Introduction

Deep learning is usually applied in supervised learning. For small or mid-sized data set, deep learning is normally not necessary as traditional machine learning algorithms can also provide comparable results but with efficient calculation. Deep learning can provide much higher quality result as the data size becomes larger and larger and the problem is with strong nonlinearity. Deep learning need more intensive calculation and need to handle local minimum.

Logistic regression derived from perspective of a neural network unit

The following derivation with minimization of the cost is essentially the same as the derivation using maximum likelihood estimation, as there is only a sign difference between them. However, the derivation below can shed some light on the relation of logistic regression to a single unit neural network. This also makes the neural network and the way to train a new work (i.e. the forward and backward propagations) less mysterious. Moreover, the introduction of vectorization of variables to replace loops in code, can often significantly improve calculation efficiency.

Logistic regression formalism

z denotes the θ introduced earlier and is separated into w and b. For one example $x^{(i)}$:

$$z^{(i)} = w^T x^{(i)} + b$$

$$\hat{y}^{(i)} = a^{(i)} = \frac{1}{1 + e^{-z^{(i)}}}$$

$$J = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(a^{(i)}, y^{(i)})$$

$$\mathcal{L}(a^{(i)}, y^{(i)}) = -y^{(i)} \log(a^{(i)}) - (1 - y^{(i)}) \log(1 - a^{(i)})$$

$$dw_j \equiv \frac{\partial J}{\partial w_j} = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial w_j} = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial a^{(i)}} \frac{\partial a^{(i)}}{\partial z^{(i)}} \frac{\partial z^{(i)}}{\partial w_j} = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial a^{(i)}} \frac{\partial a^{(i)}}{\partial z^{(i)}} \frac{\partial}{\partial w_j} \left(\sum_{k=1}^{n} w_k x_k^{(i)} \right)$$

$$= \frac{1}{m} \sum_{i=1}^{m} \frac{\partial \mathcal{L}}{\partial a^{(i)}} \frac{\partial a^{(i)}}{\partial z^{(i)}} \left(\sum_{k=1}^{n} x_k^{(i)} \delta_{jk} \right) = \frac{1}{m} \sum_{i=1}^{m} \left(\frac{-y^{(i)}}{a^{(i)}} + \frac{1 - y^{(i)}}{1 - a^{(i)}} \right) a^{(i)}(1 - a^{(i)}) x_j^{(i)} = \frac{1}{m} \sum_{i=1}^{m} (a^{(i)} - y^{(i)}) x_j^{(i)}$$

Thus we have

$$dw_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}(a^{(i)} - y^{(i)}),$$

where the (i) in $x_j^{(i)}$ above is column index. This should be noted when we vectorize the above element-wise expression. Following the similar procedure, we have

$$\frac{\partial J}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(a^{(i)} - y^{(i)})$$

Vectorized forward propagation

$$A = \sigma(w^T X + b),$$

where X is constructed from input features in terms of $n \times m$ matrix, and m is the number of experimental samples.

$$J = -\frac{1}{m}\text{sum}\left(Y\log(A) + (1-Y)\log(1-A)\right),$$

where sum denotes vector addition.

Vectorized backward propagation ¶

$$dw = \frac{\partial J}{\partial w} = \frac{1}{m}X(A-Y)$$
$$db = \frac{\partial J}{\partial b} = \frac{1}{m}\text{sum}(A-Y)$$
$$w = w - \alpha\frac{\partial J}{\partial w}, \quad b = b - \alpha\frac{\partial J}{\partial b}$$

If Y is reshaped to a 2D matrix form $(1, m)$, then a transpose sign for $(A-Y)$ is necessary. After many rounds of forward and backward propagations, the converged A is just the predictions. Using the chain rule for calculating derivatives, we can intuitively understand the backward propagation. The backward propagation, though can be understood with many steps in the chain rules of calculating derivatives, it can sometimes be implemented with only one or a few steps. See detailed implementation in other notes about deep learning, where regularization is also introduced.

Connecting generalized linear models (GLMs) to neural network

See details about GLMs in the notes "generative vs discriminative algorithms_GLM_GDA.pdf" under the folder machineLearning/fundamentals.

- In either GLMs or neural network, the same linear relation connecting parameters and input is used, i.e., $w^T x + b$ or $\theta^T x$ when bias is absorbed to θ.
- In either GLMs or neural network, the $w^T x + b$ will be used as an argument of a function to predict result. In GLMs, the function is called response function, while in neural network it is called activation function. But the purpose of these functions are same.
- In GLMs, the response function is obtained from the hypothesis function $h_\theta(x) = E[y \mid x; \theta]$. For $y \mid x \in \text{Bernoulli}$, $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$, and hence response function is a logistic or sigmoid function. For $y \mid x \in \text{Gaussian}$, $h_\theta(x) = \theta^T x$, and hence response function is an identity function.
- In GLMs, if $y \mid x$ belongs to an exponential family of probability distributions such as Gaussian, Bernoulli, Poisson, gamma, etc, then we will obtain a linear model (learns a linear decision boundary in classification for example), even the response might be a nonlinear function.
- If we regard logistic regression as a special one-layer (only output layer) neural network, then it is linear model, even though the activation function is nonlinear. This is because this activation can be obtained by the Bernoulli distribution in GLMs which belongs to the exponential family. **Note if activation function is not used in a single-output layer, then the neural network is no longer a linear model.**
- In neural network, activation function can be chosen in many different nonlinear functions such as relu, tanh, etc. These do not correspond to any response functions in GLMs, which can be obtained from a family of exponential probability distributions. However, we can also regard neural network as probabilistic models but with very complicated unknown distributions.
- Neural network is thus intrinsically a nonlinear model. Moreover, unlike logistic regression, or other algorithms in GLMs, or all probabilistic models in general, neural network and particularly deep neural network, can 'extract' many new features as the number of hidden layers become large. Thus it can learn very complicated, nonlinear problems.