Pattern Recognition 2015
Neural Networks (2)

Ad Feelders

Universiteit Utrecht

Error back-propagation

How to compute the weights in a neural network ?

- Evaluate derivatives of the error function with respect to the weights.
- Apply a gradient–based optimization algorithm to find the weight values at a local minimum of the error function.
- Most simple gradient–based optimization algorithm: gradient descent.

Back-propagation is a computationally efficient algorithm for evaluating derivatives of the error function in a multi-layered neural network.

General back-propagation

Consider an arbitrary feed-forward topology, arbitrary differentiable activation function h, and arbitrary differentiable error function E.

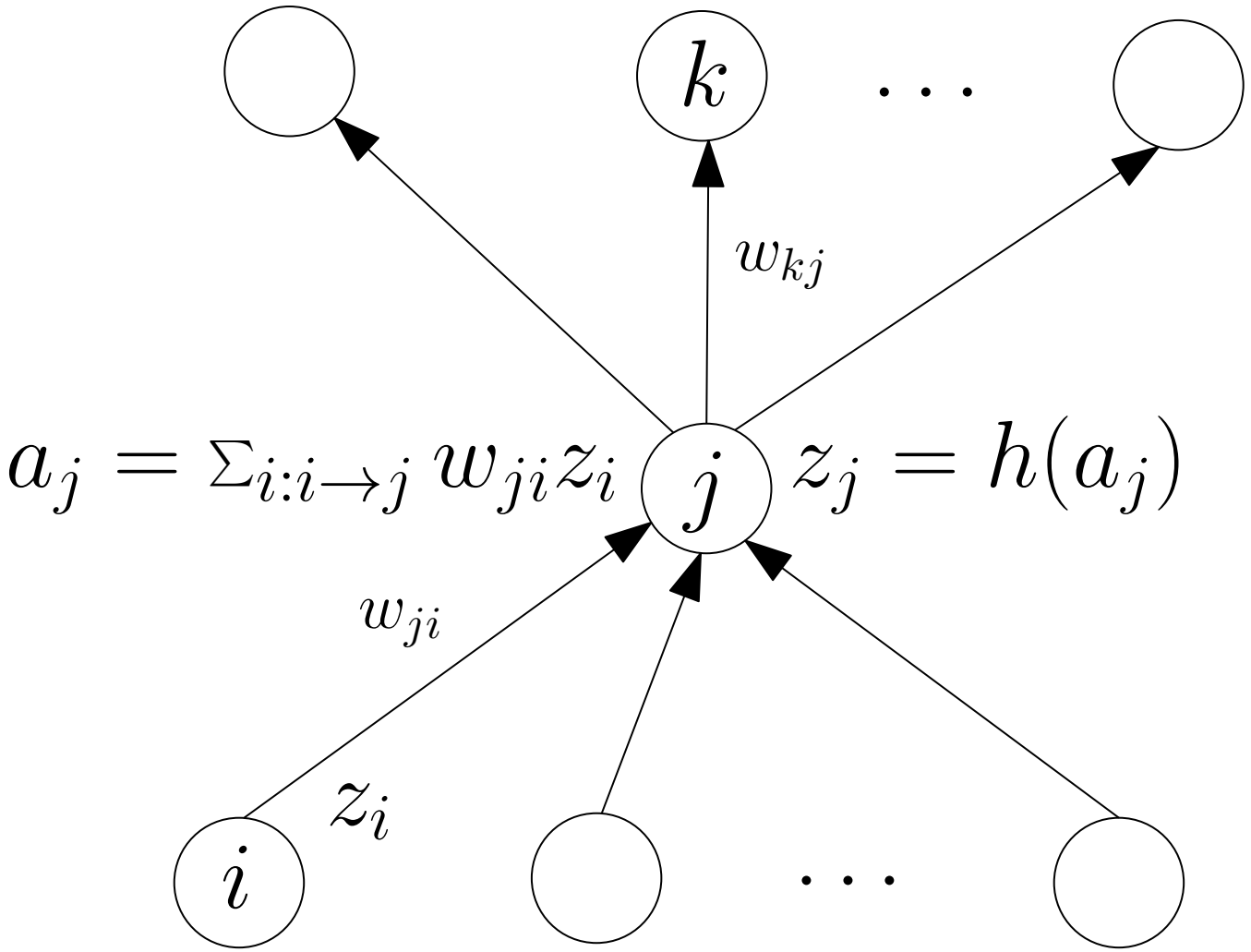
- Weighted input of neuron j

$$a_j = \sum_{i:\, i \to j} w_{ji} z_i$$

- Output / Non-linear activation function of neuron j: $z_j = h(a_j)$
- Error function (sum over all patterns n): $E = \sum_n E_n$
- Define for each neuron j the *local error* δ_j:

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j}$$

General network topology: notation



General Back-Propagation (continued)

We want to evaluate the error gradient:

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} \qquad \text{(chain rule: } E_n \text{ depends on } w_{ji} \text{ only through } a_j\text{)}$$

$$= \delta_j \frac{\partial a_j}{\partial w_{ji}} \qquad \left(\delta_j \equiv \frac{\partial E_n}{\partial a_j}\right)$$

$$= \delta_j z_i \qquad \left(\frac{\partial a_j}{\partial w_{ji}} = z_i\right)$$

We have

$$\frac{\partial a_j}{\partial w_{ji}} = z_i,$$

because

$$a_j = \sum_{i:\, i \to j} w_{ji} z_i.$$

General Back-Propagation (continued)

Only remains to compute δ_j for each neuron.

We consider output neurons and hidden neurons separately.

Output neuron:

$$\begin{aligned}\delta_k &\equiv \frac{\partial E_n}{\partial a_k} \\ &= \frac{\partial E_n}{\partial h(a_k)} \frac{\partial h(a_k)}{\partial a_k} && (E_n \text{ only depends on } a_k \text{ through } h(a_k)) \\ &= h'(a_k) \frac{\partial E_n}{\partial y_k} && (h'(a_k) \equiv \tfrac{\partial h(a_k)}{\partial a_k};\ h(a_k) \equiv y_k)\end{aligned}$$

This is how far we can go without making specific choices for the activation function and error function.

General Back-Propagation (continued)

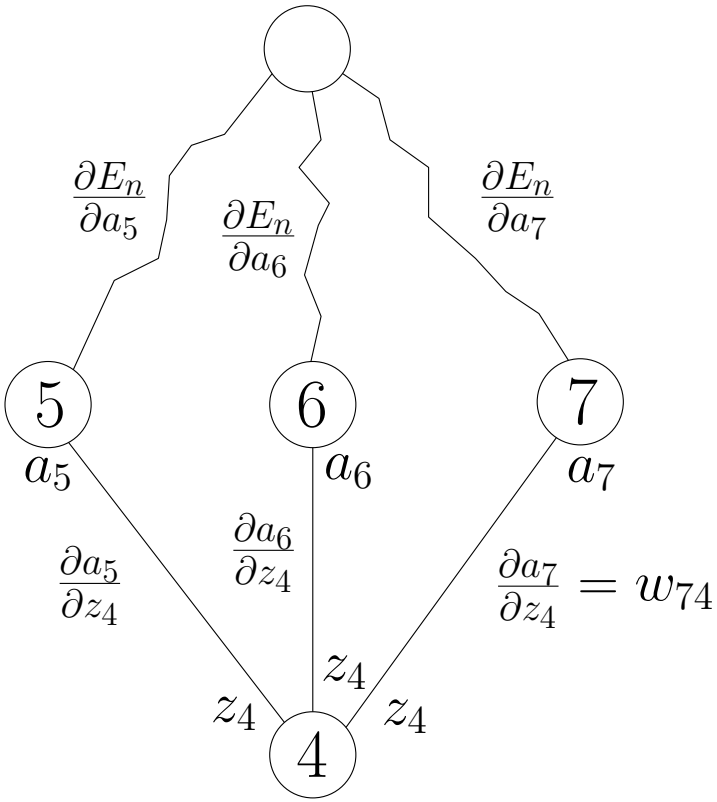
Hidden neuron:

$$\begin{aligned}
\delta_j \equiv \frac{\partial E_n}{\partial a_j} &= \frac{\partial E_n}{\partial h(a_j)} \frac{\partial h(a_j)}{\partial a_j} \\
&= h'(a_j) \frac{\partial E_n}{\partial z_j} && (h'(a_j) \equiv \tfrac{\partial h(a_j)}{\partial a_j};\ h(a_j) \equiv z_j) \\
&= h'(a_j) \sum_{k:\, j \to k} \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial z_j} && (\text{chain rule}) \\
&= h'(a_j) \sum_{k:\, j \to k} \frac{\partial E_n}{\partial a_k} w_{kj} && (\tfrac{\partial a_k}{\partial z_j} = w_{kj}) \\
&= h'(a_j) \sum_{k:\, j \to k} w_{kj} \delta_k && (\tfrac{\partial E_n}{\partial a_k} \equiv \delta_k;\ 5.56)
\end{aligned}$$

Since the formula for δ_j only contains terms in later layers, the local errors can be calculated from output to input on the network. Hence the term back-propagation.

Illustration of chain rule

$$E_n = f(a_5, a_6, a_7)$$



$$\frac{\partial E_n}{\partial z_4} = \frac{\partial E_n}{\partial a_5}\frac{\partial a_5}{\partial z_4} + \frac{\partial E_n}{\partial a_6}\frac{\partial a_6}{\partial z_4} + \frac{\partial E_n}{\partial a_7}\frac{\partial a_7}{\partial z_4}$$

Error Back-Propagation procedure

1. Forward propagation of input values: activations of all neurons
2. Compute δ_k for each output neuron k
3. Backward propagation: compute δ_j for each hidden neuron j
4. Compute derivative

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

for each weight w_{ji}

Back-Propagation for a specific case

Consider a two-layer network with logistic hidden neurons and linear output neurons:

$$y_k = w_{k0} + \sum_{j=1}^{M} w_{kj}\sigma\left(\sum_{i=0}^{D} w_{ji}x_i\right)$$

with the sum-of-squares error function:

$$E_n = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2$$

Back-Propagation for a specific case (continued)

Local error at the output neurons:

$$\delta_k = h'(a_k)\frac{\partial E_n}{\partial y_k} = y_k - t_k,$$

since $h(a_k) = a_k$ (linear output) and

$$\frac{\partial \frac{1}{2}(y_k - t_k)^2}{\partial y_k} = y_k - t_k$$

Local error at hidden neurons:

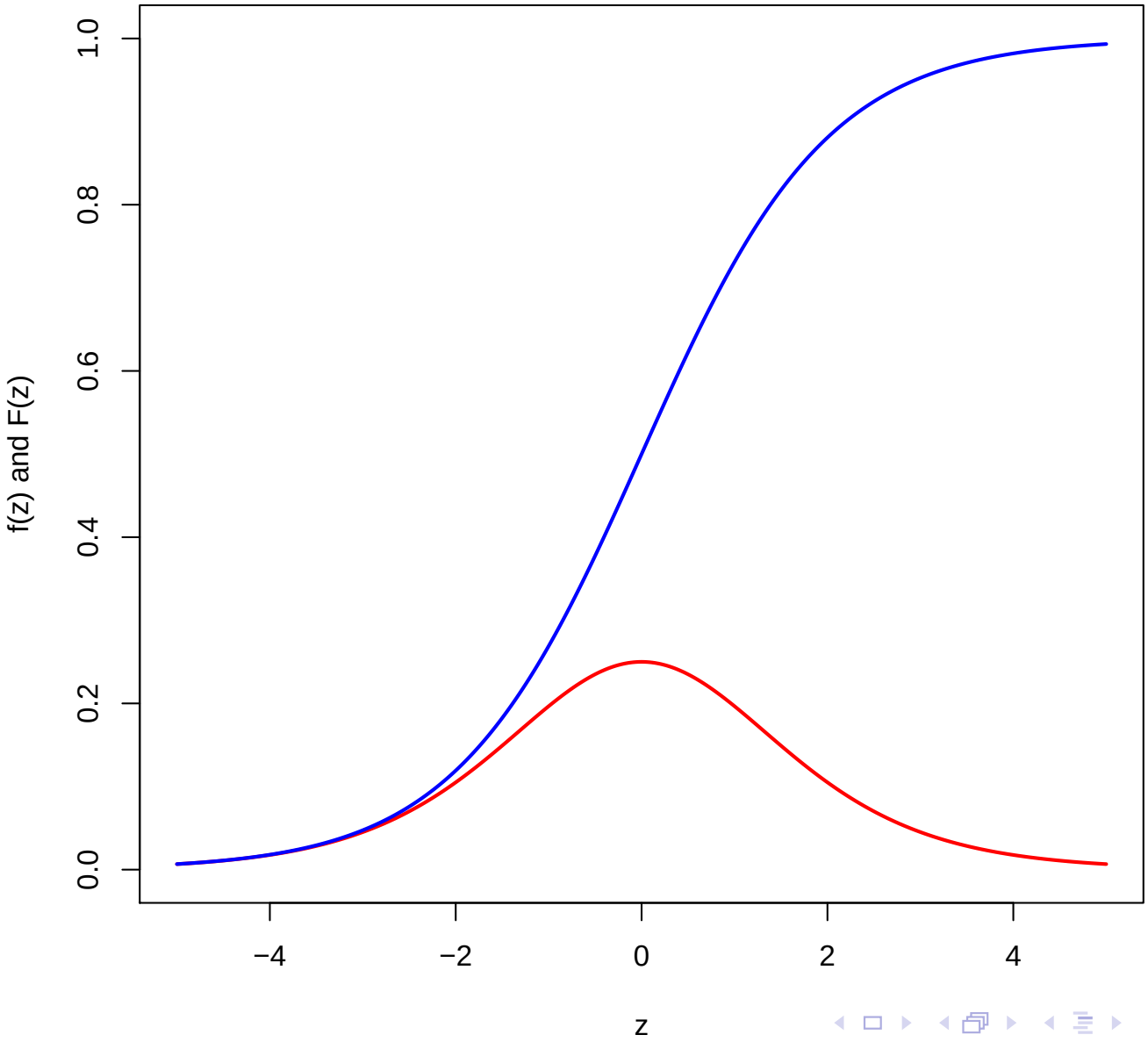
$$\delta_j = \sigma'(a_j)\sum_{k=1}^{K} w_{kj}\delta_k = z_j(1 - z_j)\sum_{k=1}^{K} w_{kj}\delta_k$$

Derivative of logistic activation function

$$\sigma(a) = \frac{e^a}{1+e^a} = (1+e^{-a})^{-1}$$

$$\begin{aligned}
\sigma'(a) &= \frac{d\,(1+e^{-a})^{-1}}{d\,a} \\
&= -(1+e^{-a})^{-2} \cdot -e^{-a} \\
&= \frac{e^{-a}}{(1+e^{-a})^2} \\
&= \frac{1}{1+e^{-a}} \cdot \frac{e^{-a}}{1+e^{-a}} \\
&= \sigma(a)(1-\sigma(a))
\end{aligned}$$

Logistic density (red) and cumulative density (blue)



Back-Propagation for a specific case (continued)

First-layer derivatives:

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j x_i$$

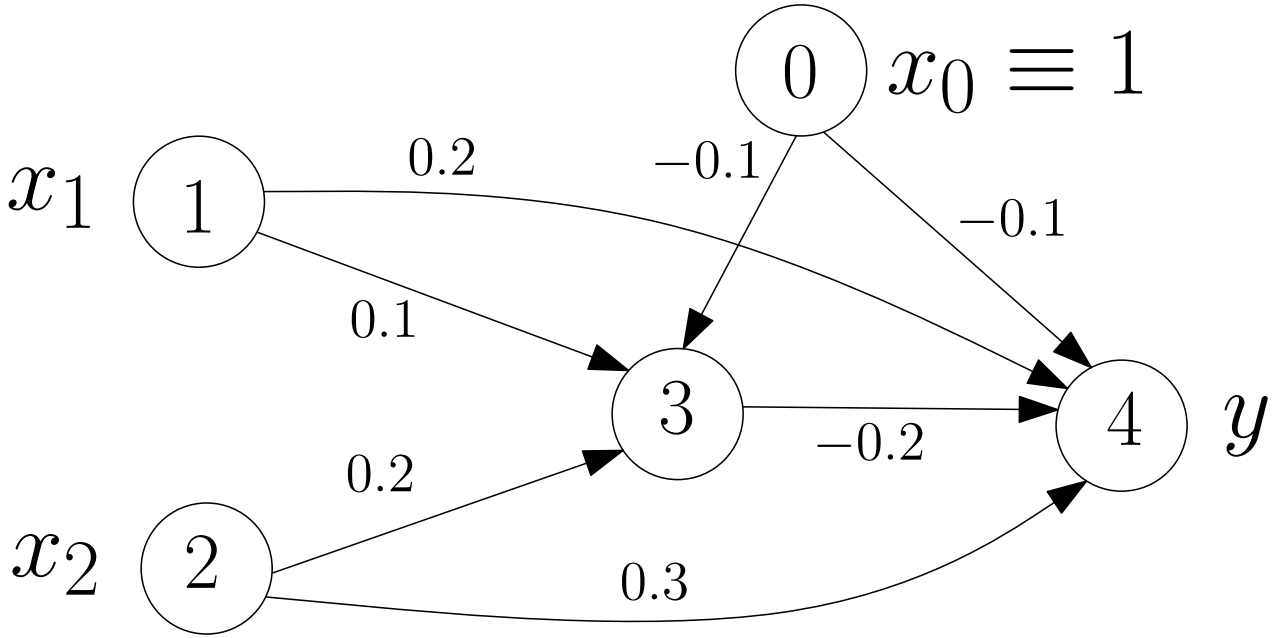
Second-layer derivatives:

$$\frac{\partial E_n}{\partial w_{kj}} = \delta_k z_j$$

Error Back-Propagation procedure: specific case

1. Apply input vector **x**, forward propagate activations through network
2. Compute δ_k for each output neuron k : $\delta_k = y_k - t_k$
3. Backward propagate δ_k's to compute δ_j for each hidden neuron j: $\delta_j = z_j(1 - z_j)\sum_{k=1}^{K} w_{kj}\delta_k$
4. Compute derivative $\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$ for each weight w_{ji}:
   1. First-layer derivatives:
   $$\frac{\partial E_n}{\partial w_{ji}} = \delta_j x_i$$
   2. Second-layer derivatives:
   $$\frac{\partial E_n}{\partial w_{kj}} = \delta_k z_j$$

Example Back-Propagation with skip-layer connections



- Output neuron: linear; hidden neuron: logistic
- Training pattern: $(x_1, x_2) = (2, 3), t = 0$
- Skip-layer connections!

Forward propagation: compute predicted target

To compute the predicted target value, we start at the input layer and work toward the output layer:

$$
\begin{aligned}
a_3 &= w_{30} + w_{31}x_1 + w_{32}x_2 \\
&= -0.1 + 0.1 \times 2 + 0.2 \times 3 = 0.7 \\
z_3 &= \frac{1}{1 + e^{-a_3}} = \frac{1}{1 + e^{-0.7}} = 0.67 \\
a_4 &= w_{40} + w_{41}x_1 + w_{42}x_2 + w_{43}z_3 \\
&= -0.1 + 0.2 \times 2 + 0.3 \times 3 - 0.2 \times 0.67 = 1.07 \\
y &= z_4 = a_4 = 1.07
\end{aligned}
$$

Backward propagation

To compute the local errors we start at the output layer, and work toward the input layer:

$$\delta_4 = y - t = 1.07 - 0 = 1.07 \qquad \text{(output unit)}$$

$$\begin{aligned} \delta_3 &= z_3(1 - z_3)w_{43}\delta_4 \qquad \text{(hidden unit)} \\ &= 0.67(1 - 0.67)(-0.2)1.07 \\ &= -0.05 \end{aligned}$$

Error derivatives

$$\frac{\partial E_n}{\partial w_{40}} = \delta_4 1 = 1.07 \qquad \frac{\partial E_n}{\partial w_{30}} = \delta_3 1 = -0.05$$

$$\frac{\partial E_n}{\partial w_{41}} = \delta_4 z_1 = 1.07 \times 2 = 2.14 \qquad \frac{\partial E_n}{\partial w_{31}} = \delta_3 z_1 = -0.05 \times 2 = -0.10$$

$$\frac{\partial E_n}{\partial w_{42}} = \delta_4 z_2 = 1.07 \times 3 = 3.21 \qquad \frac{\partial E_n}{\partial w_{32}} = \delta_3 z_2 = -0.05 \times 3 = -0.15$$

$$\frac{\partial E_n}{\partial w_{43}} = \delta_4 z_3 = 1.07 \times 0.67 = 0.72$$

Weight Update

With learning rate $\eta = 0.1$, we have

$$w_{42}^{(\tau+1)} = w_{42}^{(\tau)} - \eta \times \frac{\partial E_n}{\partial w_{42}} = 0.3 - 0.1 \times 3.21 = -0.02$$

and

$$w_{43}^{(\tau+1)} = w_{43}^{(\tau)} - \eta \times \frac{\partial E_n}{\partial w_{43}} = -0.2 - 0.1 \times 0.72 = -0.27$$

Other weight updates are computed analogously.

Error function for binary classes

Cross-entropy error function:

$$E(\mathbf{w}) = -\sum_{n=1}^{N} t_n \ln y_n + (1 - t_n) \ln(1 - y_n)$$

with $y_n = y(\mathbf{x}_n, \mathbf{w})$.
We only consider a single pattern, so

$$E_n = -t \ln y - (1 - t) \ln(1 - y),$$

where for simplicity we have dropped the subscript n on the right-hand side.

Local error

Local error for logistic output with entropy error:

$$\delta_o = \sigma'(a_o)\frac{\partial E}{\partial y} = y(1-y)\frac{\partial E}{\partial y}$$

Left to determine

$$\begin{aligned}\frac{\partial E}{\partial y} &= \frac{\partial[-t\ln y - (1-t)\ln(1-y)]}{\partial y} \\ &= \frac{1-t}{1-y} - \frac{t}{y}\end{aligned}$$

Local error (continued)

Therefore

$$\begin{aligned} y(1-y)\frac{\partial E}{\partial y} &= y(1-y)\left[\frac{1-t}{1-y} - \frac{t}{y}\right] \\ &= (1-t)y - t(1-y) = y - t \end{aligned}$$

Finally, we conclude that

$$\delta_o = y - t$$

for logistic output units and entropy error function.

Multinomial logistic regression

Let $t \in \{1, \ldots, K\}$ (where K is the number of classes). The multinomial logistic regression assumption is

$$p(t = k|\mathbf{x}) = \frac{\exp(\mathbf{w}_k^\top \mathbf{x})}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^\top \mathbf{x})} \qquad \text{(4.104 and 4.105)}$$

where we now have a weight vector $\mathbf{w}_k$ for each class.

Non-binary classes in neural networks

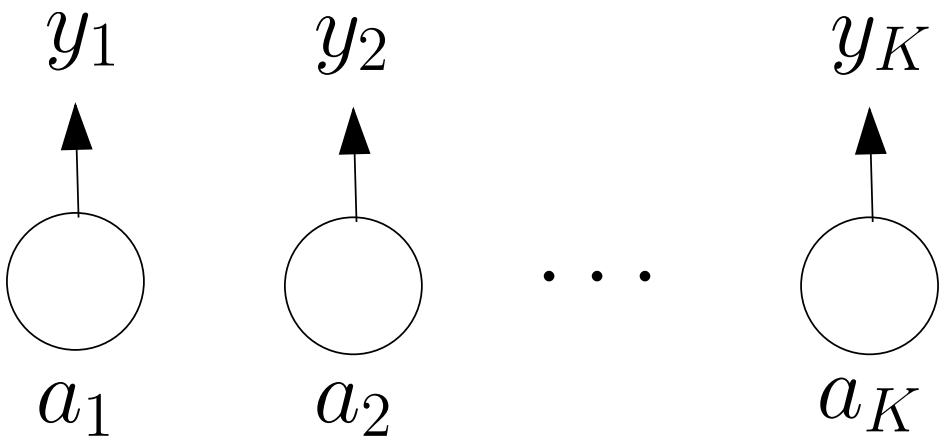
Rather than taking linear functions $a_k = \mathbf{w}_k^\top \mathbf{x}$ we can generalize this model to $a_k(\mathbf{x}, \mathbf{w})$:

$$p(t = k|\mathbf{x}) = \frac{\exp(a_k(\mathbf{x}, \mathbf{w}))}{\sum_{j=1}^{K} \exp(a_j(\mathbf{x}, \mathbf{w}))}$$

In particular, the a_k can be activations produced by a neural network.

Softmax activation function

Implementation in neural network. Create K output units:



With

$$y_k = \frac{\exp(a_k)}{\sum_{j=1}^{K} \exp(a_j)}$$

Let (1-of-K coding)

$$t_j = \begin{cases} 1 \text{ if } t = j \\ 0 \text{ otherwise} \end{cases}$$

Error function for non-binary classes

Negative log-likelihood

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w}) \tag{5.24}$$

For a single observation (pattern) we have

$$\delta_k = \frac{\partial E_n}{\partial a_k} = y_k - t_k$$

(proof omitted)

Regularization: weight decay

Punish large weights by taking

$$\widetilde{E}(\mathbf{w}) = E(\mathbf{w}) + \lambda \sum w_{ij}^2 \tag{5.112}$$

to obtain a "smooth" fit (and thereby avoid overfitting).

Also may help optimization: less local minima and faster convergence.

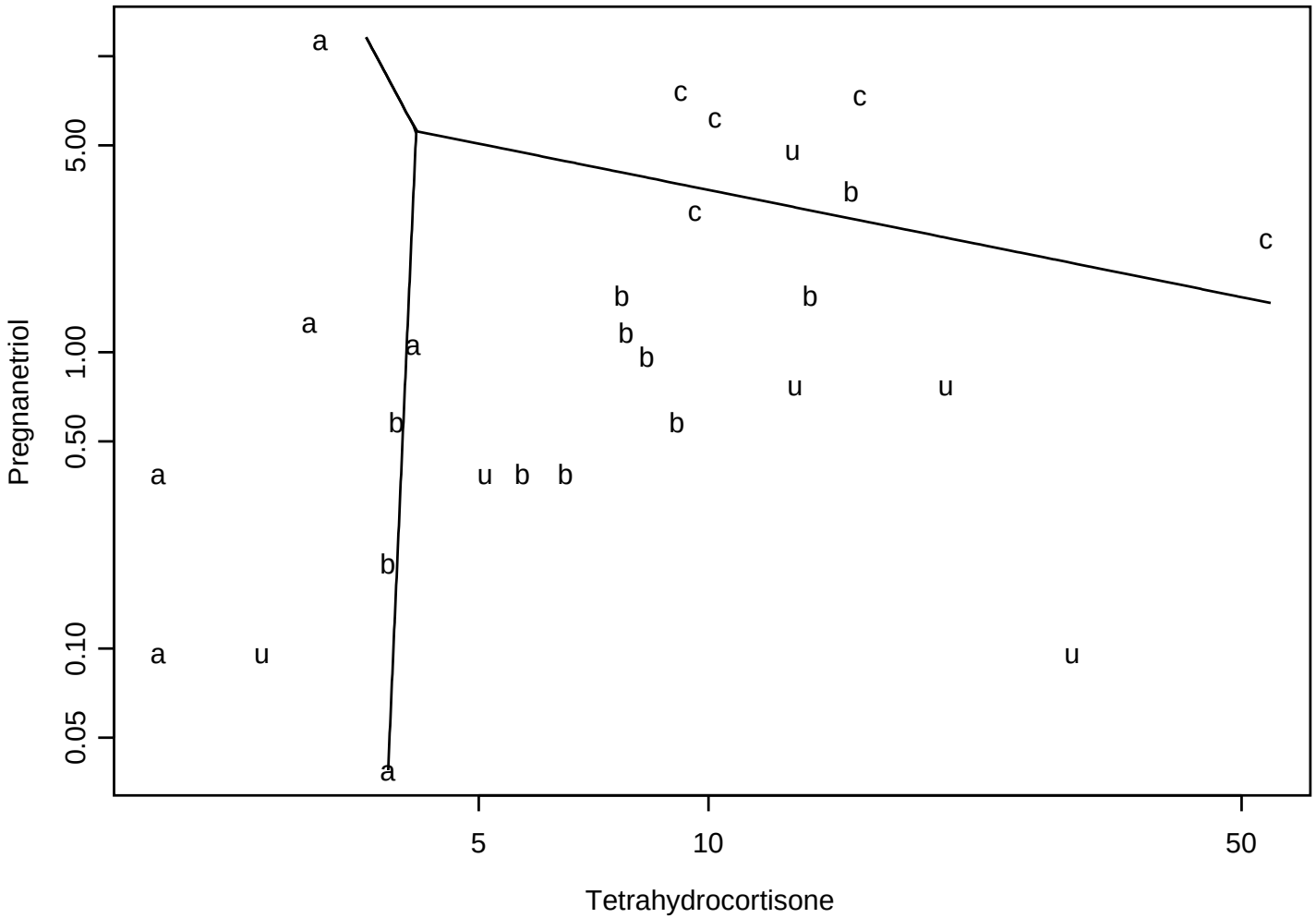
Example: Cushing's Syndrome

Hypertensive disorder associated with over-secretion of cortisol by the adrenal gland. The observations are urinary excretion rates of two steroid metabolites.

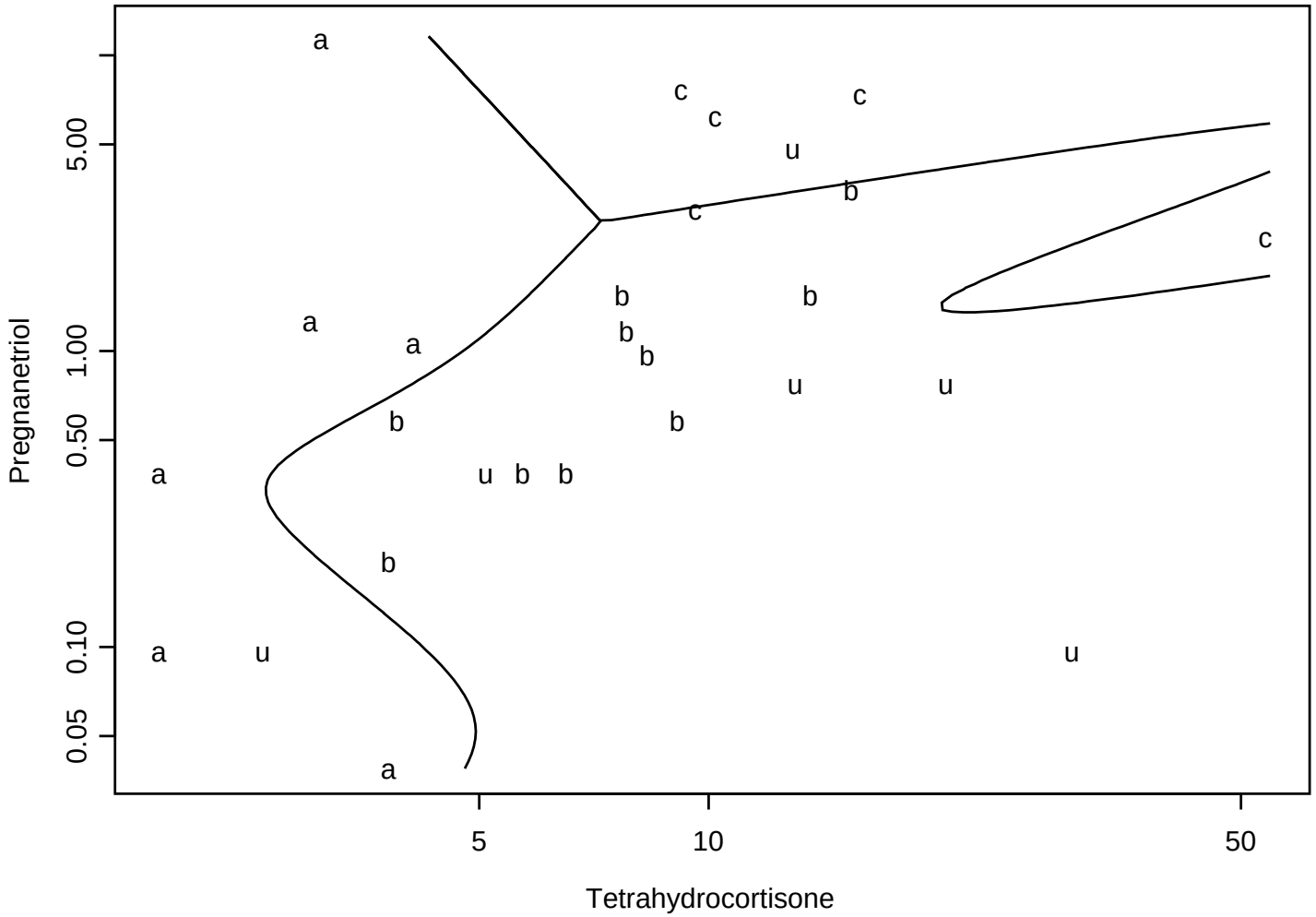
The Cushings data frame (in library MASS) has 27 rows and 3 columns:

- Tetrahydrocortisone: urinary excretion rate (mg/24hr).
- Pregnanetriol: urinary excretion rate (mg/24hr).
- Type: underlying type of syndrome
  - a (adenoma)
  - b (bilateral hyperplasia)
  - c (carcinoma)
  - u for unknown (not used in fitting models)

Cushing's Syndrome: Multinomial Logit



Cushing's Syndrome: NN1 ($\lambda = 0.001$)



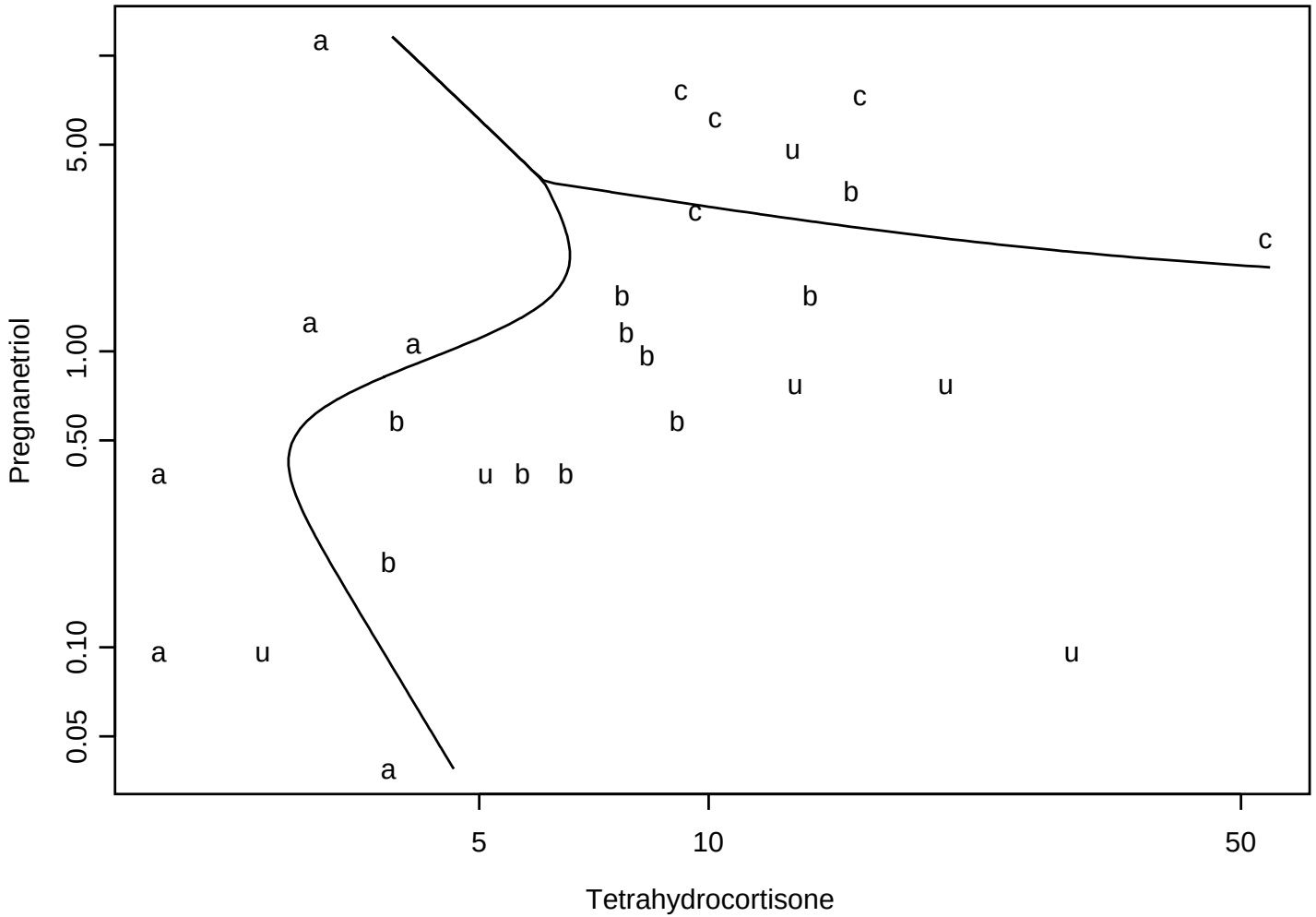
How to in R

```
> library(nnet)
> library(MASS)
> cush <- log(as.matrix(Cushings[,-3]))
> tp <- factor(Cushings$Type[1:21])
> tp

 [1] a a a a a a b b b b b b b b b b c c c c c
Levels: a b c

> tpi <- class.ind(tp)
> tpi[c(1,10,20),]

     a b c
[1,] 1 0 0
[2,] 0 1 0
[3,] 0 0 1

> set.seed(54321)
> cush.nnet1 <- nnet(cush[1:21,], tpi, skip=T, softmax=T, size=2,
decay=0.001, maxit=1000)
```

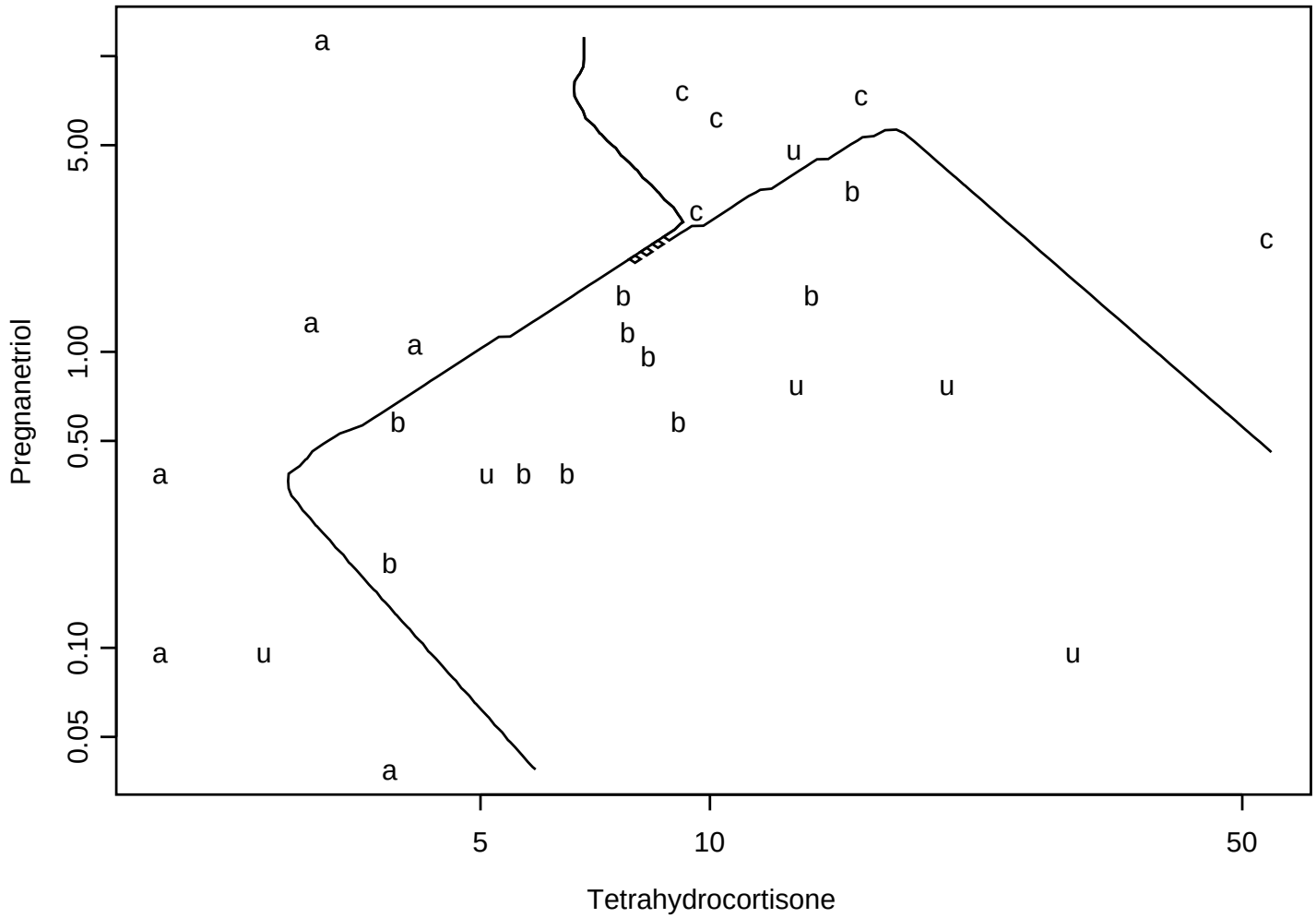
How to in R

```
# weights:  21
initial  value 19.457574
iter  10 value 5.902139
iter  20 value 3.396472
iter  30 value 3.336205
iter  40 value 3.309699
iter  50 value 3.298302
iter  60 value 3.283080
iter  70 value 3.273300
iter  80 value 2.997242
iter  90 value 2.710656
iter 100 value 2.573045
iter 110 value 2.555376
iter 120 value 2.554888
iter 130 value 2.554526
iter 140 value 2.554367
iter 150 value 2.554289
final  value 2.554278
converged
```

Cushing's Syndrome: NN2 ($\lambda = 0.001$)



Cushing's Syndrome: NN3 ($\lambda = 0$)



Model Selection: Forensic Glass

Measurements (refractive index and weight percent of oxides of Na, Mg, Al, Si, K, Ca, Ba, and Fe) on 214 fragments of glass. Glass types

1. Window float glass (70)
2. Window non-float glass (76)
3. Vehicle window glass (17)
4. Containers (13)
5. Tableware (9)
6. Vehicle headlamps (29)

Model Selection: Forensic Glass

Use 10-fold cross validation to select number of hidden units and λ. Use 10 different sets of start weights for each CV-run and average posterior probabilities of fitted networks.

	# hidden units		
λ	2	4	8
0.0001	30.8	23.8	27.1
0.001	30.4	26.2	26.2
0.01	31.8	29.9	29.9

(table displays error rates)