NEURAL NETWORKS - REGRESSION

let $\{x_i, y_i\}_{i=1,\dots,M}$ a set of input-output pairs

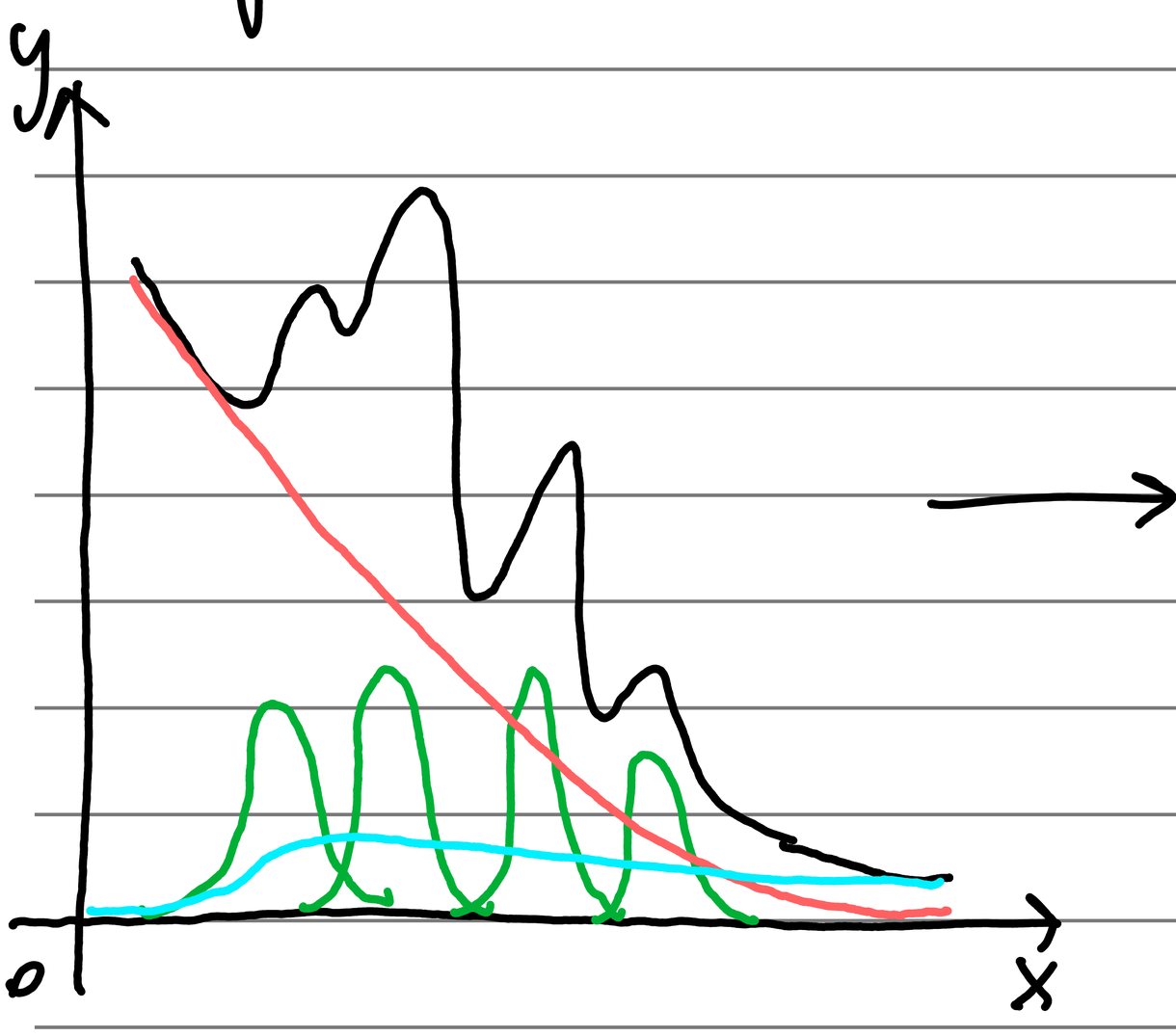
we want to find a model $\hat{y}$ that predicts new outputs given new inputs

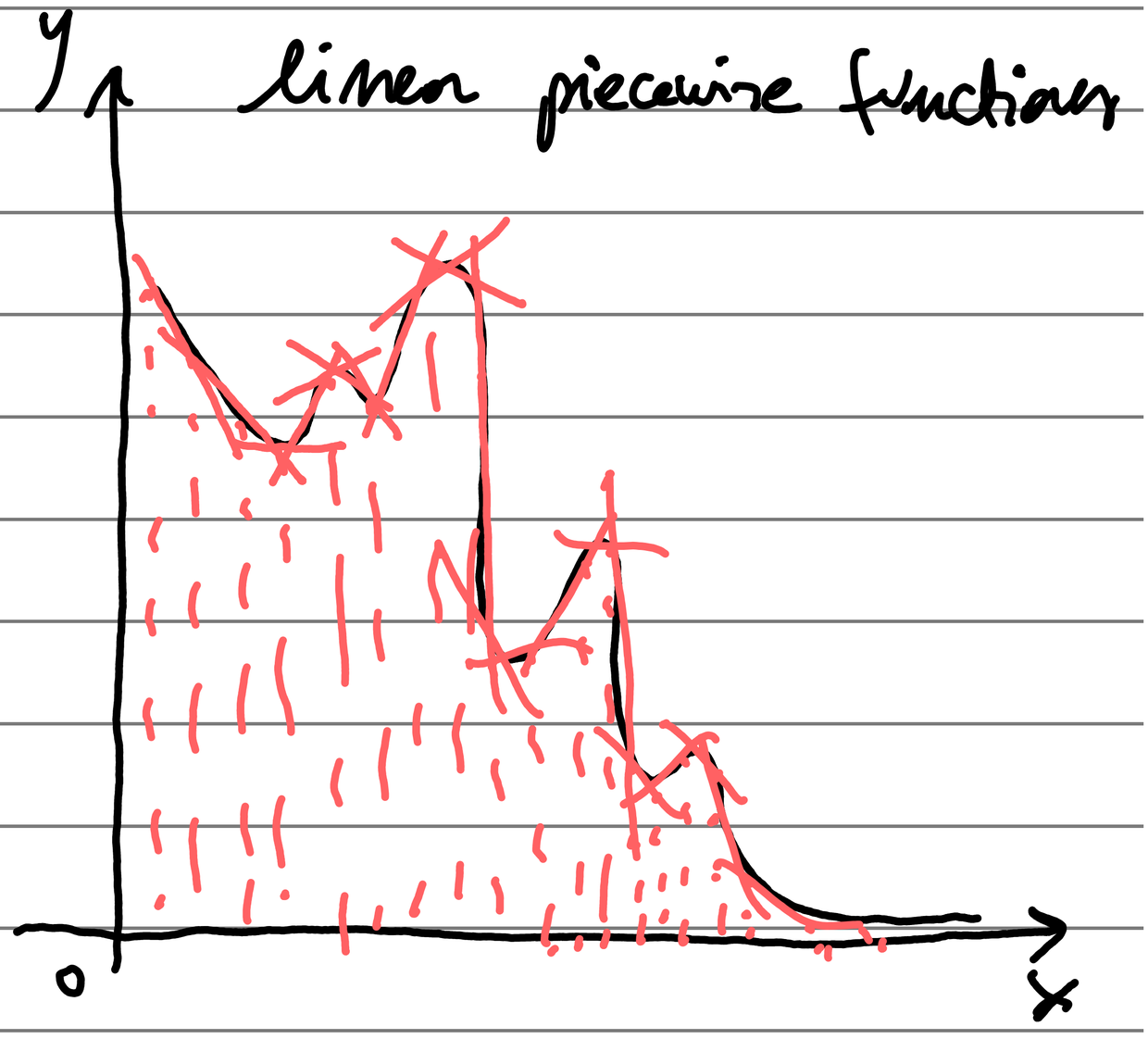
We have seen linear regression

$$f(x, w) = \sum_{i=1}^{M} w_j \phi_j(\underline{x})$$

i.e. models that are linear in the parameters

It might be necessary to consider a wider range of models





sacrifice interpretability to performance

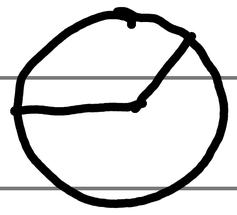
$$f(x, \underline{w}, \underline{\theta}) = \sum_{i=0}^{n} w_i \, h_i(\underline{x}, \underline{\theta})$$

where $h_i = a\left[\theta_{0i} + \theta_{1i}\, x\right]$

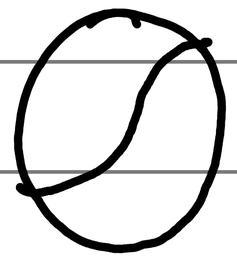
- a: non linear function (activation)
- $\theta_{0i} + \theta_{1i}\, x$: linear function

activation:

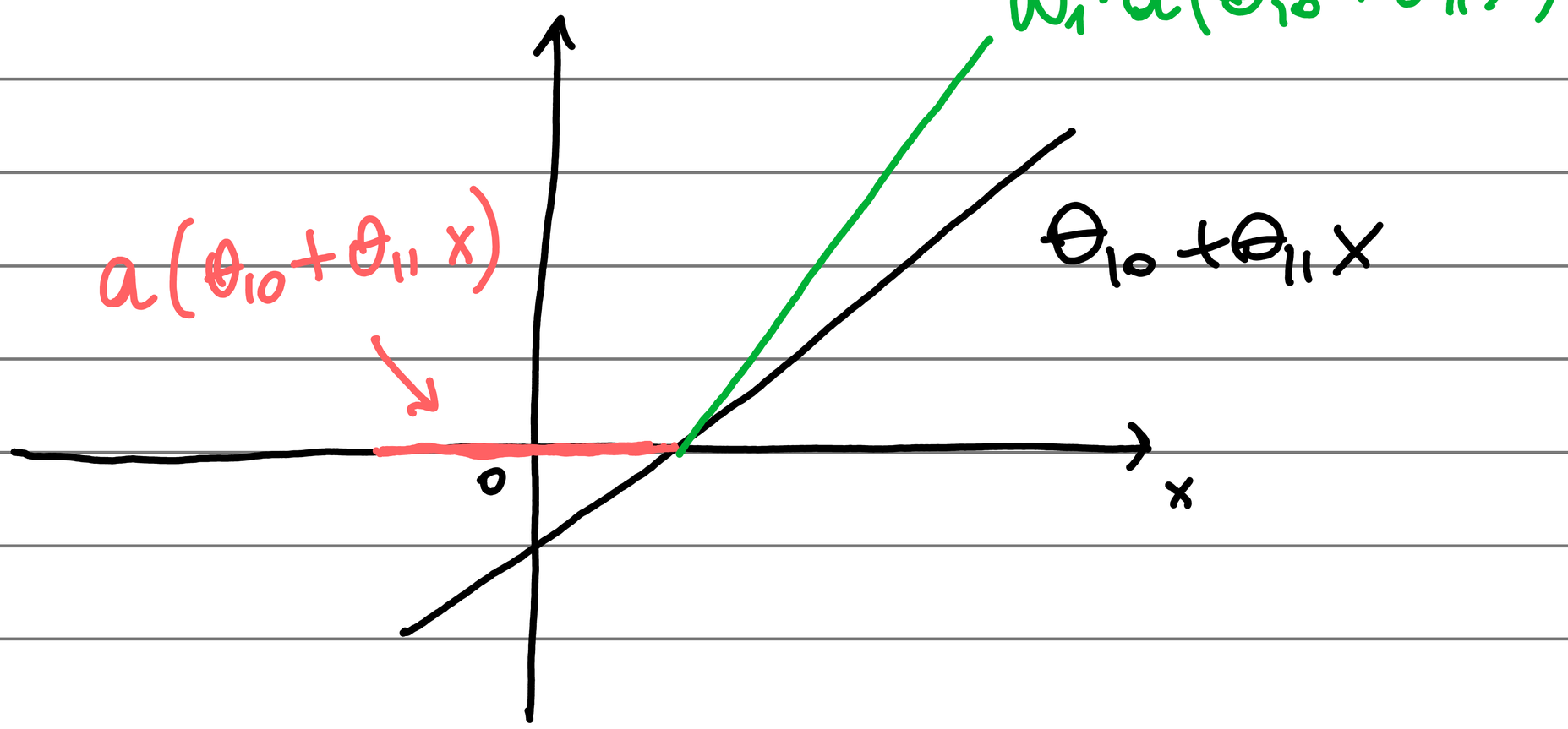
ReLU $\rightarrow$ $a(z) = \begin{cases} 0 & z < 0 \\ z & z \geq 0 \end{cases}$



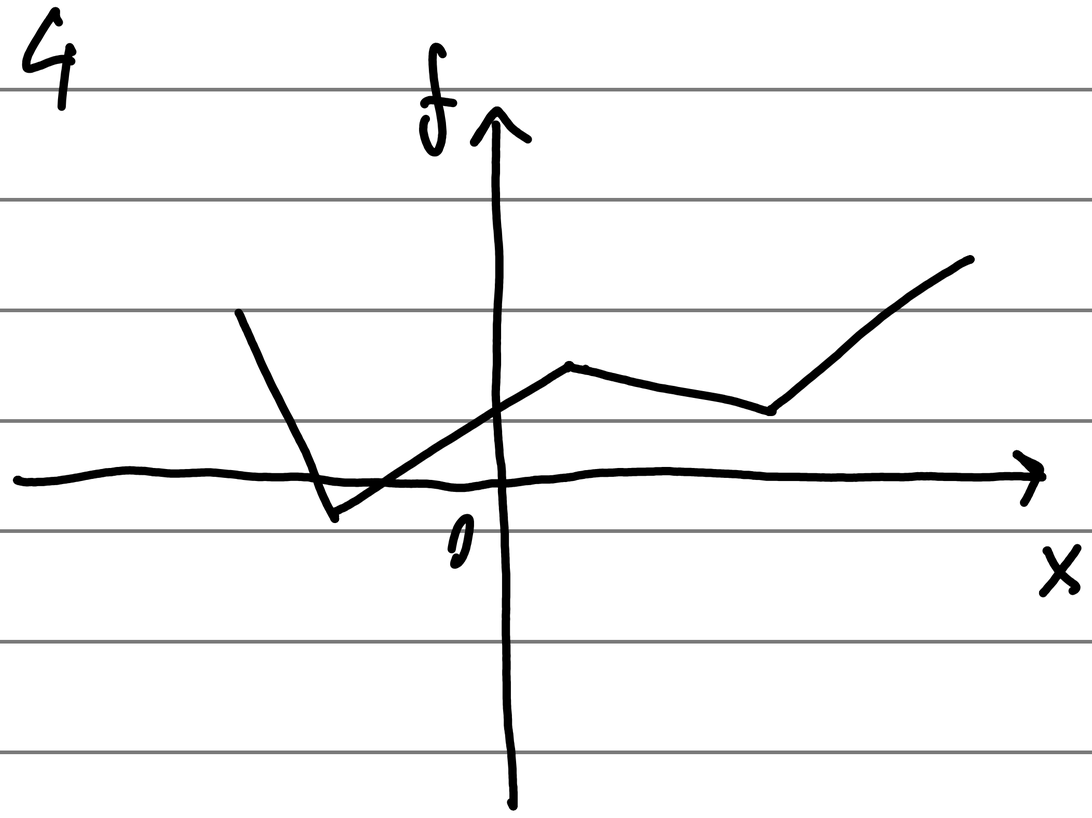
SIGMOID $\rightarrow$ $a(z) = \dfrac{1}{1 + \exp(-z)}$



e.g.

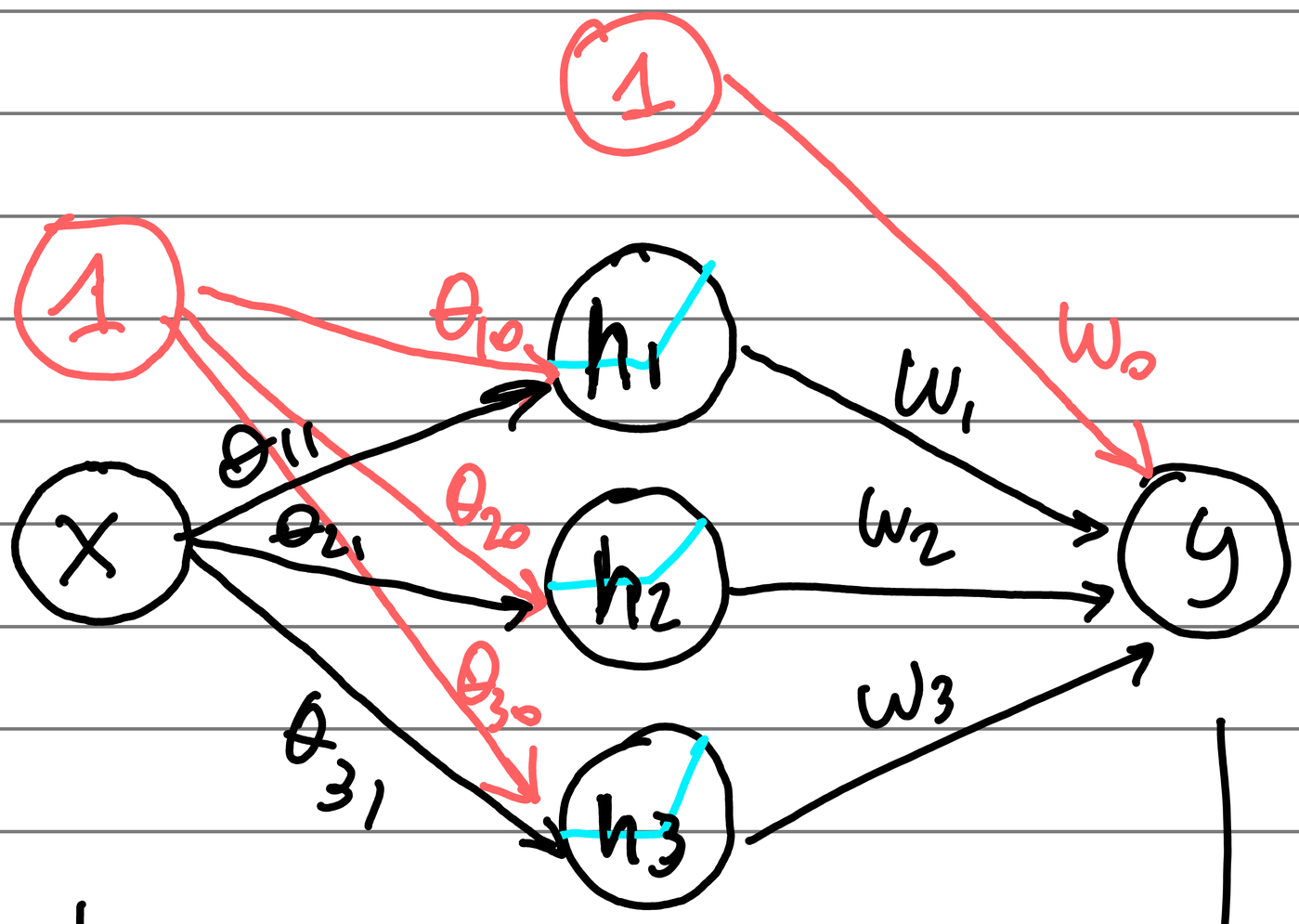


for $M = 4$



$$w_0 + w_1 h_1 + w_2 h_2 + w_3 h_3$$

the corresponding neural network can be represented as



UNIVERSAL APPROXIMATION THEOREM

Suppose we have D hidden units

$$\hat{y} = \sum_{i=0}^{D} w_i h_i \qquad (h_0 = 1)$$

D is the network capacity

TH: with enough capacity, a "shallow" network can describe any continuous 1D function defined on a compact subset of the real line to arbitrary precision

Note: I can have $x \in \mathbb{R}^d$, so each d has now D parameters and the space is divided into 2^d regions

→ dimension of the network grows very quickly with complexity of input

HOW TO BUILD A NEURAL NETWORK (FOR REGRESSION)

The setup of a regression analysis using a neural network follows the same basic steps of the linear regression approach (with a few key differences)

- TRAINING AND TEST DATASETS (AND VALIDATION)
- DEFINE MODEL (CAPACITY, DEPTH, ...)
- LOSS FUNCTION (e.g. E(w) error function in LR)
- OPTIMIZATION METHOD AND INITIACIZATION
- PERFORMANCE ESTIMATION
- REGULARIZATION

OPTIMIZATION METHOD AND INIZIALIZATION

Given a training set $\{x_i, y_i\}_{i=1,\dots,n}$

and a model $\hat{y}(\underline{x}, \underline{w})$ I want to find

the best set of parameters $\underline{w}$ so that $\hat{y}$

is a good model. The loss function $L(\underline{w})$

gives a measure of how good is a given set

of parameters.

In linear regression, we could analytically

minimize with respect to $\underline{w}$ and with

MCMC we just explored parameter space.

Both are not feasible in neural network since

1) they are non-linear models → analytical solution doesn't exist

2) they have a large parameter space → impossible to explore

the process of minimization of $L(\underline{w})$ is ITERATIVE

let's consider $\{w_i^0\}_{i=1,\dots,M}$ initial parameters

STEP 1: Compute derivative of the loss

$$\frac{\partial L}{\partial w^0} = \begin{bmatrix} \partial L/\partial w_1^0 \\ \vdots \\ \partial L/\partial w_M^0 \end{bmatrix}$$

STEP 2: update parameters

$$w^1 = w^0 - \alpha \frac{\partial L}{\partial w^0}$$

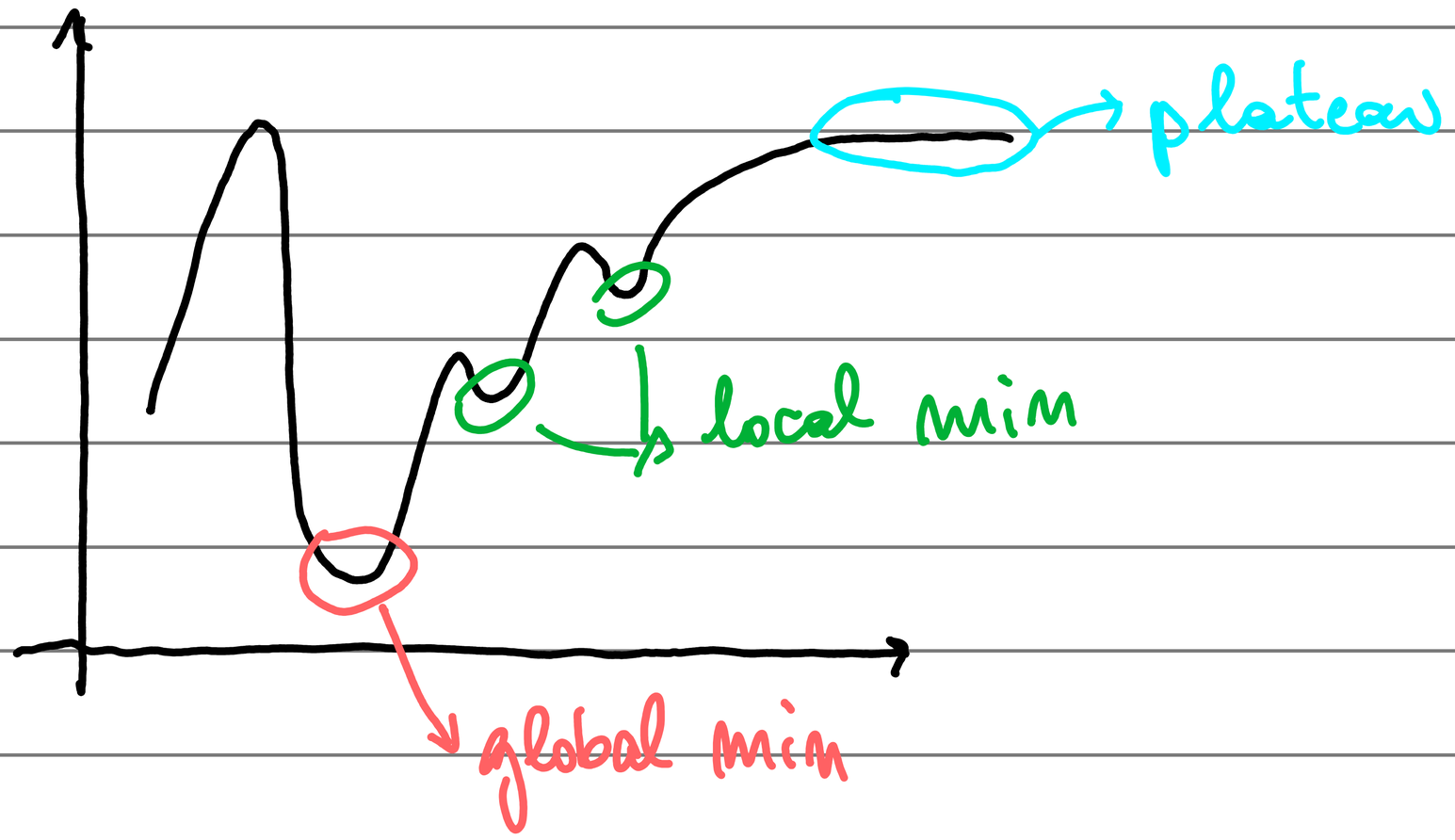
$\hookrightarrow \alpha > 0$ magnitude of change (learning rate)

$$w^2 = w^1 - \alpha \frac{\partial L}{\partial w^1}$$

$$w^{n+1} = w^n - \alpha \frac{\partial L}{\partial w^n}$$

Theoretically we should find a $\{w_i^n\}_{i=1,\dots,M}$ such that $\frac{\partial L}{\partial w^n} = 0$ but in practice this is numerically unfeasible $\rightarrow$ we monitor and terminate when $\frac{\partial L}{\partial w} \ll 0$

PROBLEM: In complex (and high-dimensional) cases $L(w)$ has local minima and saddle points



→ hard to converge to global minimum

- searching entire parameter space is unfeasible
- trying different starting points also unfeasible

STOCHASTIC GRADIENT DESCENT

IDEA: compute iterations for a subset of the training set (BATCH)

$$W^{t+1} = W^t - \alpha \sum_{i \in B_t} \frac{\partial L}{\partial W^t} \qquad B_t \subset \text{TRAINING SET}$$

once all training data has been seen → EPOCH

BINARY CLASSIFICATION

$\{x_i, y_i\}_{i=1,\dots,n}$ where now $y_i = \{0, 1\}$

how do I build a loss function?

$\longrightarrow$ how is the output distributed

BERNOULLI DISTRIBUTION

$$Pr(y|\lambda) = \begin{cases} 1-\lambda & y=0 \\ \lambda & y=1 \end{cases}$$

$$= (1-\lambda)^{1-y} \lambda^{y}$$

now we want to apply this to our model

$\hat{y} = f[x, \underline{w}, \underline{\phi}]$ $\longrightarrow$ this function needs to give values between $[0, 1]$

LOGISTIC SIGMOID $sig(z) = \dfrac{1}{1+e^{-z}}$ $z \longrightarrow \lambda$

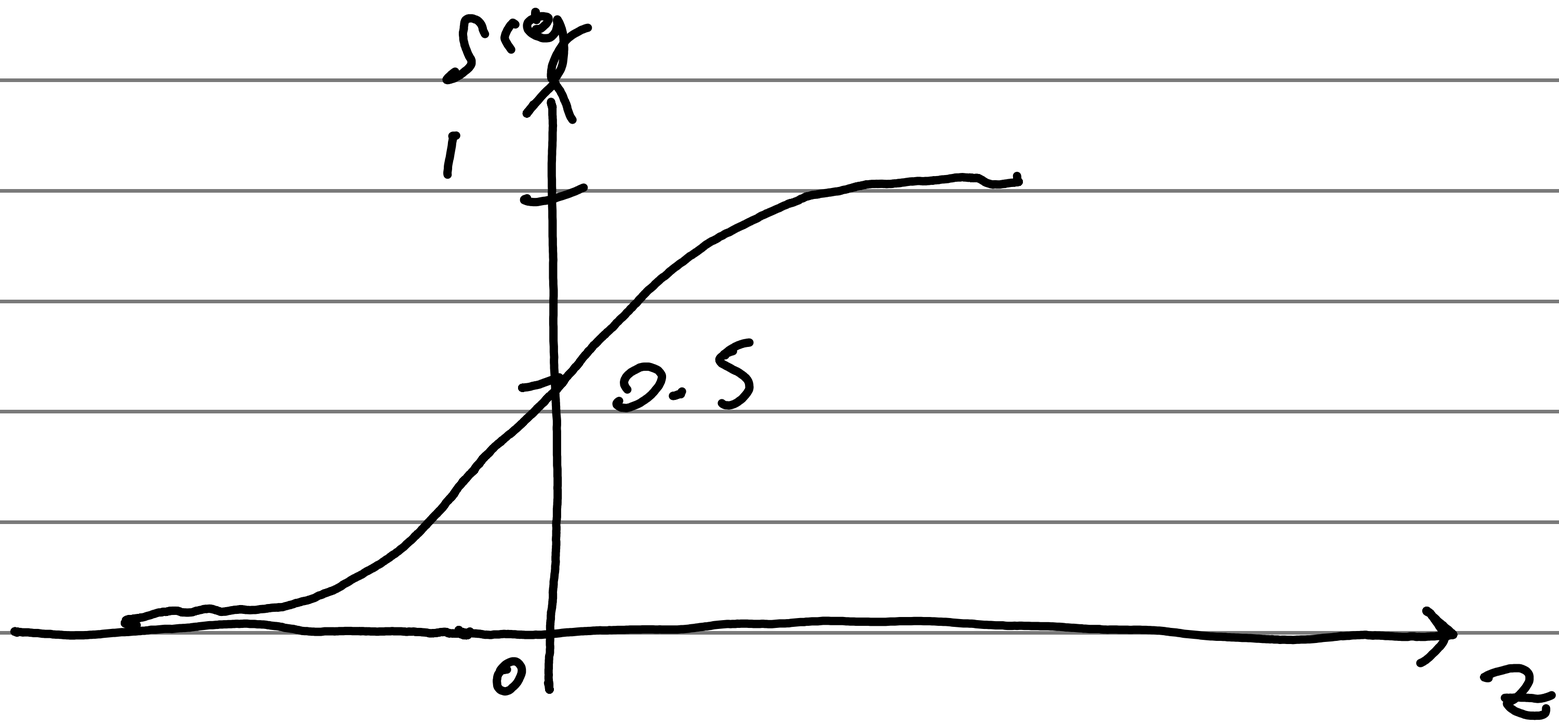
$$P(y|\hat{y}) = (1-sig[\hat{y}])^{1-y} (sig[\hat{y}])^{y}$$

LOSS FUNCTION : NEGATIVE LOG-LIKELIHOOD of P

$$\mathcal{L}[\underline{w}] = \sum_{i=1}^{n} -(1-y_i) \ln[1-sig[\hat{y}]] - y_i \ln[sig[\hat{y}]]$$

(BINARY CROSS-ENTROPY LOSS)

now $sig(z) = \frac{1}{1+e^{-z}}$



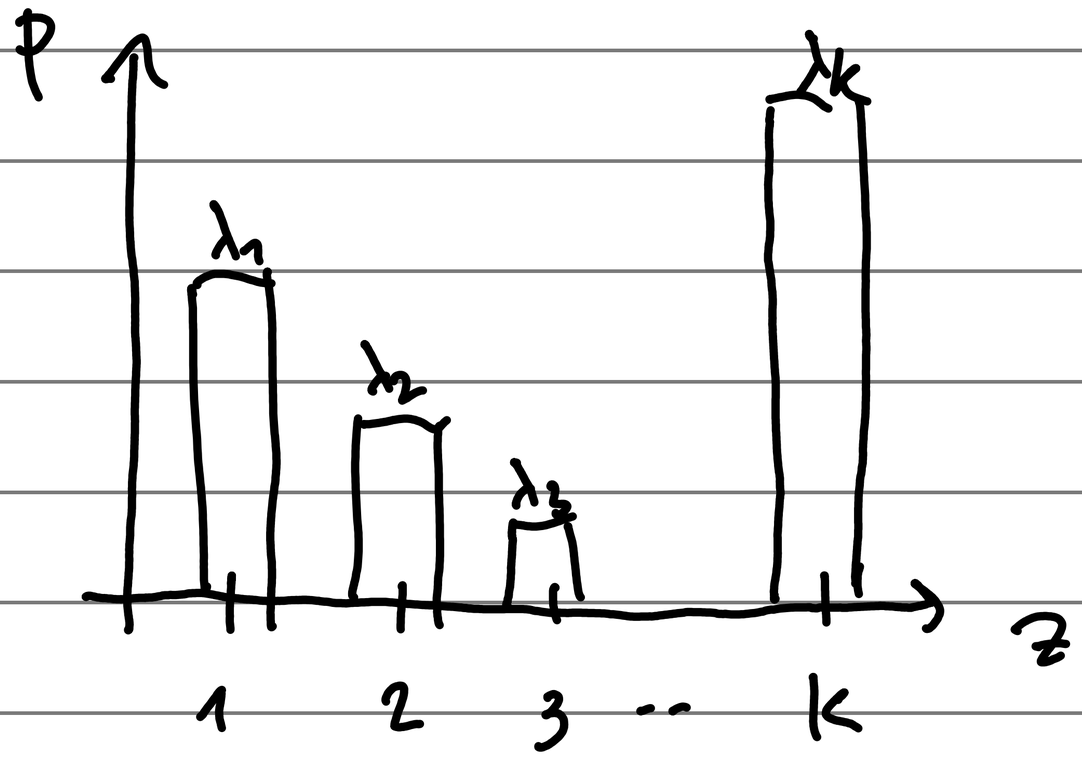
y is $\{0,1\}$ but $sig(z) \in [0,1]$ so

we just decide that $\begin{cases} sig(z) > 0.5 \rightarrow y=1 \\ sig(z) \leq 0.5 \rightarrow y=0 \end{cases}$

MULTICLASS CLASSIFICATION

$$\{x_i, y_i\}_{i=1,\dots,M} \qquad y_i \in \{1, \dots, K\}$$

CATEGORICAL DISTRIBUTION



$$P(y=k) = \lambda_k$$

the distribution to convert the prediction is now

$$\text{SOFTMAX}_k(z) = \frac{e^{z_k}}{\sum_{k'=1}^{K} e^{z_{k'}}}$$

$$P(y=k \mid \hat{y}) = \text{SOFTMAK}_k[\hat{y}] \qquad \in [0,1]$$

for each class, we have again a function between $[0,1]$

the corresponding loss is

$$\mathcal{L}[w] = -\sum_{i=1}^{n} \ln\left[\text{SOFTMAX}_{y_i}[\hat{y}_i]\right]$$

$$= -\sum_{i=1}^{n}\left(\hat{y}^i_{y_i} - \ln\left[\sum_{k'=1}^{k} e^{\hat{y}^{k'}_i}\right]\right)$$

Note that we need $\hat{y}$ to have k outputs for each $\underline{X}$

$$\hat{y}^i_j = f_i(X_j, \underline{w})$$