Numerical optimization
Numerical Optimization Methods
Many ML methods are specified via numerical optimization problem. Must be combined with a method for solving the problem to really obtain a ML algorithm.

Gradient descent—iterative algorithm for certain numerical optimization problems—is the workhorse of many ML methods.
Numerical optimization

Many numerical optimization problems in ML look like

$$\min_{\vec{w} \in \mathbb{R}^d} J(\vec{w})$$

- $\vec{w} = (w_1, \ldots, w_d)$ are optimization variables (e.g., parameters of predictor)
- $J: \mathbb{R}^d \rightarrow \mathbb{R}$ is objective function (e.g., sum of “data fitting objective” and “regularizer”)
- Here, we assume variables are unconstrained; all vectors in $\mathbb{R}^d$ are allowed

But in general, may also have constraints on allowed $\vec{w}$

Want: Algorithm that, given $J$, finds particular setting of $\vec{w} \in \mathbb{R}^d$ so that $J(\vec{w})$ is as small as possible

Questions:
1. What is meant by “given $J$”? (I.e., what do we need to “know” about $J$?)
2. For what types of objective functions is this possible (in a reasonable amount of time)?
Numerical optimization

Many numerical optimization problems in ML look like

$$\min_{\tilde{w} \in \mathbb{R}^d} J(\tilde{w})$$

- $\tilde{w} = (w_1, \ldots, w_d)$ are optimization variables (e.g., parameters of predictor)
Numerical optimization

Many numerical optimization problems in ML look like

$$\min_{\vec{w} \in \mathbb{R}^d} J(\vec{w})$$

- $\vec{w} = (w_1, \ldots, w_d)$ are **optimization variables** (e.g., parameters of predictor)
- $J : \mathbb{R}^d \rightarrow \mathbb{R}$ is **objective function** (e.g., sum of “data fitting objective” and “regularizer”)
Numerical optimization

Many numerical optimization problems in ML look like

$$\min_{\vec{w} \in \mathbb{R}^d} J(\vec{w})$$

- $\vec{w} = (w_1, \ldots, w_d)$ are optimization variables (e.g., parameters of predictor)
- $J : \mathbb{R}^d \rightarrow \mathbb{R}$ is objective function (e.g., sum of “data fitting objective” and “regularizer”)
- Here, we assume variables are unconstrained; all vectors in $\mathbb{R}^d$ are allowed
  (But in general, may also have constraints on allowed $\vec{w}$)
Numerical optimization

Many numerical optimization problems in ML look like

$$\min_{\vec{w} \in \mathbb{R}^d} J(\vec{w})$$

- $\vec{w} = (w_1, \ldots, w_d)$ are optimization variables (e.g., parameters of predictor)
- $J : \mathbb{R}^d \rightarrow \mathbb{R}$ is objective function (e.g., sum of “data fitting objective” and “regularizer”)
- Here, we assume variables are unconstrained; all vectors in $\mathbb{R}^d$ are allowed (But in general, may also have constraints on allowed $\vec{w}$)

Want: Algorithm that, given $J$, finds particular setting of $\vec{w} \in \mathbb{R}^d$ so that $J(\vec{w})$ is as small as possible
Numerical optimization

Many numerical optimization problems in ML look like

\[
\min_{\vec{w} \in \mathbb{R}^d} J(\vec{w})
\]

- \(\vec{w} = (w_1, \ldots, w_d)\) are optimization variables (e.g., parameters of predictor)
- \(J: \mathbb{R}^d \rightarrow \mathbb{R}\) is objective function (e.g., sum of “data fitting objective” and “regularizer”)
- Here, we assume variables are unconstrained; all vectors in \(\mathbb{R}^d\) are allowed
  (But in general, may also have constraints on allowed \(\vec{w}\))

Want: Algorithm that, given \(J\), finds particular setting of \(\vec{w} \in \mathbb{R}^d\) so that \(J(\vec{w})\) is as small as possible

Questions:
1. What is meant by “given \(J\)”? (I.e., what do we need to “know” about \(J\)?)
Numerical optimization

Many numerical optimization problems in ML look like

$$\min_{\vec{w} \in \mathbb{R}^d} J(\vec{w})$$

- $\vec{w} = (w_1, \ldots, w_d)$ are optimization variables (e.g., parameters of predictor)
- $J : \mathbb{R}^d \to \mathbb{R}$ is objective function (e.g., sum of “data fitting objective” and “regularizer”)
- Here, we assume variables are unconstrained; all vectors in $\mathbb{R}^d$ are allowed
  (But in general, may also have constraints on allowed $\vec{w}$)

Want: Algorithm that, given $J$, finds particular setting of $\vec{w} \in \mathbb{R}^d$ so that $J(\vec{w})$ is as small as possible

Questions:
1. What is meant by “given $J$”? (I.e., what do we need to “know” about $J$?)
2. For what types of objective functions is this possible (in a reasonable amount of time)?
Calculus detour: Differentiable functions

(In this lecture, we'll exclusively consider differentiable objective functions)
Calculus detour: Differentiable functions

(In this lecture, we'll exclusively consider differentiable objective functions)

A function \( J : \mathbb{R}^d \to \mathbb{R} \) is **differentiable** if, for every \( \vec{w}_0 \in \mathbb{R}^d \), there is an affine function \( A : \mathbb{R}^d \to \mathbb{R} \) such that

\[
\lim_{\vec{w} \to \vec{w}_0} \frac{J(\vec{w}) - A(\vec{w})}{\|\vec{w} - \vec{w}_0\|_2} = 0
\]

and the affine function \( A \) is called the *(best) affine approximation of \( J \) at \( \vec{w}_0 \)
Calculus detour: Differentiable functions

(In this lecture, we'll exclusively consider differentiable objective functions)

A function \( J : \mathbb{R}^d \to \mathbb{R} \) is \textbf{differentiable} if, for every \( \bar{w}_0 \in \mathbb{R}^d \), there is an \textit{affine function} \( A : \mathbb{R}^d \to \mathbb{R} \) such that

\[
\lim_{\bar{w} \to \bar{w}_0} \frac{J(\bar{w}) - A(\bar{w})}{\|\bar{w} - \bar{w}_0\|_2} = 0
\]

and the affine function \( A \) is called the \textbf{(best) affine approximation of} \( J \) at \( \bar{w}_0 \)

Affine function \( A \) is a good approximation of \( J \) in vicinity of \( \bar{w}_0 \)
Calculus detour: Differentiable functions

(In this lecture, we'll exclusively consider differentiable objective functions)

A function $J: \mathbb{R}^d \to \mathbb{R}$ is **differentiable** if, for every $\vec{w}_0 \in \mathbb{R}^d$, there is an *affine function* $A: \mathbb{R}^d \to \mathbb{R}$ such that

$$
\lim_{\vec{w} \to \vec{w}_0} \frac{J(\vec{w}) - A(\vec{w})}{\|\vec{w} - \vec{w}_0\|_2} = 0
$$

and the affine function $A$ is called the *(best) affine approximation of $J$ at $\vec{w}_0$*

Affine function $A$ is a good approximation of $J$ in vicinity of $\vec{w}_0$

- Affine function $A$ may depend on $\vec{w}_0$ — i.e., (possibly) different $A$ for different $\vec{w}_0$
Calculus detour: Differentiable functions

(In this lecture, we'll exclusively consider differentiable objective functions)

A function $J : \mathbb{R}^d \to \mathbb{R}$ is **differentiable** if, for every $\vec{w}_0 \in \mathbb{R}^d$, there is an **affine function** $A : \mathbb{R}^d \to \mathbb{R}$ such that

$$
\lim_{\vec{w} \to \vec{w}_0} \frac{J(\vec{w}) - A(\vec{w})}{\|\vec{w} - \vec{w}_0\|_2} = 0
$$

and the affine function $A$ is called the **(best) affine approximation of $J$ at $\vec{w}_0$**

Affine function $A$ is a good approximation of $J$ in vicinity of $\vec{w}_0$

- Affine function $A$ may depend on $\vec{w}_0$ — i.e., (possibly) different $A$ for different $\vec{w}_0$
- Not every function is differentiable 😐
Best affine approximation

How do we get a handle on the best affine approximation of $J$ at $\vec{w}_0$?
Best affine approximation

How do we get a handle on the best affine approximation of $J$ at $\vec{w}_0$?

▶ It is an affine function, so can write it as

$$A(\vec{w}) = \vec{m} \cdot \vec{w} + \theta$$

▶ $\vec{m} \in \mathbb{R}^d$ is the “slope” (and specifies a linear function)
▶ $\theta \in \mathbb{R}$ is the “intercept”
Best affine approximation

How do we get a handle on the best affine approximation of $J$ at $\vec{w}_0$?

- It is an affine function, so can write it as

$$ A(\vec{w}) = \vec{m} \cdot \vec{w} + \theta $$

- $\vec{m} \in \mathbb{R}^d$ is the "slope" (and specifies a linear function)
- $\theta \in \mathbb{R}$ is the "intercept"

- Intercept has to be $\theta = J(\vec{w}_0) - \vec{m} \cdot \vec{w}_0$, because

$$ J(\vec{w}_0) = A(\vec{w}_0) = \vec{m} \cdot \vec{w}_0 + \theta $$
How do we get a handle on the best affine approximation of $J$ at $\vec{w}_0$?

- It is an affine function, so can write it as

$$A(\vec{w}) = \vec{m} \cdot \vec{w} + \theta$$

- $\vec{m} \in \mathbb{R}^d$ is the “slope” (and specifies a linear function)
- $\theta \in \mathbb{R}$ is the “intercept”

- Intercept has to be $\theta = J(\vec{w}_0) - \vec{m} \cdot \vec{w}_0$, because

$$J(\vec{w}_0) = A(\vec{w}_0) = \vec{m} \cdot \vec{w}_0 + \theta$$

- This means we can write $A$ as

$$A(\vec{w}) = \vec{m} \cdot \vec{w} + (J(\vec{w}_0) - \vec{m} \cdot \vec{w}_0)$$

$$= J(\vec{w}_0) + \vec{m} \cdot (\vec{w} - \vec{w}_0)$$
How do we get a handle on the best affine approximation of $J$ at $\vec{w}_0$?

- It is an affine function, so can write it as
  \[ A(\vec{w}) = \vec{m} \cdot \vec{w} + \theta \]

  - $\vec{m} \in \mathbb{R}^d$ is the “slope” (and specifies a linear function)
  - $\theta \in \mathbb{R}$ is the “intercept”

- Intercept has to be $\theta = J(\vec{w}_0) - \vec{m} \cdot \vec{w}_0$, because
  \[ J(\vec{w}_0) = A(\vec{w}_0) = \vec{m} \cdot \vec{w}_0 + \theta \]

- This means we can write $A$ as
  \[ A(\vec{w}) = \vec{m} \cdot \vec{w} + (J(\vec{w}_0) - \vec{m} \cdot \vec{w}_0) \]
  \[ = J(\vec{w}_0) + \vec{m} \cdot (\vec{w} - \vec{w}_0) \]

- So what is $\vec{m}$?
  (Note: Like the intercept $\theta$, the slope $\vec{m}$ may depend on $\vec{w}_0$)
Slope of best affine approximation

Let’s get a handle on the slope $\vec{m}$ of the best affine approximation of $J$ at $\vec{w}_0$. 

$\vec{m} = (m_1, m_2, \ldots, m_d)$ in terms of standard coordinate basis $\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_d$ ($\vec{e}_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ has 1 in the $i$-th position, 0s elsewhere).

By virtue of $A(\vec{w}) = J(\vec{w}_0) + \vec{m} \cdot (\vec{w} - \vec{w}_0)$ being the best affine approximation of $J$ at $\vec{w}_0$,

$$\frac{0}{t} = \lim_{t \rightarrow 0} J(\vec{w}_0 + t\vec{e}_i) - A(\vec{w}_0 + t\vec{e}_i) \mid t \mid$$

($\vec{w}_0 + t\vec{e}_i$ differs from $\vec{w}_0$ by $t \in \mathbb{R}$ in $i$-th coordinate).

Whether $t$ approaches zero from the left or right, we find $m_i = \lim_{t \rightarrow 0} J(\vec{w}_0 + t\vec{e}_i) - J(\vec{w}_0) / t = \frac{\partial J}{\partial w_i}(\vec{w}_0)$ (partial derivative of $J$ w.r.t. variable $w_i$, evaluated at $\vec{w}_0$).

Vector of all partial derivatives of $J$ evaluated at $\vec{w}_0$ is called the gradient of $J$ at $\vec{w}_0$, written as $\nabla J(\vec{w}_0) = \frac{\partial J}{\partial w_1}(\vec{w}_0), \ldots, \frac{\partial J}{\partial w_d}(\vec{w}_0)$.
Slope of best affine approximation

Let’s get a handle on the slope $\vec{m}$ of the best affine approximation of $J$ at $\vec{w}_0$

- Write $\vec{m} = (m_1, m_2, \ldots, m_d)$ in terms of standard coordinate basis $\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_d$
  
  $(\vec{e}_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ has 1 in the $i$-th position, 0s elsewhere)
Let’s get a handle on the slope $\vec{m}$ of the best affine approximation of $J$ at $\vec{w}_0$

- Write $\vec{m} = (m_1, m_2, \ldots, m_d)$ in terms of standard coordinate basis $\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_d$
  $(\vec{e}_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ has 1 in the $i$-th position, 0s elsewhere)

- By virtue of $A(\vec{w}) = J(\vec{w}_0) + \vec{m} \cdot (\vec{w} - \vec{w}_0)$ being best affine approximation of $J$ at $\vec{w}_0$,

$$0 = \lim_{t \to 0} \frac{J(\vec{w}_0 + t\vec{e}_i) - A(\vec{w}_0 + t\vec{e}_i)}{|t|} = \lim_{t \to 0} \frac{J(\vec{w}_0 + t\vec{e}_i) - (J(\vec{w}_0) + tm_i)}{|t|}$$

($\vec{w}_0 + t\vec{e}_i$ differs from $\vec{w}_0$ by $t \in \mathbb{R}$ in $i$-th coordinate)
Let’s get a handle on the slope $\vec{m}$ of the best affine approximation of $J$ at $\vec{w}_0$

- Write $\vec{m} = (m_1, m_2, \ldots, m_d)$ in terms of standard coordinate basis $\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_d$
  
  ($\vec{e}_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ has 1 in the $i$-th position, 0s elsewhere)

- By virtue of $A(\vec{w}) = J(\vec{w}_0) + \vec{m} \cdot (\vec{w} - \vec{w}_0)$ being *best affine approximation of $J$ at $\vec{w}_0$*,

$$0 = \lim_{t \to 0} \frac{J(\vec{w}_0 + t\vec{e}_i) - A(\vec{w}_0 + t\vec{e}_i)}{|t|} = \lim_{t \to 0} \frac{J(\vec{w}_0 + t\vec{e}_i) - (J(\vec{w}_0) + tm_i)}{|t|}$$

($\vec{w}_0 + t\vec{e}_i$ differs from $\vec{w}_0$ by $t \in \mathbb{R}$ in $i$-th coordinate)

- Whether $t$ approaches zero from the left or right, we find

$$m_i = \lim_{t \to 0} \frac{J(\vec{w}_0 + t\vec{e}_i) - J(\vec{w}_0)}{t} = \frac{\partial J}{\partial w_i}(\vec{w}_0)$$

(partial derivative of $J$ w.r.t. variable $w_i$, evaluated at $\vec{w}_0$)
Slope of best affine approximation

Let's get a handle on the slope $\vec{m}$ of the best affine approximation of $J$ at $\vec{w}_0$

▶ Write $\vec{m} = (m_1, m_2, \ldots, m_d)$ in terms of standard coordinate basis $\vec{e}_1, \vec{e}_2, \ldots, \vec{e}_d$
($\vec{e}_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ has 1 in the $i$-th position, 0s elsewhere)

▶ By virtue of $A(\vec{w}) = J(\vec{w}_0) + \vec{m} \cdot (\vec{w} - \vec{w}_0)$ being best affine approximation of $J$ at $\vec{w}_0$,

$$0 = \lim_{t \to 0} \frac{J(\vec{w}_0 + t\vec{e}_i) - A(\vec{w}_0 + t\vec{e}_i)}{|t|} = \lim_{t \to 0} \frac{J(\vec{w}_0 + t\vec{e}_i) - (J(\vec{w}_0) + tm_i)}{|t|}$$
($\vec{w}_0 + t\vec{e}_i$ differs from $\vec{w}_0$ by $t \in \mathbb{R}$ in $i$-th coordinate)

▶ Whether $t$ approaches zero from the left or right, we find

$$m_i = \lim_{t \to 0} \frac{J(\vec{w}_0 + t\vec{e}_i) - J(\vec{w}_0)}{t} = \frac{\partial J}{\partial w_i}(\vec{w}_0)$$
(partial derivative of $J$ w.r.t. variable $w_i$, evaluated at $\vec{w}_0$)

▶ Vector of all partial derivatives of $J$ evaluated at $\vec{w}_0$ is called the gradient of $J$ at $\vec{w}_0$, written as

$$\nabla J(\vec{w}_0) = \left( \frac{\partial J}{\partial w_1}(\vec{w}_0), \ldots, \frac{\partial J}{\partial w_d}(\vec{w}_0) \right)$$

Think of $\nabla J : \mathbb{R}^d \to \mathbb{R}^d$ as a vector-valued function (a.k.a. vector field)
A function $J: \mathbb{R}^d \to \mathbb{R}$ is **differentiable** if, for every $\vec{w}_0 \in \mathbb{R}^d$, there is an affine function $A: \mathbb{R}^d \to \mathbb{R}$ (called the (best) **affine approximation of** $J$ **at** $\vec{w}_0$) such that

$$\lim_{\vec{w} \to \vec{w}_0} \frac{J(\vec{w}) - A(\vec{w})}{\|\vec{w} - \vec{w}_0\|_2} = 0$$

and $A$ is given by

$$A(\vec{w}) = J(\vec{w}_0) + \nabla J(\vec{w}_0) \cdot (\vec{w} - \vec{w}_0)$$

where

$$\nabla J(\vec{w}_0) = \left( \frac{\partial J}{\partial w_1}(\vec{w}_0), \ldots, \frac{\partial J}{\partial w_d}(\vec{w}_0) \right)$$

is the **gradient of** $J$ **at** $\vec{w}_0$
Local improvement via gradients

Back to optimization: \( \min_{\vec{w} \in \mathbb{R}^d} J(\vec{w}) \)

Suppose we have candidate setting of variables \( \vec{w} = \vec{w}_0 \in \mathbb{R}^d \), achieving objective value \( J(\vec{w}_0) \)

Question: How can we change \( \vec{w}_0 \) to achieve a lower objective value?

▶ Consider adding \( \vec{δ} \in \mathbb{R}^d \) to \( \vec{w}_0 \)

▶ If \( \vec{δ} \) is "small enough", then \( J(\vec{w}_0 + \vec{δ}) \approx J(\vec{w}_0) + \nabla J(\vec{w}_0) \cdot \vec{δ} \)

▶ Dot product term on right-hand side is negative if angle between \( \nabla J(\vec{w}_0) \) and \( \vec{δ} \) is obtuse

▶ Most obtuse angle is \( 180^\circ \), achieved by \( \vec{δ} := -\eta \nabla J(\vec{w}_0) \) for \( \eta > 0 \), in which case

\[
J(\vec{w}_0) + \nabla J(\vec{w}_0) \cdot \vec{δ} = J(\vec{w}_0) - \|\nabla J(\vec{w}_0)\|^2 \|\vec{δ}\|^2
\]

Upshot: Modify \( \vec{w}_0 \) by subtracting \( \eta \nabla J(\vec{w}_0) \) in hopes of getting lower objective value

Caveat: Approximation is OK only if \( \vec{δ} \) is "small enough" (i.e., \( \eta \) is "small enough")
Local improvement via gradients

Back to optimization: \( \min_{\vec{w} \in \mathbb{R}^d} J(\vec{w}) \)

Suppose we have candidate setting of variables \( \vec{w} = \vec{w}_0 \in \mathbb{R}^d \), achieving objective value \( J(\vec{w}_0) \)

**Question:** How can we change \( \vec{w}_0 \) to achieve a lower objective value?

Consider adding \( \vec{\delta} \in \mathbb{R}^d \) to \( \vec{w}_0 \)

If \( \vec{\delta} \) is "small enough", then

\[
J(\vec{w}_0 + \vec{\delta}) \approx J(\vec{w}_0) + \nabla J(\vec{w}_0) \cdot \vec{\delta}
\]

Dot product term on right-hand side is negative if angle between \( \nabla J(\vec{w}_0) \) and \( \vec{\delta} \) is obtuse

Most obtuse angle is \( 180^\circ \), achieved by \( \vec{\delta} := -\eta \nabla J(\vec{w}_0) \) for \( \eta > 0 \), in which case

\[
J(\vec{w}_0) + \nabla J(\vec{w}_0) \cdot \vec{\delta} = J(\vec{w}_0) - \| \nabla J(\vec{w}_0) \|_2^2 \| \vec{\delta} \|_2^2
\]

Upshot: Modify \( \vec{w}_0 \) by subtracting \( \eta \nabla J(\vec{w}_0) \) in hopes of getting lower objective value

**Caveat:** Approximation is OK only if \( \vec{\delta} \) is "small enough" (i.e., \( \eta \) is "small enough")
Local improvement via gradients

Back to optimization: \( \min_{\vec{w} \in \mathbb{R}^d} J(\vec{w}) \)

Suppose we have candidate setting of variables \( \vec{w} = \vec{w}_0 \in \mathbb{R}^d \), achieving objective value \( J(\vec{w}_0) \)

**Question:** How can we change \( \vec{w}_0 \) to achieve a lower objective value?

▶ Consider adding \( \vec{\delta} \in \mathbb{R}^d \) to \( \vec{w}_0 \)

\[ J(\vec{w} + \vec{\delta}) \approx J(\vec{w}) + \nabla J(\vec{w}) \cdot \vec{\delta} \]

Most obtuse angle is \( 180^\circ \), achieved by \( \vec{\delta} := -\eta \nabla J(\vec{w}_0) \) for \( \eta > 0 \), in which case

\[ J(\vec{w}_0) + \nabla J(\vec{w}_0) \cdot \vec{\delta} = J(\vec{w}_0) - \| \nabla J(\vec{w}_0) \|_2^2 \| \vec{\delta} \|_2^2 \]

**Upshot:** Modify \( \vec{w}_0 \) by subtracting \( \eta \nabla J(\vec{w}_0) \) in hopes of getting lower objective value

**Caveat:** Approximation is OK only if \( \vec{\delta} \) is "small enough" (i.e., \( \eta \) is "small enough")
Back to optimization: \( \min_{\tilde{w} \in \mathbb{R}^d} J(\tilde{w}) \)

Suppose we have candidate setting of variables \( \tilde{w} = \tilde{w}_0 \in \mathbb{R}^d \), achieving objective value \( J(\tilde{w}_0) \)

**Question:** How can we change \( \tilde{w}_0 \) to achieve a lower objective value?

- Consider adding \( \delta \in \mathbb{R}^d \) to \( \tilde{w}_0 \)
- If \( \delta \) is “small enough”, then

\[
J(\tilde{w}_0 + \delta) \approx J(\tilde{w}_0) + \nabla J(\tilde{w}_0) \cdot \delta
\]

**Upshot:** Modify \( \tilde{w}_0 \) by subtracting \( \eta \nabla J(\tilde{w}_0) \) in hopes of getting lower objective value

**Caveat:** Approximation is OK only if \( \delta \) is “small enough” (i.e., \( \eta \) is “small enough”)
Local improvement via gradients

Back to optimization: \( \min_{\vec{w} \in \mathbb{R}^d} J(\vec{w}) \)

Suppose we have candidate setting of variables \( \vec{w} = \vec{w}_0 \in \mathbb{R}^d \), achieving objective value \( J(\vec{w}_0) \)

**Question:** How can we change \( \vec{w}_0 \) to achieve a lower objective value?

- Consider adding \( \vec{\delta} \in \mathbb{R}^d \) to \( \vec{w}_0 \)
- If \( \vec{\delta} \) is “small enough”, then

\[
J(\vec{w}_0 + \vec{\delta}) \approx J(\vec{w}_0) + \nabla J(\vec{w}_0) \cdot \vec{\delta}
\]

- Dot product term on right-hand side is negative if angle between \( \nabla J(\vec{w}_0) \) and \( \vec{\delta} \) is obtuse
Local improvement via gradients

Back to optimization: \( \min_{\vec{w} \in \mathbb{R}^d} J(\vec{w}) \)

Suppose we have candidate setting of variables \( \vec{w} = \vec{w}_0 \in \mathbb{R}^d \), achieving objective value \( J(\vec{w}_0) \)

**Question:** How can we change \( \vec{w}_0 \) to achieve a lower objective value?

- Consider adding \( \vec{\delta} \in \mathbb{R}^d \) to \( \vec{w}_0 \)
- If \( \vec{\delta} \) is “small enough”, then

\[
J(\vec{w}_0 + \vec{\delta}) \approx J(\vec{w}_0) + \nabla J(\vec{w}_0) \cdot \vec{\delta}
\]

- Dot product term on right-hand side is negative if angle between \( \nabla J(\vec{w}_0) \) and \( \vec{\delta} \) is obtuse
- Most obtuse angle is \( 180^\circ \), achieved by \( \vec{\delta} := -\eta \nabla J(\vec{w}_0) \) for \( \eta > 0 \), in which case

\[
J(\vec{w}_0) + \nabla J(\vec{w}_0) \cdot \vec{\delta} = J(\vec{w}_0) - \| \nabla J(\vec{w}_0) \|_2 \| \vec{\delta} \|_2
\]

\[
= J(\vec{w}_0) - \eta \| \nabla J(\vec{w}_0) \|_2^2
\]
Local improvement via gradients

Back to optimization: \( \min_{\vec{w} \in \mathbb{R}^d} J(\vec{w}) \)

Suppose we have candidate setting of variables \( \vec{w} = \vec{w}_0 \in \mathbb{R}^d \), achieving objective value \( J(\vec{w}_0) \)

**Question:** How can we change \( \vec{w}_0 \) to achieve a lower objective value?

- Consider adding \( \vec{\delta} \in \mathbb{R}^d \) to \( \vec{w}_0 \)
- If \( \vec{\delta} \) is “small enough”, then
  \[
  J(\vec{w}_0 + \vec{\delta}) \approx J(\vec{w}_0) + \nabla J(\vec{w}_0) \cdot \vec{\delta}
  \]
  - Dot product term on right-hand side is negative if angle between \( \nabla J(\vec{w}_0) \) and \( \vec{\delta} \) is obtuse
  - Most obtuse angle is \( 180^\circ \), achieved by \( \vec{\delta} := -\eta \nabla J(\vec{w}_0) \) for \( \eta > 0 \), in which case
    \[
    J(\vec{w}_0 + \vec{\delta}) = J(\vec{w}_0) - \| \nabla J(\vec{w}_0) \|_2 \| \vec{\delta} \|_2
    = J(\vec{w}_0) - \eta \| \nabla J(\vec{w}_0) \|_2^2
    \]

**Upshot:** Modify \( \vec{w}_0 \) by subtracting \( \eta \nabla J(\vec{w}_0) \) in hopes of getting lower objective value

Caveat: Approximation is OK only if \( \vec{\delta} \) is “small enough” (i.e., \( \eta \) is “small enough”)
Local improvement via gradients

Back to optimization: \( \min_{\vec{w} \in \mathbb{R}^d} J(\vec{w}) \)

Suppose we have candidate setting of variables \( \vec{w} = \vec{w}_0 \in \mathbb{R}^d \), achieving objective value \( J(\vec{w}_0) \)

**Question:** How can we change \( \vec{w}_0 \) to achieve a lower objective value?

- Consider adding \( \vec{\delta} \in \mathbb{R}^d \) to \( \vec{w}_0 \)
- If \( \vec{\delta} \) is “small enough”, then
  \[
  J(\vec{w}_0 + \vec{\delta}) \approx J(\vec{w}_0) + \nabla J(\vec{w}_0) \cdot \vec{\delta}
  \]
- Dot product term on right-hand side is negative if angle between \( \nabla J(\vec{w}_0) \) and \( \vec{\delta} \) is obtuse
- Most obtuse angle is \( 180^\circ \), achieved by \( \vec{\delta} := -\eta \nabla J(\vec{w}_0) \) for \( \eta > 0 \), in which case
  \[
  J(\vec{w}_0) + \nabla J(\vec{w}_0) \cdot \vec{\delta} = J(\vec{w}_0) - \| \nabla J(\vec{w}_0) \|_2 \| \vec{\delta} \|_2 = J(\vec{w}_0) - \eta \| \nabla J(\vec{w}_0) \|_2^2
  \]

**Upshot:** Modify \( \vec{w}_0 \) by subtracting \( \eta \nabla J(\vec{w}_0) \) in hopes of getting lower objective value

**Caveat:** Approximation is OK only if \( \vec{\delta} \) is “small enough” (i.e., \( \eta \) is “small enough”)

Gradient descent

**Gradient descent**: Iterative algorithm that attempts to minimize a differentiable objective $J: \mathbb{R}^d \rightarrow \mathbb{R}$
Gradient descent

**Gradient descent**: Iterative algorithm that attempts to minimize a differentiable objective $J : \mathbb{R}^d \rightarrow \mathbb{R}$

**Gradient descent:**
For iteration $t = 1, 2, \ldots$ until stopping condition is satisfied:

$$\vec{w}^{(t)} := \vec{w}^{(t-1)} - \eta \nabla J(\vec{w}^{(t-1)})$$

(update rule)

Return final $\vec{w}^{(t)}$

---

Augustin-Louis Cauchy, 1847
Gradient descent

**Gradient descent**: Iterative algorithm that attempts to minimize a differentiable objective $J: \mathbb{R}^d \to \mathbb{R}$

For iteration $t = 1, 2, \ldots$ until stopping condition is satisfied:

$$w^{(t)} := w^{(t-1)} - \eta \nabla J(w^{(t-1)})$$

(see update rule)

Return final $w^{(t)}$

To compute each update: Need a subroutine for computing $\nabla J$, and a bit of vector arithmetic

Augustin-Louis Cauchy, 1847
Gradient descent

**Gradient descent:** Iterative algorithm that attempts to minimize a differentiable objective $J : \mathbb{R}^d \to \mathbb{R}$

For iteration $t = 1, 2, \ldots$ until stopping condition is satisfied:

$$\vec{w}(t) := \vec{w}(t-1) - \eta \nabla J(\vec{w}(t-1))$$

(update rule)

Return final $\vec{w}(t)$

To compute each update: Need a subroutine for computing $\nabla J$, and a bit of vector arithmetic

Not fully-specified as written above!

Augustin-Louis Cauchy, 1847
Gradient descent

Gradient descent: Iterative algorithm that attempts to minimize a differentiable objective $J : \mathbb{R}^d \rightarrow \mathbb{R}$

\[
\begin{align*}
\text{Gradient descent:} \\
\text{For iteration } t = 1, 2, \ldots \text{ until stopping condition is satisfied:} \\
\overrightarrow{w}(t) := \overrightarrow{w}(t-1) - \eta \nabla J(\overrightarrow{w}(t-1)) \quad \text{(update rule)}
\end{align*}
\]

Return final $\overrightarrow{w}(t)$

To compute each update: Need a subroutine for computing $\nabla J$, and a bit of vector arithmetic

Not fully-specified as written above!

What’s missing:

1. Initialization $\overrightarrow{w}(0) \in \mathbb{R}^d$
2. Choice of “step size” $\eta > 0$ (a.k.a. “learning rate”)
3. Stopping condition

Augustin-Louis Cauchy, 1847
Example: Sum of squared errors objective from OLS

Sum of squared errors (SSE) objective from ordinary least squares

$$\text{sse}(\vec{w}) = \sum_{i=1}^{n} (\vec{x}_i \cdot \vec{w} - y_i)^2$$

where training data are $$(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n) \in \mathbb{R}^d \times \mathbb{R}$$
Example: Sum of squared errors objective from OLS

Sum of squared errors (SSE) objective from ordinary least squares

$$\text{sse}(\vec{w}) = \sum_{i=1}^{n} (\vec{x}_i \cdot \vec{w} - y_i)^2$$

where training data are \((\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n) \in \mathbb{R}^d \times \mathbb{R}\)

Question: How to implement \(\nabla \text{sse}\) computation?
Example: Sum of squared errors objective from OLS

Sum of squared errors (SSE) objective from ordinary least squares

\[
\text{sse}(\vec{w}) = \sum_{i=1}^{n} (\vec{x}_i \cdot \vec{w} - y_i)^2
\]

where training data are \((\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n) \in \mathbb{R}^d \times \mathbb{R}\)

**Question:** How to implement \(\nabla \text{sse}\) computation?

Linearity and chain rule of differentiation gives a formula:

\[
\nabla \text{sse}(\vec{w}) = \sum_{i=1}^{n} 2 (\vec{x}_i \cdot \vec{w} - y_i) \vec{x}_i
\]
Example: Sum of squared errors objective from OLS

**Sum of squared errors (SSE) objective from ordinary least squares**

\[
\text{sse}(\vec{w}) = \sum_{i=1}^{n} (\vec{x}_i \cdot \vec{w} - y_i)^2
\]

where training data are \((\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n) \in \mathbb{R}^d \times \mathbb{R}\)

**Question:** How to implement \(\nabla \text{sse}\) computation?

Linearity and chain rule of differentiation gives a formula:

\[
\nabla \text{sse}(\vec{w}) = \sum_{i=1}^{n} 2 (\vec{x}_i \cdot \vec{w} - y_i) \vec{x}_i
\]

For iteration \(t = 1, 2, \ldots\) until stopping condition is satisfied:

\[
\vec{w}^{(t)} := \vec{w}^{(t-1)} + 2\eta \sum_{i=1}^{n} (y_i - \vec{x}_i \cdot \vec{w}^{(t-1)}) \vec{x}_i
\]

Return final \(\vec{w}^{(t)}\)
Example: Sum of logarithmic losses

Sum of logarithmic losses (SLL) (i.e., negative log-likelihood) from logistic regression

$$\text{sll}(\vec{w}) = \sum_{i=1}^{n} \left[ \ln(1 + e^{\vec{x}_i \cdot \vec{w}}) - y_i \vec{x}_i \cdot \vec{w} \right]$$

where training data are $$(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n) \in \mathbb{R}^d \times \{0, 1\}$$
Example: Sum of logarithmic losses

Sum of logarithmic losses (SLL) (i.e., negative log-likelihood) from logistic regression

\[
sll(\vec{w}) = \sum_{i=1}^{n} \left[ \ln(1 + e^{\vec{x}_i \cdot \vec{w}}) - y_i \vec{x}_i \cdot \vec{w} \right]
\]

where training data are \((\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n) \in \mathbb{R}^d \times \{0, 1\}\)

Gradient:

\[
\nabla sll(\vec{w}) = \sum_{i=1}^{n} \left[ \frac{e^{\vec{x}_i \cdot \vec{w}}}{1 + e^{\vec{x}_i \cdot \vec{w}}} \vec{x}_i - y_i \vec{x}_i \right]
\]

\[
= \sum_{i=1}^{n} \left( \text{logistic}(\vec{x}_i \cdot \vec{w}) - y_i \right) \vec{x}_i
\]
Example: Sum of logarithmic losses

Sum of logarithmic losses (SLL) (i.e., negative log-likelihood) from logistic regression

$$\text{sll}(\vec{w}) = \sum_{i=1}^{n} \left[ \ln(1 + e^{\vec{x}_i \cdot \vec{w}}) - y_i \vec{x}_i \cdot \vec{w} \right]$$

where training data are $$(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n) \in \mathbb{R}^d \times \{0, 1\}$$

Gradient:

$$\nabla \text{sll}(\vec{w}) = \sum_{i=1}^{n} \left[ \frac{e^{\vec{x}_i \cdot \vec{w}}}{1 + e^{\vec{x}_i \cdot \vec{w}}} \vec{x}_i - y_i \vec{x}_i \right]$$

$$= \sum_{i=1}^{n} (\text{logistic}(\vec{x}_i \cdot \vec{w}) - y_i) \vec{x}_i$$

For iteration $t = 1, 2, \ldots$ until stopping condition is satisfied:

$$\vec{w}^{(t)} := \vec{w}^{(t-1)} + \eta \sum_{i=1}^{n} (y_i - \text{logistic}(\vec{x}_i \cdot \vec{w}^{(t-1)})) \vec{x}_i$$

Return final $\vec{w}^{(t)}$
“Interpretations” of update rules for SSE and SLL:

Both update rules move weight vector in direction of a linear combination of feature vectors $\vec{x}_i$

\[
\vec{w}^{(t)} := \vec{w}^{(t-1)} + 2\eta \sum_{i=1}^{n} (y_i - \vec{x}_i \cdot \vec{w}^{(t-1)}) \vec{x}_i
\]

(update for SSE)

\[
\vec{w}^{(t)} := \vec{w}^{(t-1)} + \eta \sum_{i=1}^{n} (y_i - \text{logistic}(\vec{x}_i \cdot \vec{w}^{(t-1)})) \vec{x}_i
\]

(update for SLL)
“Interpretations” of update rules for SSE and SLL:

Both update rules move weight vector in direction of a linear combination of feature vectors $\vec{x}_i$

\[
\tilde{w}^{(t)} := \tilde{w}^{(t-1)} + 2\eta \sum_{i=1}^{n} (y_i - \vec{x}_i \cdot \tilde{w}^{(t-1)}) \vec{x}_i \\
\text{"residual"}
\]  

(update for SSE)

\[
\tilde{w}^{(t)} := \tilde{w}^{(t-1)} + \eta \sum_{i=1}^{n} (y_i - \text{logistic}(\vec{x}_i \cdot \tilde{w}^{(t-1)})) \vec{x}_i \\
\text{"residual"}
\]  

(update for SLL)

- Size of coefficient on $\vec{x}_i$ in “update” proportional to size of $i$-th “residual”
Example: Update rules

“Interpretations” of update rules for SSE and SLL:
Both update rules move weight vector in direction of a linear combination of feature vectors \( \vec{x}_i \)

\[
\overrightarrow{w}(t) := \overrightarrow{w}(t-1) + 2\eta \sum_{i=1}^{n} \left( y_i - \vec{x}_i \cdot \overrightarrow{w}(t-1) \right) \vec{x}_i \\
\text{"residual"}
\]

(update for SSE)

\[
\overleftarrow{w}(t) := \overleftarrow{w}(t-1) + \eta \sum_{i=1}^{n} \left( y_i - \logistic(\vec{x}_i \cdot \overleftarrow{w}(t-1)) \right) \vec{x}_i \\
\text{"residual"}
\]

(update for SLL)

- Size of coefficient on \( \vec{x}_i \) in “update” proportional to size of \( i \)-th “residual”
- Sign of coefficient on \( \vec{x}_i \) tries to make \( \overrightarrow{w}(t) \) have “better” prediction than \( \overrightarrow{w}(t-1) \) did
  (But contributions of other \( \vec{x}_j \)'s may interfere)
Example: Sum of logarithmic losses

Gradient descent update rule:

\[ \tilde{w}(t) := \tilde{w}(t-1) + \eta \sum_{i=1}^{n} (y_i - \text{logistic}(\vec{x}_i \cdot \tilde{w}(t-1))) \vec{x}_i \]
Example: Sum of logarithmic losses

Gradient descent update rule:

\[ \bar{w}^{(t)} := \bar{w}^{(t-1)} + \eta \sum_{i=1}^{n} (y_i - \text{logistic}(\vec{x}_i \cdot \bar{w}^{(t-1)})) \vec{x}_i \]

Thin red arrows originating at each \( \vec{x}_i \):

\[ \eta (y_i - \text{logistic}(\vec{x}_i \cdot \bar{w}^{(t-1)})) \vec{x}_i \]

Thick red arrow at origin:

\[ \eta \sum_{i=1}^{n} (y_i - \text{logistic}(\vec{x}_i \cdot \bar{w}^{(t-1)})) \vec{x}_i \]

Iteration \( t = 1 \)

\[ \| \bar{w}^{(1)} \|_2 = 1.31187 \]
Example: Sum of logarithmic losses

Gradient descent update rule:

\[ \bar{w}^{(t)} := \bar{w}^{(t-1)} + \eta \sum_{i=1}^{n} (y_i - \text{logistic}(\bar{x}_i \cdot \bar{w}^{(t-1)})) \bar{x}_i \]

Thin red arrows originating at each \( \bar{x}_i \):

\[ \eta (y_i - \text{logistic}(\bar{x}_i \cdot \bar{w}^{(t-1)})) \bar{x}_i \]

Thick red arrow at origin:

\[ \eta \sum_{i=1}^{n} (y_i - \text{logistic}(\bar{x}_i \cdot \bar{w}^{(t-1)})) \bar{x}_i \]
Example: Sum of logarithmic losses

\[ \bar{w}^{(t)} := \bar{w}^{(t-1)} + \eta \sum_{i=1}^{n} (y_i - \text{logistic}(\bar{x}_i \cdot \bar{w}^{(t-1)})) \bar{x}_i \]

- Gradient descent update rule:
- Thin red arrows originating at each \( \bar{x}_i \):
- Thick red arrow at origin:

\[ \eta \sum_{i=1}^{n} (y_i - \text{logistic}(\bar{x}_i \cdot \bar{w}^{(t-1)})) \bar{x}_i \]
Example: Sum of logarithmic losses

\[ \sum \text{logarithmic losses} \]

\[ -1 \quad -0.5 \quad 0 \quad 0.5 \quad 1 \]

\[ -1 \quad -0.8 \quad -0.6 \quad -0.4 \quad -0.2 \quad 0 \quad 0.2 \quad 0.4 \quad 0.6 \quad 0.8 \quad 1 \]

\begin{itemize}
  \item \text{Class 0}
  \item \text{Class 1}
  \item \text{previous boundary}
  \item \text{new boundary}
\end{itemize}

\begin{itemize}
  \item \text{Gradient descent update rule:}
  \[ \overrightarrow{w}^{(t)} := \overrightarrow{w}^{(t-1)} + \eta \sum_{i=1}^{n} \left( y_i - \text{logistic}(\overrightarrow{x}_i \cdot \overrightarrow{w}^{(t-1)}) \right) \overrightarrow{x}_i \]

  \item \text{Thin red arrows originating at each } \overrightarrow{x}_i:
  \[ \eta \left( y_i - \text{logistic}(\overrightarrow{x}_i \cdot \overrightarrow{w}^{(t-1)}) \right) \overrightarrow{x}_i \]

  \item \text{Thick red arrow at origin:}
  \[ \eta \sum_{i=1}^{n} \left( y_i - \text{logistic}(\overrightarrow{x}_i \cdot \overrightarrow{w}^{(t-1)}) \right) \overrightarrow{x}_i \]
\end{itemize}

\text{Iteration } t = 4

\[ \| \overrightarrow{w}^{(4)} \|_2 = 2.16793 \]
Example: Sum of logarithmic losses

Gradient descent update rule:

$$\tilde{w}^{(t)} := \tilde{w}^{(t-1)} + \eta \sum_{i=1}^{n} (y_i - \text{logistic}(\tilde{x}_i \cdot \tilde{w}^{(t-1)})) \tilde{x}_i$$

Thin red arrows originating at each $\tilde{x}_i$:

$$\eta (y_i - \text{logistic}(\tilde{x}_i \cdot \tilde{w}^{(t-1)})) \tilde{x}_i$$

Thick red arrow at origin:

$$\eta \sum_{i=1}^{n} (y_i - \text{logistic}(\tilde{x}_i \cdot \tilde{w}^{(t-1)})) \tilde{x}_i$$

Iteration $t = 5$

$$\|\tilde{w}^{(5)}\|_2 = 2.39059$$
Gradient descent update rule:

\[ \tilde{w}^{(t)} := \tilde{w}^{(t-1)} + \eta \sum_{i=1}^{n} (y_i - \text{logistic}(\bar{x}_i \cdot \tilde{w}^{(t-1)})) \bar{x}_i \]

Thin red arrows originating at each \( \bar{x}_i \):

\[ \eta (y_i - \text{logistic}(\bar{x}_i \cdot \tilde{w}^{(t-1)})) \bar{x}_i \]

Thick red arrow at origin:

\[ \eta \sum_{i=1}^{n} (y_i - \text{logistic}(\bar{x}_i \cdot \tilde{w}^{(t-1)})) \bar{x}_i \]
Example: Sum of logarithmic losses

\[ \bar{w}(t) := \bar{w}(t-1) + \eta \sum_{i=1}^{n} (y_i - \text{logistic}(x_i \cdot \bar{w}(t-1))) \bar{x}_i \]

- Gradient descent update rule:

- Thin red arrows originating at each \( \bar{x}_i \):

\[ \eta (y_i - \text{logistic}(x_i \cdot \bar{w}(t-1))) \bar{x}_i \]

- Thick red arrow at origin:

\[ \eta \sum_{i=1}^{n} (y_i - \text{logistic}(x_i \cdot \bar{w}(t-1))) \bar{x}_i \]
Example: Sum of logarithmic losses

Gradient descent update rule:

\[ \vec{w}^{(t)} := \vec{w}^{(t-1)} + \eta \sum_{i=1}^{n} (y_i - \text{logistic}(\vec{x}_i \cdot \vec{w}^{(t-1)})) \vec{x}_i \]

Thin red arrows originating at each \( \vec{x}_i \):

\[ \eta (y_i - \text{logistic}(\vec{x}_i \cdot \vec{w}^{(t-1)})) \vec{x}_i \]

Thick red arrow at origin:

\[ \eta \sum_{i=1}^{n} (y_i - \text{logistic}(\vec{x}_i \cdot \vec{w}^{(t-1)})) \vec{x}_i \]

Iteration \( t = 8 \)

\[ \| \vec{w}^{(8)} \|_2 = 2.90959 \]
Properties of Gradient Descent
Local improvements from gradient descent

**Theorem.** If $J$ is "smooth enough", then there's a choice for $\eta > 0$ so that for any $\tilde{w}_0 \in \mathbb{R}^d$,

$$J(\tilde{w}_0 - \eta \nabla J(\tilde{w}_0)) \leq J(\tilde{w}_0) - \frac{\eta}{2} \left\| \nabla J(\tilde{w}_0) \right\|_2^2$$
Local improvements from gradient descent

Theorem. If $J$ is “smooth enough”, then there’s a choice for $\eta > 0$ so that for any $\vec{w}_0 \in \mathbb{R}^d$,

$$J(\vec{w}_0 - \eta \nabla J(\vec{w}_0)) \leq J(\vec{w}_0) - \frac{\eta}{2} \| \nabla J(\vec{w}_0) \|^2_2$$

Gradient descent update improves objective value if gradient $\neq \vec{0}$
Local improvements from gradient descent

**Theorem.** If $J$ is “smooth enough”, then there’s a choice for $\eta > 0$ so that for any $\vec{w}_0 \in \mathbb{R}^d$,

$$J(\vec{w}_0 - \eta \nabla J(\vec{w}_0)) \leq J(\vec{w}_0) - \frac{\eta}{2} \| \nabla J(\vec{w}_0) \|^2_2$$

Gradient descent update improves objective value if gradient $\neq \vec{0}$

**Caveat:** $\nabla J(\vec{w}_0) = \vec{0}$ doesn’t necessarily mean $\vec{w}_0$ is minimizer of $J$

- Just says that gradient itself doesn’t suggest a useful direction for improvement
Local improvements from gradient descent

**Theorem.** If $J$ is “smooth enough”, then there’s a choice for $\eta > 0$ so that for any $\vec{w}_0 \in \mathbb{R}^d$,

$$J(\vec{w}_0 - \eta \nabla J(\vec{w}_0)) \leq J(\vec{w}_0) - \frac{\eta}{2} \|\nabla J(\vec{w}_0)\|_2^2$$

Gradient descent update improves objective value if gradient $\neq \vec{0}$

**Caveat:** $\nabla J(\vec{w}_0) = \vec{0}$ doesn’t necessarily mean $\vec{w}_0$ is minimizer of $J$

- Just says that gradient itself doesn’t suggest a useful direction for improvement

**Fortunately:** Both $sse(\vec{w})$ and $sll(\vec{w})$ are **convex functions** of $\vec{w}$
Theorem. If $J$ is “smooth enough”, then there’s a choice for $\eta > 0$ so that for any $\vec{w}_0 \in \mathbb{R}^d$,

$$J(\vec{w}_0 - \eta \nabla J(\vec{w}_0)) \leq J(\vec{w}_0) - \frac{\eta}{2} \|
abla J(\vec{w}_0)\|^2_2$$

Gradient descent update improves objective value if gradient $\neq \vec{0}$

Caveat: $\nabla J(\vec{w}_0) = \vec{0}$ doesn’t necessarily mean $\vec{w}_0$ is minimizer of $J$

- Just says that gradient itself doesn’t suggest a useful direction for improvement

Fortunately: Both $\text{sse}(\vec{w})$ and $\text{slf}(\vec{w})$ are convex functions of $\vec{w}$

Theorem. If $J$ is convex and “smooth enough”, then there’s a choice for $\eta > 0$ so that for any initial $\vec{w}^{(0)} \in \mathbb{R}^d$, iterates of gradient descent $\vec{w}^{(1)}, \vec{w}^{(2)}, \ldots$ satisfy

$$\lim_{t \to \infty} J(\vec{w}^{(t)}) = \min_{\vec{w} \in \mathbb{R}^d} J(\vec{w})$$
Definitions of convexity (for differentiable functions)

A function $J: \mathbb{R}^d \to \mathbb{R}$ is convex if, for all $\vec{w}_0, \vec{w}_1 \in \mathbb{R}^d$, and all $\alpha \in [0, 1]$,

$$J \left( (1 - \alpha)\vec{w}_0 + \alpha \vec{w}_1 \right) \leq (1 - \alpha)J(\vec{w}_0) + \alpha J(\vec{w}_1).$$
Definitions of convexity (for differentiable functions)

A function $J: \mathbb{R}^d \rightarrow \mathbb{R}$ is convex if, for all $\vec{w}_0, \vec{w}_1 \in \mathbb{R}^d$, and all $\alpha \in [0, 1]$,

$$J((1 - \alpha)\vec{w}_0 + \alpha\vec{w}_1) \leq (1 - \alpha)J(\vec{w}_0) + \alpha J(\vec{w}_1).$$

A differentiable function $J: \mathbb{R}^d \rightarrow \mathbb{R}$ is convex if, for all $\vec{w}_0, \vec{w} \in \mathbb{R}^d$,

$$J(\vec{w}) \geq J(\vec{w}_0) + \nabla J(\vec{w}_0) \cdot (\vec{w} - \vec{w}_0).$$
Why convexity helps?

Convexity of $J$: for all $\vec{w}_0, \vec{w} \in \mathbb{R}^d$,

$$J(\vec{w}) \geq J(\vec{w}_0) + \nabla J(\vec{w}_0) \cdot (\vec{w} - \vec{w}_0).$$
Why convexity helps?

Convexity of $J$: for all $\vec{w}_0, \vec{w} \in \mathbb{R}^d$,

$$J(\vec{w}) \geq J(\vec{w}_0) + \nabla J(\vec{w}_0) \cdot (\vec{w} - \vec{w}_0).$$

This implies the following property of $-\nabla J(\vec{w}_0)$:

$$(-\nabla J(\vec{w}_0)) \cdot (\vec{w} - \vec{w}_0) \geq J(\vec{w}_0) - J(\vec{w}).$$

(†)
Why convexity helps?

Convexity of $J$: for all $\vec{w}_0, \vec{w} \in \mathbb{R}^d$,

$$J(\vec{w}) \geq J(\vec{w}_0) + \nabla J(\vec{w}_0) \cdot (\vec{w} - \vec{w}_0).$$

This implies the following property of $-\nabla J(\vec{w}_0)$:

$$(-\nabla J(\vec{w}_0)) \cdot (\vec{w} - \vec{w}_0) \geq J(\vec{w}_0) - J(\vec{w}) \quad (†)$$

▶ Suppose $\vec{w}^*$ is a minimizer of $J$, and you currently have $\vec{w}_0$ in hand
Why convexity helps?

Convexity of $J$: for all $\vec{w}_0, \vec{w} \in \mathbb{R}^d$,

$$J(\vec{w}) \geq J(\vec{w}_0) + \nabla J(\vec{w}_0) \cdot (\vec{w} - \vec{w}_0).$$

This implies the following property of $-\nabla J(\vec{w}_0)$:

$$(-\nabla J(\vec{w}_0)) \cdot (\vec{w} - \vec{w}_0) \geq J(\vec{w}_0) - J(\vec{w}) \quad (\dagger)$$

- Suppose $\vec{w}^*$ is a minimizer of $J$, and you currently have $\vec{w}_0$ in hand

- If right-hand side of ($\dagger$) is positive, then $-\nabla J(\vec{w}_0)$ is “aligned” with direction $\vec{w}^* - \vec{w}_0$ pointing to minimizer $\vec{w}^*$
  - ...but not perfectly aligned! I.e., can’t jump to $\vec{w}^*$ in a single step)
Why convexity helps?

Convexity of $J$: for all $\vec{w}_0, \vec{w} \in \mathbb{R}^d$,

$$J(\vec{w}) \geq J(\vec{w}_0) + \nabla J(\vec{w}_0) \cdot (\vec{w} - \vec{w}_0).$$

This implies the following property of $-\nabla J(\vec{w}_0)$:

$$( -\nabla J(\vec{w}_0)) \cdot (\vec{w} - \vec{w}_0) \geq J(\vec{w}_0) - J(\vec{w}) \quad (∗)$$

- Suppose $\vec{w}^*$ is a minimizer of $J$, and you currently have $\vec{w}_0$ in hand.
- If right-hand side of (∗) is positive, then $-\nabla J(\vec{w}_0)$ is “aligned” with direction $\vec{w}^* - \vec{w}_0$ pointing to minimizer $\vec{w}^*$
  - (… but not perfectly aligned! I.e., can’t jump to $\vec{w}^*$ in a single step)
- ∴ As long as $J(\vec{w}_0) > J(\vec{w}^*)$, can make progress by gradient descent update.
Example: Logistic regression negative log-likelihood

Logistic regression negative log-likelihood (i.e., SLL) on text classification data
Practical considerations

While theory sometimes has prescriptions for initialization, step sizes, and stopping conditions, they are often restrictive and/or pessimistic 😊
Practical considerations

While theory sometimes has prescriptions for initialization, step sizes, and stopping conditions, they are often restrictive and/or pessimistic 😞

- **Initialization**: If $J$ is convex, convergence is possible starting from any $\mathbf{w}^{(0)}$
  - But still better to start closer to where you want to end up
  - For non-convex objectives, may need to try many different $\mathbf{w}^{(0)}$

- **Step sizes** $\eta$:
  - Very important; may require substantial experimentation
  - $\eta$ too large: Iterates $(\mathbf{w}(t))$ may diverge
  - $\eta$ too small: Iterates $(\mathbf{w}(t))$ may be slow to converge

- **Stopping condition**: Plot objective value and visually check for convergence
  - Can also use cross validation

Bottom line: Some experimentation is often needed to effectively apply gradient descent
Practical considerations

While theory sometimes has prescriptions for initialization, step sizes, and stopping conditions, they are often restrictive and/or pessimistic 😊

- **Initialization**: If $J$ is convex, convergence is possible starting from any $\vec{w}^{(0)}$
  - But still better to start closer to where you want to end up
  - For non-convex objectives, may need to try many different $\vec{w}^{(0)}$

- **Step sizes $\eta$**: Very important; may require substantial experimentation
  - $\eta$ too large: Iterates $(\vec{w}^{(t)})$ may diverge
  - $\eta$ too small: Iterates $(\vec{w}^{(t)})$ may be slow to converge

Many heuristics for adaptively choosing $\eta$ in literature (e.g., initially large, then decrease with $t$)

- **Stopping condition**:
  - Plot objective value and visually check for convergence
  - Can also use cross validation

Minimizing training objective is usually not the primary objective in ML

Bottom line: Some experimentation is often needed to effectively apply gradient descent
Practical considerations

While theory sometimes has prescriptions for initialization, step sizes, and stopping conditions, they are often restrictive and/or pessimistic 😊

▶ **Initialization**: If $J$ is convex, convergence is possible starting from any $\vec{w}^{(0)}$
  
  ▶ But still better to start closer to where you want to end up
  
  ▶ For non-convex objectives, may need to try many different $\vec{w}^{(0)}$

▶ **Step sizes $\eta$**: Very important; may require substantial experimentation
  
  ▶ $\eta$ too large: Iterates $(\vec{w}^{(t)})$ may diverge
  
  ▶ $\eta$ too small: Iterates $(\vec{w}^{(t)})$ may be slow to converge

Many heuristics for adaptively choosing $\eta$ in literature (e.g., initially large, then decrease with $t$)
Practical considerations

While theory sometimes has prescriptions for initialization, step sizes, and stopping conditions, they are often restrictive and/or pessimistic.

- **Initialization**: If $J$ is convex, convergence is possible starting from any $\vec{w}^{(0)}$.
  - But still better to start closer to where you want to end up.
  - For non-convex objectives, may need to try many different $\vec{w}^{(0)}$.

- **Step sizes $\eta$**: Very important; may require substantial experimentation.
  - $\eta$ too large: Iterates $(\vec{w}^{(t)})$ may diverge.
  - $\eta$ too small: Iterates $(\vec{w}^{(t)})$ may be slow to converge.

Many heuristics for adaptively choosing $\eta$ in literature (e.g., initially large, then decrease with $t$).

- **Stopping condition**: Plot objective value and visually check for convergence.
  - Can also use cross validation.
  - Minimizing training objective is usually not the primary objective in ML.
Practical considerations

While theory sometimes has prescriptions for initialization, step sizes, and stopping conditions, they are often restrictive and/or pessimistic 😊

- **Initialization:** If $J$ is convex, convergence is possible starting from any $\vec{w}^{(0)}$
  - But still better to start closer to where you want to end up
  - For non-convex objectives, may need to try many different $\vec{w}^{(0)}$

- **Step sizes** $\eta$: Very important; may require substantial experimentation
  - $\eta$ too large: Iterates ($\vec{w}^{(t)}$) may diverge
  - $\eta$ too small: Iterates ($\vec{w}^{(t)}$) may be slow to converge

  Many heuristics for adaptively choosing $\eta$ in literature (e.g., initially large, then decrease with $t$)

- **Stopping condition:** Plot objective value and visually check for convergence
  - Can also use cross validation
  - Minimizing training objective is usually not the primary objective in ML

**Bottom line:** Some experimentation is often needed to effectively apply gradient descent
Stochastic Gradient Descent
Finite sum objectives

In many ML applications . . .

Objective has finite sum form

$$J(\vec{w}) = \sum_{i=1}^{n} \text{Term}_i(\vec{w})$$

E.g., one “loss” per training example
Finite sum objectives

In many ML applications . . .

Objective has **finite sum** form

\[ J(\vec{w}) = \sum_{i=1}^{n} \text{Term}_i(\vec{w}) \]

E.g., one “loss” per training example

\[ \nabla J(\vec{w}) = \sum_{i=1}^{n} \nabla \text{Term}_i(\vec{w}) \]

∴ Gradient \( \nabla J \) also has finite sum form
Finite sum objectives

In many ML applications . . .

Objective has \textbf{finite sum} form

\[ J(\vec{w}) = \sum_{i=1}^{n} \text{Term}_i(\vec{w}) \]

E.g., one “loss” per training example

\[ \nabla J(\vec{w}) = \sum_{i=1}^{n} \nabla \text{Term}_i(\vec{w}) \]

\[ \therefore \] Gradient \( \nabla J \) also has finite sum form

**Computational time per iteration:** Proportional to number of terms \( n \)
Finite sum objectives

In many ML applications . . .

Objective has **finite sum** form

\[ J(\vec{w}) = \sum_{i=1}^{n} \text{Term}_i(\vec{w}) \]

\[ \nabla J(\vec{w}) = \sum_{i=1}^{n} \nabla \text{Term}_i(\vec{w}) \]

E.g., one “loss” per training example

**Computational time per iteration:** Proportional to number of terms \( n \)

**Idea:** To speed up, estimate sum using (scaling of) just one randomly chosen term

“So stochastic approximation”
(Robbins & Monro, 1951)
Gradient estimation for finite sum objectives

**Stochastic gradient** at \( \tilde{w} \):

Pick a random number \( I \) uniformly at random from \( \{1, \ldots , n\} \), and compute \( n \nabla \text{Term}_I(\tilde{w}) \)
Gradient estimation for finite sum objectives

**Stochastic gradient** at \( \bar{w} \):
Pick a random number \( I \) uniformly at random from \( \{1, \ldots, n\} \), and compute \( n \nabla \text{Term}_I(\bar{w}) \)

\[
\mathbb{E} [n \nabla \text{Term}_I(\bar{w})]
\]
Gradient estimation for finite sum objectives

**Stochastic gradient** at $\vec{w}$:

Pick a random number $I$ uniformly at random from $\{1, \ldots, n\}$, and compute $n \nabla \text{Term}_I(\vec{w})$

$$
\mathbb{E} \left[ n \nabla \text{Term}_I(\vec{w}) \right] = \sum_{i=1}^{n} \Pr(I = i) \times n \nabla \text{Term}_i(\vec{w})
$$
Gradient estimation for finite sum objectives

**Stochastic gradient** at $\vec{w}$:
Pick a random number $I$ uniformly at random from $\{1, \ldots, n\}$, and compute $n \nabla \text{Term}_I(\vec{w})$

$$
\mathbb{E} \left[ n \nabla \text{Term}_I(\vec{w}) \right] = \sum_{i=1}^{n} \Pr(I = i) \times n \nabla \text{Term}_i(\vec{w}) = \sum_{i=1}^{n} \nabla \text{Term}_i(\vec{w})
$$

Stochastic gradient "descent" (SGD):
For iteration $t = 1, 2, \ldots$ until stopping condition is satisfied:
▶ Pick $I_t$ uniformly at random from $\{1, \ldots, n\}$, and compute

$$
\vec{w}(t) = \vec{w}(t-1) - \eta n \nabla \text{Term}_{I_t}(\vec{w}(t-1))
$$

Return final $\vec{w}(t)$

Minibatch SGD:
To reduce "variance" of gradient estimates, average multiple independent estimates together in each iteration
Gradient estimation for finite sum objectives

Stochastic gradient at $\vec{w}$:
Pick a random number $I$ uniformly at random from $\{1, \ldots, n\}$, and compute $n \nabla \text{Term}_I(\vec{w})$

$$\mathbb{E} \left[ n \nabla \text{Term}_I(\vec{w}) \right] = \sum_{i=1}^{n} \Pr(I = i) \times n \nabla \text{Term}_i(\vec{w}) = \sum_{i=1}^{n} \nabla \text{Term}_i(\vec{w}) = \nabla J(\vec{w})$$
Gradient estimation for finite sum objectives

**Stochastic gradient** at $\vec{w}$:
Pick a random number $I$ uniformly at random from $\{1, \ldots, n\}$, and compute $n \nabla \text{Term}_I(\vec{w})$

$$
\mathbb{E}[n \nabla \text{Term}_I(\vec{w})] = \sum_{i=1}^{n} \Pr(I = i) \times n \nabla \text{Term}_i(\vec{w}) = \sum_{i=1}^{n} \nabla \text{Term}_i(\vec{w}) = \nabla J(\vec{w})
$$

So $n \nabla \text{Term}_I(\vec{w})$ is an unbiased estimate of $\nabla J(\vec{w})$, and $n$ times faster to compute than $\nabla J(\vec{w})$!
Gradient estimation for finite sum objectives

**Stochastic gradient** at $\vec{w}$:
Pick a random number $I$ uniformly at random from $\{1, \ldots, n\}$, and compute $n \nabla \text{Term}_I(\vec{w})$

$$
\mathbb{E}[n \nabla \text{Term}_I(\vec{w})] = \sum_{i=1}^{n} \Pr(I = i) \times n \nabla \text{Term}_i(\vec{w}) = \sum_{i=1}^{n} \nabla \text{Term}_i(\vec{w}) = \nabla J(\vec{w})
$$

So $n \nabla \text{Term}_I(\vec{w})$ is an unbiased estimate of $\nabla J(\vec{w})$, and $n$ times faster to compute than $\nabla J(\vec{w})$!

**Stochastic gradient “descent” (SGD):**
For iteration $t = 1, 2, \ldots$ until stopping condition is satisfied:

- Pick $I_t$ uniformly at random from $\{1, \ldots, n\}$, and compute

$$
\vec{w}(t) := \vec{w}(t-1) - \eta \ n \nabla \text{Term}_{I_t}(\vec{w}(t-1))
$$

Return final $\vec{w}(t)$
Gradient estimation for finite sum objectives

**Stochastic gradient** at $\vec{w}$:
Pick a random number $I$ uniformly at random from $\{1, \ldots, n\}$, and compute $n \nabla \text{Term}_I(\vec{w})$

$$
\mathbb{E}[n \nabla \text{Term}_I(\vec{w})] = \sum_{i=1}^{n} \Pr(I = i) \times n \nabla \text{Term}_i(\vec{w}) = \sum_{i=1}^{n} \nabla \text{Term}_i(\vec{w}) = \nabla J(\vec{w})
$$

So $n \nabla \text{Term}_I(\vec{w})$ is an unbiased estimate of $\nabla J(\vec{w})$, and $n$ times faster to compute than $\nabla J(\vec{w})$!

**Stochastic gradient “descent” (SGD):**
For iteration $t = 1, 2, \ldots$ until stopping condition is satisfied:

- Pick $I_t$ uniformly at random from $\{1, \ldots, n\}$, and compute

  $$
  \vec{w}^{(t)} := \vec{w}^{(t-1)} - \eta n \nabla \text{Term}_{I_t}(\vec{w}^{(t-1)})
  $$

Return final $\vec{w}^{(t)}$

**Minibatch SGD:** To reduce “variance” of gradient estimates, average multiple independent estimates together in each iteration
Example: SGD for sum of logarithmic losses

- SGD update rule: For uniformly random $I_t$,

$$\tilde{w}^{(t)} := \tilde{w}^{(t-1)} + \eta n (y_{I_t} - \text{logistic}(\vec{x}_{I_t} \cdot \tilde{w}^{(t-1)})) \vec{x}_{I_t}$$
Example: SGD for sum of logarithmic losses

- SGD update rule: For uniformly random $I_t$,

$$\tilde{w}(t) := \tilde{w}(t-1) + \eta n (y_{I_t} - \text{logistic}(\tilde{x}_{I_t} \cdot \tilde{w}(t-1))) \tilde{x}_{I_t}$$

- Thin red arrow originating at $\tilde{x}_{I_t}$:

$$\eta n (y_{I_t} - \text{logistic}(\tilde{x}_{I_t} \cdot \tilde{w}(t-1))) \tilde{x}_{I_t}$$

Iteration $t = 1$

$$\|\tilde{w}^{(1)}\|_2 = 1.56159$$
Example: SGD for sum of logarithmic losses

- SGD update rule: For uniformly random $I_t$,
  \[ \bar{w}^{(t)} := \bar{w}^{(t-1)} + \eta n \left( y_{I_t} - \text{logistic}(\bar{x}_{I_t} \cdot \bar{w}^{(t-1)}) \right) \bar{x}_{I_t} \]

- Thin red arrow originating at $\bar{x}_{I_t}$:
  \[ \eta n \left( y_{I_t} - \text{logistic}(\bar{x}_{I_t} \cdot \bar{w}^{(t-1)}) \right) \bar{x}_{I_t} \]

Iteration $t = 2$

\[ \|\bar{w}^{(2)}\|_2 = 1.36024 \]
Example: SGD for sum of logarithmic losses

- SGD update rule: For uniformly random $I_t$,
  \[ \bar{w}^{(t)} := \bar{w}^{(t-1)} + \eta n (y_{I_t} - \text{logistic}(\bar{x}_{I_t} \cdot \bar{w}^{(t-1)})) \bar{x}_{I_t} \]

- Thin red arrow originating at $\bar{x}_{I_t}$:
  \[ \eta n (y_{I_t} - \text{logistic}(\bar{x}_{I_t} \cdot \bar{w}^{(t-1)})) \bar{x}_{I_t} \]

Iteration $t = 3$
\[ \| \bar{w}^{(3)} \|_2 = 1.89095 \]
Example: SGD for sum of logarithmic losses

- SGD update rule: For uniformly random $I_t$,
  \[ \vec{w}^{(t)} := \vec{w}^{(t-1)} + \eta n (y_{I_t} - \text{logistic}(\vec{x}_{I_t} \cdot \vec{w}^{(t-1)})) \vec{x}_{I_t} \]

- Thin red arrow originating at $\vec{x}_{I_t}$:
  \[ \eta n (y_{I_t} - \text{logistic}(\vec{x}_{I_t} \cdot \vec{w}^{(t-1)})) \vec{x}_{I_t} \]

\[ \text{Iteration } t = 4 \]
\[ \| \vec{w}^{(4)} \|_2 = 2.57627 \]
Example: SGD for sum of logarithmic losses

- SGD update rule: For uniformly random $I_t$,
  \[
  \tilde{w}^{(t)} := \tilde{w}^{(t-1)} + \eta \cdot (y_{I_t} - \text{logistic}(\tilde{x}_{I_t} \cdot \tilde{w}^{(t-1)})) \cdot \tilde{x}_{I_t}
  \]

- Thin red arrow originating at $\tilde{x}_{I_t}$:
  \[
  \eta \cdot (y_{I_t} - \text{logistic}(\tilde{x}_{I_t} \cdot \tilde{w}^{(t-1)})) \cdot \tilde{x}_{I_t}
  \]

Iteration $t = 5$
\[
\|\tilde{w}^{(5)}\|_2 = 3.12006
\]
Example: SGD for sum of logarithmic losses

- SGD update rule: For uniformly random $I_t$,
  \[
  \vec{w}(t) := \vec{w}(t-1) + \eta n \left(y_{I_t} - \text{logistic}(\vec{x}_{I_t} \cdot \vec{w}(t-1))\right) \vec{x}_{I_t}
  \]

- Thin red arrow originating at $\vec{x}_{I_t}$:
  \[
  \eta n \left(y_{I_t} - \text{logistic}(\vec{x}_{I_t} \cdot \vec{w}(t-1))\right) \vec{x}_{I_t}
  \]

Iteration $t = 6$
\[\|\vec{w}^{(6)}\|_2 = 2.63168\]
Example: SGD for sum of logarithmic losses

- SGD update rule: For uniformly random $I_t$, 
  \[ \vec{w}^{(t)} := \vec{w}^{(t-1)} + \eta_n (y_{I_t} - \text{logistic}(\vec{x}_{I_t} \cdot \vec{w}^{(t-1)})) \vec{x}_{I_t} \]

- Thin red arrow originating at $\vec{x}_{I_t}$:
  \[ \eta_n (y_{I_t} - \text{logistic}(\vec{x}_{I_t} \cdot \vec{w}^{(t-1)})) \vec{x}_{I_t} \]

\[ \text{Iteration } t = 7 \]
\[ \|\vec{w}^{(7)}\|_2 = 3.12368 \]
Example: SGD for sum of logarithmic losses

- SGD update rule: For uniformly random $I_t$,
  \[
  \tilde{w}^{(t)} := \tilde{w}^{(t-1)} + \eta n \left( y_{I_t} - \text{logistic}(\tilde{x}_{I_t} \cdot \tilde{w}^{(t-1)}) \right) \tilde{x}_{I_t}
  \]

- Thin red arrow originating at $\tilde{x}_{I_t}$:
  \[
  \eta n \left( y_{I_t} - \text{logistic}(\tilde{x}_{I_t} \cdot \tilde{w}^{(t-1)}) \right) \tilde{x}_{I_t}
  \]

Iteration $t = 8$
\[
\|\tilde{w}^{(8)}\|_2 = 3.58322
\]
Example: SGD for sum of logarithmic losses

SGD update rule: For uniformly random $I_t$,

$$\vec{w}(t) := \vec{w}(t-1) + \eta n (y_{I_t} - \text{logistic}(\vec{x}_{I_t} \cdot \vec{w}(t-1))) \, \vec{x}_{I_t}$$

Thin red arrow originating at $\vec{x}_{I_t}$:

$$\eta n (y_{I_t} - \text{logistic}(\vec{x}_{I_t} \cdot \vec{w}(t-1))) \, \vec{x}_{I_t}$$

Iteration $t = 9$

$$\|\vec{w}(9)\|_2 = 3.99752$$
Example: SGD for sum of logarithmic losses

- SGD update rule: For uniformly random $I_t$,
  \[
  \tilde{w}(t) := \tilde{w}(t-1) + \eta n (y_{I_t} - \text{logistic}(\tilde{x}_{I_t} \cdot \tilde{w}(t-1))) \tilde{x}_{I_t}
  \]

- Thin red arrow originating at $\tilde{x}_{I_t}$:
  \[
  \eta n (y_{I_t} - \text{logistic}(\tilde{x}_{I_t} \cdot \tilde{w}(t-1))) \tilde{x}_{I_t}
  \]

- Iteration $t = 10$
  \[
  \|\tilde{w}^{(10)}\|_2 = 4.18768
  \]
Example: SGD for sum of logarithmic losses

- SGD update rule: For uniformly random $I_t$,

$$\vec{w}^{(t)} := \vec{w}^{(t-1)} + \eta_n \left( y_{I_t} - \text{logistic}(\vec{x}_{I_t} \cdot \vec{w}^{(t-1)}) \right) \vec{x}_{I_t}$$

- Thin red arrow originating at $\vec{x}_{I_t}$:

$$\eta_n \left( y_{I_t} - \text{logistic}(\vec{x}_{I_t} \cdot \vec{w}^{(t-1)}) \right) \vec{x}_{I_t}$$

Iteration $t = 11$

$$\|\vec{w}^{(11)}\|_2 = 2.82955$$
Example: SGD for sum of logarithmic losses

- SGD update rule: For uniformly random $I_t$,
\[ \mathbf{w}^{(t)} := \mathbf{w}^{(t-1)} + \eta_n \left( y_{I_t} - \text{logistic}(\mathbf{x}_{I_t} \cdot \mathbf{w}^{(t-1)}) \right) \mathbf{x}_{I_t} \]

- Thin red arrow originating at $\mathbf{x}_{I_t}$:
\[ \eta_n \left( y_{I_t} - \text{logistic}(\mathbf{x}_{I_t} \cdot \mathbf{w}^{(t-1)}) \right) \mathbf{x}_{I_t} \]

\[ \text{Iteration } t = 12 \]
\[ \| \mathbf{w}^{(12)} \|_2 = 3.31838 \]
Example: SGD for sum of logarithmic losses

- SGD update rule: For uniformly random $I_t$,
  \[ \vec{w}(t) := \vec{w}(t-1) + \eta n (y_{I_t} - \text{logistic}(\vec{x}_{I_t} \cdot \vec{w}(t-1))) \vec{x}_{I_t} \]

- Thin red arrow originating at $\vec{x}_{I_t}$:
  \[ \eta n (y_{I_t} - \text{logistic}(\vec{x}_{I_t} \cdot \vec{w}(t-1))) \vec{x}_{I_t} \]

Iteration $t = 13$

\[ \|\vec{w}(13)\|_2 = 3.74972 \]
Example: SGD for sum of logarithmic losses

- SGD update rule: For uniformly random $I_t$, 
  
  $$
  \vec{w}(t) := \vec{w}(t-1) + \eta n \left( y_{I_t} - \text{logistic}(\vec{x}_{I_t} \cdot \vec{w}(t-1)) \right) \vec{x}_{I_t}
  $$

- Thin red arrow originating at $\vec{x}_{I_t}$:
  
  $$
  \eta n \left( y_{I_t} - \text{logistic}(\vec{x}_{I_t} \cdot \vec{w}(t-1)) \right) \vec{x}_{I_t}
  $$

$\vec{w}(14)$:

- Iteration $t = 14$
- $\|\vec{w}(14)\|_2 = 4.11770$
Example: SGD for sum of logarithmic losses

- SGD update rule: For uniformly random $I_t$,
  \[
  \bar{w}^{(t)} := \bar{w}^{(t-1)} + \eta n (y_{I_t} - \text{logistic}(\bar{x}_{I_t} \cdot \bar{w}^{(t-1)})) \bar{x}_{I_t}
  \]

- Thin red arrow originating at $\bar{x}_{I_t}$:
  \[
  \eta n (y_{I_t} - \text{logistic}(\bar{x}_{I_t} \cdot \bar{w}^{(t-1)})) \bar{x}_{I_t}
  \]

Iteration $t = 15$
\[
\|\bar{w}^{(15)}\|_2 = 2.16104
\]
Example: SGD for sum of logarithmic losses

- SGD update rule: For uniformly random $I_t$, 
  \[
  \overline{w}^{(t)} := \overline{w}^{(t-1)} + \eta \ n \ (y_{I_t} - \text{logistic}(\overline{x}_{I_t} \cdot \overline{w}^{(t-1)})) \overline{x}_{I_t}
  \]

- Thin red arrow originating at $\overline{x}_{I_t}$:
  \[
  \eta \ n \ (y_{I_t} - \text{logistic}(\overline{x}_{I_t} \cdot \overline{w}^{(t-1)})) \overline{x}_{I_t}
  \]

Iteration $t = 16$
\[
\|\overline{w}^{(16)}\|_2 = 2.79472
\]
Example: SGD for sum of logarithmic losses

- SGD update rule: For uniformly random $I_t$,
  $$\overrightarrow{w}(t) := \overrightarrow{w}(t-1) + \eta \cdot n (y_{I_t} - \text{logistic}(\overrightarrow{x}_{I_t} \cdot \overrightarrow{w}(t-1))) \cdot \overrightarrow{x}_{I_t}$$

- Thin red arrow originating at $\overrightarrow{x}_{I_t}$:
  $$\eta \cdot n (y_{I_t} - \text{logistic}(\overrightarrow{x}_{I_t} \cdot \overrightarrow{w}(t-1))) \cdot \overrightarrow{x}_{I_t}$$

- Iteration $t = 17$
  $$\|\overrightarrow{w}^{(17)}\|_2 = 2.88693$$
Example: SGD for sum of logarithmic losses

- SGD update rule: For uniformly random $I_t$,
  \[ \bar{w}^{(t)} := \bar{w}^{(t-1)} + \eta n \left( y_{I_t} - \text{logistic}(\bar{x}_{I_t} \cdot \bar{w}^{(t-1)}) \right) \bar{x}_{I_t} \]

- Thin red arrow originating at $\bar{x}_{I_t}$:
  \[ \eta n \left( y_{I_t} - \text{logistic}(\bar{x}_{I_t} \cdot \bar{w}^{(t-1)}) \right) \bar{x}_{I_t} \]

\[ \text{Iteration } t = 18 \]
\[ \| \bar{w}^{(18)} \|_2 = 3.34341 \]
Example: SGD for sum of logarithmic losses

- SGD update rule: For uniformly random $I_t$,
  \[ \bar{w}(t) := \bar{w}(t-1) + \eta n \left( y_{I_t} - \text{logistic}(\bar{x}_{I_t} \cdot \bar{w}(t-1)) \right) \bar{x}_{I_t} \]

- Thin red arrow originating at $\bar{x}_{I_t}$:
  \[ \eta n \left( y_{I_t} - \text{logistic}(\bar{x}_{I_t} \cdot \bar{w}(t-1)) \right) \bar{x}_{I_t} \]
Example: SGD for sum of logarithmic losses

- SGD update rule: For uniformly random $I_t$,
  \[
  \mathbf{w}^{(t)} := \mathbf{w}^{(t-1)} + \eta n \left( y_{I_t} - \text{logistic}(\mathbf{x}_{I_t} \cdot \mathbf{w}^{(t-1)}) \right) \mathbf{x}_{I_t}
  \]

- Thin red arrow originating at $\mathbf{x}_{I_t}$:
  \[
  \eta n \left( y_{I_t} - \text{logistic}(\mathbf{x}_{I_t} \cdot \mathbf{w}^{(t-1)}) \right) \mathbf{x}_{I_t}
  \]

Iteration $t = 20$

$\|\mathbf{w}^{(20)}\|_2 = 4.13358$
Automatic Differentiation
Gradient computation

Both gradient descent and SGD rely on subroutines for gradient computation
Gradient computation

Both gradient descent and SGD rely on subroutines for gradient computation

- Can derive formula by hand in simple cases, and manually implement formula
Both gradient descent and SGD rely on subroutines for gradient computation

- Can derive formula by hand in simple cases, and manually implement formula
- But can get unwieldy in cases that are just moderately more complicated
Gradient computation

Both gradient descent and SGD rely on subroutines for gradient computation

- Can derive formula by hand in simple cases, and manually implement formula
- But can get unwieldy in cases that are just moderately more complicated
- Even less obvious is how to organize gradient computation in an efficient manner
Gradient computation

Both gradient descent and SGD rely on subroutines for gradient computation

- Can derive formula by hand in simple cases, and manually implement formula
- But can get unwieldy in cases that are just moderately more complicated
- Even less obvious is how to organize gradient computation in an efficient manner

Key idea: **Automatic differentiation (autodiff)**
(a.k.a. “backpropagation” in neural net context)

Seppo Linnainmaa, 1976
Typical gradient computation required in ML

Typical setup in ML:

- Prediction on input \( \vec{x} \) when "parameter vector" is \( \vec{w} \)
- "Loss" of prediction \( p \) when "correct" label is \( y \) (e.g., \( \ell(y, p) = (y - p)^2 \))
- Objective function:
  \[
  J(\vec{w}) = \sum_{i=1}^{n} \ell(y_i, f(\vec{w}, \vec{x}_i))
  \]
  (omit regularizers, for simplicity)

Gradient of objective function with respect to \( \vec{w} \):

\[
\nabla J(\vec{w}) = \sum_{i=1}^{n} \nabla \left[ \ell(y_i, f(\vec{w}, \vec{x}_i)) \right]
\]

So, for each example \((\vec{x}_i, y_i)\):

1. Compute prediction \( f(\vec{w}, \vec{x}_i) \)
2. Compute partial derivative of loss for \( i \)-th example, evaluated at prediction
3. Compute gradient of prediction on \( i \)-th example
Typical gradient computation required in ML

Typical setup in ML:

▸ $f_{\vec{w}}(\vec{x})$: prediction on input $\vec{x}$ when “parameter vector” is $\vec{w}$
Typical gradient computation required in ML

Typical setup in ML:

- $f_{\vec{w}}(\vec{x})$: prediction on input $\vec{x}$ when “parameter vector” is $\vec{w}$
- $\ell(y, p)$: “loss” of prediction $p$ when “correct” label is $y$ (e.g., $\ell(y, p) = (y - p)^2$)

1. Compute prediction $f_{\vec{w}}(\vec{x})$
2. Compute partial derivative of loss for $i$-th example, evaluated at prediction
3. Compute gradient of prediction on $i$-th example
Typical gradient computation required in ML

Typical setup in ML:

- $f_{\vec{w}}(\vec{x})$: prediction on input $\vec{x}$ when “parameter vector” is $\vec{w}$
- $\ell(y, p)$: “loss” of prediction $p$ when “correct” label is $y$ (e.g., $\ell(y, p) = (y - p)^2$)
- Objective function: $J(\vec{w}) = \sum_{i=1}^{n} \ell(y_i, f_{\vec{w}}(\vec{x}_i))$ (omit regularizers, for simplicity)
Typical setup in ML:

- $f_{\vec{w}}(\vec{x})$: prediction on input $\vec{x}$ when “parameter vector” is $\vec{w}$
- $\ell(y, p)$: “loss” of prediction $p$ when “correct” label is $y$ (e.g., $\ell(y, p) = (y - p)^2$)
- Objective function: $J(\vec{w}) = \sum_{i=1}^{n} \ell(y_i, f_{\vec{w}}(\vec{x}_i))$ (omit regularizers, for simplicity)

Gradient of objective function with respect to $\vec{w}$:

$$\nabla J(\vec{w}) = \nabla \left\{ \sum_{i=1}^{n} \ell(y_i, f_{\vec{w}}(\vec{x}_i)) \right\}$$
Typical gradient computation required in ML

Typical setup in ML:

- $f_{\vec{w}}(\vec{x})$: prediction on input $\vec{x}$ when “parameter vector” is $\vec{w}$
- $\ell(y, p)$: “loss” of prediction $p$ when “correct” label is $y$ (e.g., $\ell(y, p) = (y - p)^2$)
- Objective function: $J(\vec{w}) = \sum_{i=1}^{n} \ell(y_i, f_{\vec{w}}(\vec{x}_i))$ (omit regularizers, for simplicity)

Gradient of objective function with respect to $\vec{w}$:

$$\nabla J(\vec{w}) = \nabla \left\{ \sum_{i=1}^{n} \ell(y_i, f_{\vec{w}}(\vec{x}_i)) \right\} = \sum_{i=1}^{n} \nabla \left\{ \ell(y_i, f_{\vec{w}}(\vec{x}_i)) \right\}$$
Typical gradient computation required in ML

Typical setup in ML:

- $f_{\mathbf{w}}(\mathbf{x})$: prediction on input $\mathbf{x}$ when “parameter vector” is $\mathbf{w}$
- $\ell(y, p)$: “loss” of prediction $p$ when “correct” label is $y$  
  \[ (e.g., \ell(y, p) = (y - p)^2) \]
- Objective function: $J(\mathbf{w}) = \sum_{i=1}^{n} \ell(y_i, f_{\mathbf{w}}(\mathbf{x}_i))$  
  (omit regularizers, for simplicity)

Gradient of objective function with respect to $\mathbf{w}$:

\[
\nabla J(\mathbf{w}) = \nabla\left\{ \sum_{i=1}^{n} \ell(y_i, f_{\mathbf{w}}(\mathbf{x}_i)) \right\} = \sum_{i=1}^{n} \nabla\left\{ \ell(y_i, f_{\mathbf{w}}(\mathbf{x}_i)) \right\}
\]
\[
= \sum_{i=1}^{n} \left[ \frac{\partial \ell}{\partial p}(y_i, f_{\mathbf{w}}(\mathbf{x}_i)) \right] \nabla\left\{ f_{\mathbf{w}}(\mathbf{x}_i) \right\}
\]
Typical gradient computation required in ML

Typical setup in ML:

- \( f_\vec{w}(\vec{x}) \): prediction on input \( \vec{x} \) when “parameter vector” is \( \vec{w} \)
- \( \ell(y, p) \): “loss” of prediction \( p \) when “correct” label is \( y \)
- Objective function: \( J(\vec{w}) = \sum_{i=1}^{n} \ell(y_i, f_{\vec{w}}(\vec{x}_i)) \) (e.g., \( \ell(y, p) = (y - p)^2 \))

Gradient of objective function with respect to \( \vec{w} \):

\[
\nabla J(\vec{w}) = \nabla \left\{ \sum_{i=1}^{n} \ell(y_i, f_{\vec{w}}(\vec{x}_i)) \right\} = \sum_{i=1}^{n} \nabla \left\{ \ell(y_i, f_{\vec{w}}(\vec{x}_i)) \right\} \\
= \sum_{i=1}^{n} \left[ \frac{\partial \ell}{\partial p}(y_i, f_{\vec{w}}(\vec{x}_i)) \right] \nabla \left\{ f_{\vec{w}}(\vec{x}_i) \right\}
\]

So, for each example (\( \vec{x}_i, y_i \)):

1. Compute prediction \( f_{\vec{w}}(\vec{x}_i) \)
Typical gradient computation required in ML

Typical setup in ML:

- $f_{\vec{w}}(\vec{x})$: prediction on input $\vec{x}$ when “parameter vector” is $\vec{w}$
- $\ell(y, p)$: “loss” of prediction $p$ when “correct” label is $y$ (e.g., $\ell(y, p) = (y - p)^2$)
- Objective function: $J(\vec{w}) = \sum_{i=1}^{n} \ell(y_i, f_{\vec{w}}(\vec{x}_i))$ (omit regularizers, for simplicity)

Gradient of objective function with respect to $\vec{w}$:

$$\nabla J(\vec{w}) = \nabla \left\{ \sum_{i=1}^{n} \ell(y_i, f_{\vec{w}}(\vec{x}_i)) \right\} = \sum_{i=1}^{n} \nabla \left\{ \ell(y_i, f_{\vec{w}}(\vec{x}_i)) \right\}$$

$$= \sum_{i=1}^{n} \left[ \frac{\partial \ell}{\partial p}(y_i, f_{\vec{w}}(\vec{x}_i)) \right] \nabla \left\{ f_{\vec{w}}(\vec{x}_i) \right\}$$

So, for each example $(\vec{x}_i, y_i)$:

1. Compute prediction $f_{\vec{w}}(\vec{x}_i)$
2. Compute partial derivative of loss for $i$-th example, evaluated at prediction
Typical gradient computation required in ML

Typical setup in ML:
- $f_{\vec{w}}(\vec{x})$: prediction on input $\vec{x}$ when “parameter vector” is $\vec{w}$
- $\ell(y, p)$: “loss” of prediction $p$ when “correct” label is $y$  
  (e.g., $\ell(y, p) = (y - p)^2$)
- Objective function: $J(\vec{w}) = \sum_{i=1}^{n} \ell(y_i, f_{\vec{w}}(\vec{x}_i))$  
  (omit regularizers, for simplicity)

Gradient of objective function with respect to $\vec{w}$:

\[
\nabla J(\vec{w}) = \nabla \left\{ \sum_{i=1}^{n} \ell(y_i, f_{\vec{w}}(\vec{x}_i)) \right\} = \sum_{i=1}^{n} \nabla \left\{ \ell(y_i, f_{\vec{w}}(\vec{x}_i)) \right\} = \sum_{i=1}^{n} \left[ \frac{\partial \ell}{\partial p} (y_i, f_{\vec{w}}(\vec{x}_i)) \right] \nabla \{ f_{\vec{w}}(\vec{x}_i) \}
\]

So, for each example $(\vec{x}_i, y_i)$:
1. Compute prediction $f_{\vec{w}}(\vec{x}_i)$
2. Compute partial derivative of loss for $i$-th example, evaluated at prediction
3. Compute gradient of prediction on $i$-th example
Simple examples

Example 1:
Linear predictor $f_{\vec{w}}(\vec{x}) = \vec{x} \cdot \vec{w}$:

- Time to compute prediction: $O(d)$
- Time to compute gradient: $O(d)$
Simple examples

Example 1:
Linear predictor \( f_{\overrightarrow{w}}(\overrightarrow{x}) = \overrightarrow{x} \cdot \overrightarrow{w} \):

for each \( j \in [d] \),
\[
\frac{\partial}{\partial w_j} f_{\overrightarrow{w}}(\overrightarrow{x}) = \frac{\partial}{\partial w_j} \{ \overrightarrow{x} \cdot \overrightarrow{w} \}
\]

▶ Time to compute prediction: \( O(d) \)
▶ Time to compute gradient: \( O(d) \)
Simple examples

Example 1:
Linear predictor \( f_{\bar{w}}(\bar{x}) = \bar{x} \cdot \bar{w} \):

for each \( j \in [d] \),

\[
\frac{\partial}{\partial w_j} f_{\bar{w}}(\bar{x}) = \frac{\partial}{\partial w_j} \{ \bar{x} \cdot \bar{w} \} = x_j
\]
Simple examples

Example 1:
Linear predictor $f_{\vec{w}}(\vec{x}) = \vec{x} \cdot \vec{w}$:

for each $j \in [d]$, \[ \frac{\partial}{\partial w_j} f_{\vec{w}}(\vec{x}) = \frac{\partial}{\partial w_j} \{\vec{x} \cdot \vec{w}\} = x_j \]

- Time to compute prediction: $O(d)$
- Time to compute gradient: $O(d)$
Simple examples

Example 1:
Linear predictor \( f_{\vec{w}}(\vec{x}) = \vec{x} \cdot \vec{w} \):

\[
\text{for each } j \in [d], \quad \frac{\partial}{\partial w_j} f_{\vec{w}}(\vec{x}) = \frac{\partial}{\partial w_j} \{\vec{x} \cdot \vec{w}\} = x_j
\]

▶ Time to compute prediction: \( O(d) \)
▶ Time to compute gradient: \( O(d) \)

Example 2:
Nonlinear function \( f_{\vec{w}}(\vec{x}) = g(\vec{x} \cdot \vec{w}) \), for \( g(t) = \text{logistic}(t) = e^t/(1 + e^t) \):

\[
\text{for each } j \in [d], \quad \frac{\partial}{\partial w_j} g(\vec{x} \cdot \vec{w}) = \frac{\partial}{\partial w_j} \left\{ e^{\vec{x} \cdot \vec{w}} / (1 + e^{\vec{x} \cdot \vec{w}}) \right\} = x_j \frac{e^{\vec{x} \cdot \vec{w}}}{1 + e^{\vec{x} \cdot \vec{w}}} - \frac{e^{\vec{x} \cdot \vec{w}} \cdot \vec{x}}{1 + e^{\vec{x} \cdot \vec{w}}} = x_j \frac{e^{\vec{x} \cdot \vec{w}}}{(1 + e^{\vec{x} \cdot \vec{w}})^2}
\]
Simple examples

Example 1:
Linear predictor \( f_{\vec{w}}(\vec{x}) = \vec{x} \cdot \vec{w} \):

\[
\text{for each } j \in [d], \quad \frac{\partial}{\partial w_j} f_{\vec{w}}(\vec{x}) = \frac{\partial}{\partial w_j} \{\vec{x} \cdot \vec{w}\} = x_j
\]

- Time to compute prediction: \( O(d) \)
- Time to compute gradient: \( O(d) \)

Example 2:
Nonlinear function \( f_{\vec{w}}(\vec{x}) = g(\vec{x} \cdot \vec{w}) \), for \( g(t) = \text{logistic}(t) = e^t/(1 + e^t) \):

\[
\text{for each } j \in [d], \quad \frac{\partial}{\partial w_j} f_{\vec{w}}(\vec{x}) = \frac{\partial}{\partial w_j} \{g(\vec{x} \cdot \vec{w})\}
\]

- Time to compute prediction: \( O(d) \)
- Time to compute gradient: \( \text{naively, } O(d^2) \), but easy to make it \( O(d) \)
Simple examples

Example 1:
Linear predictor \( f_{\vec{w}}(\vec{x}) = \vec{x} \cdot \vec{w} \):

\[
\text{for each } j \in [d], \quad \frac{\partial}{\partial w_j} f_{\vec{w}}(\vec{x}) = \frac{\partial}{\partial w_j} \{\vec{x} \cdot \vec{w}\} = x_j
\]

- Time to compute prediction: \( O(d) \)
- Time to compute gradient: \( O(d) \)

Example 2:
Nonlinear function \( f_{\vec{w}}(\vec{x}) = g(\vec{x} \cdot \vec{w}) \), for \( g(t) = \text{logistic}(t) = e^t / (1 + e^t) \):

\[
\text{for each } j \in [d], \quad \frac{\partial}{\partial w_j} f_{\vec{w}}(\vec{x}) = \frac{\partial}{\partial w_j} \{g(\vec{x} \cdot \vec{w})\} = \left[ \frac{\partial g}{\partial t}(\vec{x} \cdot \vec{w}) \right] \frac{\partial}{\partial w_j} \{\vec{x} \cdot \vec{w}\}
\]
Simple examples

Example 1:
Linear predictor $f_{\vec{w}}(\vec{x}) = \vec{x} \cdot \vec{w}$:

$$\text{for each } j \in [d], \quad \frac{\partial}{\partial w_j} f_{\vec{w}}(\vec{x}) = \frac{\partial}{\partial w_j} \{\vec{x} \cdot \vec{w}\} = x_j$$

- **Time to compute prediction:** $O(d)$
- **Time to compute gradient:** $O(d)$

Example 2:
Nonlinear function $f_{\vec{w}}(\vec{x}) = g(\vec{x} \cdot \vec{w})$, for $g(t) = \text{logistic}(t) = e^t/(1 + e^t)$:

$$\text{for each } j \in [d], \quad \frac{\partial}{\partial w_j} f_{\vec{w}}(\vec{x}) = \frac{\partial}{\partial w_j} \{g(\vec{x} \cdot \vec{w})\} = \left[ \frac{\partial g}{\partial t} (\vec{x} \cdot \vec{w}) \right] \frac{\partial}{\partial w_j} \{\vec{x} \cdot \vec{w}\}$$

$$= \left[ \frac{\partial g}{\partial t} (\vec{x} \cdot \vec{w}) \right] x_j$$

- **Time to compute prediction:** $O(d)$
- **Time to compute gradient:** naïvely, $O(d^2)$, but easy to make it $O(d)$
Simple examples

Example 1:
Linear predictor \( f_{\bar{w}}(\bar{x}) = \bar{x} \cdot \bar{w} \):

for each \( j \in [d] \), \( \frac{\partial}{\partial w_j} f_{\bar{w}}(\bar{x}) = \frac{\partial}{\partial w_j} \{ \bar{x} \cdot \bar{w} \} = x_j \)

- Time to compute prediction: \( O(d) \)
- Time to compute gradient: \( O(d) \)

Example 2:
Nonlinear function \( f_{\bar{w}}(\bar{x}) = g(\bar{x} \cdot \bar{w}) \), for \( g(t) = \text{logistic}(t) = e^t/(1 + e^t) \):

for each \( j \in [d] \), \( \frac{\partial}{\partial w_j} f_{\bar{w}}(\bar{x}) = \frac{\partial}{\partial w_j} \{ g(\bar{x} \cdot \bar{w}) \} = \left[ \frac{\partial g}{\partial t}(\bar{x} \cdot \bar{w}) \right] \frac{\partial}{\partial w_j} \{ \bar{x} \cdot \bar{w} \} \\
= \left[ \frac{\partial g}{\partial t}(\bar{x} \cdot \bar{w}) \right] x_j \\
= \frac{e^{\bar{x} \cdot \bar{w}}}{(1 + e^{\bar{x} \cdot \bar{w}})^2} x_j \)
Simple examples

Example 1:
Linear predictor $f_{\vec{w}}(\vec{x}) = \vec{x} \cdot \vec{w}$:

for each $j \in [d]$, $\frac{\partial}{\partial w_j} f_{\vec{w}}(\vec{x}) = \frac{\partial}{\partial w_j} \{\vec{x} \cdot \vec{w}\} = x_j$

- Time to compute prediction: $O(d)$
- Time to compute gradient: $O(d)$

Example 2:
Nonlinear function $f_{\vec{w}}(\vec{x}) = g(\vec{x} \cdot \vec{w})$, for $g(t) = \text{logistic}(t) = e^t/(1 + e^t)$:

for each $j \in [d]$, $\frac{\partial}{\partial w_j} f_{\vec{w}}(\vec{x}) = \frac{\partial}{\partial w_j} \{g(\vec{x} \cdot \vec{w})\} = \left[ \frac{\partial g}{\partial t}(\vec{x} \cdot \vec{w}) \right] \frac{\partial}{\partial w_j} \{\vec{x} \cdot \vec{w}\}$

$= \left[ \frac{\partial g}{\partial t}(\vec{x} \cdot \vec{w}) \right] x_j$

$= \frac{e^{\vec{x} \cdot \vec{w}}}{(1 + e^{\vec{x} \cdot \vec{w}})^2} x_j$

- Time to compute prediction: $O(d)$
- Time to compute gradient: naively, $O(d^2)$, but easy to make it $O(d)$
Example 3:
Tower of exponentials $f_w(x) = \exp(\exp(\cdots \exp(xw) \cdots))$ (assume $x$ and $w$ are scalar-valued)
Example 3:
Tower of exponentials $f_w(x) = \exp(\exp(\cdots \exp(xw) \cdots))$  
(assume $x$ and $w$ are scalar-valued)

$$\frac{\partial}{\partial w} \left\{ e^{\cdots e^{xw}} \right\} = e^{\cdots e^{xw}} e^{\cdots e^{xw}} e^{\cdots e^{xw}} e^{\cdots e^{xw}} e^{\cdots e^{xw}} e^{\cdots e^{xw}} e^{\cdots e^{xw}} x$$
Example 3: Tower of exponentials $f_w(x) = \exp(\exp(\cdots \exp(xw) \cdots))$ (assume $x$ and $w$ are scalar-valued)

\[
\frac{\partial}{\partial w} \begin{cases} 
\text{exw} \\
\text{exw} \\
\text{exw} \\
\text{exw} \\
\text{exw} \\
\text{exw} \\
\text{exw} \\
\text{exw} \\
\text{exw}
\end{cases} = \begin{cases} 
\text{exw} \\
\text{exw} \\
\text{exw} \\
\text{exw} \\
\text{exw} \\
\text{exw} \\
\text{exw} \\
\text{exw} \\
\text{exw}
\end{cases} \begin{cases} 
\text{xw} \\
\text{xw} \\
\text{xw} \\
\text{xw} \\
\text{xw} \\
\text{xw} \\
\text{xw} \\
\text{xw} \\
\text{xw}
\end{cases}
\]

- Time to compute tower of exponentials of height $h$: $O(h)$
- Time to compute derivative: naively, $O(h^2)$, but can make it $O(h)$
Example 4:
\[ f_w(x) = \exp(xw + \sin(xw)) + \sin^2(xw)w \]
Example 4: 
\[ f_w(x) = \exp(xw + \sin(xw)) + \sin^2(xw)w \]

Function computation

\[
\begin{align*}
  v_1 &:= \text{prod}(x, w) \\
  v_2 &:= \sin(v_1) \\
  v_3 &:= \text{sum}(v_1, v_2) \\
  v_4 &:= \text{square}(v_2) \\
  v_5 &:= \exp(v_3) \\
  v_6 &:= \text{prod}(v_4, w) \\
  v_7 &:= \text{sum}(v_5, v_6) \\
  \text{out} &:= v_7
\end{align*}
\]
Example 4:
$f_w(x) = \exp(xw + \sin(xw)) + \sin^2(xw)w$

Function computation

\[
\begin{align*}
  v_1 & := \text{prod}(x, w) \\
  v_2 & := \sin(v_1) \\
  v_3 & := \text{sum}(v_1, v_2) \\
  v_4 & := \text{square}(v_2) \\
  v_5 & := \exp(v_3) \\
  v_6 & := \text{prod}(v_4, w) \\
  v_7 & := \text{sum}(v_5, v_6) \\
  \text{out} & := v_7
\end{align*}
\]

Computation graph
**Example 4:**
\[ f_w(x) = \exp(xw + \sin(xw)) + \sin^2(xw)w \]

**Function computation**

\[
\begin{align*}
v_1 &:= \text{prod}(x, w) \\
v_2 &:= \sin(v_1) \\
v_3 &:= \text{sum}(v_1, v_2) \\
v_4 &:= \text{square}(v_2) \\
v_5 &:= \exp(v_3) \\
v_6 &:= \text{prod}(v_4, w) \\
v_7 &:= \text{sum}(v_5, v_6) \\
\text{out} &:= v_7
\end{align*}
\]

**Computation graph**

**Derivative computation**

\[ \frac{\partial \text{out}}{\partial w} = ??? \]
Even more complicated example

**Example 4:**

\[ f_w(x) = \exp(xw + \sin(xw)) + \sin^2(xw)w \]

**Function computation**

\[
\begin{align*}
    v_1 & := \text{prod}(x, w) \\
    v_2 & := \sin(v_1) \\
    v_3 & := \text{sum}(v_1, v_2) \\
    v_4 & := \text{square}(v_2) \\
    v_5 & := \exp(v_3) \\
    v_6 & := \text{prod}(v_4, w) \\
    v_7 & := \text{sum}(v_5, v_6) \\
    \text{out} & := v_7
\end{align*}
\]

**Computation graph**

**Derivative computation**

\[
\frac{\partial \text{out}}{\partial w} = \ ??
\]

“Local” partial derivatives

\[
\begin{align*}
    \frac{\partial v_4}{\partial v_2} & = 2v_2 \\
    \vdots
\end{align*}
\]
Forward pass:

- Compute value of each node given inputs
  (Here, inputs are $x$ and $w$)

$v_1 := \text{prod}(x, w)$
$v_2 := \sin(v_1)$
$v_3 := \text{sum}(v_1, v_2)$
$v_4 := \text{square}(v_2)$
$v_5 := \exp(v_3)$
$v_6 := \text{prod}(v_4, w)$
$v_7 := \text{sum}(v_5, v_6)$
out := $v_7$
Backward pass:

- Compute partial derivatives of output w.r.t. node variables

\[
\frac{\partial \text{out}}{\partial v},
\]

evaluated at current node values (computed in forward pass)
\[
\frac{\partial \text{out}}{\partial v_3} = \frac{\partial \text{out}}{\partial v_5} \cdot \frac{\partial v_5}{\partial v_3} = \frac{\partial \text{out}}{\partial v_5} \cdot \exp(v_3)
\]

\[
\frac{\partial \text{out}}{\partial v_2} = \frac{\partial \text{out}}{\partial v_3} \cdot \frac{\partial v_3}{\partial v_2} + \frac{\partial \text{out}}{\partial v_4} \cdot \frac{\partial v_4}{\partial v_2} = \frac{\partial \text{out}}{\partial v_3} \cdot 1 + \frac{\partial \text{out}}{\partial v_4} \cdot 2v_2
\]

\[
\frac{\partial \text{out}}{\partial v_1} = \frac{\partial \text{out}}{\partial v_2} \cdot \frac{\partial v_2}{\partial v_1} + \frac{\partial \text{out}}{\partial v_3} \cdot \frac{\partial v_3}{\partial v_1} = \frac{\partial \text{out}}{\partial v_2} \cdot \cos(v_1) + \frac{\partial \text{out}}{\partial v_3} \cdot 1
\]
\[
\frac{\partial \text{out}}{\partial w} = \frac{\partial \text{out}}{\partial v_1} \cdot \frac{\partial v_1}{\partial w} + \frac{\partial \text{out}}{\partial v_6} \cdot \frac{\partial v_6}{\partial w} \\
= \frac{\partial \text{out}}{\partial v_1} \cdot x + \frac{\partial \text{out}}{\partial v_6} \cdot v_4
\]

\[
\frac{\partial \text{out}}{\partial x} = \frac{\partial \text{out}}{\partial v_1} \cdot \frac{\partial v_1}{\partial x} \\
= \frac{\partial \text{out}}{\partial v_1} \cdot w
\]
\[
\frac{\partial \text{out}}{\partial w} = \frac{\partial \text{out}}{\partial v_1} \cdot \frac{\partial v_1}{\partial w} + \frac{\partial \text{out}}{\partial v_6} \cdot \frac{\partial v_6}{\partial w} \\
= \frac{\partial \text{out}}{\partial v_1} \cdot x + \frac{\partial \text{out}}{\partial v_6} \cdot v_4
\]

\[
\frac{\partial \text{out}}{\partial x} = \frac{\partial \text{out}}{\partial v_1} \cdot \frac{\partial v_1}{\partial x} \\
= \frac{\partial \text{out}}{\partial v_1} \cdot w
\]

- **Time to compute function:** \(O(\text{size of computation graph})\)
- **Time to compute all partial derivatives:** \(O(\text{size of computation graph})\)
Backward pass (fin)

\[
\frac{\partial \text{out}}{\partial w} = \frac{\partial \text{out}}{\partial v_1} \cdot \frac{\partial v_1}{\partial w} + \frac{\partial \text{out}}{\partial v_6} \cdot \frac{\partial v_6}{\partial w}
\]

\[
= \frac{\partial \text{out}}{\partial v_1} \cdot x + \frac{\partial \text{out}}{\partial v_6} \cdot v_4
\]

\[
\frac{\partial \text{out}}{\partial x} = \frac{\partial \text{out}}{\partial v_1} \cdot \frac{\partial v_1}{\partial x}
\]

\[
= \frac{\partial \text{out}}{\partial v_1} \cdot w
\]

- **Time to compute function:** $O(\text{size of computation graph})$
- **Time to compute all partial derivatives:** $O(\text{size of computation graph})$

Modern numerical software (e.g., pytorch, tensorflow) facilitate construction of computation graph
Backward pass (fin)

\[
\frac{\partial \text{out}}{\partial w} = \frac{\partial \text{out}}{\partial v_1} \cdot \frac{\partial v_1}{\partial w} + \frac{\partial \text{out}}{\partial v_6} \cdot \frac{\partial v_6}{\partial w} \\
= \frac{\partial \text{out}}{\partial v_1} \cdot x + \frac{\partial \text{out}}{\partial v_6} \cdot v_4
\]

\[
\frac{\partial \text{out}}{\partial x} = \frac{\partial \text{out}}{\partial v_1} \cdot \frac{\partial v_1}{\partial x} \\
= \frac{\partial \text{out}}{\partial v_1} \cdot w
\]

- **Time to compute function:** \(O(\text{size of computation graph})\)
- **Time to compute all partial derivatives:** \(O(\text{size of computation graph})\)

Modern numerical software (e.g., pytorch, tensorflow) facilitate construction of computation graph

- Sometimes “parameters” not explicitly shown in computation graph
Backward pass (fin)

\[
\frac{\partial \text{out}}{\partial w} = \frac{\partial \text{out}}{\partial v_1} \cdot \frac{\partial v_1}{\partial w} + \frac{\partial \text{out}}{\partial v_6} \cdot \frac{\partial v_6}{\partial w} \\
= \frac{\partial \text{out}}{\partial v_1} \cdot x + \frac{\partial \text{out}}{\partial v_6} \cdot v_4
\]

\[
\frac{\partial \text{out}}{\partial x} = \frac{\partial \text{out}}{\partial v_1} \cdot \frac{\partial v_1}{\partial x} \\
= \frac{\partial \text{out}}{\partial v_1} \cdot w
\]

- **Time to compute function:** \(O(\text{size of computation graph})\)
- **Time to compute all partial derivatives:** \(O(\text{size of computation graph})\)

Modern numerical software (e.g., pytorch, tensorflow) facilitate construction of computation graph

- Sometimes “parameters” not explicitly shown in computation graph
- Build prediction function out of library functions (which come with partial derivative subroutines)
Backward pass (fin)

\[
\begin{align*}
\frac{\partial \text{out}}{\partial w} &= \frac{\partial \text{out}}{\partial v_1} \cdot \frac{\partial v_1}{\partial w} + \frac{\partial \text{out}}{\partial v_6} \cdot \frac{\partial v_6}{\partial w} \\
&= \frac{\partial \text{out}}{\partial v_1} \cdot x + \frac{\partial \text{out}}{\partial v_6} \cdot v_4 \\
\frac{\partial \text{out}}{\partial x} &= \frac{\partial \text{out}}{\partial v_1} \cdot \frac{\partial v_1}{\partial x} \\
&= \frac{\partial \text{out}}{\partial v_1} \cdot w
\end{align*}
\]

- **Time to compute function:** $O(\text{size of computation graph})$
- **Time to compute all partial derivatives:** $O(\text{size of computation graph})$

Modern numerical software (e.g., pytorch, tensorflow) facilitate construction of computation graph

- Sometimes “parameters” not explicitly shown in computation graph
- Build prediction function out of library functions (which come with partial derivative subroutines)
- Some library functions pack in a lot of computation (e.g., matrix-vector multiply)
Actually, don’t need to treat “loss” separately; just include as another node in computation graph.

Can also build graph for sum of losses on all training examples.

(Figure omitted for your sanity . . . )
Actually, don’t need to treat “loss” separately; just include as another node in computation graph.

Can also build graph for sum of losses on all training examples. (Figure omitted for your sanity . . . )
Postscript and recap

- Gradient descent and its variants are among the most commonly used numerical optimization algorithms in ML
  - Some experimentation may be needed for effective use
- Many other numerical optimization algorithms:
  - Newton’s method
  - Conjugate gradient methods
  - Quasi-Newton methods
  - Frank-Wolfe methods
  - . . .
- Autodiff: Facilitates gradient computation by efficiently organizing computation