5.9 Stochastic Gradient Descent

Tuesday, June 12, 2018 9:56 PM

Stochastic gradient descent: powers nearly all deep learning

gradient descent (4.3) (Cauchy, 1847)

to minimize $f(x)$, use first derivative of $f'(x)$:

$f(x - \epsilon \operatorname{sign}(f'(x)))$ must be $< f(x)$ for small enough ϵ

if $f'(x) = 0$, no useful gradient: "stationary points"

local min, local max, or neither ("saddle point")

directional derivative: in direction u, slope of f in direction u.

= derivative of $f(x + \alpha u)$ with respect to α (at $\alpha = 0$).

$= u^T \nabla_x f(x)$

vector of all partial derivatives (in each input variable)

"steepest descent" - gradient descent:

$x' = x - \epsilon \nabla_x f(x)$ move in direction that decreases fastest

learning rate = size of step

hill climbing - generalizes gradient descent (continuous spaces) to discrete parameters.

Jacobian matrix - all partial derivatives (for functions whose inputs & outputs are both vectors)

Hessian matrix - matrix of second derivatives

$H(f)(x)_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(x)$ Jacobian of the gradient

[resume 5.9]

cost functions often decompose to sum over training examples of per-example loss

$$J(\theta) = \mathbb{E}_{x,y \sim \hat{p}_{data}} L(x, y, \theta) = \frac{1}{m} \sum_{i=1}^{m} L(x^{(i)}, y^{(i)}, \theta)$$

per-example loss $= -\log p(y|x; \theta)$

gradient descent requires computing

$$\nabla_\theta J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \nabla_\theta L(x^{(i)}, y^{(i)}; \theta)$$

↳ cost of each gradient step in $O(m)$, scales linearly with # of training samples

$i=1$

↳ cost of each gradient step is $\Omega(m)$, scales linearly with # of training samples

SGD insight: gradient is an expectation - can approximate with small set of samples

minibatch - uniformly draw m' samples for each step

$\mathbb{B} = \{x^{(1)}, \dots, x^{(m')}\}$ m' doesn't need to scale with training set size m

provides a scalable way to train on large datasets

(how often is ∇ from $\mathbb{B}$ wrong?)

(5.10) Building a Machine Learning Algorithm

dataset + cost function + optimization procedure + model

linear regression:

X, y $\quad J(w,b) = -\mathbb{E}_{x,y\sim\hat{p}_{data}} \log p_{model}(y|x)$ solve for gradient 0 $\quad p_{model}(y|x) = \mathcal{N}(y; x^T w + b, 1)$

can modify any of these for new ML algorithms

cost function:

most common: negative log likelihood (minizing → maximum likelihood est.)

+ regularization terms

model:

linear: can solve in closed form

non-linear: iterative optimization procedure (e.g., gradient descent)

PCA: loss function, $J(w) = \mathbb{E}_{x\sim\hat{p}_{data}} \| x - r(x; w) \|_2^2$

model: w with norm one, reconstruction $r(x) = w^T x w$

special case optimizes -

decision trees, k-means clustering

(5.11) Challenges that Motivate Deep Learning

traditional ML algorithms couldn't do well enough on many tasks:

object recognition, speech, etc.

challenge of generalizing to new examples increases exponentially with dimensions

(5.11.1) Curse of Dimensionality

(5.11.1) Curse of Dimensionality

# of configurations is exponential in # of variables

high-dimensional space ⇒ no training example near most points

(5.11.2) Local Constancy + Smoothness Regularization

prior beliefs about what function to learn

Smoothness prior (or local constancy prior) - function shouldn't change much within a small region

$f^*(x) \approx f^*(x+\epsilon)$ encourage learning f with this property

k nearest neighbors assume this

local kernels $k(u,v)$ is large for $u=v$, decreases as u & v separate

template matching

k samples to distinguish k regions (why does book use $O(k)$?)

is there a way to have more regions than training examples?

can define $O(2^k)$ regions with $O(k)$ examples - need to introduce dependencies between regions through additional assumptions about data-generating distribution.

or, task-specific assumptions

Core idea of deep learning:

assume data was generated by the composition of factors (features)

potentially at multiple levels of hierarchy

(5.11.3) Manifold Learning

manifold - connected region

set of points associated with neighborhood around each point

for each point, locally appears to be Euclidean space

surface of world is spherical manifold in 3D space! (appears 2D)

manifold learning - assume most of $\mathbb{R}^n$ is invalid inputs

interesting inputs are only on a collection of manifolds

small subset of points

manifold hypothesis

small subset of points

manifold hypothesis - much real world data lies along low-dim. manifolds
- random noise doesn't look like an image
- experiments support this assumption