CSC321 Lecture 5
Learning in a Single Neuron

Roger Grosse and Nitish Srivastava

January 21, 2015

Overview

So far, we've talked about -

- Predicting scalar targets using linear regression. $y = \mathbf{w}^T \mathbf{x}$

Overview

So far, we've talked about -

- Predicting scalar targets using linear regression. $y = \mathbf{w}^T\mathbf{x}$
- Classifying between 2 classes using the perceptron.

$$y = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Overview

So far, we've talked about -

- Predicting scalar targets using linear regression. $y = \mathbf{w}^T\mathbf{x}$
- Classifying between 2 classes using the perceptron.
  $y = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x} \geq 0 \\ 0 & \text{otherwise} \end{cases}$
- Converting linear models into nonlinear models using basis functions (features). $y = \mathbf{w}^T\mathbf{x}$, becomes $y = \mathbf{w}^T\Phi(\mathbf{x})$

Overview

So far, we've talked about -

- Predicting scalar targets using linear regression. $y = \mathbf{w}^T\mathbf{x}$
- Classifying between 2 classes using the perceptron.
  $$y = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$
- Converting linear models into nonlinear models using basis functions (features). $y = \mathbf{w}^T\mathbf{x}$, becomes $y = \mathbf{w}^T\Phi(\mathbf{x})$

But this raises some questions:

- What if the thing we're trying to predict isn't real-valued or binary-valued?
- What if we don't know the right features Φ?

This week, we cover a much more general learning framework which gets around both of these issues.

Overview

Linear regression and the perceptron algorithm were both one-off tricks.

- For linear regression, we derived a closed-form solution by setting the partial derivatives to 0. This only works for a handful of learning algorithms.

Overview

Linear regression and the perceptron algorithm were both one-off tricks.

- For linear regression, we derived a closed-form solution by setting the partial derivatives to 0. This only works for a handful of learning algorithms.
- For the perceptron, we gave a simple add/subtract algorithm which works under an unrealistic "linear separability" assumption.

Overview

Linear regression and the perceptron algorithm were both one-off tricks.

- For linear regression, we derived a closed-form solution by setting the partial derivatives to 0. This only works for a handful of learning algorithms.
- For the perceptron, we gave a simple add/subtract algorithm which works under an unrealistic "linear separability" assumption.

For most of this course, we will instead write down an objective function and optimize it using a technique called gradient descent.

Overview

Loss function : A measure of unhappiness.

How unhappy should we be if the model predicts y when we want it to predict t ?

Overview

Loss function : A measure of unhappiness.

How unhappy should we be if the model predicts y when we want it to predict t ?

Some examples of loss functions we'll learn about later

Setting	Example	Loss function $C(\mathbf{w})$
least-squares regression	predict stock prices	$(y-t)^2$
robust regression	predict stock prices	$\|y-t\|$
classification	predict object category from image	$-\log p(t\|\mathbf{x})$
generative modeling	model distribution of English sentences	$-\log p(\mathbf{x})$

Overview

Loss function : A measure of unhappiness.

How unhappy should we be if the model predicts y when we want it to predict t ?

Some examples of loss functions we'll learn about later

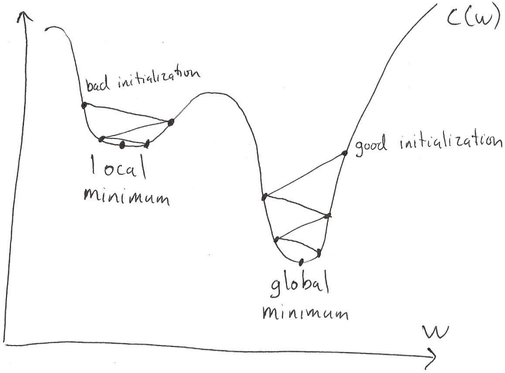
Setting	**Example**	**Loss function** $C(\mathbf{w})$
least-squares regression	predict stock prices	$(y-t)^2$
robust regression	predict stock prices	$\|y-t\|$
classification	predict object category from image	$-\log p(t\|\mathbf{x})$
generative modeling	model distribution of English sentences	$-\log p(\mathbf{x})$

Synonyms : Loss Function/Error Function/Cost Function/Objective Function.

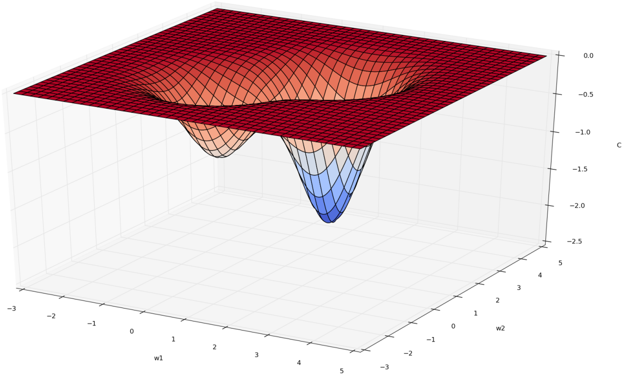
Learning $\equiv$ minimizing unhappiness.

Optimization

Visualizing gradient descent in one dimension: $w \leftarrow w - \epsilon \frac{\mathrm{d}C}{\mathrm{d}w}$



Optimization



Optimization

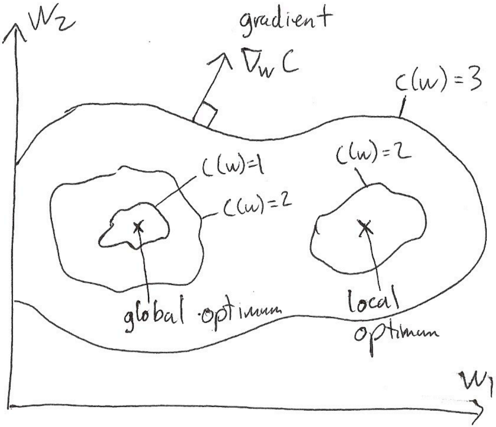
Visualizing it in two dimensions is a bit tricker.

- Level sets (or contours): sets of points on which $C(\mathbf{w})$ is constant
- Gradient: the vector of partial derivatives

$$\nabla_{\mathbf{w}} C = \left(\frac{\partial C}{\partial w_1}, \frac{\partial C}{\partial w_2} \right)$$

  - points in the direction of maximum increase
  - orthogonal to the level set
- The gradient descent updates are opposite the gradient direction.

Optimization



Question 1: Geometry of optimization

Suppose we have a linear regression problem with two training cases and no bias term:

- $\mathbf{x}^{(1)} = (1, 0)$, $t^{(1)} = 0.5$
- $\mathbf{x}^{(2)} = (0, 3)$, $t^{(2)} = 0$

Recall that the objective function is

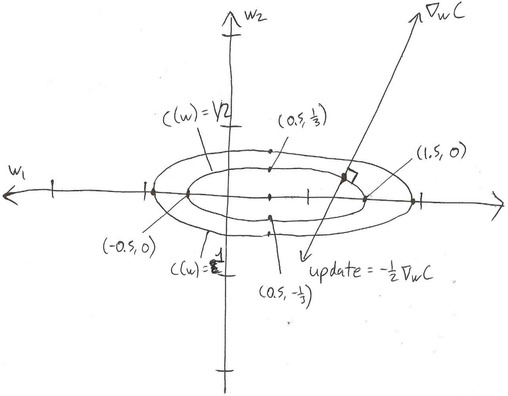
$$C(\mathbf{w}) = \frac{1}{2}(\mathbf{w}^T\mathbf{x}^{(1)} - t^{(1)})^2 + \frac{1}{2}(\mathbf{w}^T\mathbf{x}^{(2)} - t^{(2)})^2.$$

1. In weight space, sketch the level set of this objective function corresponding to $C(\mathbf{w}) = 1/2$. **Draw both axes with the same scale.**
   - Hint: Write the equation $C(\mathbf{w}) = 1/2$ explicitly in the form

     $$\frac{(w_1 - c)^2}{a^2} + \frac{(w_2 - d)^2}{b^2} = 1.$$

     What geometric object is this, and what do (a, b, c, d) represent?
2. The point $\mathbf{w} = (1.25, 0.22)$ is on the level set for $C(\mathbf{w}) = 1/2$. Sketch the gradient $\nabla_{\mathbf{w}} C$ at this point.
3. Now sketch the gradient descent update if we use a learning rate of $1/2$.

Question 1: Geometry of optimization

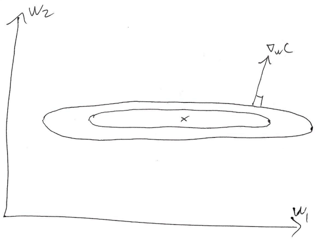


Question 1: Geometry of optimization

Note: we chose the numbers for this problem so that the ellipse would be axis-aligned. In general, the level sets for linear regression will be ellipses, but they won't be axis-aligned.

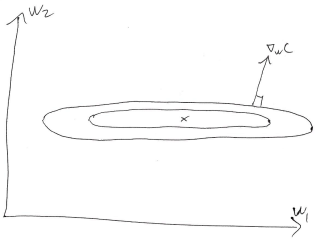
- But all ellipses are rotations of axis-aligned ellipses, so the axis-aligned case gives us all the intuition we need.
- You may have learned how to draw non-axis-aligned ellipses in a linear algebra class. If not, don't worry about it.

Question 1: Geometry of optimization



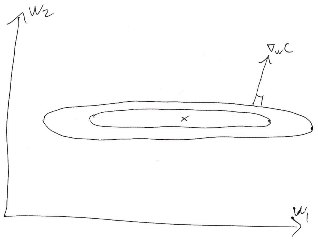
1. Suppose you perform gradient descent with a very small learning rate (e.g. 0.01). Will the objective function increase or decrease?

Question 1: Geometry of optimization



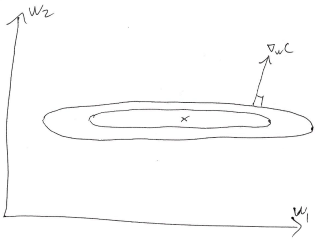
1. Suppose you perform gradient descent with a very small learning rate (e.g. 0.01). Will the objective function increase or decrease?
2. Now suppose you use a very large learning rate (e.g. 100). Will the objective function increase or decrease?

Question 1: Geometry of optimization



1. Suppose you perform gradient descent with a very small learning rate (e.g. 0.01). Will the objective function increase or decrease?
2. Now suppose you use a very large learning rate (e.g. 100). Will the objective function increase or decrease?
3. Which weight will change faster: w_1 or w_2? Which one do you want to change faster?

Question 1: Geometry of optimization



1. Suppose you perform gradient descent with a very small learning rate (e.g. 0.01). Will the objective function increase or decrease?
2. Now suppose you use a very large learning rate (e.g. 100). Will the objective function increase or decrease?
3. Which weight will change faster: w_1 or w_2? Which one do you want to change faster?

Note: a more representative picture would show it MUCH more elongated!

Question 2: Computing the gradient

Now let's consider a non-linear neuron whose activation is computed as follows:

$$z = \mathbf{w}^T \mathbf{x} = \sum_j w_j x_j \qquad y = \log(1 + z^2)$$

We will use the cost function

$$C(\mathbf{w}) = |y - t|.$$

Show how to compute the partial derivative $\partial C / \partial w_1$ using the Chain Rule as follows:

1. Compute the derivative $\mathrm{d}C/\mathrm{d}y$. (You may assume $y \neq t$.)
2. Express the derivative $\mathrm{d}C/\mathrm{d}z$ in terms of $\mathrm{d}C/\mathrm{d}y$.
3. Express the partial derivative $\partial C / \partial w_1$ in terms of $\mathrm{d}C/\mathrm{d}z$

Question 2: Computing the gradient

Solution:

$$\frac{\mathrm{d}C}{\mathrm{d}y} = \begin{cases} 1 & \text{if } y > t \\ -1 & \text{if } y < t \end{cases}$$

$$\frac{\mathrm{d}C}{\mathrm{d}z} = \frac{\mathrm{d}y}{\mathrm{d}z}\frac{\mathrm{d}C}{\mathrm{d}y}$$

$$= \frac{2z}{1+z^2}\frac{\mathrm{d}C}{\mathrm{d}y}$$

$$\frac{\partial C}{\partial w_1} = \frac{\partial z}{\partial w_1}\frac{\mathrm{d}C}{\mathrm{d}z}$$

$$= x_1 \frac{\mathrm{d}C}{\mathrm{d}z}$$

Question 2: Computing the gradient

Solution:

$$\frac{\mathrm{d}C}{\mathrm{d}y} = \begin{cases} 1 & \text{if } y > t \\ -1 & \text{if } y < t \end{cases}$$

$$\frac{\mathrm{d}C}{\mathrm{d}z} = \frac{\mathrm{d}y}{\mathrm{d}z}\frac{\mathrm{d}C}{\mathrm{d}y}$$

$$= \frac{2z}{1+z^2}\frac{\mathrm{d}C}{\mathrm{d}y}$$

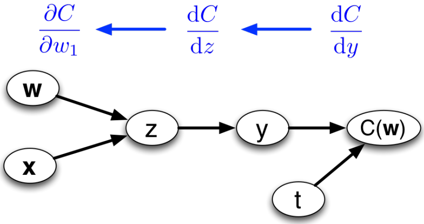
$$\frac{\partial C}{\partial w_1} = \frac{\partial z}{\partial w_1}\frac{\mathrm{d}C}{\mathrm{d}z}$$

$$= x_1 \frac{\mathrm{d}C}{\mathrm{d}z}$$

Note: If this were a calculus class, you'd do the substitutions to get $\partial C / \partial w_1$ explicitly. We won't do that here, since at this point you've already derived everything you need to implement the computation in Python.

Question 2: Computing the gradient

Observe that we compute derivatives going backwards through the computation graph:



This is true in general, not just for neural nets.

This is how we get the term "backpropagation."