Neural Networks

Jong-Han Kim

EE787 Machine learning
Kyung Hee University

Features

- neural networks can be thought of as a way to form features that works directly from the data (as opposed to hand-engineered features)
- the resulting features are often useful for multiple regression/classification tasks
- they often require a lot of data

Features

- so far we have considered predictors which depend *linearly* on θ

$$\hat{y} = g(x) = \theta^\mathsf{T} x$$

  called a *linear model*

- if we believe v and u are not related linearly, we add *features*, *e.g.*,

$$x = \phi(u) = (1, u, u^2, u^3, \ldots, u^{d-1})$$

- this gives a better fit, *i.e.*, reduces the training loss

- we do not get a better fit using linear features, *e.g.*,

$$x = \phi(u) = (1, u_1, u_2, u_1 + u_2)$$

Features

- a useful class of features consists of a nonlinear function $h : \mathbf{R} \to \mathbf{R}$ composed with a linear function

$$\phi(u) = h(w_1 + w_2 u_1 + \cdots + w_{d+1} u_d)$$

- h must be nonlinear; if h is linear, then this does not improve the fit
- common choices are $h(x) = (x)_+$ or $h(x) = \log(1 + e^x)$
- coefficients $w_1, \ldots, w_{d+1}$ are called *weights*
- one possibility: add features by *randomly* choosing weights

Neurons

- a *neuron* is a feature map of the form

$$\phi(u) = h(w_1 + w_2 u_1 + \cdots + w_{d+1} u_d)$$

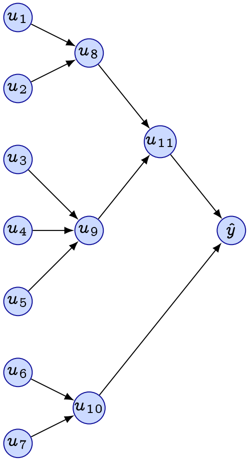
- the function h is called the *activation function*
- common choices of activation function:
  - sigmoid: $h(u) = 1/(1 + e^{-u})$
  - tanh: $h(u) = \mathtt{tanh}(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$
  - hinge or relu: $h(u) = \max(u, 0)$
- any nonlinear function can be used

Composing features

- we can compose features, *e.g.*,

$$\begin{aligned} u_8 &= \phi_1(u_1, u_2) \\ u_9 &= \phi_2(u_3, u_4, u_5) \\ u_{10} &= \phi_3(u_6, u_7) \\ u_{11} &= \phi_4(u_8, u_9) \end{aligned}$$

- predictor is $\hat{y} = \theta_1 + \theta_2 u_{11} + \theta_3 u_{10}$
- the composition defines a graph
- each node corresponds to a feature variable
- left-most nodes, called *input nodes*, correspond to raw data records

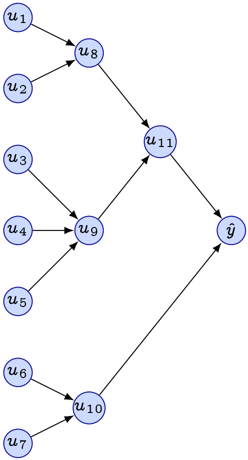


Neural networks

- feature maps

$$
\begin{aligned}
u_8 &= \phi_1(u_1, u_2) \\
u_9 &= \phi_2(u_3, u_4, u_5) \\
u_{10} &= \phi_3(u_6, u_7) \\
u_{11} &= \phi_3(u_8, u_9)
\end{aligned}
$$

- in a linear model, choose θ to minimize regularized loss
- in a *neural network*
  - each feature map is a neuron
  - we minimize over θ and all weights w_{ij}

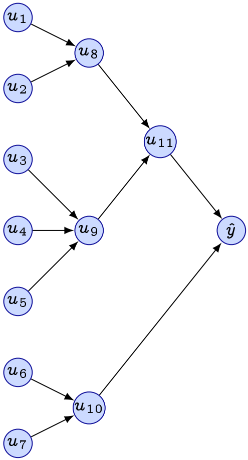


Neural networks

- in a neural network, we optimize over both θ and the weights w_{ij}
- by optimizing w_{ij} we are selecting features
- the resulting features are often useful for many problems
- called *pre-trained* neural networks
- pre-training chooses weights w_{ij} by extensive training on a large amount of data
- resulting neurons are used as features for ERM
- often applications only choose the *output weights* θ

Terminology

- such networks are sometimes called *multi-layer perceptrons* or *feedforward neural networks*
- other types are *recurrent neural networks* and *convolutional neural networks*
- $\hat{y}$ is called the *output node*
- left-most nodes are called the *input nodes*
- other nodes are called *hidden layers*



Optimization

- use optimization to choose weights θ and w_{ij}
- gradient method (and variants) are widely used
- since the predictor is not linear in the weights w_{ij}, convexity of the loss function does not help

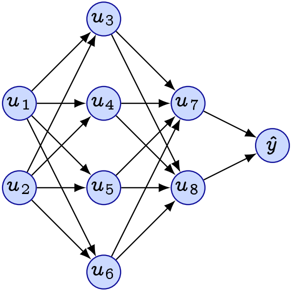
Computing gradients

- apply chain rule to differentiate composite functions
- called *back propagation*
- simpler alternative: *automatic differentiation*
- distinct from *numerical differentiation*, which computes approximate derivatives via
$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$
- automatic differentiation
  - implemented either symbolically or by operator overloading
  - returns exact derivatives (when activation functions are differentiable)

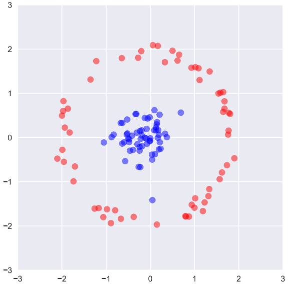
Computing derivatives

```
import Base: *,+,exp

struct Var
    x
    dx
end

*(a::Var, b::Var) = Var(a.x*b.x,  b.x*a.dx + a.x*b.dx)
*(a::Number, b::Var) =  Var(a*b.x,  a*b.dx)
*(a::Var, b::Number) = b*a
+(a::Var, b::Var) =  Var(a.x+b.x,  a.dx + b.dx)
exp(a::Var) = Var(exp(a.x), exp(a.x)*a.dx)

f(a) = a*exp(a^3 + 7*a)   # define function f
x = 2                     # evaluate derivative at x=2
xvar = Var(x,1)
dfdx = f(xvar).dx         # returns derivative
```

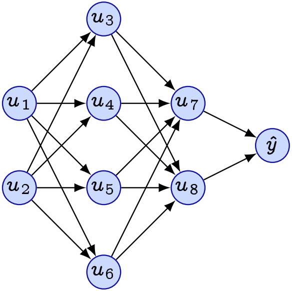
Example: classification



- logistic loss $l(\hat{y}, y) = \log(1 + e^{-y\hat{y}})$
- 2 hidden layers
- sigmoid activation $h(u) = 1/(1 + e^{-x})$
- weights $w \in \mathbf{R}^{22}$ and $\theta \in \mathbf{R}^3$



Example: classification

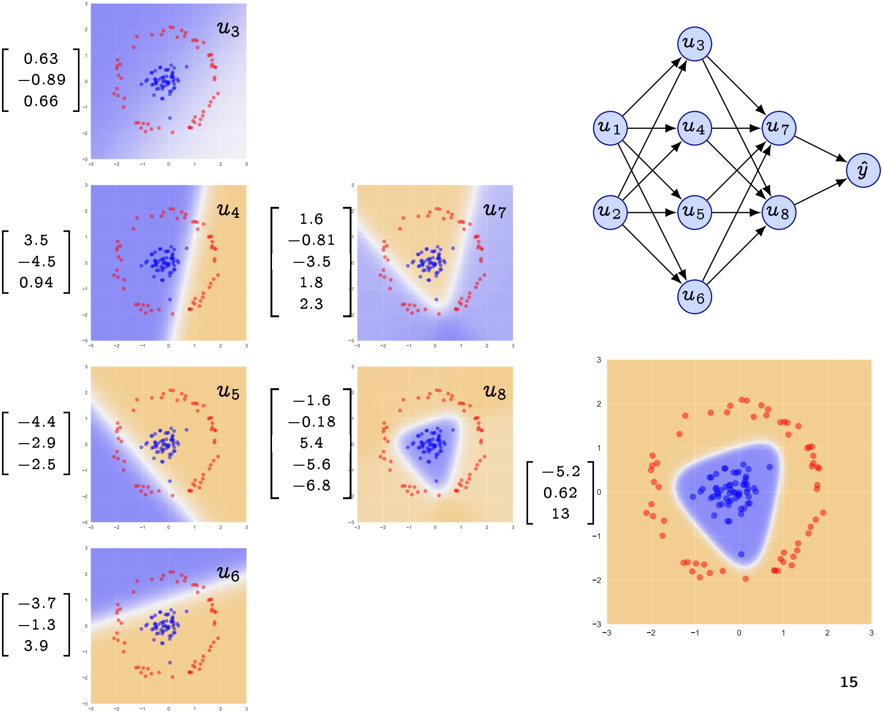


- the predictor is

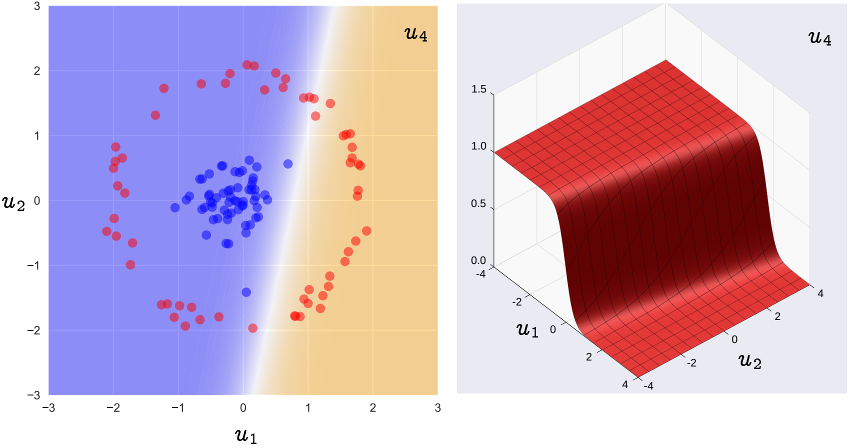
$$
\begin{aligned}
u_3 &= h(w_1 + w_2 u_1 + w_3 u_2) \\
u_4 &= h(w_4 + w_5 u_1 + w_6 u_2) \\
u_5 &= h(w_7 + w_8 u_1 + w_9 u_2) \\
u_6 &= h(w_{10} + w_{11} u_1 + w_{12} u_2) \\
u_7 &= h(w_{13} + w_{14} u_3 + w_{15} u_4 + w_{16} u_5 + w_{17} u_6) \\
u_8 &= h(w_{18} + w_{19} u_3 + w_{20} u_4 + w_{21} u_5 + w_{22} u_6) \\
\hat{y} &= \theta_1 + \theta_2 u_7 + \theta_3 u_8
\end{aligned}
$$

- we choose θ, w to minimize

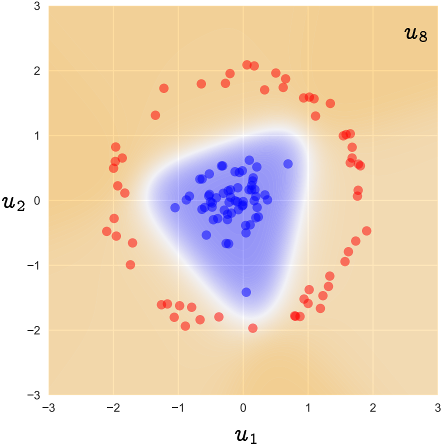
$$\frac{1}{n} \sum_{i=1}^{n} l(\hat{y}^i, y^i) + \lambda ||\theta||^2 + \mu ||w||^2$$

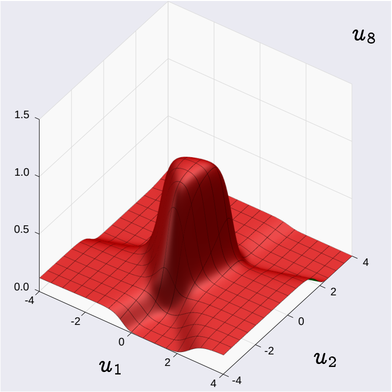

u_3
$\begin{bmatrix} 0.63 \\ -0.89 \\ 0.66 \end{bmatrix}$
u_4
$\begin{bmatrix} 3.5 \\ -4.5 \\ 0.94 \end{bmatrix}$
u_5
$\begin{bmatrix} -4.4 \\ -2.9 \\ -2.5 \end{bmatrix}$
u_6
$\begin{bmatrix} -3.7 \\ -1.3 \\ 3.9 \end{bmatrix}$
u_7
$\begin{bmatrix} 1.6 \\ -0.81 \\ -3.5 \\ 1.8 \\ 2.3 \end{bmatrix}$
u_8
$\begin{bmatrix} -1.6 \\ -0.18 \\ 5.4 \\ -5.6 \\ -6.8 \end{bmatrix}$
u_1
u_2
$\hat{y}$
$\begin{bmatrix} -5.2 \\ 0.62 \\ 13 \end{bmatrix}$

Neurons

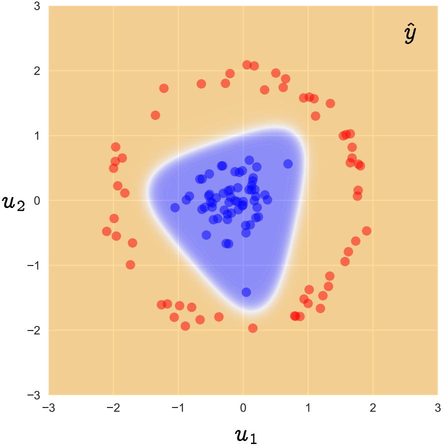


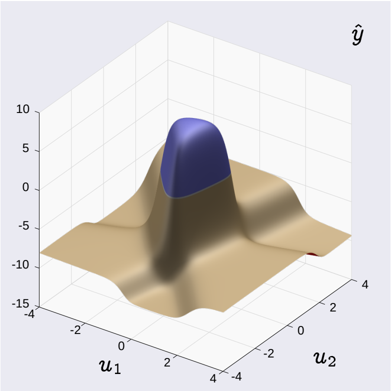
Neurons





Predictor





Example: classification

- plots above show approximate convergence to a local minimum after 250 iterations
- can subsequently use only the important neurons, *i.e.*, remove neurons for which corresponding coefficients are small and solve again