CS 229: Machine Learning

Christian Shelton

UC Riverside

Lecture 8

Slides from Lecture 8

- From UC Riverside
  - CS 229: Machine Learning
  - Professor Christian Shelton
- DO NOT REDISTRIBUTE
  - These slides contain copyrighted material (used with permission) from
    - Elements of Statistical Learning (Hastie, et al.)
    - Pattern Recognition and Machine Learning (Bishop)
    - Machine Learning: A Probabilistic Perspective (Murphy)
  - For use only by enrolled students in the course

Non-linear methods

In

$$f(x) = x^\top w \qquad\qquad f(x) = \sigma(x^\top w)$$

replace x with $\phi(x)$:

$$f(x) = \phi(x)^\top w \qquad\qquad f(x) = \sigma(\phi(x)^\top w)$$

Non-linear methods

In

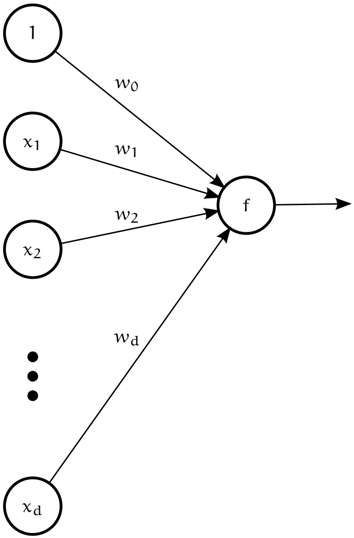
$$f(x) = x^\top w \qquad\qquad f(x) = \sigma(x^\top w)$$

replace x with $\phi(x)$:

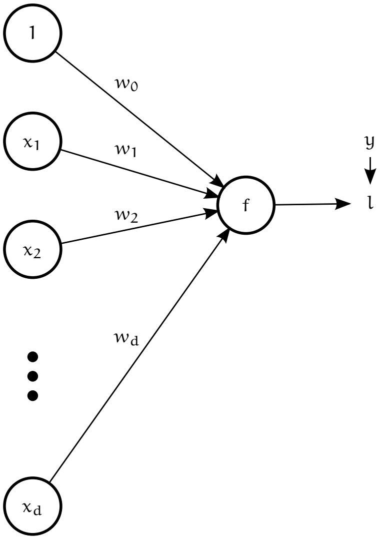
$$f(x) = \phi(x)^\top w \qquad\qquad f(x) = \sigma(\phi(x)^\top w)$$

If $\phi(x)$ selected by hand, we are done!

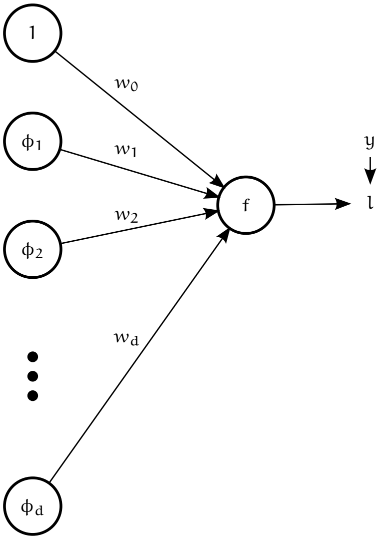
Learning Non-linearity



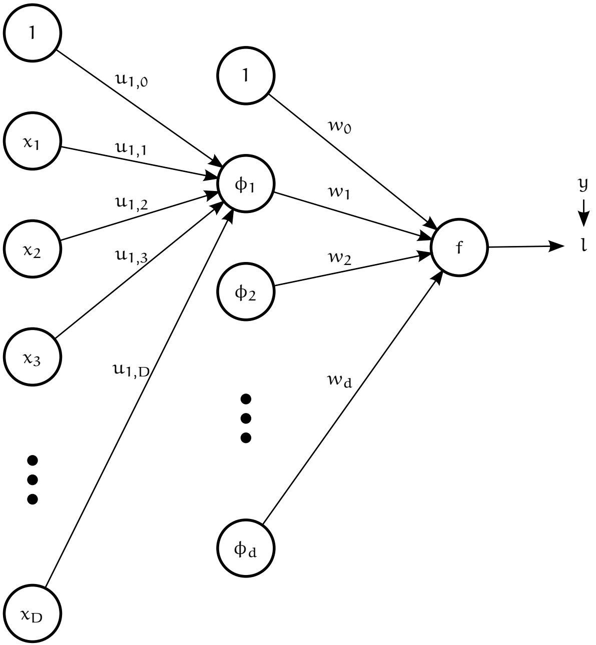
Learning Non-linearity



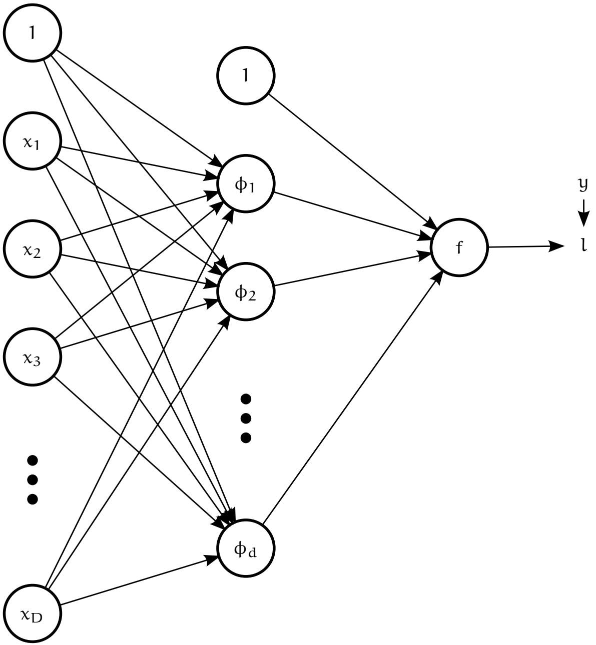
Learning Non-linearity



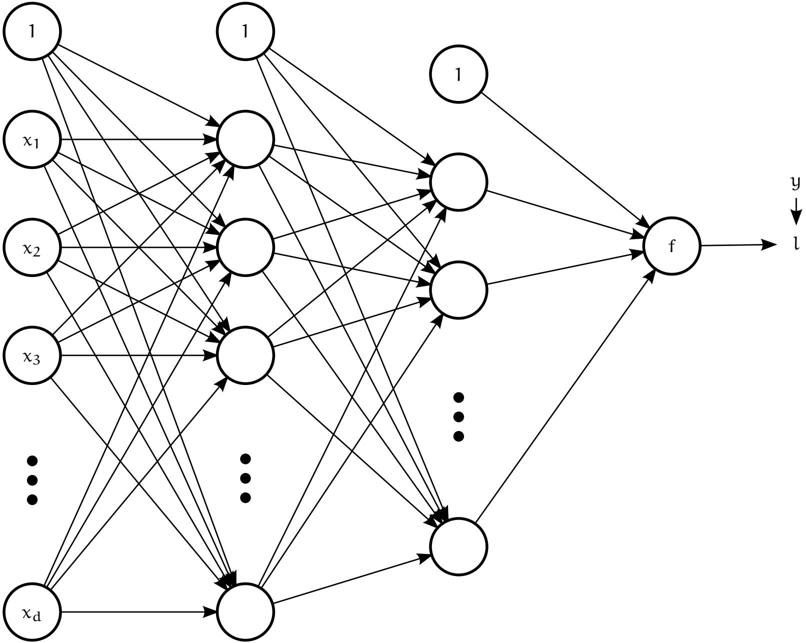
Learning Non-linearity



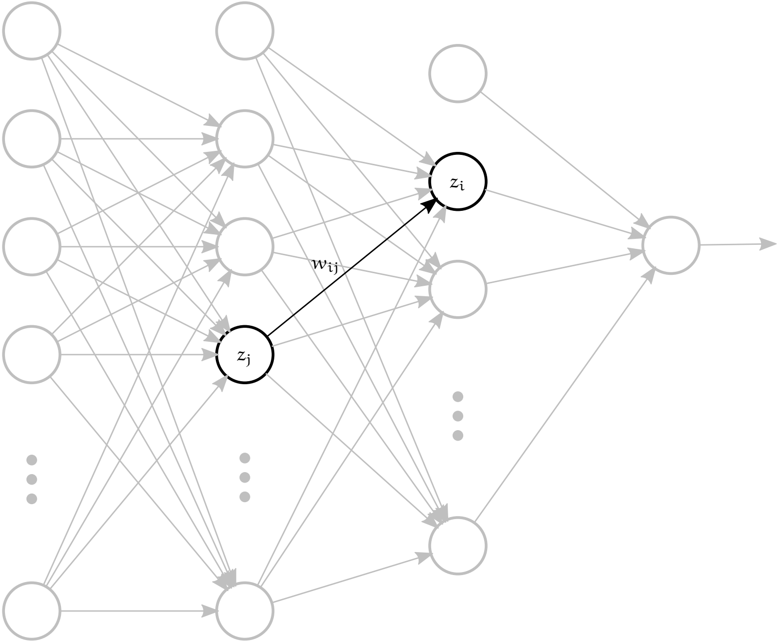
Learning Non-linearity

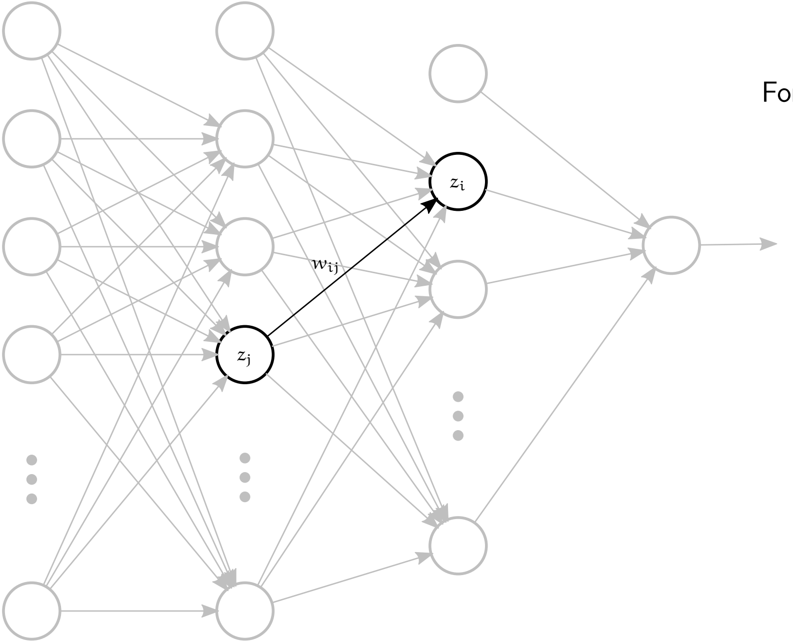


Learning Non-linearity



Learning Non-linearity

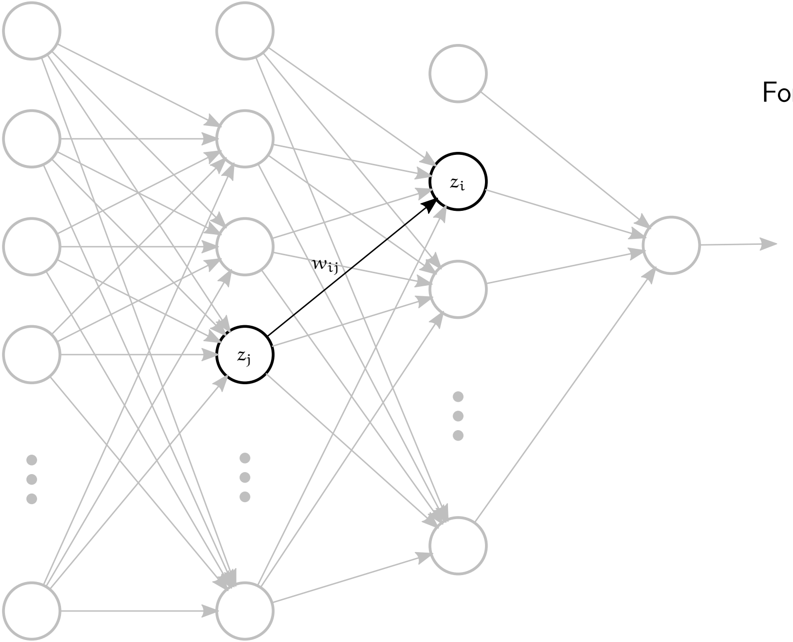




For a given x, y

$$z_i = g_i(a_i)$$
$$a_i = \sum_j w_{ij} z_j$$

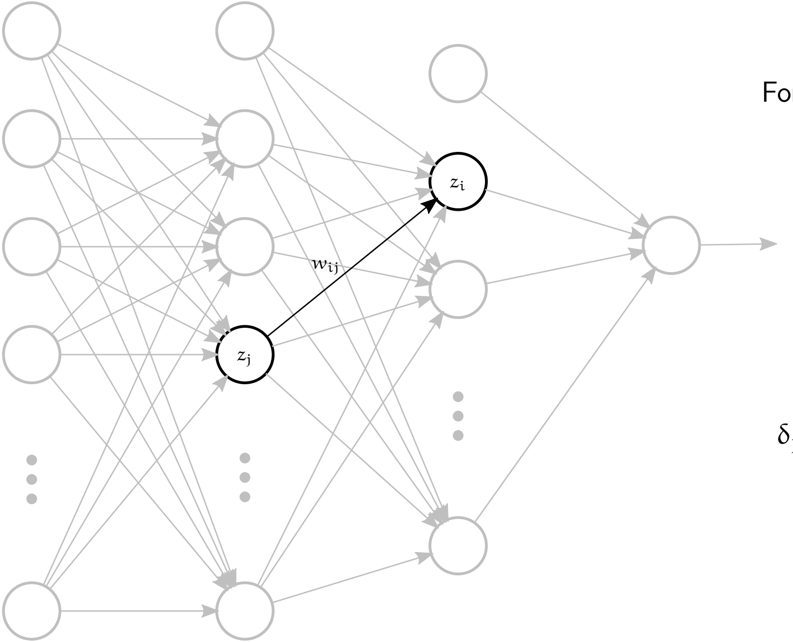
Learning Non-linearity



For a given x, y

$$
\begin{aligned}
z_i &= g_i(a_i) \\
a_i &= \sum_j w_{ij} z_j \\
\frac{\partial l}{\partial w_{ij}} &= \frac{\partial l}{\partial a_i} \frac{\partial a_i}{\partial w_{ij}} \\
&= \delta_i z_j
\end{aligned}
$$

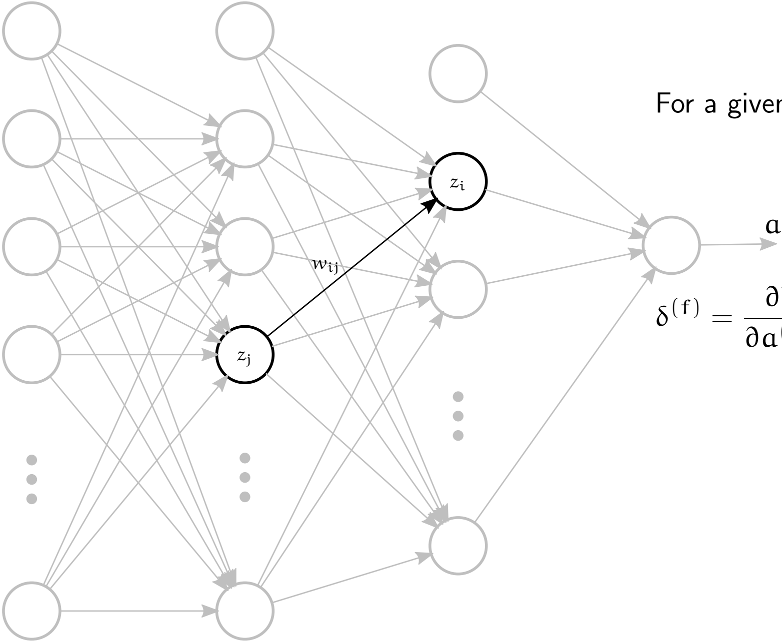
Learning Non-linearity



For a given x, y

$$
\begin{aligned}
z_i &= g_i(a_i) \\
a_i &= \sum_j w_{ij} z_j \\
\frac{\partial l}{\partial w_{ij}} &= \frac{\partial l}{\partial a_i} \frac{\partial a_i}{\partial w_{ij}} \\
&= \delta_i z_j \\
\delta_j = \frac{\partial l}{\partial a_j} &= \frac{\partial l}{\partial z_j} \frac{\partial z_j}{\partial a_j} \\
&= \sum_i \frac{\partial l}{\partial a_i} \frac{\partial a_i}{\partial z_j} \frac{\partial z_j}{\partial a_j} \\
&= \sum_i \delta_i w_{ij} g'(a_j)
\end{aligned}
$$

Learning Non-linearity



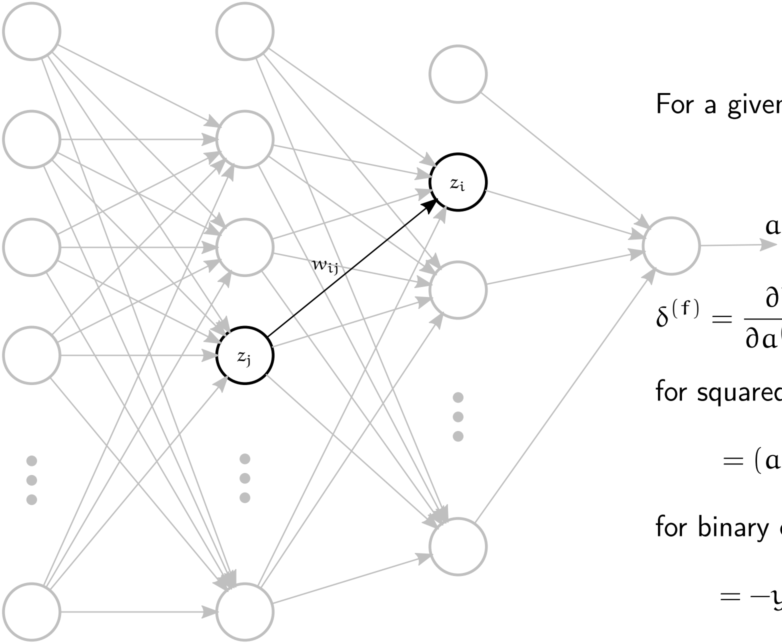
For a given x, y

$$f = g_f(a^{(f)})$$

$$a^{(f)} = \sum_j w_{fj} z_j$$

$$\delta^{(f)} = \frac{\partial l}{\partial a^{(f)}} = \frac{\partial l}{\partial f}\frac{\partial f}{\partial a^{(f)}} = \frac{\partial l}{\partial f} g_f'(a^{(f)})$$

Learning Non-linearity



For a given x, y

$$f = g_f(a^{(f)})$$

$$a^{(f)} = \sum_j w_{fj} z_j$$

$$\delta^{(f)} = \frac{\partial l}{\partial a^{(f)}} = \frac{\partial l}{\partial f}\frac{\partial f}{\partial a^{(f)}} = \frac{\partial l}{\partial f} g_f'(a^{(f)})$$

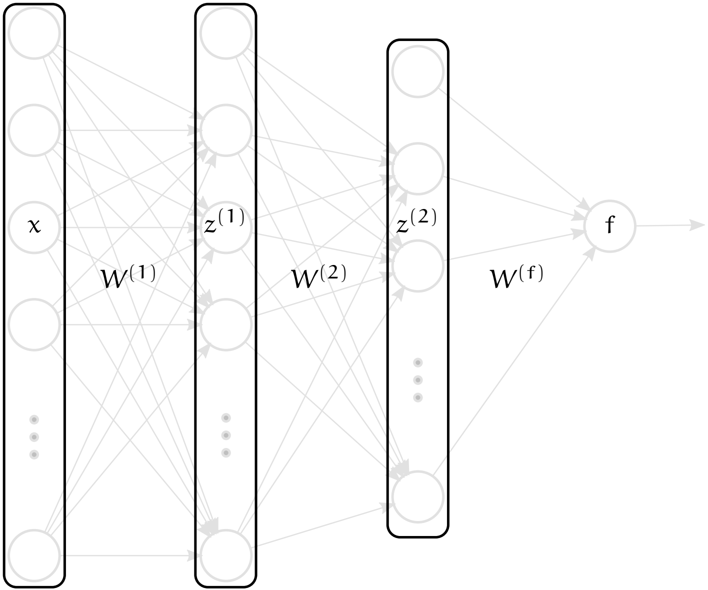
for squared error

$$= (a^{(f)} - y) = (f - y)$$

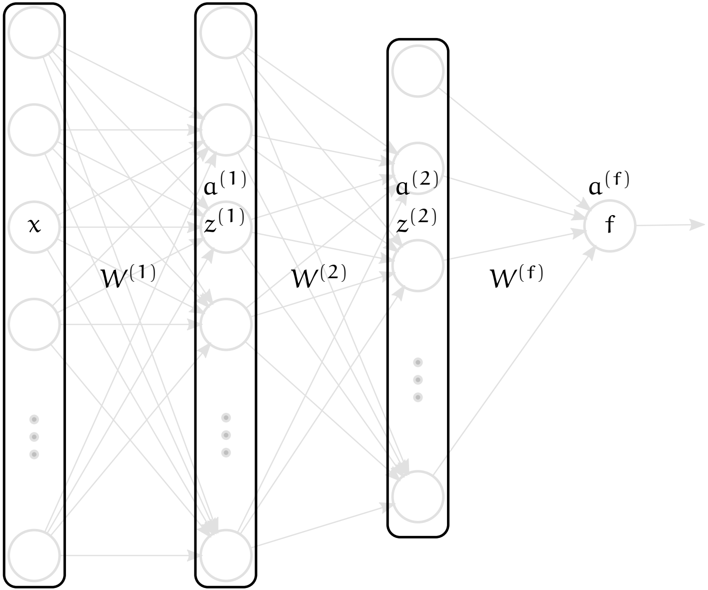
for binary classification

$$= -y(1 - \sigma(ya^{(f)})) = (f - y_{01})$$

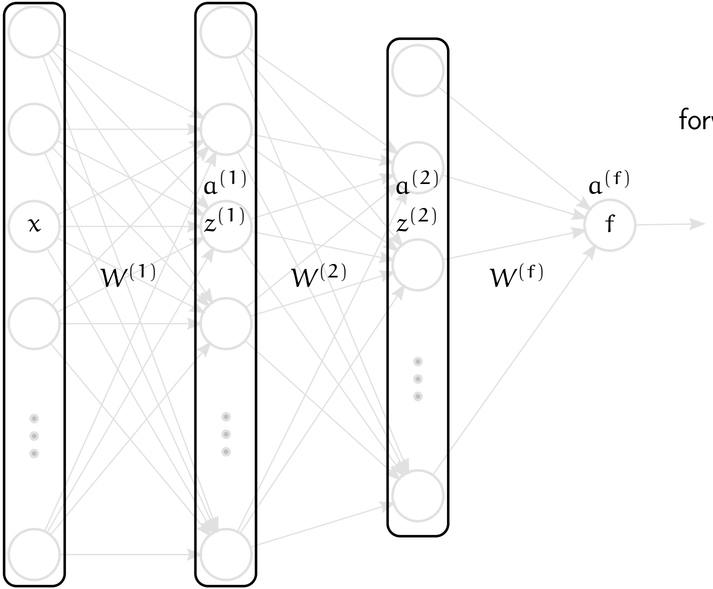
Layered Neural Network



Layered Neural Network



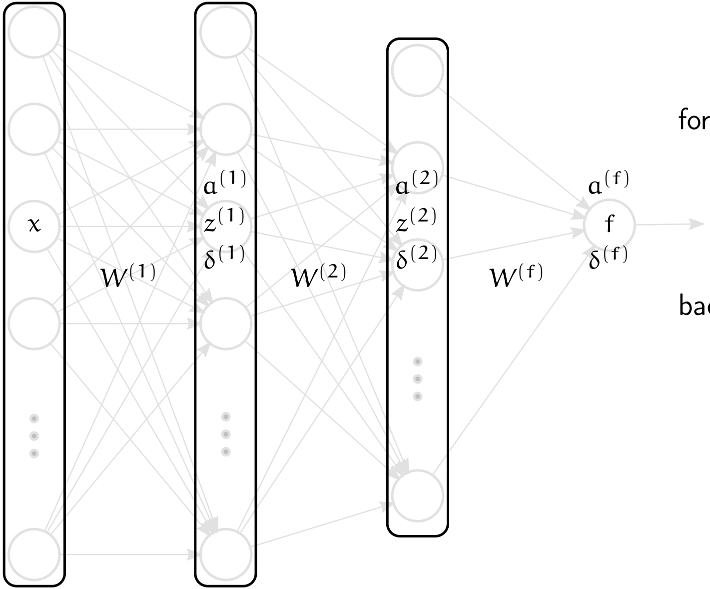
Layered Neural Network



forward:

$$a^{(i)} = W^{(i)} z^{(i-1)}$$
$$z^{(i)} = g(a^{(i)})$$

Layered Neural Network



forward:

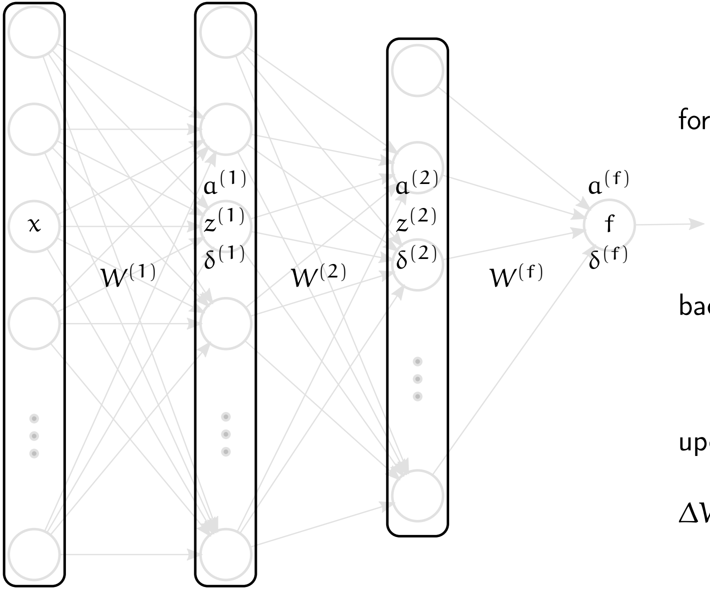
$$a^{(i)} = W^{(i)} z^{(i-1)}$$

$$z^{(i)} = g(a^{(i)})$$

backward:

$$\delta^{(i)} = g'(a^{(i)}) \odot {W^{(i+1)}}^\top \delta^{(i+1)}$$

Layered Neural Network



forward:

$$a^{(i)} = W^{(i)} z^{(i-1)}$$
$$z^{(i)} = g(a^{(i)})$$

backward:

$$\delta^{(i)} = g'(a^{(i)}) \odot {W^{(i+1)}}^\top \delta^{(i+1)}$$

update:

$$\Delta W^{(i)} = -\eta \delta^{(i)} {z^{(i-1)}}^\top$$

Output Layer

For regression:

$$g_f(a) = a$$

$$\delta^{(f)} = (f - y)$$

$$l(f, y) = \frac{1}{2}(f - y)^2$$

Output Layer

For regression:

$$g_f(a) = a \qquad l(f, y) = \frac{1}{2}(f - y)^2$$

$$\delta^{(f)} = (f - y)$$

For binary classification, $y \in \{-1, +1\}$:

$$g_f(a) = \sigma(a) \qquad l(f, y) = \log(1 + e^{-yf})$$

$$\delta^{(f)} = -y(1 - \sigma(ya^{(f)}))$$

Output Layer

For regression:

$$g_f(a) = a \qquad\qquad l(f,y) = \frac{1}{2}(f-y)^2$$

$$\delta^{(f)} = (f-y)$$

For binary classification, $y \in \{-1,+1\}$:

$$g_f(a) = \sigma(a) \qquad\qquad l(f,y) = \log(1+e^{-yf})$$

$$\delta^{(f)} = -y(1-\sigma(ya^{(f)}))$$

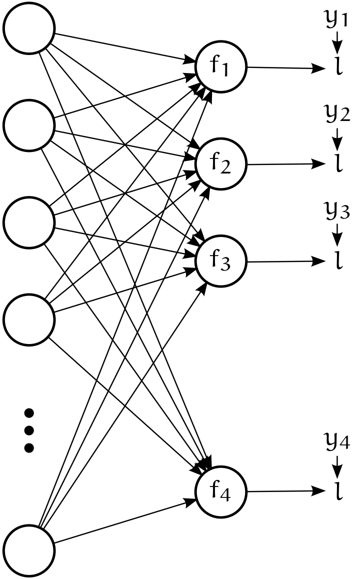
For binary classification, $y \in \{0,1\}$:

$$g_f(a) = \sigma(a) \qquad\qquad l(f,y) = -(y\log(f) + (1-y)\log(1-f))$$

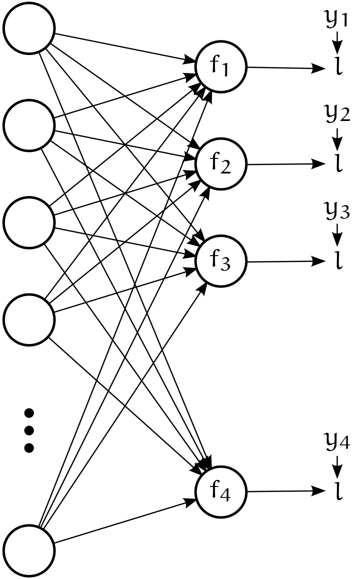
$$\delta^{(f)} = (f-y)$$

Output Layer

For multiclass classification, $y_c \in \{0, 1\}$:



Output Layer



For multiclass classification, $y_c \in \{0, 1\}$:

$$g_{f,c}(a^{(f)}) = \frac{e^{a_c^{(f)}}}{\sum_{c'} e^{a_{c'}^{(f)}}} \qquad l(f, y) = -\sum_c y_c \log(f_c)$$

$$\delta^{(f_c)} = (f_c - y_c)$$

Non-linearities

Most common:

$$g(a) = \sigma(a) = \tanh(2a)/2 + 1$$

$$g(a) = \tanh(a)$$

$$g(a) = \max(0, a)$$

$$g(a) = \max(0, 0.99a) + 0.01a$$

Non-linearities

Most common:

$$
\begin{aligned}
g(a) &= \sigma(a) = \tanh(2a)/2 + 1 \\
g(a) &= \tanh(a) \\
g(a) &= \max(0, a) \\
g(a) &= \max(0, 0.99a) + 0.01a
\end{aligned}
$$

Also used:

$$z_i = e^{-\sum_j (z_j - w_{ij})^2}$$

(radial basis function network)

Hidden Units

How many layers?

Hidden Units

How many layers?

- Historically, 2 (1 layer of hidden units)
- 2 can approximate any function (with enough hidden units)

Hidden Units

How many layers?

- Historically, 2 (1 layer of hidden units)
- 2 can approximate any function (with enough hidden units)
- More cause problems with minimization

Hidden Units

How many layers?

- Historically, 2 (1 layer of hidden units)
- 2 can approximate any function (with enough hidden units)
- More cause problems with minimization
- Now common to have many (deep learning)

Hidden Units

How many layers?

- Historically, 2 (1 layer of hidden units)
- 2 can approximate any function (with enough hidden units)
- More cause problems with minimization
- Now common to have many (deep learning)

How many units?

- Too many can cause overfitting...
- ...see next slide

Hidden Units

How many layers?

- Historically, 2 (1 layer of hidden units)
- 2 can approximate any function (with enough hidden units)
- More cause problems with minimization
- Now common to have many (deep learning)

How many units?

- Too many can cause overfitting...
- ...see next slide
- Usually err on side of too many

Overfitting

Start (stochastic) gradient descent:

- With weights near 0
- But, random!

Overfitting

Start (stochastic) gradient descent:

- With weights near 0
- But, random!

Add regularization to loss:

$$L = \sum_i l(f(x_i), y_i) + \frac{\lambda}{2} \sum_{ij} w_{ij}^2$$

Overfitting

Start (stochastic) gradient descent:

- With weights near 0
- But, random!

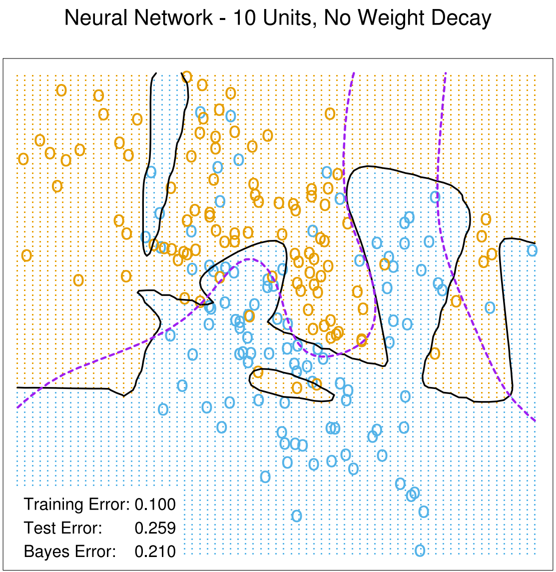
Add regularization to loss:

$$L = \sum_i l(f(x_i), y_i) + \frac{\lambda}{2} \sum_{ij} w_{ij}^2$$

Same as adding $-\eta\lambda w_{ij}$ to batch update of w_{ij}
(or $-\frac{1}{n}\eta\lambda w_{ij}$ to online update)

Called "weight decay"

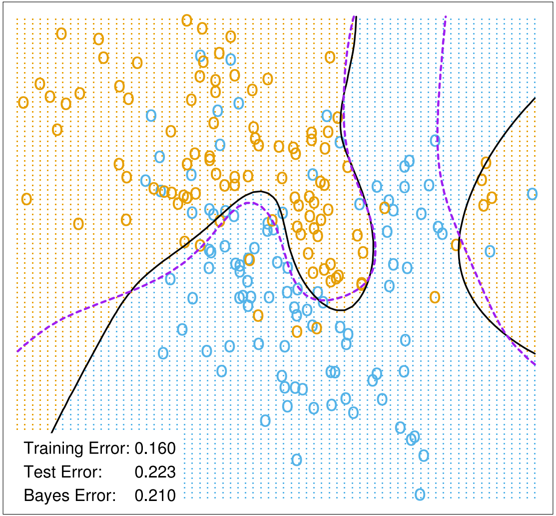
Example



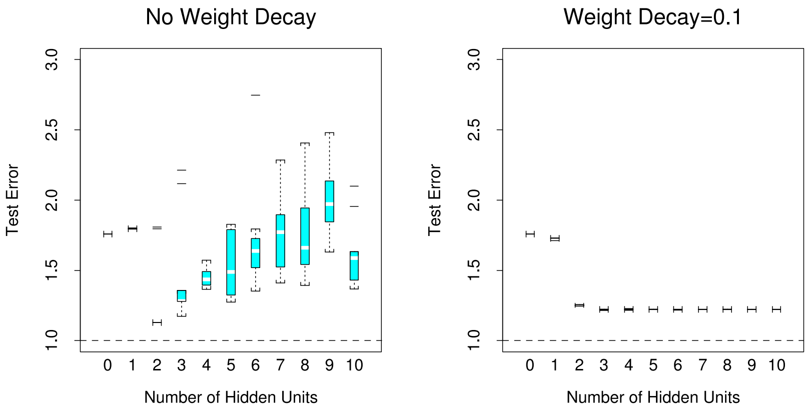
Neural Network - 10 Units, No Weight Decay
Training Error: 0.100
Test Error: 0.259
Bayes Error: 0.210

Example

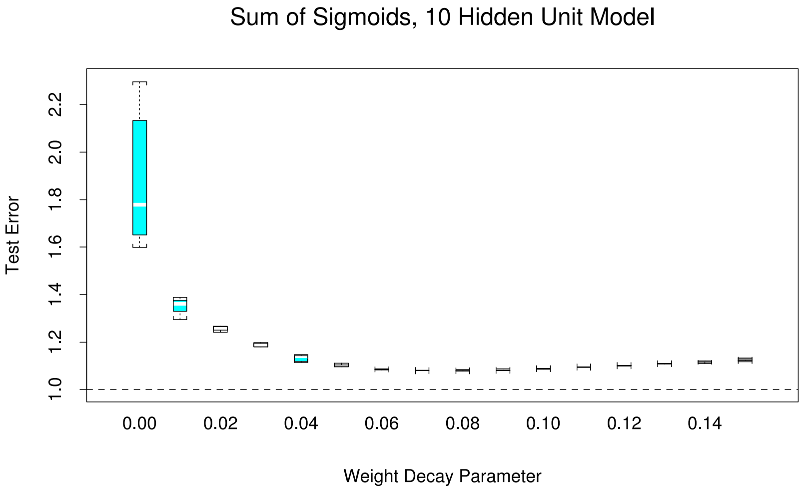
Neural Network - 10 Units, Weight Decay=0.02



Sum of 2 Sigmoids



Sum of 2 Sigmoids



Optimization

Either batch (gradient descent) or online (stochastic gradient descent) used.

Optimization

Either batch (gradient descent) or online (stochastic gradient descent) used.

2^{nd}-order methods possible, but expensive

Conjungate-gradient methods (and similar) more popular

Optimization

Either batch (gradient descent) or online (stochastic gradient descent) used.

2^{nd}-order methods possible, but expensive

Conjungate-gradient methods (and similar) more popular

Many many local minima...

Optimization

Either batch (gradient descent) or online (stochastic gradient descent) used.

2^{nd}-order methods possible, but expensive

Conjungate-gradient methods (and similar) more popular

Many many local minima... use random restart

Optimization

Either batch (gradient descent) or online (stochastic gradient descent) used.

2$^{\text{nd}}$-order methods possible, but expensive

Conjungate-gradient methods (and similar) more popular

Many many local minima... use random restart

Normalize data

Weight Tieing

If we want two weights to be the same...

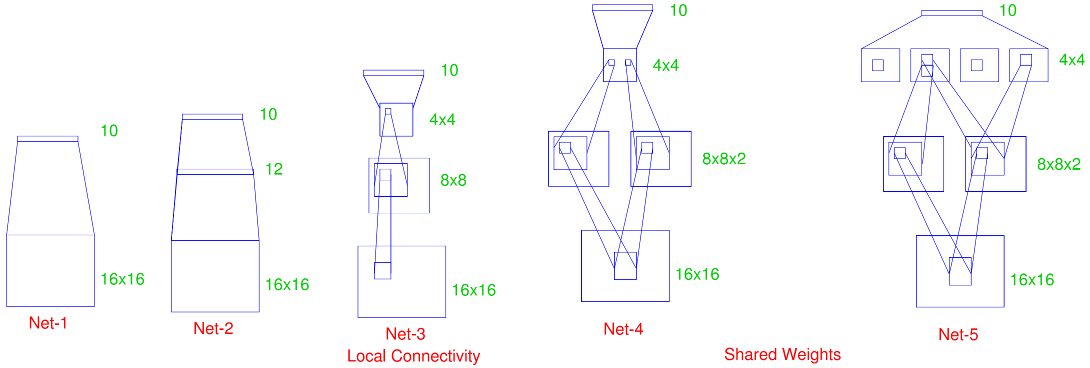
Weight Tieing

If we want two weights to be the same...

- Start them the same
- Sum their updates and apply to both:

$$\begin{aligned} g(a,b) &= \ldots \\ f(x) &= g(x,x) \\ \frac{\partial f}{\partial x} &= \frac{\partial g}{\partial a}\frac{\partial a}{\partial x} + \frac{\partial g}{\partial b}\frac{\partial b}{\partial x} \\ \frac{\partial f}{\partial x} &= \frac{\partial g}{\partial a} + \frac{\partial g}{\partial b} \end{aligned}$$

Digit Recognizer Networks



Digit Recognizer Networks