CSE 559A: Computer Vision

Fall 2018: T-R: 11:30-1pm @ Lopata 101

Instructor: Ayan Chakrabarti (ayan@wustl.edu).
Course Staff: Zhihao Xia, Charlie Wu, Han Liu

http://www.cse.wustl.edu/~ayan/courses/cse559a/

Oct 30, 2018

GENERAL

- Proposal Feedback Out
  - Do a pull on your existing proposal repo
  - Read feedback.txt
  - In some cases, there are additional steps you need to take. So do this now !
- Problem Set 4 ready to Clone
  - Due two weeks from today

MACHINE LEARNING

- Obtain a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ from data
  - Maps inputs from domain $\mathcal{X}$ to outputs from domain $\mathcal{Y}$
- Components
  - Training set of pairs (x_i, y_i)
  - Loss function $L(y, \hat{y})$
  - Hypothesis Space $\mathcal{H}$ to search over for f

$$f = \arg\min_{f \in \mathcal{H}} \sum_i L(y_i, f(x_i))$$

- Basically, algorithm design by trial and error (on training set)
- A better way of solving problems when the problems are **ill-posed**
- Need to watch out for over-fitting the training set

Classification

Consider the case when y is binary, i.e., $\mathcal{Y} = \{0, 1\}$.

How do you define the loss function then ?

- Ideally, $L(y, \hat{y})$ is 0 if they are equal, 1 otherwise.

But don't know how to solve that. What if we solved by regression ?

$$w = \arg\min_{w} \frac{1}{T} \sum_{t} (y_t - w^T \tilde{x}_t)^2$$

And at test time, we can output $y = 1$ if $w^T \tilde{x} > 0.5$ and 0 otherwise.

The problem is the loss function will penalize $w^T \tilde{x}_t > 1$ when $y_t = 1$. While at test time, this would give us exactly the right answer !

MACHINE LEARNING

Logistic regression

- Learn a function $f(x) = P(y = 1)$ which regresses to the probability y is 1.
- We have to choose f such that the domain of $f(x)$ lies between $[0, 1]$.

$$f(x; w) = \sigma\left(w^T \tilde{x}\right), \quad \sigma(p) = \frac{\exp(p)}{1 + \exp(p)}$$

- This ensures that the output of f is between $[0, 1]$
- $w^T \tilde{x}$ can be interpreted as the log of the odds, or log of ratio between $P(y = 1)$ to $P(y = 0)$
- $\tilde{x}$ is some augmented "feature vector" derived from x.
  - "Linear Classifier" if $\tilde{x} = [x^T; 1]^T$ (log-odds are linear)
  - Could be polynomial $\tilde{x} = [1, x, x^2, x^3]$
  - Or other arbitrary non-linear functions of x
  - Can apply even when x is non-numeric, as long as $\hat{x}$ is numeric.

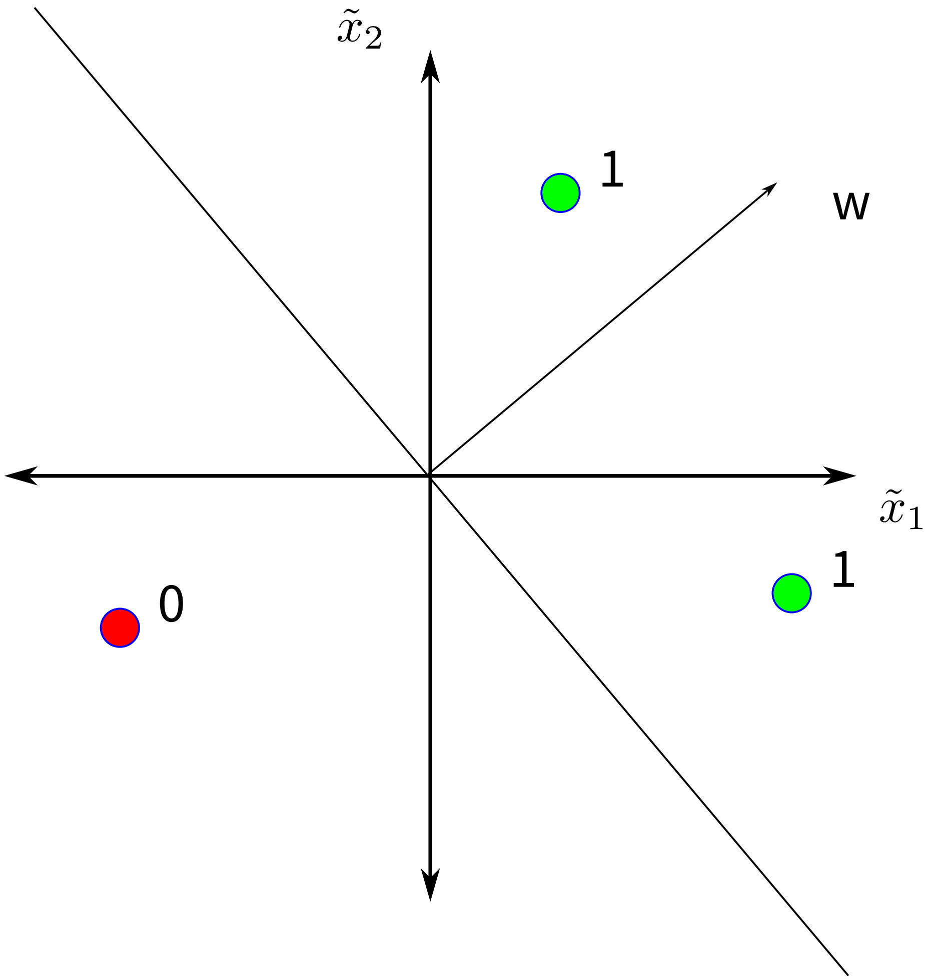
Logistic Regression

For Binary Classification: $\mathcal{X} \to [0, 1]$

$$f(x; w) = \sigma(w^T \tilde{x}) = \frac{\exp(w^T \tilde{x})}{1 + \exp(w^T \tilde{x})}$$

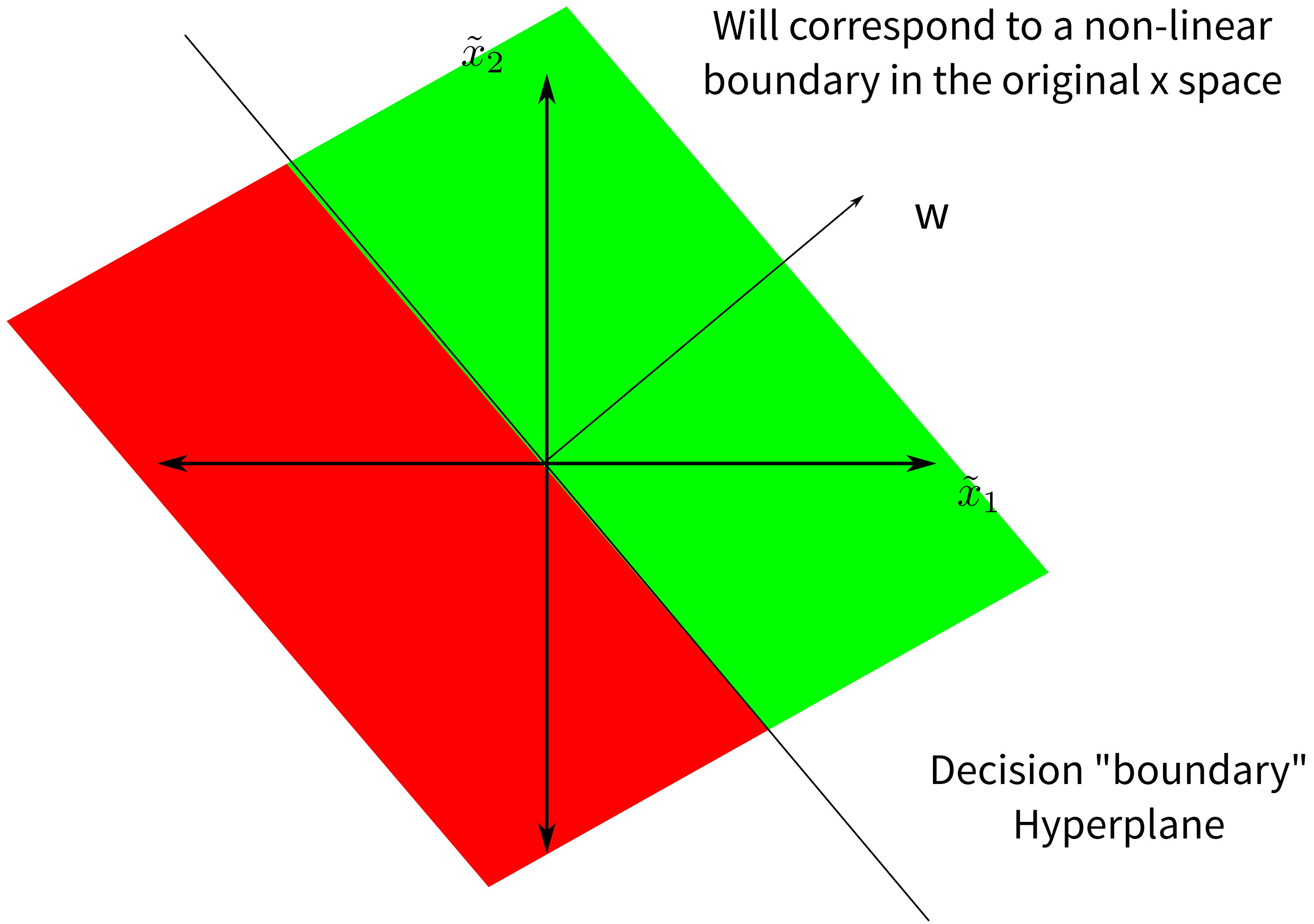
- To classify, $y = 1$ if $P(y = 1) > 0.5$ or 0 otherwise
- That is, $y = 1$ if $w^T \tilde{x} > 0$ and 0 otherwise.

- Note: Classifier is linear in chosen encoding $\tilde{x}$.
- $w^T \tilde{x} <> 0$ defines a "separating hyperplane" between positive and negative part of the space of $\tilde{x}$.



$\tilde{x}_2$
1
w
$\tilde{x}_1$
0
1

MACHINE LEARNING



Logistic regression

$$P(y=1) = f(x) = \sigma\left(w^T \tilde{x}\right)$$

What about the loss ?

Cross-Entropy Loss

If true y is 1, we want $f(x)$ to be high, and if it is 0, we want it to be low.

$$L(y, f(x)) = -\begin{cases} \log P(y=1) = \log f(x) & \text{if } \ y = 1 \\ \log P(y=1) = \log 1 - f(x) & \text{if } \ y = 0 \end{cases}$$

$$L(y, f(x)) = -y \log f(x) - (1-y) \log(1 - f(x))$$

There's a minus because this is the loss.

Minimizing $\sum_t L(y_t, f(x_t))$ can be viewed as maximizing the sum of the log-probabilities, or the product of the probabilities of the labels y_t under our predicted distribution.

Promotes a high probability for the correct label > uniform distribution (low confidence) over both labels > high probability for incorrect label.

But now, how do we minimize this function in terms of w ? No longer least-squares.

GRADIENT DESCENT

Logistic Regression

$$f(x; w) = \sigma(w^T \tilde{x}) = \frac{\exp(w^T \tilde{x})}{1 + \exp(w^T \tilde{x})}$$

- Cross-entropy / Negative Likelihood Loss

$$L(y, f(x; w)) = -y \log f(x; w) - (1 - y) \log(1 - f(x; w))$$

$$f(x; w) = \frac{1}{1 + \exp(-w^T \tilde{x})} \qquad 1 - f(x; w) = \frac{1}{1 + \exp(w^T \tilde{x})}$$

GRADIENT DESCENT

Logistic Regression

$$f(x; w) = \sigma(w^T \tilde{x}) = \frac{\exp(w^T \tilde{x})}{1 + \exp(w^T \tilde{x})}$$

- Cross-entropy / Negative Likelihood Loss

$$L(y, f(x; w)) = y \log\left[1 + \exp(-w^T \tilde{x})\right] + (1 - y) \log\left[1 + \exp(w^T \tilde{x})\right]$$

- Putting it all together, given a training set of $\{(x_t, y_t)\}$:

$$w = \arg\min_w \frac{1}{T} \sum_{t=1}^{T} y_t \log\left[1 + \exp(-w^T \tilde{x}_t)\right] + (1 - y_t) \log\left[1 + \exp(w^T \tilde{x}_t)\right]$$

GRADIENT DESCENT

Logistic Regression

$$w = \arg\min_{w} \frac{1}{T} \sum_{t=1}^{T} y_t \log\left[1 + \exp(-w^T \tilde{x}_t)\right] + (1 - y_t) \log\left[1 + \exp(w^T \tilde{x}_t)\right]$$

- You can show that this loss is a convex function of w
  (compute the Hessian matrix and show that it's eigenvalues are non-negative)
- So it has a single global minimum.

But how do we find it ?

GRADIENT DESCENT

Logistic Regression

$$w = \arg\min_w \frac{1}{T} \sum_{t=1}^{T} y_t \log\left[1 + \exp(-w^T \tilde{x}_t)\right] + (1 - y_t) \log\left[1 + \exp(w^T \tilde{x}_t)\right]$$

More General Form

$$w = \arg\min_w C(w) \quad C(w) = \frac{1}{T} \sum_t C_t(w)$$

Iterative algorithm

- Given a current estimate of w, approximate $C(w)$ as a linear function of w
  - $C(w) = \alpha^T w$
- Do this fit by computing the gradient of $C(w)$ wrt w
  - $\alpha = \nabla_w C(w)$ (would be true if $C(w) = \alpha^T w$)

Think of $[C(w), w]$ as the co-ordinates on a plane. Which direction to move in w-space to reduce $C(w)$?

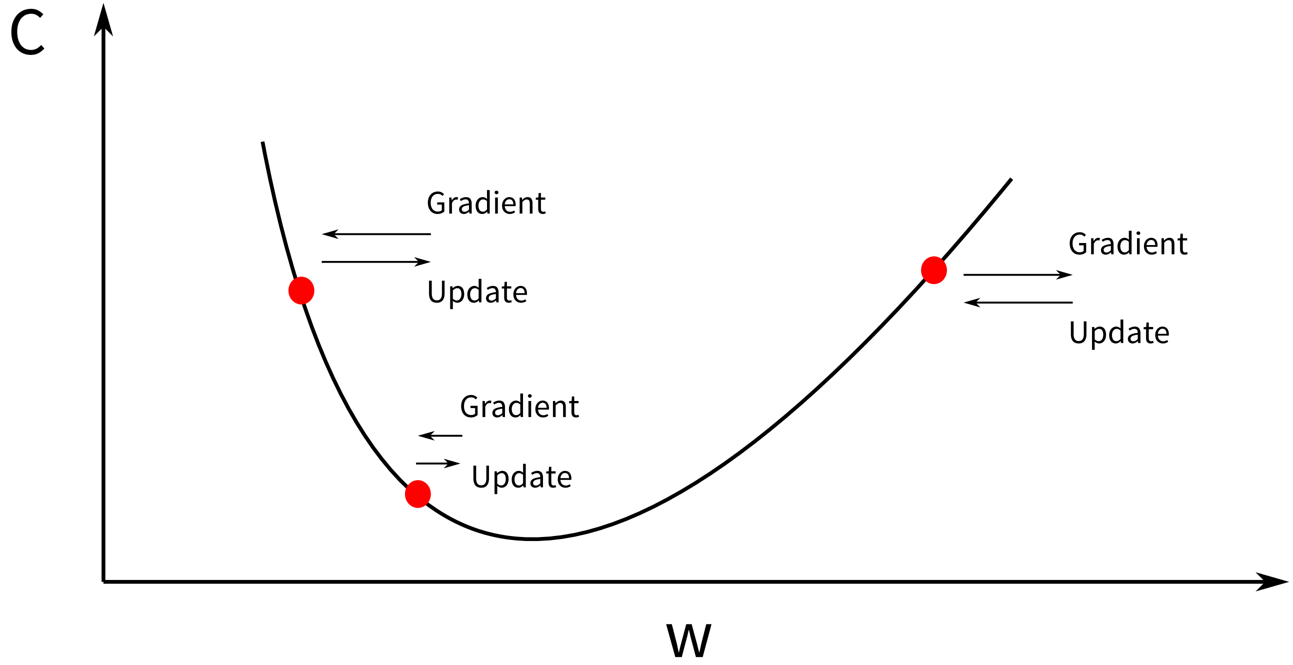
$-\alpha$

GRADIENT DESCENT

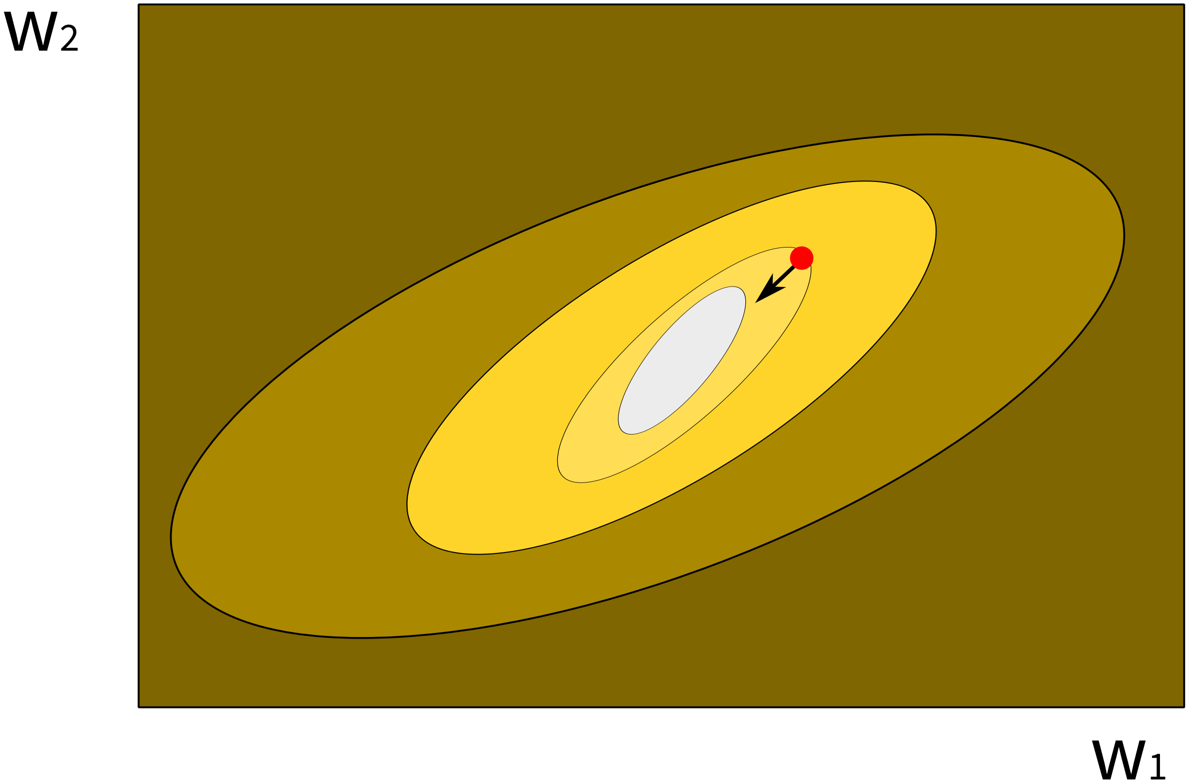
$$w = \arg\min_w C(w) \quad C(w) = \frac{1}{T}\sum_t C_t(w)$$

- Begin with initial guess w_0
- At each iteration i:
  - $w_{i+1} \leftarrow w_i - \gamma \nabla_w C(w_i)$
- At each iteration, we update the parameters w by "moving", in w-space, in the opposite direction of the gradient (at that point w_i).
- γ is the step-size. When running optimization for training, often called the "learning rate".
- In some cases, γ can be set by doing a line-search
  - Check values of $C(w - \gamma \nabla_w)$ and pick γ which minimizes the cost
- In other cases, we choose a fixed value of γ (or change it in some pre-determined schedule per iteration)
  - Then, we are moving by a distance that is proportional to magnitude of the gradient

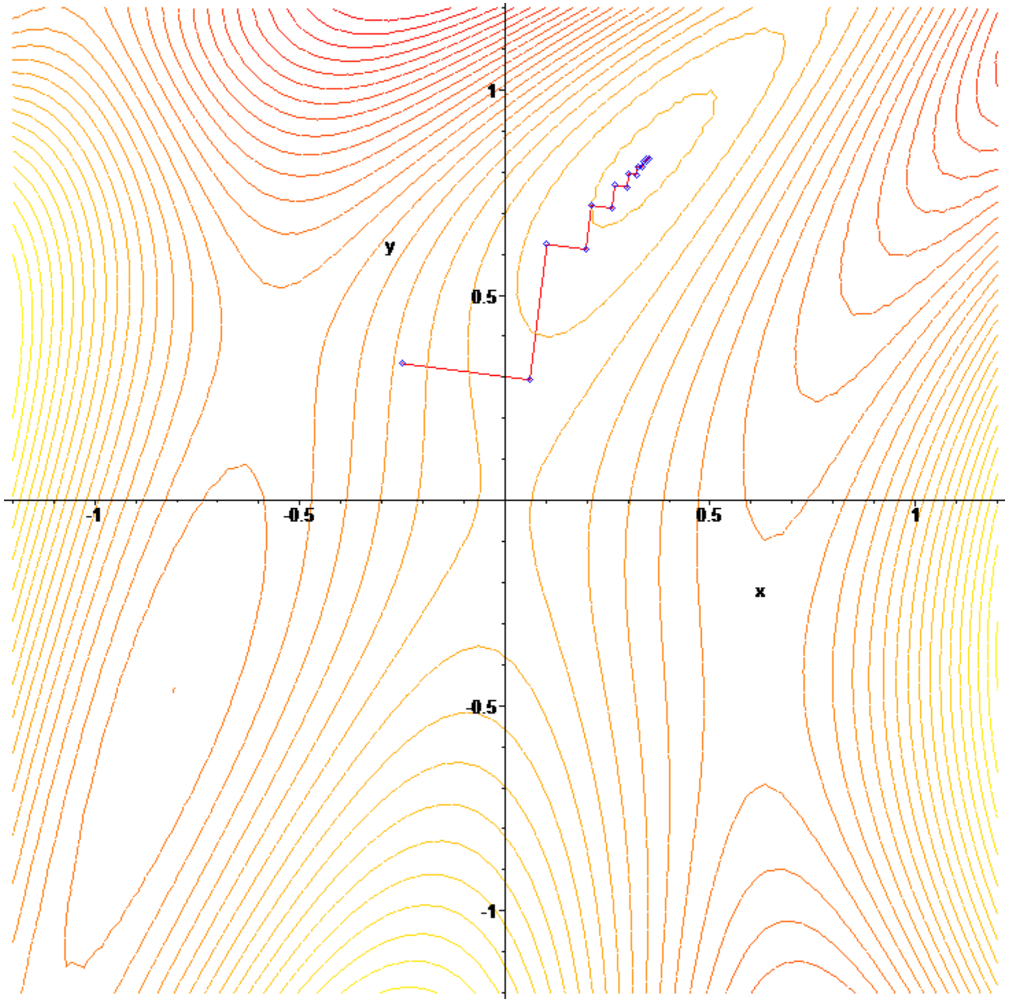
GRADIENT DESCENT



GRADIENT DESCENT



GRADIENT DESCENT



GRADIENT DESCENT

- If you select optimal step size by doing a "line search" for γ, can prove that gradient-descent will converge.
- If function is convex, converge to unique global minimum.
- Second order variants that consider the Hessian matrix: Newton & Quasi-Newton Methods
  - Gauss-Newton, Levenberg-Marquardt, ...

But simple gradient descent suffices / our only choice when:

- Function isn't convex.
- Can't afford to do line search.
- So many parameters that can't compute Hessian.

Also, no theoretical guarantees.

Theory still catching up. Meanwhile, we'll try to understand the "behavior" of the gradients.

GRADIENT DESCENT

$$\nabla_w C(w) = \begin{bmatrix} \frac{\partial}{\partial w_1} C(w) \\ \frac{\partial}{\partial w_2} C(w) \\ \vdots \end{bmatrix}$$

$$\text{If} \quad C(w) = \frac{1}{T} \sum_t C_t(w), \quad \text{then} \quad \nabla_w C(w) = \frac{1}{T} \nabla_w C_t(w)$$

Logistic Regression

$$C_t(w) = y_t \log\left[1 + \exp(-w^T \tilde{x}_t)\right] + (1 - y_t) \log\left[1 + \exp(w^T \tilde{x}_t)\right]$$

What is $\nabla_w C_t(w)$, the gradient of the loss from a singe training example ?

GRADIENT DESCENT

$$C_t(w) = y_t \log\left[1 + \exp(-w^T \tilde{x}_t)\right] + (1 - y_t) \log\left[1 + \exp(w^T \tilde{x}_t)\right]$$

Ok, what is the derivative of

$$C_t(p) = y_t \log\left[1 + \exp(-p)\right] + (1 - y_t) \log\left[1 + \exp(p)\right]$$

with respect to p (where p is a scalar).

$$\frac{\partial}{\partial p} C_t(p) = \frac{\exp(p)}{1 + \exp(p)} - y_t$$

$$\frac{\partial}{\partial p} C_t(p) = y_t \quad \frac{-\exp(-p)}{1 + \exp(-p)} + (1 - y_t) \frac{\exp(p)}{1 + \exp(p)}$$

$$= \frac{\exp(p)}{1 + \exp(p)} - y_t \left[\frac{\exp(-p)}{1 + \exp(-p)} + \frac{\exp(p)}{1 + \exp(p)}\right]$$

$$= \frac{\exp(p)}{1 + \exp(p)} - y_t \left[\frac{1}{1 + \exp(p)} + \frac{\exp(p)}{1 + \exp(p)}\right]$$

GRADIENT DESCENT

$$C_t(w) = y_t \log\left[1 + \exp(-w^T \tilde{x}_t)\right] + (1 - y_t) \log\left[1 + \exp(w^T \tilde{x}_t)\right]$$

$$C_t(p) = y_t \log\left[1 + \exp(-p)\right] + (1 - y_t) \log\left[1 + \exp(p)\right]$$
$$\frac{\partial}{\partial p} C_t(p) = \frac{\exp(p)}{1 + \exp(p)} - y_t$$

Observations

- $\frac{\exp(p)}{1+\exp(p)}$ is basically the output $f(x_t; w)$, predicted probability that $y_t = 1$.
- Remember: this is the expression for gradient of p, i.e. logit / log-odds.
- Gradient 0 if $y_t = 0$ and probability 0, $y = 1$ and probability 1.
  - Do nothing if predicting right answer with perfect confidence.
- If we say probability > 0, and $y_t = 0$. Gradient is positive.
- If we say probability < 1, and $y_t = 1$. Gradient is negative.

Remember we move in the opposite direction of gradient.

GRADIENT DESCENT

$$C_t(w) = y_t \log\left[1 + \exp(-w^T \tilde{x}_t)\right] + (1 - y_t) \log\left[1 + \exp(w^T \tilde{x}_t)\right]$$

$$C_t(p) = y_t \log\left[1 + \exp(-p)\right] + (1 - y_t) \log\left[1 + \exp(p)\right]$$

$$\frac{\partial}{\partial p} C_t(p) = \frac{\exp(p)}{1 + \exp(p)} - y_t$$

Also, changing p makes a much bigger difference in the corresponding probability, when p is near 0 / probability near 0.5.

GRADIENT DESCENT

$$C_t(w) = y_t \log\left[1 + \exp(-w^T \tilde{x}_t)\right] + (1 - y_t) \log\left[1 + \exp(w^T \tilde{x}_t)\right]$$

$$C_t(p) = y_t \log\left[1 + \exp(-p)\right] + (1 - y_t) \log\left[1 + \exp(p)\right]$$

$$\frac{\partial}{\partial p} C_t(p) = \frac{\exp(p)}{1 + \exp(p)} - y_t$$

But this is still derivative with respect to p. We want gradient with respect to w.

$$\frac{\partial}{\partial w^j} C_t(w) = \tilde{x}_t^j \quad \times \quad \left[\frac{\exp(w^T \tilde{x}_t)}{1 + \exp(w^T \tilde{x}_t)} - y_t\right]$$

$$\nabla_w C_t(w) = \tilde{x}_t \quad \left[\frac{\exp(w^T \tilde{x}_t)}{1 + \exp(w^T \tilde{x}_t)} - y_t\right]$$

$$\nabla_w C_t(w) = \nabla_w (w^T \tilde{x}_t) \quad \left[\frac{\exp(w^T \tilde{x}_t)}{1 + \exp(w^T \tilde{x}_t)} - y_t\right]$$

$$\nabla_w C_t(w) = \nabla_w p(w) \quad \frac{\partial C_t(p)}{\partial p}$$

GRADIENT DESCENT

$$w = \arg\min_w \frac{1}{T} \sum_{i=1}^{T} y_t \log\left[1 + \exp(-w^T \tilde{x}_t)\right] + (1 - y_t) \log\left[1 + \exp(w^T \tilde{x}_t)\right]$$

Putting it together:

- At each iteration i,
  - Based on current w, compute $f(x_t, w) = \hat{y}_t$
  - Compute derivative of the "output" as $\hat{y}_t - y_t$
  - Multiply by x_t to get ∇_w
  - Change w by subtracting some γ times this gradient.

GRADIENT DESCENT

$$w = \arg\min_{w} \frac{1}{T} \sum_{i=1}^{T} y_t \log\left[1 + \exp(-w^T \tilde{x}_t)\right] + (1 - y_t) \log\left[1 + \exp(w^T \tilde{x}_t)\right]$$

Putting it together:

- At each iteration i,
  - Based on current w, compute $f(x_t, w) = \hat{y}_t$ for every training sample
  - Compute derivative of the "output" as $\hat{y}_t - y_t$ for every training sample
  - Multiply by x_t and average across all training samples to get ∇_w
  - Change w by subtracting some γ times this gradient.

$$C(w) = \frac{1}{T} \sum_t C_t(w) \Rightarrow \nabla_w C = \frac{1}{T} \sum_t \nabla_w C_t$$

Expensive when we have a LOT of training data.

STOCHASTIC GRADIENT DESCENT

$$w = \arg\min_w \frac{1}{T} \sum_t C(x_t, y_t; w)$$

$$\nabla_w = \frac{1}{T} \sum_t \nabla_w C(x_t, y_t; w)$$

Remember, summation over training samples meant to approximate an expectation over $P_{XY}(x, y)$.

$$\frac{1}{T} \sum_t C(x_t, y_t; w) \rightarrow \mathbb{E}_{P_{XY}(x,y)} C(x, y; w)$$

$$\frac{1}{T} \sum_t \nabla_w C(x_t, y_t; w) \rightarrow \mathbb{E}_{P_{XY}(x,y)} \nabla_w C(x, y; w)$$

In other words, we are approximating the "true" gradient with gradients over samples.

What if we used a smaller number of samples in each iteration, but different samples in different iterations ?

STOCHASTIC GRADIENT DESCENT

- Single sample
  $$w_{i+1} \leftarrow w_i - \gamma \nabla_w C_t(x_t, y_t; w_i)$$
  At each iteration, choose a random $t \in \{1, 2, \dots, T\}$.
- "Mini"-batched SGD (sometimes GD is called Batched GD)
  $$w_{i+1} \leftarrow w_i - \gamma \nabla_w \frac{1}{B} \sum_{t \in \mathcal{B}} C_t(x_t, y_t; w_i)$$
  At each iteration, choose a random smaller batch $\mathcal{B}$ of size $B << T$.

With replacement ? Without replacement ?

STOCHASTIC GRADIENT DESCENT

In practice:

- Shuffle order of training examples
- Choose a batch size
- Take consecutive groups of B samples as you loop through iterations
  - [1,B] in iteration 1
  - [B+1,2B] in iteration 2
  - ...
- Once you reach the end of the training set (called one "epoch"), shuffle the order again.

STOCHASTIC GRADIENT DESCENT

$$w_{i+1} \leftarrow w_i - \gamma \frac{1}{B} \sum_{t \in \mathcal{B}} \nabla_w C_t(x_t, y_t; w_i)$$

General Notes

- The gradient over a mini-batch is an "approximation", or a "noisy" version of the gradient over the true training set.

$$\frac{1}{B} \sum_{t \in \mathcal{B}} \nabla_w C_t(x_t, y_t; w_i) = \frac{1}{T} \sum_{t=1}^{T} \nabla_w C_t(x_t, y_t; w_i) + \epsilon$$

- Typically, if you decrease the batch-size, you will want to decrease your step size (because you are "less sure" about the gradient).

STOCHASTIC GRADIENT DESCENT

$$w_{i+1} \leftarrow w_i - \gamma \frac{1}{B} \sum_{t \in \mathcal{B}} \nabla_w C_t(x_t, y_t; w_i)$$

General Notes

Say your cost function is convex, and you care only about decreasing this cost (not worried about overfitting)

- Larger batch size will always give you "better" gradients.
- But diminishing returns after a batch size.
- Computational cost is number of examples per iteration $\times$ number of iterations for convergence
  - Higher batch means more computation per iteration, but may mean fewer iterations required to converge.
- Best combination of step size and batch size is an empirical question.
- Another factor: parallelism.
  - Note that you can compute the gradient of all samples of your batch in parallel.
  - Ideally, you want to at least "saturate" all available parallel threads.

STOCHASTIC GRADIENT DESCENT

$$w_{i+1} \leftarrow w_i - \gamma \frac{1}{B} \sum_{t \in \mathcal{B}} \nabla_w C_t(x_t, y_t; w_i)$$

General Notes

If your cost function is NOT convex, and/or you are worried about overfitting.

- Noise in your gradients might be a good thing !
- Might help you escape local minima.
- Might prevent you from overfitting to train set.
- Try different batch sizes, check performance on dev set, not just train set.

STOCHASTIC GRADIENT DESCENT

Momentum

Standard SGD:

$$g_{i+1} = \frac{1}{B} \sum_{t \in \mathcal{B}} \nabla_w C_t(x_t, y_t; w_i)$$
$$w_{i+1} \leftarrow w_i - \gamma g_{i+1}$$

With Momentum:

For $\beta < 1$:

$$g_{i+1} = \frac{1}{B} \sum_{t \in \mathcal{B}} \nabla_w C_t(x_t, y_t; w_i) + \beta g_i$$
$$w_{i+1} \leftarrow w_i - \gamma g_{i+1}$$

- Keep adding the gradient from a previous batch, again and again across iterations, with decaying weight.
- Remember: g_i was computed with respect to a different position in w space.
- People often use β as high as 0.9 or 0.99.
- Will need to revisit "best" value of γ when you change β.