Gradient Descent and Loss Functions

Mengye Ren

NYU

September 12, 2023

Review: ERM

Our Machine Learning Setup

Prediction Function

A **prediction function** gets input x and produces an output $y = f(x)$.

Our Machine Learning Setup

Prediction Function

A **prediction function** gets input x and produces an output $y = f(x)$.

Loss Function

A **loss function** $\ell(\hat{y}, y)$ evaluates an action in the context of the outcome y.

Risk and the Bayes Prediction Function

Definition

The **risk** of a prediction function $f : \mathcal{X} \to \mathcal{Y}$ is

$$R(f) = \mathbb{E}\ell(f(x), y).$$

In words, it's the **expected loss** of f on a new example (x, y) drawn randomly from $P_{\mathcal{X}\times\mathcal{Y}}$.

Risk and the Bayes Prediction Function

Definition

The **risk** of a prediction function $f : \mathcal{X} \to \mathcal{Y}$ is

$$R(f) = \mathbb{E}\ell(f(x), y).$$

In words, it's the **expected loss** of f on a new example (x, y) drawn randomly from $P_{\mathcal{X}\times\mathcal{Y}}$.

Definition

A **Bayes prediction function** f^* is a function that achieves the *minimal risk* among all possible functions:

$$f^* \in \arg\min_f R(f),$$

- The risk of a Bayes prediction function is called the **Bayes risk**.

The Empirical Risk

Let $\mathcal{D}_n = ((x_1, y_1), \ldots, (x_n, y_n))$ be drawn i.i.d. from $\mathcal{P}_{\mathcal{X}\times\mathcal{Y}}$.

Definition

The **empirical risk** of f with respect to $\mathcal{D}_n$ is

$$\hat{R}_n(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(x_i), y_i).$$

- The **unconstrained** empirical risk minimizer can overfit.
  - i.e. if we minimize $\hat{R}_n(f)$ over **all functions**, we overfit.

Constrained Empirical Risk Minimization

Definition

A **hypothesis space** $\mathcal{F}$ is a set of functions mapping $\mathcal{X} \to \mathcal{Y}$.

- This is the collection of prediction functions we are choosing from.

Constrained Empirical Risk Minimization

Definition

A **hypothesis space** $\mathcal{F}$ is a set of functions mapping $\mathcal{X} \to \mathcal{Y}$.

- This is the collection of prediction functions we are choosing from.

- An **empirical risk minimizer** (ERM) in $\mathcal{F}$ is

$$\hat{f}_n \in \underset{f \in \mathcal{F}}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i).$$

- From now on "ERM" always means "constrained ERM".

- So we should always specify the hypothesis space when we're doing ERM.

Example: Linear Least Squares Regression

Setup

- Loss: $\ell(\hat{y}, y) = (y - \hat{y})^2$

Example: Linear Least Squares Regression

Setup

- Loss: $\ell(\hat{y}, y) = (y - \hat{y})^2$
- **Hypothesis space:** $\mathcal{F} = \left\{ f : \mathrm{R}^d \to \mathrm{R} \mid f(x) = w^T x,\ w \in \mathrm{R}^d \right\}$

- Given a data set $\mathcal{D}_n = \{(x_1, y_1), \ldots, (x_n, y_n)\}$,
  - Our goal is to find the ERM $\hat{f} \in \mathcal{F}$.

Example: Linear Least Squares Regression

Objective Function: Empirical Risk

We want to find the function in $\mathcal{F}$, parametrized by $w \in \mathrm{R}^d$, that minimizes the empirical risk:

$$\hat{R}_n(w) = \frac{1}{n} \sum_{i=1}^{n} \left(w^T x_i - y_i\right)^2$$

- How do we solve this optimization problem?

$$\min_{w \in \mathrm{R}^d} \hat{R}_n(w)$$

- (For OLS there's a closed form solution, but in general there isn't.)

Gradient Descent

Unconstrained Optimization

Setting

We assume that the objective function $f : \mathbb{R}^d \to \mathbb{R}$ is *differentiable.*
We want to find

$$x^* = \arg\min_{x \in \mathbb{R}^d} f(x)$$

The Gradient

- Let $f : \mathrm{R}^d \to \mathrm{R}$ be differentiable at $x_0 \in \mathrm{R}^d$.
- The **gradient** of f at the point x_0, denoted $\nabla_x f(x_0)$, is the direction in which $f(x)$ **increases fastest**, if we start from x_0.

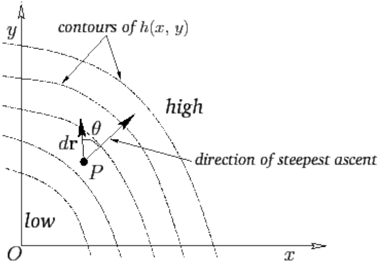


Figure A.111 from Newtonian Dynamics, by Richard Fitzpatrick.

Gradient Descent

- To reach a local minimum as fast as possible, we want to go in the opposite direction from the gradient.

Gradient Descent

- To reach a local minimum as fast as possible, we want to go in the opposite direction from the gradient.

Gradient Descent

- Initialize $x \leftarrow 0$.
- Repeat:
  - $x \leftarrow x - \eta \nabla f(x)$
- until the stopping criterion is satisfied.

Gradient Descent

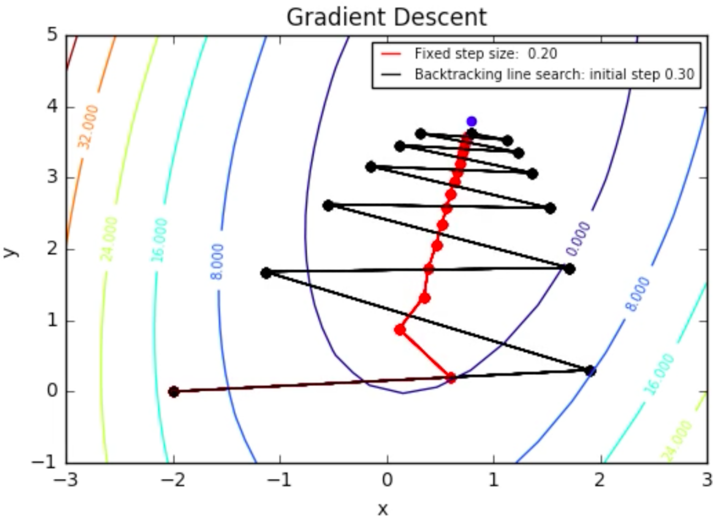
- To reach a local minimum as fast as possible, we want to go in the opposite direction from the gradient.

Gradient Descent

- Initialize $x \leftarrow 0$.
- Repeat:
  - $x \leftarrow x - \eta \nabla f(x)$
- until the stopping criterion is satisfied.

- The "step size" η is not the amount by which we update x!

Gradient Descent Path



Gradient Descent: Step Size

- A fixed step size will work, eventually, as long as it's small enough (roughly — details to come)

Gradient Descent: Step Size

- A fixed step size will work, eventually, as long as it's small enough (roughly — details to come)
  - If η is too large, the optimization process might diverge

Gradient Descent: Step Size

- A fixed step size will work, eventually, as long as it's small enough (roughly — details to come)
  - If η is too large, the optimization process might diverge
  - In practice, it often makes sense to try several fixed step sizes
- Intuition on when to take big steps and when to take small steps?

Convergence Theorem for Fixed Step Size

Theorem

Suppose $f : \mathbb{R}^d \to \mathbb{R}$ is convex and differentiable, and ∇f is ***Lipschitz continuous*** *with constant $L > 0$, i.e.*

$$\|\nabla f(x) - \nabla f(x')\| \leqslant L\|x - x'\|$$

for any $x, x' \in \mathbb{R}^d$. Then gradient descent with fixed step size $\eta \leqslant 1/L$ ***converges****. In particular,*

$$f(x^{(k)}) - f(x^*) \leqslant \frac{\|x^{(0)} - x^*\|^2}{2\eta k}.$$

This says that gradient descent is guaranteed to converge and that it converges with rate $O(1/k)$.

Gradient Descent: When to Stop?

- Wait until $\|\nabla f(x)\|_2 \leqslant \varepsilon$, for some ε of your choosing.
  - (Recall $\nabla f(x) = 0$ at a local minimum.)

Gradient Descent: When to Stop?

- Wait until $\|\nabla f(x)\|_2 \leqslant \varepsilon$, for some ε of your choosing.
  - (Recall $\nabla f(x) = 0$ at a local minimum.)
- Early stopping:
  - evalute loss on validation data after each iteration;
  - stop when the loss does not improve (or gets worse).

Gradient Descent for Empirical Risk - Scaling Issues

Quick recap: Gradient Descent for ERM

- We have a hypothesis space of functions $\mathcal{F} = \left\{ f_w : \mathcal{X} \to \mathcal{Y} \mid w \in \mathbf{R}^d \right\}$
  - Parameterized by $w \in \mathbf{R}^d$.
- Finding an empirical risk minimizer entails finding a w that minimizes

$$\hat{R}_n(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_w(x_i), y_i)$$

- Suppose $\ell(f_w(x_i), y_i)$ is differentiable as a function of w.
- Then we can do gradient descent on $\hat{R}_n(w)$

Gradient Descent: Scalability

- At every iteration, we compute the gradient at the current w:

$$\nabla \hat{R}_n(w) = \frac{1}{n}\sum_{i=1}^{n} \nabla_w \ell(f_w(x_i), y_i)$$

- How does this scale with n?

Gradient Descent: Scalability

- At every iteration, we compute the gradient at the current w:

$$\nabla \hat{R}_n(w) = \frac{1}{n}\sum_{i=1}^{n} \nabla_w \ell(f_w(x_i), y_i)$$

- How does this scale with n?
- We have to iterate over all n training points to take a single step. [$O(n)$]

Gradient Descent: Scalability

- At every iteration, we compute the gradient at the current w:

$$\nabla \hat{R}_n(w) = \frac{1}{n} \sum_{i=1}^{n} \nabla_w \ell(f_w(x_i), y_i)$$

- How does this scale with n?
- We have to iterate over all n training points to take a single step. [$O(n)$]
- Can we make progress without looking at all the data before updating w?

Stochastic Gradient Descent

"Noisy" Gradient Descent

- Instead of using the gradient, we use a noisy estimate of the gradient.
- Turns out this can work just fine!

“Noisy” Gradient Descent

- Instead of using the gradient, we use a noisy estimate of the gradient.
- Turns out this can work just fine!
- **Intuition**:
  - Gradient descent is an iterative procedure anyway.
  - At every step, we have a chance to recover from previous missteps.

Minibatch Gradient

- The **full gradient** is

$$\nabla \hat{R}_n(w) = \frac{1}{n}\sum_{i=1}^{n} \nabla_w \ell(f_w(x_i), y_i)$$

- It's an average over the **full batch** of data $\mathcal{D}_n = \{(x_1, y_1), \ldots, (x_n, y_n)\}$.

Minibatch Gradient

- The **full gradient** is

$$\nabla \hat{R}_n(w) = \frac{1}{n}\sum_{i=1}^{n} \nabla_w \ell(f_w(x_i), y_i)$$

- It's an average over the **full batch** of data $\mathcal{D}_n = \{(x_1, y_1), \ldots, (x_n, y_n)\}$.
- Let's take a random subsample of size N (called a **minibatch**):

$$(x_{m_1}, y_{m_1}), \ldots, (x_{m_N}, y_{m_N})$$

Minibatch Gradient

- The **full gradient** is

$$\nabla \hat{R}_n(w) = \frac{1}{n} \sum_{i=1}^{n} \nabla_w \ell(f_w(x_i), y_i)$$

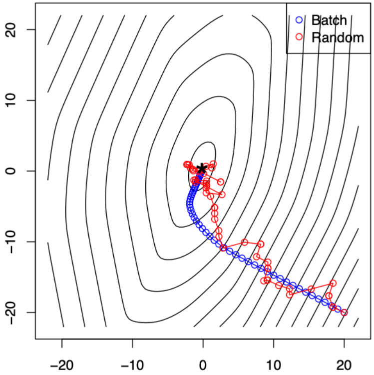
- It's an average over the **full batch** of data $\mathcal{D}_n = \{(x_1, y_1), \ldots, (x_n, y_n)\}$.
- Let's take a random subsample of size N (called a **minibatch**):

$$(x_{m_1}, y_{m_1}), \ldots, (x_{m_N}, y_{m_N})$$

- The **minibatch gradient is**

$$\nabla \hat{R}_N(w) = \frac{1}{N} \sum_{i=1}^{N} \nabla_w \ell(f_w(x_{m_i}), y_{m_i})$$

Batch vs Stochastic Methods



(Slide adapted from Ryan Tibshirani)

Rule of thumb for stochastic methods:

- Stochastic methods work well far from the optimum
- But struggle close the the optimum

Minibatch Gradient Properties

- The minibatch gradient is an **unbiased estimator** for the [full] batch gradient. What does that mean?

Minibatch Gradient Properties

- The minibatch gradient is an **unbiased estimator** for the [full] batch gradient. What does that mean?

$$\mathbb{E}\left[\nabla \hat{R}_N(w)\right] = \nabla \hat{R}_n(w)$$

Minibatch Gradient Properties

- The minibatch gradient is an **unbiased estimator** for the [full] batch gradient. What does that mean?

$$\mathbb{E}\left[\nabla \hat{R}_N(w)\right] = \nabla \hat{R}_n(w)$$

- The bigger the minibatch, the better the estimate.

$$\frac{1}{N}\mathrm{Var}\left[\nabla \hat{R}_1(w)\right] = \mathrm{Var}\left[\nabla \hat{R}_N(w)\right]$$

Minibatch Gradient Properties

- The minibatch gradient is an **unbiased estimator** for the [full] batch gradient. What does that mean?

$$\mathbb{E}\left[\nabla \hat{R}_N(w)\right] = \nabla \hat{R}_n(w)$$

- The bigger the minibatch, the better the estimate.

$$\frac{1}{N}\mathrm{Var}\left[\nabla \hat{R}_1(w)\right] = \mathrm{Var}\left[\nabla \hat{R}_N(w)\right]$$

- Tradeoffs of minibatch size:
  - Bigger $N \implies$ Better estimate of gradient, but slower (more data to process)
  - Smaller $N \implies$ Worse estimate of gradient, but can be quite fast

Minibatch Gradient Properties

- The minibatch gradient is an **unbiased estimator** for the [full] batch gradient. What does that mean?

$$\mathbb{E}\left[\nabla \hat{R}_N(w)\right] = \nabla \hat{R}_n(w)$$

- The bigger the minibatch, the better the estimate.

$$\frac{1}{N}\text{Var}\left[\nabla \hat{R}_1(w)\right] = \text{Var}\left[\nabla \hat{R}_N(w)\right]$$

- Tradeoffs of minibatch size:
  - Bigger $N \implies$ Better estimate of gradient, but slower (more data to process)
  - Smaller $N \implies$ Worse estimate of gradient, but can be quite fast
- Because of vectorization, we can often get minibatches of certain sizes for free

Convergence of SGD

- For convergence guarantee, use **diminishing step sizes**, e.g. $\eta_k = 1/k$
- Theoretically, GD is much faster than SGD in terms of convergence rate:
  - much faster to add a digit of accuracy.
  - but most of that advantage comes into play once we're already pretty close to the minimum.
  - However, in many ML problems we don't care about optimizing to high accuracy

Step Sizes in Minibatch Gradient Descent

Minibatch Gradient Descent (minibatch size N)

- initialize $w = 0$
- repeat
  - randomly choose N points $\{(x_i, y_i)\}_{i=1}^{N} \subset \mathcal{D}_n$
  - $w \leftarrow w - \eta \left[\frac{1}{N} \sum_{i=1}^{N} \nabla_w \ell(f_w(x_i), y_i) \right]$

- For SGD, fixed step size can work well in practice.
- Typical approach: Fixed step size reduced by constant factor whenever validation performance stops improving.
- Other schedules: inverse time decay, cosine decay, etc.

Summary

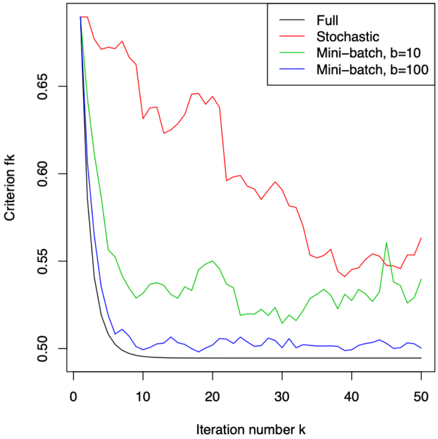
- **Gradient descent** or **"full-batch" gradient descent**
  - Use full data set of size n to determine step direction
- **Minibatch gradient descent**
  - Use a **random** subset of size N to determine step direction
- **Stochastic gradient descent**
  - Minibatch with $N = 1$.
  - Use a single randomly chosen point to determine step direction.

These days terminology isn't used so consistently, so always clarify the [mini]batch size.

SGD is much more efficient in time and memory cost and has been quite successful in large-scale ML.

Example: Logistic regression with ℓ_2 regularization

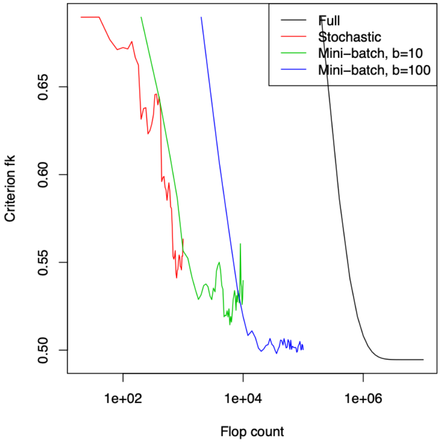
Batch methods converge faster :



(Example from Ryan Tibshirani)

Example: Logistic regression with ℓ_2 regularization

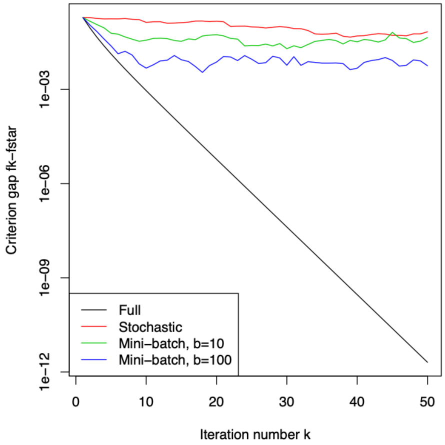
Stochastic methods are computationally more efficient:



(Example from Ryan Tibshirani)

Example: Logistic regression with ℓ_2 regularization

Batch methods are much faster close to the optimum:



(Example from Ryan Tibshirani)

Loss Functions: Regression

Regression Problems

- Examples:
  - Predicting the stock price given history prices
  - Predicting medical cost of given age, sex, region, BMI etc.
  - Predicting the age of a person based on their photos

Regression Problems

- Examples:
  - Predicting the stock price given history prices
  - Predicting medical cost of given age, sex, region, BMI etc.
  - Predicting the age of a person based on their photos
- Notation:
  - $\hat{y}$ is the predicted value (the action)
  - y is the actual observed value (the outcome)

Loss Functions for Regression

- A loss function in general:

$$(\hat{y}, y) \mapsto \ell(\hat{y}, y) \in R$$

Loss Functions for Regression

- A loss function in general:

$$(\hat{y}, y) \mapsto \ell(\hat{y}, y) \in \mathrm{R}$$

- Regression losses usually only depend on the **residual $r = y - \hat{y}$.**
  - what you have to add to your prediction to get the correct answer.

Loss Functions for Regression

- A loss function in general:

$$(\hat{y}, y) \mapsto \ell(\hat{y}, y) \in \mathbf{R}$$

- Regression losses usually only depend on the **residual** $r = y - \hat{y}$.
  - what you have to add to your prediction to get the correct answer.
- A loss $\ell(\hat{y}, y)$ is called **distance-based** if:
  1. It only depends on the residual:

$$\ell(\hat{y}, y) = \psi(y - \hat{y}) \quad \text{for some } \psi : \mathbf{R} \to \mathbf{R}$$

  2. It is zero when the residual is 0:

$$\psi(0) = 0$$

Distance-Based Losses are Translation Invariant

- Distance-based losses are translation-invariant. That is,

$$\ell(\hat{y}+b, y+b) = \ell(\hat{y}, y) \qquad \forall b \in \mathrm{R}.$$

- When might you not want to use a translation-invariant loss?

Distance-Based Losses are Translation Invariant

- Distance-based losses are translation-invariant. That is,

$$\ell(\hat{y}+b, y+b) = \ell(\hat{y}, y) \qquad \forall b \in \mathrm{R}.$$

- When might you not want to use a translation-invariant loss?
- Sometimes the relative error $\frac{\hat{y}-y}{y}$ is a more natural loss (but not translation-invariant)

Distance-Based Losses are Translation Invariant

- Distance-based losses are translation-invariant. That is,

$$\ell(\hat{y}+b, y+b) = \ell(\hat{y}, y) \qquad \forall b \in \mathrm{R}.$$

- When might you not want to use a translation-invariant loss?
- Sometimes the relative error $\frac{\hat{y}-y}{y}$ is a more natural loss (but not translation-invariant)
- Often you can transform response y so it's translation-invariant (e.g. log transform)

Some Losses for Regression

- **Residual:** $r = y - \hat{y}$
- **Square** or ℓ_2 Loss: $\ell(r) = r^2$

Some Losses for Regression

- **Residual:** $r = y - \hat{y}$
- **Square** or ℓ_2 Loss: $\ell(r) = r^2$
- **Absolute** or **Laplace** or ℓ_1 Loss: $\ell(r) = |r|$

Some Losses for Regression

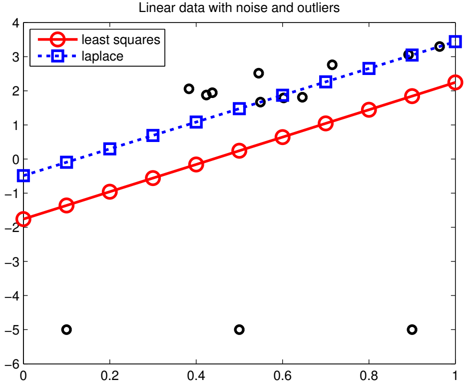
- **Residual:** $r = y - \hat{y}$
- **Square** or ℓ_2 Loss: $\ell(r) = r^2$
- **Absolute** or **Laplace** or ℓ_1 Loss: $\ell(r) = |r|$

y	$\hat{y}$	$\|r\| = \|y - \hat{y}\|$	$r^2 = (y - \hat{y})^2$
1	0	1	1
5	0	5	25
10	0	10	100
50	0	50	2500

- Outliers typically have large residuals. (What is an outlier?)
- Square loss much more affected by outliers than absolute loss.

Loss Function Robustness

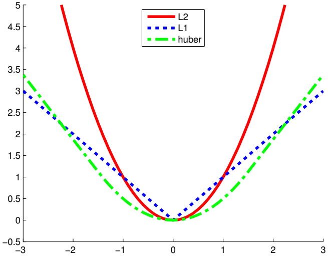
- **Robustness** refers to how affected a learning algorithm is by outliers.



KPM Figure 7.6

Some Losses for Regression

- **Square** or ℓ_2 Loss: $\ell(r) = r^2$ (*not robust*)
- **Absolute** or **Laplace** Loss: $\ell(r) = |r|$ (*not differentiable*)
  - gives **median regression**
- **Huber** Loss: Quadratic for $|r| \leqslant \delta$ and linear for $|r| > \delta$ (*robust and differentiable*)
  - Equal values and slopes at $r = \delta$



KPM Figure 7.6

Classification Loss Functions

The Classification Problem

- Examples:
  - Predict whether the image contains a cat
  - Predict whether the email is SPAM

The Classification Problem

- Examples:
  - Predict whether the image contains a cat
  - Predict whether the email is SPAM
- Classification spaces:

The Classification Problem

- Examples:
  - Predict whether the image contains a cat
  - Predict whether the email is SPAM
- Classification spaces:
- Inference:

$$
\begin{aligned}
f(x) > 0 &\implies \text{Predict } 1 \\
f(x) < 0 &\implies \text{Predict } -1
\end{aligned}
$$

The Score Function

- Output space $\mathcal{Y} = \{-1, 1\}$
- **Real-valued prediction function** $f : \mathcal{X} \to \mathrm{R}$

Definition

The value $f(x)$ is called the **score** for the input x.

- In this context, f may be called a **score function**.
- The magnitude of the score can be interpreted as our **confidence of our prediction**.

The Margin

Definition

The **margin** (or **functional margin**) for a predicted score $\hat{y}$ and the true class $y \in \{-1, 1\}$ is $y\hat{y}$.

The Margin

Definition

The **margin** (or **functional margin**) for a predicted score $\hat{y}$ and the true class $y \in \{-1, 1\}$ is $y\hat{y}$.

- The margin is often written as $yf(x)$, where $f(x)$ is our score function.
- The margin is a measure of how **correct** we are:
  - If y and $\hat{y}$ are the same sign, prediction is **correct** and margin is **positive**.
  - If y and $\hat{y}$ have different sign, prediction is **incorrect** and margin is **negative**.
- We want to **maximize the margin**
- Most classification losses depend only on the margin (they are **margin-based losses**).

Classification Losses: $0-1$ Loss

- If $\tilde{f}$ is the inference function (1 if $f(x) > 0$ and -1 otherwise), then
- The **0-1 loss** for $f : \mathcal{X} \to \{-1, 1\}$:

$$\ell(f(x), y) = 1(\tilde{f}(x) \neq y)$$

- Empirical risk for $0-1$ loss:

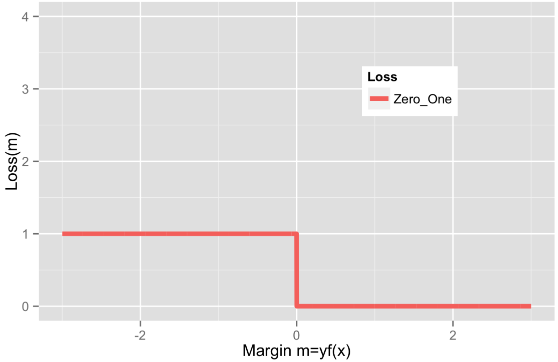
$$\hat{R}_n(f) = \frac{1}{n}\sum_{i=1}^{n} 1(y_i f(x_i) \leqslant 0)$$

Minimizing empirical $0-1$ risk not computationally feasible

$\hat{R}_n(f)$ is non-convex, not differentiable, and even discontinuous.

Classification Losses

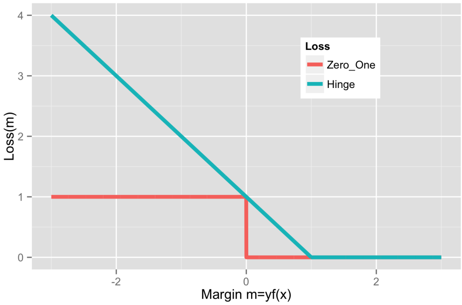
Zero-One loss: $\ell_{0\text{-}1} = 1(m \leqslant 0)$



- x-axis is **margin**: $m > 0 \iff$ correct classification

Hinge Loss

SVM/Hinge loss: $\ell_{\text{Hinge}} = \max(1 - m, 0)$



Hinge is a **convex**, **upper bound** on $0-1$ loss. Not differentiable at $m=1$.

We will cover SVM and Hinge loss in more details in week 4.

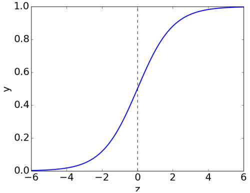
Logistic Regression

- Aka linear classification. Logistic regression is not actually "regression."
- Two equivalent types of logistic regression losses.

Logistic Regression

- Aka linear classification. Logistic regression is not actually "regression."
- Two equivalent types of logistic regression losses.
- If the label is 0 or 1:
- $\hat{y} = \sigma(z)$, where σ is the sigmoid function.

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$



Logistic Regression

- If the label is 0 or 1:
- $\hat{y} = \sigma(z)$, where σ is the sigmoid function.

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

Logistic Regression

- If the label is 0 or 1:
- $\hat{y} = \sigma(z)$, where σ is the sigmoid function.

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

- The loss is binary cross entropy:

$$\ell_{\text{Logistic}} = -y \log(\hat{y}) - (1-y) \log(1-\hat{y})$$

Logistic Regression

- If the label is 0 or 1:
- $\hat{y} = \sigma(z)$, where σ is the sigmoid function.

$$\sigma(z) = \frac{1}{1+\exp(-z)}$$

- The loss is binary cross entropy:

$$\ell_{\text{Logistic}} = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

- Remember the negative sign!

Logistic Regression

- If the label is -1 o 1:
- Note: $1 - \sigma(z) = \sigma(-z)$

Logistic Regression

- If the label is -1 o 1:
- Note: $1 - \sigma(z) = \sigma(-z)$
- Now we can derive an equivalent loss form:

$$
\begin{aligned}
\ell_{\text{Logistic}} &= \begin{cases} -\log(\sigma(z)) & \text{if} \quad y = 1 \\ -\log(\sigma(-z)) & \text{if} \quad y = -1 \end{cases} \\
&= -\log(\sigma(yz)) \\
&= -\log(\frac{1}{1 + e^{-yz}}) \\
&= \log(1 + e^{-m}).
\end{aligned}
$$

Logistic Regression

- If the label is -1 o 1:
- Note: $1 - \sigma(z) = \sigma(-z)$
- Now we can derive an equivalent loss form:

$$\ell_{\text{Logistic}} = \begin{cases} -\log(\sigma(z)) & \text{if} \quad y = 1 \\ -\log(\sigma(-z)) & \text{if} \quad y = -1 \end{cases}$$

$$= -\log(\frac{1}{1 + e^{-yz}})$$
$$= \log(1 + e^{-m}).$$

Logistic Regression

- If the label is -1 o 1:
- Note: $1-\sigma(z)=\sigma(-z)$
- Now we can derive an equivalent loss form:

$$\ell_{\text{Logistic}} = \begin{cases} -\log(\sigma(z)) & \text{if} \quad y=1 \\ -\log(\sigma(-z)) & \text{if} \quad y=-1 \end{cases}$$

$$= \log(1+e^{-m}).$$

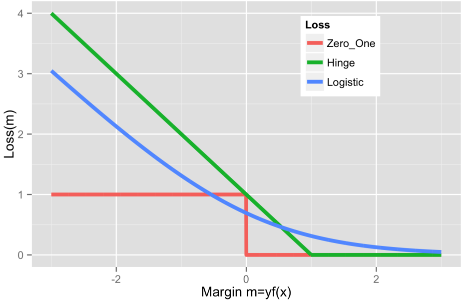
Logistic Regression

- If the label is -1 o 1:
- Note: $1-\sigma(z)=\sigma(-z)$
- Now we can derive an equivalent loss form:

$$
\begin{aligned}
\ell_{\text{Logistic}} &= \begin{cases} -\log(\sigma(z)) & \text{if} \quad y=1 \\ -\log(\sigma(-z)) & \text{if} \quad y=-1 \end{cases} \\
&= -\log(\sigma(yz)) \\
&= -\log(\frac{1}{1+e^{-yz}}) \\
&= \log(1+e^{-m}).
\end{aligned}
$$

Logistic Loss

Logistic/Log loss: $\ell_{\text{Logistic}} = \log\left(1 + e^{-m}\right)$



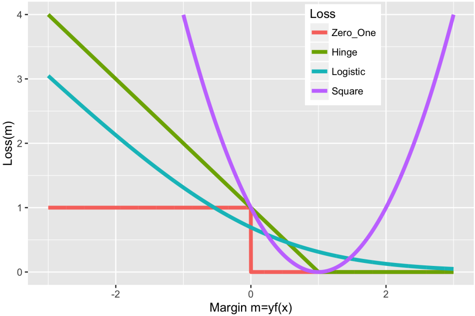
Logistic loss is differentiable. Logistic loss always rewards a larger margin (the loss is never 0).

What About Square Loss for Classification?

- Loss $\ell(f(x), y) = (f(x) - y)^2$.
- Turns out, can write this in terms of margin $m = f(x)y$:
- Using fact that $y^2 = 1$, since $y \in \{-1, 1\}$.

$$\begin{aligned}
\ell(f(x), y) &= (f(x) - y)^2 \\
&= f^2(x) - 2f(x)y + y^2 \\
&= f^2(x)y^2 - 2f(x)y + 1 \\
&= (1 - f(x)y)^2 \\
&= (1 - m)^2
\end{aligned}$$

What About Square Loss for Classification?



Heavily penalizes outliers (e.g. mislabeled examples).

Summary

- Gradient descent: step size/learning rate, batch size, convergence

Summary

- Gradient descent: step size/learning rate, batch size, convergence
- Loss functions for regression and classification problems.

Summary

- Gradient descent: step size/learning rate, batch size, convergence
- Loss functions for regression and classification problems.
- Regression: Squared (L2) loss, Absolute (L1) loss, Huber loss.

Summary

- Gradient descent: step size/learning rate, batch size, convergence
- Loss functions for regression and classification problems.
- Regression: Squared (L2) loss, Absolute (L1) loss, Huber loss.
- Classification: Hinge loss, Logistic loss.

Summary

- Gradient descent: step size/learning rate, batch size, convergence
- Loss functions for regression and classification problems.
- Regression: Squared (L2) loss, Absolute (L1) loss, Huber loss.
- Classification: Hinge loss, Logistic loss.
- Residual, margin

Summary

- Gradient descent: step size/learning rate, batch size, convergence
- Loss functions for regression and classification problems.
- Regression: Squared (L2) loss, Absolute (L1) loss, Huber loss.
- Classification: Hinge loss, Logistic loss.
- Residual, margin
- Logistic regression