Deep Learning I: Gradient Descent

Hinrich Schütze

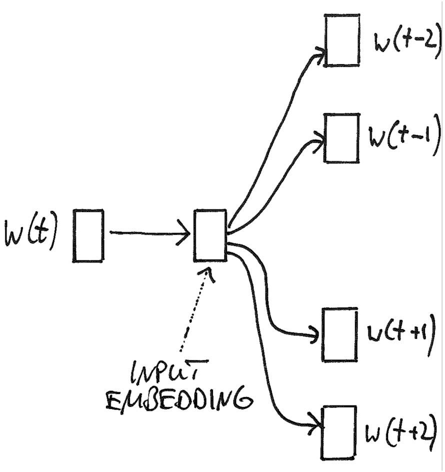
Center for Information and Language Processing, LMU Munich

2017-07-19

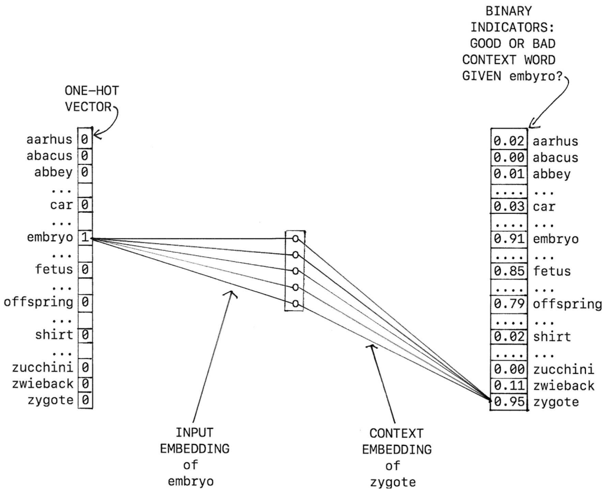
Overview

Outline

word2vec skipgram
predict, based on input word, a context word



word2vec skipgram
predict, based on input word, a context word



word2vec parameter estimation: Historical development vs. presentation in this lecture

- Mikolov et al. (2013) introduce word2vec, estimating parameters by gradient descent.
  (today)
  - Still the learning algorithm used by default and in most cases
- Levy and Goldberg (2014) show near-equivalence to a particular type of matrix factorization.
  (yesterday)
  - Important because it links two important bodies of research: neural networks and distributional semantics

Gradient descent (GD)

- Gradient descent is a learning algorithm.
- Given:
  - a hypothesis space (or model family)
  - an objective function or cost function
  - a training set
- Gradient descent (GD) finds a set of parameters, i.e., a member of the hypothesis space (or specified model) that performs well on the objective for the training set.
- GD is arguably the most important learning algorithm, notably in deep learning.

Simple learning problem → word2vec

- Gradient descent for housing prices
- Gradient descent for word2vec skipgram

Inverted Classroom
Andrew Ng: "Machine Learning"
http://coursera.org

Outline

Intro, model, cost: Andrew Ng videos

- Introduction
  - Supervised learning
- Model and cost function
  - Model representation
  - Cost function
  - Cost function, intuition 1
  - Cost function, intuition 2

Housing prices in Portland

input variable **x** size (feet^2)	output variable **y** price ($) in 1000s
2104	460
1416	232
1534	315
852	178

We will use m for the number of training examples.

Setup to learn a housing price predictor using GD
Next: Setup for word2vec skipgram

- Hypothesis:

$$h_\theta = \theta_0 + \theta_1 x$$

- Parameters:

$$\theta = (\theta_0, \theta_1)$$

- Cost function:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

- Objective: $\text{minimize}_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

graph: hypothesis, parameters

Basic idea: gradient descent finds "close" hypothesis

Choose θ_0, θ_1
so that, for our training examples (x, y),
$h_\theta(x)$ is close to y.

Cost as a function of θ_1 (for $\theta_0 = 0$)

- Left:
  housing price y as a function of square footage x
- Right:
  cost $J(\theta_1)$ as a function of θ_1

one-variable case

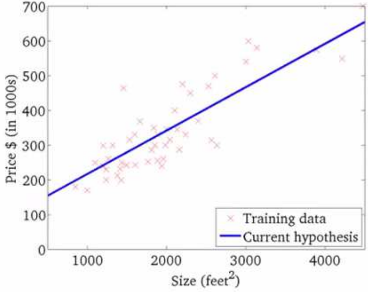
Cost as a function of θ_0, θ_1

- Left:
  housing price y as a function of square footage x
- Right: contour plot
  cost $J(\theta_0, \theta_1)$ as a function of (θ_0, θ_1)

Hypothesis (left)
Point corresponding to its cost in contour (right)

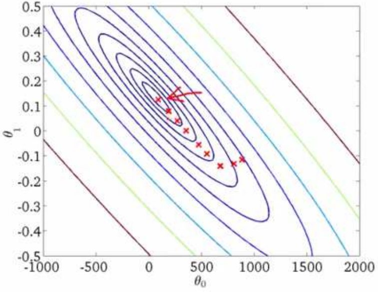
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)

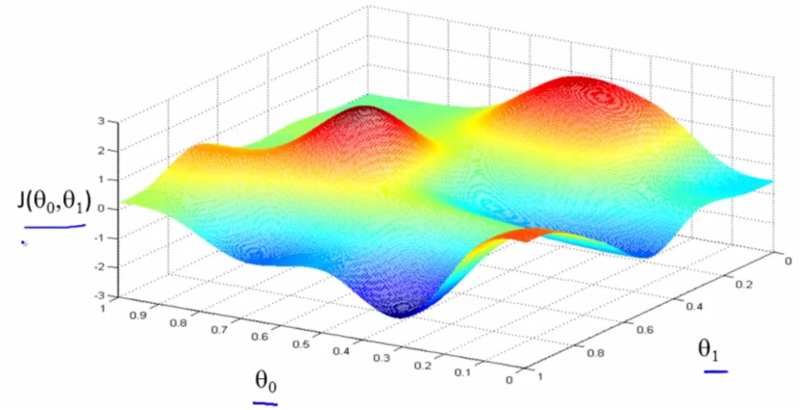


$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



Surface plot (instead of contour plot)



Outline

Gradient descent: Andrew Ng videos

- Gradient descent
- Gradient descent intuition
- Gradient descent for linear regression

Gradient descent: Basic idea

- Start with some values of parameters, e.g., $\theta_0 = 0.3$, $\theta_1 = -2$
- (These initial values are randomly chosen.)
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$
- Hopefully, we will end up at a minimum of $J(\theta_0, \theta_1)$.
- "keep changing": how exactly do we do that?

Gradient descent: One step (one variable)

Repeat until convergence

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

Or: Repeat until convergence

$$\theta_1 := \theta_1 - \alpha J'(\theta_1)$$

α is the learning rate.

positive/negative slope

Exercise: Gradient descent

- Suppose you have arrived at a point where the gradient is 0
- Draw an example of this situation
- Mark the current point at which the gradient is zero and the point that gradient descent will move to next

Gradient descent: One step (>1 variables)

Repeat until convergence

$$\theta_0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \ldots, \theta_k)$$

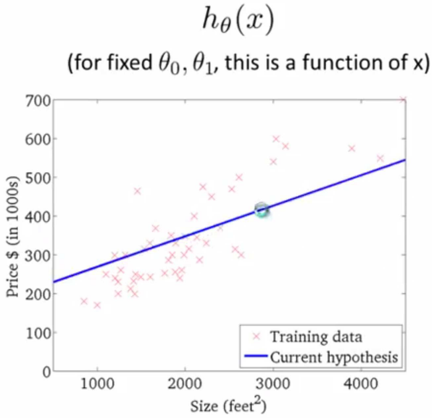
$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \ldots, \theta_k)$$

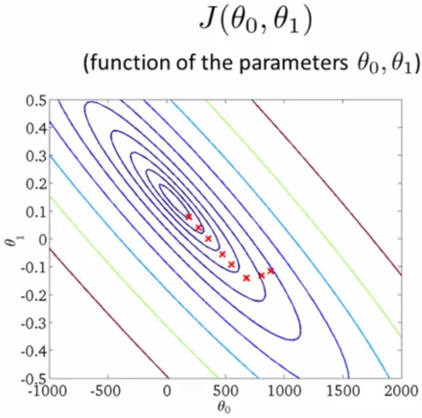
$$\ldots$$

$$\theta_k := \theta_k - \alpha \frac{\partial}{\partial \theta_k} J(\theta_0, \ldots, \theta_k)$$

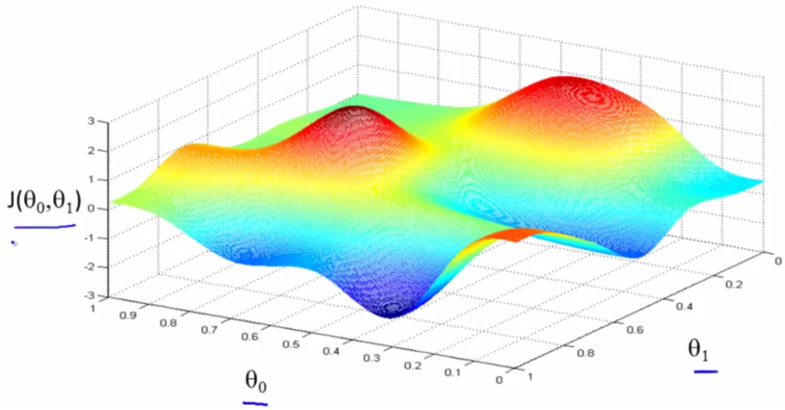
α is the learning rate.

Path down to minimum: Two parameters





Surface plot (instead of contour plot)



Gradient descent: One step (regression)

Repeat until convergence

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} [(h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}]$$

α is the learning rate.

derivatives

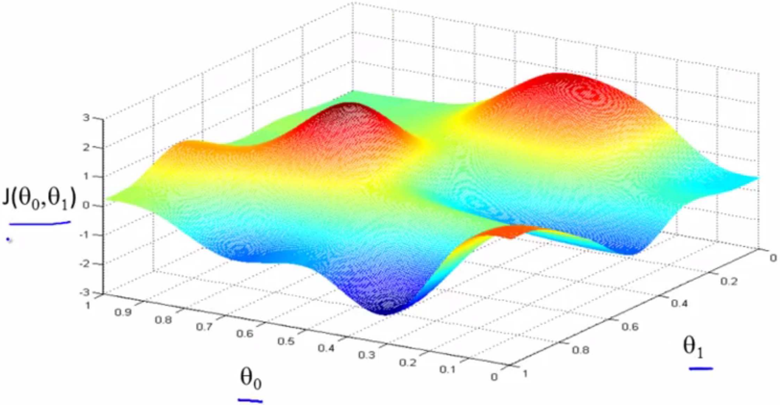
Learning rate α

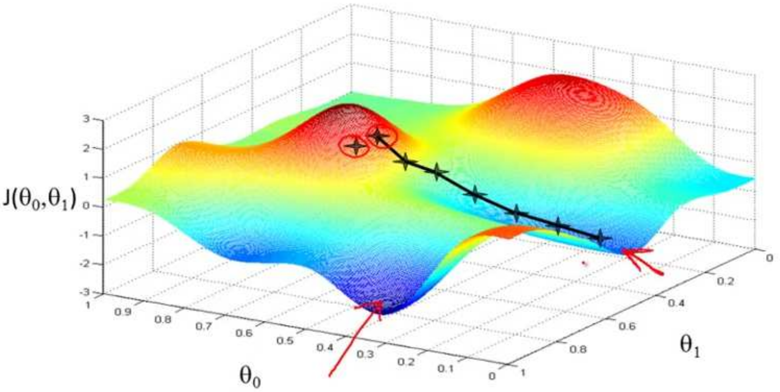
- If α is too small:
  gradient descent is very slow.
- If α is too large:
  gradient descent overshoots, doesn't converge or (!) diverges.

divergence

Different minima: Exercise

Depending on the starting point, we can arrive at different minima here. Find two starting points that will result in different minima.





$J(\theta_0,\theta_1)$
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ_0
0
0.2
0.4
0.6
0.8
1
θ_1

Gradient descent: batch vs stochastic

- So far: batch gradient descent
- Recall cost function:
  $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$
- We sum over the entire training set ...
- ... and compute the gradient for the entire traning set.

Stochastic gradient descent

- Instead of computing the gradient for the entire traning set,
- we compute it for one training example
- or for a minibatch of k training examples.
- E.g., $k \in \{10, 50, 100, 500, 1000\}$
- We usually add randomization,
- e.g., shuffling the training set.
- This is called stochastic gradient descent.

Gradient descent: stochastic vs batch

Advantages of stochastic gradient descent

More “user-friendly” for very large training sets, converges faster for most hard problems, often converges to better optimum

Advantages of batch gradient descent

Is easier to justify as doing the right thing (e.g., no bad move based on outlier training example), converges faster for some problems, requires a lot fewer parameter updates per epoch

Exercise

- Prepare a short (perhaps three-sentence) summary of how gradient descent works.
- Present this summary to your neighbor
- You may want to refer to these notions:

Hypothesis	$h_\theta = \theta_0 + \theta_1 x$
Parameters	$\theta = (\theta_0, \theta_1)$
Cost function	$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$
Derivative of cost function	
Objective	$\text{minimize}_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

Gradient descent: Summary

- n-dimensional hyperplane for n parameters
- Height (dim $n+1$) is cost $\rightarrow$ the surface we move on.
- a series of steps, each step in steepest direction down
- The derivative gives us the steepest direction down.
- The learning rate determines how big the steps are we take.
- We will eventually converge (stop) at a minimum (valley).
- There is no guarantee we will reach the *global* minimum.