Neural Networks

Sanjay Lall and Stephen Boyd

EE104
Stanford University

Neural networks

- a neural network (NN) is a nonlinear predictor $\hat{y} = g_\theta(x)$ with a particular layered form
- NNs can be thought of as incorporating aspects of feature engineering into the predictor (and indeed are often used as 'automatic feature engineering')
- the parameter dimension p can be very large
- training NNs can be tricky, and take a long time
- NNs can perform very well, especially when there's lots of training data
- it is very hard to interpret a NN predictor $\hat{y} = g_\theta(x)$ (*cf.* a linear predictor $\hat{y} = \theta^\mathsf{T} x$)

Nomenclature

Neural network layers

- a (feedforward) *neural network* predictor consists of a composition of functions

$$\hat{y} = g^3(g^2(g^1(x)))$$

  (we show three here, but we can have any number)

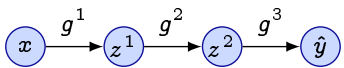
- written as $g = g^3 \circ g^2 \circ g^1$ (the symbol $\circ$ means function composition)
- each g^i is called a *layer*; here we have 3 layers
- sometimes called a *multi-layer perceptron*

Neural network layers

- we can write the predictor $\hat{y} = g^3(g^2(g^1(x)))$ as

$$z^1 = g^1(x), \qquad z^2 = g^2(z^1), \qquad \hat{y} = g^3(z^2)$$

- the vector $z^i \in \mathbf{R}^{d^i}$ is called the *activation* or *output* of layer i
- layer output dimensions d_i need not be the same
- we sometimes write $z^0 = x$, $d^0 = d$, and $z^3 = \hat{y}$, $d^3 = m$
  (so the predictor input x and predictor output y are also considered activations of layers)
- sometimes visualized as flow graph



Layer functions

- each layer g^i is a composition of a function h with an affine function

$$g^i(z^{i-1}) = h\big(\theta_i^T(1, z^{i-1})\big), \quad i = 1, \ldots, m$$

- the matrix $\theta_i \in \mathbf{R}^{(d_{i-1}+1)\times d_i}$ is the *parameter* for layer i
- an M-layer neural network predictor is parameterized by $\theta = (\theta_1, \ldots, \theta_M)$ (for M layers)
- the function $h : \mathbf{R} \to \mathbf{R}$ is a scalar *activation function*, which acts elementwise on a vector argument (*i.e.*, it is applied to each entry of a vector)

Weights and biases

- we can write the entries of the parameter matrix θ_i as

$$\theta_i = \begin{bmatrix} \alpha^i \\ \beta^i \end{bmatrix} = \begin{bmatrix} \alpha_1^i & \cdots & \alpha_{d_i}^i \\ \beta_1^i & \cdots & \beta_{d_i}^i \end{bmatrix}$$

  where $\alpha_j^i \in \mathbf{R}$ and $\beta_j^i \in \mathbf{R}^{d_{i-1}}$, so

$$\theta_i^{\mathsf{T}}(1, z^{i-1}) = \theta_i^{\mathsf{T}} \begin{bmatrix} 1 \\ z^{i-1} \end{bmatrix} = \begin{bmatrix} \alpha_1^i \\ \vdots \\ \alpha_{d_i}^i \end{bmatrix} + \begin{bmatrix} \beta_1^{i\mathsf{T}} z^{i-1} \\ \vdots \\ \beta_{d_i}^{i\mathsf{T}} z^{i-1} \end{bmatrix}$$

- and the layer map $z^i = g^i(z^{i-1})$ means

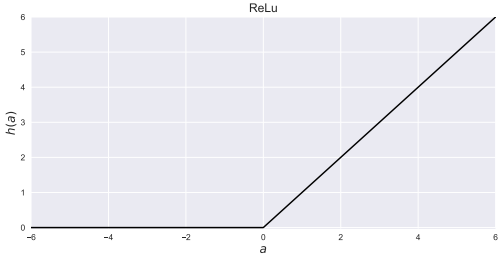
$$z_j^i = h(\alpha_j^i + \beta_j^{i\mathsf{T}} z^{i-1}), \quad j = 1, \ldots, d_i$$

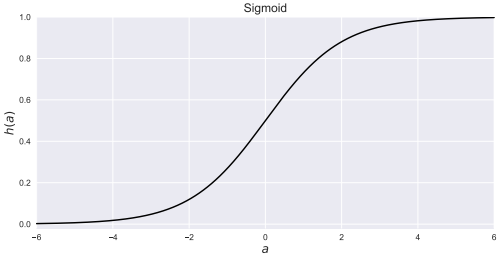
  a composition of h with an affine function

- such a function is called a *neuron*
- α_j^i is called the *bias* of the neuron; β_j^i are its (input) *weights*

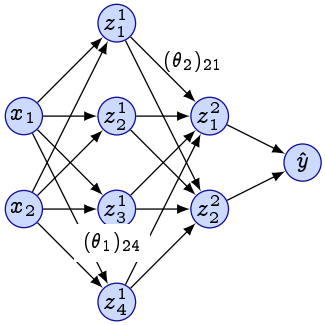
Activation functions

- the activation function $h : \mathbf{R} \to \mathbf{R}$ is nonlinear
- (if it were linear or affine, then g_θ would be an affine function of x, *i.e.*, a linear predictor)
- common activation functions include
  - $h(a) = (a)_+ = \max(a, 0)$, called *ReLu* (rectified linear unit)
  - $h(a) = e^a/(1 + e^a)$, called *sigmoid function*





Network depiction



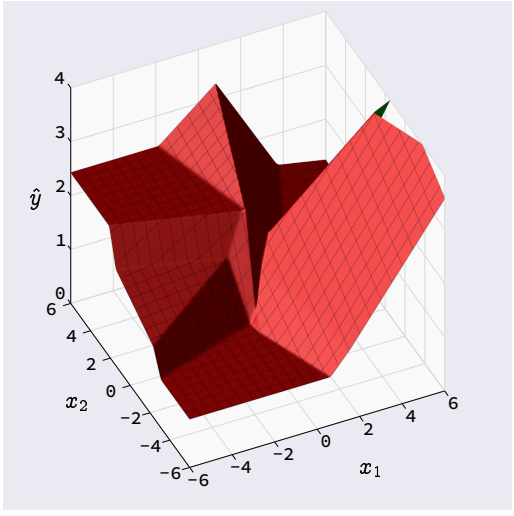
- neural networks are often represented by *network diagrams*
  - each vertex is a component of an activation
  - edges are individual weights or parameters
- example above has 3 layers, with $d^0 = 2$, $d^1 = 4$, $d_2 = 2$, $d_4 = 1$

Example neural network predictor

$$\theta_1 = \begin{bmatrix} 0.80 & 0.10 & 1.30 & 1.20 \\ -0.50 & 0.70 & 0.80 & 2.90 \\ -1.80 & 0.20 & -1.50 & -0.60 \end{bmatrix}$$

$$\theta_2 = \begin{bmatrix} 1.40 & 1.10 \\ -0.10 & -0.90 \\ 0.50 & 0.20 \\ -0.40 & 0.90 \\ -0.40 & -0.10 \end{bmatrix}$$

$$\theta_3 = \begin{bmatrix} 0.90 \\ 0.70 \\ 0.50 \end{bmatrix}$$



Layer terminology

- in an M layer network, layers 1 to $M-1$ are called *hidden layers*
- layer M is called the *output layer*
- often for regression, there is no activation function (*i.e.*, $h(a) = a$) in the output layer
- number of layers M is called the *depth* of the network
- using large M (say more than 3) is called *deep learning*

Training

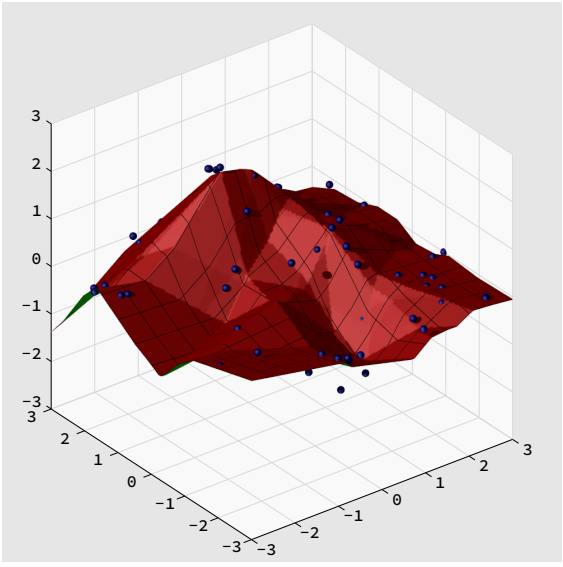
Training

- for RERM, we minimize over $\theta_1, \ldots, \theta_m$ the regularized empirical risk

$$\frac{1}{n}\sum_{i=1}^{n} \ell(g_\theta(x^i), y^i) + \lambda \sum_{j=1}^{m} r(\beta^j)$$

- we do not regularize the bias parameters α_j^i
- common regularizers include sum-squares and ℓ_1
- the RERM minimization problem is very hard or impossible to solve exactly
- so training algorithms find an approximately optimal solution, using iterative methods we'll see later
- these algorithms can take a long time (*e.g.*, weeks)

Example



Julia

```
function nnregression(X, Y, lambda)
    d = size(X,2);  m = size(Y,2)
    model = Chain(
        Dense(d, 10, relu),
        Dense(10, 10, relu),
        Dense(10, m, identity))
    data = zip(eachrow(X), eachrow(Y))
    reg() = normsquared(model[1].W) + normsquared(model[2].W) + normsquared(model[3].W)
    predicty(x) = model(x)
    loss(x,y) = normsquared(predicty(x)-y)
    cost(x,y) = loss(x,y) + lambda*reg()
    train(cost, Flux.params(model), data)
    return model
end
predictall(model,X) = vcat([model(x) for x in eachrow(X)]...)
train_error = rmse(predictall(model,Xtrain), Ytrain)
```

Neural networks as feature engineering

Neural networks versus feature engineering

- NNs have similar form to feature engineering pipeline
  - start with x
  - carry out a sequence of transforms or mappings
- feature engineering mappings are chosen by hand, have few (or zero) parameters, and are interpretable
- NN mappings have a specific form, with many parameters
- we can think of NNs as doing data-driven *automatic feature engineering*

Pre-trained neural networks

- first train a NN to predict some target
- this is usually done with a very large data set, and takes considerable time
- now fix the parameters in the NN
- use the last hidden layer output $z^{M-1} \in \mathbf{R}^{d_{M-1}}$ as a set of features for other prediction tasks
- can work well even when the other prediction tasks are quite different from the original one
- called a *pre-trained neural network*
- examples:
  - word2vec maps English words to vectors in $\mathbf{R}^{300}$
  - VGG16 maps images to vectors in $\mathbf{R}^{1000}$

Summary

Summary

- needs lots of data
- training can be tricky and take much time
- not interpretable
- often work well
- incorporate aspects of feature engineering
- can be used to do automatic, data-driven feature engineering