USC Viterbi
School of Engineering

DECISION THEORY AND LOGISTIC REGRESSION

EE 541 – UNIT 4

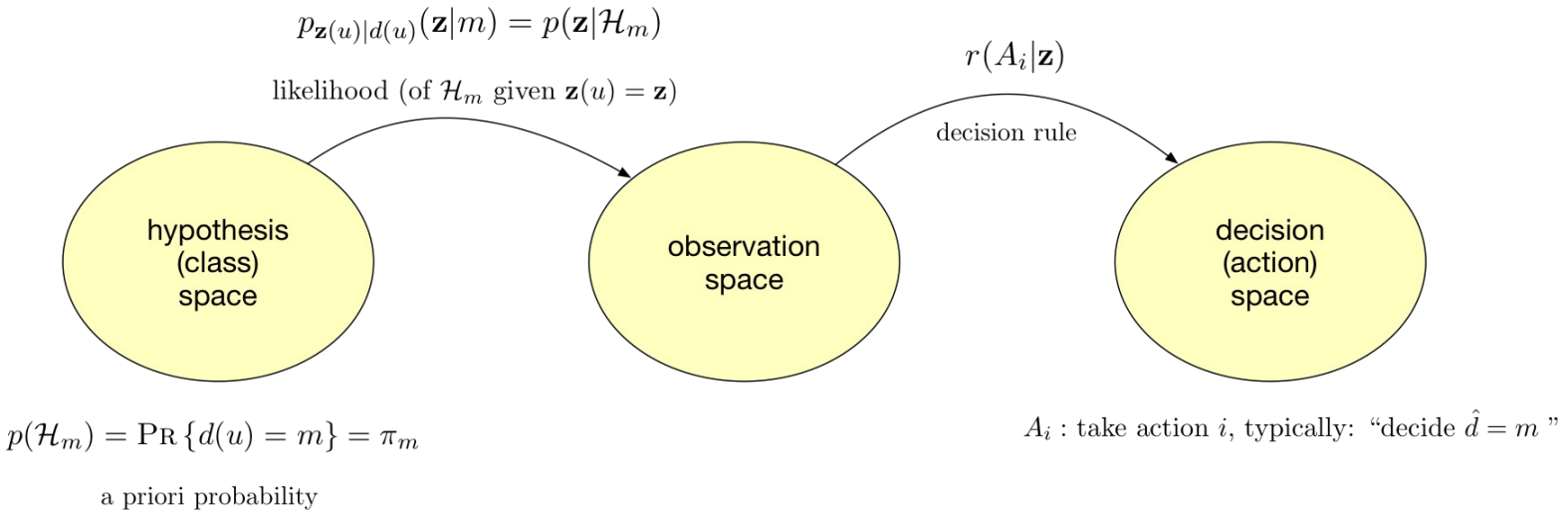
DR. BRANDON FRANZKE

Spring 2023

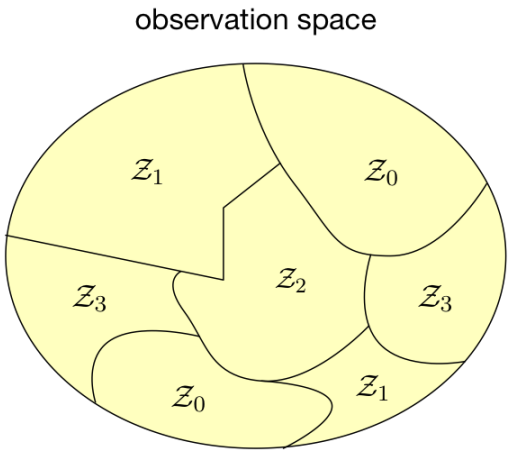
DECISION/DETECTION THEORY

- **Bayesian Decision Theory**
  - Bayes decision rule
  - MAP rule - minimum error probability rule
- Maximum Likelihood
  - Likelihood, Negative-Log-Likelihood, Likelihood ratios
- Neyman-Pearson test and the ROC
  - Detection and False Alarm trade off

DECISION THEORY FRAMEWORK (STATISTICAL)



Goal: design a good decision rule using the statistical model



typically try to implement the decision rule as a partitioning of the sample space

$\mathcal{Z}_m = \text{decision region } m = \{\mathbf{z} \in \mathcal{Z} : r(A_m|\mathbf{z}) = 1\}$

DECISION THEORY FRAMEWORK (STATISTICAL)

Bayes risk for decision rule r

$$Risk(r) = \int_{\mathcal{Z}} p(\boldsymbol{z}) \left[\sum_j r(A_j|\boldsymbol{z}) C(A_j|\boldsymbol{z}) \right] d\boldsymbol{z}$$

Cost for action m given observation $\boldsymbol{z}$

$$C(A_j|\boldsymbol{z}) = \sum_i C(\mathcal{H}_i, A_j) p(\mathcal{H}_i|\boldsymbol{z})$$

$$C_{ij} = C(\mathcal{H}_i, A_j) = \text{Cost of deciding } \mathcal{H}_j \text{ when } \mathcal{H}_i \text{ true}$$

Bayes decision rule (minimizes Bayes risk)

$$r_{Bayes}(A_m|\boldsymbol{z}) = \begin{cases} 1 & m = \arg\min_j C(A_j|\boldsymbol{z}) \\ 0 & else \end{cases}$$

a posteriori factoring

$$p(\mathcal{H}_m|\boldsymbol{z}) = \frac{p(\boldsymbol{z}|\mathcal{H}_m)\pi_m}{p(\boldsymbol{z})}$$

$$\cong p(z|\mathcal{H}_m)\pi_m$$

a posteriori probability (APP)

equivalent for making decisions – normalizing $p(\boldsymbol{z})$ is constant given H_m, i.e., does not depend on H_m

MAXIMUM A POSTERIORI PROBABILITY (MAP) RULE

MAP is special case of Bayesian Decision Rule

$$C = \begin{bmatrix} 0 & 1 & 1 & \dots & 1 \\ 1 & 0 & 1 & \dots & 1 \\ 1 & 1 & 0 & \dots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \dots & 0 \end{bmatrix}$$

$$\begin{aligned} C(A_j|\boldsymbol{z}) &= \sum_i C_{ij} p(\mathcal{H}_i|\boldsymbol{z}) \\ &= \sum_i (1 - \delta[i-j])\, p(\mathcal{H}_i|\boldsymbol{z}) \\ &= \sum_{i \neq j} p(\mathcal{H}_i|\boldsymbol{z}) \\ &= 1 - p(\mathcal{H}_j|\boldsymbol{z}) \end{aligned}$$

For these costs, the Bayes risk is the probability of decision error

MAP rule

$$\max_m p(\mathcal{H}_m|\boldsymbol{z}) \quad \Leftrightarrow \quad \max_m p(z|\mathcal{H}_M)\pi_m$$

MAP rule minimizes probability of decision error over finite number of hypotheses

ASIDE: HARD AND SOFT DECISIONS

Consider MMSE Estimation of a digital/discrete random variable

Hard decision: $\hat{d} = 3$

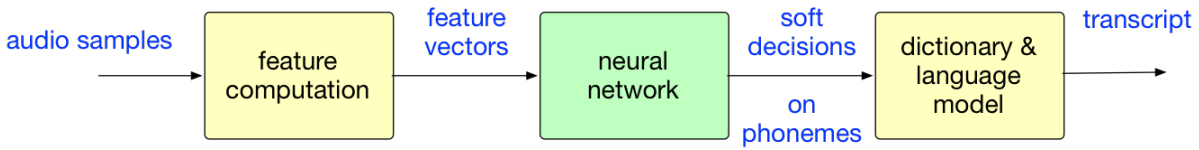
Soft Decision A: $p(\tilde{d} = 0) = 0.11 \quad p(\tilde{d} = 1) = 0.39 \quad p(\tilde{d} = 2) = 0.1 \quad p(\tilde{d} = 3) = 0.4$

Soft Decision B: $p(\tilde{d} = 0) = 0.01 \quad p(\tilde{d} = 1) = 0.01 \quad p(\tilde{d} = 2) = 0.01 \quad p(\tilde{d} = 3) = 0.97$

Both soft decisions A and B are consistent with the same hard decision, but B corresponds to much higher confidence

APPs are (typically) the *ideal* soft decisions

Soft decisions are often desired when the output of the classifier feeds additional processing — *e.g.*, Automatic Speech Recognition (ASR):



BINARY MAP RULE

$$M = 2$$

$$P(\mathcal{E}) = P(\mathcal{E}|\mathcal{H}_0)\pi_0 + P(\mathcal{E}|\mathcal{H}_1)\pi_1$$

$$= \int_{\mathcal{Z}_1} f(\boldsymbol{z}|\mathcal{H}_0)\pi_0 d\boldsymbol{z} + \int_{\mathcal{Z}_0} f(\boldsymbol{z}|\mathcal{H}_1)\pi_1 d\boldsymbol{z}$$

$$f(\boldsymbol{z}|\mathcal{H}_1)\pi_1 \underset{\mathcal{H}_1}{\overset{\mathcal{H}_0}{\lesseqgtr}} f(\boldsymbol{z}|\mathcal{H}_0)\pi_0$$

$$\Lambda(\boldsymbol{z}) = \frac{f(\boldsymbol{z}|\mathcal{H}_1)}{f(\boldsymbol{z}|\mathcal{H}_0)} \underset{\mathcal{H}_1}{\overset{\mathcal{H}_0}{\lesseqgtr}} \frac{\pi_0}{\pi_1} = T$$

Likelihood Ratio Test

likelihood ratio

when $M > 2$, still work with $p(\boldsymbol{z}|H_m)$ / $p(\boldsymbol{z}|H_0)$

OTHER RULES (MAP SPECIAL CASES)

Maximum Likelihood (ML): $\max_m f(\boldsymbol{z}|\mathcal{H}_m)$

Minimum Distance: $\min_m d(\boldsymbol{z}, \boldsymbol{s}_m)$

Min. Euclidean (squared) Distance: $\min_m \|\boldsymbol{z} - \boldsymbol{s}_m\|^2$

MAP reduces to ML when a priori probabilities are uniform

$\boldsymbol{M = 2}$

Maximum Likelihood (ML): $\underset{\mathcal{H}_1}{\overset{\mathcal{H}_0}{f(\boldsymbol{z}|\mathcal{H}_1) \lessgtr f(\boldsymbol{z}|\mathcal{H}_0)}}$

Minimum Distance: $\underset{\mathcal{H}_1}{\overset{\mathcal{H}_0}{d(\boldsymbol{z}, \boldsymbol{s}_0) \lessgtr f(\boldsymbol{z}, \boldsymbol{s}_1)}}$

Min. Euclidean (squared) Distance: $\underset{\mathcal{H}_1}{\overset{\mathcal{H}_0}{\|\boldsymbol{z} - \boldsymbol{s}_0\|^2 \lessgtr \|\boldsymbol{z} - \boldsymbol{s}_1\|^2}}$

OTHER DECISION CRITERION (NON-BAYES)

minimax rule: the Bayes full for the worst case *a priori* probabilities

Neyman-Pearson rule:
maximize detection probability for a given false alarm probability

$$P_D = P(decide\ \mathcal{H}_1|\mathcal{H}_1)$$ Detection Probability

$$P_{FA} = P(decide\ \mathcal{H}_1|\mathcal{H}_0)$$ False Alarm Probability

NP rule example:
Given $P_{fa} < 0.1$, maximize P_d

Can always maximize detection probability by always deciding H_1, but this will have high false alarm probability.

Note: NP and minimax rules do not need knowledge of priors

LOGISTIC REGRESSION

LOGISTIC REGRESSION MOTIVATION

Note that a linear classifier has hard decision:

$$\hat{y} = \text{sign}(\boldsymbol{w}^T \boldsymbol{x}) \qquad \hat{y} \in \{-1, +1\}$$

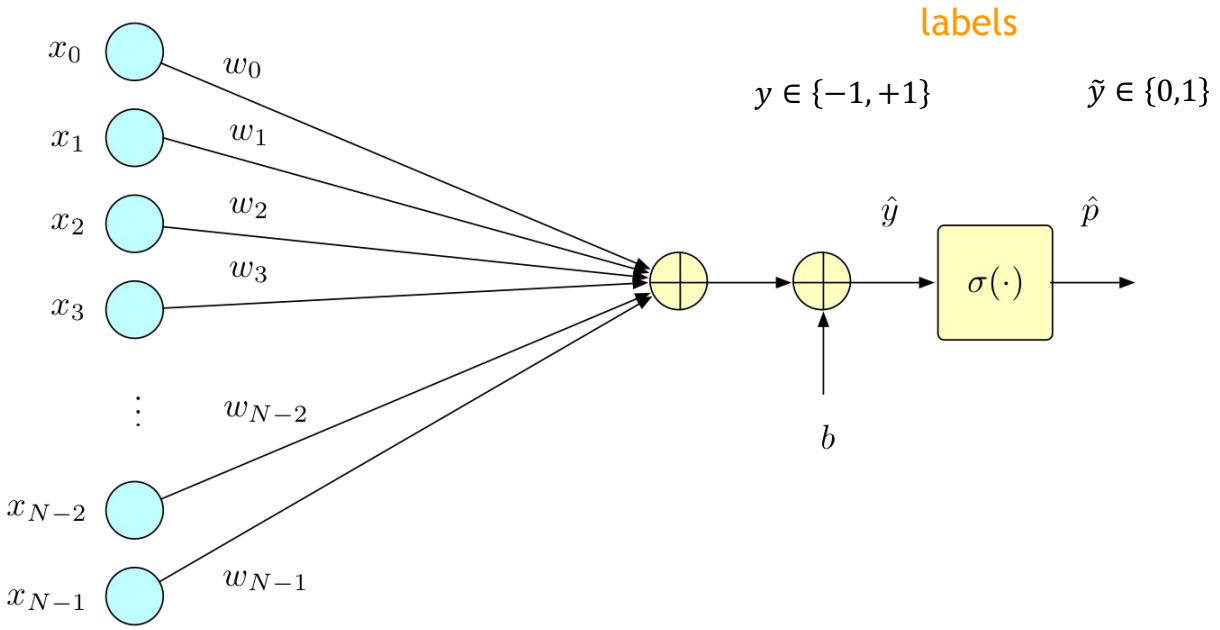
with corresponding soft decision:

$$\hat{y} = \boldsymbol{w}^T \boldsymbol{x}$$

Note that this is a real number – the magnitude is the "confidence" and the hard decision is the sign

How can we convert this to a soft decision that is a probability?

LOGISTIC REGRESSION



Two problems to address:

1. What is a good "sigma" function to map from reals targeting +/- 1 to a probability of a 1?

2. What is a good loss function between the binary labels $\{0,1\}$ and the regressor output $\hat{p} \sim P(1)$?

RECALL: ML INTERPRETATION OF LLSE REGRESSION

model for ML estimation of $\boldsymbol{w}$:

$$y = \boldsymbol{w}^T \mathbf{x} + v(t) \qquad p(v) = \mathcal{N}(v; 0, \sigma_v^2)$$

if we adopt the convention that:

$$y = +1 \quad \Leftrightarrow \quad \tilde{y} = 1$$

$$y = -1 \quad \Leftrightarrow \quad \tilde{y} = 0$$

Likelihood ratio for y (binary classification):

$$\frac{p(y = +1|\boldsymbol{x}, \boldsymbol{w})}{p(y = -1|\boldsymbol{x}, \boldsymbol{w})} = \frac{\mathcal{N}(+1; \boldsymbol{w}^T \boldsymbol{x}, \sigma_v^2)}{\mathcal{N}(-1; \boldsymbol{w}^T \boldsymbol{x}, \sigma_v^2)}$$

$$= \exp\left[\frac{2}{\sigma_v^2} \boldsymbol{w}^T \boldsymbol{x}\right]$$

Log-likelihood ratio:

$$L = \frac{2}{\sigma_v^2} \boldsymbol{w}^T \boldsymbol{x}$$

Takeaway: $\boldsymbol{w}$ dotted with $\boldsymbol{x}$ can be viewed as a log-likelihood ratio or log ratio of (scaled) probabilities

LOGISTIC REGRESSION MOTIVATION

First, suppose we have the log ratio of two probability-like values *(maybe not normalized)*

$$\begin{aligned} L &= \ln\left(\frac{p_1}{p_0}\right) \\ &= \ln p_1 - \ln p_0 \\ &= \ell_1 - \ell_0 \end{aligned}$$

$$p_0 = \frac{e^{\ell_0}}{e^{\ell_0} + e^{\ell_1}} = \frac{1}{1 + e^{+L}}$$

$$p_1 = \frac{e^{\ell_1}}{e^{\ell_0} + e^{\ell_1}} = \frac{1}{1 + e^{-L}}$$

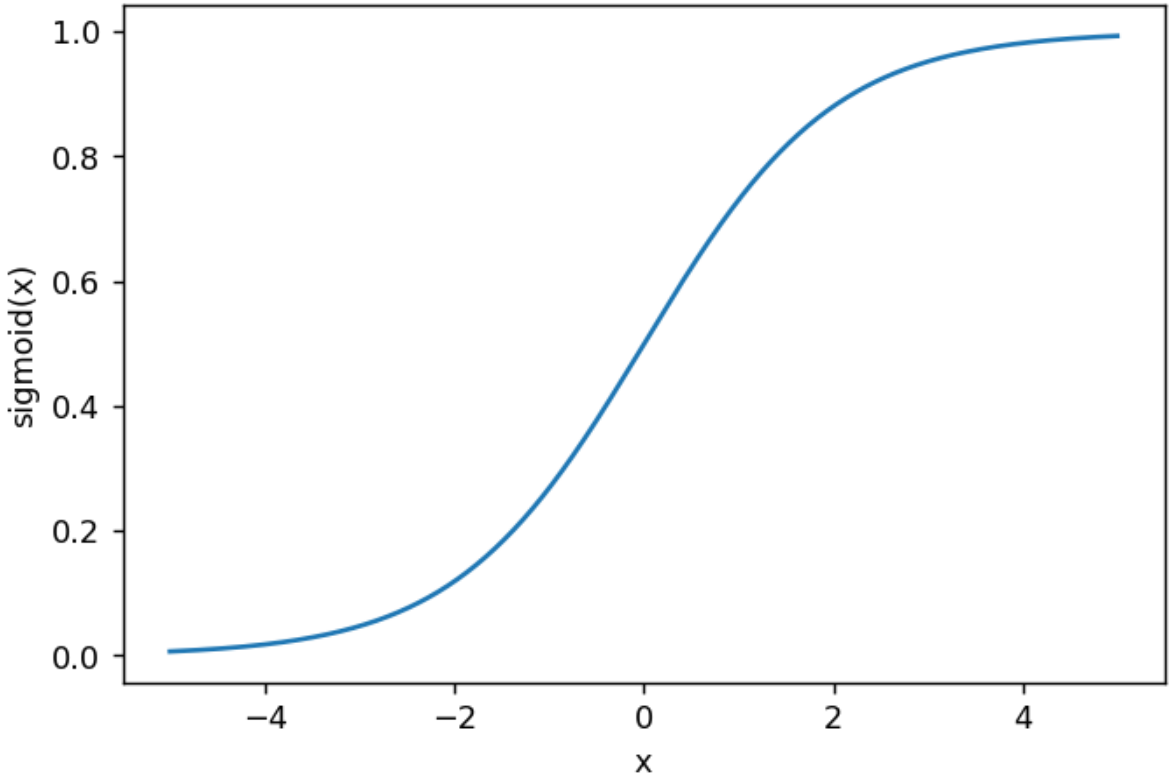
logistic function maps a log-likelihood ratio to the "probability of a +1"

$$p_1 = 1 - p_0 = \sigma(L) = \frac{e^L}{1 + e^L} = \frac{1}{1 + e^{-L}}$$

σ is called the logistic function or sigmoid function – sigmoid is *overloaded*

LOGISTIC REGRESSION MOTIVATION

maps a log-likelihood ratio to a probability (p_1 or $p_{numerator}$)



Note: $\sigma(0) = 0.5$, *i.e.*, $P(A) = 0.5 = P(A^C)$

LOGISTIC FUNCTION USEFUL PROPERTIES

$$\dot{\sigma}(s) = \frac{d}{ds}\left[\frac{e^s}{1+e^s}\right]$$

$$= \frac{e^s}{1+e^s} - \frac{(e^s)^2}{(1+e^s)^2}$$

$$= \left[\frac{e^s}{1+e^s}\right]\left[\frac{1}{1+e^s}\right]$$

$$= \sigma(s)\big(1-\sigma(s)\big)$$

$$\sigma(s) = \frac{1}{2}\left[1+\tanh\left(\frac{s}{2}\right)\right]$$

LOGISTIC REGRESSION MOTIVATION

This motivates a model for a binary variable y:

$$\tilde{y}(u) \sim Bernoulli(p)$$

$$p = p_1 = \sigma(\boldsymbol{w}^T \boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{w}^T \boldsymbol{x})}$$

$$(1 - p) = p_0 = \frac{1}{1 + \exp(+\boldsymbol{w}^T \boldsymbol{x})}$$

Model the values of the binary targets

$$\tilde{y}_n \sim Bernoulli\big(\sigma(\boldsymbol{w}^T \boldsymbol{x})\big) \qquad \text{i.i.d.}$$

LOGISTIC REGRESSION

Take ML approach to determining $\boldsymbol{w}$ for this model:

$$p(y|X;w) = \prod_{n=1}^{N} p_n^{\tilde{y}_n}[1-p_n]^{(1-\tilde{y}_n)} \qquad p_n = \sigma(\boldsymbol{w}^T \boldsymbol{x}_n)$$

$$= \prod_{n=1}^{N} p_n^{\mathbb{I}[y_n=+1]}[1-p_n]^{\mathbb{I}[y_n=-1]}$$

$$p_n^{\tilde{y}_n}[1-p_n]^{(1-\tilde{y}_n)} = \begin{cases} p_n & \tilde{y}_n = 1 \ \ (y_n = +1) \\ 1-p_n & \tilde{y}_n = 0 \ \ (y_n = -1) \end{cases}$$

The negative log-likelihood is….

Example:

Labels ($\tilde{y}$): 1, 0, 1

Output ($\sigma(w^T x)$): 0.9, 0.1, 0.2

$p(y|x;w)$: 0.9 * 0.9 * 0.2

LOGISTIC REGRESSION

The negative log-likelihood is:

$$NLL(\boldsymbol{w}) = -\left(\sum_{n=1}^{N} \tilde{y}_n \log(p_n) + (1-\tilde{y}_n)\log(1-p_n)\right)$$

$$= -\left(\sum_{n=1}^{N} \tilde{y}_n \log\big(\sigma(\boldsymbol{w}^T\boldsymbol{x})\big) + (1-\tilde{y}_n)\log\big(1-\sigma(\boldsymbol{w}^T\boldsymbol{x})\big)\right)$$

$$= \sum_{n=1}^{N} \log(1+\exp[-y_n\boldsymbol{w}^T\boldsymbol{x}])$$

$$p_n = \sigma(\boldsymbol{w}^T\boldsymbol{x}_n)$$

This is called the binary cross-entropy loss. We will see that this can be viewed as a distance between two distributions

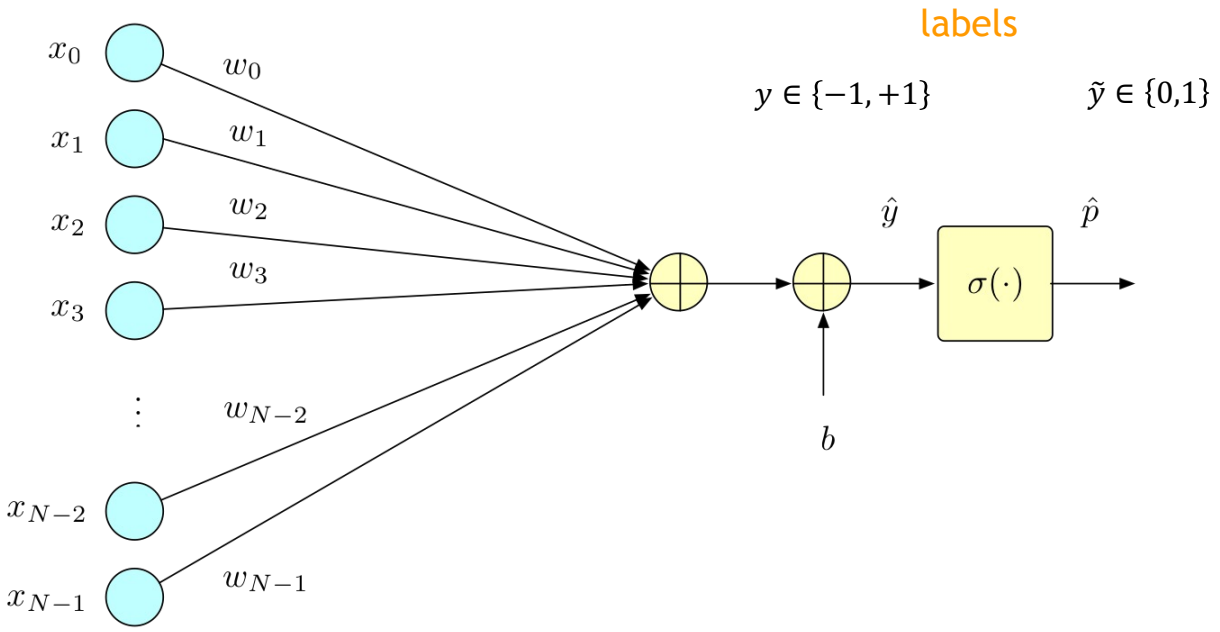
Example:

Labels ($\tilde{y}$): 1, 0, 1

Output ($\sigma(w^T x)$): 0.9, 0.1, 0.2

$NLL(w)$: 0.11 + 0.11 + 1.6

LOGISTIC REGRESSION



Two problems addressed:

1. Logistic function maps LLR to p_1
2. Binary cross-entropy is the loss that arises from a Bernoulli model and maximum likelihood parameter estimation.

LOGISTIC REGRESSION

Summary:

Logistical regression is ML estimation of $\boldsymbol{w}$ for an i.i.d. Bernoulli model with

$$p_n = \sigma(\boldsymbol{w}^T \boldsymbol{x})$$

which can be viewed as regression with the (empirical) binary cross-entropy cost function

no closed form, usually use SGD to perform the regression

We will see that this is a special case of two concepts:

1. It is a single-perceptron and MLP (neural networks) are many of these combined (with slight modification).
2. The loss function derived is the binary cross-entropy between the output probability mass function $(p, 1-p)$ and the "one-hot" encoded label pmf $\tilde{y}$.

SINGLE PERCEPTRON HISTORY

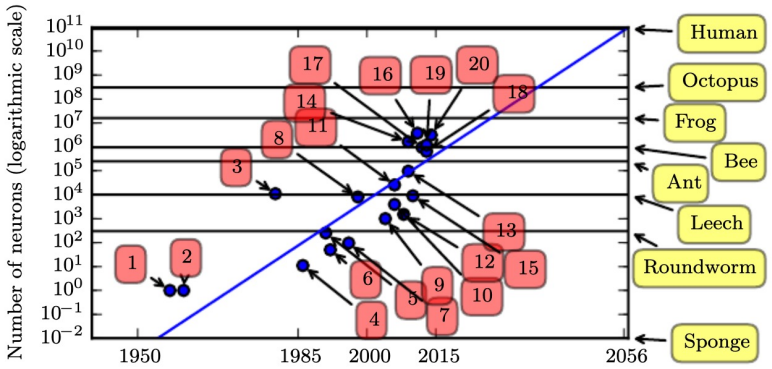


Figure 1.11: Increasing neural network size over time. Since the introduction of hidden units, artificial neural networks have doubled in size roughly every 2.4 years. Biological neural network sizes from Wikipedia (2015).

1. Perceptron (Rosenblatt, 1958, 1962)
2. Adaptive linear element (Widrow and Hoff, 1960)
3. Neocognitron (Fukushima, 1980)
4. Early back-propagation network (Rumelhart *et al.*, 1986b)
5. Recurrent neural network for speech recognition (Robinson and Fallside, 1991)
6. Multilayer perceptron for speech recognition (Bengio *et al.*, 1991)
7. Mean field sigmoid belief network (Saul *et al.*, 1996)
8. LeNet-5 (LeCun *et al.*, 1998b)
9. Echo state network (Jaeger and Haas, 2004)
10. Deep belief network (Hinton *et al.*, 2006)
11. GPU-accelerated convolutional network (Chellapilla *et al.*, 2006)
12. Deep Boltzmann machine (Salakhutdinov and Hinton, 2009a)
13. GPU-accelerated deep belief network (Raina *et al.*, 2009)
14. Unsupervised convolutional network (Jarrett *et al.*, 2009)
15. GPU-accelerated multilayer perceptron (Ciresan *et al.*, 2010)
16. OMP-1 network (Coates and Ng, 2011)
17. Distributed autoencoder (Le *et al.*, 2012)
18. Multi-GPU convolutional network (Krizhevsky *et al.*, 2012)
19. COTS HPC unsupervised convolutional network (Coates *et al.*, 2013)
20. GoogLeNet (Szegedy *et al.*, 2014a)

this model was proposed with a simple learning algorithm (special case of SGD)

BACKGROUND SUMMARY

MAIN IDEAS FROM BACKGROUND

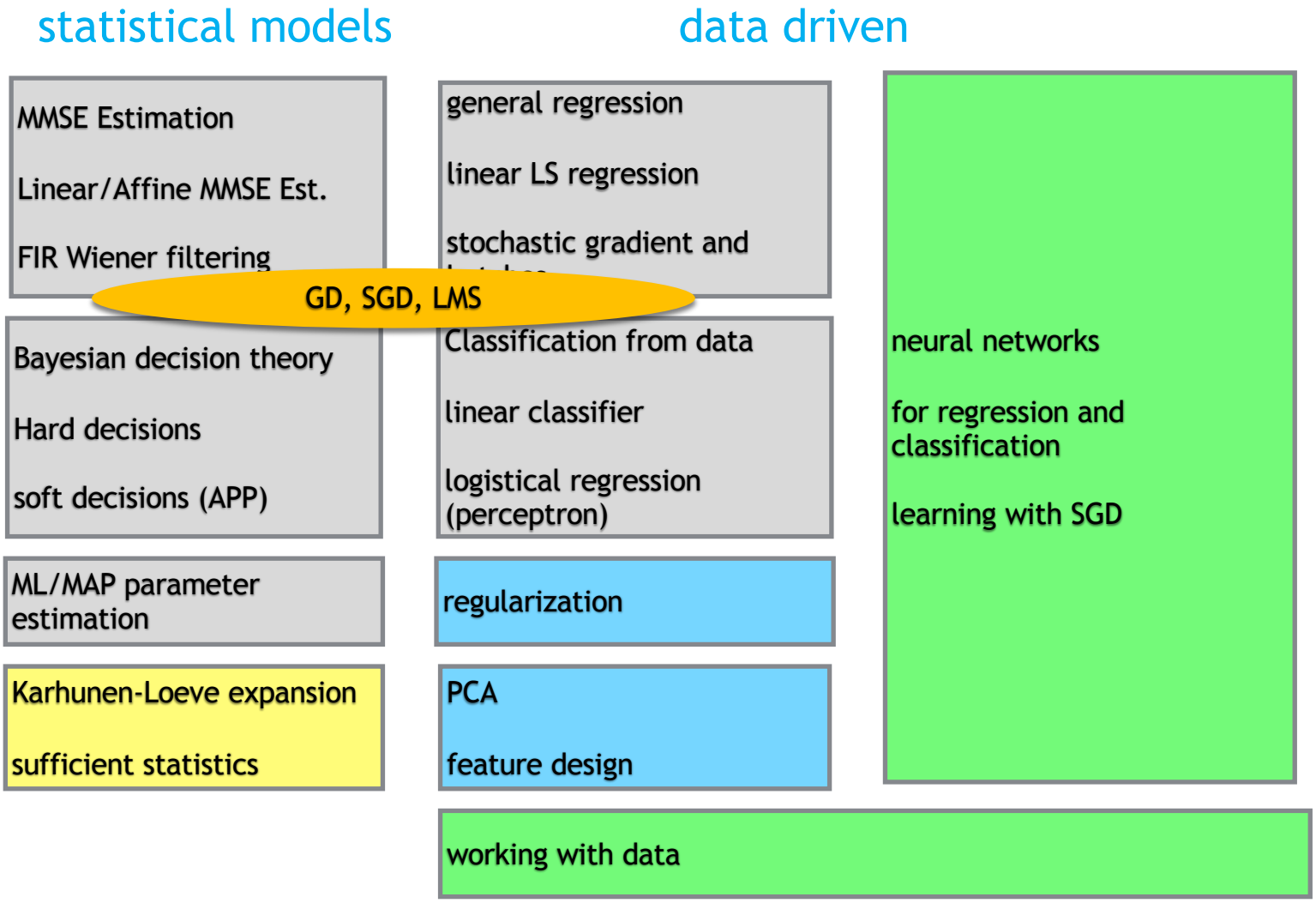
- Random vectors
  - Eigenvalues of covariance matrix provides information regarding direction preferences (principal components)
  - May drop directions with very little energy/power
- Estimation
  - MMSE estimator is conditional expectation — difficult to find
  - Linear/Affine MMSE is simple and only depends on second moments
  - For jointly-Gaussian observed/desired, affine is optimal
- Detection
  - MAP rule is minimum error probability.
  - Requires complete statistical description

MAIN IDEAS FROM BACKGROUND

- Regression (from data)
  - Linear regression is same as affine/linear MMSE estimation, but with data averaging replacing ensemble averaging
  - Stacking interpretation
  - ML parameter interpretation
  - MAP parameter interpretation for regularization
- Classification (form data)
  - Linear classifier: linear regression with +/- 1 target and “slicer”
  - Logistic regression
- Information Theory:
  - ML parameter estimation ==> Empirical Cross-entropy loss function
  - Only called CE for classification tasks

REGULARIZATION

ESTIMATION, REGRESSION, CLASSIFICATION



REGULARIZATION

What is regularization and why do it?

We have seen an example: enforce penalty on weights to bias toward a prior distribution.

effect is to reduce over-fitting (get smaller weights)

Not all regularization methods can be viewed this way

in some cases, intuitive, empirical penalty enforcing functions are used

What is a more general definition of regularization?

REGULARIZATION

What is regularization and why do it?

We have seen an example: enforce penalty on weights to bias toward a prior distribution.

effect is to reduce over-fitting (get smaller weights)

Not all regularization methods can be viewed this way

in some cases, intuitive, empirical penalty enforcing functions are used

What is a more general definition of regularization?

regularization is anything you do in training that is aimed at improving generalization over accuracy — *i.e.*, anything that does not optimize the cost on the training data

we will see very different versions of this — *e.g.*, drop-out

REGULARIZATION INTERPRETATION

$$\max_{\theta} p_{y(t)|\boldsymbol{x}(t),\Theta(t)}(y|\boldsymbol{x},\theta)p_{\Theta(t)|\boldsymbol{x}(t)}(\theta|\boldsymbol{x}) \quad \Leftrightarrow \quad \min_{\boldsymbol{w}} \|\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w}\|^2 + \lambda\|\boldsymbol{w}\|^2$$

The *a-priori* Gaussian distribution on the weights leads to "L2 regularization"

penalizes large $\boldsymbol{w}$ — even if large $\boldsymbol{w}$ cause smaller squared error

this can be viewed a method to combat **over-fitting**

λ is called the regularization coefficient in this context

Larger $\lambda \rightarrow$ penalize larger weights more aggressively (at expense of SE)

REGULARIZATION INTERPRETATION

Another popular type of regularizer is "L1 regularizer"

for example, for squared-error cost function with L1 regularization:

$$\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}\|^2 + \lambda \|\boldsymbol{w}\|_1$$

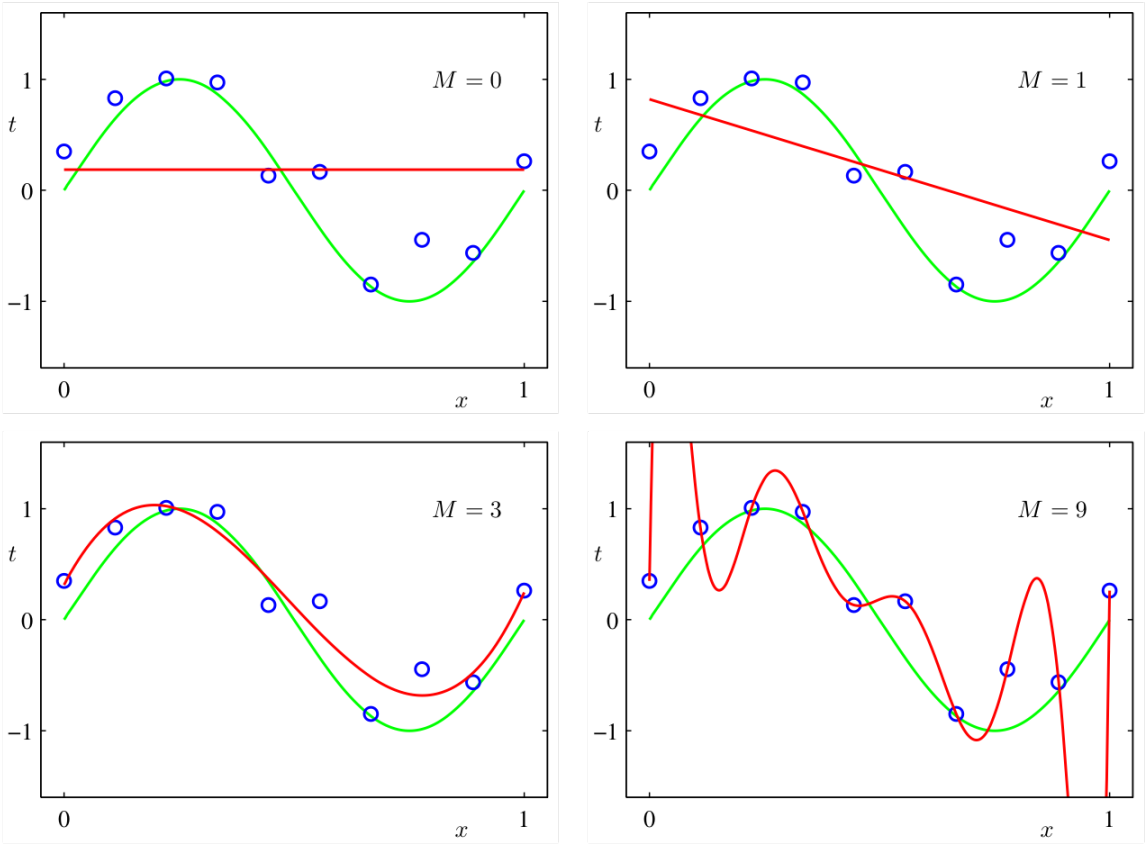
$$\|\boldsymbol{w}\|_1 = \sum_i |w_i|$$

Questions:

- does this correspond to an *a-priori* distribution on the weights?
- If so, which one?
- Qualitatively, what is the difference between L1 and L2 regularization?

REGRESSION FROM DATA

under-fitting



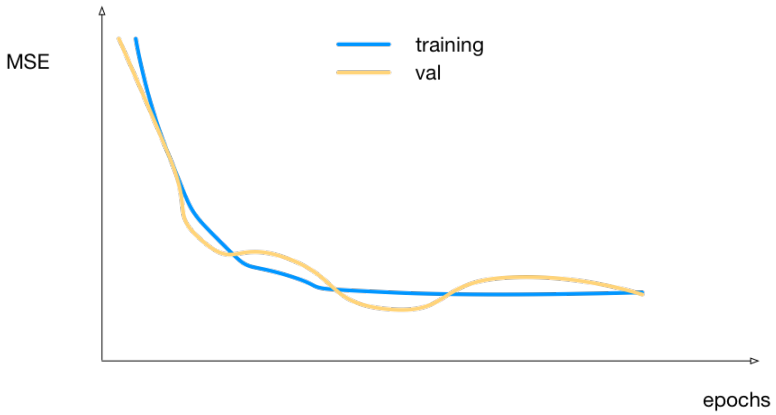
desired behavior

over-fitting

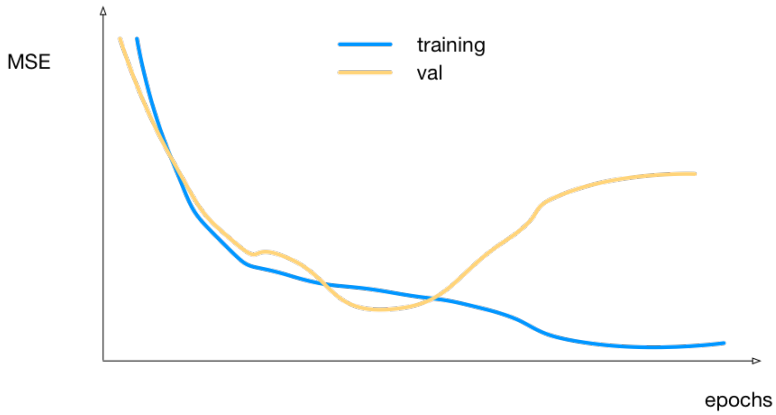
Figure 1.4 Plots of polynomials having various orders M, shown as red curves, fitted to the data set shown in Figure 1.2.

Choosing the right model (complexity) is challenging given a finite data set and no good model for what generated it!!!

OVER-FITTING



desired behavior



typical over-fitting

Better performance on training and worse or non-improving on validation (for $P[\text{correct}]$ classification, it gets higher)

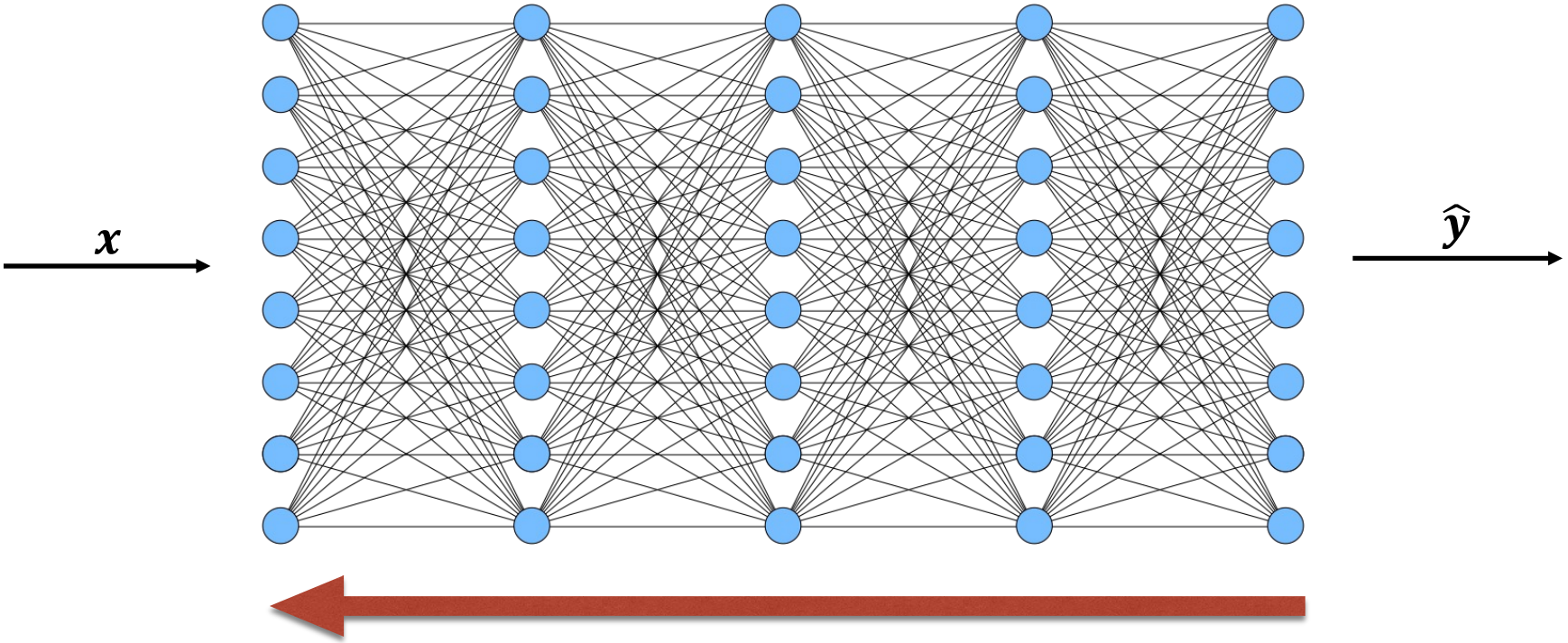
MULTILAYER PERCEPTRONS (MLP)

MULTILAYER PERCEPTRON NETWORKS (MLPS)

Forward propagation (inference and training)

$$\boldsymbol{a}^{(l)} = \underline{h}\left(\boldsymbol{W}_l \boldsymbol{a}^{l-1} + \boldsymbol{b}_l\right) \qquad \Theta = \{\boldsymbol{W}_l, \boldsymbol{b}_l\}_{l=1}^{L}$$

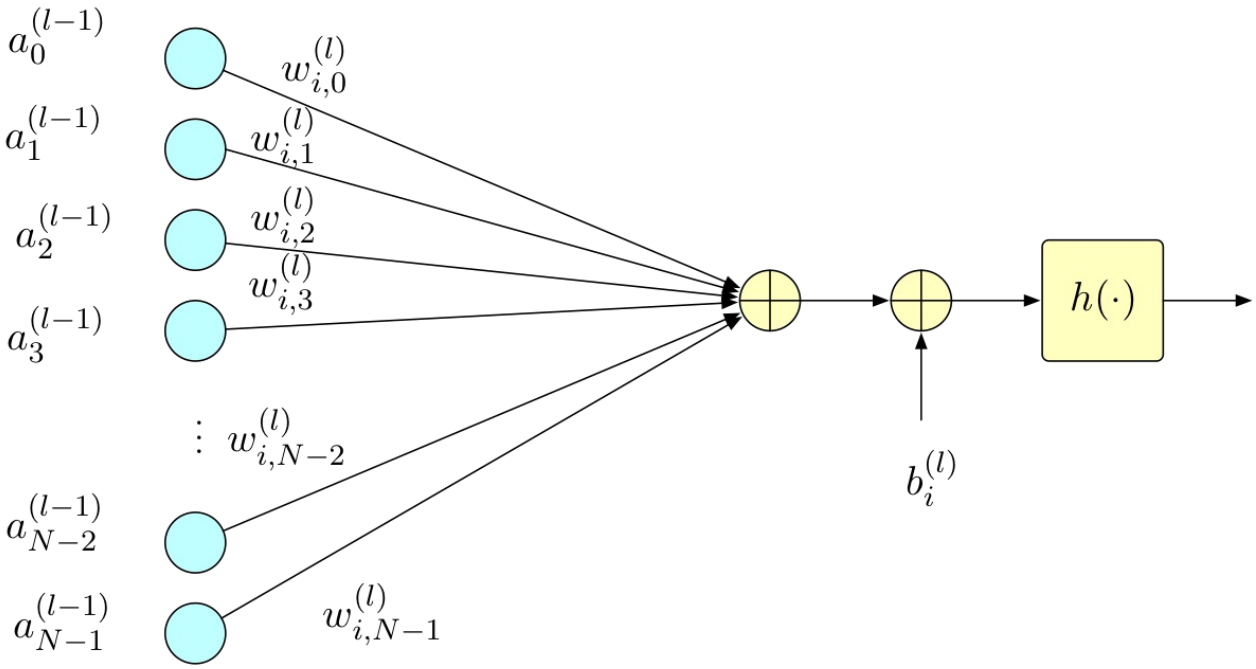
(trainable parameters)



Backward propagation (training)

Learn the trainable parameters using SGD and the chain-rule

MLP FORWARD PROPAGATION DETAILS



$$a_i^{(l)} = h\left(\left[\mathbf{w}_i^{(l)}\right]^{\mathsf{t}} \mathbf{a}^{(l-1)} + b_i^{(l)}\right)$$

processing at the i^{th} neuron (node) at layer l

look familiar?