S&DS 265 / 565

Introductory Machine Learning

Stochastic Gradient Descent

September 22

Goings on

- Assn 1 due next week
- Assn 2 will be posted by same time
- Quiz 2 is posted after class
- Due in 48 hours; 20 mins once started
- Questions?

Outline for today

- Stochastic gradient descent
- Application to logistic regression
- Regularization
- Learning rate and scaling
- Jupyter notebook example

Stochastic gradient descent

- Suppose that we want to fit a really big model, where the number of samples n and number of variables p are very large
- The classical algorithms in standard software packages will fail
- How can we train such models?

Example

- We want to classify ads according to whether or not they will be clicked on by a user
- We have a very large collection of training data
- Ads are represented in terms of a sparse list of features

$$1 \mid 5 : 1.1789641e{-}01 \quad 39 : 6.0373064e{-}02 \quad 45 : 1.3163488e{-}01$$

- The dataset is too large to load into memory, and the number of features is also very large
- New data are continually arriving
- How can we efficiently train a classifier?

Online learning

We will introduce a method that

- Reads in the data points one (or a few) at a time
- Updates the model for each sample
- Exploits sparsity of the features
- Uses little memory, never reads in the entire dataset

Stochastic gradient descent

Initialize all parameters to zero: $\beta_j = 0$, $j = 1, \ldots, p$.

Read through the data one record at a time, and update the model.

1. Read data item x
2. Make a prediction $\widehat{y}(x)$
3. Observe the true response/label y
4. Update the parameters β so $\widehat{y}$ is closer to y

Stochastic gradient descent

To begin, suppose we are doing *linear regression*. We initialize all parameters to zero: $\beta_j = 0$, $j = 1, \ldots, p$.

We read through the data one record at a time, and update the model.

1. Read data item x
2. Make a prediction $\widehat{y}(x) = \sum_{j=1}^{p} \beta_j x_j$
3. Observe the true response/label y
4. Update the parameters β so $\widehat{y}$ is closer to y

SGD idea

Here's the idea:

- For each parameter β_j, see what happens to the loss if that parameter is increased a little bit.
- If the loss goes down (up), then increase (decrease) β_j proportionately
- Do this simultaneously for all of the parameters
- Rinse and repeat

SGD idea

Change β_j by a little bit:

$$\beta_j \to \beta_j + \varepsilon$$

What happens to the squared error?

$$\begin{aligned}(y - \widehat{y})^2 &\to (y - \widehat{y} - \varepsilon x_j)^2 \\ &\approx (y - \widehat{y})^2 - 2(y - \widehat{y})\varepsilon x_j\end{aligned}$$

SGD idea

We then change the parameter as follows:

$$\beta_j \to \beta_j + \underbrace{2\eta(y - \widehat{y})x_j}_{\varepsilon}$$

SGD idea

Why does this work? With this choice of ε the squared error decreases:

$$\begin{aligned}(y-\widehat{y})^2 &\to (y-\widehat{y}-\varepsilon x_j)^2\\ &\approx (y-\widehat{y})^2 - 2(y-\widehat{y})\varepsilon x_j\\ &= (y-\widehat{y})^2 - 4\eta(y-\widehat{y})^2 x_j^2\\ &< (y-\widehat{y})^2\end{aligned}$$

so we're moving "downhill"

SGD for general loss

Suppose $L(y, \beta^T x)$ is the loss for an input (x, y), e.g., $(y - \beta^T x)^2$

SGD update:

$$\beta_j \longleftarrow \beta_j - \eta \frac{\partial L(y, \boldsymbol{\beta}^T x)}{\partial \beta_j}$$

$$\boldsymbol{\beta} \longleftarrow \boldsymbol{\beta} - \eta \nabla_{\beta} L(y, \boldsymbol{\beta}^T x) \quad \text{(vector notation)}$$

- η is the *learning rate* or “step size”
- Needs to be chosen carefully, getting smaller over time

Gradient descent for general loss

If $L(\beta)$ is the loss function over subset of training set:

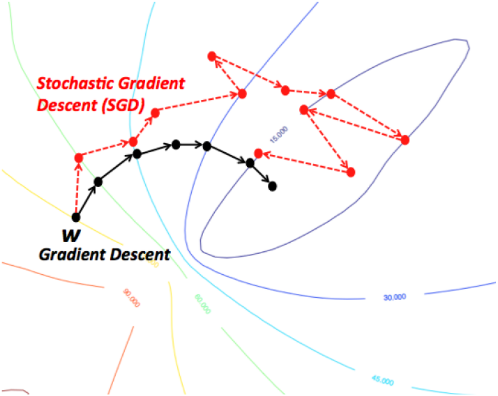
$$L(\beta + \eta v) \approx L(\beta) + \eta v^T \nabla L(\beta)$$
$$L(\beta - \eta \nabla L(\beta)) \approx L(\beta) - \eta \|\nabla L(\beta)\|^2$$

This is why gradient descent is going downhill — if η is small enough.

"Batch" gradient descent uses the entire training set in each step of gradient descent.

Stochastic gradient descent computes a quick approximation to this gradient, using only a single or a small "mini-batch" of data points

Batch vs. stochastic gradient descent



https://wikidocs.net/3413

SGD for logistic regression

SGD Update:

$$\beta_j \longleftarrow \beta_j + \eta(y - p(x))x_j$$

$$\beta_j x_j \longleftarrow \beta_j x_j + \eta(y - p(x))x_j^2$$

$$p(x) = \frac{1}{1 + \exp(-\beta^T x)}$$

Case checking:

- Suppose $y = 1$ and probability $p(x)$ is high?

SGD for logistic regression

SGD Update:

$$\beta_j \longleftarrow \beta_j + \eta(y - p(x))x_j$$

$$\beta_j x_j \longleftarrow \beta_j x_j + \eta(y - p(x))x_j^2$$

$$p(x) = \frac{1}{1 + \exp(-\beta^T x)}$$

Case checking:

- Suppose $y = 1$ and probability $p(x)$ is high? *small change*
- Suppose $y = 1$ and probability $p(x)$ is small?

SGD for logistic regression

SGD Update:

$$\beta_j \longleftarrow \beta_j + \eta(y - p(x))x_j$$

$$\beta_j x_j \longleftarrow \beta_j x_j + \eta(y - p(x))x_j^2$$

$$p(x) = \frac{1}{1 + \exp(-\beta^T x)}$$

Case checking:

- Suppose $y = 1$ and probability $p(x)$ is high? *small change*
- Suppose $y = 1$ and probability $p(x)$ is small? *big change* $\uparrow$
- Suppose $y = 0$ and probability $p(x)$ is small?

SGD for logistic regression

SGD Update:

$$\beta_j \longleftarrow \beta_j + \eta(y - p(x))x_j$$

$$\beta_j x_j \longleftarrow \beta_j x_j + \eta(y - p(x))x_j^2$$

$$p(x) = \frac{1}{1 + \exp(-\boldsymbol{\beta}^T x)}$$

Case checking:

- Suppose $y = 1$ and probability $p(x)$ is high? *small change*
- Suppose $y = 1$ and probability $p(x)$ is small? *big change* $\uparrow$
- Suppose $y = 0$ and probability $p(x)$ is small? *small change*
- Suppose $y = 0$ and probability $p(x)$ is big?

SGD for logistic regression

SGD Update:

$$\beta_j \longleftarrow \beta_j + \eta(y - p(x))x_j$$

$$\beta_j x_j \longleftarrow \beta_j x_j + \eta(y - p(x))x_j^2$$

$$p(x) = \frac{1}{1 + \exp(-\beta^T x)}$$

Case checking:

- Suppose $y = 1$ and probability $p(x)$ is high? *small change*
- Suppose $y = 1$ and probability $p(x)$ is small? *big change* $\uparrow$
- Suppose $y = 0$ and probability $p(x)$ is small? *small change*
- Suppose $y = 0$ and probability $p(x)$ is big? *big change* $\downarrow$

SGD: choice of learning rate

A conservative choice of learning rate is

$$\eta_t = \frac{1}{t}$$

A more agressive choice is

$$\eta_t = \frac{1}{\sqrt{t}}$$

In practice: Try learning rates $C/\sqrt{t}$ for different choices of C, and monitor the error

$$\frac{1}{T}\sum_{t=1}^{T}(Y_t - \widehat{Y}_t)^2$$

Demo

Open the demo notebook `sgd.ipynb` and follow along...

SGD: choice of learning rate

Learning rate should scale as

$$\eta_t = \frac{1}{\sqrt{t}}$$

Problem: Some of the updates may be on different scales.

SGD: choice of learning rate

Learning rate should scale as

$$\eta_t = \frac{1}{\sqrt{t}}$$

Problem: Some of the updates may be on different scales.

Solution: Let $g_{tj} = \dfrac{\partial L(y_t, \beta^T x_t)}{\partial \beta_j}$

Scale gradients to get update rule

$$\beta_j \longleftarrow \beta_j - \eta \frac{g_{tj}}{\sqrt{\sum_{s=1}^{t} g_{sj}^2}}$$

SGD: scaling issues

For a linear model, the SGD update is

$$\beta_j \longleftarrow \beta_j - C_t x_j$$

If x_j increases by a factor of two, the parameter β_j should decrease by a factor of two.

This update doesn't respect that scaling

SGD: scaling issues

Usual solution is to "standardize" each variable — subtract out the mean and divide by the standard deviation

$$x_j \leftarrow \frac{x_j - \text{mean}(x_j)}{\sqrt{\text{var}(x_j)}}$$

But this involves "looking ahead" to compute the mean and variance, and destroys the online property of the algorithm

SGD: scaling issues

Usual solution is to "standardize" each variable — subtract out the mean and divide by the standard deviation

$$x_j \leftarrow \frac{x_j - \text{mean}(x_j)}{\sqrt{\text{var}(x_j)}}$$

But this involves "looking ahead" to compute the mean and variance, and destroys the online property of the algorithm

Solution: The mean and variance can be updated in an online manner, in constant time, by storing auxiliary variables for each component j.

SGD: Regularization

A "ridge" penalty $\lambda \sum_{j=1}^{p} \beta_j^2$ is easily handled.

Gradient changes by an additive term $2\lambda\beta$. Update becomes

$$
\begin{aligned}
\beta_j &\longleftarrow \beta_j + \eta\{(y - p(x))x_j - 2\lambda\beta_j\} \\
&= (1 - 2\eta\lambda)\beta_j + \eta(y - p(x))x_j
\end{aligned}
$$

Observe that this "does the right thing" whether β_j wants to be large positive or negative.

- *The penalty shrinks β_j toward zero*

What did we learn today?

- Stochastic gradient descent is a simple algorithm that can be applied to large classification and regression problems
- A parameter is updated according to how much the loss changes when that parameter is changed by a little bit
- This is the “go to” algorithm for fitting large or complex machine learning models
- Choosing the learning rate is a little tricky