STAT 432: Basics of Statistical Learning

Neural Networks

Shiwei Lan, Ph.D. <shiwei@illinois.edu>

http://shiwei.stat.illinois.edu/stat432.html

April 6, 2019

University of Illinois at Urbana-Champaign

Neural Networks

- Motivation
- Feedforward neural network
- Connections with other models
- Deep Neural Networks

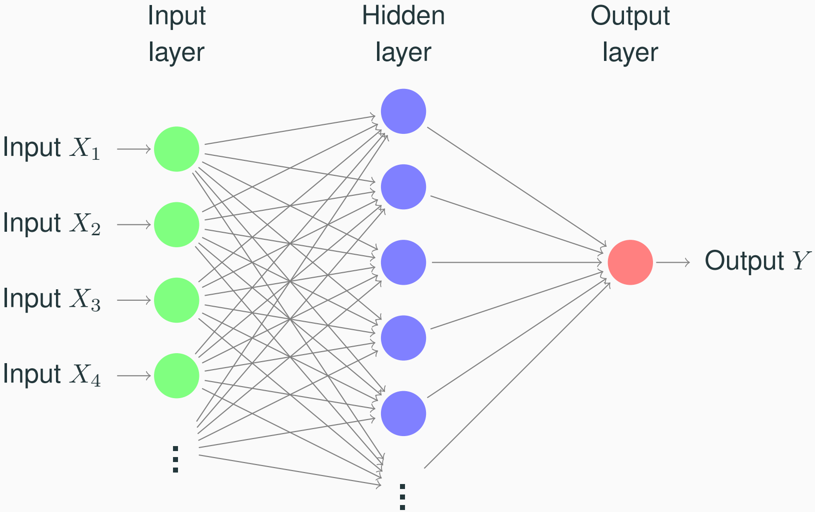
Neural Networks

Samples of cats and dogs images from Kaggle: Link

Neural Networks

- Neural Networks were first developed as models for the human brain, where we have many units (neurons) that simultaneously process signals to give a joint decision.
- The neurons fire when the total signal passed to that unit exceeds a certain threshold.
- The collective signal from all neurons tells you whether its a dog or a cat.

Neural Networks



Formulate the problem

- Given a training set $\{x_i, y_i\}_{i=1}^n$,
  - For regression: $y_i \in \mathbb{R}^K$ is a K dimensional continuous outcome
  - For classification: $y_i \in \{1, 2, \ldots, K\}$
- The goal is still to model the relationship

$$E(Y|X) = f(X)$$

- Instead of modeling the probabilities directly using X, we build M hidden neurons as a hidden layer between X and Y:

$$\begin{aligned} Z &= \big(1, Z_1, Z_2, \ldots, Z_M\big) \\ &= \big(1, \sigma(X^\mathsf{T}\boldsymbol{\alpha}_1), \sigma(X^\mathsf{T}\boldsymbol{\alpha}_2), \ldots, \sigma(X^\mathsf{T}\boldsymbol{\alpha}_M)\big) \end{aligned}$$

Formulate the problem

- $\sigma(\cdot)$ is an activation function. Some examples?
- We model Y using the hidden layer variables Z through some link function $g(\cdot)$

$$X \overset{\sigma(\cdot)}{\Longrightarrow} Z \overset{g(\cdot)}{\Longrightarrow} Y$$

- In classification problems (K class), we can use logit link g_k to model the probability of $Y = k$, for $k = 1, \ldots K$:

$$g_k(Z) = \frac{\exp(Z^{\mathsf{T}}\boldsymbol{\beta}_k)}{\sum_{l=1}^{K}\exp(Z^{\mathsf{T}}\boldsymbol{\beta}_k)}$$

- In regression problems (could be multidimensional), we can simply use a linear function to model the kth entry of Y:

$$g_k(Z) = Z^{\mathsf{T}}\boldsymbol{\beta}_k$$

Formulate the problem

- The multidimensional function $\mathbf{f}(x)$ can be represented as a convoluted way of mapping $x \in \mathbb{R}^p$ to $y \in \mathbb{R}^K$

$$\mathbf{f}(x) = \mathbf{g} \circ \boldsymbol{\sigma}(x)$$

- The notations $\mathbf{g}$ and $\boldsymbol{\sigma}$ here are multidimensional.
- The parameters involved are: $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_M$, and $\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_K$.

Examples of activation functions

- The activation function $\sigma(\cdot)$ takes a linear combination of the input variables, and output a scaler through nonlinear transformation. Examples:
  - sigmoid:

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

  - hyperbolic tangent (tanh):

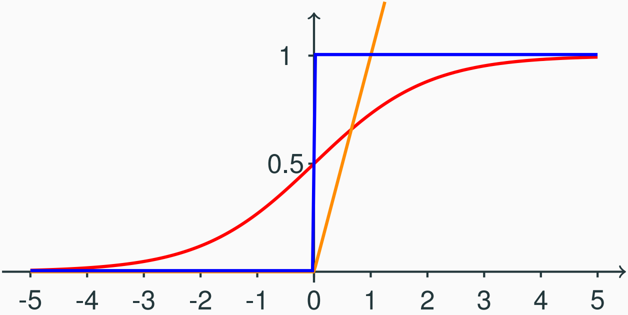
$$\sigma(v) = \frac{e^v - e^{-v}}{e^v + e^{-v}}$$

  - rectified linear unit (ReLU):

$$\sigma(v) = \max(0, v), \quad \text{soft approx.} \quad \ln(1+e^v)$$

  - And many others: exponential linear unit, arctangent, etc.

Activation Functions



Sigmoid: $(1 + e^{-v})^{-1}$

ReLU: $\max(0, v)$,

Step function: $I(v > 0)$

Activation Functions

- Originally, a step function $I(v > 0)$ was considered as the activation function (to mimic the biological interpretation). Hence for each neuron, signal is triggered only when $x^\mathsf{T}\alpha$ is above a certain threshold
- It was later recognized that the step function is not smooth enough for optimization, hence was replaced by a smoother threshold function, the sigmoid function
- “Feedforward” as signals can only pass to the next layer. There is no “cycle” in the model

Why Neural Networks work

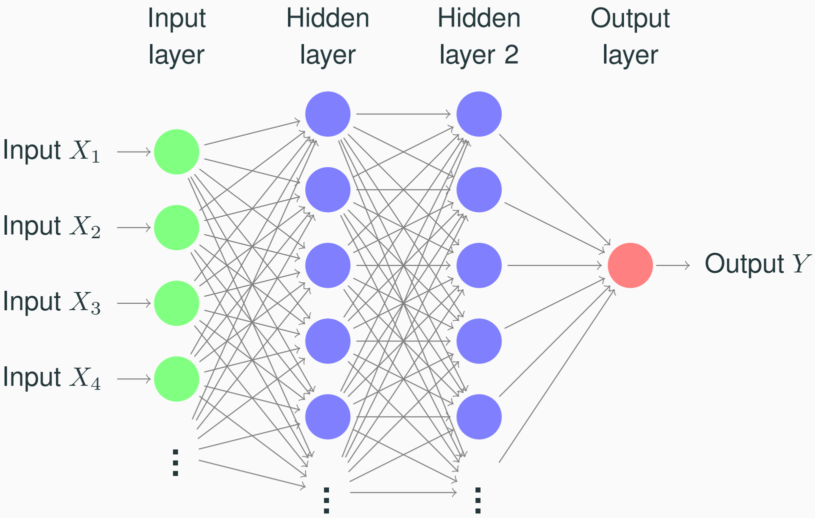
Universal Approximation Theorem (Cybenko, 1989; Hornik 1991)

Any continuous function $f(x)$ on the space $[0,1]^p$ can be approximated (for any $\epsilon > 0$) by a finite set of neurons with a bounded monotone-increasing activation function $\sigma(\cdot)$:

$$\left| f(x) - \sum_k w_k \sigma(\beta_k^\mathsf{T} x + b_k) \right| < \epsilon$$

for some w_k, β_k, and b_k. Hence, the functions defined by the neurons is dense.

Multiple Layers



Neural Networks

- Try this a really cool website: http://playground.tensorflow.org/
- Implementation in R:
  - packages: neuralnet, nnet
  - nnet fits a single layer of hidden neurons; neuralnet can fit multiple layers
  - The initial parameters α's and β's are generated randomly and then optimized. The model fitting can be different depends on the initial value. To fix initial parameters: nnet: Wts; neuralnet: startweights
  - Number of neurons: nnet: size; neuralnet: hidden (if hidden is specified as a vector, then there will be multiple layers)

Fitting Neural Networks

- The parameters (weights) $\boldsymbol{\alpha}$'s and $\boldsymbol{\beta}$'s need to be optimized.
- For a single hidden layer NN, we have

$$\{\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_M\} : \quad \text{M(p+1) weights}$$
$$\{\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_K\} : \quad \text{K(M+1) weights}$$

- where p is the number of non-intercept X features; M is the number of hidden neurons in a single layer; and K is the number of categories for classification.
- $K = 1$ if its a univariate regression problem.

Fitting Neural Networks

- Neural Networks training is based on error minimization using a Gradient Descent algorithm, known as error back-propagation.
- For K classification, we minimize Deviance:

$$-\sum_{i=1}^{n}\sum_{k}^{K}\mathbf{1}\{y_i = k\}\log f_k(x_i)$$

- For univariate regression, we minimize RSS (since g is linear):

$$\sum_{i=1}^{n}(y_i - f(x_i))^2 = \sum_{i=1}^{n}(y_i - \beta_0 - \beta_1\sigma(x^{\mathsf{T}}\boldsymbol{\alpha}_1) - \cdots \sigma(x^{\mathsf{T}}\boldsymbol{\alpha}_M))^2$$

Fitting Neural Networks

- The objective function can be written as

$$R(\boldsymbol{\theta}) = \sum_{i=1}^{n} R_i(\boldsymbol{\theta})$$

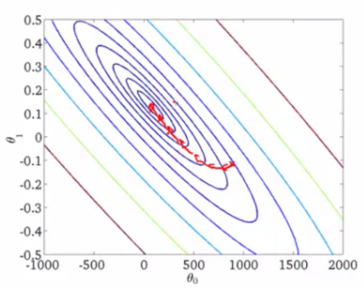
  where R_i represents the deviance or residual sum of squares for the ith data point, and $\boldsymbol{\theta}$ represents an aggregated vector of all weights
- Initiate weights $\boldsymbol{\theta}^{(0)}$
- We then calculate the derivative wrt each of the weights evaluated at the current iteration value $\boldsymbol{\theta}^{(t)}$:

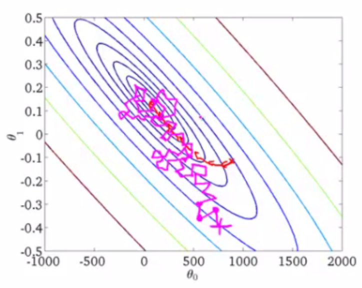
$$\sum_{i=1}^{n} \frac{\partial R_i}{\beta_{km}}\bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}} \qquad \sum_{i=1}^{n} \frac{\partial R_i}{\alpha_{mj}}\bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}}$$

- Stochastic GD: the summation can be taken over a random subset of the n samples

GD vs. Stochastic GD

Gradient Descent vs. Stochastic Gradient Descent





Fitting Neural Networks

- The derivatives for $K = 1$ regression case is essentially

$$\frac{\partial R_i}{\beta_m} = -2(y_i - f(x_i))z_{mi}$$
$$\frac{\partial R_i}{\alpha_{ml}} = -2(y_i - f(x_i))\beta_m \sigma'(\boldsymbol{\alpha}_m^T x_i)x_{il}$$

- Some redundant calculations can be saved in the above equations. The property is called back-propagation.
- We then do the update, at the t-th iteration

$$\beta_m^{(t+1)} = \beta_m^{(t)} - \gamma \sum_{i=1}^{n} \frac{\partial R_i}{\beta_m^{(t)}}$$
$$\alpha_{ml}^{(t+1)} = \alpha_{ml}^{(t)} - \gamma \sum_{i=1}^{n} \frac{\partial R_i}{\alpha_{ml}^{(t)}}$$

where γ is a step size for gradient descent.

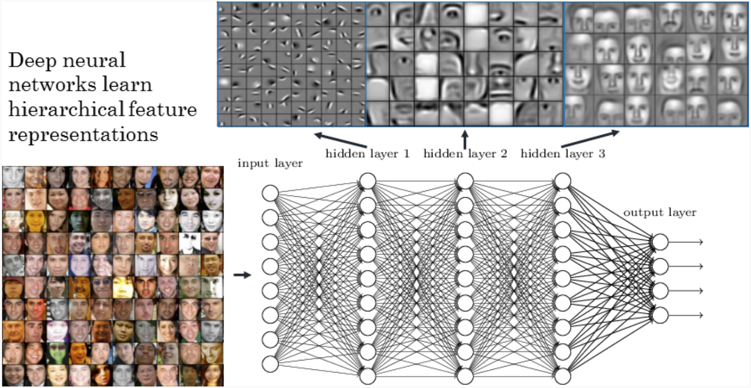
Fitting Neural Networks

- The derivatives can be calculated by Chain Rules
- The algorithm can be implemented by a forward-backward sweep over the network
- In the forward pass, compute the hidden variables and the output $\widehat{f}(x_i)$ based on the current weights $\boldsymbol{\theta}^{(t)}$
- In the backward pass, compute the derivatives, and update $\boldsymbol{\theta}^{(t)} \to \boldsymbol{\theta}^{(t+1)}$

Going Deeper...

- Deep Neural Networks are one type of deep learning models.
- Deep neural Networks are just ... Neural Networks with more than one hidden layer.
- But neural networks have been around for more than 70 years... why it gets popular just in recent years?
    - computational issues
    - a better way to generate/construct features
    - ...

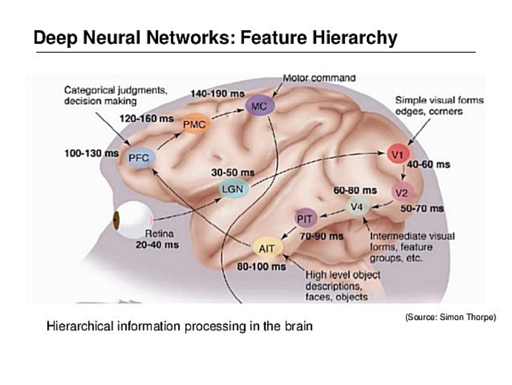
Deep Neural Networks



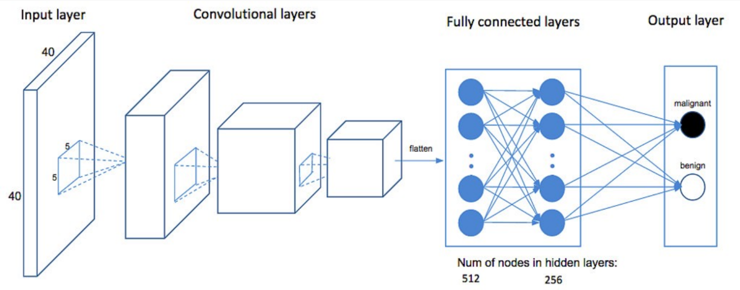
Convolutional Neural Networks

- One example is the Convolutional Neural Networks, which attempts to generate better features
- Instead of using all input features to create the linear combination, a “convolutional layer” builds neurons that each takes a subset (a local region) of the input features.
- This is motivated by the fact that biologically, the neurons only take signals from neighboring neurons.

Convolutional Neural Networks



Convolutional Neural Networks



See this hand digit writing recognition example, and this interesting application by Tesla.