Supervised Learning (in 1 Lecture)

Lucas Janson

CS/Stat 184(0): Introduction to Reinforcement Learning

Fall 2024

Today

- Feedback from last lecture
- Recap
- Supervised learning setup
- Linear regression
- Neural networks

Feedback from feedback forms

Feedback from feedback forms

1. Thank you to everyone who filled out the forms!

Feedback from feedback forms

1. Thank you to everyone who filled out the forms!
2. Posted lecture slides

Today

- ✓ Feedback from last lecture
- Recap
- Supervised learning setup
- Linear regression
- Neural networks

Contextual bandit environment

Contextual bandit environment

Context at time t encoded into a variable x_t that we see before choosing our action

Contextual bandit environment

Context at time t encoded into a variable x_t that we see before choosing our action

x_t is drawn i.i.d. at each time point from a distribution ν_x on sample space $\mathcal{X}$

Contextual bandit environment

Context at time t encoded into a variable x_t that we see before choosing our action

x_t is drawn i.i.d. at each time point from a distribution ν_x on sample space $\mathcal{X}$

x_t then affects the reward distributions of each arm, i.e., if we choose arm k, we get a reward that is drawn from a distribution that depends on x_t, namely, $\nu^{(k)}(x_t)$

Contextual bandit environment

Context at time t encoded into a variable x_t that we see before choosing our action

x_t is drawn i.i.d. at each time point from a distribution ν_x on sample space $\mathcal{X}$

x_t then affects the reward distributions of each arm, i.e., if we choose arm k, we get a reward that is drawn from a distribution that depends on x_t, namely, $\nu^{(k)}(x_t)$

Accordingly, we should also choose our action a_t in a way that depends on x_t, i.e., our action should be chosen by a function of x_t (a policy), namely, $\pi_t(x_t)$

Contextual bandit environment

Context at time t encoded into a variable x_t that we see before choosing our action

x_t is drawn i.i.d. at each time point from a distribution ν_x on sample space $\mathcal{X}$

x_t then affects the reward distributions of each arm, i.e., if we choose arm k, we get a reward that is drawn from a distribution that depends on x_t, namely, $\nu^{(k)}(x_t)$

Accordingly, we should also choose our action a_t in a way that depends on x_t, i.e., our action should be chosen by a function of x_t (a policy), namely, $\pi_t(x_t)$

If we knew everything about the environment, we'd want to use the optimal policy

$$\pi^\star(x_t) := \arg\max_{k\in\{1,\ldots,K\}} \mu^{(k)}(x_t), \qquad \text{where } \mu^{(k)}(x) := \mathbb{E}_{r\sim\nu^{(k)}(x)}[r]$$

Contextual bandit environment

Context at time t encoded into a variable x_t that we see before choosing our action

x_t is drawn i.i.d. at each time point from a distribution ν_x on sample space $\mathscr{X}$

x_t then affects the reward distributions of each arm, i.e., if we choose arm k, we get a reward that is drawn from a distribution that depends on x_t, namely, $\nu^{(k)}(x_t)$

Accordingly, we should also choose our action a_t in a way that depends on x_t, i.e., our action should be chosen by a function of x_t (a policy), namely, $\pi_t(x_t)$

If we knew everything about the environment, we'd want to use the optimal policy

$$\pi^\star(x_t) := \arg\max_{k \in \{1,\ldots,K\}} \mu^{(k)}(x_t), \qquad \text{where } \mu^{(k)}(x) := \mathbb{E}_{r \sim \nu^{(k)}(x)}[r]$$

$\pi^\star$ is the policy we compare to in computing regret

UCB for contextual bandits

UCB for contextual bandits

UCB algorithm also conceptually identical as long as $|\mathcal{X}|$ finite:

$$\pi_t(x_t) = \arg\max_k \hat{\mu}_t^{(k)}(x_t) + \sqrt{\ln(2TK|\mathcal{X}|/\delta)/2N_t^{(k)}(x_t)}$$

UCB for contextual bandits

UCB algorithm also conceptually identical as long as $|\mathcal{X}|$ finite:

$$\pi_t(x_t) = \arg\max_k \hat{\mu}_t^{(k)}(x_t) + \sqrt{\ln(2TK|\mathcal{X}|/\delta)/2N_t^{(k)}(x_t)}$$

- Added x_t argument to $\hat{\mu}_t^{(k)}$ and $N_t^{(k)}$ since we now keep track of the sample mean and number of arm pulls *separately* for each value of the context

UCB for contextual bandits

UCB algorithm also conceptually identical as long as $|\mathcal{X}|$ finite:

$$\pi_t(x_t) = \arg\max_k \hat{\mu}_t^{(k)}(x_t) + \sqrt{\ln(2TK|\mathcal{X}|/\delta)/2N_t^{(k)}(x_t)}$$

- Added x_t argument to $\hat{\mu}_t^{(k)}$ and $N_t^{(k)}$ since we now keep track of the sample mean and number of arm pulls *separately* for each value of the context
- Added $|\mathcal{X}|$ inside the log because our union bound argument is now over all arm mean estimates $\hat{\mu}_t^{(k)}(x)$, of which there are $K|\mathcal{X}|$ instead of just K

UCB for contextual bandits

UCB algorithm also conceptually identical as long as $|\mathcal{X}|$ finite:

$$\pi_t(x_t) = \arg\max_k \hat{\mu}_t^{(k)}(x_t) + \sqrt{\ln(2TK|\mathcal{X}|/\delta)/2N_t^{(k)}(x_t)}$$

- Added x_t argument to $\hat{\mu}_t^{(k)}$ and $N_t^{(k)}$ since we now keep track of the sample mean and number of arm pulls *separately* for each value of the context
- Added $|\mathcal{X}|$ inside the log because our union bound argument is now over all arm mean estimates $\hat{\mu}_t^{(k)}(x)$, of which there are $K|\mathcal{X}|$ instead of just K

But when $|\mathcal{X}|$ is really big (or even infinite), this will be really bad!

UCB for contextual bandits

UCB algorithm also conceptually identical as long as $|\mathcal{X}|$ finite:

$$\pi_t(x_t) = \arg\max_k \hat{\mu}_t^{(k)}(x_t) + \sqrt{\ln(2TK|\mathcal{X}|/\delta)/2N_t^{(k)}(x_t)}$$

- Added x_t argument to $\hat{\mu}_t^{(k)}$ and $N_t^{(k)}$ since we now keep track of the sample mean and number of arm pulls *separately* for each value of the context
- Added $|\mathcal{X}|$ inside the log because our union bound argument is now over all arm mean estimates $\hat{\mu}_t^{(k)}(x)$, of which there are $K|\mathcal{X}|$ instead of just K

But when $|\mathcal{X}|$ is really big (or even infinite), this will be really bad!

Solution: share information across contexts x_t, i.e., don't treat $\nu^{(k)}(x)$ and $\nu^{(k)}(x')$ as completely different distributions which have nothing to do with one another

UCB for contextual bandits

UCB algorithm also conceptually identical as long as $|\mathcal{X}|$ finite:

$$\pi_t(x_t) = \arg\max_k \hat{\mu}_t^{(k)}(x_t) + \sqrt{\ln(2TK|\mathcal{X}|/\delta)/2N_t^{(k)}(x_t)}$$

- Added x_t argument to $\hat{\mu}_t^{(k)}$ and $N_t^{(k)}$ since we now keep track of the sample mean and number of arm pulls *separately* for each value of the context
- Added $|\mathcal{X}|$ inside the log because our union bound argument is now over all arm mean estimates $\hat{\mu}_t^{(k)}(x)$, of which there are $K|\mathcal{X}|$ instead of just K

But when $|\mathcal{X}|$ is really big (or even infinite), this will be really bad!

Solution: share information across contexts x_t, i.e., don't treat $\nu^{(k)}(x)$ and $\nu^{(k)}(x')$ as completely different distributions which have nothing to do with one another

Example: showing an ad on a NYT article on politics vs a NYT article on sports:

UCB for contextual bandits

UCB algorithm also conceptually identical as long as $|\mathcal{X}|$ finite:

$$\pi_t(x_t) = \arg\max_k \hat{\mu}_t^{(k)}(x_t) + \sqrt{\ln(2TK|\mathcal{X}|/\delta)/2N_t^{(k)}(x_t)}$$

- Added x_t argument to $\hat{\mu}_t^{(k)}$ and $N_t^{(k)}$ since we now keep track of the sample mean and number of arm pulls *separately* for each value of the context
- Added $|\mathcal{X}|$ inside the log because our union bound argument is now over all arm mean estimates $\hat{\mu}_t^{(k)}(x)$, of which there are $K|\mathcal{X}|$ instead of just K

But when $|\mathcal{X}|$ is really big (or even infinite), this will be really bad!

Solution: share information across contexts x_t, i.e., don't treat $\nu^{(k)}(x)$ and $\nu^{(k)}(x')$ as completely different distributions which have nothing to do with one another

Example: showing an ad on a NYT article on politics vs a NYT article on sports: Not *identical* readership, but still both on NYT, so probably still *similar* readership!

Modeling in contextual bandits

Modeling in contextual bandits

Need a model for $\mu^{(k)}(x)$, e.g., a linear model: $\mu^{(k)}(x) = \theta_k^\top x$

Modeling in contextual bandits

Need a model for $\mu^{(k)}(x)$, e.g., a linear model: $\mu^{(k)}(x) = \theta_k^\top x$

E.g., placing ads on NYT or WSJ (encoded as 0 or 1 in the first entry of x), for articles on politics or sports (encoded as 0 or 1 in the second entry of x) $\Rightarrow x \in \{0,1\}^2$

Modeling in contextual bandits

Need a model for $\mu^{(k)}(x)$, e.g., a linear model: $\mu^{(k)}(x) = \theta_k^\top x$

E.g., placing ads on NYT or WSJ (encoded as 0 or 1 in the first entry of x), for articles on politics or sports (encoded as 0 or 1 in the second entry of x) $\Rightarrow x \in \{0,1\}^2$

$|\mathcal{X}| = 4 \Rightarrow$ w/o linear model, need to learn 4 different $\mu^{(k)}(x)$ values for each arm k

Modeling in contextual bandits

Need a model for $\mu^{(k)}(x)$, e.g., a linear model: $\mu^{(k)}(x) = \theta_k^\top x$

E.g., placing ads on NYT or WSJ (encoded as 0 or 1 in the first entry of x), for articles on politics or sports (encoded as 0 or 1 in the second entry of x) $\Rightarrow x \in \{0,1\}^2$

$|\mathcal{X}| = 4 \Rightarrow$ w/o linear model, need to learn 4 different $\mu^{(k)}(x)$ values for each arm k

With linear model there are just 2 parameters: the two entries of $\theta_k \in \mathbb{R}^2$

Modeling in contextual bandits

Need a model for $\mu^{(k)}(x)$, e.g., a linear model: $\mu^{(k)}(x) = \theta_k^\top x$

E.g., placing ads on NYT or WSJ (encoded as 0 or 1 in the first entry of x), for articles on politics or sports (encoded as 0 or 1 in the second entry of x) $\Rightarrow x \in \{0,1\}^2$

$|\mathcal{X}| = 4$ $\Rightarrow$ w/o linear model, need to learn 4 different $\mu^{(k)}(x)$ values for each arm k

With linear model there are just 2 parameters: the two entries of $\theta_k \in \mathbb{R}^2$

Lower dimension makes learning easier, but model could be wrong/biased

Modeling in contextual bandits

Need a model for $\mu^{(k)}(x)$, e.g., a linear model: $\mu^{(k)}(x) = \theta_k^\top x$

E.g., placing ads on NYT or WSJ (encoded as 0 or 1 in the first entry of x), for articles on politics or sports (encoded as 0 or 1 in the second entry of x) $\Rightarrow x \in \{0,1\}^2$

$|\mathscr{X}| = 4$ $\Rightarrow$ w/o linear model, need to learn 4 different $\mu^{(k)}(x)$ values for each arm k

With linear model there are just 2 parameters: the two entries of $\theta_k \in \mathbb{R}^2$

Lower dimension makes learning easier, but model could be wrong/biased

Choosing the best model, fitting it, and quantifying uncertainty are really questions of supervised learning

Today

- ✓ Feedback from last lecture
- ✓ Recap
- Supervised learning setup
- Linear regression
- Neural networks

Supervised learning setup

Supervised learning setup

Data: i.i.d. pairs $(y_1, x_1), \ldots, (y_n, x_n)$ drawn from distribution $\mathbb{P}(y, x) = \mathbb{P}(y \mid x)\mathbb{P}(x)$

Supervised learning setup

Data: i.i.d. pairs $(y_1, x_1), \ldots, (y_n, x_n)$ drawn from distribution $\mathbb{P}(y, x) = \mathbb{P}(y \mid x)\mathbb{P}(x)$

Goal: learn a good predictor $f(x)$ of y

Supervised learning setup

Data: i.i.d. pairs $(y_1, x_1), \ldots, (y_n, x_n)$ drawn from distribution $\mathbb{P}(y, x) = \mathbb{P}(y \mid x)\mathbb{P}(x)$

Goal: learn a good predictor $f(x)$ of y

Note: $\mathbb{E}[y \mid x]$ minimizes mean squared error

Supervised learning setup

Data: i.i.d. pairs $(y_1, x_1), \ldots, (y_n, x_n)$ drawn from distribution $\mathbb{P}(y, x) = \mathbb{P}(y \mid x)\mathbb{P}(x)$

Goal: learn a good predictor $f(x)$ of y

Note: $\mathbb{E}[y \mid x]$ minimizes mean squared error

$\text{MSE}(f) = \mathbb{E}[(y - f(x))^2]$

Supervised learning setup

Data: i.i.d. pairs $(y_1, x_1), \ldots, (y_n, x_n)$ drawn from distribution $\mathbb{P}(y, x) = \mathbb{P}(y \mid x)\mathbb{P}(x)$

Goal: learn a good predictor $f(x)$ of y

Note: $\mathbb{E}[y \mid x]$ minimizes mean squared error

$$\text{MSE}(f) = \mathbb{E}[(y - f(x))^2]$$
$$= \mathbb{E}[(y - \mathbb{E}[y \mid x] + \mathbb{E}[y \mid x] - f(x))^2]$$

Supervised learning setup

Data: i.i.d. pairs $(y_1, x_1), \ldots, (y_n, x_n)$ drawn from distribution $\mathbb{P}(y, x) = \mathbb{P}(y \mid x)\mathbb{P}(x)$

Goal: learn a good predictor $f(x)$ of y

Note: $\mathbb{E}[y \mid x]$ minimizes mean squared error

$$
\begin{aligned}
\text{MSE}(f) &= \mathbb{E}[(y - f(x))^2] \\
&= \mathbb{E}[(y - \mathbb{E}[y \mid x] + \mathbb{E}[y \mid x] - f(x))^2] \\
&= \mathbb{E}[(y - \mathbb{E}[y \mid x])^2] + \mathbb{E}[(\mathbb{E}[y \mid x] - f(x))^2] + 2\mathbb{E}[(y - \mathbb{E}[y \mid x])(\mathbb{E}[y \mid x] - f(x))]
\end{aligned}
$$

Supervised learning setup

Data: i.i.d. pairs $(y_1, x_1), \ldots, (y_n, x_n)$ drawn from distribution $\mathbb{P}(y, x) = \mathbb{P}(y \mid x)\mathbb{P}(x)$

Goal: learn a good predictor $f(x)$ of y

Note: $\mathbb{E}[y \mid x]$ minimizes mean squared error

$$\text{MSE}(f) = \mathbb{E}[(y - f(x))^2]$$

$$= \mathbb{E}[(y - \mathbb{E}[y \mid x] + \mathbb{E}[y \mid x] - f(x))^2]$$

$$= \mathbb{E}[(y - \mathbb{E}[y \mid x])^2] + \mathbb{E}[(\mathbb{E}[y \mid x] - f(x))^2] + 2\mathbb{E}[(y - \mathbb{E}[y \mid x])(\mathbb{E}[y \mid x] - f(x))]$$

$$\mathbb{E}\left[\left(y - \mathbb{E}[y \mid x]\right)\left(\mathbb{E}[y \mid x] - f(x)\right)\right] = \mathbb{E}\left[\mathbb{E}\left[\left(y - \mathbb{E}[y \mid x]\right)\left(\mathbb{E}[y \mid x] - f(x)\right) \Big| x\right]\right]$$

Supervised learning setup

Data: i.i.d. pairs $(y_1, x_1), \ldots, (y_n, x_n)$ drawn from distribution $\mathbb{P}(y, x) = \mathbb{P}(y \mid x)\mathbb{P}(x)$

Goal: learn a good predictor $f(x)$ of y

Note: $\mathbb{E}[y \mid x]$ minimizes mean squared error

$$\text{MSE}(f) = \mathbb{E}[(y - f(x))^2]$$

$$= \mathbb{E}[(y - \mathbb{E}[y \mid x] + \mathbb{E}[y \mid x] - f(x))^2]$$

$$= \mathbb{E}[(y - \mathbb{E}[y \mid x])^2] + \mathbb{E}[(\mathbb{E}[y \mid x] - f(x))^2] + 2\mathbb{E}[(y - \mathbb{E}[y \mid x])(\mathbb{E}[y \mid x] - f(x))]$$

$$\mathbb{E}\left[\left(y - \mathbb{E}[y \mid x]\right)\left(\mathbb{E}[y \mid x] - f(x)\right)\right] = \mathbb{E}\left[\mathbb{E}\left[\left(y - \mathbb{E}[y \mid x]\right)\left(\mathbb{E}[y \mid x] - f(x)\right) \middle| x\right]\right]$$

$$= \mathbb{E}\left[\mathbb{E}\left[\left(y - \mathbb{E}[y \mid x]\right) \middle| x\right]\left(\mathbb{E}[y \mid x] - f(x)\right)\right]$$

Supervised learning setup

Data: i.i.d. pairs $(y_1, x_1), \ldots, (y_n, x_n)$ drawn from distribution $\mathbb{P}(y, x) = \mathbb{P}(y \mid x)\mathbb{P}(x)$

Goal: learn a good predictor $f(x)$ of y

Note: $\mathbb{E}[y \mid x]$ minimizes mean squared error

$$\text{MSE}(f) = \mathbb{E}[(y - f(x))^2]$$
$$= \mathbb{E}[(y - \mathbb{E}[y \mid x] + \mathbb{E}[y \mid x] - f(x))^2]$$
$$= \mathbb{E}[(y - \mathbb{E}[y \mid x])^2] + \mathbb{E}[(\mathbb{E}[y \mid x] - f(x))^2] + 2\mathbb{E}[(y - \mathbb{E}[y \mid x])(\mathbb{E}[y \mid x] - f(x))]$$

$$\mathbb{E}\left[\left(y - \mathbb{E}[y \mid x]\right)\left(\mathbb{E}[y \mid x] - f(x)\right)\right] = \mathbb{E}\left[\mathbb{E}\left[\left(y - \mathbb{E}[y \mid x]\right)\left(\mathbb{E}[y \mid x] - f(x)\right) \middle| x\right]\right]$$
$$= \mathbb{E}\left[\mathbb{E}\left[\left(y - \mathbb{E}[y \mid x]\right) \middle| x\right]\left(\mathbb{E}[y \mid x] - f(x)\right)\right] = 0$$

Supervised learning setup

Data: i.i.d. pairs $(y_1, x_1), \ldots, (y_n, x_n)$ drawn from distribution $\mathbb{P}(y, x) = \mathbb{P}(y \mid x)\mathbb{P}(x)$

Goal: learn a good predictor $f(x)$ of y

Note: $\mathbb{E}[y \mid x]$ minimizes mean squared error

$$\text{MSE}(f) = \mathbb{E}[(y - f(x))^2]$$
$$= \mathbb{E}[(y - \mathbb{E}[y \mid x] + \mathbb{E}[y \mid x] - f(x))^2]$$
$$= \mathbb{E}[(y - \mathbb{E}[y \mid x])^2] + \mathbb{E}[(\mathbb{E}[y \mid x] - f(x))^2] + 2\mathbb{E}[(y - \mathbb{E}[y \mid x])(\mathbb{E}[y \mid x] - f(x))]$$

$$\mathbb{E}\left[\left(y - \mathbb{E}[y \mid x]\right)\left(\mathbb{E}[y \mid x] - f(x)\right)\right] = \mathbb{E}\left[\mathbb{E}\left[\left(y - \mathbb{E}[y \mid x]\right)\left(\mathbb{E}[y \mid x] - f(x)\right) \middle| x\right]\right]$$
$$= \mathbb{E}\left[\mathbb{E}\left[\left(y - \mathbb{E}[y \mid x]\right) \middle| x\right]\left(\mathbb{E}[y \mid x] - f(x)\right)\right] = 0$$

How do we now know that $f(x) = \mathbb{E}[y \mid x]$ minimizes MSE?

Empirical risk minimization (ERM)

Fact: $\mathbb{E}[y \mid x] = \arg\min_f \mathbb{E}[(y - f(x))^2]$

Empirical risk minimization (ERM)

Fact: $\mathbb{E}[y \mid x] = \arg\min_f \mathbb{E}[(y - f(x))^2]$

This fact both motivates $\mathbb{E}[y \mid x]$ as a target for learning, and suggests how to do it

Empirical risk minimization (ERM)

Fact: $\mathbb{E}[y \mid x] = \arg\min_f \mathbb{E}[(y - f(x))^2]$

This fact both motivates $\mathbb{E}[y \mid x]$ as a target for learning, and suggests how to do it

Law of large numbers: $\mathbb{E}[(y - f(x))^2] \approx \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$

Empirical risk minimization (ERM)

Fact: $\mathbb{E}[y \mid x] = \arg\min_f \mathbb{E}[(y - f(x))^2]$

This fact both motivates $\mathbb{E}[y \mid x]$ as a target for learning, and suggests how to do it

Law of large numbers: $\mathbb{E}[(y - f(x))^2] \approx \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$ =: training error of f

Empirical risk minimization (ERM)

Fact: $\mathbb{E}[y \mid x] = \arg\min_{f} \mathbb{E}[(y - f(x))^2]$

This fact both motivates $\mathbb{E}[y \mid x]$ as a target for learning, and suggests how to do it

Law of large numbers: $\mathbb{E}[(y - f(x))^2] \approx \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$ =: training error of f

Empirical risk minimization (ERM): $\hat{f}(x) = \arg\min_{f} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$

Empirical risk minimization (ERM)

Fact: $\mathbb{E}[y \mid x] = \arg\min_f \mathbb{E}[(y - f(x))^2]$

This fact both motivates $\mathbb{E}[y \mid x]$ as a target for learning, and suggests how to do it

Law of large numbers: $\mathbb{E}[(y - f(x))^2] \approx \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$ =: training error of f

Empirical risk minimization (ERM): $\hat{f}(x) = \arg\min_f \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$

Seems great, but if we allow f in the argmin to range over *all* functions, we can get ridiculous solutions. Can anyone think of one?

Empirical risk minimization (ERM)

Fact: $\mathbb{E}[y \mid x] = \arg\min_f \mathbb{E}[(y - f(x))^2]$

This fact both motivates $\mathbb{E}[y \mid x]$ as a target for learning, and suggests how to do it

Law of large numbers: $\mathbb{E}[(y - f(x))^2] \approx \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$ =: training error of f

Empirical risk minimization (ERM): $\hat{f}(x) = \arg\min_f \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$

Seems great, but if we allow f in the argmin to range over *all* functions, we can get ridiculous solutions. Can anyone think of one?

E.g., $f(x) = \sum_{i=1}^{n} y_i 1_{\{x=x_i\}}$ achieves zero training error (as long as no ties in the x_i's)

Empirical risk minimization (ERM)

Fact: $\mathbb{E}[y \mid x] = \arg\min_f \mathbb{E}[(y - f(x))^2]$

This fact both motivates $\mathbb{E}[y \mid x]$ as a target for learning, and suggests how to do it

Law of large numbers: $\mathbb{E}[(y - f(x))^2] \approx \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$ =: training error of f

Empirical risk minimization (ERM): $\hat{f}(x) = \arg\min_f \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$

Seems great, but if we allow f in the argmin to range over *all* functions, we can get ridiculous solutions. Can anyone think of one?

E.g., $f(x) = \sum_{i=1}^{n} y_i 1_{\{x = x_i\}}$ achieves zero training error (as long as no ties in the x_i's)

But it predicts 0 at every x value not in the training data, regardless of the data!

Function classes

Function classes

Need to constrain ERM to a function class $\mathcal{F}$: $\hat{f}(x) = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$

Function classes

Need to constrain ERM to a function class $\mathcal{F}$: $\hat{f}(x) = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$

E.g. (if x scalar) quadratic functions: $\mathcal{F} = \{f(x) = ax^2 + bx + c : (a, b, c) \in \mathbb{R}^3\}$

Function classes

Need to constrain ERM to a function class $\mathcal{F}$: $\hat{f}(x) = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$

E.g. (if x scalar) quadratic functions: $\mathcal{F} = \{f(x) = ax^2 + bx + c : (a, b, c) \in \mathbb{R}^3\}$

How to choose $\mathcal{F}$?

Function classes

Need to constrain ERM to a function class $\mathcal{F}$: $\hat{f}(x) = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$

E.g. (if x scalar) quadratic functions: $\mathcal{F} = \{f(x) = ax^2 + bx + c : (a, b, c) \in \mathbb{R}^3\}$

How to choose $\mathcal{F}$? Three main high-level criteria:

1. Approximation: $\mathbb{E}[y \mid x] \approx \arg\min_{f \in \mathcal{F}} \mathbb{E}[(y - f(x))^2]$
2. Complexity: $\mathcal{F}$ doesn't contain "too many" functions/dimensions
3. Optimizable: need to be able to compute the argmin (or something like it)

Function classes

Need to constrain ERM to a function class $\mathscr{F}$: $\hat{f}(x) = \arg\min_{f \in \mathscr{F}} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$

E.g. (if x scalar) quadratic functions: $\mathscr{F} = \{f(x) = ax^2 + bx + c : (a, b, c) \in \mathbb{R}^3\}$

How to choose $\mathscr{F}$? Three main high-level criteria:

1. Approximation: $\mathbb{E}[y \mid x] \approx \arg\min_{f \in \mathscr{F}} \mathbb{E}[(y - f(x))^2]$
2. Complexity: $\mathscr{F}$ doesn't contain "too many" functions/dimensions
3. Optimizable: need to be able to compute the argmin (or something like it)

<u>Statistical learning theory</u>: the ERM optimum (criterion 3) $\hat{f}$ will perform well if $\mathscr{F}$'s approximation error (criterion 1) and complexity (criterion 2) are low

Optimization

Optimization

Typically our function class $\mathscr{F}$ is parameterized by a parameter vector $\theta \in \mathbb{R}^d$, i.e., every $f \in \mathscr{F}$ can be written as $f_\theta(x)$ for some $\theta \in \mathbb{R}^d$

Optimization

Typically our function class $\mathscr{F}$ is parameterized by a parameter vector $\theta \in \mathbb{R}^d$, i.e., every $f \in \mathscr{F}$ can be written as $f_\theta(x)$ for some $\theta \in \mathbb{R}^d$

Parameterized ERM optimization: $\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2$; $\hat{f} = f_{\hat{\theta}}$

Optimization

Typically our function class $\mathscr{F}$ is parameterized by a parameter vector $\theta \in \mathbb{R}^d$,
i.e., every $f \in \mathscr{F}$ can be written as $f_\theta(x)$ for some $\theta \in \mathbb{R}^d$

Parameterized ERM optimization: $\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2$; $\hat{f} = f_{\hat{\theta}}$

<u>Notation</u>: $L_i(\theta) = (y_i - f_\theta(x_i))^2$, $L(\theta) = \frac{1}{n} \sum_{i=1}^{n} L_i(\theta)$, gradient operator ∇_θ

Optimization

Typically our function class $\mathscr{F}$ is parameterized by a parameter vector $\theta \in \mathbb{R}^d$, i.e., every $f \in \mathscr{F}$ can be written as $f_\theta(x)$ for some $\theta \in \mathbb{R}^d$

Parameterized ERM optimization: $\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2;$ $\hat{f} = f_{\hat{\theta}}$

Notation: $L_i(\theta) = (y_i - f_\theta(x_i))^2,$ $L(\theta) = \frac{1}{n} \sum_{i=1}^{n} L_i(\theta),$ gradient operator ∇_θ

Gradient descent: initialize at θ_0, update via $\theta^{(i+1)} = \theta^{(i)} - \eta \nabla_\theta L(\theta^{(i)})$

Optimization

Typically our function class $\mathscr{F}$ is parameterized by a parameter vector $\theta \in \mathbb{R}^d$, i.e., every $f \in \mathscr{F}$ can be written as $f_\theta(x)$ for some $\theta \in \mathbb{R}^d$

Parameterized ERM optimization: $\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2$; $\hat{f} = f_{\hat{\theta}}$

<u>Notation</u>: $L_i(\theta) = (y_i - f_\theta(x_i))^2$, $L(\theta) = \frac{1}{n} \sum_{i=1}^{n} L_i(\theta)$, gradient operator ∇_θ

Gradient descent: initialize at θ_0, update via $\theta^{(i+1)} = \theta^{(i)} - \eta \nabla_\theta L(\theta^{(i)})$

Downside: computing $\nabla_\theta L(\theta^{(i)})$ at each step expensive for big data

Optimization

Typically our function class $\mathscr{F}$ is parameterized by a parameter vector $\theta \in \mathbb{R}^d$, i.e., every $f \in \mathscr{F}$ can be written as $f_\theta(x)$ for some $\theta \in \mathbb{R}^d$

Parameterized ERM optimization: $\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2;$ $\hat{f} = f_{\hat{\theta}}$

Notation: $L_i(\theta) = (y_i - f_\theta(x_i))^2,$ $L(\theta) = \frac{1}{n} \sum_{i=1}^{n} L_i(\theta),$ gradient operator ∇_θ

Gradient descent: initialize at θ_0, update via $\theta^{(i+1)} = \theta^{(i)} - \eta \nabla_\theta L(\theta^{(i)})$

Downside: computing $\nabla_\theta L(\theta^{(i)})$ at each step expensive for big data

Stochastic gradient descent: initialize at θ_0, update via $\theta^{(i+1)} = \theta^{(i)} - \eta \nabla_\theta L_i(\theta^{(i)})$

Optimization

Typically our function class $\mathscr{F}$ is parameterized by a parameter vector $\theta \in \mathbb{R}^d$,
i.e., every $f \in \mathscr{F}$ can be written as $f_\theta(x)$ for some $\theta \in \mathbb{R}^d$

Parameterized ERM optimization: $\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2$; $\hat{f} = f_{\hat{\theta}}$

Notation: $L_i(\theta) = (y_i - f_\theta(x_i))^2$, $L(\theta) = \frac{1}{n} \sum_{i=1}^{n} L_i(\theta)$, gradient operator ∇_θ

Gradient descent: initialize at θ_0, update via $\theta^{(i+1)} = \theta^{(i)} - \eta \nabla_\theta L(\theta^{(i)})$

Downside: computing $\nabla_\theta L(\theta^{(i)})$ at each step expensive for big data

Stochastic gradient descent: initialize at θ_0, update via $\theta^{(i+1)} = \theta^{(i)} - \eta \nabla_\theta L_i(\theta^{(i)})$

Can do multiple passes of data, or uses batch size $b > 1$ at each step

Optimization

Typically our function class $\mathcal{F}$ is parameterized by a parameter vector $\theta \in \mathbb{R}^d$,
i.e., every $f \in \mathcal{F}$ can be written as $f_\theta(x)$ for some $\theta \in \mathbb{R}^d$

Parameterized ERM optimization: $\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (y_i - f_\theta(x_i))^2$; $\hat{f} = f_{\hat{\theta}}$

Notation: $L_i(\theta) = (y_i - f_\theta(x_i))^2$, $L(\theta) = \frac{1}{n} \sum_{i=1}^{n} L_i(\theta)$, gradient operator ∇_θ

Gradient descent: initialize at θ_0, update via $\theta^{(i+1)} = \theta^{(i)} - \eta \nabla_\theta L(\theta^{(i)})$

Downside: computing $\nabla_\theta L(\theta^{(i)})$ at each step expensive for big data

Stochastic gradient descent: initialize at θ_0, update via $\theta^{(i+1)} = \theta^{(i)} - \eta \nabla_\theta L_i(\theta^{(i)})$

Can do multiple passes of data, or uses batch size $b > 1$ at each step

Main takeaway: this works (for good choices of b and η, which may vary with i)

Today

- ✓ Feedback from last lecture
- ✓ Recap
- ✓ Supervised learning setup
- Linear regression
- Neural networks

Linear model

Linear model

Linear model (if $d = \dim(x)$, let $\theta \in \mathbb{R}^d$): $f_\theta(x) = x^\top \theta$

Linear model

Linear model (if $d = \dim(x)$, let $\theta \in \mathbb{R}^d$): $f_\theta(x) = x^\top \theta$

ERM optimization: $\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^\top \theta)^2$

Linear model

Linear model (if $d = \dim(x)$, let $\theta \in \mathbb{R}^d$): $f_\theta(x) = x^\top \theta$

ERM optimization: $\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^\top \theta)^2$

Let $Y := (y_1, \ldots, y_n) \in \mathbb{R}^n$ and $X := (x_1^\top; \ldots; x_n^\top) \in \mathbb{R}^{n \times d}$, can rewrite ERM as:

$$\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{2} \|Y - X\theta\|^2$$

Linear model

Linear model (if $d = \dim(x)$, let $\theta \in \mathbb{R}^d$): $f_\theta(x) = x^\top \theta$

ERM optimization: $\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^\top \theta)^2$

Let $Y := (y_1, \ldots, y_n) \in \mathbb{R}^n$ and $X := (x_1^\top; \ldots; x_n^\top) \in \mathbb{R}^{n \times d}$, can rewrite ERM as:

$$\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{2} \|Y - X\theta\|^2$$

Let $L(\theta) = \frac{1}{2}\|Y - X\theta\|^2$: $\quad \nabla_\theta L(\theta) = -X^\top(Y - X\theta), \quad \nabla_\theta L_i(\theta) = -x_i(y_i - x_i^\top \theta)$

Linear model

Linear model (if $d = \dim(x)$, let $\theta \in \mathbb{R}^d$): $f_\theta(x) = x^\top \theta$

ERM optimization: $\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^\top \theta)^2$

Let $Y := (y_1, \ldots, y_n) \in \mathbb{R}^n$ and $X := (x_1^\top; \ldots; x_n^\top) \in \mathbb{R}^{n \times d}$, can rewrite ERM as:

$$\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{2} \|Y - X\theta\|^2$$

Let $L(\theta) = \frac{1}{2}\|Y - X\theta\|^2$: $\quad \nabla_\theta L(\theta) = -X^\top (Y - X\theta), \quad \nabla_\theta L_i(\theta) = -x_i(y_i - x_i^\top \theta)$

Instead of (S)GD, $\nabla_\theta L(\theta) = 0$ leads to closed-form solution $\hat{\theta} = (X^\top X)^{-1} X^\top Y$

Linear model

Linear model (if $d = \dim(x)$, let $\theta \in \mathbb{R}^d$): $f_\theta(x) = x^\top \theta$

ERM optimization: $\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^\top \theta)^2$

Let $Y := (y_1, \ldots, y_n) \in \mathbb{R}^n$ and $X := (x_1^\top; \ldots; x_n^\top) \in \mathbb{R}^{n \times d}$, can rewrite ERM as:

$$\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{2} \|Y - X\theta\|^2$$

Let $L(\theta) = \frac{1}{2}\|Y - X\theta\|^2$: $\quad \nabla_\theta L(\theta) = -X^\top (Y - X\theta), \quad \nabla_\theta L_i(\theta) = -x_i(y_i - x_i^\top \theta)$

Instead of (S)GD, $\nabla_\theta L(\theta) = 0$ leads to closed-form solution $\hat{\theta} = (X^\top X)^{-1} X^\top Y$

If $n < d$, $X^\top X$ non-invertible; many solutions exists (think: fitting line through 1 point)

Linear model

Linear model (if $d = \dim(x)$, let $\theta \in \mathbb{R}^d$): $f_\theta(x) = x^\top \theta$

ERM optimization: $\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^\top \theta)^2$

Let $Y := (y_1, \ldots, y_n) \in \mathbb{R}^n$ and $X := (x_1^\top; \ldots; x_n^\top) \in \mathbb{R}^{n \times d}$, can rewrite ERM as:

$$\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^d} \frac{1}{2} \|Y - X\theta\|^2$$

Let $L(\theta) = \frac{1}{2}\|Y - X\theta\|^2$: $\quad \nabla_\theta L(\theta) = -X^\top(Y - X\theta), \quad \nabla_\theta L_i(\theta) = -x_i(y_i - x_i^\top \theta)$

Instead of (S)GD, $\nabla_\theta L(\theta) = 0$ leads to closed-form solution $\hat{\theta} = (X^\top X)^{-1} X^\top Y$

If $n < d$, $X^\top X$ non-invertible; many solutions exists (think: fitting line through 1 point)

Surprising fact: GD initialized at 0 finds solution with smallest norm!

Notes on linear models

Notes on linear models

1. Can work surprisingly well in practice, especially in high dimensions

Notes on linear models

1. Can work surprisingly well in practice, especially in high dimensions
   a) Linear functions approximate smooth functions pretty well, if very smooth

Notes on linear models

1. Can work surprisingly well in practice, especially in high dimensions
   a) Linear functions approximate smooth functions pretty well, if very smooth
2. Need good features

Notes on linear models

1. Can work surprisingly well in practice, especially in high dimensions
    a) Linear functions approximate smooth functions pretty well, if very smooth
2. Need good features
    a) Can use domain knowledge to construct transformation $\phi(x)$ which can be higher- or lower-dimensional than x, and then just use linear model in $\phi(x)$

Notes on linear models

1. Can work surprisingly well in practice, especially in high dimensions
    a) Linear functions approximate smooth functions pretty well, if very smooth
2. Need good features
    a) Can use domain knowledge to construct transformation $\phi(x)$ which can be higher- or lower-dimensional than x, and then just use linear model in $\phi(x)$
3. Adding penalty to ERM objective can help a lot, especially in high dimensions

Notes on linear models

1. Can work surprisingly well in practice, especially in high dimensions
   a) Linear functions approximate smooth functions pretty well, if very smooth
2. Need good features
   a) Can use domain knowledge to construct transformation $\phi(x)$ which can be higher- or lower-dimensional than x, and then just use linear model in $\phi(x)$
3. Adding penalty to ERM objective can help a lot, especially in high dimensions
   a) Ridge penalty: add $\lambda \sum_{j=1}^{d} \theta_j^2$ to training loss to discourage huge $\hat{\theta}$ entries

Notes on linear models

1. Can work surprisingly well in practice, especially in high dimensions
   a) Linear functions approximate smooth functions pretty well, if very smooth
2. Need good features
   a) Can use domain knowledge to construct transformation $\phi(x)$ which can be higher- or lower-dimensional than x, and then just use linear model in $\phi(x)$
3. Adding penalty to ERM objective can help a lot, especially in high dimensions
   a) Ridge penalty: add $\lambda \sum_{j=1}^{d} \theta_j^2$ to training loss to discourage huge $\hat{\theta}$ entries
   b) Lasso penalty: add $\lambda \sum_{j=1}^{d} |\theta_j|$ to training loss to encourage sparse $\hat{\theta}$

Today

- ✓ Feedback from last lecture
- ✓ Recap
- ✓ Supervised learning setup
- ✓ Linear regression
- Neural networks

Neural network model

Neural network model

Building blocks:

Neural network model

Building blocks:

1. Linear transformation (multiplication by matrix W, then addition by vector b)

Neural network model

Building blocks:

1. Linear transformation (multiplication by matrix W, then addition by vector b)
2. Nonlinear transformation σ, e.g., ReLU $\sigma(a) = \max(a, 0)$, applied element-wise

Neural network model

Building blocks:

1. Linear transformation (multiplication by matrix W, then addition by vector b)
2. Nonlinear transformation σ, e.g., ReLU $\sigma(a) = \max(a, 0)$, applied element-wise

Simplest nontrivial NN is $f(x) = W_2\sigma(W_1x + b_1) + b_2$. Can think of as:

Neural network model

Building blocks:

1. Linear transformation (multiplication by matrix W, then addition by vector b)
2. Nonlinear transformation σ, e.g., ReLU $\sigma(a) = \max(a, 0)$, applied element-wise

Simplest nontrivial NN is $f(x) = W_2\sigma(W_1x + b_1) + b_2$. Can think of as:

1. Start with input $x \in \mathbb{R}^d$,

Neural network model

Building blocks:

1. Linear transformation (multiplication by matrix W, then addition by vector b)
2. Nonlinear transformation σ, e.g., ReLU $\sigma(a) = \max(a, 0)$, applied element-wise

Simplest nontrivial NN is $f(x) = W_2 \sigma(W_1 x + b_1) + b_2$. Can think of as:

1. Start with input $x \in \mathbb{R}^d$,
2. Linearly transform with $W_1 \in \mathbb{R}^{m \times d}$ and $b_1 \in \mathbb{R}^m$ to get $W_1 x + b_1 \in \mathbb{R}^m$

Neural network model

Building blocks:

1. Linear transformation (multiplication by matrix W, then addition by vector b)
2. Nonlinear transformation σ, e.g., ReLU $\sigma(a) = \max(a, 0)$, applied element-wise

Simplest nontrivial NN is $f(x) = W_2\sigma(W_1x+b_1)+b_2$. Can think of as:

1. Start with input $x \in \mathbb{R}^d$,
2. Linearly transform with $W_1 \in \mathbb{R}^{m\times d}$ and $b_1 \in \mathbb{R}^m$ to get $W_1x+b_1 \in \mathbb{R}^m$
3. Apply (element-wise) the nonlinearity σ to get $\sigma(W_1x+b_1) \in \mathbb{R}^m$

Neural network model

Building blocks:

1. Linear transformation (multiplication by matrix W, then addition by vector b)
2. Nonlinear transformation σ, e.g., ReLU $\sigma(a) = \max(a,0)$, applied element-wise

Simplest nontrivial NN is $f(x) = W_2\sigma(W_1x+b_1)+b_2$. Can think of as:

1. Start with input $x \in \mathbb{R}^d$,
2. Linearly transform with $W_1 \in \mathbb{R}^{m\times d}$ and $b_1 \in \mathbb{R}^m$ to get $W_1x+b_1 \in \mathbb{R}^m$
3. Apply (element-wise) the nonlinearity σ to get $\sigma(W_1x+b_1) \in \mathbb{R}^m$
4. Linearly transform with $W_2 \in \mathbb{R}^{1\times m}$ and $b_2 \in \mathbb{R}$ to get $W_2\sigma(W_1x+b_1)+b_2 \in \mathbb{R}$

Neural network model

Building blocks:

1. Linear transformation (multiplication by matrix W, then addition by vector b)
2. Nonlinear transformation σ, e.g., ReLU $\sigma(a) = \max(a, 0)$, applied element-wise

Simplest nontrivial NN is $f(x) = W_2\sigma(W_1x + b_1) + b_2$. Can think of as:

1. Start with input $x \in \mathbb{R}^d$,
2. Linearly transform with $W_1 \in \mathbb{R}^{m\times d}$ and $b_1 \in \mathbb{R}^m$ to get $W_1x + b_1 \in \mathbb{R}^m$
3. Apply (element-wise) the nonlinearity σ to get $\sigma(W_1x + b_1) \in \mathbb{R}^m$
4. Linearly transform with $W_2 \in \mathbb{R}^{1\times m}$ and $b_2 \in \mathbb{R}$ to get $W_2\sigma(W_1x + b_1) + b_2 \in \mathbb{R}$

With p layers: $f(x) = W_p\sigma(W_{p-1}\sigma(\cdots\ \sigma(W_1x + b_1)\ \cdots) + b_{p-1}) + b_p$

Neural network model

Building blocks:

1. Linear transformation (multiplication by matrix W, then addition by vector b)
2. Nonlinear transformation σ, e.g., ReLU $\sigma(a) = \max(a, 0)$, applied element-wise

Simplest nontrivial NN is $f(x) = W_2\sigma(W_1x + b_1) + b_2$. Can think of as:

1. Start with input $x \in \mathbb{R}^d$,
2. Linearly transform with $W_1 \in \mathbb{R}^{m \times d}$ and $b_1 \in \mathbb{R}^m$ to get $W_1x + b_1 \in \mathbb{R}^m$
3. Apply (element-wise) the nonlinearity σ to get $\sigma(W_1x + b_1) \in \mathbb{R}^m$
4. Linearly transform with $W_2 \in \mathbb{R}^{1 \times m}$ and $b_2 \in \mathbb{R}$ to get $W_2\sigma(W_1x + b_1) + b_2 \in \mathbb{R}$

With p layers: $f(x) = W_p\sigma(W_{p-1}\sigma(\cdots\ \sigma(W_1x + b_1)\ \cdots) + b_{p-1}) + b_p$

Parameter vector θ concatenates all W's and b's; $\dim(\theta)$ scales as width$^2 \times$ depth

Optimizing the neural network

Optimizing the neural network

Computing gradients, even stochastic gradients $\nabla_\theta L_i(\theta)$, is daunting

Optimizing the neural network

Computing gradients, even stochastic gradients $\nabla_\theta L_i(\theta)$, is daunting

A trick called backpropagation allows such gradients to be computed efficiently

Optimizing the neural network

Computing gradients, even stochastic gradients $\nabla_\theta L_i(\theta)$, is daunting

A trick called backpropagation allows such gradients to be computed efficiently

Too notationally cumbersome to cover here, but basically the hierarchical structure of neural networks plays very nicely with the chain rule (see Wikipedia or many other sources on internet for more)

Optimizing the neural network

Computing gradients, even stochastic gradients $\nabla_\theta L_i(\theta)$, is daunting

A trick called backpropagation allows such gradients to be computed efficiently

Too notationally cumbersome to cover here, but basically the hierarchical structure of neural networks plays very nicely with the chain rule (see Wikipedia or many other sources on internet for more)

Unfortunately, $L(\theta)$ is non-convex, i.e., it will in general have many local optima

Optimizing the neural network

Computing gradients, even stochastic gradients $\nabla_\theta L_i(\theta)$, is daunting

A trick called backpropagation allows such gradients to be computed efficiently

Too notationally cumbersome to cover here, but basically the hierarchical structure of neural networks plays very nicely with the chain rule (see Wikipedia or many other sources on internet for more)

Unfortunately, $L(\theta)$ is non-convex, i.e., it will in general have many local optima

We hope that SGD finds a good one… in practice there are optimization tricks that are like SGD but perform better, e.g., one very popular one is called Adam

Notes on NNs

Notes on NNs

1. Work well for all problems, breaking criterion 1 (approximation)

Notes on NNs

1. Work well for all problems, breaking criterion 1 (approximation)
    a) Actually, NNs need a lot of data, and are often worse than classical methods on smaller data sets

Notes on NNs

1. Work well for all problems, breaking criterion 1 (approximation)
   a) Actually, NNs need a lot of data, and are often worse than classical methods on smaller data sets
   b) Many of the most famous / impressive NNs, such as CNNs for vision or AlphaFold for protein structure, heavily incorporate problem-specific structure into their models

Notes on NNs

1. Work well for all problems, breaking criterion 1 (approximation)
   a) Actually, NNs need a lot of data, and are often worse than classical methods on smaller data sets
   b) Many of the most famous / impressive NNs, such as CNNs for vision or AlphaFold for protein structure, heavily incorporate problem-specific structure into their models
2. Work better when larger / more complex, breaking criterion 2 (complexity)

Notes on NNs

1. Work well for all problems, breaking criterion 1 (approximation)
    a) Actually, NNs need a lot of data, and are often worse than classical methods on smaller data sets
    b) Many of the most famous / impressive NNs, such as CNNs for vision or AlphaFold for protein structure, heavily incorporate problem-specific structure into their models
2. Work better when larger / more complex, breaking criterion 2 (complexity)
    a) This is true, though larger / more complex NNs also need more data to train

Notes on NNs

1. Work well for all problems, breaking criterion 1 (approximation)
    a) Actually, NNs need a lot of data, and are often worse than classical methods on smaller data sets
    b) Many of the most famous / impressive NNs, such as CNNs for vision or AlphaFold for protein structure, heavily incorporate problem-specific structure into their models
2. Work better when larger / more complex, breaking criterion 2 (complexity)
    a) This is true, though larger / more complex NNs also need more data to train
    b) The number of NN parameters is not a good measure of its “complexity”

Notes on NNs

1. Work well for all problems, breaking criterion 1 (approximation)
    a) Actually, NNs need a lot of data, and are often worse than classical methods on smaller data sets
    b) Many of the most famous / impressive NNs, such as CNNs for vision or AlphaFold for protein structure, heavily incorporate problem-specific structure into their models
2. Work better when larger / more complex, breaking criterion 2 (complexity)
    a) This is true, though larger / more complex NNs also need more data to train
    b) The number of NN parameters is not a good measure of its “complexity”
3. Are highly non-convex, breaking criterion 3 (optimization)

Notes on NNs

1. Work well for all problems, breaking criterion 1 (approximation)
    a) Actually, NNs need a lot of data, and are often worse than classical methods on smaller data sets
    b) Many of the most famous / impressive NNs, such as CNNs for vision or AlphaFold for protein structure, heavily incorporate problem-specific structure into their models
2. Work better when larger / more complex, breaking criterion 2 (complexity)
    a) This is true, though larger / more complex NNs also need more data to train
    b) The number of NN parameters is not a good measure of its "complexity"
3. Are highly non-convex, breaking criterion 3 (optimization)
    a) The optimizers used for NNs don't find arbitrary solutions, they actually find "low-complexity" solutions!

Today

- ✓ Feedback from last lecture
- ✓ Recap
- ✓ Supervised learning setup
- ✓ Linear regression
- ✓ Neural networks

Summary:

- Given data comprised of a bunch of (y, x) pairs, there exists a huge toolbox (a whole field's worth) to approximate the function $\mathbb{E}[y \mid x]$
- Generally, we write down a squared-error loss function for a parameterized function class and optimize it via (possibly stochastic) gradient descent

Attendance:

bit.ly/3RcTC9T

Feedback:

bit.ly/3RHtlxy