Introduction to Neural Networks and Backpropagation Algorithm

August 07, 2018

Quick Review

What you've learned so far

- Unsupervised learning

  dimension reduction (PCA, multilinear PCA)

  clustering algorithms (k-means, SUP or blurring mean-shift)

- Supervised learning

  LDA, logistic regression (linear classification)

  SVMs (linear and kernel)

All the above methods fall into the category of **shallow models**.

Here we will introduce **"deeper" models** using neural networks.

- Data visualization

Distance

- Euclidean
- Mahalanobis distance

  (standardization by data covariance matrix)
- kernel map, feature Hilbert space

  inner product: $\Phi(x_1) \cdot \Phi(x_2) = \Phi(x_1)^\top \Phi(x_2) = K(x_1, x_2)$

  inner product induced norm: Let $z_j = \Phi(x_j)$.

$$\|z_1 - z_2\| = \sqrt{z_1^\top z_1 + z_2^\top z_2 - 2 z_1^\top z_2}$$

All the above are L_2-type.

- L_1: in SVM we used $\max\{0, 1 - y(w^\top x + b)\}$

- Distance (or statistical distance, or divergence) between two distributions

  KL divergence (used in logistic regression)

  $D_{\mathsf{KL}}(p, \widehat{p}_\theta) = \sum_{j=1}^{k} \left[p_j \ln(p_j) - p_j \ln(\widehat{p}_j) \right]$ (cross-entropy loss)

Two key concepts:

model building & model fitting (or model training in ML language)

Going deeper and nonlinear

- Stacking or composition of linear functions is still linear. Thus, we need nonlinear transforms for nonlinear model.
- Deep model structure enables us to describe very complex model.
- However, deep model is difficult to train. It also requires heavy computation.

Single Neuron Neural Network

There are three major components of a neural network algorithm: network architecture, activation function and learning rule.

(model building and model fitting)

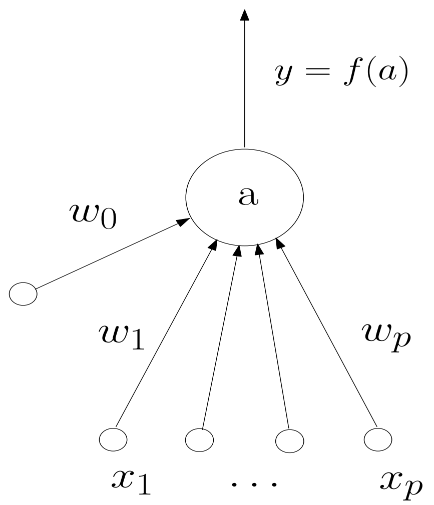
A single-neuron network

Below we introduce a single-neuron network for binary-class **logistic regression**

Suppose we have training data set $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, where $\boldsymbol{x}$'s are explanatory variables in $\mathbb{R}^p$ and y's are associated class membership labeled by $\mathbf{\{0, 1\}}$.

♠ **Architecture**

(single neuron network)



The network consists of input attributes $\boldsymbol{x} = (x_0, x_1, \dots, x_p) \in \Re^{p+1}$, *connecting weights* (also known as *synaptic weights*) $\boldsymbol{w} = (w_0, w_1, \dots, w_p)$ for combining $\boldsymbol{x}$, and a single output unit y. Here $x_0 \equiv 1$ and where w_0 is an intercept, also named "offset" or "bias".

♠ **Activation rule** (transferring derived features to output)

♠ **Learning rule** (training the network, fitting the model)

Activation rule (transferring derived features to output)

- *Sigmoid* transfer function:

$$y = f(a) = \frac{1}{1 + \exp(-a)}, \quad \text{where } a = \boldsymbol{w}^\top \boldsymbol{x}.$$

  At the output neuron, the derived linear feature $\boldsymbol{w}^\top \boldsymbol{x}$ is transferred by the activation function f to a value in $(0, 1)$ often interpreted as the probability of being in class 1. The classification prediction is then given by sign$(y - 0.5)$ or floor$(y + 0.5)$.

- There are other transfer functions: linear, ReLU, tanh, etc.

Learning rule (training the network, fitting the model)

- An error function has to be specified to measure the discrepancy between the network model and data.
- Two common error functions

  -Squared error between two vectors: $\|y_i - f(\boldsymbol{w}^\top \boldsymbol{x}_i)\|_2^2$

  -KL divergence between two probability distributions (also known as cross-entropy loss):

$$\mathcal{D}_{KL}(\text{data}, \text{model}) = -\frac{1}{n}\sum_{i=1}^{n}\left\{y_i \ln f(\boldsymbol{w}^\top \boldsymbol{x}_i) + (1 - y_i)\ln(1 - f(\boldsymbol{w}^\top \boldsymbol{x}_i))\right\}.$$

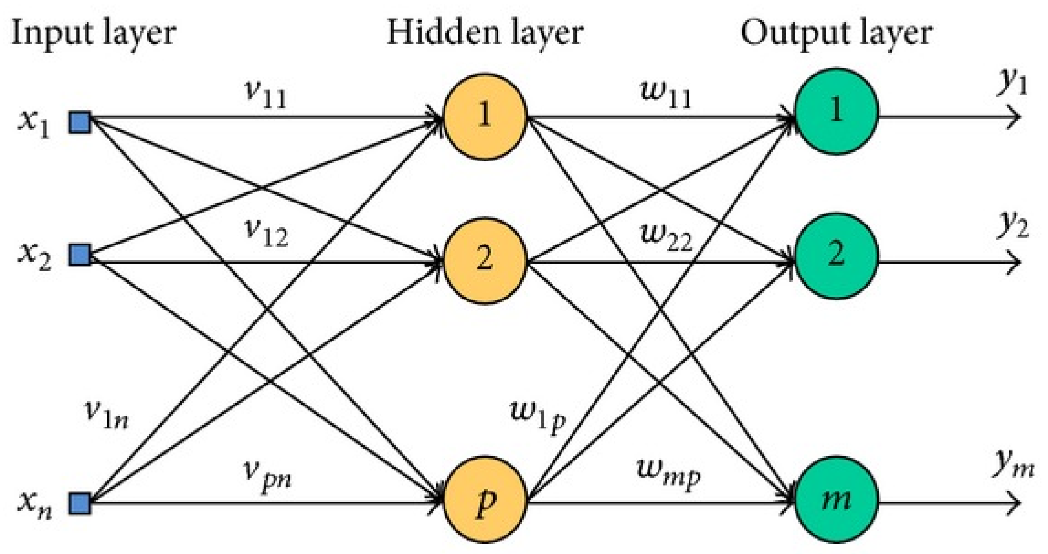
- To train the neural network is to fit (or to estimate) $\boldsymbol{w}$ based on the observational data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$.
- (logistic regression, stochastic gradient descent, batch, epoch,...)

You can try out LDA, linear SVM, logistic regression and single neuron NN (with sigmoid-transfer and cross-entropy loss) for **linear** classification.

Compare results.

Stacking Neural Network Layers for Nonlinearity

One-hidden layer neural network



Universal approximation theorem

The universal approximation theorem states that a feedforward network with a single hidden layer containing a finite number of neurons can approximate continuous functions on compact subsets of $\mathbb{R}^n$, under mild assumptions on the activation function. (from wiki) (not imply one can get reliable parameter estimate)

One-hidden layer neural network for m-class classification

- Inputs: $\boldsymbol{x}^{(1)} = (x_0, x_1, \ldots, x_{p_1})$, $x_0 = 1$;

  Derived features: $\boxed{\boldsymbol{z}^{(1)},\ z_j^{(1)} = \boldsymbol{v}_j^\top \boldsymbol{x}}$, $j = 1, \ldots, p_2$.

- Hidden: $\boxed{\boldsymbol{x}^{(2)},\ x_j^{(2)} = h(z_j^{(1)})}$, where $h(t)$ is a transfer (or activation) function of our choice;

  Hidden features: $\boxed{\boldsymbol{z}^{(2)},\ z_k^{(2)} = \boldsymbol{w}_k^\top \boldsymbol{x}^{(2)}}$, $k = 1, \ldots, m$.

- Outputs: $y_k = f(z_k^{(2)})$.

  Softmax activation function:

$$f(a_k|\boldsymbol{a}) = \frac{e^{a_k}}{\sum_{j=1}^m e^{a_j}} = \frac{e^{a_k}}{e^{a_k} + \sum_{j\neq k}^m e^{a_j}} = \frac{e^{\tilde{a}_k}}{1 + e^{\tilde{a}_k}}.$$

- **Fitting criterion**: least squares, minimun divergence, ...

Two error criteria

- $R(\boldsymbol{\theta}) \overset{\text{squared error}}{=} \frac{1}{2}\sum_{i=1}^{n} \|\boldsymbol{y}_i - \boldsymbol{f}_i\|^2 \qquad (\boldsymbol{f}_i = \widehat{\boldsymbol{y}}_i\text{: fitted})$

  $= \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{m}(y_{ki} - f_{ki})^2$

- $R(\boldsymbol{\theta}) \overset{\text{KL (cross-entropy)}}{=} \sum_{i=1}^{n} C(\boldsymbol{y}_i, \boldsymbol{f}_i) = \sum_{i=1}^{n}\sum_{k=1}^{m} y_{ki} \ln(y_{ki}/f_{ki})$

- $\boldsymbol{y}_i = (y_{1i}, \ldots, y_{mi})^\top$ denotes the observed class label for the ith instance coded using indicator dummy variables.

  $f_{ki} = f(z_k^{(2)}(\boldsymbol{x}_i))$, which is the predicted probability for $\boldsymbol{x}_i$ being in class k.

Deep Neural Network

Notation:

- ℓ-th layer variables: $\boldsymbol{x}^{(\ell)}$
- $\sigma^{(\ell)}$: activation (transfer) function
- parameters: $W^{(\ell)}$, $b^{(\ell)}$

Neural network with L layers for m-class classification

- Inputs: $\boldsymbol{x}^{(1)} = (x_1, \ldots, x_{p_1}) \in \mathbb{R}^{p_1}$;

  Derived features: $\boxed{\boldsymbol{z}^{(1)} = \boldsymbol{W}^{(1)}\boldsymbol{x}^{(1)} + \boldsymbol{b}^{(1)}}$,

- Hidden: $\boxed{\boldsymbol{x}^{(2)} = \sigma^{(1)}(\boldsymbol{z}^{(1)}) \in \mathbb{R}^{p_2}}$, where $\sigma^{(1)}(t)$ is a transfer (or activation) function;

- $\boldsymbol{x}^{(1)}, \ldots \boldsymbol{x}^{(\ell)}, \ldots, \boldsymbol{x}^{(L)}$

- Top layer (output layer): $\boxed{\hat{\boldsymbol{y}} = \sigma^{(L)}\left(\boldsymbol{W}^{(L)}\boldsymbol{x}^{(L)} + b^{(L)}\right) \in \mathbb{R}^m}$,

- Softmax outputs: $\widehat{y}_k = \dfrac{\exp(z_k^{(L)})}{\sum_{j=1}^m \exp(z_j^{(L)})}$.

- Cross entropy: $\sum_{\text{all data}} C(\boldsymbol{y}, \hat{\boldsymbol{y}})$, $\quad C(\boldsymbol{y}, \hat{\boldsymbol{y}}) = -\sum_{k=1}^m y_k \ln(\widehat{y}_k)$

Ideas of BP algorithm

- Forward pass: parameters $\{\boldsymbol{W}^{(\ell)}, \boldsymbol{b}^{(\ell)}\}_{\ell=1}^{L}$ are fixed, and predicted values $\widehat{\boldsymbol{y}}$ are updated.
- Backward pass: Backpropagation is commonly used by the gradient descent optimization algorithm to **adjust the weights of neurons** by calculating the gradient of the loss function.

Back-propagation (using cross-entropy loss as example)

$$\sum_{i\in\text{mini batch}} \frac{\partial C(\boldsymbol{y}_i,\widehat{\boldsymbol{y}}_i)}{\partial \text{vec}\left(\boldsymbol{W}^{(L)}\right)^\top} = \sum_i \frac{\partial C(\boldsymbol{y}_i,\widehat{\boldsymbol{y}}_i)}{\partial \widehat{\boldsymbol{y}}_i^\top}\frac{\partial \widehat{\boldsymbol{y}}_i}{\partial \text{vec}(\boldsymbol{W}^{(L)})^\top}$$

$$= -\sum_i \boldsymbol{r}_i^\top \frac{\partial \sigma(\boldsymbol{z}_i^{(L)})}{\partial \text{vec}(\boldsymbol{W}^{(L)})^\top}, \quad \text{where } \boldsymbol{r}_i^\top = \left(\frac{y_1}{\widehat{y}_1},\ldots,\frac{y_m}{\widehat{y}_m}\right)$$

$$\frac{\partial \boldsymbol{z}^{(L)}}{\partial \text{vec}(\boldsymbol{W}^{(L)})^\top} = \boldsymbol{x}^{(L)\top} \otimes \boldsymbol{I}_m$$

$$= -\sum_i \boldsymbol{r}_i^\top \left[\text{diag}\dot\sigma(\boldsymbol{z}_i^{(L)}).\right]_{m\times m}\left(\boldsymbol{x}_i^{(L)\top} \otimes \boldsymbol{I}_m\right)_{m\times mp_L}$$

$$\overset{\text{folded}}{\longrightarrow} \begin{bmatrix} \dot\sigma(z_{1i}^{(L)}) & 0 & \ldots & \ldots \\ 0 & \dot\sigma(z_{2i}^{(L)}) & 0 & \ldots \\ \vdots & \vdots & \vdots & \vdots \\ \ldots & \ldots & 0 & \dot\sigma(z_{mi}^{(L)}) \end{bmatrix} \boldsymbol{r}_i\, \boldsymbol{x}_i^{(L)\top}$$

$$\sum_{i=1}^n \frac{\partial C(\boldsymbol{y}_i,\widehat{\boldsymbol{y}}_i)}{\partial \boldsymbol{x}_i^{(L)\top}} = \sum_{i=1}^n \frac{\partial C(\boldsymbol{y}_i,\widehat{\boldsymbol{y}}_i)}{\partial \widehat{\boldsymbol{y}}_i^\top}\frac{\partial \widehat{\boldsymbol{y}}_i}{\partial \boldsymbol{x}_i^{(L)\top}}$$

Back-propagation -2

- With **upper layers gradients** $\frac{\partial C}{\partial \boldsymbol{W}^{(\ell+1)}}$ and $\frac{\partial C}{\partial \boldsymbol{x}^{(\ell+1)}}$ being computed, we go for the **next lower layer** and have

$$\frac{\partial C}{\partial \text{vec}(\boldsymbol{W}^{(\ell)})^\top} = \frac{\partial C}{\partial \boldsymbol{x}^{(\ell+1)\top}} \frac{\partial \boldsymbol{x}^{(\ell+1)}}{\partial \text{vec}(\boldsymbol{W}^{(\ell)})^\top}$$

$$\frac{\partial C}{\partial \boldsymbol{x}^{(\ell)\top}} = \frac{\partial C}{\partial \boldsymbol{x}^{(\ell+1)\top}} \frac{\partial \boldsymbol{x}^{(\ell+1)}}{\partial \boldsymbol{x}^{(\ell)\top}}$$

  where $\boldsymbol{x}^{(\ell+1)} = \sigma\left(\boldsymbol{W}^{(\ell)}\boldsymbol{x}^{(\ell)} + \boldsymbol{b}^{(\ell)}\right)$.

- Updates: $\theta_{t+1} = \theta_t - \gamma_t \frac{\partial C}{\partial \theta_t}$,

  γ_t: learning rate; $\theta = \left\{\boldsymbol{W}^{(\ell)}, \boldsymbol{b}^{(\ell)}\right\}_\ell$.

- stochastic gradient descent, batch, batch size, epoch

You can try out kernel SVM and deep NN for **nonlinear** classification, and compare results.

End of Basic Introduction

some quick review for (a) model building, (b) model fitting and (c) loss criterion (distance metric)

2 examples of deep models (PCANet, CNN) below

PCA Net

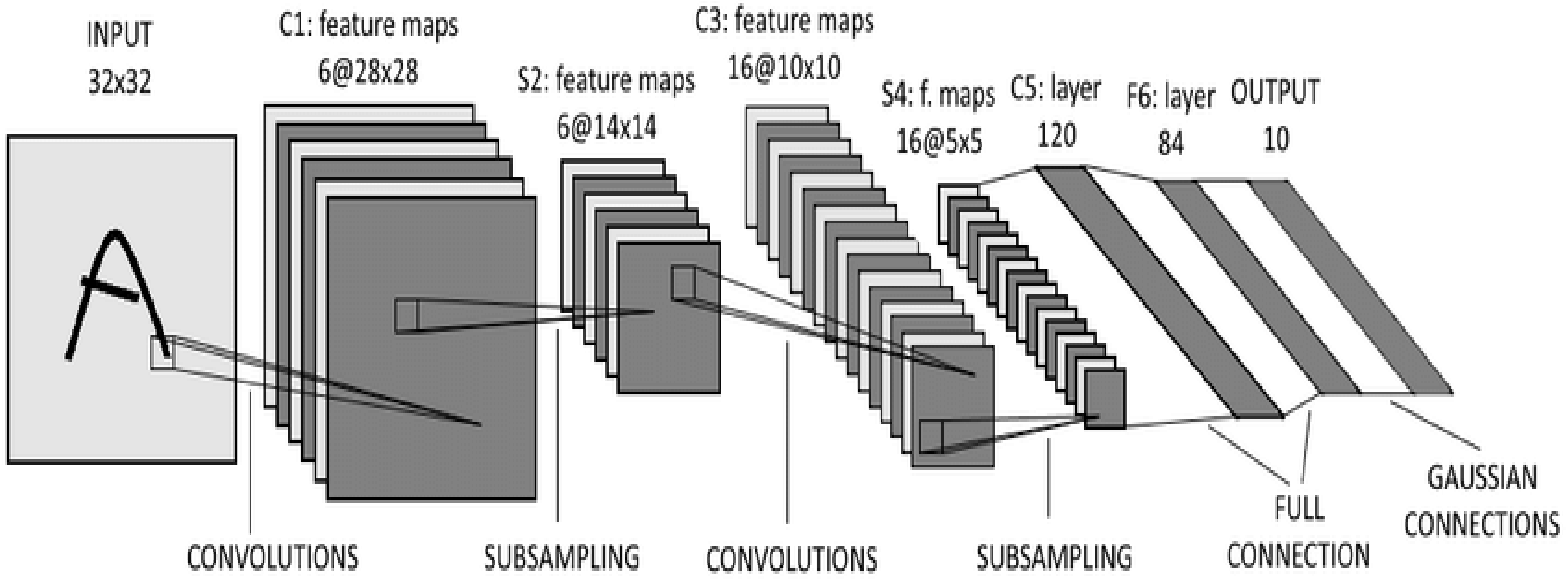
Chan et al., IEEE Transactions on Image Processing, 2015

(some quick review for PCA, multilinear PCA)

PCANet demo using Olivetti faces dataset & MNIST

Convolution neural network

Convolution neural network LeNet5 (Yann LeCun)



What is a convolution?

- As an illustration example, let W be a 3×3 convolution kernel with stride $= 2$.

$$\boldsymbol{x} = \begin{bmatrix} 1 & -2 & 3 & 1 & 2 \\ 4 & 5 & 4 & 1 & 5 \\ 3 & 6 & 0 & 1 & 5 \\ 2 & 6 & -1 & -1 & 5 \\ 7 & 8 & -2 & 1 & -8 \end{bmatrix}, \quad \boldsymbol{B} = \begin{bmatrix} 1 & 3 & 3 & 0 \\ 4 & 2 & 4 & -1 \\ 3 & 7 & 0 & -2 \\ -2 & 6 & 1 & 1 \\ 5 & 6 & 1 & -1 \\ 6 & 8 & 1 & 1 \\ 3 & 0 & 2 & 5 \\ 4 & -1 & 5 & 5 \\ 0 & -2 & 5 & 8 \end{bmatrix}.$$

$$\text{vec}(\boldsymbol{W})^\top * \boldsymbol{B} = \text{vec}(\boldsymbol{W})^\top * \phi(\boldsymbol{x}) \xrightarrow{\text{folded}} 2 \times 2 \text{ matrix}$$

- Fully connected layer can also be viewed as a convolution layer.

For pooling in ℓ^{th} layer

- e.g., average pooling $\boldsymbol{P} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$,

  fixed matrix, no parameter involved

- no $\frac{\partial C}{\partial \text{vec}(\boldsymbol{W}^{(\ell)\top})}$,

  still have $\frac{\partial C}{\partial \boldsymbol{x}^{(\ell)\top}} = \frac{\partial C}{\partial \boldsymbol{x}^{(\ell+1)\top}} \frac{\partial \boldsymbol{x}^{(\ell+1)}}{\partial \boldsymbol{x}^{(\ell)\top}}$.

- $\frac{\partial C}{\partial \text{vec}(\boldsymbol{W}^{(\ell-1)\top})} = \frac{\partial C}{\partial \boldsymbol{x}^{(\ell)\top}} \frac{\partial \boldsymbol{x}^{(\ell)}}{\partial \text{vec}(\boldsymbol{W}^{(\ell-1)})^\top}$,

  $\cdots\cdots$

examples demo

logistic regression vs. neural networks

PCA filters vs. convolution filters

robust loss functions against contamination