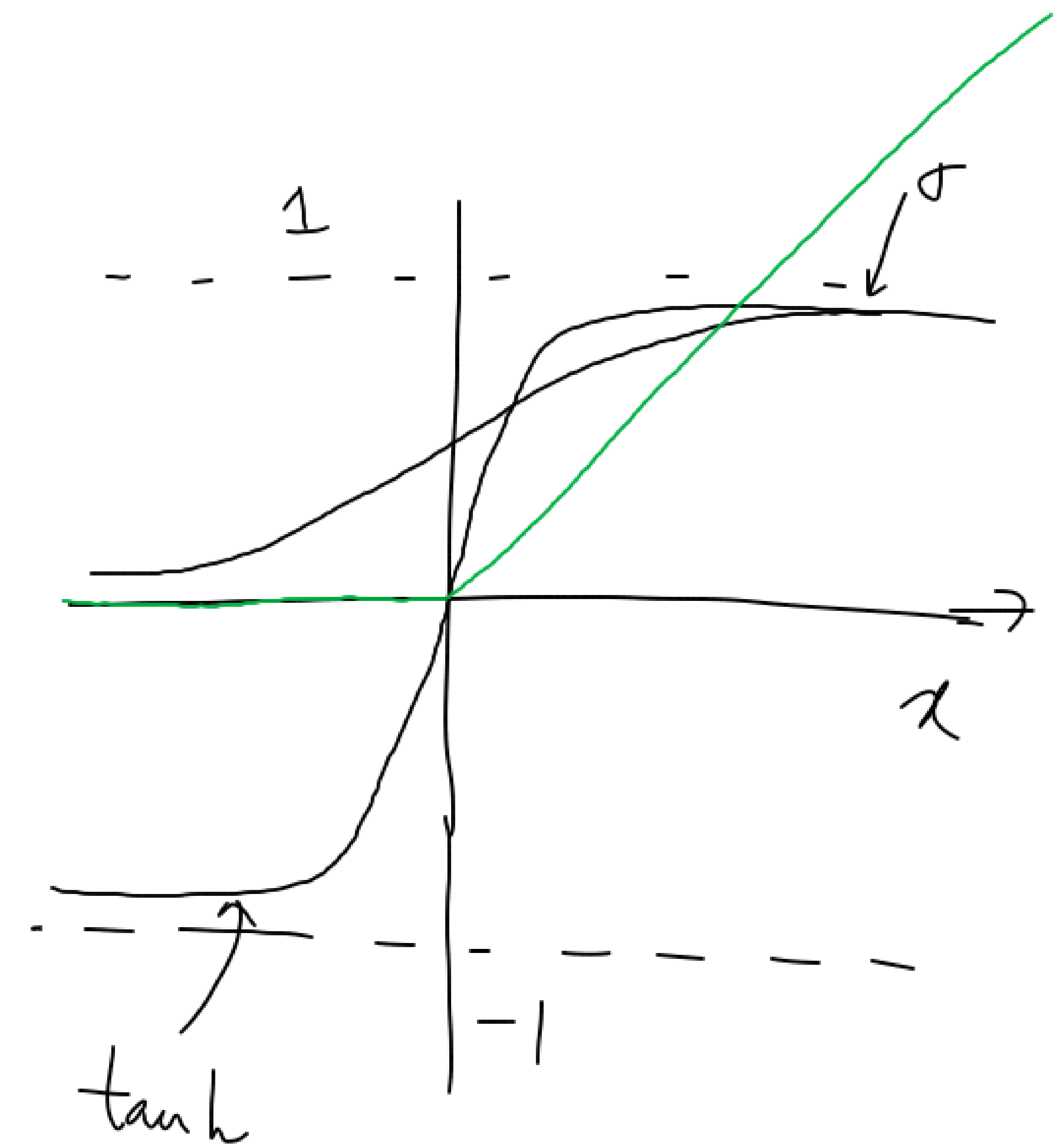
Hyperbolic tan: $\tanh(x) = \dfrac{e^{2x}-1}{e^{2x}+1}$

(Scaled sigmoid) $= 2\sigma(2x)-1$

Rectified Linear Units:

$\underline{ReLU(x)} = \max\{0, x\}$

gradient does not vanish for ReLU.



<u>Vanishing gradient problem</u>:

for deeper NNs, the gradient vanishes for large x.

$D = (x_i, y_i)_{i=1}^{m} \rightarrow$ given this dataset

find the values of W's and b's that minimize the loss function [Training NN]

Feed forward NN

e.g. $\underline{f(x_i, \theta)} = NN(x_i, \theta)$ $\rightarrow P(y_i = 1 | x_i, \theta)$

$\rightarrow$ Softmax (x, θ)

For a full blown NN

<u>Step 1</u>: Define a loss function $J(\overbrace{W, b}^{\theta})$, [e.g. cross entropy loss]

<u>Step 2</u>:

ell

$$J(\theta) = \sum_{i=1}^{n} \ell\left(NN(x_i, \theta), y_i\right)$$

$$\ell\left(NN(x_i, \theta), y_i\right) = -\left[y_i \log\left(NN(x_i, \theta)\right) + (1 - y_i)\log\left(1 - NN(x_i, \theta)\right)\right]$$

example.

Loss function is defined, SGD to optimize

NN training algorithm

- Inputs: $NN(x, \theta)$, training examples $x_1, \cdots, x_n$, labels $y_1, \cdots, y_n$ and a loss function ℓ

- Randomly initialize θ

mini-batch $B \subseteq \{1, \cdots, n\}$

- do until stopping criteria:

a batch B of examples.

Pick randomly $\boxed{\text{an example } (x_i, y_i)}$

Compute gradient of ℓ, $\boxed{\nabla_\theta \ell(x_i, y_i)}$

$\theta_{t+1} \leftarrow \theta_t - \eta \boxed{\nabla_\theta \ell}$

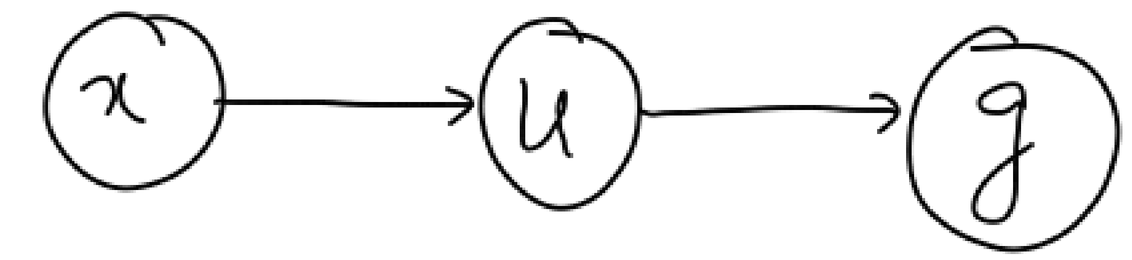
- Return θ_{t+1}

How to compute $\nabla_\theta l$ efficiently?

via Backpropagation. → uses the chain rule of differentiation in a clever way.

<u>Scalars</u>:

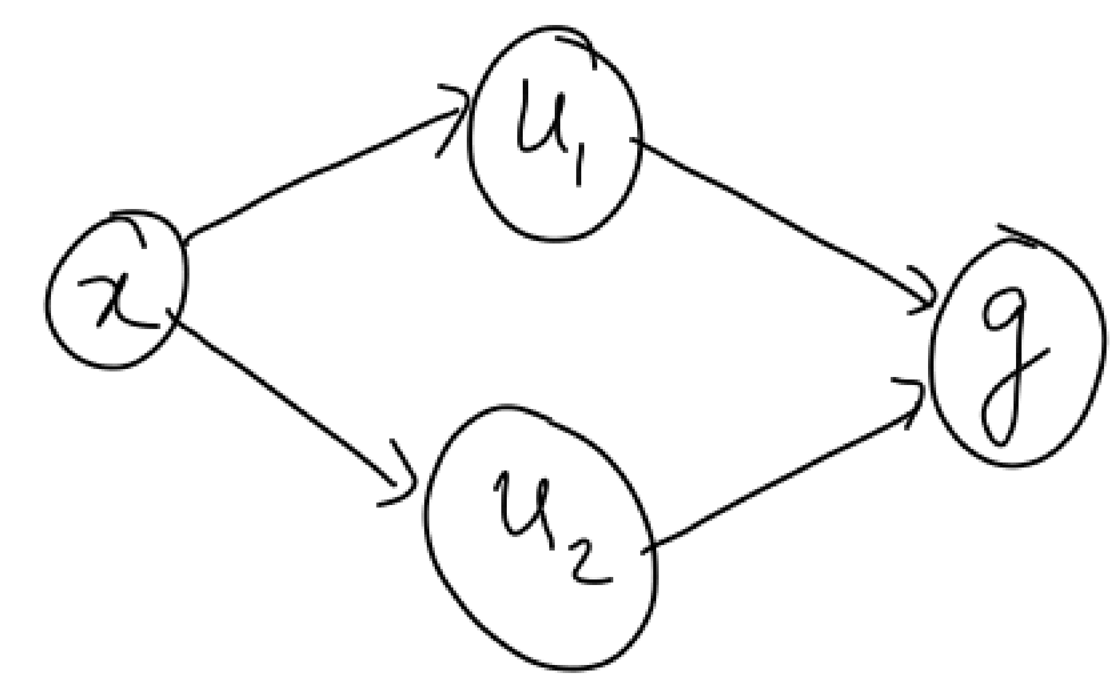
$$\frac{dg}{dx} = \frac{du}{dx} \cdot \frac{dg}{du}$$



<u>Vectors</u>:

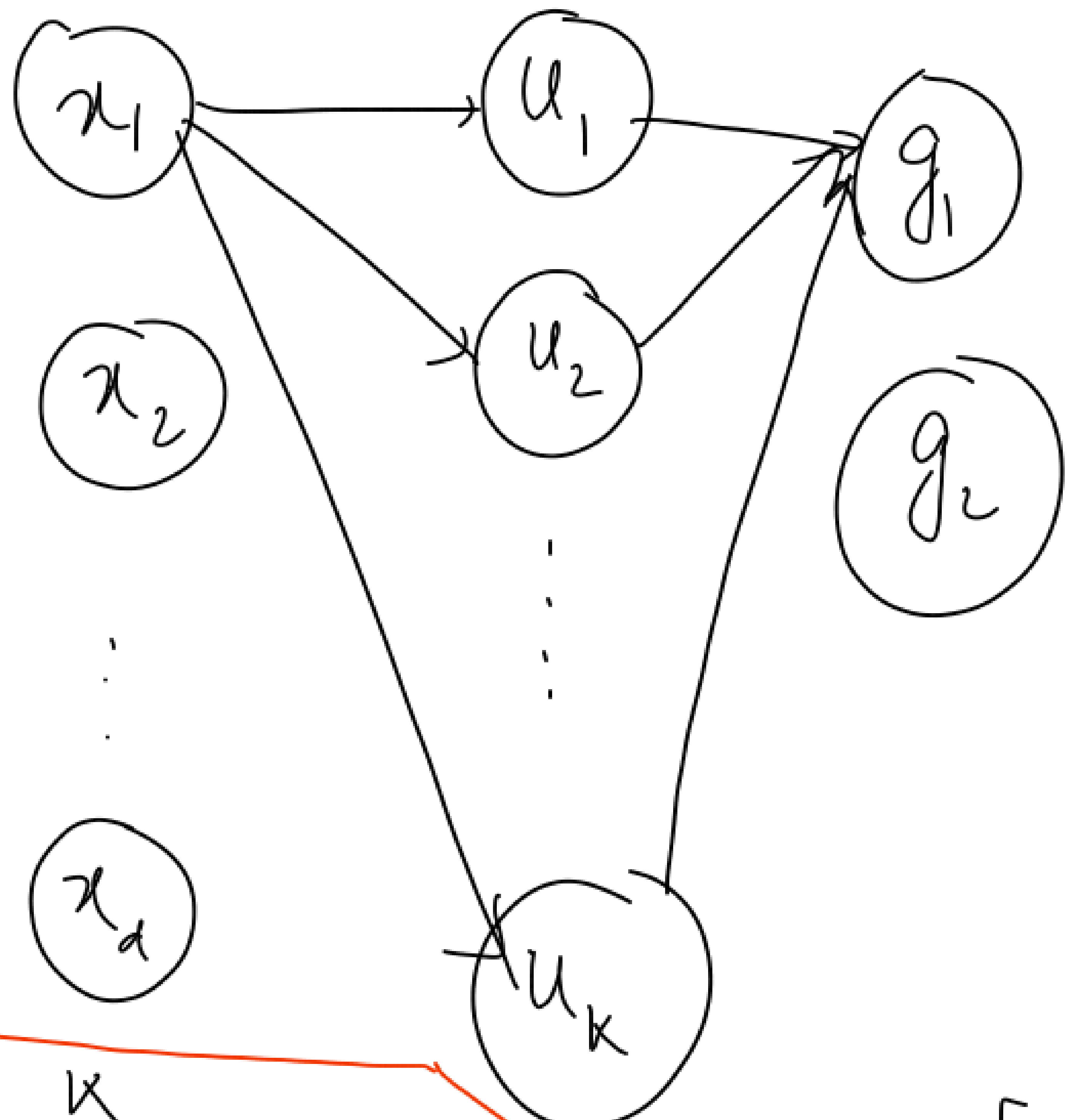
$$\frac{\partial g}{\partial x} = \frac{\partial u_1}{\partial x} \cdot \frac{\partial g}{\partial u_1} + \frac{\partial u_2}{\partial x} \cdot \frac{\partial g}{\partial u_2}$$

$$= \frac{\partial \underline{u}}{\partial x} \cdot \frac{\partial g}{\partial \underline{u}}$$



$$\begin{bmatrix} \frac{\partial u_1}{\partial x} & \frac{\partial u_2}{\partial x} \end{bmatrix}$$

$$\frac{\partial g}{\partial \underline{u}} = \begin{bmatrix} \frac{\partial g}{\partial u_1} & \frac{\partial g}{\partial u_2} \end{bmatrix}$$



$$\frac{\partial g}{\partial \underline{x}} = \frac{\partial \underline{u}}{\partial \underline{x}} \cdot \frac{\partial \underline{g}}{\partial \underline{u}}$$

$$\frac{\partial \underline{u}}{\partial \underline{x}} = \begin{bmatrix} \frac{\partial u_1}{\partial x_1} & \frac{\partial u_2}{\partial x_1} & \cdots & \frac{\partial u_k}{\partial x_1} \\ \frac{\partial u_1}{\partial x_2} & \frac{\partial u_2}{\partial x_2} & & \vdots \\ \vdots & \vdots & & \vdots \end{bmatrix}$$

$$\boxed{\frac{\partial g_1}{\partial x_1} = \sum_{j=1}^{K} \frac{\partial u_j}{\partial x_1} \cdot \frac{\partial g_1}{\partial u_j}}$$

Backpropagation.

$$\frac{\partial g}{\partial \underline{u}} = \begin{bmatrix} \frac{\partial g_1}{\partial u_1} & \cdots & \frac{\partial g_m}{\partial u_1} \\ \frac{\partial g_1}{\partial u_2} & & \vdots \\ \vdots & & \frac{\partial g_m}{\partial u_k} \\ \frac{\partial g_1}{\partial u_k} & & \end{bmatrix}$$