Report on UFLDL-Part II

Yunfei WANG

[1]School of Computer Science & Technology
Huazhong University of Science & Technology

June 18th, 2013

Table of contents

1. Softmax Regression
   - Logistic Regression
   - Generalize Logistic Regression to Softmax Regression
   - Cost Function
   - Properties of softmax regression parameterization
   - Weight Decay
   - Softmax Regression vs. Logistic Regression
2. Linear Decoders with Autoencoders
3. Self-Taught Learning

Logistic Regression-Binary Classification

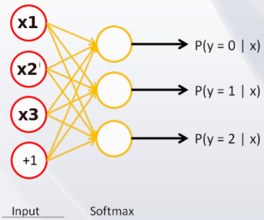
$$h_\theta(x) = \frac{1}{1+\exp(-\theta^T x)} \propto \exp(\theta^T x) \quad (1)$$

Training set $\{(x^1, y^1), (x^2, y^2), \cdots, (x^m, y^m) | x^i \in \Re^{n+1}, y^i \in \{0, 1\}\}$.

Train the model parameters θ to minimize the following cost function:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^i \log h_\theta(x^i) + (1-y^i)\log(1-h_\theta(x^i))\right] \qquad (2)$$

Generalize Logistic Regression to Softmax Regression



Training set $\{(x^1, y^1), (x^2, y^2), \cdots, (x^m, y^m) | x^i \in \Re^{n+1}, y^i \in \{1, \cdots, k\}\}$.
Estimate the probability for each class:

$$h_\theta(x^i) = \begin{bmatrix} p(y^i = 1|x;\theta) \\ p(y^i = 2|x;\theta) \\ \vdots \\ p(y^i = k|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} \exp(\theta_j^T x^i)} \begin{bmatrix} \exp(\theta_1^T x^i) \\ \exp(\theta_2^T x^i) \\ \vdots \\ \exp(\theta_k^T x^i) \end{bmatrix} \tag{3}$$

where $\theta_1, \theta_2, \cdots, \theta_k \in \Re^{n+1}$ are parameters of model, θ is a matrix obtained by stacking up $\theta_1, \theta_2, \cdots, \theta_k$ in rows.

Cost Function used for softmax regression:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k} 1\{y^i = j\}\log\frac{\exp(\theta_j^T x^i)}{\sum_{l=1}^{k}\exp(\theta_l^T x^i)}\right] \tag{4}$$

where $1\{\cdot\}$ is indicator function:$1\{True\} = 1, 1\{False\} = 0$.

Taking partial derivatives,we get the following gradient:

$$\nabla_{\theta_j} J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[x^i(1\{y^i = j\} - p(y^i = j|x^i;\theta))\right] \tag{5}$$

Armed with the derivative,algorithm such as gradient descent an L-BFGS can be used to minimize $J(\theta)$.

Cost Function II

Next,make a brief derivation for the gradient above.Considering a single sample (x^i, y^i):

- when $y^i = j$:

$$J(\theta; x^i) = \sum_{j=1}^{k} 1\{y^i = j\} \log \frac{\exp(\theta_j^T x^i)}{\sum_{l=1}^{k} \exp(\theta_l^T x^i)} = \log \frac{\exp(\theta_j^T x^i)}{\sum_{l=1}^{k} \exp(\theta_l^T x^i)} \tag{6}$$

$$\nabla_{\theta_j} J(\theta; x^i) = x^i (1 - p(y^i = j | x^i; \theta)) \tag{7}$$

- when $y^i = t \neq j$:

$$J(\theta; x^i) = \sum_{j=1}^{k} 1\{y^i = j\} \log \frac{\exp(\theta_j^T x^i)}{\sum_{l=1}^{k} \exp(\theta_l^T x^i)} = \log \frac{\exp(\theta_t^T x^i)}{\sum_{l=1}^{k} \exp(\theta_l^T x^i)} \tag{8}$$

$$\nabla_{\theta_j} J(\theta; x^i) = x^i (0 - p(y^i = j | x^i; \theta)) \tag{9}$$

Properties of softmax regression parameterization

Softmax regression has a "redundant" set of parameters:

$$
\begin{aligned}
p(y^i = j|x^i;\theta) &= \frac{\exp((\theta_j - \psi)^T x^i)}{\sum_{l=1}^{k} \exp((\theta_l - \psi)^T x^i)} \\
&= \frac{\exp(\theta_j^T x^i)\exp(-\psi^T x^i)}{\sum_{l=1}^{k} \exp(\theta_l^T x^i)\exp(-\psi^T x^i)} \\
&= \frac{\exp(\theta_j^T x^i)}{\sum_{l=1}^{k} \exp(\theta_l^T x^i)}
\end{aligned}
\tag{10}
$$

In other words,subtracting ψ from every θ_j doesn't affect predictions at all.

- By setting $\psi = \theta_1$,we could eliminate one parameter θ_1
- Preprocessing inner product to avoid overflow when computing $\exp(\theta_l^T x^i)$

Weight Decay

Add weight decay term to penalize large values of parameters.
Our cost function is now:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k}1\{y^i = j\}\log\frac{\exp(\theta_j^T x^i)}{\sum_{l=1}^{k}\exp(\theta_l^T x^i)}\right] + \frac{\lambda}{2}\sum_{i=1}^{k}\sum_{j=1}^{n}\theta_{ij}^2 \tag{11}$$

With weight decay term(for any $\lambda \geq 0$),the cost function $J(\theta)$ is strictly convex and is guaranteed to have a unique solution.
To apply an optimization algorithm,compute its derivative:

$$\nabla_{\theta_j}J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[x^i(1\{y^i = j\} - p(y^i = j|x^i;\theta))\right] + \lambda\theta_j \tag{12}$$

Because $J(\theta)$ is convex,algorithms such as gradient descent and L-BFGS are guaranteed to converge to global minimum.

Softmax Regression vs. Logistic Regression

For multi-class classification application,which one is more suitable?

Mutually exclusive classes

- indoor_scene
- outdoor_urban_scene
- outdoor_wilderness_scene

Softmax regression is appropriate in this situation.

Classes not mutually exclusive

- pop music
- dance music
- vocal music

Building separate logistic regression classifiers.

Experiments on Softmax/Logistic Regression

Dataset	Dimensionality	Number of Samples
Training Data	784	60000
Testing Data	784	10000

Table: MNIST

Classifier	Accuracy	Time Consumed
Softmax Regression	92.63%	352sec
Logistic Regression	81.84%	218sec

Table: Experiments Results

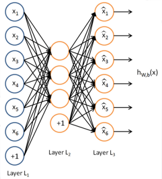


Figure: Autoencoder

Each neuron in output layer compute the following:

$$z^3 = W^2 a^2 + b^2 \tag{13}$$

$$\hat{x} = a^3 = f(z^3) \tag{14}$$

Limitation:Input must be scaled in the range $[0, 1]$.

Linear decoder:modify the activation function of output layer into an identity one,leaving the others unchanged.

$$\hat{x} = a^3 = z^3 = W^2 a^2 + b^2 \tag{15}$$

Advantage:Real-valued inputs without pre-scale can be processed!
We only need to change the error terms of output layer:

$$\delta_i^3 = \frac{\partial}{\partial z_i} \frac{1}{2} \|y - \hat{x}\|^2 = (\hat{x}_i - y_i) \cdot f'(z_i^3) = (\hat{x}_i - y_i) \tag{16}$$

Experiments on Linear Decoder

Dataset: $100,000$ small 8×8 patches sampled from STL-10 color images.

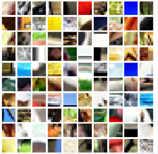


Figure: Training Samples

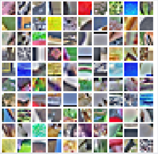


Figure: PCA-Whitened Samples

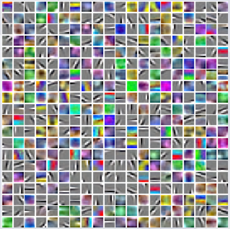


Figure: Learned Features(Time Consumed: 2305sec)

Self-Taught Learning

An aphorism in machine learning,"sometimes it's not who has the best algorithm than wins;it's who has the most data."

Self-Taught Learning:exploit massive unlabeled data effectively to learn good feature representations of input.

For classification task,we will combine feature representations extracted from labeled data and their labels to train a classifier.

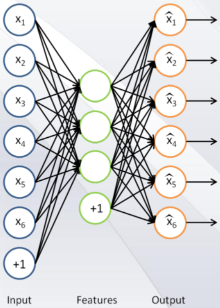
Self-Taught Learning and Semi-Supervised Learning

- *Self-Taught Learning*
  Unlabeled and labeled data may come from different distributions.
- *Semi-Supervised Learning*
  Unlabeled and labeled data comes from exactly the same distribution.

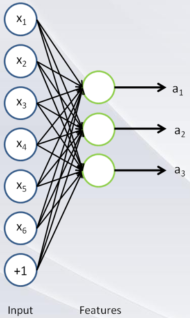
Apparently,Self-Taught Learning is more broadly applicable.

Learning Features

Train a sparse autoencoder on unlabeled data to get parameters W^1, b^1, W^2, b^2:



Given any input x,compute the corresponding vector of activations a of hidden units:



Next,for a labeled sample (x_l, y_l),we can use (a_l, y_l) or $((x_l, a_l), y_l)$ to represent it.

Experiments on Self-Taught Learning

Learned Features+Softmax Classifier

Dataset	Dimensionality	Number of Samples
Unlabeled Data($5 \sim 10$)	784	29404
Training Data($0 \sim 4$)	784	15298
Testing Data($0 \sim 4$)	784	15298

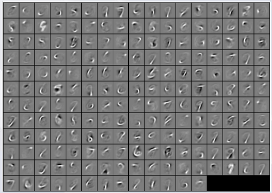


Figure: Learned Features on unlabeled data

Accuray:98.3%,Time Consumed:6064sec