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 \( f : \mathcal{X} \to \mathcal{Y} \) 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 \( f \)

\[
\begin{align*}
  f &= \arg\min_{h \in \mathcal{H}} \sum_{i} L(y_i, f(x_i)) \\
\end{align*}
\]

- 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?

\[
\begin{align*}
  w &= \arg\min_w \frac{1}{T} \sum_{t} (y_t - w^T \hat{x}_t)^2 \\
\end{align*}
\]

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

The problem is the loss function will penalize \( w^T \hat{x} > 1 \) when \( y_i = 1 \). While at test time, this would give us exactly the right answer!
**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 $[0, 1]$.

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

This ensures that the output of $f$ is between $[0, 1]$.

$w^T \hat{x}$ can be interpreted as the log of the odds, or log of ratio between $P(y = 1)$ to $P(y = 0)$.

$\hat{x}$ is some augmented “feature vector” derived from $x$.

- “Linear Classifier” if $\hat{x} = [x^T; 1]^T$ (log-odds are linear)
- Could be polynomial $\hat{x} = [1, x, x^2, x^3]$
- 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: $\mathcal{X} \rightarrow [0, 1]$

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

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

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

**Will correspond to a non-linear boundary in the original x space**
Logistic regression

\[ P(y = 1) = f(x) = \sigma(w^T \tilde{x}) \]

What about the loss?

Cross-Entropy Loss

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

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

There’s a minus because this is the loss.

Minimizing \( \sum L(y_i, f(x_i)) \) can be viewed as maximizing the sum of the log-probabilities, or the product of the probabilities of the labels \( y_i \) 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.

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

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?
Logistic Regression

More General Form

Iterative algorithm

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

$-\alpha$

\[ w = \arg \min_w \frac{1}{T} \sum_{t=1}^T y_t \log [1 + \exp(-w^T x_t)] + (1 - y_t) \log [1 + \exp(w^T x_t)] \]

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

- Begin with initial guess $w_0$
- At each iteration $i$:
  - $w_{i+1} \leftarrow w_i - \gamma \nabla w_i C(w_i)$
- At each iteration, we update the parameters $w$ by "moving", in $w$-space, in the opposite direction of the gradient (at that point $w_i$).
- $\gamma$ is the step-size. When running optimization for training, often called the "learning rate".
- In some cases, $\gamma$ can be set by doing a line-search
  - Check values of $C(w - \gamma \nabla w)$ and pick $\gamma$ which minimizes the cost
- In other cases, we choose a fixed value of $\gamma$ (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
If you select optimal step size by doing a "line search" for $\gamma$, you can prove that gradient-descent will converge. If the function is convex, it will converge to the 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.

**Logistic Regression**

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

$$\nabla_w C(w) = \left[ \frac{\partial}{\partial w_1} C(w) \right]$$

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

**Logistic Regression**

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

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

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

$$= \frac{\exp(p)}{1 + \exp(p)} - y_t \left[\frac{\exp(-p)}{1 + \exp(-p)} + \frac{\exp(p)}{1 + \exp(p)}\right]$$
But this is still derivative with respect to . We want gradient with respect to .

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

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

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

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 \( y = 0 \). Gradient is positive.
- If we say probability < 1, and \( y = 1 \). Gradient is negative.

Remember we move in the opposite direction of gradient.

Putting it together:

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

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

\[
\frac{\partial}{\partial w} C_t(w) = \hat{x}_t \times \frac{\exp(w^T \hat{x}_t)}{1 + \exp(w^T \hat{x}_t)} - y_t
\]

\[
V_w C_t(w) = \hat{x}_t \left[ \frac{\exp(w^T \hat{x}_t)}{1 + \exp(w^T \hat{x}_t)} - y_t \right]
\]

\[
V_w C_t(w) = V_w(w^T \hat{x}_t) \left[ \frac{\exp(w^T \hat{x}_t)}{1 + \exp(w^T \hat{x}_t)} - y_t \right]
\]

\[
V_w C_t(w) = V_w(p(w) \frac{\partial C_t(p)}{\partial p})
\]

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

At each iteration \( i \),

- Based on current \( w \), compute \( f(x_i, w) = \hat{y}_i \)
- Compute derivative of the "output" as \( \hat{y}_i - y_i \)
- Multiply by \( x_i \) to get \( V_w \)
- Change \( w \) by subtracting some \( y \) times this gradient.
GRADIENT DESCENT

\[ w = \arg \min_w \frac{1}{T} \sum_{i=1}^{T} y_i \log[1 + \exp(-w^T x_i)] + (1 - y_i) \log[1 + \exp(w^T x_i)] \]

Putting it together:

- At each iteration \( i \),
  - Based on current \( w \), compute \( f(x_i, w) = \hat{y}_i \) for every training sample
  - Compute derivative of the "output" as \( \nabla w \) for every training sample
  - Multiply by \( x_i \) and average across all training samples to get \( \nabla w \)
  - Change \( w \) by subtracting some \( \gamma \) 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 \( P_{XY}(x, y) \).

\[ \frac{1}{T} \sum_t C(x_t, y_t; w) \to E_{P_{XY}(x,y)} C(x, y; w) \]

\[ \frac{1}{T} \sum_t \nabla_w C(x_t, y_t; w) \to E_{P_{XY}(x,y)} \nabla_w C(x, y; w) \]

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
  \[ w_{i+1} \leftarrow w_i - \gamma \nabla w C(x_t, y_t; w_i) \]
  At each iteration, choose a random \( t \in \{1, 2, \ldots, T\} \).

- "Mini"-batched SGD (sometimes GD is called Batched GD)
  \[ w_{i+1} \leftarrow w_i - \gamma \nabla w \frac{1}{B} \sum_t C(x_t, y_t; w_i) \]
  At each iteration, choose a random smaller batch \( B \) of size \( B < T \).

With replacement? Without replacement?
STOCHASTIC GRADIENT DESCENT

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

General Notes

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

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

- 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

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

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

Momentum

Standard SGD:

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

$$w_{i+1} \leftarrow w_i - \gamma g_{i+1}$$

With Momentum:

$$\beta < 1$$

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

$$w_{i+1} \leftarrow w_i - \gamma g_{i+1}$$

- 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 $\beta$ as high as 0.9 or 0.99.  
- Will need to revisit "best" value of $\gamma$ when you change $\beta$.  

### Note
- $\gamma$ is the step size (learning rate).  
- $\beta$ is the momentum coefficient.