TTIC 31230, Fundamentals of Deep Learning

David McAllester, Winter 2018

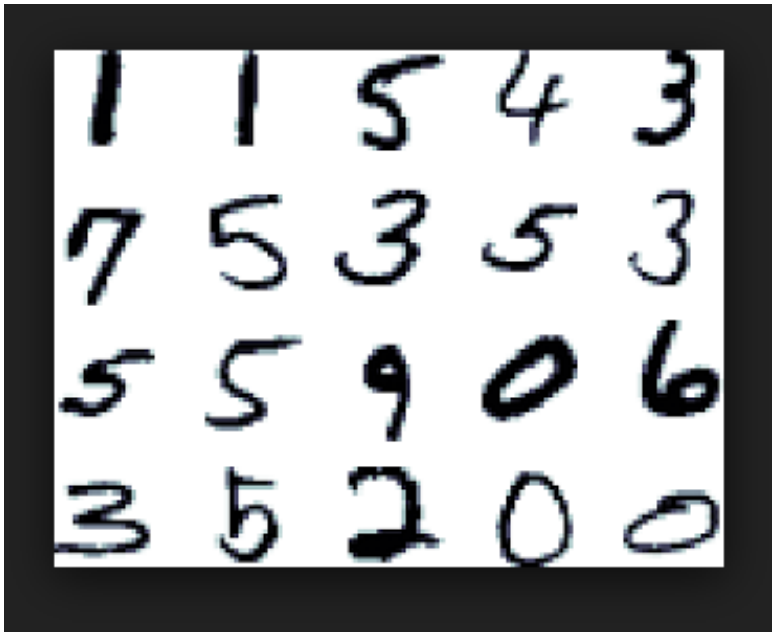
Multiclass Logistic Regression

Multilayer Perceptrons (MLPs)

Stochastic Gradient Descent (SGD)

Multiclass Classification

We consider the problem of taking an input x (such as an image of a hand written digit) and classifying it into some small number of classes (such as the digits 0 through 9).



Multiclass Classification

Assume a population distribution on pairs (x, y) for $x \in \mathbb{R}^d$ and $y \in \mathcal{C}$.

For MNIST x is a 28×28 image which we take to be a 784 dimensional vector giving $x \in \mathbb{R}^{784}$.

For MNIST $\mathcal{C}$ is the set $\{0, \ldots, 9\}$.

Assume a sample $(x_0, y_0), \ \ldots, \ (x_{N-1}, y_{N-1})$ drawn IID from the population.

We want to use the sample to construct a rule for predicting y given x when we draw new pairs from the population.

Multiclass Logistic Regression

Assume a sample $(x_0, y_0), \ \ldots, \ (x_{N-1}, y_{N-1})$ drawn IID from the population with $x \in \mathbb{R}^d$ and $y \in \{0, \ldots, K\}$.

For a new x we compute a score $s(\hat{y})$ for each possible label $\hat{y}$.

$$s = Wx + b$$

Multiclass Logistic Regression for MNIST

j — image pixel

$\hat{y}$ — possible image label (0 through 9)

$$s(\hat{y}) = \sum_j W[\hat{y}, j]\ x[j] + b[\hat{y}]$$

Note that $W[\hat{y}, :]$ is an image.

Softmax

Softmax converts scores (or energies or logits) to probabilities.

$$Q(\hat{y}) = \frac{1}{Z} e^{s(\hat{y})}$$

$$Z = \sum_{\hat{y}} e^{s(\hat{y})}$$

In vector notation

$$Q = \text{softmax } s$$

Log Loss and Logistic Regression

Let $Q_\Phi(\hat{y}|x)$ be defined by a model with parammeters Φ.

In logistic regression Φ is the pair (W, b).

Let n range over training instances.

$$W^*, b^* = \underset{W,b}{\text{argmin}} \frac{1}{N} \sum_{n=1}^{N} -\log Q_{W,b}(y_n|x_n)$$

$$\Phi^* = \underset{\Phi}{\text{argmin}} \frac{1}{N} \sum_{n=1}^{N} -\log Q_{\Phi}(y_n|x_n)$$

Information Theoretic Formulation

Let Φ be the parameters of a probabilistc predictor Q_Φ.

We want $\Phi^* = \text{argmin}_\Phi \; E_{(x,y)\sim\mathcal{P}} \; -\log Q_\Phi(y|x)$.

This is **cross-entropy** loss:

$$H(\mathcal{P}, \mathcal{Q}) = E_{y\sim\mathcal{P}} \; -\log \; \mathcal{Q}(y)$$

$$H(\mathcal{P}) = H(\mathcal{P}, \mathcal{P}) = E_{y\sim\mathcal{P}} \; -\log \; \mathcal{P}(y)$$

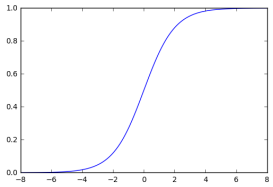
$$H(\mathcal{P}, \mathcal{Q}) \geq H(\mathcal{P})$$

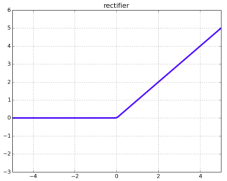
$$E_{(x,y)\sim\mathcal{P}} \; -\log Q_\Phi(y|x) = E_{x\sim\mathcal{P}} \; H(\mathcal{P}(:\,|x), Q_\Phi(:\,|x))$$

Multi Layer Perceptrons (MLPs)

Activation functions:

$$\sigma(u) = \frac{1}{1+e^{-u}} \qquad \text{Relu}(u) = \max(u, 0)$$





$$L^0 = \text{Relu}(W^0 x + b^0)$$

$$L^1 = \sigma(W^1 L^0 + b^1)$$

$$Q_\Phi = \text{softmax}(L^1)$$

Explicit Index Notation with *Typed Index Variables*

i — pixels

j — image features

$\hat{y}$ — possible image labels

$$L^0[j] = \text{Relu}\left(\left(\sum_i W^0[j,i]\; x[i]\right) + b^0[j]\right)$$

$$L^1[\hat{y}] = \sigma\left(\left(\sum_j W^1[\hat{y},j]\; L^0[j]\right) + b^1[\hat{y}]\right)$$

$$Q_\Phi(\hat{y}) = \frac{1}{Z}\, e^{L^1[\hat{y}]}$$

Loss Vs. Error Rate

While training (gradient descent) is generally done on log loss, performance is often judged by other measures such as error rate.

The "loss" is often used as a synonym for log loss (or whatever loss defined the gradient descent training).

Hence one often reports both "loss" and "error rate".

Note that error rate is not differentiable.

Train Data, Development Data and Test Data

Data is typically divided into **a training set**, **a development set** and **a test set** each drawn IID from the population.

A learning algorithm optimizes training loss.

One then optimizes algorithm design (and hyper-parameters) on the development set. (graduate student descent).

Ultimate performance should be done on a test set not used for development. Test data is often withheld from developers.

Gradients with Respect to Systems of Parameters

$\nabla_\Phi \ell(\Phi, x, y)$ denotes the partial derivative of $\ell(\Phi, x, y)$ with respect to the parameter system Φ.

Here can think of Φ as a single vector with

$$(\nabla_\Phi \ell(\Phi, x, y))_i = \partial \ell(\Phi, x, y)/\partial \Phi_i$$

But in general Φ can be a multi-dimensional array (an ndarray in NumPy). If Φ is four dimensional we can write $\Phi[i, j, k, l]$.

For scalar loss, $\nabla_\Phi \ell(\Phi, x, y)$ has the same shape as Φ.

$$(\nabla_\Phi \ell(\Phi, x, y)) .\text{shape} = \Phi.\text{shape}$$

Total Gradient Descent

$$\ell_{\text{train}}(\Phi) = \frac{1}{N} \sum_n \ell(\Phi, x_n, y_n)$$

We want: $\Phi^* = \text{argmin}_\Phi \, \ell_{\text{train}}(\Phi)$

$$\Phi \quad \texttt{-=} \quad \eta \nabla_\Phi \ell_{\text{train}}(\Phi)$$

Stochastic Gradient Descent (SGD) on the training set.

repeat: Select n at random. Φ `-=` $\eta \, \nabla_\Phi \, \ell(\Phi, x_n, y_n)$

$$E_n \, \nabla_\Phi \, \ell(\Phi, x_n, y_n) = \sum_n P(n) \nabla_\Phi \, \ell_{\text{train}}(\Phi, x_n, y_n)$$

$$= \frac{1}{N} \sum_n \nabla_\Phi \, \ell_{\text{train}}(\Phi, x_n, y_n)$$

$$= \nabla_\Phi \frac{1}{N} \sum_n \, \ell_{\text{train}}(\Phi, x_n, y_n)$$

$$= \nabla_\Phi \, \ell_{\text{train}}(\Phi)$$

Epochs

In practice we cycle through the training data visiting each training pair once.

One pass through the training data is called an Epoch.

One typically imposes a random suffle of the training data before each epoch.

SGD for MLPs

j: feature $\quad$ ℓ: layer $\quad$ $\hat{y}$: possible label

$$L^0[j] = \text{Input}$$

$$L^{\ell+1}[j'] = \sigma\left(\left(\sum_j W^{\ell+1}[j', j]\, L^{\ell+1}[j]\right) + b^{\ell+1}[j']\right)$$

$$L^N[\hat{y}] = \sigma\left(\left(\sum_j W^{N-1}[\hat{y}, j]\, L^{N-1}[j]\right) + b^{N-1}[\hat{y}]\right)$$

$$P(\hat{y}) = \frac{1}{Z}\, e^{L^N[\hat{y}]}$$

END