10-701: Introduction to Machine Learning

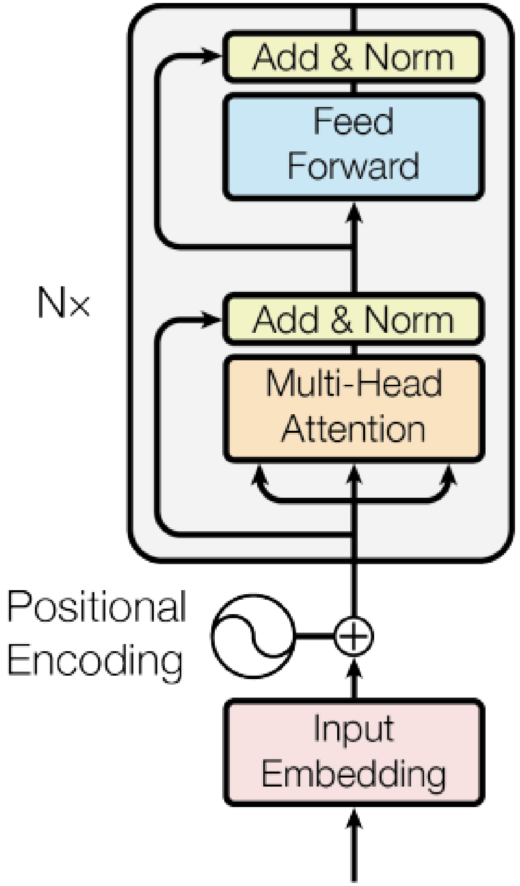
Lecture 14 – Unsupervised Learning: Clustering

Hoda Heidari

* Slides adopted from F24 offering of 10701 by Henry Chai.

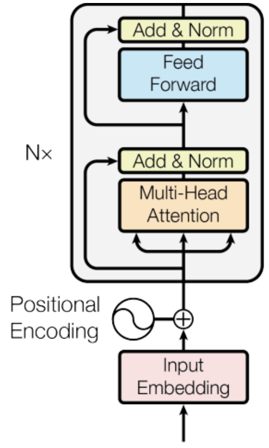
Transformers

1. **Embed words:** Words are turned into vectors (points in a space that roughly encode semantic meaning).
2. **Attention Scoring:** Each word looks at all other words and asks: “How relevant are you to me given the task at hand?”
3. **Blend information:** Each word builds a new representation by taking a weighted mix of the other words, weighted by their scores.
4. **Repeat:** Stack that many times--layers repeat the same pattern, so deeper layers see richer relational structure/representations.



Source: https://arxiv.org/pdf/1706.03762.pdf

The Attention Mechanism



Analogy: attention as soft lookup in a dictionary

1. The token x' broadcasts its **query** $q = w_Q^T x'$
2. Every other token x_t offers up its **key** $k_t = W_K x_t$
3. The model computes a *similarity* between the query and keys--scoring how relevant each token is to the query.

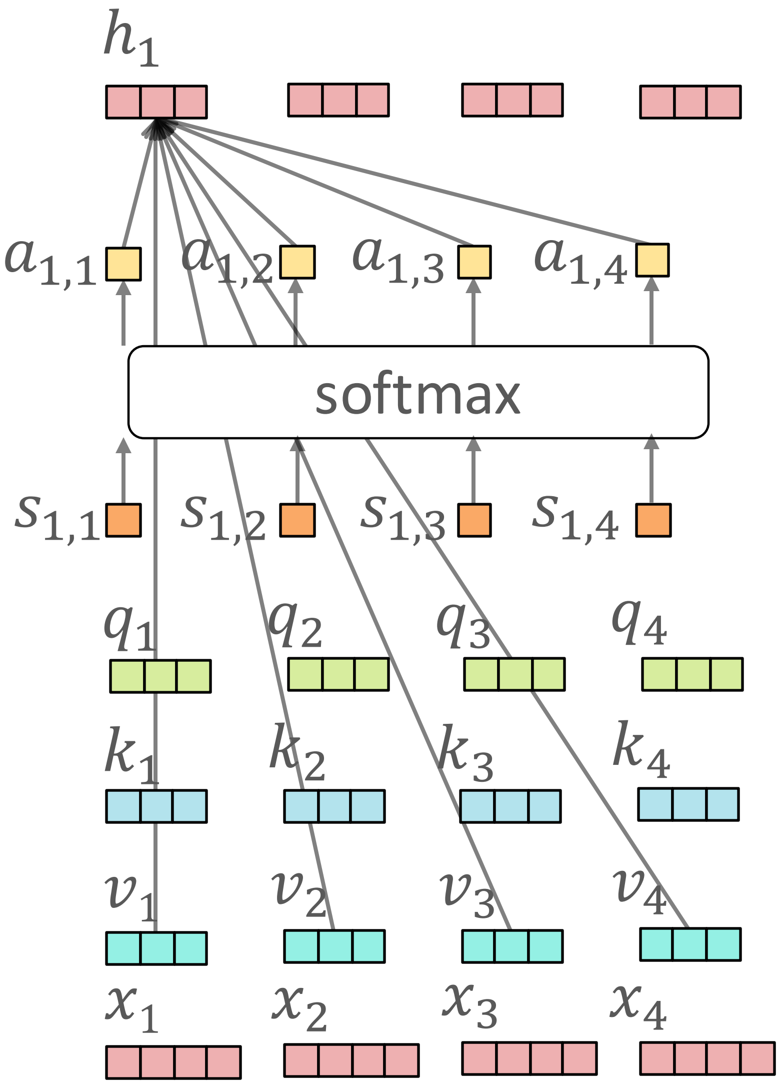
$$s_t(x', x_t) = \frac{k_t^T q}{\sqrt{\text{length}(q)}}$$

4. Those *scores* are turned into weights: $\text{softmax}(s(x', x_t))$
5. The model returns a weighted sum of the **values** as the contextualized representation:

new representation for x' $= \sum_t \text{softmax}(s(x', x_t))\, v(x_t)$

Attention Head

- Approach: compute a representation for each token in the *input sequence* by attending to all the input tokens



$$h_1 = \sum_{j=1}^{4} \text{softmax}(s_{1,j}) v_j$$

attention weights

scores: $s_{1,j} = \frac{k_j^T q_1}{\sqrt{\text{length}(k_j)}}$

queries: $q_t = W_Q x_t$

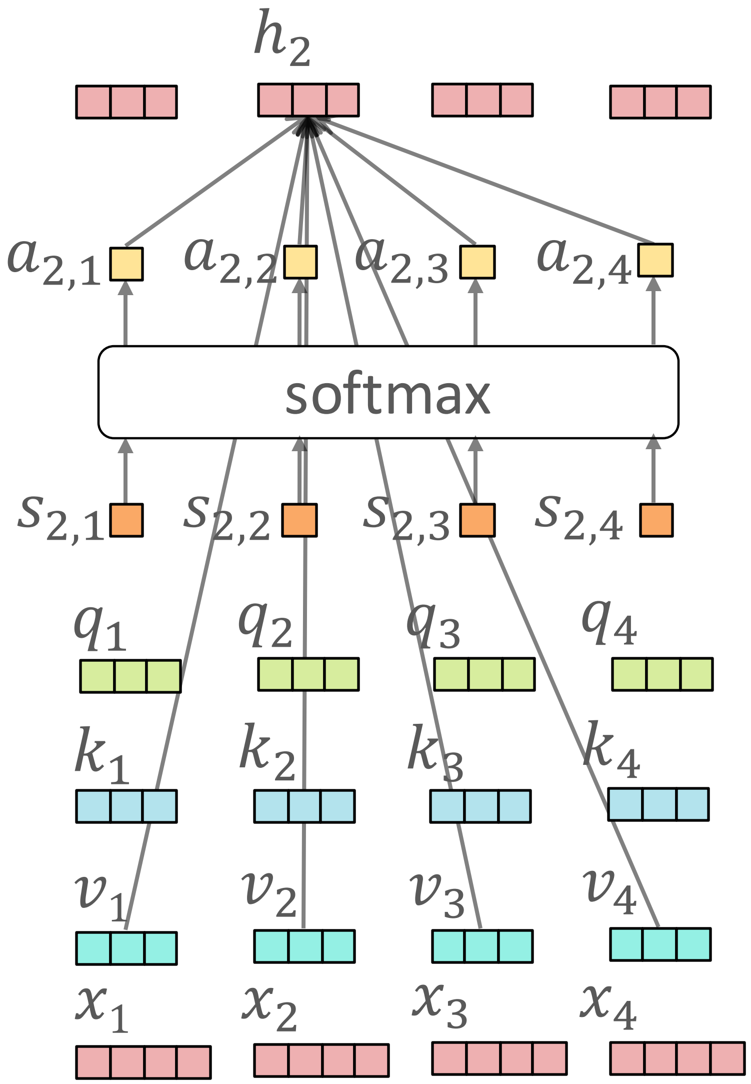
keys: $k_t = W_K x_t$

values: $v_t = W_V x_t$

input tokens

Attention Head

- Approach: compute a representation for each token in the *input sequence* by attending to all the input tokens



$$h_2 = \sum_{j=1}^{4} \text{softmax}(s_{2,j}) v_j$$

attention weights

scores: $s_{2,j} = \dfrac{k_j^T q_2}{\sqrt{\text{length}(k_j)}}$

queries: $q_t = W_Q x_t$

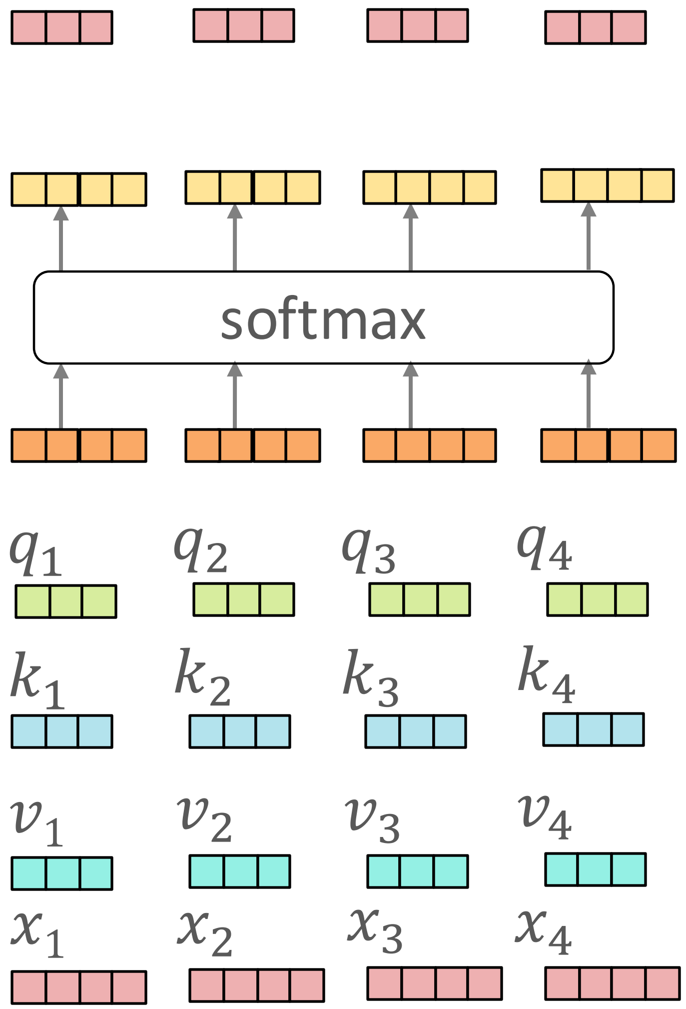
keys: $k_t = W_K x_t$

values: $v_t = W_V x_t$

input tokens

Attention Head: Matrix Form

- Approach: compute a representation for each token in the *input sequence* by attending to all the input tokens



$H = \text{softmax}(S)V \in \mathbb{R}^{N \times d_v}$

attention weights

N×dk dk×N

scores: $S = \frac{QK^T}{\sqrt{d_k}} \in \mathbb{R}^{N \times N}$

D×dk

queries: $Q = XW_Q \in \mathbb{R}^{N \times d_k}$

D×dk

keys: $K = XW_K \in \mathbb{R}^{N \times d_k}$

D×dv

values: $V = XW_V \in \mathbb{R}^{N \times d_v}$

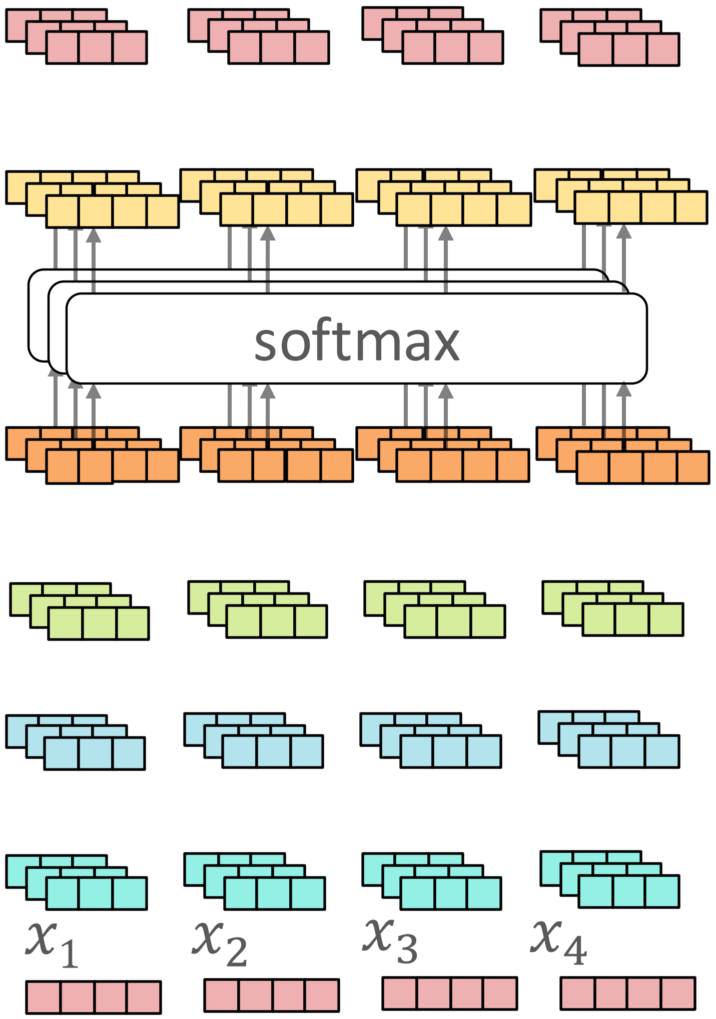
design matrix: $X \in \mathbb{R}^{N \times D}$

token

dim of embedding space

Multi-head Attention Layer

- Idea: just like we might want multiple convolutional filters in a convolutional layer, we might want multiple attention weights to learn different relationships between tokens!



$$H^{(h)} = \text{softmax}\left(S^{(h)}\right)V^{(h)}$$

attention weights

scores: $S^{(h)} = \frac{Q^{(h)} {K^{(h)}}^T}{\sqrt{d_k^{(h)}}}$

queries: $Q^{(h)} = XW_Q^{(h)}$

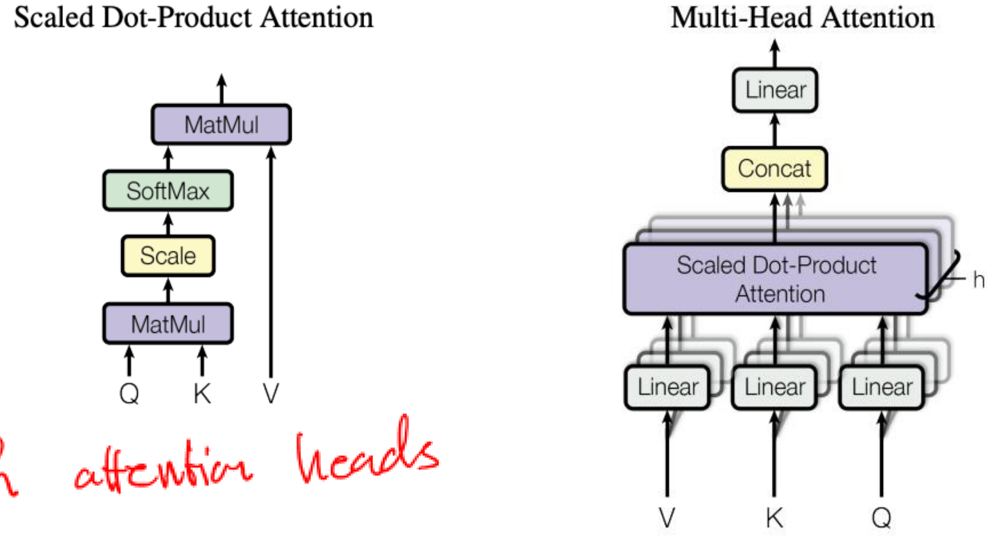
keys: $K^{(h)} = XW_K^{(h)}$

values: $V^{(h)} = XW_V^{(h)}$

design matrix: X

Multi-head Attention Layer

- Idea: just like we might want multiple convolutional filters in a convolutional layer, we might want multiple attention weights to learn different relationships between tokens!



- Suppose h attention heads

- The outputs from all the attention heads are concatenated together to get the final representation

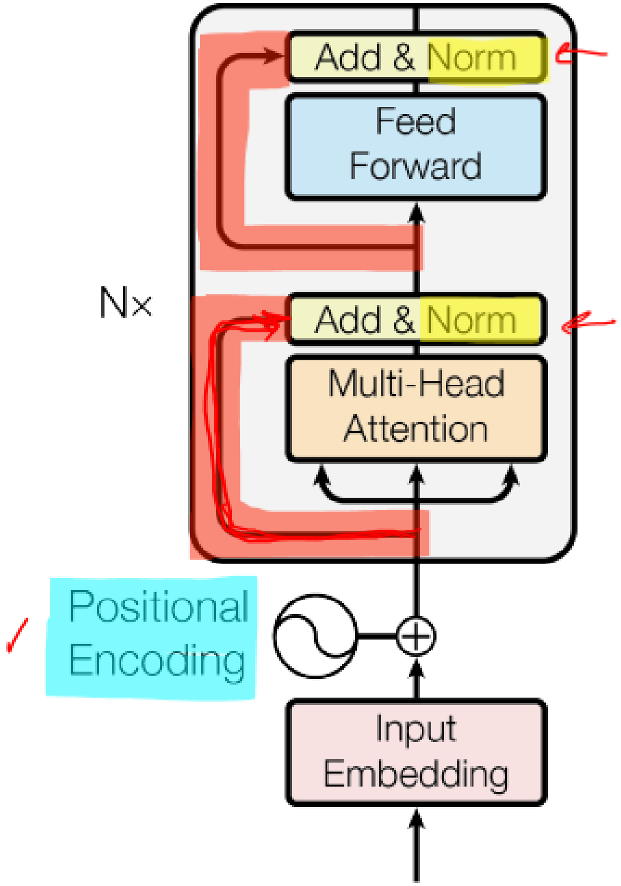
$$H = [\underbrace{H^{(1)}}_{d_v}, \underbrace{H^{(2)}}_{d_v}, \dots, \underbrace{H^{(h)}}_{d_v}]$$

$d_v \times h = D$

- Common architectural choice: $\underbrace{d_v = D/h}$ $\rightarrow |H| = D$

Source: https://arxiv.org/pdf/1706.03762.pdf

Transformers



- In addition to multi-head attention, transformer architectures use
  1. Positional encodings
  2. Layer normalization
  3. Residual connections
  4. A fully-connected feed-forward network

Source: https://arxiv.org/pdf/1706.03762.pdf

Positional Encodings

- Issue: if all tokens attend to every token in the sequence, then how does the model infer the order of tokens?
- Idea: add a position-specific embedding p_t to the token embedding x_t

$$x_t' = x_t + p_t$$

- Positional encodings can be
  - *fixed* i.e., some predetermined function of t or *learned* alongside the token embeddings
  - *absolute* i.e., only dependent on the token's location in the sequence or *relative* to the query token's location

Layer Normalization

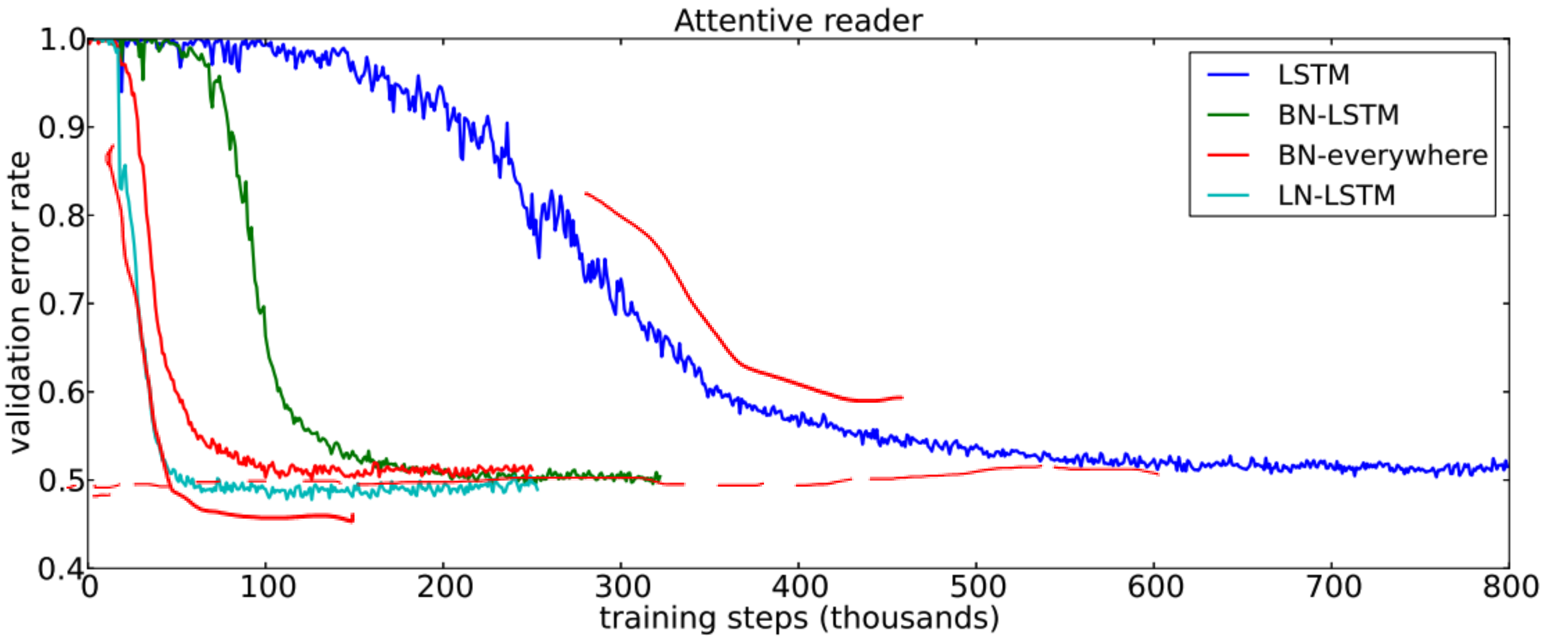
- Issue: for certain activation functions, the weights in later layers are **highly sensitive** to changes in the earlier layers
  - Small changes to weights in early layers are amplified so weights in deeper layers have to deal with massive dynamic ranges → slow optimization convergence

- Idea: normalize the output of a layer to always have the same (learnable) mean, β, and variance, γ^2

$$H' = \gamma \left(\frac{H - \mu}{\sigma} \right) + \beta$$

where μ is the mean and σ is the standard deviation of the values in the vector H

Layer Normalization



- Idea: normalize the output of a layer to always have the same (learnable) mean, β, and variance, γ^2

$$H' = \gamma\left(\frac{H-\mu}{\sigma}\right) + \beta$$

where μ is the mean and σ is the standard deviation of the values in the vector H

Source: https://arxiv.org/pdf/1607.06450.pdf

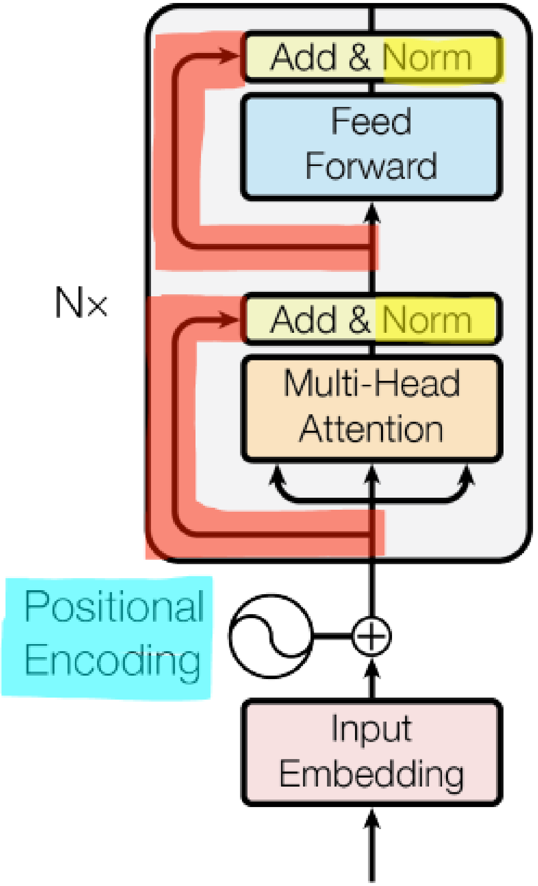
Residual Connections

- Observation: early deep neural networks suffered from the "degradation" problem where adding more layers actually made performance worse!
- Idea: add the input embedding back to the output of a layer

$$H' = \underbrace{\boxed{H\left(x^{(i)}\right)} + x^{(i)}}$$

- Suppose the target function is f
  - Now instead of having to learn $f\left(x^{(i)}\right)$, the hidden layer just needs to learn the residual $r = f\left(x^{(i)}\right) - x^{(i)}$
  - If f is the identity function, then the hidden layer just needs to learn $r = 0$, which is easy for a neural network!

How on earth do we train these things?



- In addition to multi-head attention, transformer architectures use
    1. Positional encodings
    2. Layer normalization
    3. Residual connections
    4. A fully-connected feed-forward network

Source: https://arxiv.org/pdf/1706.03762.pdf

Learning Paradigms

- Supervised learning - $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{N}$
  - Regression - $y^{(i)} \in \mathbb{R}$
  - Classification - $y^{(i)} \in \{1, \dots, C\}$
- Unsupervised learning - $\mathcal{D} = \left\{\boldsymbol{x}^{(i)}\right\}_{i=1}^{N}$
  - Clustering
  - Dimensionality reduction
- Reinforcement learning
- Active learning
- Semi-supervised learning
- Online learning

Learning Paradigms

- Supervised learning - $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{N}$
  - Regression - $y^{(i)} \in \mathbb{R}$
  - Classification - $y^{(i)} \in \{1, \ldots, C\}$
- Unsupervised learning - $\mathcal{D} = \left\{\boldsymbol{x}^{(i)}\right\}_{i=1}^{N}$
  - **Clustering**
  - Dimensionality reduction
- Reinforcement learning
- Active learning
- Semi-supervised learning
- Online learning

$D = \{x^{(i)}\}_{i=1}^{N}$

Clustering

- Goal: split an unlabeled data set into groups or clusters of "similar" data points
- Use cases:
  - Organizing data
  - Discovering patterns or structure
  - Preprocessing for downstream machine learning tasks
- Applications:

Recall: Similarity for kNN

- Classify a point as the label of the "most similar" training point
- **Idea: given real-valued features, we can use a distance metric to determine how similar two data points are**
- A common choice is Euclidean distance:

$$d(\boldsymbol{x}, \boldsymbol{x}') = \|\boldsymbol{x} - \boldsymbol{x}'\|_2 = \sqrt{\sum_{d=1}^{D} (x_d - x_d')^2}$$

- An alternative is the Manhattan distance:

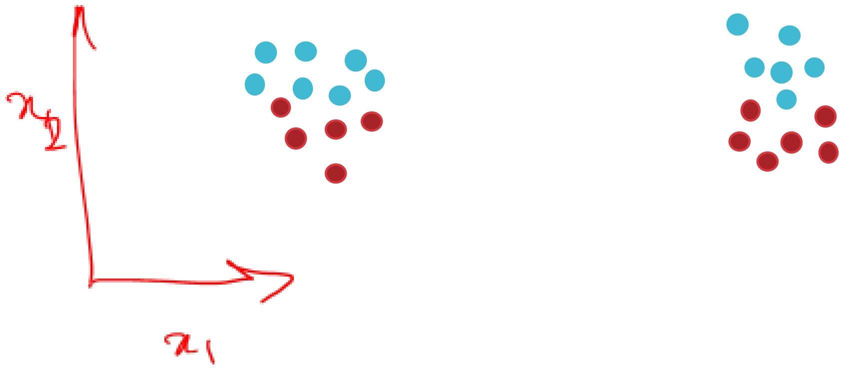
$$d(\boldsymbol{x}, \boldsymbol{x}') = \|\boldsymbol{x} - \boldsymbol{x}'\|_1 = \sum_{d=1}^{D} |x_d - x_d'|$$

Partition-Based Clustering

- Given a desired number of clusters, K, return a partition of the data set into K groups or clusters, $\{C_1, \ldots, C_K\}$, that optimize some objective function

1. What objective function should we optimize?

2. How can we perform optimization in this setting?



Option A

Option B

Which partition is best?

General Recipe for Machine Learning

- Define a model and model parameters
- Write down an objective function
- Optimize the objective w.r.t. the model parameters

Recipe for K-means

- Define a model and model parameters $\mathcal{D} = \{x^{(i)}\}_{i=1}^{N}$
    - Assume K clusters and use the Euclidean distance
    - Parameters: $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ and $z^{(1)}, \ldots, z^{(N)}$ cluster assigned to $x^{(i)}$

center of cluster 1, ..., K

- Write down an objective function

$$\rightarrow \sum_{i=1}^{N} \left\| \boldsymbol{x}^{(i)} - \boldsymbol{\mu}_{z^{(i)}} \right\|_2$$

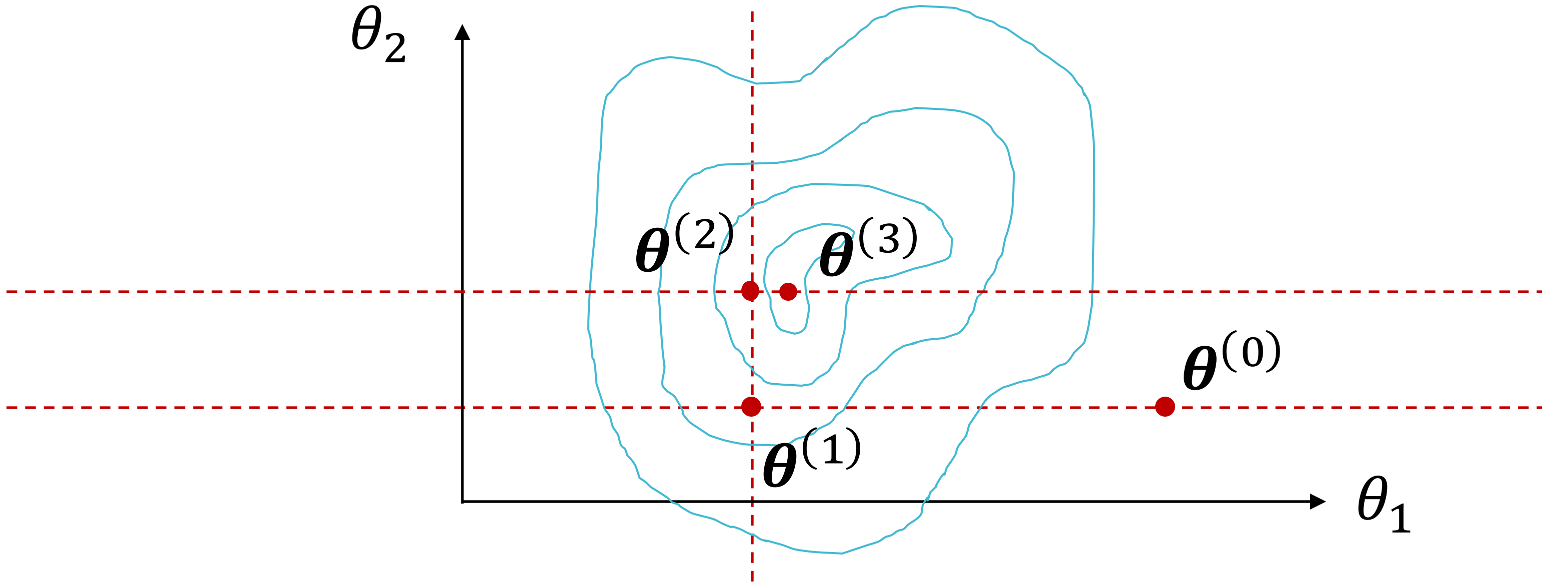
- Optimize the objective w.r.t. the model parameters
    - Use (block) coordinate descent

Coordinate Descent

- Goal: minimize some objective (θ_1, θ_2)

$$\hat{\boldsymbol{\theta}} = \operatorname{argmin} J(\boldsymbol{\theta})$$

- Idea: iteratively pick one variable and minimize the objective w.r.t. just that variable, *keeping all others fixed*.



Block Coordinate Descent

- Goal: minimize some objective

$$\hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}} = \operatorname{argmin} \ J(\boldsymbol{\alpha}, \boldsymbol{\beta})$$

- Idea: iteratively pick one *block* of variables ($\boldsymbol{\alpha}$ or $\boldsymbol{\beta}$) and minimize the objective w.r.t. that block, keeping the other(s) fixed.
  - Ideally, blocks should be the largest possible set of variables that can be efficiently optimized simultaneously

Optimizing the K-means objective

$$\hat{\boldsymbol{\mu}}_1, \dots, \hat{\boldsymbol{\mu}}_K, \hat{z}^{(1)}, \dots, \hat{z}^{(N)} = \underset{\mu, z}{\operatorname{argmin}} \sum_{i=1}^{N} \left\| \boldsymbol{x}^{(i)} - \boldsymbol{\mu}_{z^{(i)}} \right\|_2$$

- If $\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K$ are fixed

$$\hat{z}^{(i)} = \underset{k \in \{1, \dots, K\}}{\operatorname{argmin}} \left\| \boldsymbol{x}^{(i)} - \boldsymbol{\mu}_k \right\|_2$$

- If $z^{(1)}, \dots, z^{(N)}$ are fixed

$$\hat{\boldsymbol{\mu}}_k = \underset{\boldsymbol{\mu}}{\operatorname{argmin}} \sum_{i \,: z^{(i)} = k} \left\| \boldsymbol{x}^{(i)} - \boldsymbol{\mu} \right\|_2$$

$$= \frac{1}{N_k} \sum_{i \,: z^{(i)} = k} \boldsymbol{x}^{(i)}$$

K-means Algorithm

- Input: $\mathcal{D} = \left\{\left(\boldsymbol{x}^{(i)}\right)\right\}_{i=1}^{N}, K$

1. Initialize cluster centers $\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K$
2. While NOT CONVERGED
   a. Assign each data point to the cluster with the nearest cluster center:
   $$z^{(i)} = \underset{k}{\operatorname{argmin}} \left\|\boldsymbol{x}^{(i)} - \boldsymbol{\mu}_k\right\|_2$$
   b. Recompute the cluster centers:
   $$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i\,: z^{(i)}=k} \boldsymbol{x}^{(i)}$$
   where N_k is the number of data points in cluster k

- Output: cluster assignments $z^{(1)}, \dots, z^{(N)}$

K-means: Example ($K = 3$)

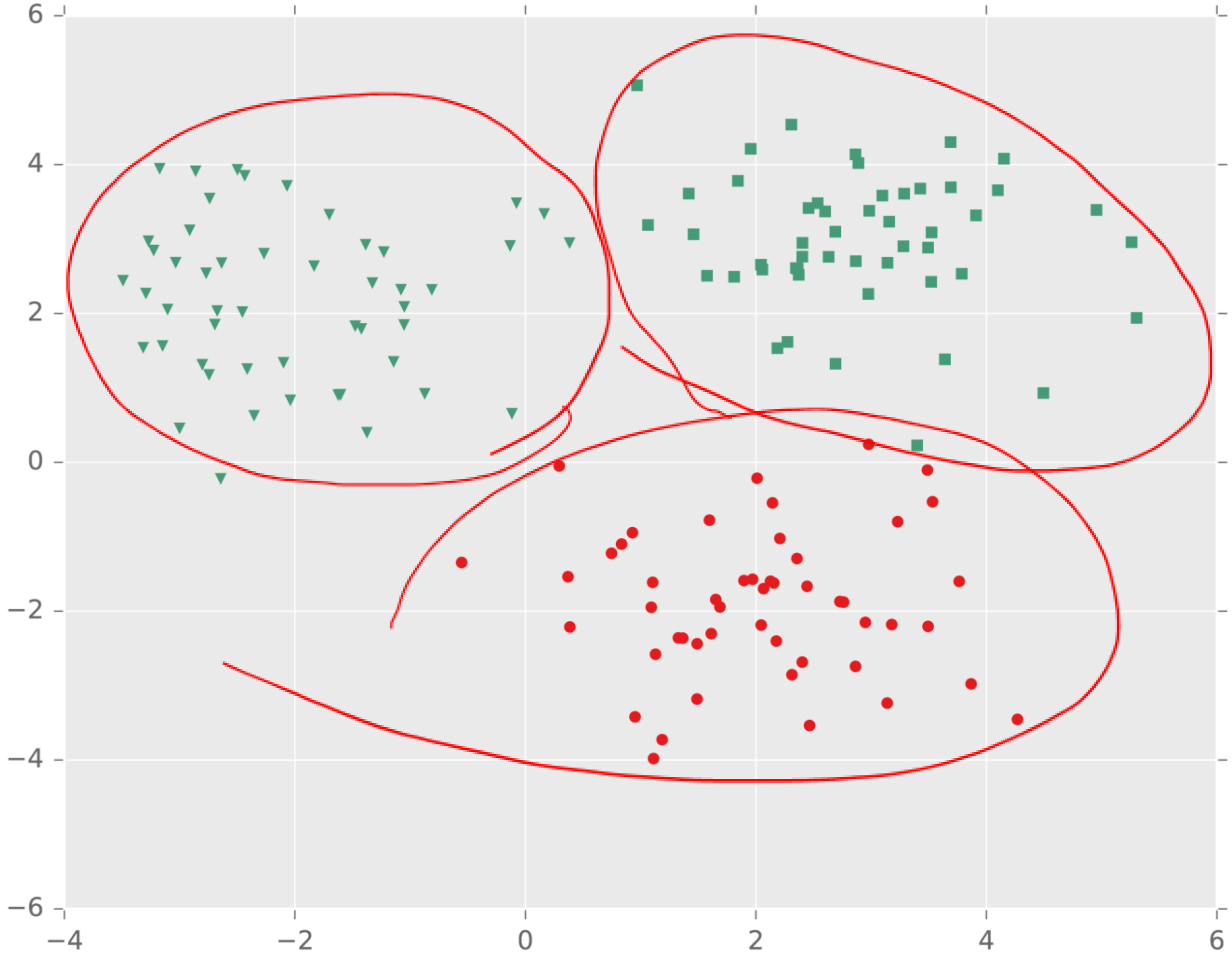


Figure courtesy of Matt Gormley

K-means: Example ($K = 3$)

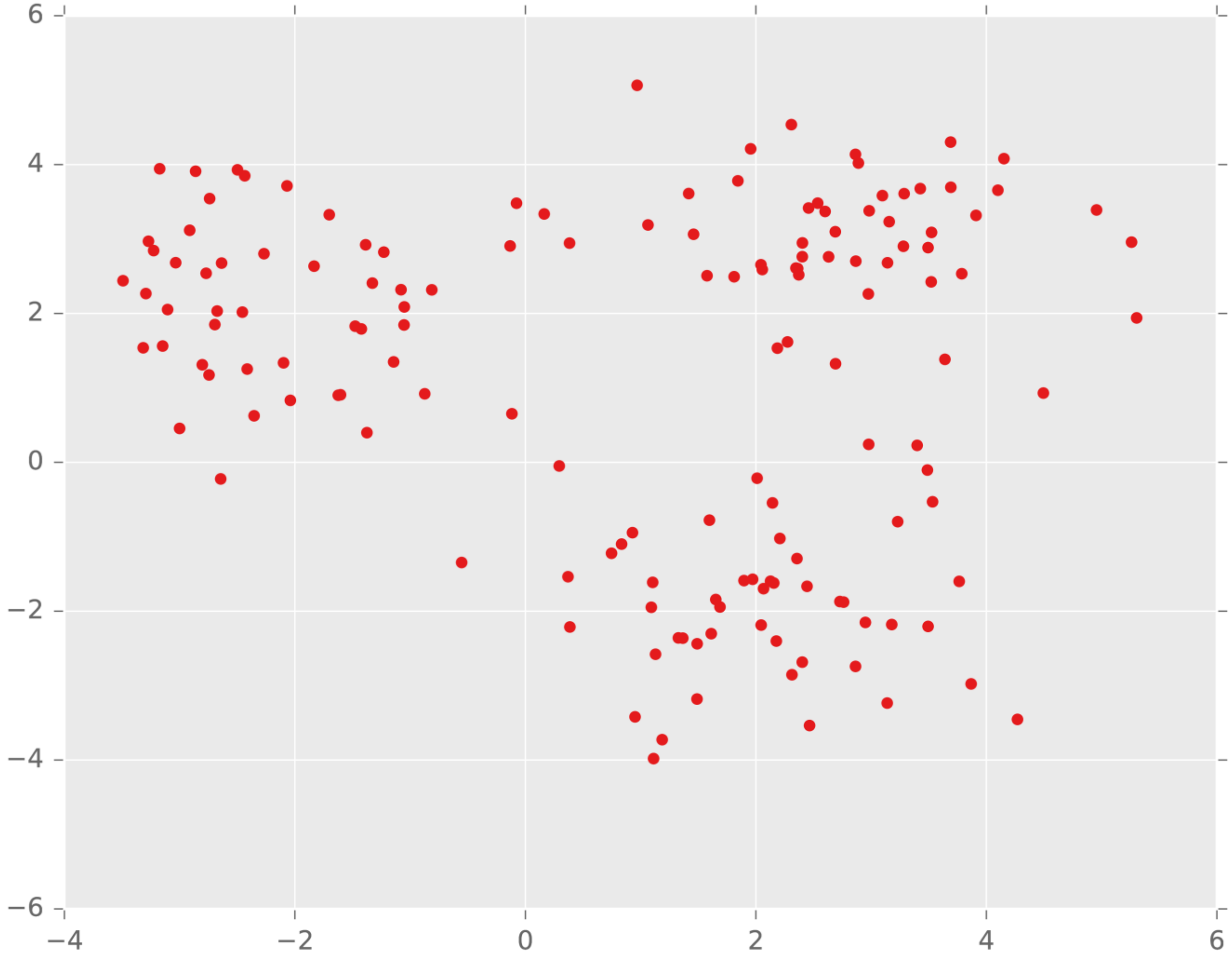


Figure courtesy of Matt Gormley

K-means: Example ($K = 3$)

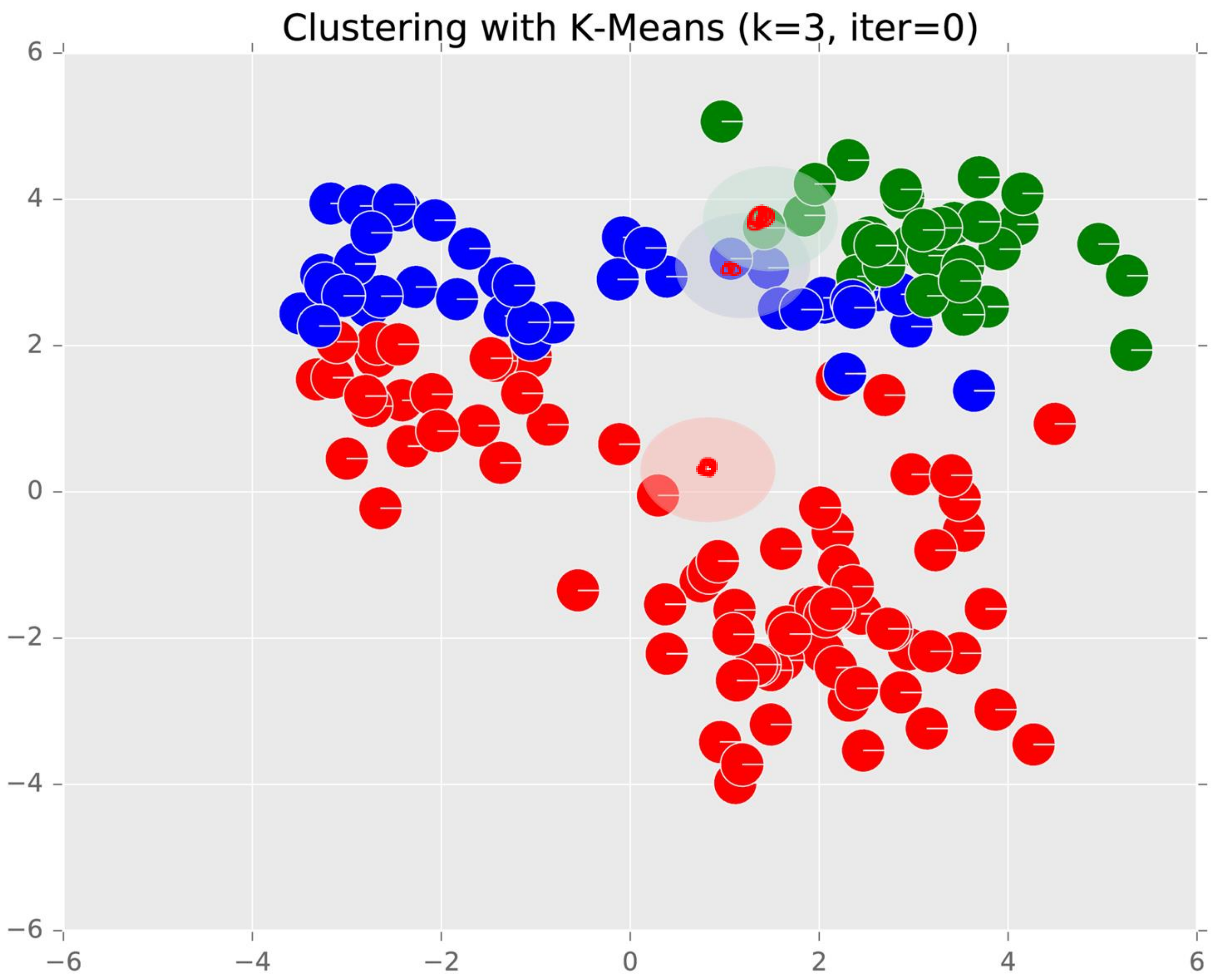


Figure courtesy of Matt Gormley

K-means: Example ($K = 3$)

Figure courtesy of Matt Gormley

K-means: Example ($K = 3$)

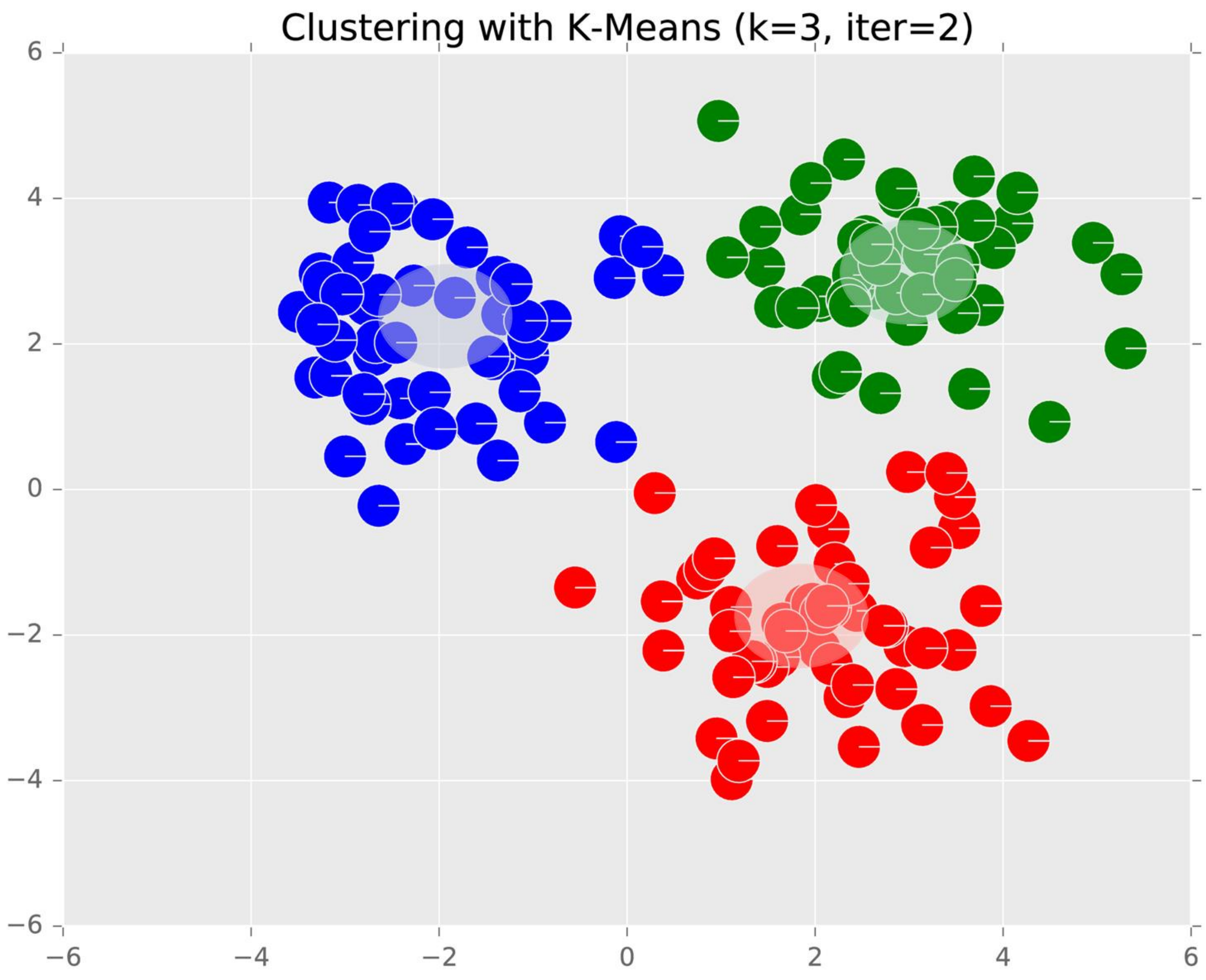


Figure courtesy of Matt Gormley

K-means: Example ($K = 3$)

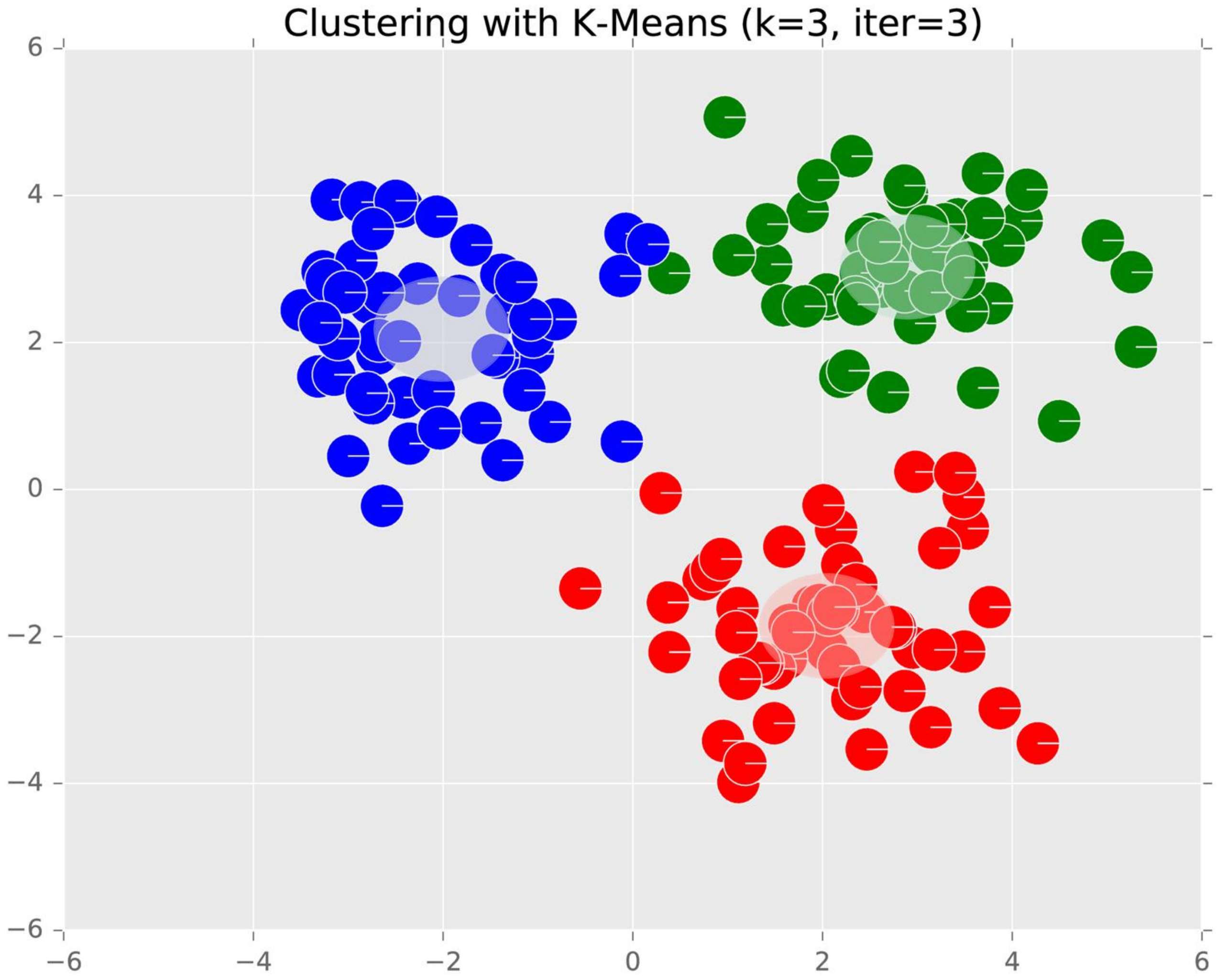


Figure courtesy of Matt Gormley

K-means: Example ($K = 3$)

Figure courtesy of Matt Gormley

K-means: Example ($K = 3$)

Figure courtesy of Matt Gormley

K-means: Example ($K = 2$)

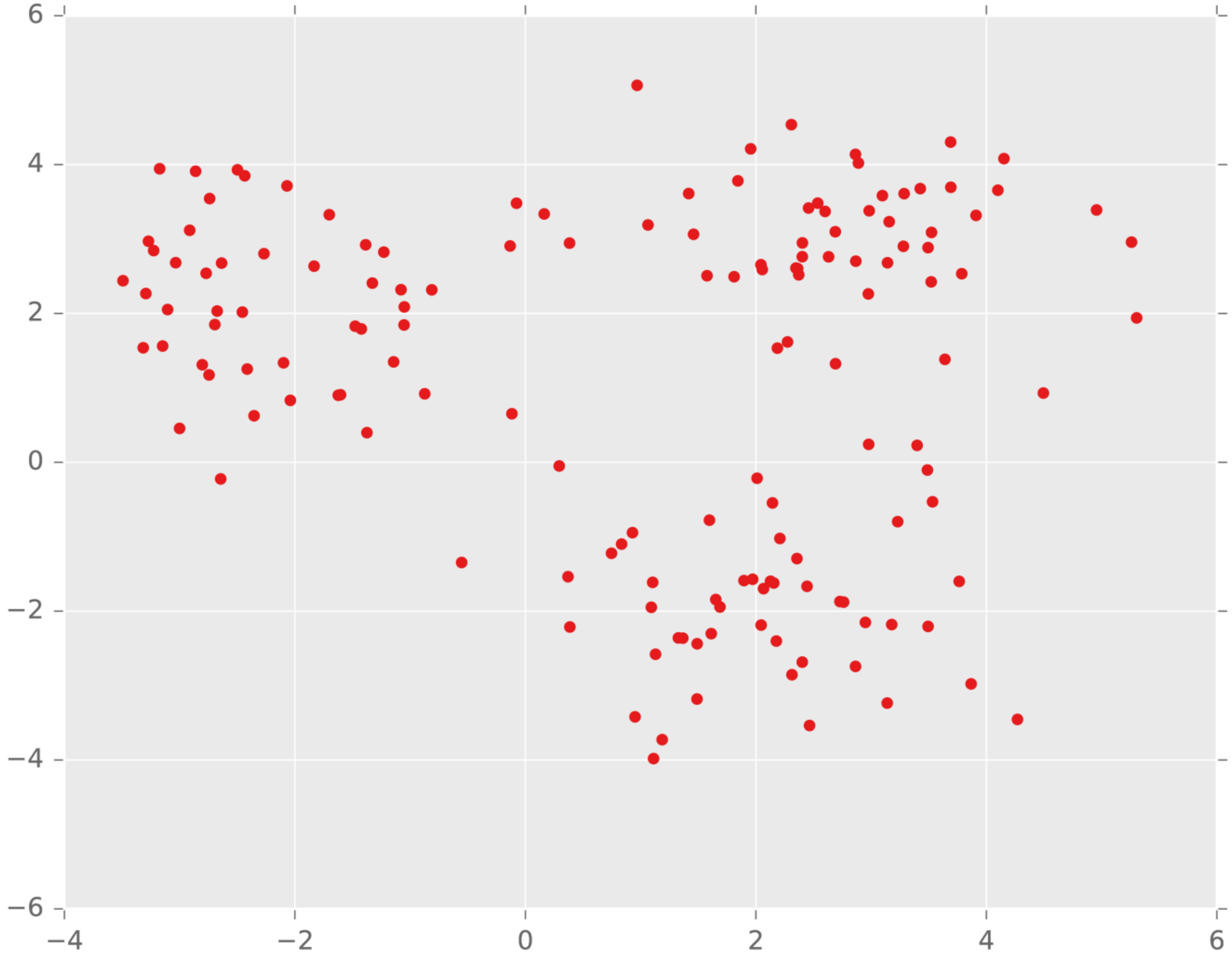


Figure courtesy of Matt Gormley

K-means: Example ($K = 2$)

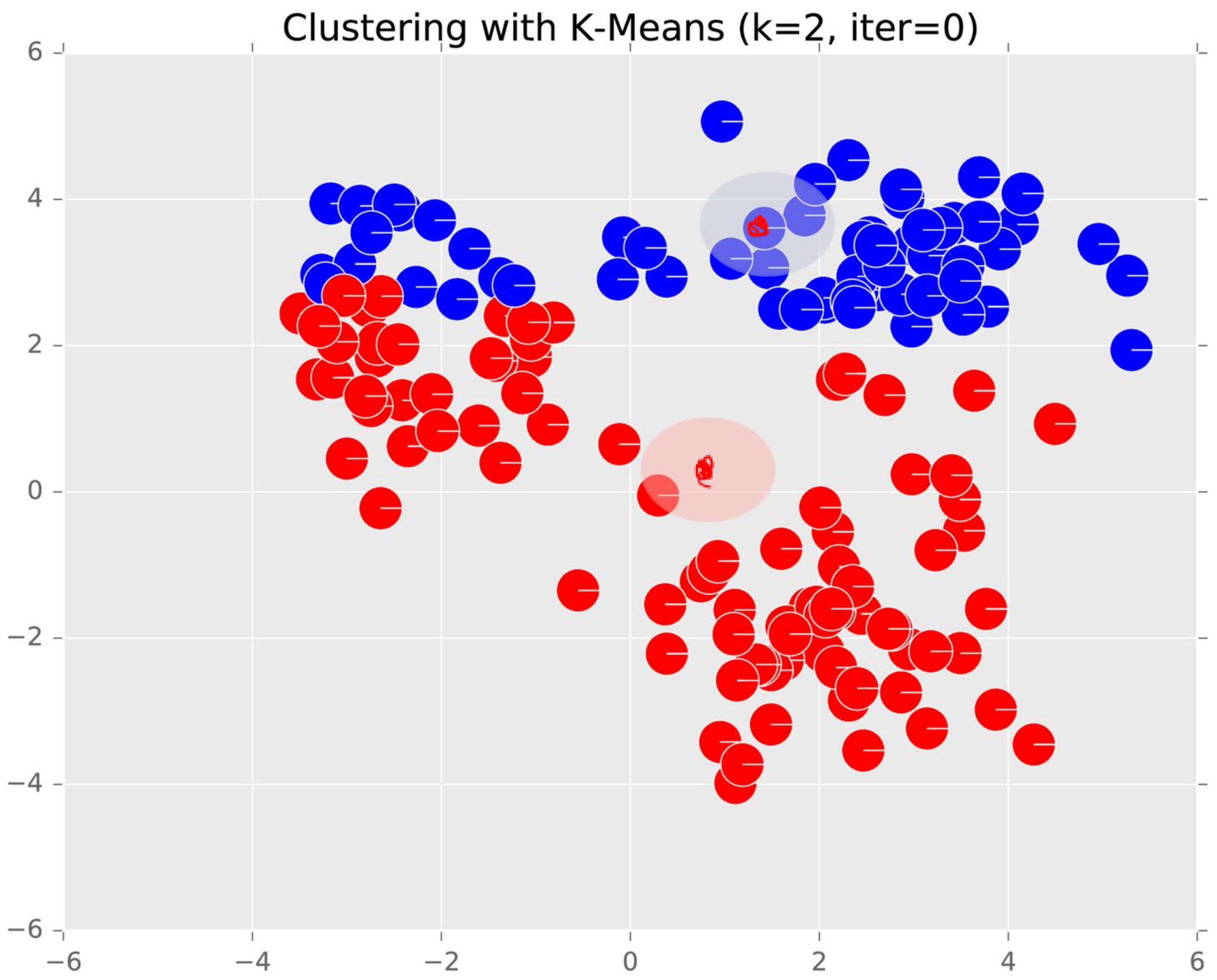


Figure courtesy of Matt Gormley

K-means: Example ($K = 2$)

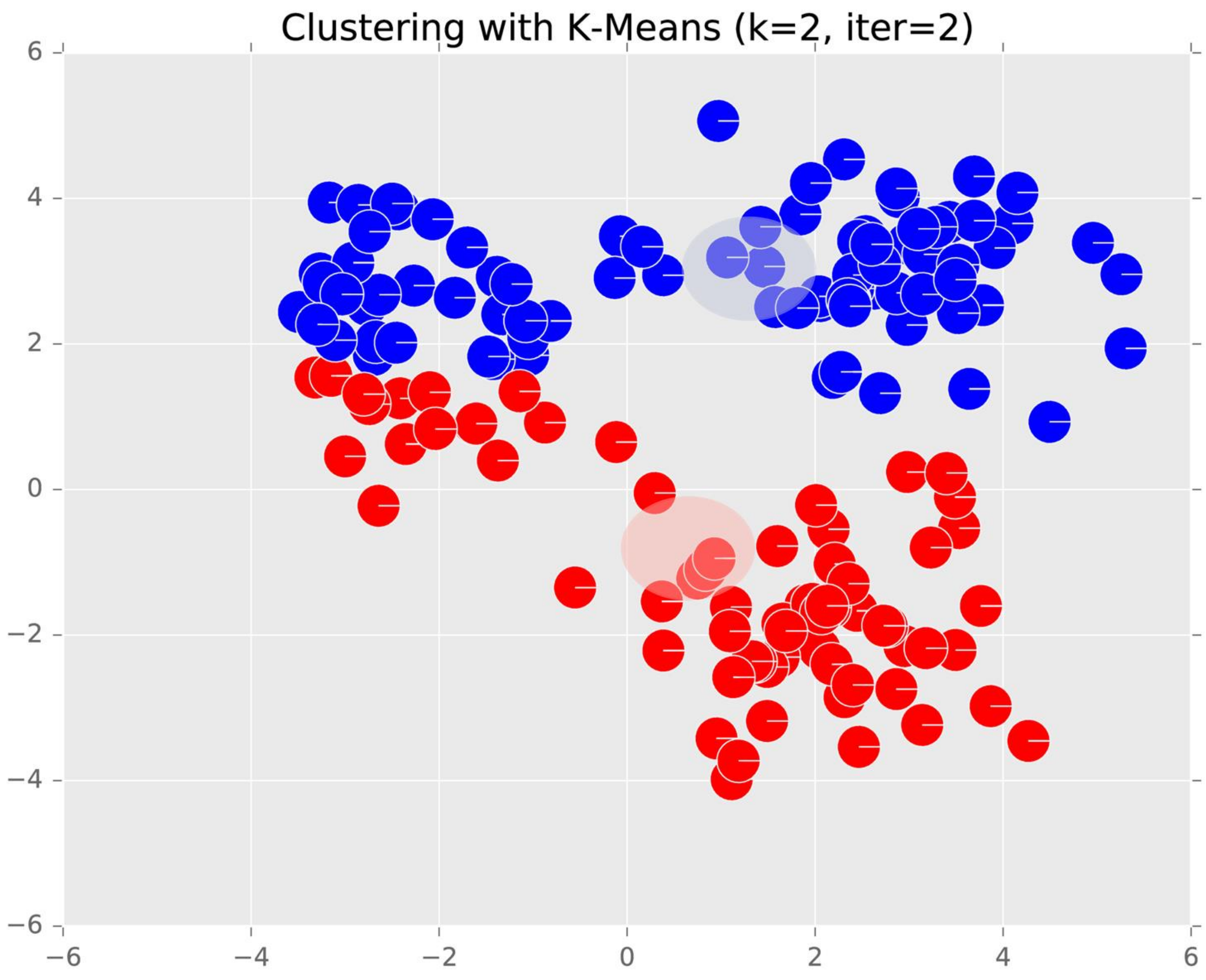


Figure courtesy of Matt Gormley

K-means: Example ($K = 2$)

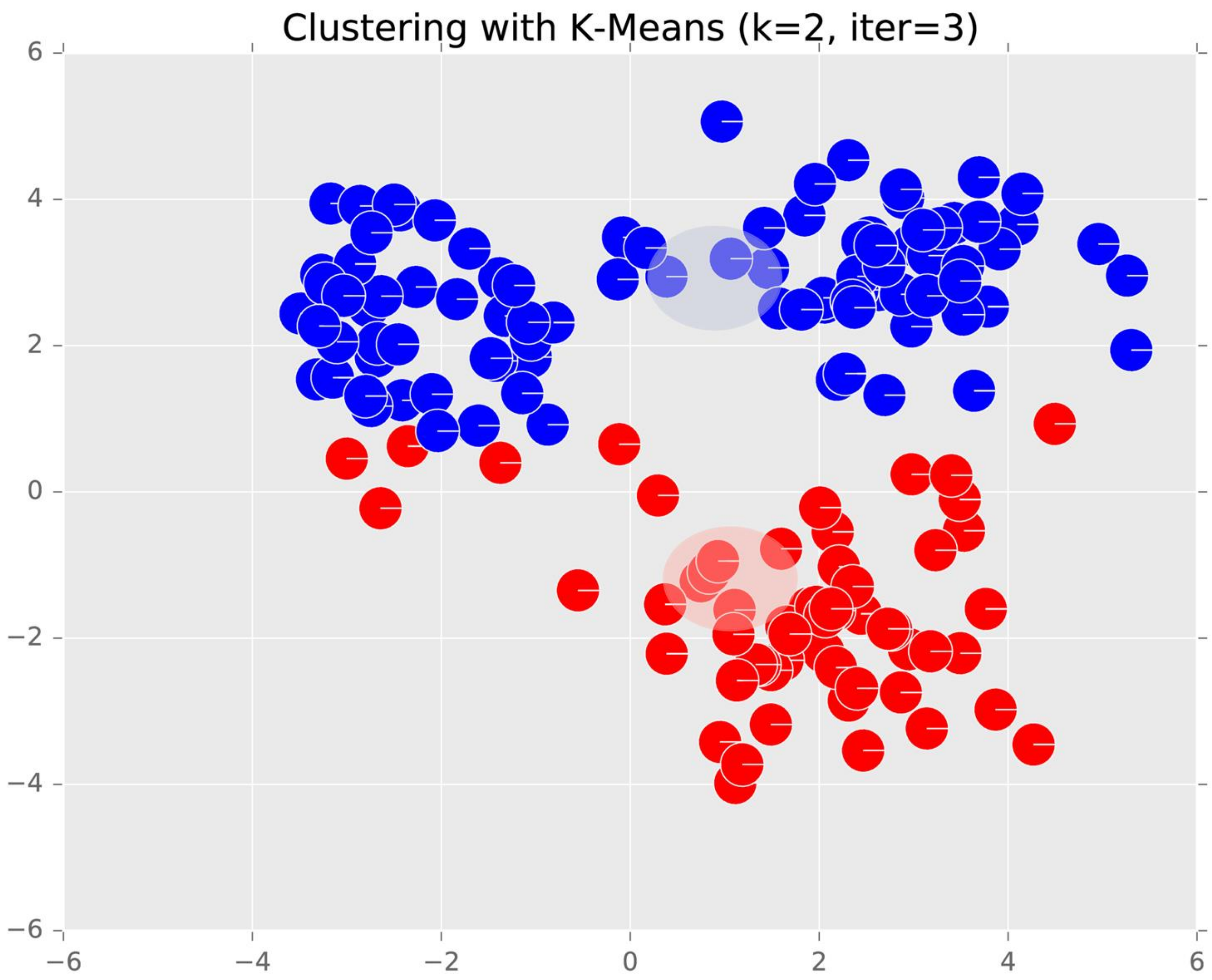


Figure courtesy of Matt Gormley

K-means: Example ($K = 2$)

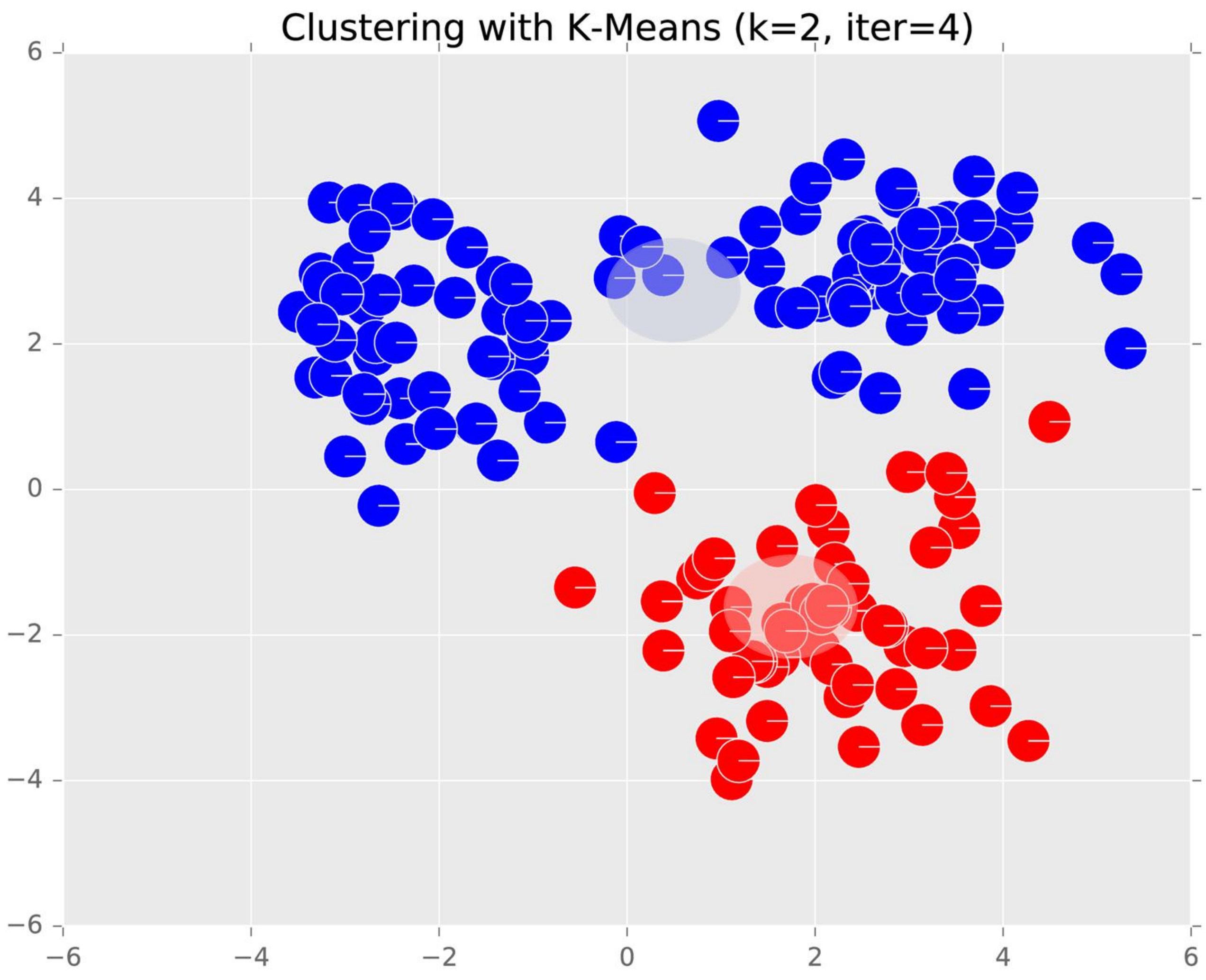


Figure courtesy of Matt Gormley

K-means: Example ($K = 2$)

Figure courtesy of Matt Gormley

K-means: Example ($K = 2$)

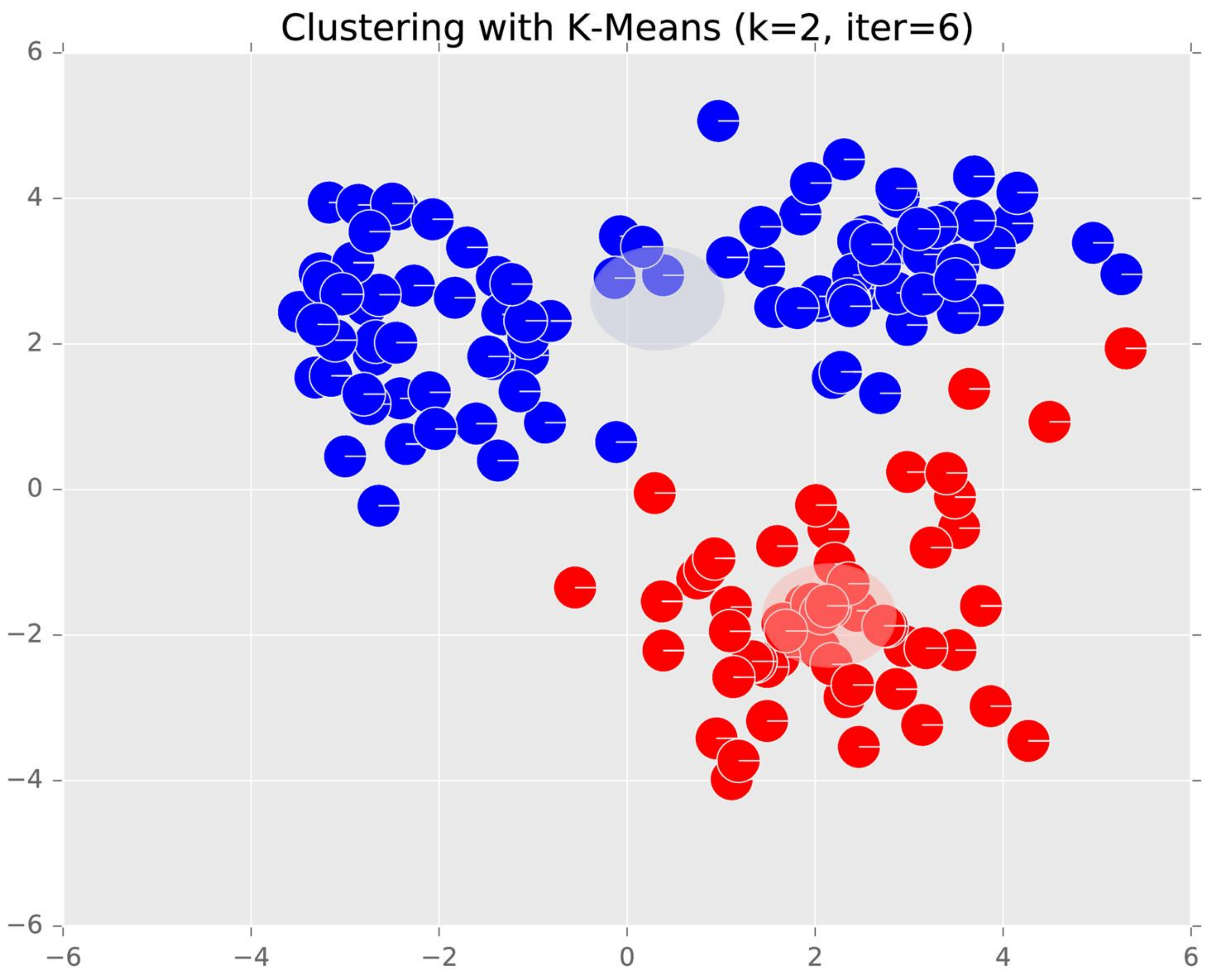


Figure courtesy of Matt Gormley

K-means: Example ($K = 2$)

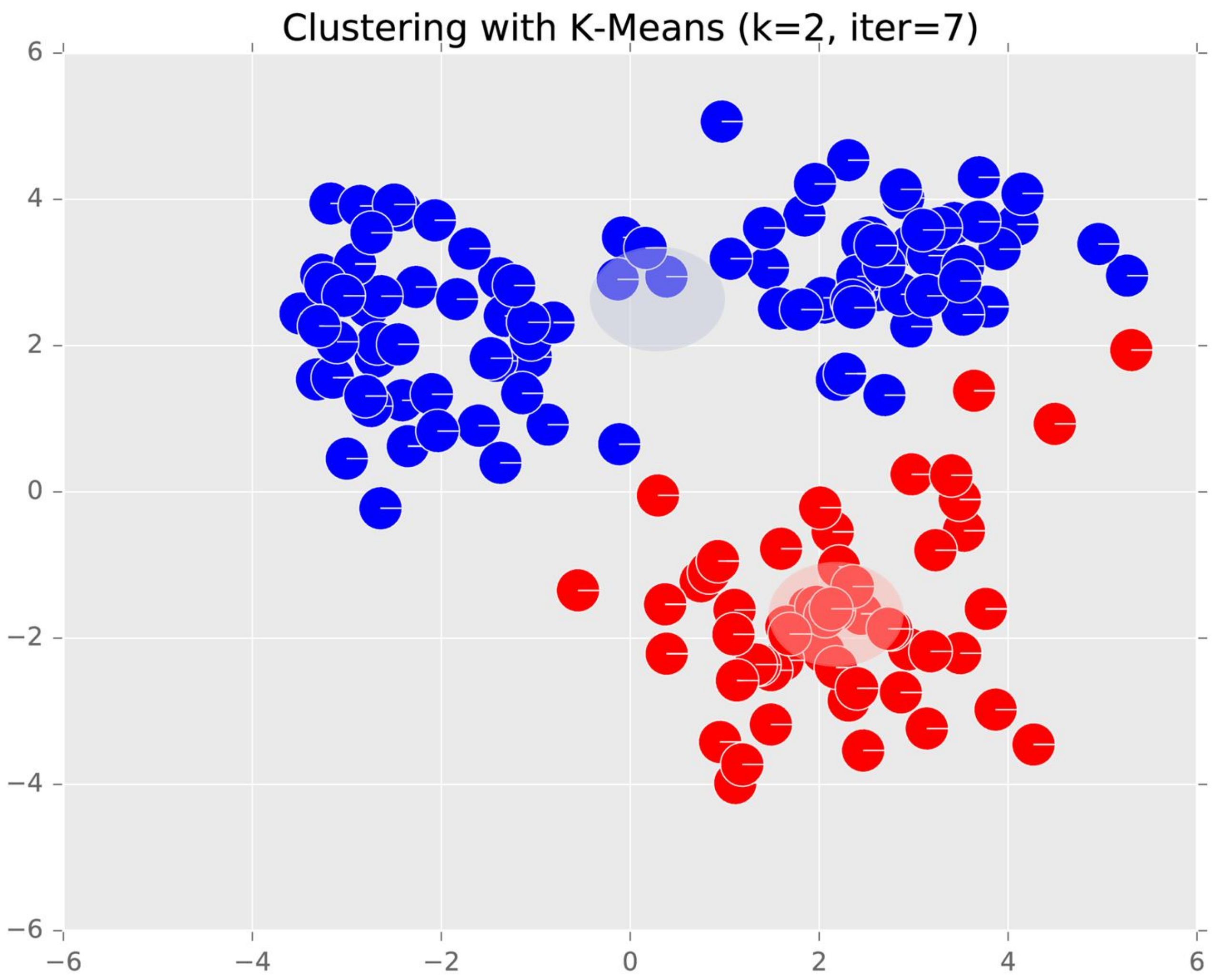


Figure courtesy of Matt Gormley

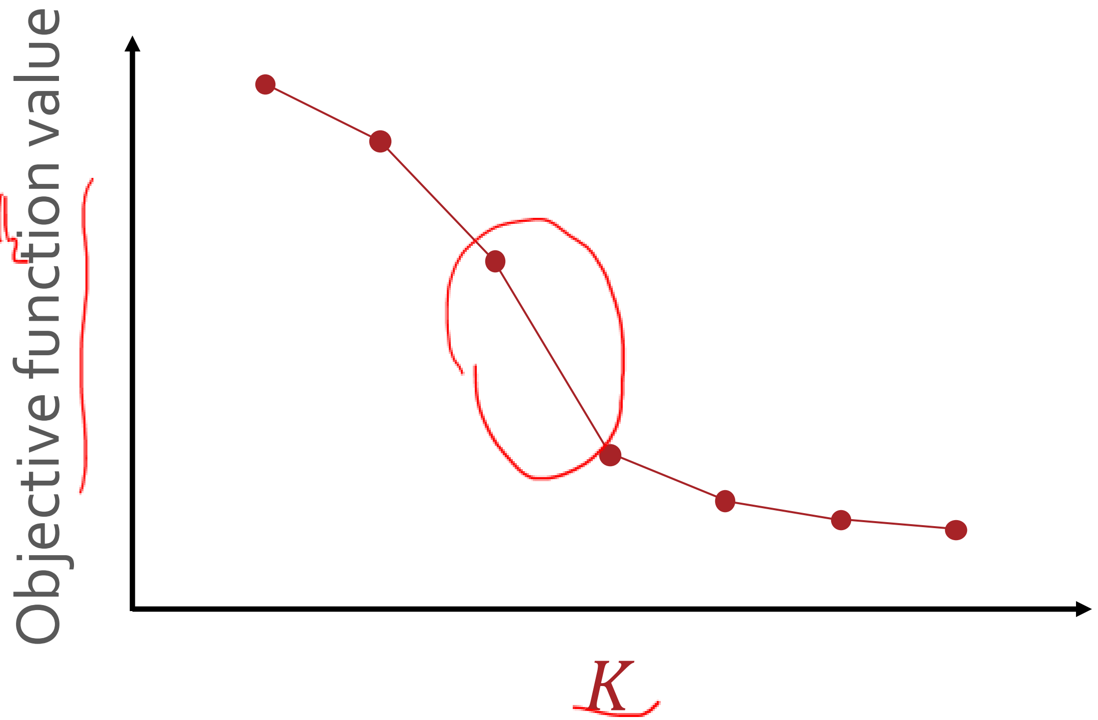
Setting K

- Idea: choose the value of K that minimizes the objective function

Setting K

- Idea: choose the value of K that minimizes the objective function

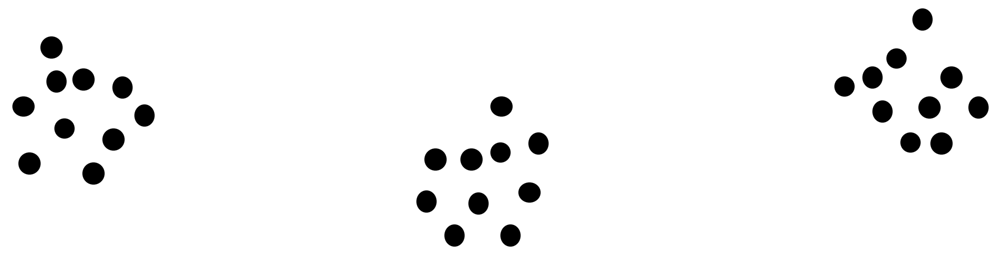
$\sum_i \| x^{(i)} - \mu_{z^{(i)}} \|_2$



- Better idea: look for the characteristic "elbow" or largest decrease when going from $K - 1$ to K

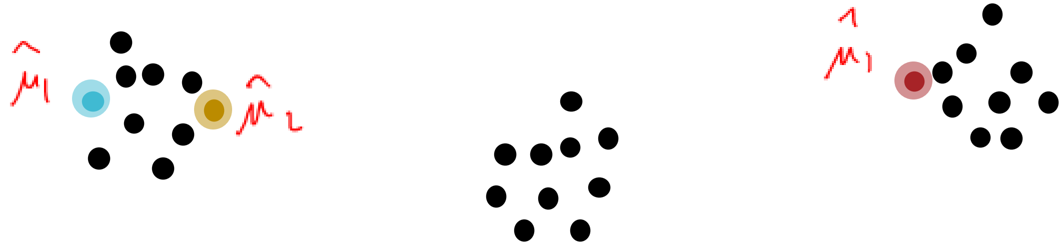
Initializing K-means

- Common choice: choose K data points at random to be the initial cluster centers (Lloyd's method)



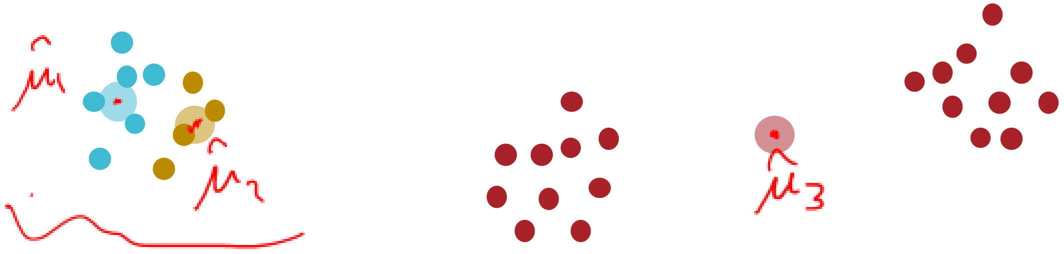
Initializing K-means

- Common choice: choose K data points at random to be the initial cluster centers (Lloyd's method)



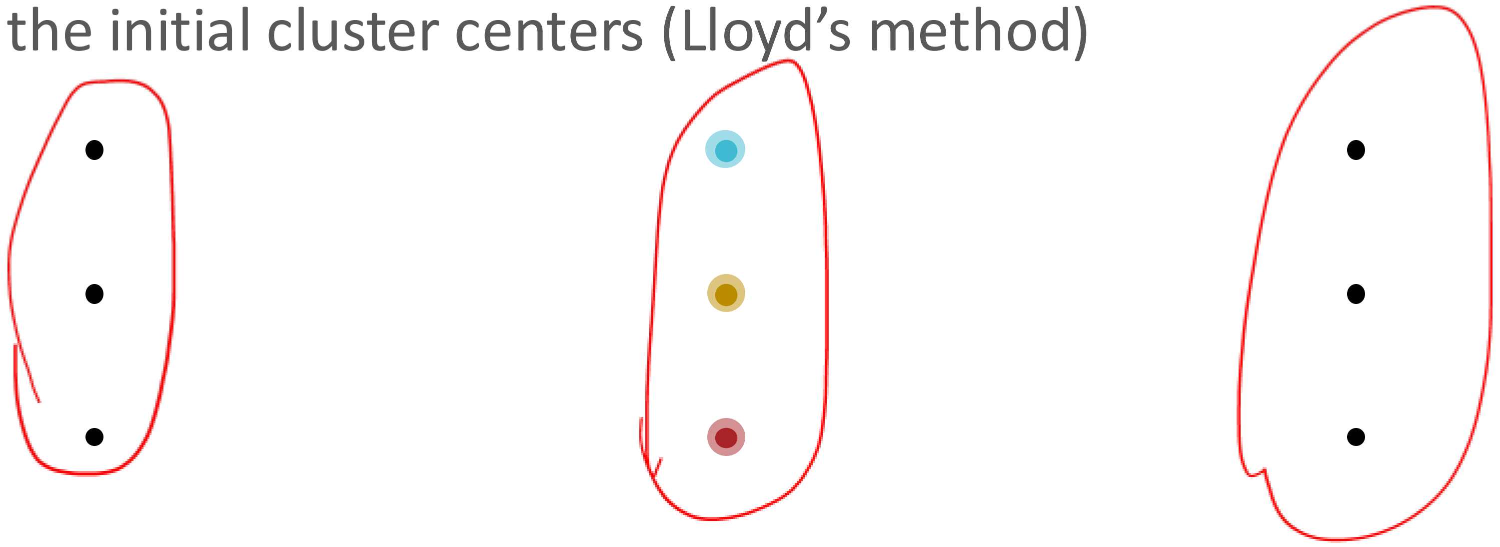
Initializing K-means

- Common choice: choose K data points at random to be the initial cluster centers (Lloyd's method)



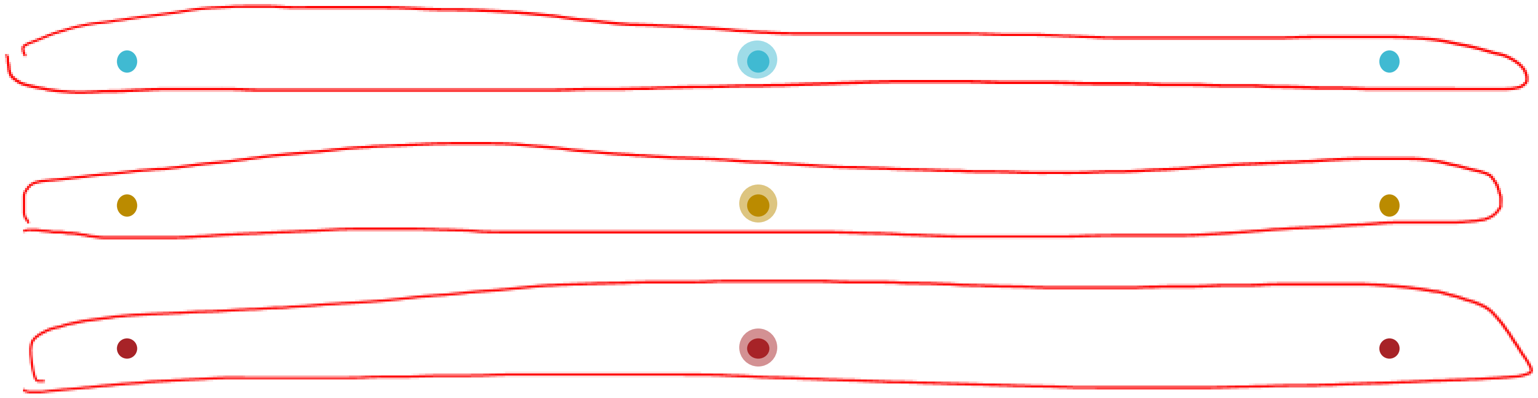
Initializing K-means

- Common choice: choose K data points at random to be the initial cluster centers (Lloyd's method)



Initializing K-means

- Common choice: choose K data points at random to be the initial cluster centers (Lloyd's method)



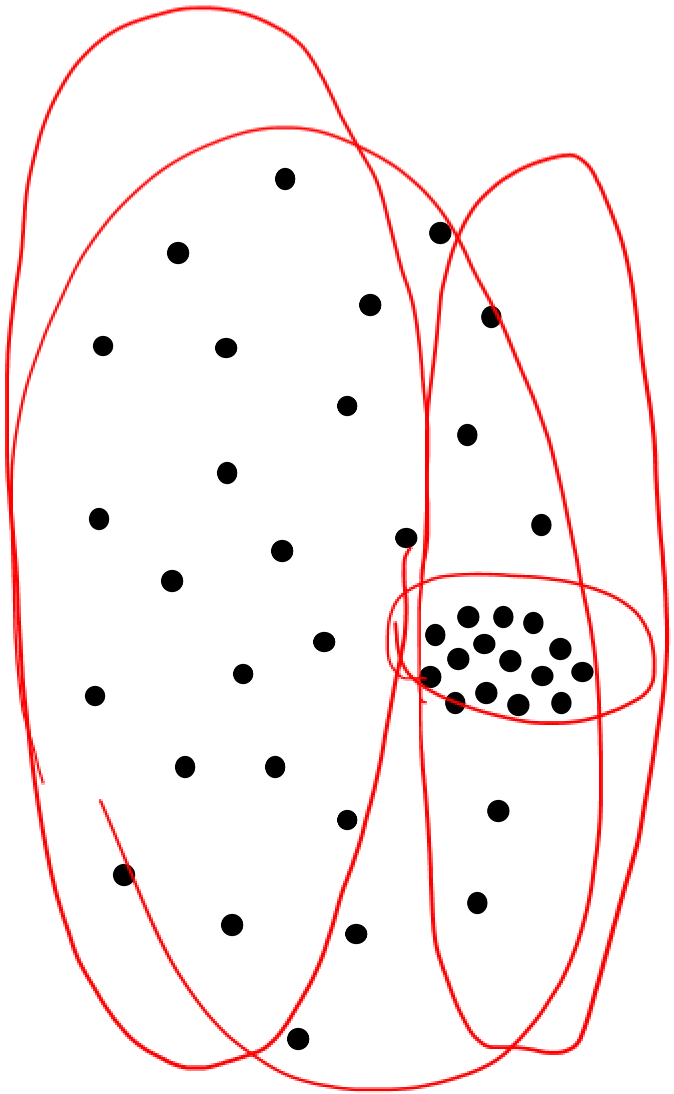
- Lloyd's method converges to a local minimum and that local minimum can be arbitrarily bad (relative to the optimal clusters)
- Intuition: want initial cluster centers to be far apart from one another

K-means++ (Arthur and Vassilvitskii, 2007)

1. Choose the first cluster center randomly from the data points.
2. For each other data point $\boldsymbol{x}$, compute $D(\boldsymbol{x})$, the distance between $\boldsymbol{x}$ and the closest cluster center.
3. Select the next cluster center proportional to $D(\boldsymbol{x})^2$.
4. Repeat 2 and 3 $K - 1$ times.

- K-means++ achieves a $O(\log K)$ approximation to the optimal clustering in expectation
- Both Lloyd's method and K-means++ can benefit from multiple random restarts.

Shortcomings of K-means



- Clusters cannot overlap
- Clusters must all be of the same “width”
- Clusters must be linearly separable

Probabilistic or “Soft” Assignments

- Instead of $z^{(i)}$ being a deterministic scalar, let $\boldsymbol{z}^{(i)}$ be a 1-of-K vector indicating cluster membership
  - For example, $z^{(1)} = [0, 1, 0, \dots, 0]$ indicates that the first data point belongs to the second cluster
  - Let $\pi_k \coloneqq p\left(z_k^{(i)} = 1\right)$

Gaussian Mixture Models (GMMs)

Assume the following data-generating model for our dataset, $\mathcal{D} = \left\{\boldsymbol{x}^{(i)}\right\}_{i=1}^{N}$

1. Sample a cluster at random: $p\left(z_k^{(i)} = 1\right) = \pi_k$
2. Sample a data point from the chosen cluster: $p\left(\boldsymbol{x}^{(i)} \middle| z_k^{(i)} = 1\right) \sim N(\mu_k, \Sigma_k)$

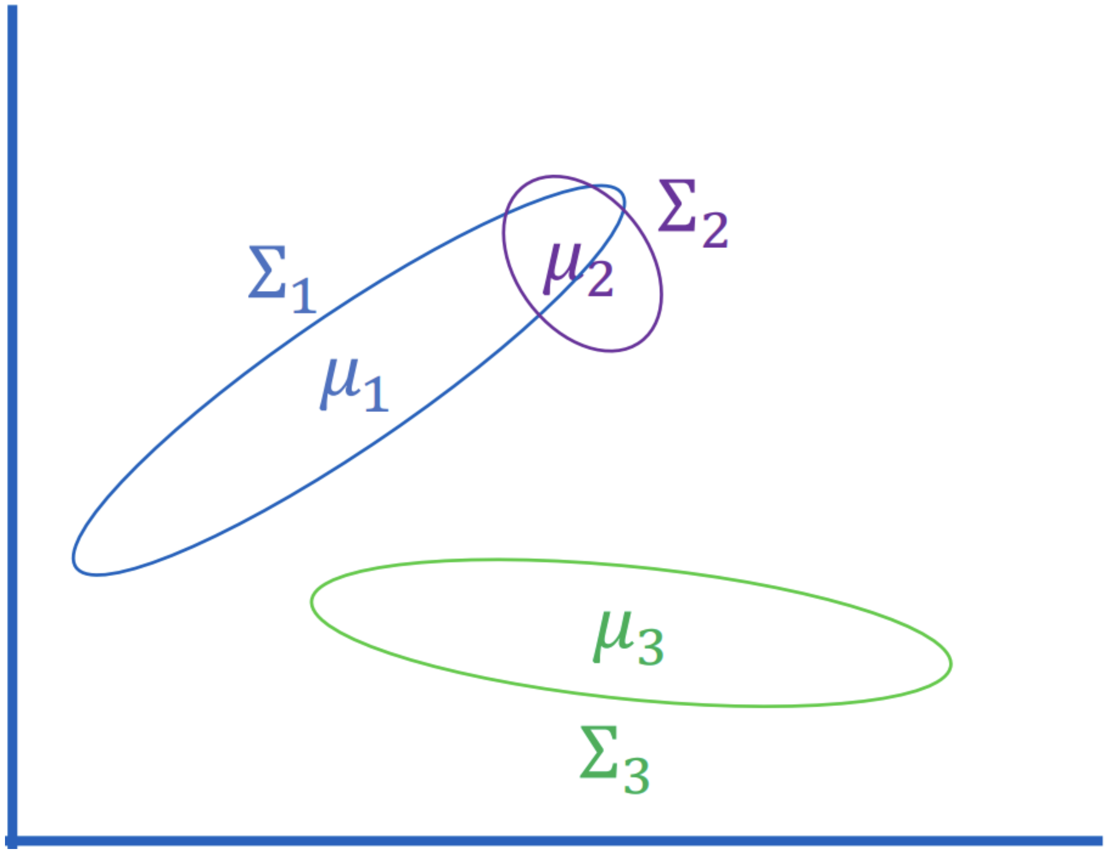
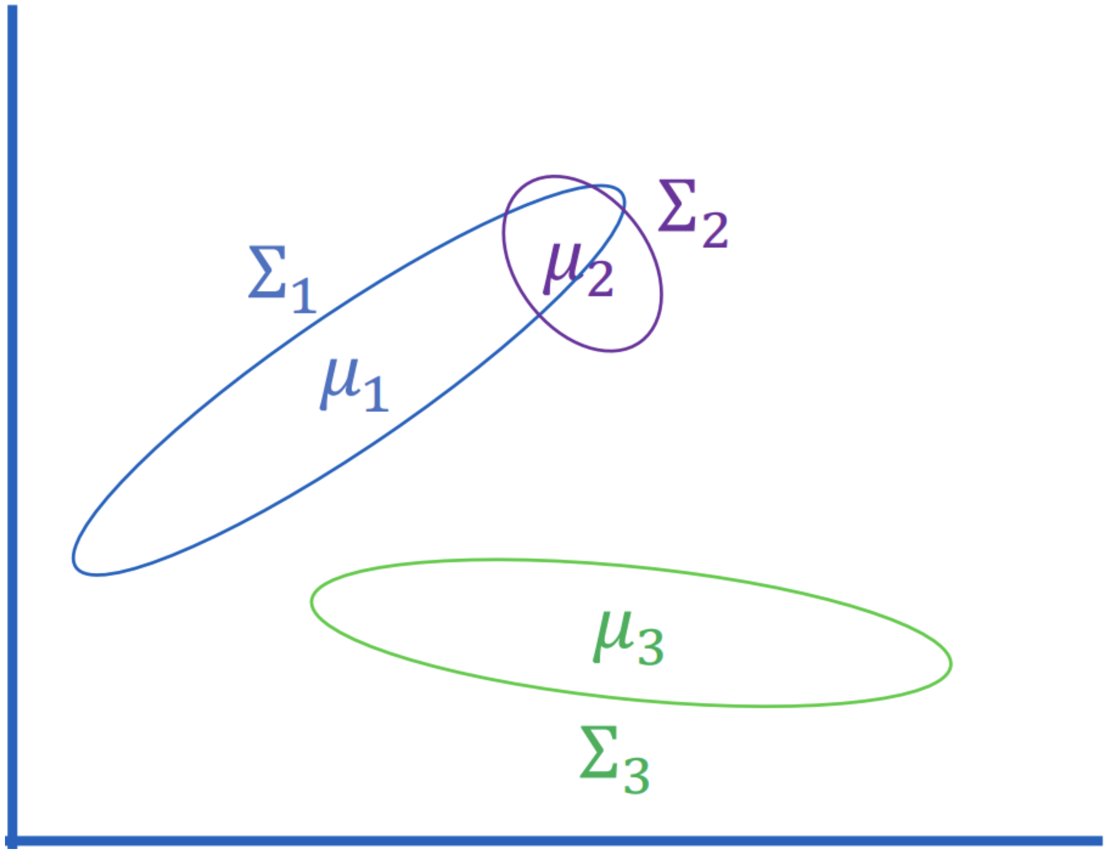


Figure courtesy of Pat Virtue

Gaussian Mixture Models (GMMs)

Assume the following data-generating model for our dataset, $\mathcal{D} = \left\{\boldsymbol{x}^{(i)}\right\}_{i=1}^{N}$

1. Sample a cluster at random: $p\left(z_k^{(i)} = 1\right) = \pi_k$
2. Sample a data point from the chosen cluster: $p\left(\boldsymbol{x}^{(i)} \middle| z_k^{(i)} = 1\right) \sim N(\mu_k, \Sigma_k)$



Let $\theta = \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K, \Sigma_1, \ldots, \Sigma_K, \pi_1, \ldots, \pi_K\}$

Figure courtesy of Pat Virtue

Maximizing the Likelihood?

- The log likelihood of $\mathcal{D} = \left\{\boldsymbol{x}^{(i)}, \boldsymbol{z}^{(i)}\right\}_{i=1}^{N}$ is

  $\ell_{\mathcal{D}}(\theta) =$

Maximizing the Likelihood?

- The log likelihood of $\mathcal{D} = \left\{\boldsymbol{x}^{(i)}, \boldsymbol{z}^{(i)}\right\}_{i=1}^{N}$ is

$$\ell_{\mathcal{D}}(\theta) = \log \prod_{i=1}^{N} p\left(\boldsymbol{x}^{(i)}, \boldsymbol{z}^{(i)} \middle| \theta\right) = \sum_{i=1}^{N} \log p\left(\boldsymbol{x}^{(i)}, \boldsymbol{z}^{(i)} \middle| \theta\right)$$

$$= \sum_{i=1}^{N} \log p\left(\boldsymbol{x}^{(i)} \middle| \boldsymbol{z}^{(i)}, \theta\right) + \log p\left(\boldsymbol{z}^{(i)} \middle| \theta\right)$$

$$= \sum_{i=1}^{N} \log \prod_{k=1}^{K} p\left(\boldsymbol{x}^{(i)} \middle| z_k^{(i)} = 1, \theta\right)^{z_k^{(i)}} + \log \prod_{k=1}^{K} p\left(z_k^{(i)} = 1 \middle| \theta\right)^{z_k^{(i)}}$$

$$= \sum_{i=1}^{N} \sum_{k=1}^{K} z_k^{(i)} \log p\left(\boldsymbol{x}^{(i)} \middle| z_k^{(i)} = 1, \theta\right) + \sum_{k=1}^{K} z_k^{(i)} \log p\left(z_k^{(i)} = 1 \middle| \theta\right)$$

$$= \sum_{i=1}^{N} \sum_{k=1}^{K} z_k^{(i)} \left(\log N\left(\boldsymbol{x}^{(i)}; \mu_k, \Sigma_k\right) + \log \pi_k\right)$$

Maximizing the Complete Likelihood is easy but requires $z^{(i)}$!

- The log complete likelihood of $\mathcal{D} = \left\{\boldsymbol{x}^{(i)}, \boldsymbol{z}^{(i)}\right\}_{i=1}^{N}$ is

$$\ell_{\mathcal{D}}(\theta) = \log \prod_{i=1}^{N} p\left(\boldsymbol{x}^{(i)}, \boldsymbol{z}^{(i)} \middle| \theta\right) = \sum_{i=1}^{N} \log p\left(\boldsymbol{x}^{(i)}, \boldsymbol{z}^{(i)} \middle| \theta\right)$$

$$= \sum_{i=1}^{N} \log p\left(\boldsymbol{x}^{(i)} \middle| \boldsymbol{z}^{(i)}, \theta\right) + \log p\left(\boldsymbol{z}^{(i)} \middle| \theta\right)$$

$$= \sum_{i=1}^{N} \log \prod_{k=1}^{K} p\left(\boldsymbol{x}^{(i)} \middle| z_k^{(i)} = 1, \theta\right)^{z_k^{(i)}} + \log \prod_{k=1}^{K} p\left(z_k^{(i)} = 1 \middle| \theta\right)^{z_k^{(i)}}$$

$$= \sum_{i=1}^{N} \sum_{k=1}^{K} z_k^{(i)} \log p\left(\boldsymbol{x}^{(i)} \middle| z_k^{(i)} = 1, \theta\right) + \sum_{k=1}^{K} z_k^{(i)} \log p\left(z_k^{(i)} = 1 \middle| \theta\right)$$

$$= \sum_{i=1}^{N} \sum_{k=1}^{K} z_k^{(i)} \left(\log N\left(\boldsymbol{x}^{(i)}; \mu_k, \Sigma_k\right) + \log \pi_k\right)$$

- Parameters decoupled → set partial derivatives equal to 0

Maximizing the Marginal Likelihood

- The log *marginal* likelihood of $\mathcal{D} = \{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$ is

$$\ell(\theta|\mathcal{D}) = \log \prod_{i=1}^{N} p\left(\boldsymbol{x}^{(i)}\middle|\theta\right) = \sum_{i=1}^{N} \log p\left(\boldsymbol{x}^{(i)}\middle|\theta\right)$$

$$= \sum_{i=1}^{N} \log \sum_{\boldsymbol{z}^{(i)}} p\left(\boldsymbol{x}^{(i)}\middle|\boldsymbol{z}^{(i)},\theta\right) p\left(\boldsymbol{z}^{(i)}\middle|\theta\right)$$

$$= \sum_{i=1}^{N} \log \sum_{\boldsymbol{z}^{(i)}} \prod_{k=1}^{K} \left(p\left(\boldsymbol{x}^{(i)}\middle|z_k^{(i)} = 1,\theta\right) p\left(z_k^{(i)} = 1\middle|\theta\right)\right)^{z_k^{(i)}}$$

$$= \sum_{i=1}^{N} \log \sum_{\boldsymbol{z}^{(i)}} \prod_{k=1}^{K} \left(N\left(\boldsymbol{x}^{(i)};\mu_k,\Sigma_k\right)\pi_k\right)^{z_k^{(i)}}$$

- Parameters coupled and *constrained* → gradient ascent is possible but complicated and slow to converge

Recipe for GMMs

- Define a model and model parameters
  - Assume K Gaussian clusters
  - Parameters: $\theta = \{\boldsymbol{\mu}_1, \dots, \boldsymbol{\mu}_K, \Sigma_1, \dots, \Sigma_K, \pi_1, \dots, \pi_K\}$
- Write down an objective function
  - Maximize the log marginal likelihood

$$\ell_{\mathcal{D}}(\theta) = \log \prod_{i=1}^{N} p\left(\boldsymbol{x}^{(i)} \middle| \theta\right)$$

- Optimize the objective w.r.t. the model parameters
  - Expectation-maximization

Expectation-Maximization for GMMs: Intuition

- Insight: if we knew the cluster assignments, $\boldsymbol{z}^{(i)}$, we could maximize the log complete likelihood instead of the log marginal likelihood
- Idea: replace $\boldsymbol{z}^{(i)}$ in the log complete likelihood with our "best guess" for $\boldsymbol{z}^{(i)}$ given the parameters and the data
- Observation: changing the parameters changes our "best guess" and vice versa
- Approach: iterate between updating our "best guess" and updating the parameters