

Neural networks

Geir Storvik and Aliaksandr Hubin

March 18, 2022

- Flexible parametric models
- Universal approximators
- Successful in applications
- Image analysis
- Natural language processing
- No black magic same math as for Gaussian and/or Logistic regression
- But over-parametrization
- Lack of interpretability
- Difficult to reproduce the results due to non-concavity of the loss

- Principal components:

$$z_{ij} = \sum_{h=1}^p \alpha_{hj} x_{ih} = \boldsymbol{\alpha}_j^T \mathbf{x}_i \quad j = 1, \dots, J$$

$$y_i = \beta_0 + \sum_{j=1}^J \beta_j z_{ij} + \varepsilon_i = \underbrace{\beta_0 + \boldsymbol{\beta}^T \mathbf{z}_i}_{T_i} + \varepsilon_i$$

- $z_{ij}, j = 1, \dots, J$ can be considered as some **latent** (or hidden) variables.
- Generalization, **One-layer Neural network**:

$$z_{ij} = f_0(\boldsymbol{\alpha}_j^T \mathbf{x}_i), \quad j = 1, \dots, J \quad (1)$$

$$T_i = \beta_0 + \boldsymbol{\beta}^T \mathbf{z}_i \quad (2)$$

$$y_i = f_1(T_i) + \varepsilon_i = f_1(\beta_0 + \boldsymbol{\beta}^T \mathbf{z}_i) + \varepsilon_i \quad (3)$$

where both $f_0(\cdot)$ and $f_1(\cdot)$ can be **non-linear** functions (aka **activation functions**).

Graphical display of 1-layer NN

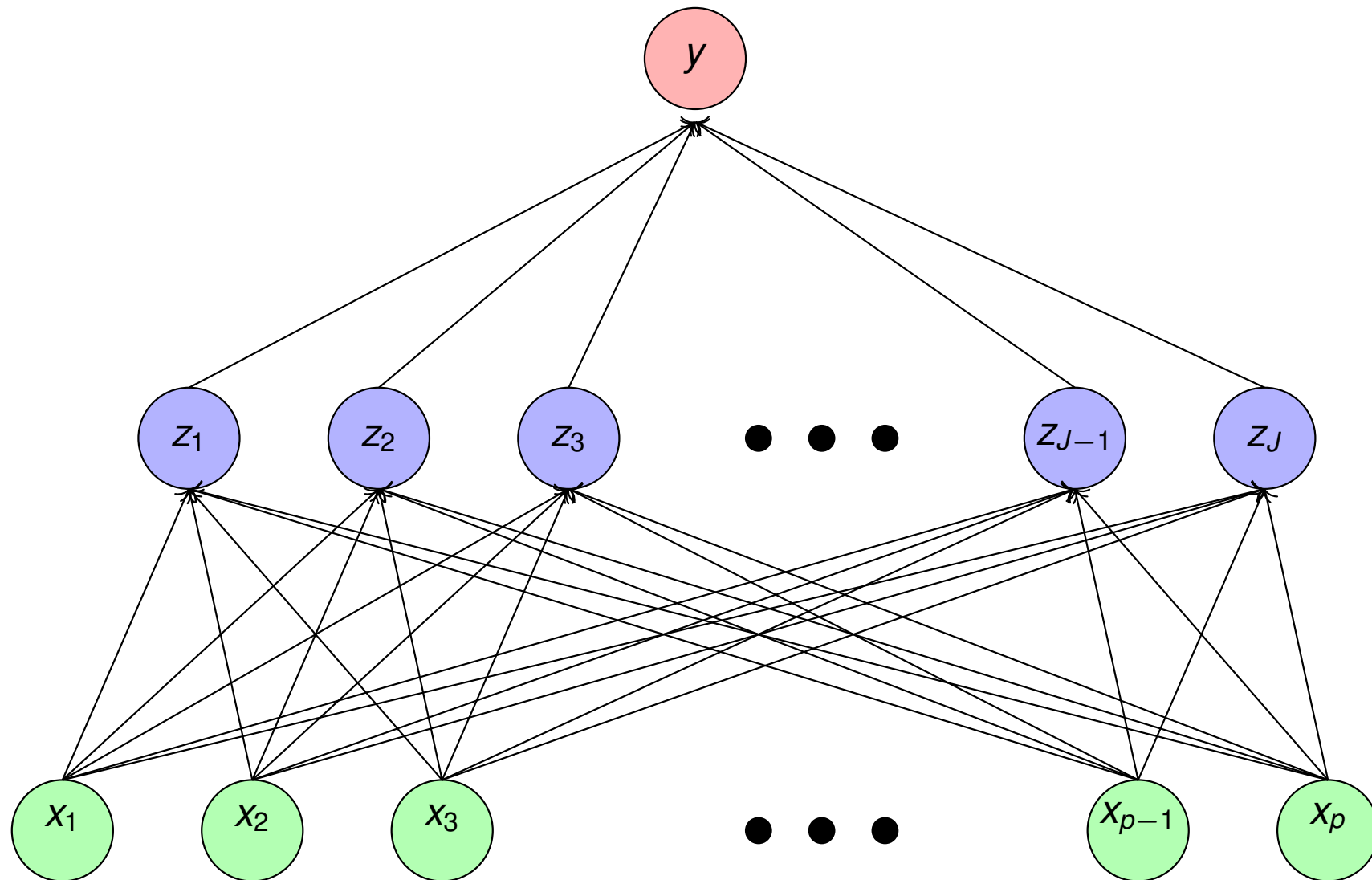


Figure: Visualisation of neural network with one hidden layer.

Rewriting the model

- Model

$$z_{ij} = f_0(\alpha_j^T \mathbf{x}_i), \quad j = 1, \dots, J \quad (4)$$

$$y_i = f_1(\beta_0 + \beta^T \mathbf{z}_i) + \varepsilon_i \quad (5)$$

- Equivalent to

$$y_i = f(\mathbf{x}_i) + \varepsilon_i$$

where

$$f(\mathbf{x}_i) = f_1\left(\beta_0 + \sum_{j=1}^J \beta_j f_0\left(\sum_{h=1}^p \alpha_{hj} x_{ih}\right)\right).$$

- When making predictions **predictions**

$$z_{ij} = f_0(\hat{\alpha}_j^T \mathbf{x}_i), \quad j = 1, \dots, J \quad (6)$$

$$T_i = \hat{\beta}_0 + \hat{\beta}^T \mathbf{z}_i \quad (7)$$

$$\hat{y}_i = f_1(T_i) = f(\mathbf{x}_i) \quad (8)$$

Choice of f_0 and f_1

- Many **possible** choices
- Most common in regression:

$$f_0(x) = \frac{1}{1 + \exp(-x)}$$

sigmoid/logistic function;

$$f_1(z) = z$$

identity function.

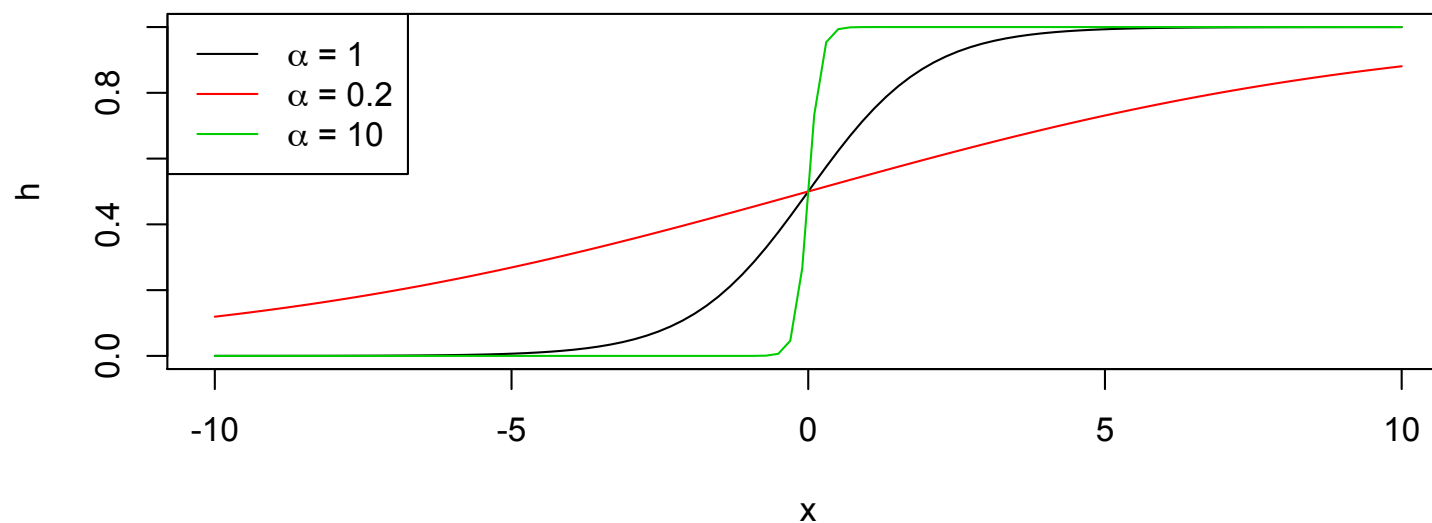
- Alternatives

$$f_0(x) = \begin{cases} \tanh(x) & \text{Hyperbolic tangent} \\ \max\{0, x\} & \text{ReLU function} \\ \max\{ax, x\} & \text{Leaky ReLU} \end{cases}$$

The sigmoid function

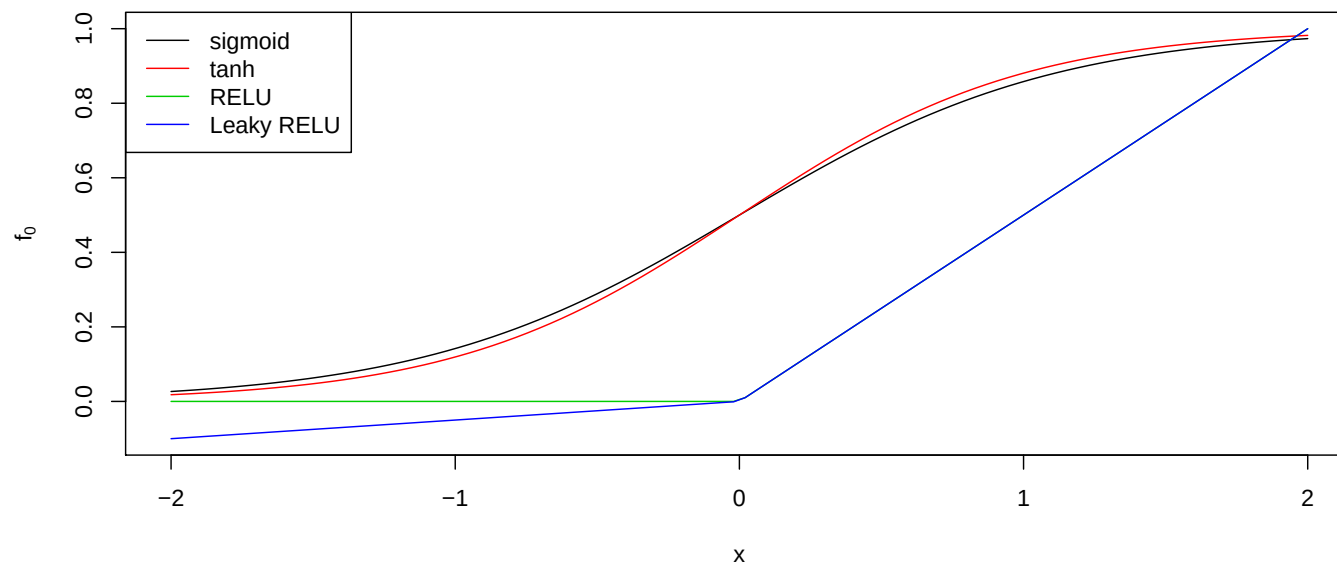
- Form:

$$f_0(x) = 1 / (1 + \exp(-\alpha * x))$$



- Small values of α : Almost linear
- Large values of α : Threshold function

Comparison of different activation functions



Estimation of parameters

- **Many** parameters to be estimated:

$$\theta = \{\alpha_{hj}, j = 1, \dots, J, h = 1, \dots, p, \beta_j, j = 0, \dots, J\}$$

- Criterion (typically negative log likelihood), e.g. for the Gaussian responses:

$$D(\theta) = \sum_{i=1}^n (y_i - f(\mathbf{x}_i))^2 = \sum_{i=1}^n \left(y_i - f_1 \left(\beta_0 + \sum_{j=1}^J \beta_j f_0 \left(\sum_{h=1}^p \alpha_{hj} x_{ih} \right) \right) \right)^2$$

- Or for the Bernoulli responses:

$$D(\theta) = - \sum_{i=1}^n (y_i \log(f(\mathbf{x}_i)) + (1 - y_i) \log(1 - f(\mathbf{x}_i)))$$

- Due to possible overfitting we introduce a **penalty term** with a **weight decay** λ):

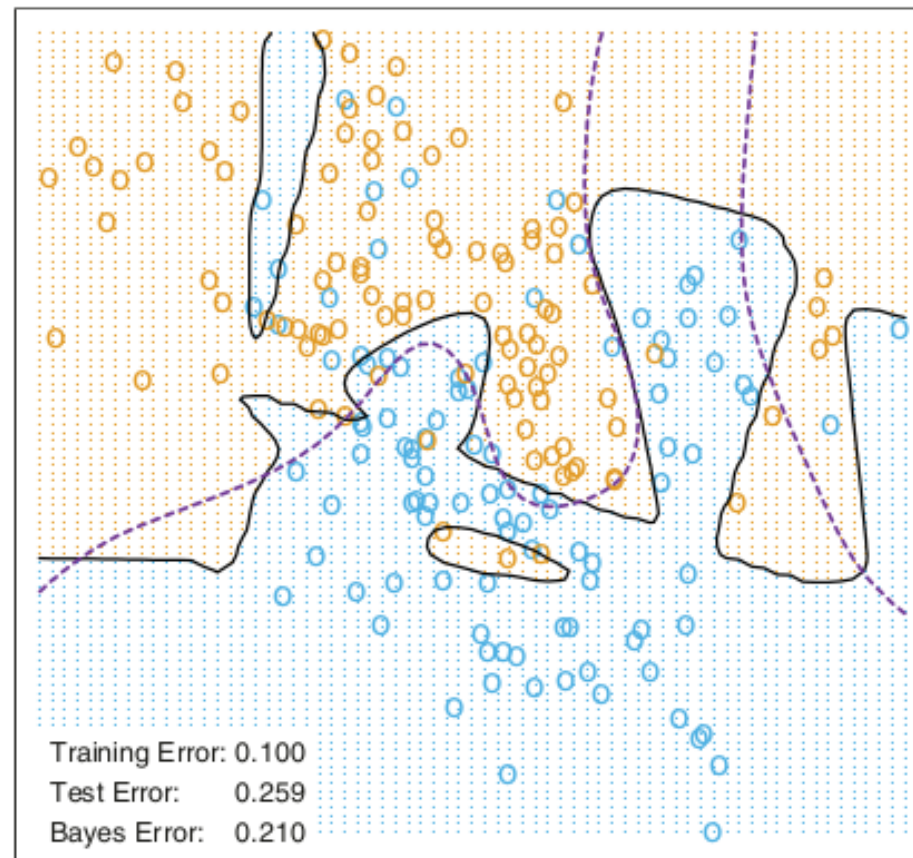
$$D_0(\theta) = D(\theta) + \lambda J(\theta)$$

$$J(\theta) = \sum_{j=1}^J \sum_{h=1}^p \alpha_{hj}^2 + \sum_{j=1}^J \beta_j^2 \text{ or } J(\theta) = \sum_{j=1}^J \sum_{h=1}^p \frac{\alpha_{hj}^2}{1 + \alpha_{hj}^2} + \sum_{j=1}^J \frac{\beta_j^2}{1 + \beta_j^2}$$

- Estimation typically through **gradient descent methods**

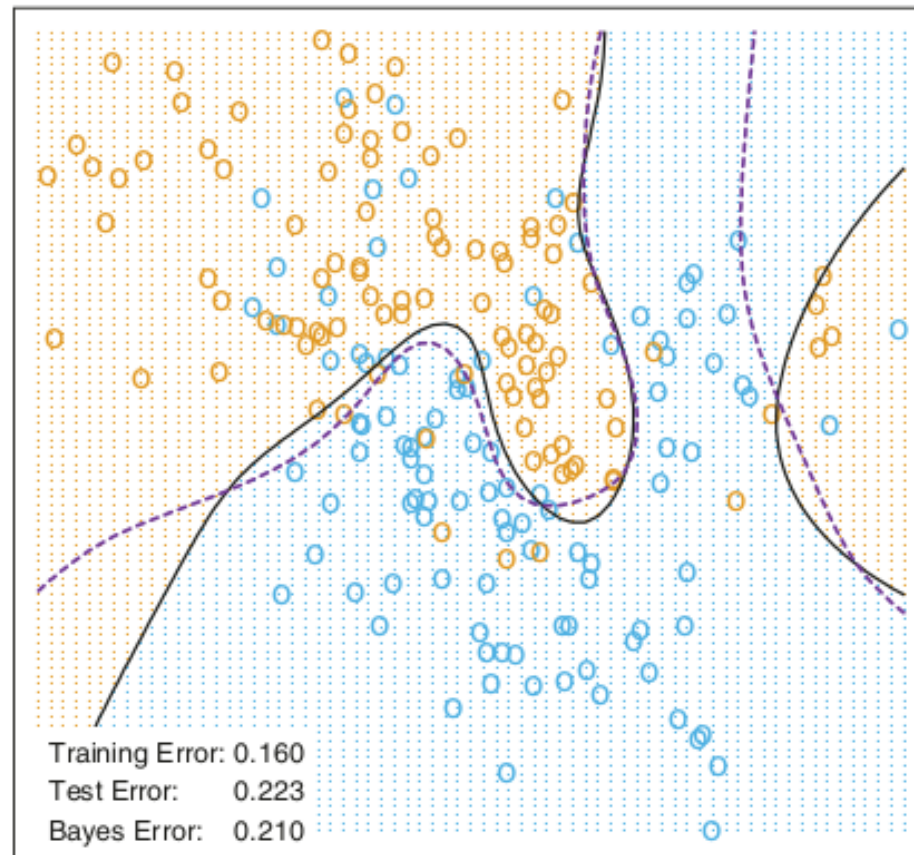
Flexible decision boundaries

Neural Network - 10 Units, No Weight Decay



Flexible decision boundaries

Neural Network - 10 Units, Weight Decay=0.02



The derivatives

- We want to minimize

$$D_0(\boldsymbol{\theta}) = \sum_{i=1}^n \left(y_i - f_1 \left(\beta_0 + \sum_{j=1}^J \beta_j f_0 \left(\sum_{h=1}^p \alpha_{hj} x_{ih} \right) \right) \right)^2 + \lambda \left(\sum_{j=1}^J \sum_{h=1}^p \alpha_{hj}^2 + \sum_{j=1}^J \beta_j^2 \right)$$

- Assume $f_0(\cdot)$ and $f_1(\cdot)$ are smooth and have smooth derivatives:

$$\frac{\partial D_0(\boldsymbol{\theta})}{\partial \beta_{j^*}} = -2 \sum_{i=1}^n (y_i - f(\mathbf{x}_i)) \frac{\partial f(\mathbf{x}_i)}{\partial \beta_{j^*}} + 2\lambda \beta_{j^*}$$

$$\begin{aligned} \frac{\partial f(\mathbf{x}_i)}{\partial \beta_{j^*}} &= f_1' \left(\beta_0 + \sum_{j=1}^J \beta_j f_0 \left(\sum_{h=1}^p \alpha_{hj} x_{ih} \right) \right) \cdot f_0 \left(\sum_{h=1}^p \alpha_{hj^*} x_{ih} \right) \\ &= f_1'(T_i) z_{ij^*} \end{aligned}$$

$$\frac{\partial D_0(\boldsymbol{\theta})}{\partial \alpha_{h^* j^*}} = -2 \sum_{i=1}^n (y_i - f(\mathbf{x}_i)) \frac{\partial f(\mathbf{x}_i)}{\partial \alpha_{h^* j^*}} + 2\lambda \alpha_{h^* j^*}$$

$$\begin{aligned} \frac{\partial f(\mathbf{x}_i)}{\partial \alpha_{h^* j^*}} &= f_1' \left(\beta_0 + \sum_{j=1}^J \beta_j f_0 \left(\sum_{h=1}^p \alpha_{hj} x_{ih} \right) \right) \cdot \beta_{j^*} f_0' \left(\sum_{h=1}^p \alpha_{hj^*} x_{ih} \right) \cdot x_{ih^*} \\ &= f_1'(T_i) \cdot \beta_{j^*} \cdot f_0' \left(\sum_{h=1}^p \alpha_{hj^*} x_{ih} \right) \cdot x_{ih^*} \end{aligned}$$

Back-propagation algorithm

- Gradient descent:

$$\beta_j^{r+1} = \beta_j^r - \gamma_r \frac{\partial D_0(\theta)}{\beta_j} \Big|_{\theta=\theta^r} \quad (9)$$

$$\alpha_{hj}^{r+1} = \alpha_{hj}^r - \gamma_r \frac{\partial D_0(\theta)}{\alpha_{hj}} \Big|_{\theta=\theta^r} \quad (10)$$

- Iteration until convergence (local optimum)
- Calculation: **Back-progragation**
 - **Forward** for calculating z_{ij} and T_i
 - **Backwards** for calculating $f'_1(T_i)$ and $f'_0(\sum_{h=1}^p \alpha_{hj} * x_{ih})$
- Tuning parameter γ_r called **learning rate**.
Usually constant but can be a sequence decreasing slowly towards zero
- Converge can be made faster by using **second derivatives**
- Typically non-concave/convex optimization
- A good idea to normalize the covariates to have the same regularization
 - But more computation at each iteration
- `credit_nn.R`, `auto_nn.R`

Extensions

- Several response variables. Regression

$$z_{ij} = f_0(\alpha_j^T \mathbf{x}_i), \quad j = 1, \dots, J \quad (11)$$

$$y_{ik} = f_1(T_{ik}) + \varepsilon_i = f_1(\beta_{0k} + \beta_k^T \mathbf{z}_{ik}) + \varepsilon_i \quad (12)$$

- Classification: Start by defining dummy variables

$$y_{ik} = \begin{cases} 1 & \text{if } y_i = k \\ 0 & \text{otherwise} \end{cases}$$

Note that $y_i = \arg \max_k y_{ik}$.

- Neural network: Predict y_{ik} through

$$z_{ij} = f_0(\hat{\alpha}_j^T \mathbf{x}_i), \quad j = 1, \dots, J \quad (13)$$

$$T_{ik} = \hat{\beta}_{0k} + \hat{\beta}_k^T \mathbf{z}_{ik}, \quad k = 1, \dots, K \quad (14)$$

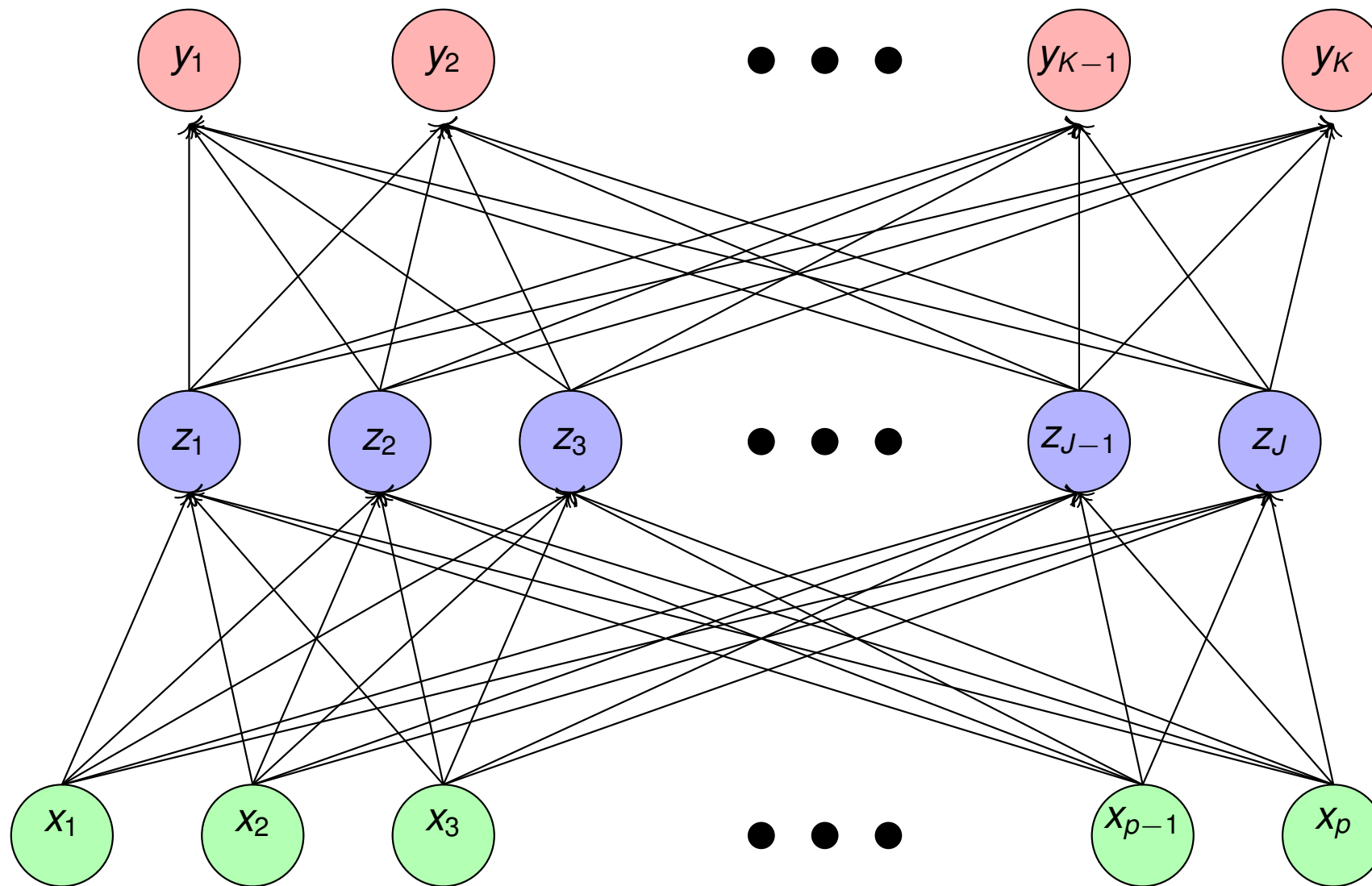
$$\hat{y}_i = \arg \max_k f_1(T_{ik}) \quad (15)$$

- Alternative through soft-max function just like in Multinomial regression:

$$\hat{\Pr}(Y_i = k) = \frac{e^{T_{ik}}}{\sum_{l=1}^L e^{T_{il}}}$$

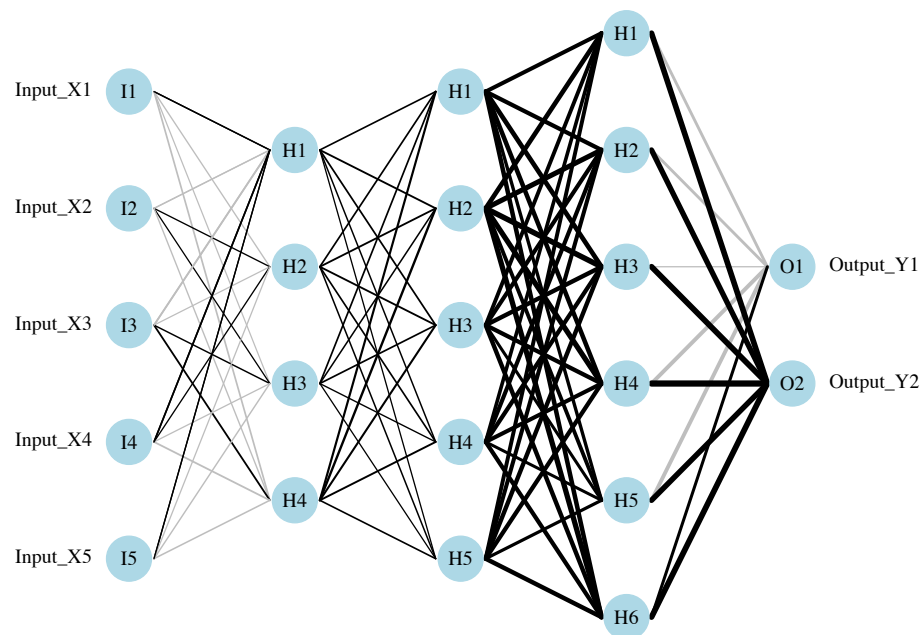
- `phonemne_nn.R`, `zip_nn.R`

Neural net with K classes



Deep learning

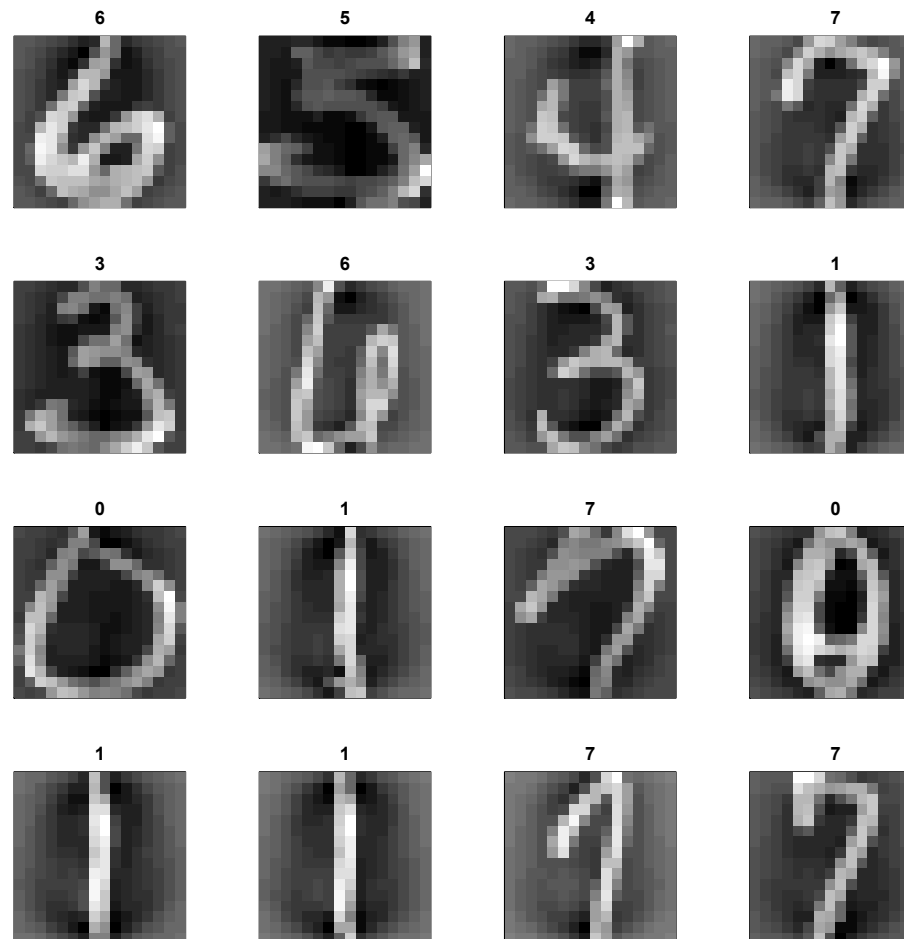
- So far: Only **one layer** of latent variables
- Also possible with **several layers**



- Gives more flexibility, more possibilities for interactions and non-linearities
- **Can** give many parameters/weights
 - Example gives $5 * 4 + 4 * 5 + 5 * 6 + 6 * 2 = 82$ weights
 - With biases we have $5 * 4 + 4 + 4 * 5 + 5 + 5 * 6 + 5 + 6 * 2 + 2 = 99$ parameters
 - One layer with 15 latent variables gives $5 * 15 + 15 * 2 = 105$ weights or 120 parameters
- Overfitting also a problem here.

Example: Zip data

- 16×16 greytone images of handwritten numbers
- $p = 256$ explanatory variables
- $K = 10$ classes



Neural net in R

- Many different packages
- `nnet` package. 10 latent variable:

```
ntrain = nrow(zip.train)
library(nnet)
zip.nnet = nnet(c1~., data=zip[1:ntrain,], size=10, decay=0.1,
  MaxNWts=3000, maxit=300)
```

- `zip_nn.R`

Number of latent variables	2	5	10	15	20	30
Number of parameters	532	1330	2660	3990	5320	7980
Error rate	0.42	0.17	0.11	0.09	0.07	0.065

Example zip-code

- 3 layers, 10 variables in each layer

	0	1	2	3	4	5	6	7	8	9
0	348	0	3	1	2	1	1	0	2	1
1	0	252	1	2	1	2	4	1	0	1
2	3	0	175	2	6	2	2	1	7	0
3	5	0	8	135	0	12	0	0	5	1
4	1	2	5	2	176	0	2	1	2	9
5	4	0	1	7	1	140	3	1	2	1
6	4	0	4	0	2	3	156	0	1	0
7	1	1	1	1	3	1	0	132	1	6
8	3	2	3	2	3	7	5	2	134	5
9	0	0	0	0	2	1	0	2	3	169

- Error rate 0.095

Different number of layers

- For vizualisation, input only first 10 principal components

Layer:	Number of z-variables	Number of weights	Error rate
(12)	12	240	0.160
(6,6)	12	156	0.209
(4,4,4)	12	112	0.254
(3,3,3,3)	12	87	0.449
(11,6)	17	236	0.160
(6,10,6)	22	240	0.160
(5,8,8,5)	26	244	0.218

Properties neural networks

- Advantages
 - Flexibility
 - Impressive results in many large scale applications (image analysis, text processing)
 - Sequential upgrading
- Disadvantages
 - Can be time-consuming
 - Instability in estimation stage
 - Difficulties in choosing the number of latent variables, penalty value λ
 - Difficulties in obtaining standard errors and uncertainty measures
 - Interpretation
- Next time
 - Simple architectures
 - Bayesian NNs
 - A powerful and flexible R library `torch`.