STAT 5630, Fall 2019

Neural Networks

Xiwei Tang, Ph.D. <xt4yj@virginia.edu>

University of Virginia
November 19, 2019

Neural Networks

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

Neural Networks

Dogs vs. Cats classification problem at Kaggle: Link

Neural Networks

- Neural Networks were first developed as models for the human brain, where each unit represents a neuron.
- 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

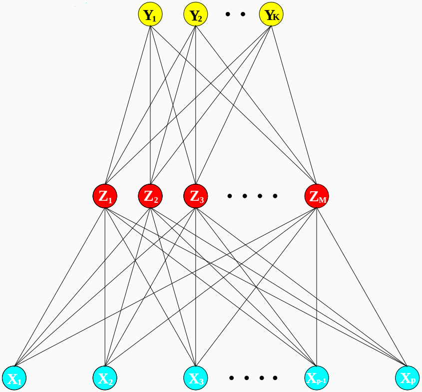


FIGURE 11.2. *Schematic of a single hidden layer, feed-forward neural network.*

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:

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

- Where $\sigma(\cdot)$ is an activation function. Some examples?
- Then we model Y using the hidden layer variables $\mathbf{Z}$ through some link function $g(\cdot)$

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

Formulate the problem

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

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

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

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

- 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.

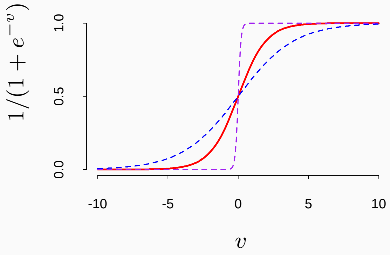


FIGURE 11.3. *Plot of the sigmoid function* $\sigma(v) = 1/(1 + \exp(-v))$ *(red curve), commonly used in the hidden layer of a neural network. Included are* $\sigma(sv)$ *for* $s = \frac{1}{2}$ *(blue curve) and* $s = 10$ *(purple curve). The scale parameter* s *controls the activation rate, and we can see that large* s *amounts to a hard activation at* $v = 0$*. Note that* $\sigma(s(v - v_0))$ *shifts the activation threshold from* 0 *to* v_0*.*

Formulate the problem

- Originally, a step function $\sigma(v)$ 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

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)

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 (with any $\epsilon > 0$) by a finite set of neurons with a bounded monotone-increasing activation function $\varphi(\cdot)$:

$$\left| f(x) - \sum_{i=1}^{n} v_i \varphi(w_i^\intercal x + b_i) \right| < \epsilon$$

for some v_i, w_i, and b_i. Hence, the functions defined by the neurons is dense.

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

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

- 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

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

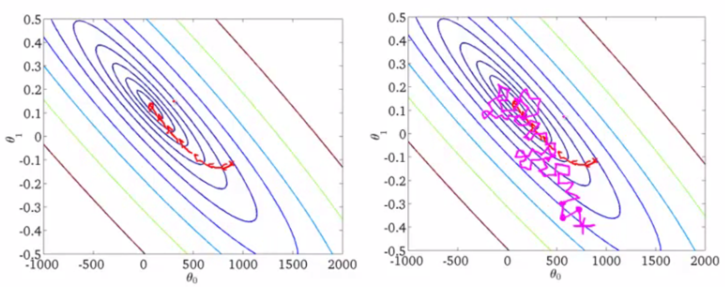
  where R_i represents the deviance or residual sum of squares for the ith data point, and $\boldsymbol{\theta}$ represents an aggregated vector of all weights
- Initiate weights $\boldsymbol{\theta}^{(0)}$
- We then calculate the derivative wrt each of the weights evaluated at the current iteration value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

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

- 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.

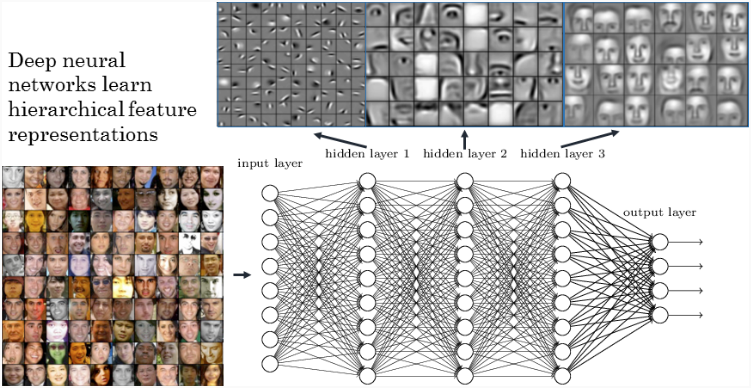
Fitting Neural Networks

- 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
    - ...

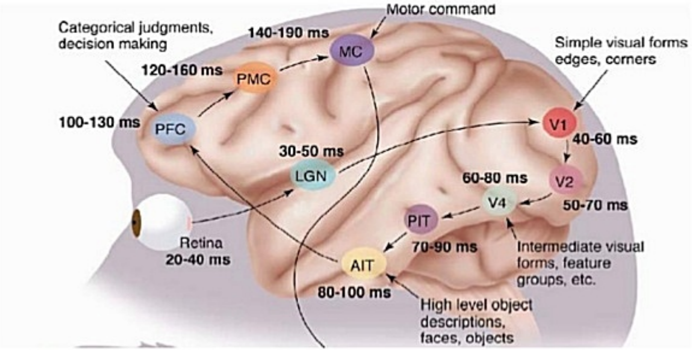
Deep Neural Networks



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.

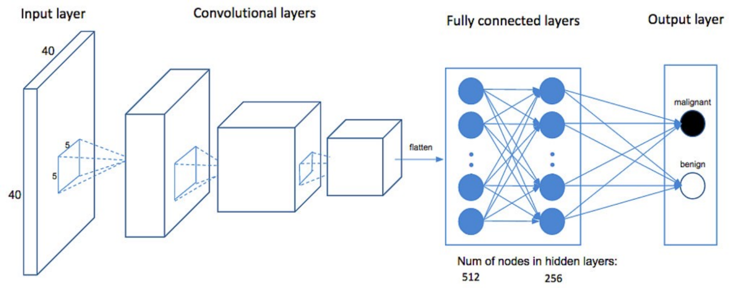
Deep Neural Networks: Feature Hierarchy



(Source: Simon Thorpe)

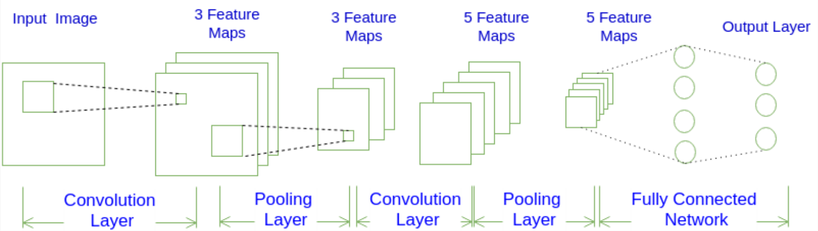
Hierarchical information processing in the brain

Convolutional Neural Networks



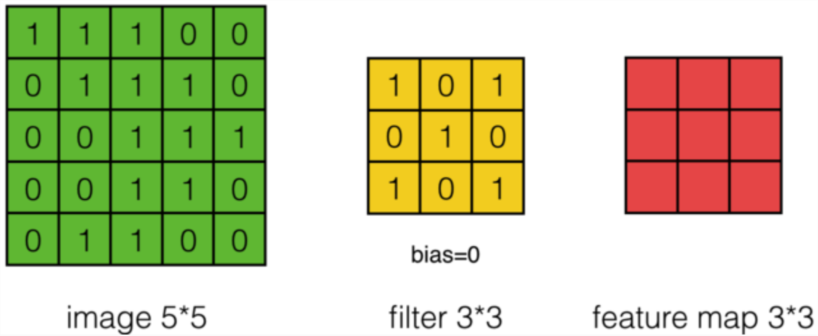
See this hand digit writing recognition example, and this interesting application by Tesla.

Convolutional Neural Networks



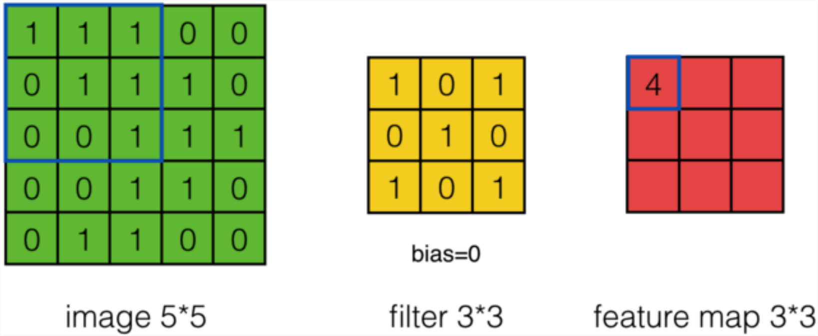
- The CNN consists of input an layer, convolution layers, pooling layers, a fully connected network and an output layer.
- Also known as shift invariant or space invariant artificial neural networks (SIANN).

Convolutional Layers



- Choose a filter which is assigned with certain weights $w_{m,n}$'s.

Convolutional Layers

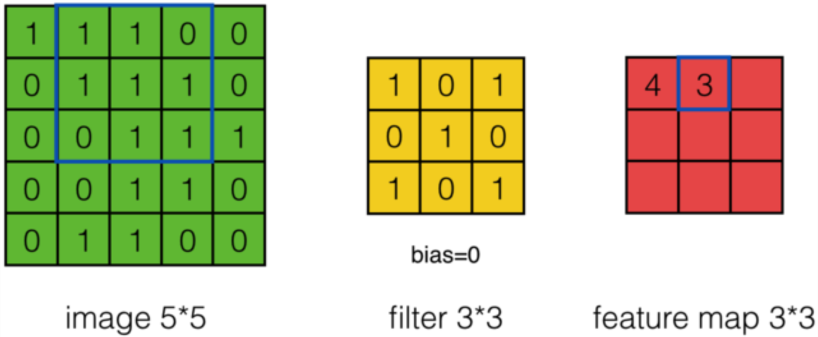


- Calculate the convolution (cross-correlation) by filter locally.

$$a_{i,j} = f\left(\sum_{m=0}^{2}\sum_{n=0}^{2} w_{m,n} x_{i+m,j+n} + w_b\right)$$

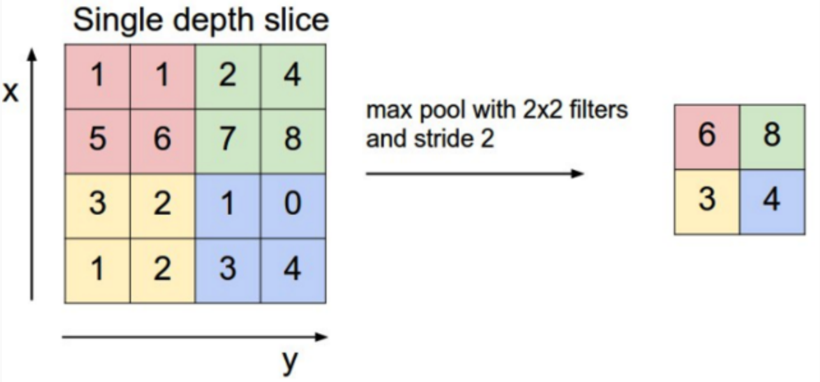
- $f(\cdot)$ is the activiation function.

Convolutional Layers



- Moving the filter across the image to obtain a feature map.
- Choose different stride size.
- Use multiple filters which results in multiple feature maps (depth)

Pooling Layers



- Dimension Reduction
- Choose pooling window size and stride size
- Max pooling, Mean Pooling, Median Pooling...
- Pooling on each layers individually

Some Points

- Vanishing gradient and exploding gradient
- Epoch: one epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE.
- Learning rate: gradient descent step size
- Batch size and number of batches (iterations)
- Momentum and normalization
- Weight Decay, regularization and dropout: overfitting

Some Other NN

- Recurrent Neural Network (RNN)
- Long Short Term Memory Network, LSTM
- Recursive Neural Network, RNN

Happy Thanksgiving!!!