Lecture 8: Stochastic Gradient Descent; Large Margin Learning

TTIC 31020: Introduction to Machine Learning

Instructor: Kevin Gimpel

TTI-Chicago

October 24, 2019

Administrivia

- Problem set 2 due Monday 11:59pm (see yesterday's update on Canvas regarding the `infer` function)
- Office Hours:
  - Mondays 3-4pm (me, room 531)
  - Tuesdays 1-2pm (TA)
  - Wednesdays 3:30-4:30pm (TA)
  - Thursdays 3:30-4:30pm (TA)
  - TA office hours held in 4th floor commons
  - **Special office hours this week only: Friday 3-4:30pm (TA)**
- Recitations:
  - Tues 3:30-4:20pm or Thurs 1:00-1:50pm (expanded times so that there is more time for asking questions in a group setting)
  - Next week: logistic regression and regularization
- Regression through the origin (see Discussions on Canvas)

Review: optimal classification

- Assuming $0/1$ loss

$$\ell(h(\mathbf{x}), y) \; = \; \begin{cases} 0 & \text{if } h(\mathbf{x}) = y \\ 1 & \text{if } h(\mathbf{x}) \neq y \end{cases}$$

  the conditional risk of the classifier h is minimized by

$$h(\mathbf{x}) = \underset{c}{\operatorname{argmax}}\, p\,(y = c \,|\, \mathbf{x})$$

  which is equivalent to the log-odds ratio test:

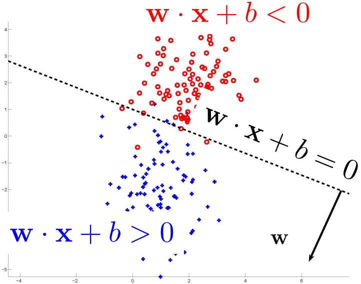
$$h(\mathbf{x}) \; = \; \hat{c} \quad \Leftrightarrow \quad \log \frac{p\,(y = \hat{c} \,|\, \mathbf{x})}{p\,(y = c \,|\, \mathbf{x})} \geq 0 \quad \forall c$$

Review: logistic regression

- Linear model for log-odds:

$$p\,(y = 1 \,|\, \mathbf{x}; \mathbf{w}, b) \;=\; \frac{1}{1 + \exp(-\mathbf{w} \cdot \mathbf{x} - b)}$$

- After training, we don't need to compute probabilities to pick a class
- Decision rule: $\widehat{y} \;=\; \mathrm{sign}\,(\mathbf{w} \cdot \mathbf{x} + b)$



- Probabilities only affect learning

Review: logistic regression gradient

- We can derive, assuming $y \in \{0, 1\}$

$$p\left(y = 1 \,|\, \mathbf{x}; \mathbf{w}, b\right) \;=\; \sigma(\mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}) + b)$$

$$\log p(\mathbf{y} \mid \mathbf{X}; \mathbf{w}, b) \;= \sum_{i=1}^{n} y_i \log \sigma(b + \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_i)) + (1 - y_i) \log\left(1 - \sigma(b + \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_i))\right)$$

$$\nabla_{\mathbf{w}}^{(t)} \log p\left(y_i \,|\, \mathbf{x}_i; \mathbf{w}^{(t)}\right) \;=\; \left[y_i - \sigma(\mathbf{w}^{(t)} \cdot \boldsymbol{\phi}(\mathbf{x}_i))\right] \boldsymbol{\phi}(\mathbf{x}_i)$$

- Verify this. Use the following fact (verify this too):

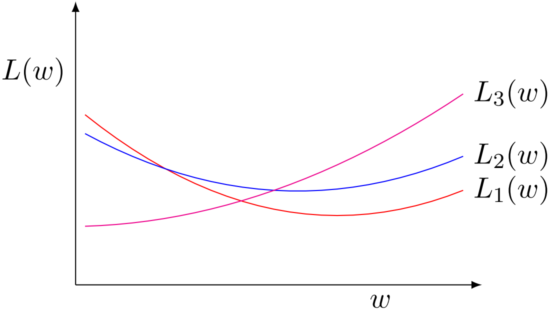
$$\frac{d\sigma(z)}{dz} \;=\; \sigma(z)\left(1 - \sigma(z)\right)$$

Stochastic gradient descent: intuition

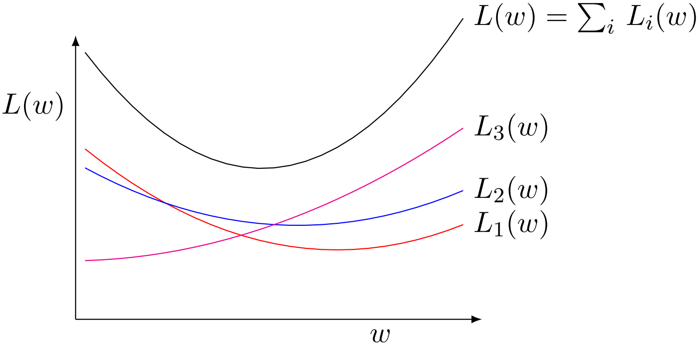
- Computing gradient on all n examples is expensive and may be wasteful
- Many data points provide similar information
- Idea: present examples one at a time, and pretend that the gradient on the entire set is the same as gradient on one example
- Formally: estimate gradient of the loss L

$$\frac{1}{n}\sum_{i=1}^{n}\frac{\partial}{\partial \mathbf{w}}L(y_i, \mathbf{x}_i; \mathbf{w}) \approx \frac{\partial}{\partial \mathbf{w}}L(y_t, \mathbf{x}_t; \mathbf{w})$$

Stochastic gradient descent: intuition

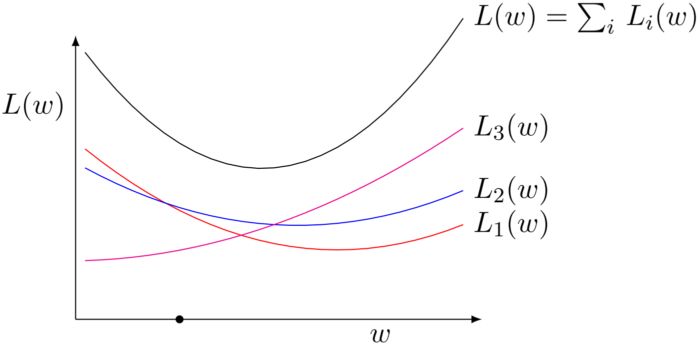


Stochastic gradient descent: intuition



- Objective: $\min_w L(w) = \min_w \sum_{i=1}^{n} L_i(w)$

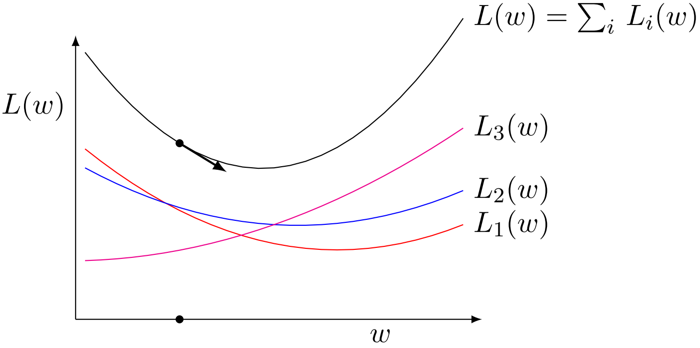
Stochastic gradient descent: intuition



- Objective: $\min_w L(w) = \min_w \sum_{i=1}^{n} L_i(w)$
- Stochastic approximation: given an i, estimate

$$\frac{1}{n}\nabla L(w) \approx \nabla L_i(w)$$

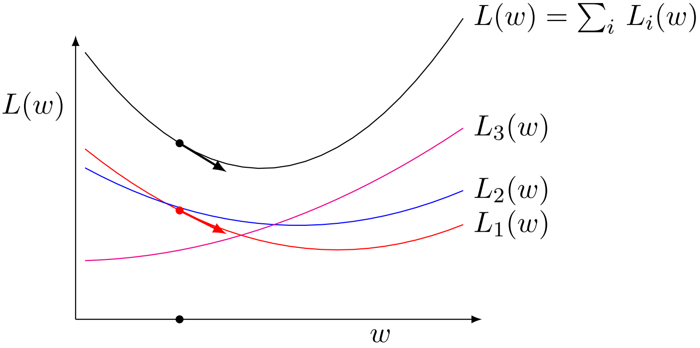
Stochastic gradient descent: intuition



- Objective: $\min_w L(w) = \min_w \sum_{i=1}^{n} L_i(w)$
- Stochastic approximation: given an i, estimate

$$\frac{1}{n}\nabla L(w) \approx \nabla L_i(w)$$

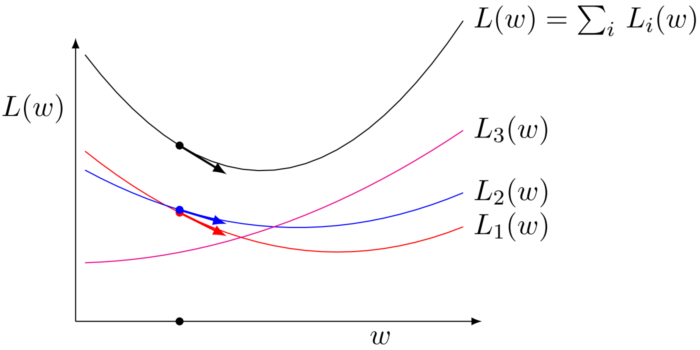
Stochastic gradient descent: intuition



- Objective: $\min_w L(w) = \min_w \sum_{i=1}^{n} L_i(w)$
- Stochastic approximation: given an i, estimate

$$\frac{1}{n}\nabla L(w) \approx \nabla L_i(w)$$

Stochastic gradient descent: intuition

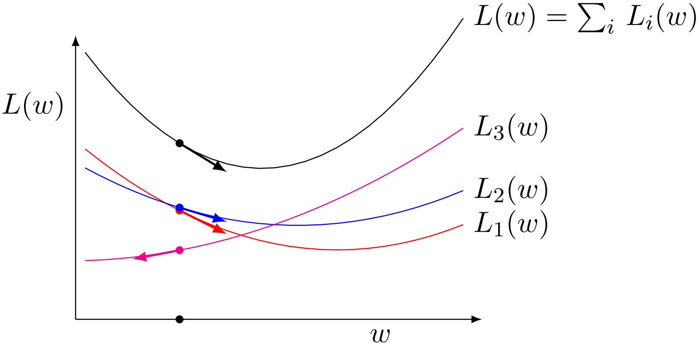


- Objective: $\min_w L(w) = \min_w \sum_{i=1}^{n} L_i(w)$
- Stochastic approximation: given an i, estimate

$$\frac{1}{n}\nabla L(w) \approx \nabla L_i(w)$$

- Could be a noisy estimate

Stochastic gradient descent: intuition



- Objective: $\min_w L(w) = \min_w \sum_{i=1}^{n} L_i(w)$
- Stochastic approximation: given an i, estimate

$$\frac{1}{n}\nabla L(w) \approx \nabla L_i(w)$$

- Could be a noisy estimate

Stochastic gradient descent

- An incremental algorithm:
  - Present examples $(\mathbf{x}_i, y_i)$ one at a time,
  - Modify $\mathbf{w}$ slightly to increase the log-probability of observed y_i:

$$\mathbf{w} \ := \ \mathbf{w} + \eta \frac{\partial}{\partial \mathbf{w}} \log p\left(y_i \,|\, \mathbf{x}_i; \mathbf{w}\right)$$

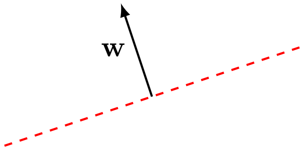
  where the **learning rate** η determines how "slightly".
- Epoch (full pass through data) contains n updates instead of one
- Good practice: shuffle the data
- Can add gradient of regularizer if wanted

Stochastic gradient descent

- Linear model (assume $b = 0$)

$$\mathbf{w}_{new} := \mathbf{w} + \eta \frac{\partial}{\partial \mathbf{w}} \log p(y \mid \mathbf{x}; \mathbf{w})$$
$$= \mathbf{w} + \eta (y - \sigma(\mathbf{w} \cdot \mathbf{x})) \mathbf{x}$$

- Contribution of one example:

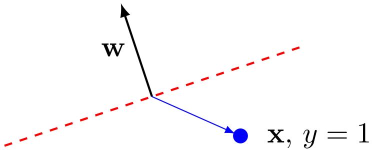


Stochastic gradient descent

- Linear model (assume $b = 0$)

$$\mathbf{w}_{new} := \mathbf{w} + \eta \frac{\partial}{\partial \mathbf{w}} \log p\,(y \,|\, \mathbf{x}; \mathbf{w})$$
$$= \mathbf{w} + \eta\,(y - \sigma(\mathbf{w} \cdot \mathbf{x}))\,\mathbf{x}$$

- Contribution of one example:

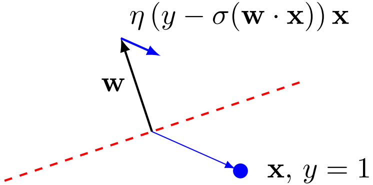


Stochastic gradient descent

- Linear model (assume $b = 0$)

$$\begin{aligned} \mathbf{w}_{new} \; &:= \; \mathbf{w} \, + \, \eta \frac{\partial}{\partial \mathbf{w}} \log p\,(y \,|\, \mathbf{x}; \mathbf{w}) \\ &= \; \mathbf{w} \, + \, \eta\,(y - \sigma(\mathbf{w} \cdot \mathbf{x}))\,\mathbf{x} \end{aligned}$$

- Contribution of one example:

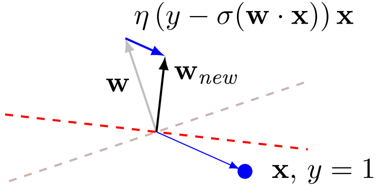


Stochastic gradient descent

- Linear model (assume $b = 0$)

$$\begin{aligned} \mathbf{w}_{new} \; &:= \; \mathbf{w} \,+\, \eta \frac{\partial}{\partial \mathbf{w}} \log p\,(y \,|\, \mathbf{x}; \mathbf{w}) \\ &= \; \mathbf{w} \,+\, \eta\,(y - \sigma(\mathbf{w} \cdot \mathbf{x}))\,\mathbf{x} \end{aligned}$$

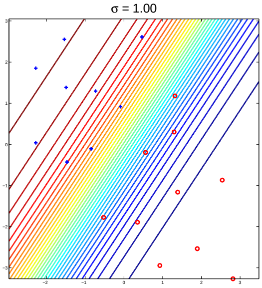
- Contribution of one example:



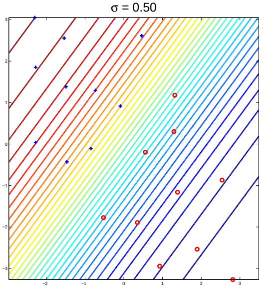
Regularized logistic regression

- As with linear regression, we can regularize the ERM learning objective, e.g.,

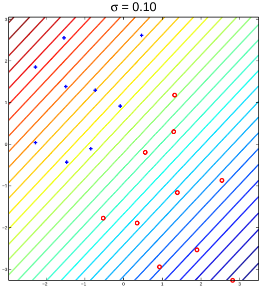
$$\underset{\mathbf{w}}{\operatorname{argmin}} \left\{ -\frac{1}{n} \sum_{i=1}^{n} \log p\left(y_i \mid \mathbf{x}_i; \mathbf{w}\right) \;+\; \lambda \|\mathbf{w}\|^2 \right\}$$



$\lambda = 1$



$\lambda = 2$

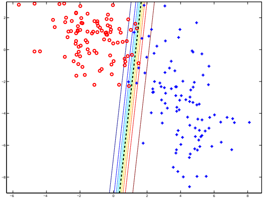


$\lambda = 10$

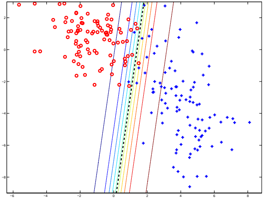
The effect of regularization: non-separable data

$$\underset{\mathbf{w}}{\operatorname{argmin}} \left\{ -\frac{1}{n} \sum_{i=1}^{n} \log p\left(y_i \,|\, \mathbf{x}_i; \mathbf{w}\right) \;+\; {\color{red}\lambda \|\mathbf{w}\|^2} \right\}$$

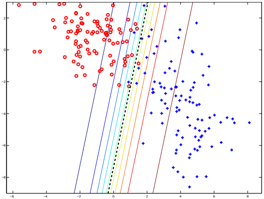
$\lambda = 0$



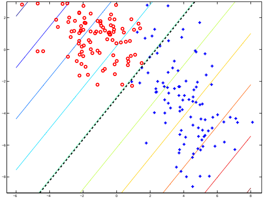
$\lambda = 1$



$\lambda = 10$



$\lambda = 100$



Softmax

- Logistic regression computes a **score** $f(\mathbf{x}; \mathbf{w}) = \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x})$, which is converted to a "posterior"

$$p\,(y = 1\,|\,\mathbf{x}) \;=\; \frac{\exp\left(f(\mathbf{x}; \mathbf{w})\right)}{1 + \exp\left(f(\mathbf{x}; \mathbf{w})\right)}$$

  (verify that this is equivalent to the form we had before)

- The **softmax** model: we now have C classes, and C scores $f_c(\mathbf{x}; \mathbf{W}) \;=\; \mathbf{w}_c \cdot \boldsymbol{\phi}(\mathbf{x})$
- To get posteriors from scores, exponentiate and normalize:

$$p\,(y = c\,|\,\mathbf{x}) \;=\; \frac{\exp\left(\mathbf{w}_c \cdot \boldsymbol{\phi}(\mathbf{x})\right)}{\sum_{k=1}^{C} \exp\left(\mathbf{w}_k \cdot \boldsymbol{\phi}(\mathbf{x})\right)}$$

  Note: decision on $\mathbf{x}$ depends on all $\mathbf{w}_c$ for $c = 1, \ldots, C$.

- For $C = 2$, this is identical to logistic regression (homework)
- Note: for prediction, just argmax over scores; no need to exponentiate/normalize

Softmax parameterization

$$p\,(y = c\,|\,\mathbf{x}) \;=\; \frac{\exp(\mathbf{w}_c \cdot \boldsymbol{\phi}(\mathbf{x}))}{\sum_{k=1}^{C} \exp(\mathbf{w}_k \cdot \boldsymbol{\phi}(\mathbf{x}))}$$

- The posteriors are invariant to shifting scores

Softmax parameterization

$$p\,(y = c \,|\, \mathbf{x}) \;=\; \frac{\exp(\mathbf{w}_c \cdot \boldsymbol{\phi}(\mathbf{x}) - a)}{\sum_{k=1}^{C} \exp(\mathbf{w}_k \cdot \boldsymbol{\phi}(\mathbf{x}) - a)}$$

- The posteriors are invariant to shifting scores
- A common problem: overflow in $\exp(\mathbf{w}_c \cdot \boldsymbol{\phi}(\mathbf{x}))$
- Solution: subtract $a = \max_c(\mathbf{w}_c \cdot \boldsymbol{\phi}(\mathbf{x}))$

Softmax parameterization

$$p\,(y = c \,|\, \mathbf{x}) \;=\; \frac{\exp(\mathbf{w}_c \cdot \boldsymbol{\phi}(\mathbf{x}) {\color{red}-a})}{\sum_{k=1}^{C} \exp(\mathbf{w}_k \cdot \boldsymbol{\phi}(\mathbf{x}) {\color{red}-a})}$$

- The posteriors are invariant to shifting scores
- A common problem: overflow in $\exp(\mathbf{w}_c \cdot \boldsymbol{\phi}(\mathbf{x}))$
- Solution: subtract $a = \max_c(\mathbf{w}_c \cdot \boldsymbol{\phi}(\mathbf{x}))$
- Then, max score is 0, and the rest are negative; underflow is OK (some may turn to zero)
- Examples: scores $= [1000, 995, 10, 10, 1]$
  Naïve exponentiation: $\approx\; [\infty, \infty, 2.2\mathrm{e}4, 2.2\mathrm{e}4, 2.7]$
  After shifting dynamic range: $\approx\; [1, 0.007, 0, 0, 0]$

Perceptron

- Consider binary classification task, $\mathcal{Y} = \{\pm 1\}$
- The perceptron is a very simple learning algorithm for linear classifiers of the form $h(\mathbf{x}) = \mathrm{sign}(\mathbf{w} \cdot \mathbf{x} + b)$
- With logistic regression, we begin by defining the probability of each label, then maximize log-likelihood (minimize log loss) using gradient-based optimization
- The perceptron learning procedure was originally designed as an algorithm, but we can reverse engineer it to recover a loss function (which can be directly optimized with subgradient descent)

Perceptron algorithm

- Binary classification task: $\mathcal{Y} = \{\pm 1\}$
- Linear classifier: $h(\mathbf{x}) = \mathrm{sign}(\mathbf{w} \cdot \mathbf{x} + b)$
- Algorithm:
  initialize $\mathbf{w}^{(0)} = \mathbf{0}$, $b^{(0)} = 0$
  take one example $(\mathbf{x}_i, y_i)$ at a time
  if $y_i \left(\mathbf{w}^{(t)} \cdot \mathbf{x}_i + b^{(t)}\right) \leq 0$ (i.e., classifier was incorrect), update:

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} + y_i \mathbf{x}_i, \quad b^{(t+1)} := b^{(t)} + y_i$$

  otherwise (i.e., classifier was correct), do nothing
  stop when all data are classified correctly

Perceptron algorithm

- Binary classification task: $\mathcal{Y} = \{\pm 1\}$
- Linear classifier: $h(\mathbf{x}) = \mathrm{sign}(\mathbf{w} \cdot \mathbf{x} + b)$
- Algorithm:
  initialize $\mathbf{w}^{(0)} = \mathbf{0}$, $b^{(0)} = 0$
  take one example $(\mathbf{x}_i, y_i)$ at a time
  if $y_i \left(\mathbf{w}^{(t)} \cdot \mathbf{x}_i + b^{(t)}\right) \leq 0$ (i.e., classifier was incorrect), update:

$$\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} + y_i \mathbf{x}_i, \quad b^{(t+1)} := b^{(t)} + y_i$$

  otherwise (i.e., classifier was correct), do nothing
  stop when all data are classified correctly
- Compare this to logistic regression; what is the loss?
- What is the model for $p\,(y \,|\, \mathbf{x})$?

Perceptron updates

- Consider an example $(\mathbf{x}, y)$ misclassified in iteration t,

$$y\left(\mathbf{w}^{(t)} \cdot \mathbf{x} + b^{(t)}\right) < 0$$

- After the update $\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} + y\mathbf{x}$, $b^{(t+1)} := b^{(t)} + y$:

$$y\left(\mathbf{w}^{(t+1)} \cdot \mathbf{x} + b^{(t+1)}\right) = y\left(\mathbf{w}^{(t)} \cdot \mathbf{x} + y\mathbf{x} \cdot \mathbf{x} + b^{(t)} + y\right)$$

Perceptron updates

- Consider an example $(\mathbf{x}, y)$ misclassified in iteration t,

$$y\left(\mathbf{w}^{(t)} \cdot \mathbf{x} + b^{(t)}\right) \;<\; 0$$

- After the update $\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} + y\mathbf{x}$, $b^{(t+1)} := b^{(t)} + y$:

$$\begin{aligned} y\left(\mathbf{w}^{(t+1)} \cdot \mathbf{x} + b^{(t+1)}\right) &= y\left(\mathbf{w}^{(t)} \cdot \mathbf{x} + y\mathbf{x} \cdot \mathbf{x} + b^{(t)} + y\right) \\ &= y\left(\mathbf{w}^{(t)} \cdot \mathbf{x} + b^{(t)}\right) + y^2\|\mathbf{x}\|^2 + y^2 \end{aligned}$$

Perceptron updates

- Consider an example $(\mathbf{x}, y)$ misclassified in iteration t,

$$y\left(\mathbf{w}^{(t)} \cdot \mathbf{x} + b^{(t)}\right) < 0$$

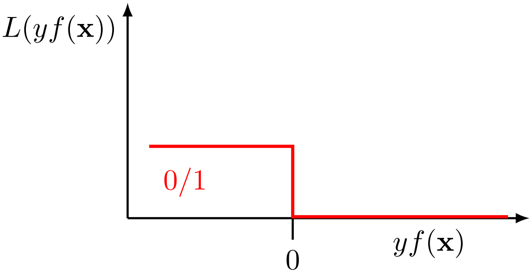
- After the update $\mathbf{w}^{(t+1)} := \mathbf{w}^{(t)} + y\mathbf{x}$, $b^{(t+1)} := b^{(t)} + y$:

$$\begin{aligned} y\left(\mathbf{w}^{(t+1)} \cdot \mathbf{x} + b^{(t+1)}\right) &= y\left(\mathbf{w}^{(t)} \cdot \mathbf{x} + y\mathbf{x} \cdot \mathbf{x} + b^{(t)} + y\right) \\ &= y\left(\mathbf{w}^{(t)} \cdot \mathbf{x} + b^{(t)}\right) + y^2\|\mathbf{x}\|^2 + y^2 \\ &\geq y\left(\mathbf{w}^{(t)} \cdot \mathbf{x} + b^{(t)}\right) \end{aligned}$$

- Similar intuition to logistic regression with SGD: each example "pulls" the model to classify it better
- In contrast to logistic regression, strength of the pull is not dependent on $\mathbf{w}$

Loss functions for binary classification

- Recall that we really want to minimize 0/1 loss
- In plot below,
  $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$
  $\mathcal{Y} = \{\pm 1\}$
  y is true class label
  L is "loss"

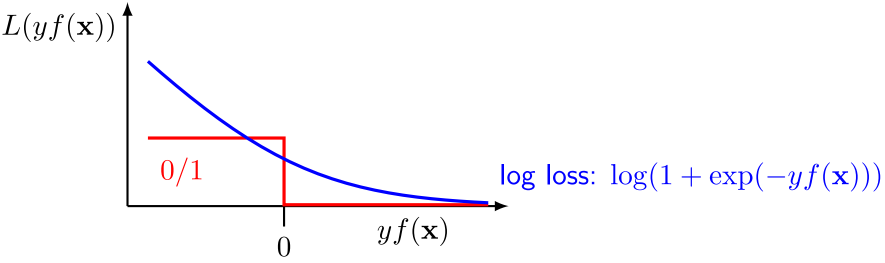


Loss functions for binary classification

- Logistic regression minimizes **log loss**:

$$\underset{\mathbf{w},b}{\operatorname{argmin}} - \sum_{i=1}^{n} \log p(y_i \mid \mathbf{x}_i; \mathbf{w}, b)$$

- This is a **surrogate** loss; we use it because it's not computationally feasible to optimize 0/1 loss directly
- In plot below, $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$, $\mathcal{Y} = \{\pm 1\}$, y is true class label



Perceptron loss

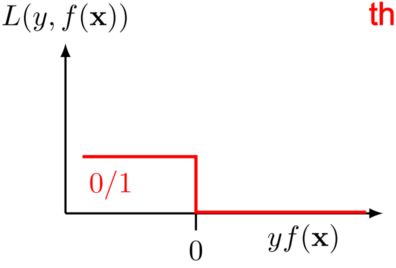
- A mistake driven algorithm: updates weights only when making a mistake on the example

$$\mathbf{w} := \mathbf{w} + y_i \mathbf{x}_i \quad \text{iff } \operatorname{sign}(\mathbf{w} \cdot \mathbf{x}_i) \neq y_i$$

- What loss does this minimize?

$$\begin{cases} 0 & \text{if } y_i(\mathbf{w} \cdot \mathbf{x}_i) > 0 \\ y_i x_i & \text{if } y_i(\mathbf{w} \cdot \mathbf{x}_i) \leq 0 \end{cases}$$

this is the loss



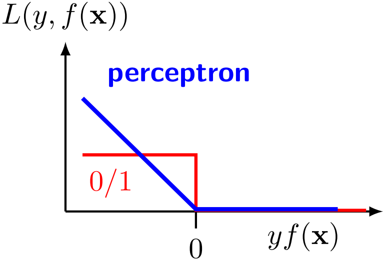
Perceptron loss

- A mistake driven algorithm: updates weights only when making a mistake on the example

$$\mathbf{w} := \mathbf{w} + y_i \mathbf{x}_i \quad \text{iff } \operatorname{sign}(\mathbf{w} \cdot \mathbf{x}_i) \neq y_i$$

- What loss does this minimize?

$$\begin{cases} 0 & \text{if } y_i(\mathbf{w} \cdot \mathbf{x}_i) > 0 \\ y_i x_i & \text{if } y_i(\mathbf{w} \cdot \mathbf{x}_i) \leq 0 \end{cases}$$



- "Perceptron" loss
- Continuous but non-smooth
- Perceptron performs descent on this loss
- **Subgradient** descent (next time)

Perceptron: analysis

- Assume data are linearly separable (otherwise will never stop!)
- The final classifier:

$$\mathbf{w} = \sum_{t=1}^{T} \alpha_t \mathbf{x}_{i(t)}$$

  where T is the total number of iterations, $i(t)$ is the index of example used in t-th iteration, and α_t is 0 or y_i, depending on what happened in t-th iteration.
- Let $\mathbf{w}$, b be a linear separator, $\|\mathbf{w}\| = 1$, and the margin be γ. (we can always ensure $\|\mathbf{w}\| = 1$)
- Theorem (Novikoff, 1962): perceptron will converge after

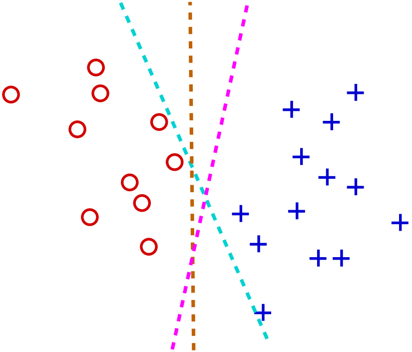
$$O\left(\frac{(\max_i \|\mathbf{x}_i\|)^2}{\gamma^2}\right)$$

Hardness of learning linear classifiers

- If data linearly separable: perceptron will learn *a* separator (not necessarily the "best" separator)
- What if the data are not separable?
  assume data are separable with error $\epsilon > 0$
- Might want to look for linear separator achieving this error
- Or approximate this, i.e., find a separator achieving $\alpha\epsilon$ $0/1$ loss
- Result (2006): even approximating for constant α is NP-hard
- Thus we (almost) always replace $0/1$ loss with differentiable surrogates

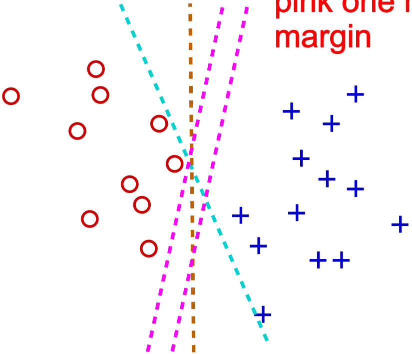
Optimal linear classifier

- Which decision boundary is better?



Optimal linear classifier

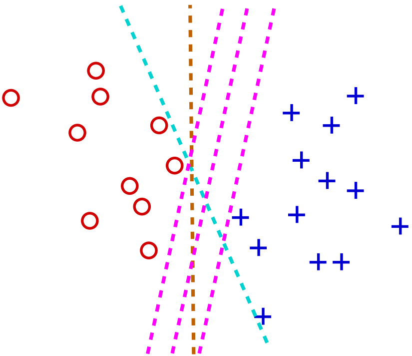
- Which decision boundary is better?



pink one has larger margin

Optimal linear classifier

- Which decision boundary is better?



- We will want to capture this intuition when learning linear classifiers