Gaussian Mixture, EM Algorithm, Bias-Variance Trade-off

Milan Straka

📅 December 19, 2022

Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

CC BY SA
unless otherwise stated

K-Means Clustering

Input: Input points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, number of clusters K.

- Initialize $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ as K random input points.
- Repeat until convergence (or until patience runs out):
  - Compute the best possible $z_{i,k}$. It is easy to see that the smallest J is achieved by

$$z_{i,k} = \begin{cases} 1 & \text{if } k = \arg\min_j \|\boldsymbol{x}_i - \boldsymbol{\mu}_j\|^2, \\ 0 & \text{otherwise.} \end{cases}$$

  - Compute the best possible $\boldsymbol{\mu}_k = \arg\min_{\boldsymbol{\mu}} \sum_i z_{i,k} \|\boldsymbol{x}_i - \boldsymbol{\mu}\|^2$. By computing a derivative with respect to $\boldsymbol{\mu}$, we get

$$\boldsymbol{\mu}_k = \frac{\sum_i z_{i,k} \boldsymbol{x}_i}{\sum_i z_{i,k}}.$$

K-Means Clustering

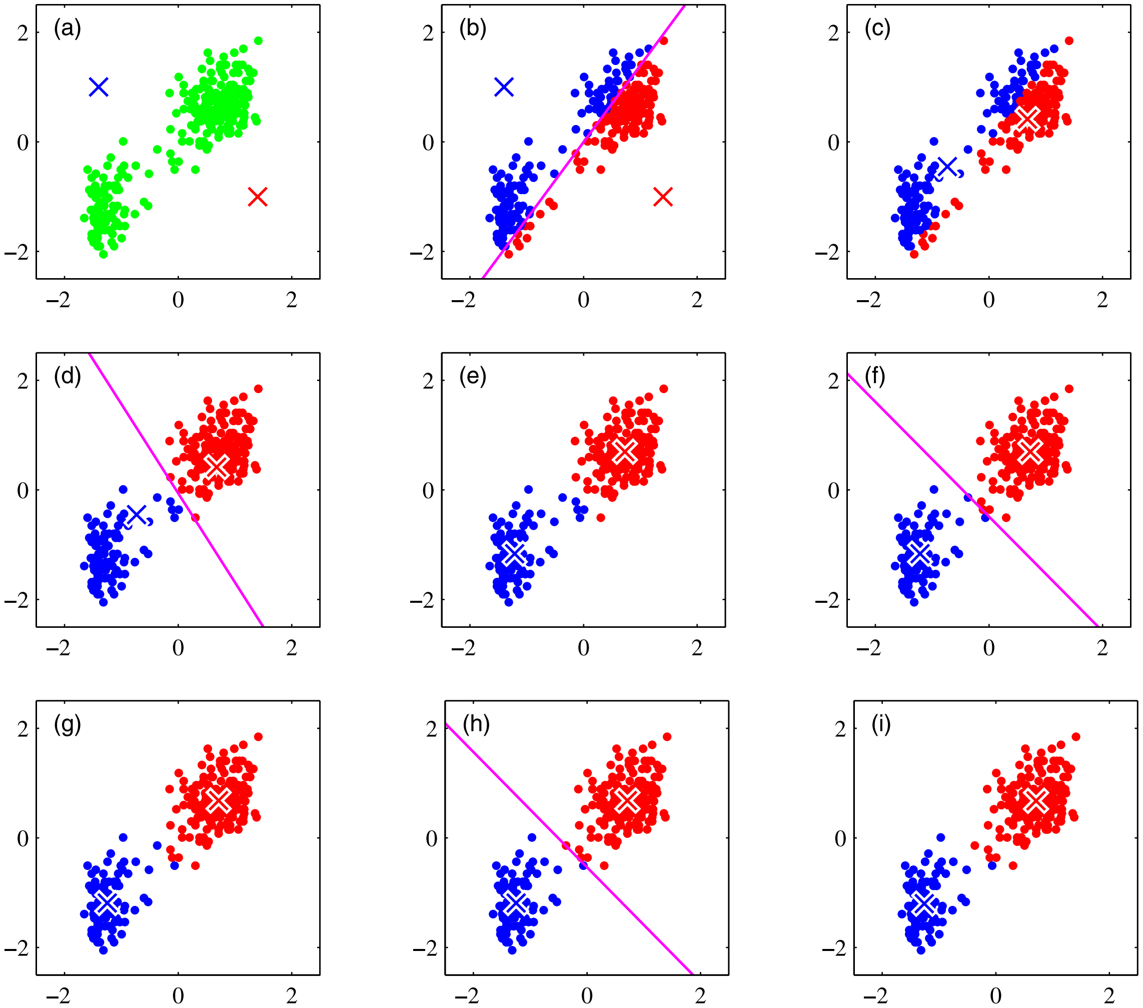
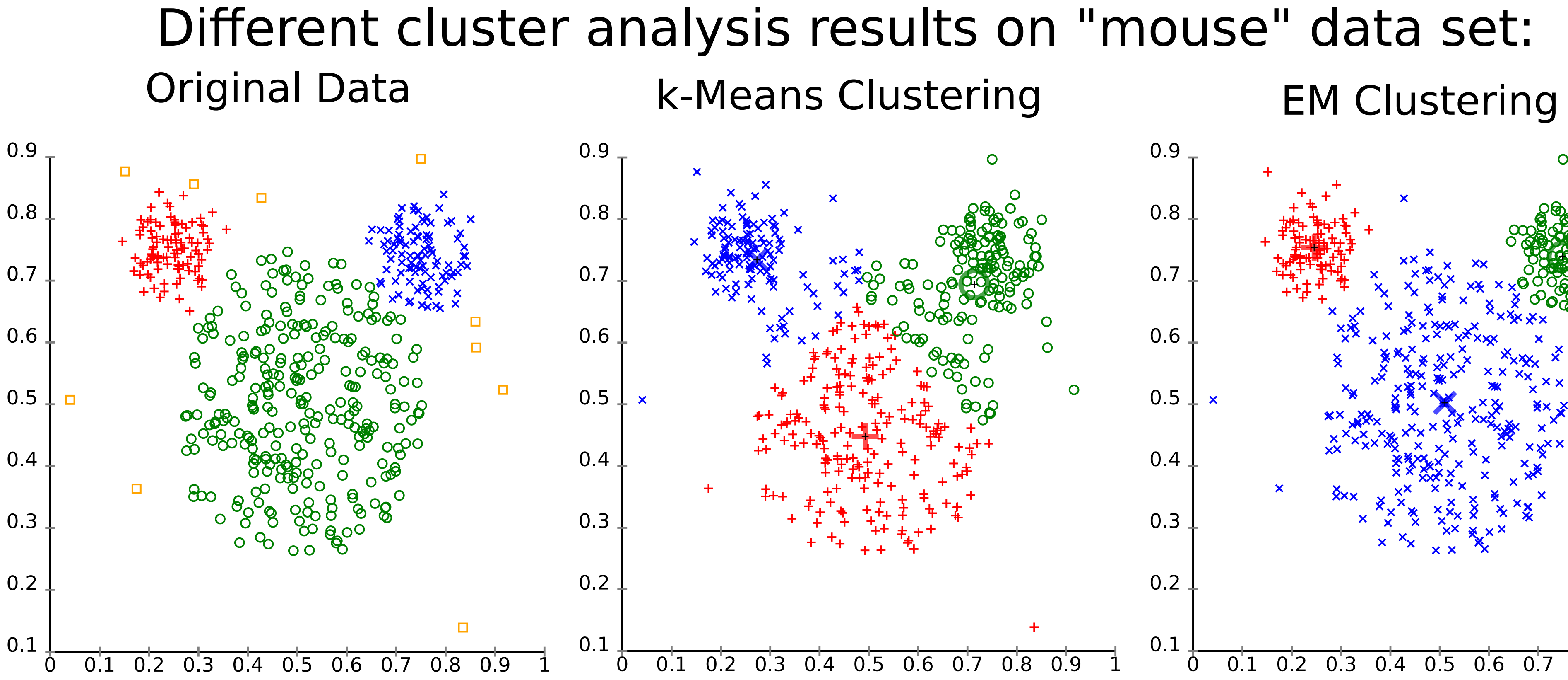


Figure 9.1 of Pattern Recognition and Machine Learning.

Gaussian Mixture vs K-Means

It could be useful to consider that different clusters might have different radii or even be ellipsoidal.



https://commons.wikimedia.org/wiki/File:ClusterAnalysis_Mouse.svg

Multivariate Gaussian Distribution

Recall that

$$\mathcal{N}(x;\mu,\sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}}\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

For D-dimensional vector $\boldsymbol{x}$, the multivariate Gaussian distribution takes the form

$$\mathcal{N}(\boldsymbol{x};\boldsymbol{\mu},\boldsymbol{\Sigma}) \stackrel{\text{def}}{=} \frac{1}{\sqrt{(2\pi)^D|\boldsymbol{\Sigma}|}}\exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right).$$

The biggest difference compared to the single-dimensional case is the *covariance matrix* $\boldsymbol{\Sigma}$, which is (in the non-degenerate case, which is the only one considered here) a *symmetric positive-definite matrix* of size $D \times D$.

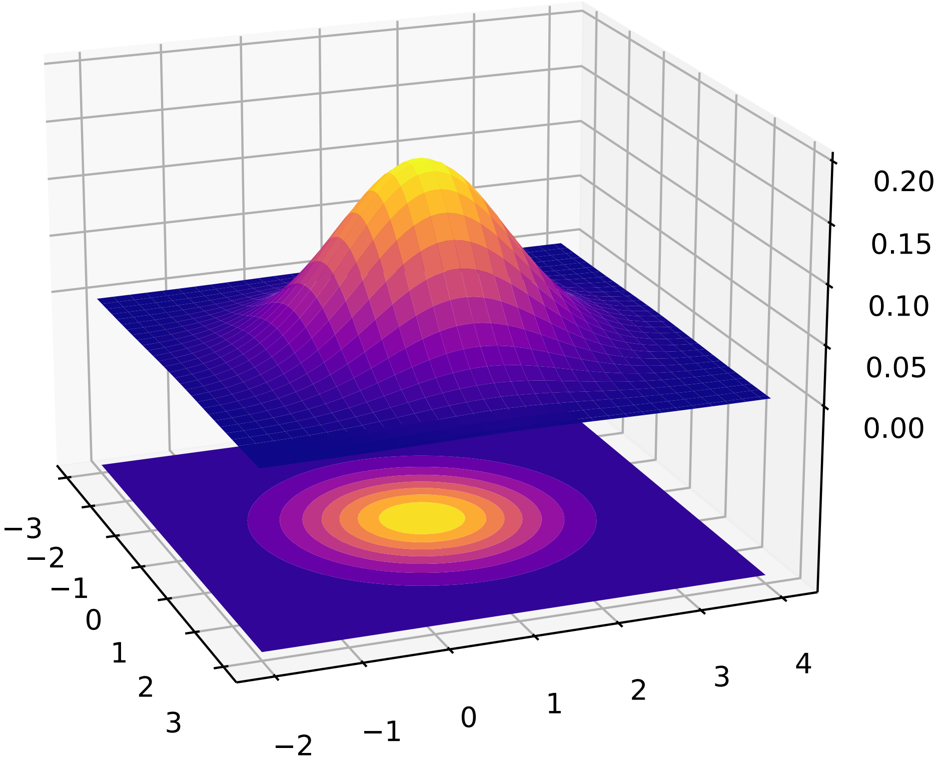
Multivariate Gaussian Distribution

If the covariance matrix is an identity, then the multivariate Gaussian distribution simplifies to

$$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{I}) = \frac{1}{\sqrt{(2\pi)^D}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T(\boldsymbol{x}-\boldsymbol{\mu})\right).$$

We can rewrite the exponent in this case as

$$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{I}) \propto \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{\mu}\|^2}{2}\right).$$



Therefore, the constant surfaces are concentric hyperspheres (circles in 2D, spheres in 3D) centered at the mean $\boldsymbol{\mu}$.

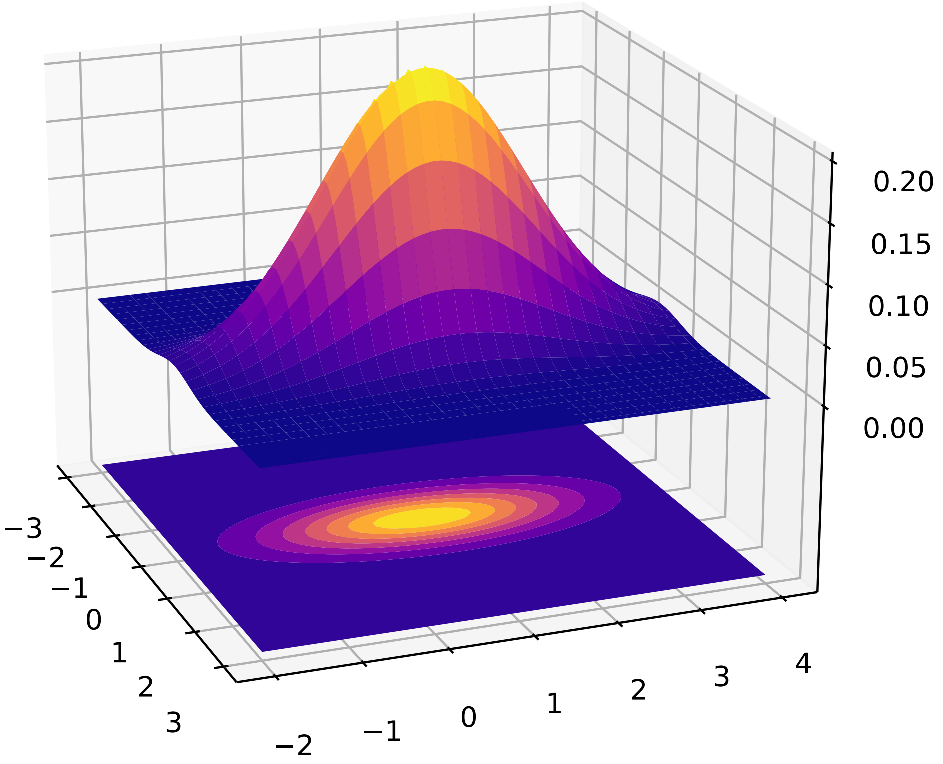
The same holds if the covariance is $\sigma^2\boldsymbol{I}$, only the hyperspheres' diameter changes.

Multivariate Gaussian Distribution

Now consider a diagonal covariance matrix $\boldsymbol{\Lambda}$. The exponent then simplifies to

$$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Lambda}) \propto \exp\left(-\sum\nolimits_i \frac{1}{2\boldsymbol{\Lambda}_{i,i}}\left(\boldsymbol{x}_i - \boldsymbol{\mu}_i\right)^2\right).$$

The constant surfaces in this case are axis-aligned hyperellipsoids (ellipses in 2D, ellipsoids in 3D) centered at the mean $\boldsymbol{\mu}$ with the size of the axes depending on the corresponding diagonal entries in the covariance matrix.



Multivariate Gaussian Distribution

In the general case of a full covariance matrix, the fact that it is positive definite implies it has real positive *eigenvalues* λ_i. Considering the corresponding eigenvectors $\boldsymbol{u}_i$, it can be shown that the constant surfaces are again hyperellipsoids centered at $\boldsymbol{\mu}$, but this time rotated so that their axes are the eigenvectors $\boldsymbol{u}_i$ with sizes $\lambda_i^{1/2}$.

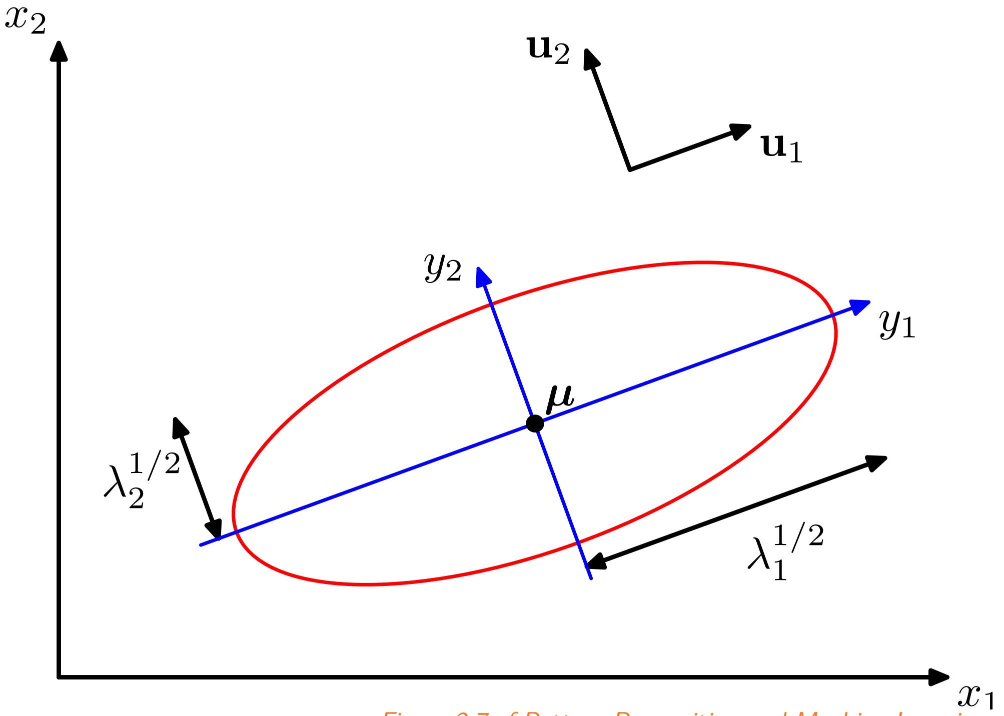
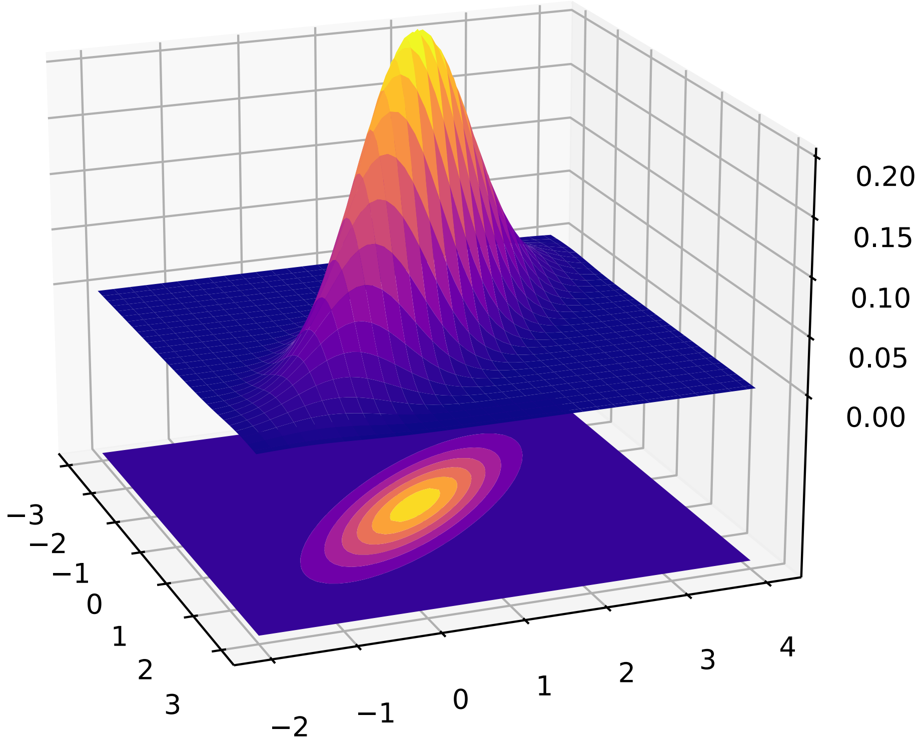


Figure 2.7 of Pattern Recognition and Machine Learning.



Multivariate Gaussian Distribution

Generally, we can rewrite a positive-definite matrix $\boldsymbol\Sigma$ as $\boldsymbol{U\Lambda U}^T = (\boldsymbol{U\Lambda}^{1/2})(\boldsymbol{U\Lambda}^{1/2})^T$, and then

$$\boldsymbol x \sim \mathcal N(\boldsymbol\mu, \boldsymbol\Sigma) \iff \boldsymbol x \sim \boldsymbol\mu + \boldsymbol{U\Lambda}^{1/2}\mathcal N(0, \boldsymbol I).$$

Therefore, when sampling from a distribution with a full covariance matrix, we can sample from a standard multivariate $\mathcal N(0, \boldsymbol I)$, scale by the eigenvalues of the covariance matrix, rotate according to the eigenvectors of the covariance matrix and finally shift by $\boldsymbol\mu$.

Note that different forms of covariance allow more generality, but also require more parameters:

- the $\sigma^2\boldsymbol I$ has a single parameter,
- the $\boldsymbol\Lambda$ has D parameters,
- the full covariance matrix $\boldsymbol\Sigma$ has $\binom{D+1}{2}$ parameters, i.e., $\Theta(D^2)$.

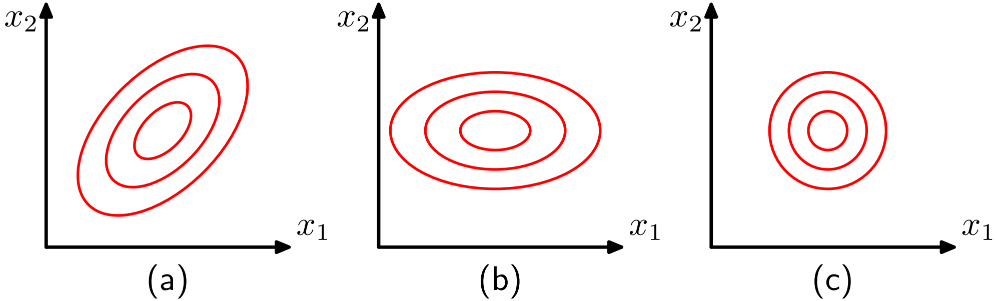


Figure 2.8 of Pattern Recognition and Machine Learning.

Gaussian Mixture

Let $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$ be a collection of N input examples, each being a D-dimensional vector $\boldsymbol{x}_i \in \mathbb{R}^D$. Let K, the number of target clusters, be given.

Our goal is to represent the data as a Gaussian mixture, which is a combination of K Gaussians in the form

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k).$$

Therefore, each cluster is parametrized as $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$.

Gaussian Mixture

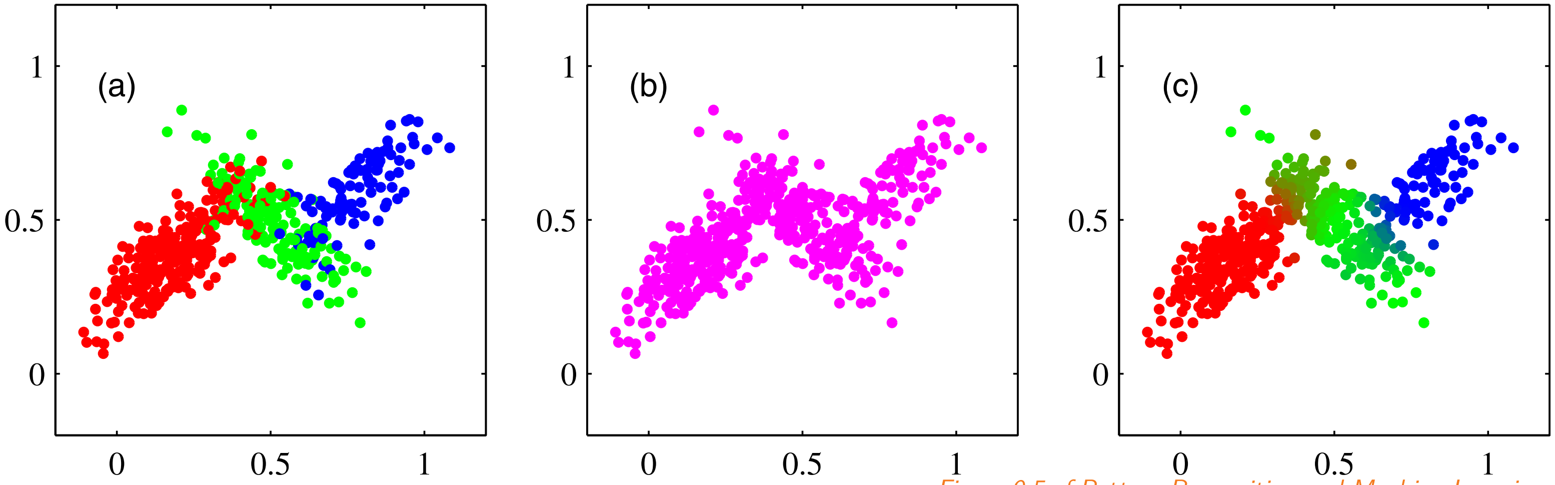


Figure 9.5 of Pattern Recognition and Machine Learning.

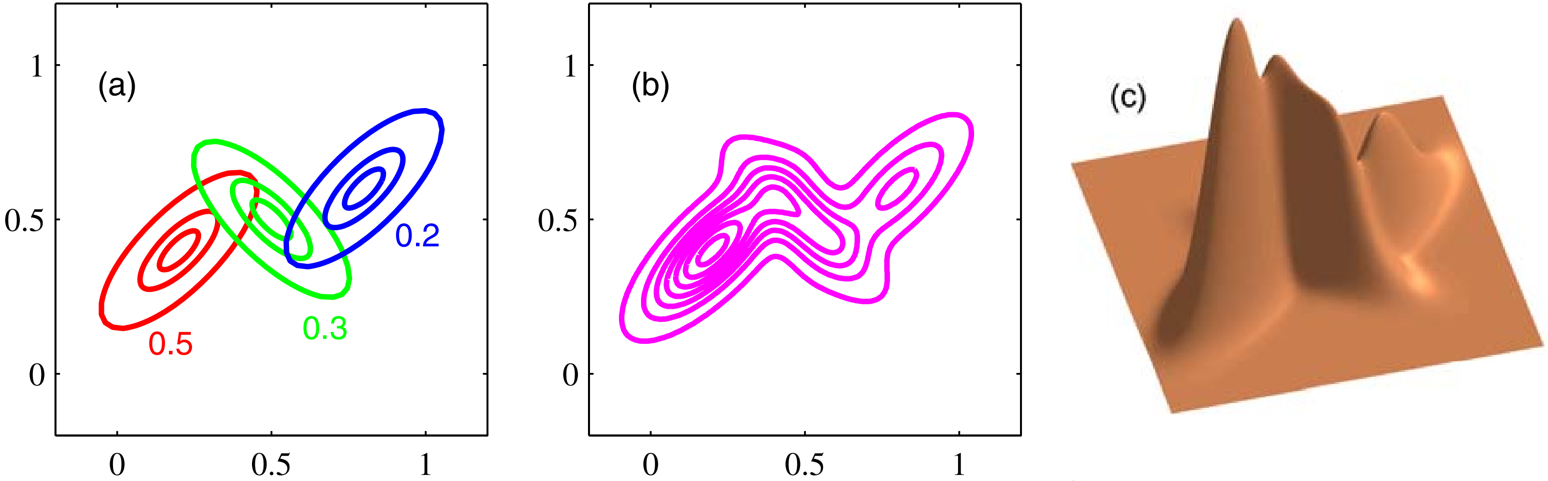


Figure 2.23 of Pattern Recognition and Machine Learning.

Gaussian Mixture

Let $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$ be a collection of N input examples, each being a D-dimensional vector $\boldsymbol{x}_i \in \mathbb{R}^D$. Let K, the number of target clusters, be given.

Our goal is to represent the data as a Gaussian mixture, which is a combination of K Gaussians in the form

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k).$$

Therefore, each cluster is parametrized as $\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$.

Let $\boldsymbol{z} \in \{0, 1\}^K$ be a K-dimensional random variable, such that exactly one z_k is 1, denoting to which cluster a training example belongs. Let the marginal distribution of z_k be

$$p(z_k = 1) = \pi_k,$$

so that the priors π_k represent the “fertility” of the clusters. Then, $p(\boldsymbol{z}) = \prod_k \pi_k^{z_k}$.

Gaussian Mixture

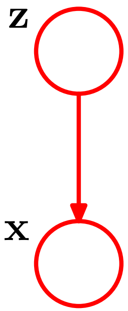


Figure 9.4 of Pattern Recognition and Machine Learning.

We can write

$$p(\boldsymbol{x}) = \sum_{\boldsymbol{z}} p(\boldsymbol{x}, \boldsymbol{z}) = \sum_{\boldsymbol{z}} p(\boldsymbol{z}) p(\boldsymbol{x}|\boldsymbol{z}) = \sum_{k=1}^K \pi_k \mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k),$$

and the log-probability of the whole clustering is therefore

$$\log p(\boldsymbol{X}; \boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{i=1}^N \log \left(\sum_{k=1}^K \pi_k \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right).$$

To fit a Gaussian mixture model, we utilize maximum likelihood estimation and minimize

$$L(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}; \boldsymbol{X}) = -\sum_i \log \sum_{k=1}^K \pi_k \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k).$$

Gaussian Mixture

The derivative of the loss with respect to $\boldsymbol{\mu}_k$ gives

$$\frac{\partial L(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}; \boldsymbol{X})}{\partial \boldsymbol{\mu}_k} = -\sum_i \frac{\pi_k \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{l=1}^K \pi_l \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)} \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{x}_i - \boldsymbol{\mu}_k).$$

Denoting $r(z_{i,k}) = \frac{\pi_k \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{l=1}^K \pi_l \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}$, setting the derivative equal to zero and multiplying by $\boldsymbol{\Sigma}_k$, we get

$$\boldsymbol{\mu}_k = \frac{\sum_i r(z_{i,k}) \boldsymbol{x}_i}{\sum_i r(z_{i,k})}.$$

The $r(z_{i,k})$ are usually called **responsibilities** and denote the probability $p(z_k = 1 | \boldsymbol{x}_i)$. Note that the responsibilities depend on $\boldsymbol{\mu}_k$, so the above equation is not an analytical solution for $\boldsymbol{\mu}_k$, but can be used as an *iterative* algorithm for converging to a local optimum.

Gaussian Mixture

For $\boldsymbol{\Sigma}_k$, we again compute the derivative of the loss, which is technically complicated (we need to compute a derivative of a matrix inverse, and also we need to differentiate a matrix determinant) and results in an analogous equation:

$$\boldsymbol{\Sigma}_k = \frac{\sum_i r(z_{i,k})(\boldsymbol{x}_i - \boldsymbol{\mu}_k)(\boldsymbol{x}_i - \boldsymbol{\mu}_k)^T}{\sum_i r(z_{i,k})}.$$

To minimize the loss with respect to $\boldsymbol{\pi}$, we need to include the constraint $\sum_k \pi_k = 1$, so we form a Lagrangian $\mathcal{L}(\boldsymbol{\pi}) = L(\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}; \boldsymbol{X}) - \lambda\left(\sum_k \pi_k - 1\right)$, and get

$$\frac{\partial \mathcal{L}(\boldsymbol{\pi})}{\partial \pi_k} = \sum_i \frac{\mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{l=1}^K \pi_l \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)} - \lambda.$$

Setting the derivative to zero and multiplying by π_k, we obtain $\pi_k = \frac{1}{\lambda} \cdot \sum_i r(z_{i,k})$, so

$$\pi_k = 1/N \cdot \sum_i r(z_{i,k}).$$

Gaussian Mixture

Input: Input points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, number of clusters K.

- Initialize $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ and π_k. It is common to start by running the K-Means algorithm to obtain $z_{i,k}$, set $r(z_{i,k}) \leftarrow z_{i,k}$ and use the **M step** below.
- Repeat until convergence (or until patience runs out):
  - **E step**. Evaluate the responsibilities as

$$r(z_{i,k}) = \frac{\pi_k \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{l=1}^{K} \pi_l \mathcal{N}(\boldsymbol{x}_i; \boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)} = p(z_k = 1 | \boldsymbol{x}_i, \boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}).$$

  - **M step**. Maximize the log-likelihood by setting

$$\boldsymbol{\mu}_k = \frac{\sum_i r(z_{i,k}) \boldsymbol{x}_i}{\sum_i r(z_{i,k})}, \quad \boldsymbol{\Sigma}_k = \frac{\sum_i r(z_{i,k}) (\boldsymbol{x}_i - \boldsymbol{\mu}_k)(\boldsymbol{x}_i - \boldsymbol{\mu}_k)^T}{\sum_i r(z_{i,k})}, \quad \pi_k = \frac{\sum_i r(z_{i,k})}{N}.$$

Gaussian Mixture

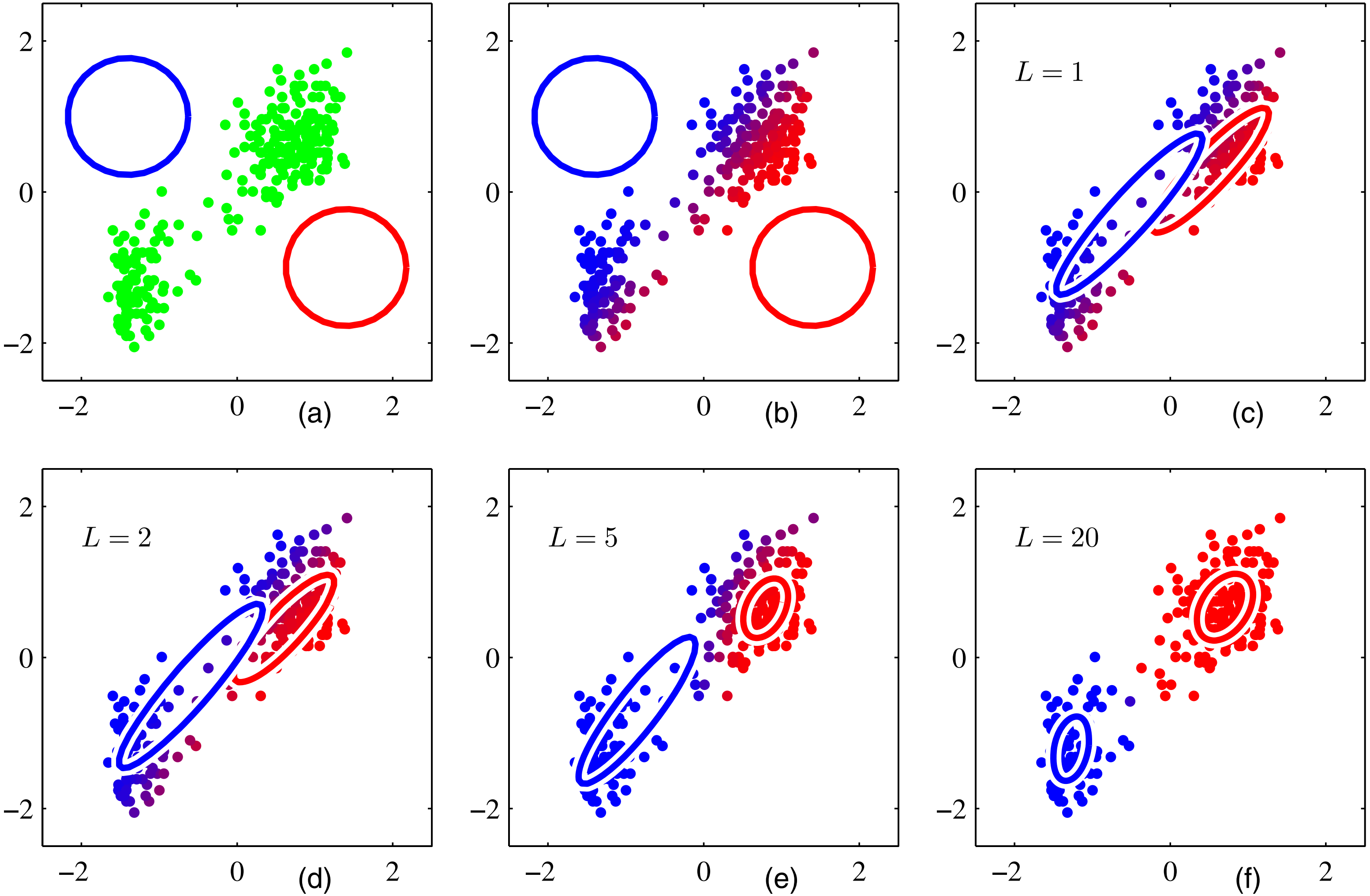


Figure 9.8 of Pattern Recognition and Machine Learning.

EM Algorithm

The algorithm for estimating the Gaussian mixture is an example of an **EM algorithm**.

The **EM algorithm** algorithm can be used when given a joint distribution $p(\boldsymbol{X}, \boldsymbol{Z}; \boldsymbol{w})$ over observed variables $\boldsymbol{X}$ and latent (hidden, unseen) variables $\boldsymbol{Z}$, parametrized by $\boldsymbol{w}$, we maximize

$$\log p(\boldsymbol{X}; \boldsymbol{w}) = \log \left(\sum_{\boldsymbol{Z}} p(\boldsymbol{X}, \boldsymbol{Z}; \boldsymbol{w}) \right)$$

with respect to $\boldsymbol{w}$.

Usually, the latent variables $\boldsymbol{Z}$ indicate membership of the data in one of the set of groups.

The main idea is to replace the computation of the logarithm of the sum over all latent variable values by the expectation of a logarithm of the joint probability under the posterior latent variable distribution $p(\boldsymbol{Z}|\boldsymbol{X}; \boldsymbol{w})$.

EM Algorithm

- Initialize the parameters $\boldsymbol{w}^\text{new}$.
- Repeat until convergence (or until patience runs out):
  - $\boldsymbol{w}^\text{old} \leftarrow \boldsymbol{w}^\text{new}$
  - **E step**. Evaluate

$$Q\big(\boldsymbol{w}|\boldsymbol{w}^\text{old}\big) = \mathbb{E}_{\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{w}^\text{old}}\big[\log p(\boldsymbol{X}, \boldsymbol{Z}; \boldsymbol{w})\big].$$

  - **M step**. Maximize the log-likelihood by computing

$$\boldsymbol{w}^\text{new} \leftarrow \operatorname*{arg\,max}_{\boldsymbol{w}} Q\big(\boldsymbol{w}|\boldsymbol{w}^\text{old}\big).$$

EM Algorithm – Proof

The EM algorithm updates $\boldsymbol{w}$ to maximize $Q\big(\boldsymbol{w}|\boldsymbol{w}^\mathrm{old}\big)$ on every step, and we now prove that this update of weights also causes the $\log p(\boldsymbol{X};\boldsymbol{w})$ to increase.

First note that for any $\boldsymbol{Z}$ with nonzero probability, we can write

$$\log p(\boldsymbol{X};\boldsymbol{w}) = \log p(\boldsymbol{X},\boldsymbol{Z};\boldsymbol{w}) - \log p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{w}).$$

Computing the expectation with respect to $p(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{w}^\mathrm{old})$, we get

$$\begin{aligned}\log p(\boldsymbol{X};\boldsymbol{w}) &= \sum_{\boldsymbol{Z}} p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{w}^\mathrm{old})\log p(\boldsymbol{X},\boldsymbol{Z};\boldsymbol{w}) - \sum_{\boldsymbol{Z}} p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{w}^\mathrm{old})\log p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{w})\\ &= Q\big(\boldsymbol{w}|\boldsymbol{w}^\mathrm{old}\big) + H\big(p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{w}^\mathrm{old}), p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{w})\big).\end{aligned}$$

The above equation holds for any $\boldsymbol{w}$, so also for $\boldsymbol{w}^\mathrm{old}$:

$$\log p(\boldsymbol{X};\boldsymbol{w}^\mathrm{old}) = Q\big(\boldsymbol{w}^\mathrm{old}|\boldsymbol{w}^\mathrm{old}\big) + H\big(p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{w}^\mathrm{old}), p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{w}^\mathrm{old})\big).$$

EM Algorithm – Proof

Subtracting the second term $\log p(\boldsymbol X; \boldsymbol w^\mathrm{old})$ from the first $\log p(\boldsymbol X; \boldsymbol w)$, we obtain

$$\begin{aligned}
&\log p(\boldsymbol X; \boldsymbol w) - \log p(\boldsymbol X; \boldsymbol w^\mathrm{old}) \\
&= Q\big(\boldsymbol w|\boldsymbol w^\mathrm{old}\big) - Q\big(\boldsymbol w^\mathrm{old}|\boldsymbol w^\mathrm{old}\big) + H\big(p(\boldsymbol Z|\boldsymbol X; \boldsymbol w^\mathrm{old}), p(\boldsymbol Z|\boldsymbol X; \boldsymbol w)\big) - H\big(p(\boldsymbol Z|\boldsymbol X; \boldsymbol w^\mathrm{old})\big) \\
&= Q\big(\boldsymbol w|\boldsymbol w^\mathrm{old}\big) - Q\big(\boldsymbol w^\mathrm{old}|\boldsymbol w^\mathrm{old}\big) + D_\mathrm{KL}\big(p(\boldsymbol Z|\boldsymbol X; \boldsymbol w^\mathrm{old})\big\|p(\boldsymbol Z|\boldsymbol X; \boldsymbol w)\big).
\end{aligned}$$

Given that KL divergence is nonnegative, we get

$$\log p(\boldsymbol X; \boldsymbol w) - \log p(\boldsymbol X; \boldsymbol w^\mathrm{old}) \geq Q\big(\boldsymbol w|\boldsymbol w^\mathrm{old}\big) - Q\big(\boldsymbol w^\mathrm{old}|\boldsymbol w^\mathrm{old}\big),$$

so if $\arg\max_{\boldsymbol w} Q\big(\boldsymbol w|\boldsymbol w^\mathrm{old}\big)$ is larger than $Q\big(\boldsymbol w^\mathrm{old}|\boldsymbol w^\mathrm{old}\big)$, we also increase $\log p(\boldsymbol X; \boldsymbol w)$.

To show that $\log p(\boldsymbol X; \boldsymbol w)$ actually converges to a stationary point, some additional regularity conditions are needed (one possibility is to require $Q(\boldsymbol w|\boldsymbol w^\mathrm{old})$ to be continuous in both $\boldsymbol w$ and $\boldsymbol w^\mathrm{old}$). For a more detailed treatment, see the 1983 paper *On the Convergence Properties of the EM Algorithm* by C. F. Jeff Wu.

Bias-Variance Trade-off

Consider a model $y(\boldsymbol{x})$ solving a regression problem with MSE loss

$$L = \mathbb{E}_{\mathbf{x},\mathrm{t}}\left[(y(\boldsymbol{x}) - t)^2\right].$$

Denoting $g(\boldsymbol{x}) \overset{\text{def}}{=} \mathbb{E}_{\mathrm{t}|\mathbf{x}}\left[t\right]$, we can rewrite $\big(y(\boldsymbol{x}) - t\big)^2$ as

$$\begin{aligned}\big(y(\boldsymbol{x}) - t\big)^2 &= \big(y(\boldsymbol{x}) - g(\boldsymbol{x}) + g(\boldsymbol{x}) - t\big)^2 \\ &= \big(y(\boldsymbol{x}) - g(\boldsymbol{x})\big)^2 + 2\big(y(\boldsymbol{x}) - g(\boldsymbol{x})\big)\big(g(\boldsymbol{x}) - t\big) + \big(g(\boldsymbol{x}) - t\big)^2.\end{aligned}$$

When computing an expectation with respect to $p_{\mathrm{data}}(\boldsymbol{x}, t)$, we obtain

$$\begin{aligned}L &= \mathbb{E}_{\mathbf{x},\mathrm{t}}\left[(y(\boldsymbol{x}) - g(\boldsymbol{x}))^2\right] + 2\mathbb{E}_{\mathbf{x},\mathrm{t}}\left[(y(\boldsymbol{x}) - g(\boldsymbol{x}))(g(\boldsymbol{x}) - t)\right] + \mathbb{E}_{\mathbf{x},\mathrm{t}}\left[(g(\boldsymbol{x}) - t)^2\right] \\ &= \mathbb{E}_{\mathbf{x},\mathrm{t}}\left[(y(\boldsymbol{x}) - g(\boldsymbol{x}))^2\right] + \mathbb{E}_{\mathbf{x},\mathrm{t}}\left[(g(\boldsymbol{x}) - t)^2\right],\end{aligned}$$

because $\mathbb{E}_{\mathrm{t}|\mathbf{x}}\left[g(\boldsymbol{x}) - t\right] = 0.$

Bias-Variance Trade-off

We have decomposed the loss into two components, where the second is the “label noise” called **irreducible error**.

We now further decompose the first component $\mathbb{E}_{\mathbf{x},\mathrm{t}}\big[(y(\boldsymbol{x}) - g(\boldsymbol{x}))^2\big]$.

Assuming $\mathcal{D}$ is a dataset obtained from the data-generating distribution, we denote the prediction of a model trained using this dataset as $y(\boldsymbol{x};\mathcal{D})$.

$$\begin{aligned}\big(y(\boldsymbol{x};\mathcal{D}) - g(\boldsymbol{x})\big)^2 &= \big(y(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}\big[y(\boldsymbol{x};\mathcal{D})\big] + \mathbb{E}_{\mathcal{D}}\big[y(\boldsymbol{x};\mathcal{D})\big] - g(\boldsymbol{x})\big)^2 \\ &= \big(y(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}\big[y(\boldsymbol{x};\mathcal{D})\big]\big)^2 \\ &+ 2\big(y(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}\big[y(\boldsymbol{x};\mathcal{D})\big]\big)\big(\mathbb{E}_{\mathcal{D}}\big[y(\boldsymbol{x};\mathcal{D})\big] - g(\boldsymbol{x})\big) \\ &+ \big(\mathbb{E}_{\mathcal{D}}\big[y(\boldsymbol{x};\mathcal{D})\big] - g(\boldsymbol{x})\big)^2.\end{aligned}$$

Note that $\mathbb{E}_{\mathcal{D}}\big[y(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})]\big] = 0$, therefore, for a given $\boldsymbol{x}$, we have

$$\mathbb{E}_{\mathcal{D}}\big[(y(\boldsymbol{x};\mathcal{D}) - g(\boldsymbol{x}))^2\big] = \mathbb{E}_{\mathcal{D}}\big[(y(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})])^2\big] + \big(\mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})] - g(\boldsymbol{x})\big)^2.$$

Bias-Variance Trade-off

Putting all the parts together, we get that

$$\begin{aligned}\mathbb{E}_{\mathcal{D}}\big[L\big] &= \mathbb{E}_{\mathcal{D}}\Big[\mathbb{E}_{\mathbf{x},\mathrm{t}}\big[(y(\boldsymbol{x};\mathcal{D}) - t)^2\big]\Big] \\ &= \mathbb{E}_{\mathbf{x},\mathrm{t}}\Bigg[\underbrace{\big(\mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})] - g(\boldsymbol{x})\big)^2}_{\text{bias}^2} + \underbrace{\mathbb{E}_{\mathcal{D}}\Big[\big(y(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\boldsymbol{x};\mathcal{D})]\big)^2\Big]}_{\text{variance}} + \underbrace{\big(g(\boldsymbol{x}) - t\big)^2}_{\text{irreducible error}}\Bigg].\end{aligned}$$

This is the so-called **bias-variance trade-off**, showing that the expected loss decomposes into the three above components.

For classification problems, we can use the same decomposition on the MSE of probabilities, and it is also possible to derive an analogy using the so-called 0-1 loss (see *A Unified Bias-Variance Decomposition* by P. Domingos for the exact formulation).

This decomposition has been long interpreted as:

> *The price to pay for achieving low bias is high variance.*

Bias-Variance Trade-off

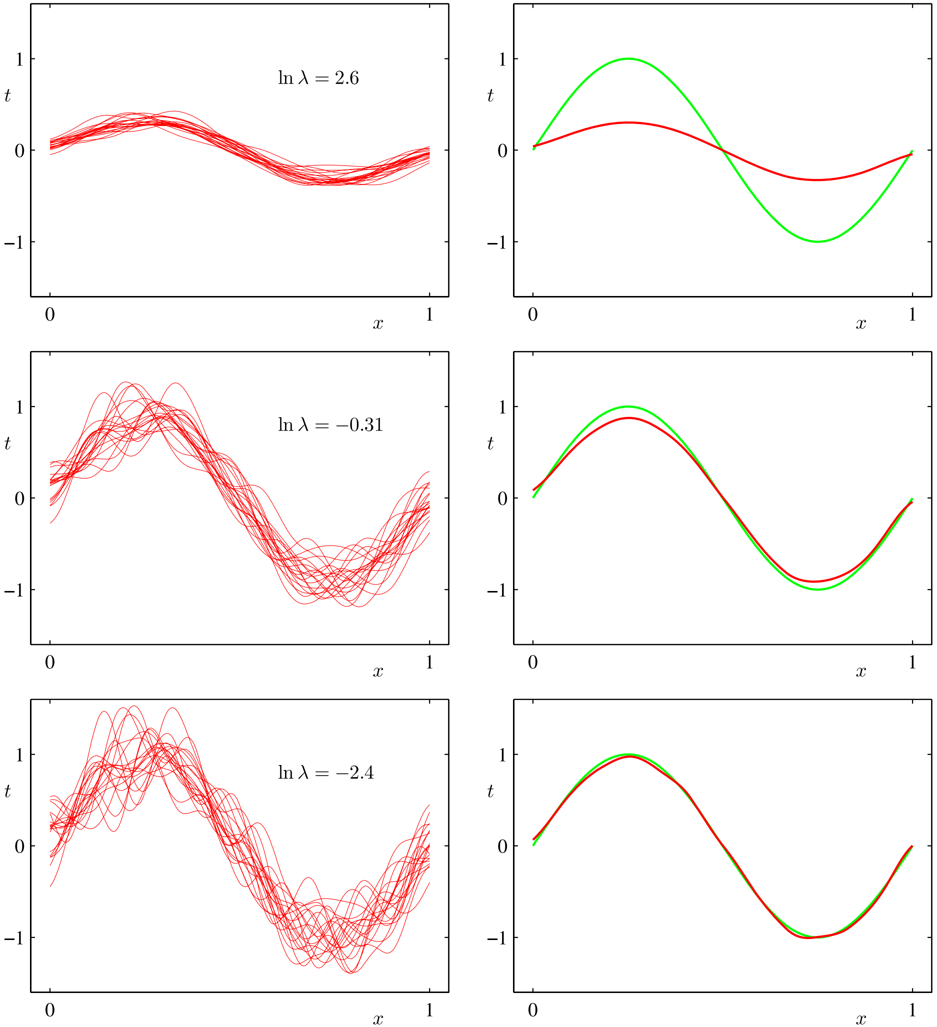


Figure 3.5 of Pattern Recognition and Machine Learning.

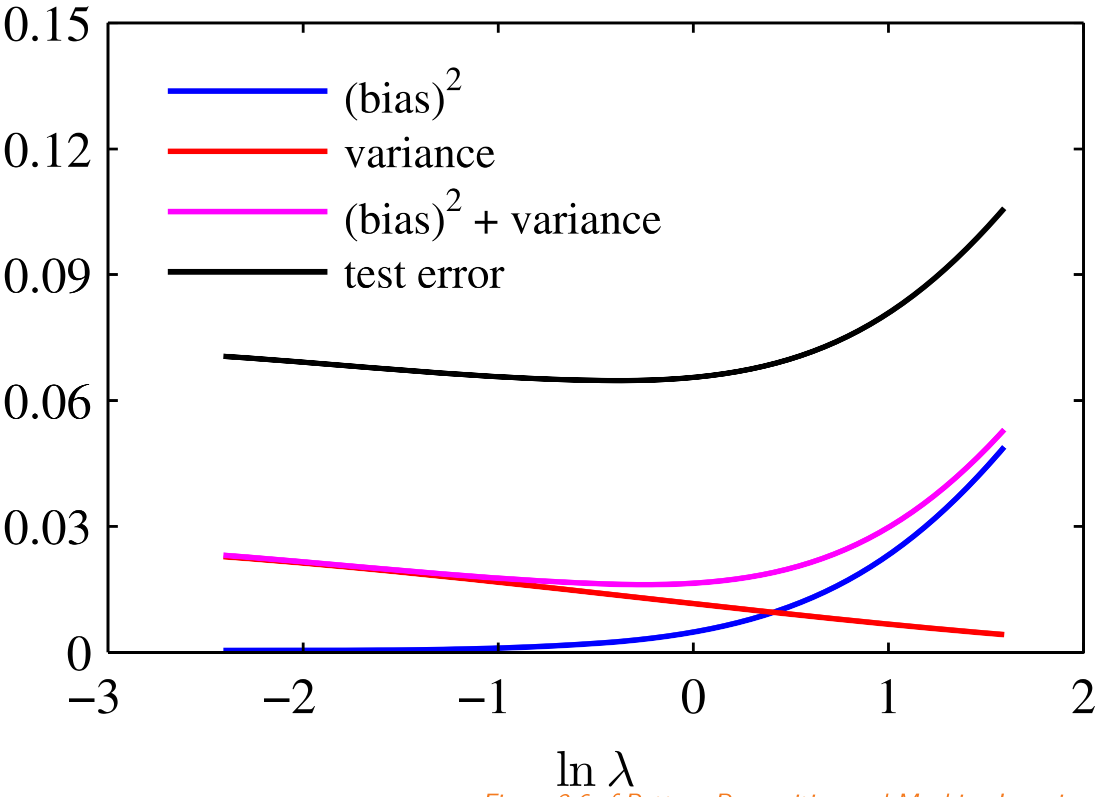


Figure 3.6 of Pattern Recognition and Machine Learning.

Bias-Variance Trade-off

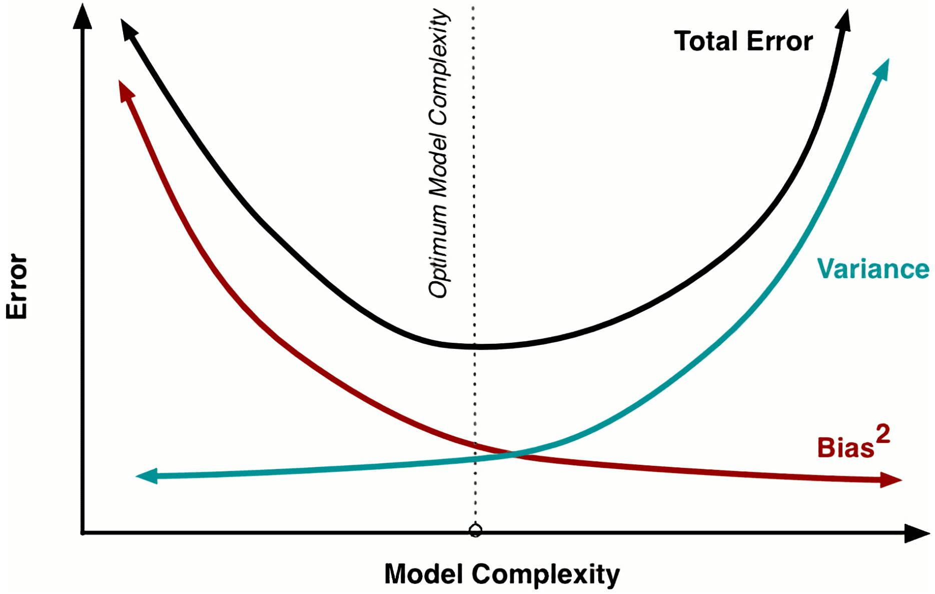


Figure 3.2 of "On the Bias-Variance Tradeoff: Textbooks Need an Update" by B. Neal.

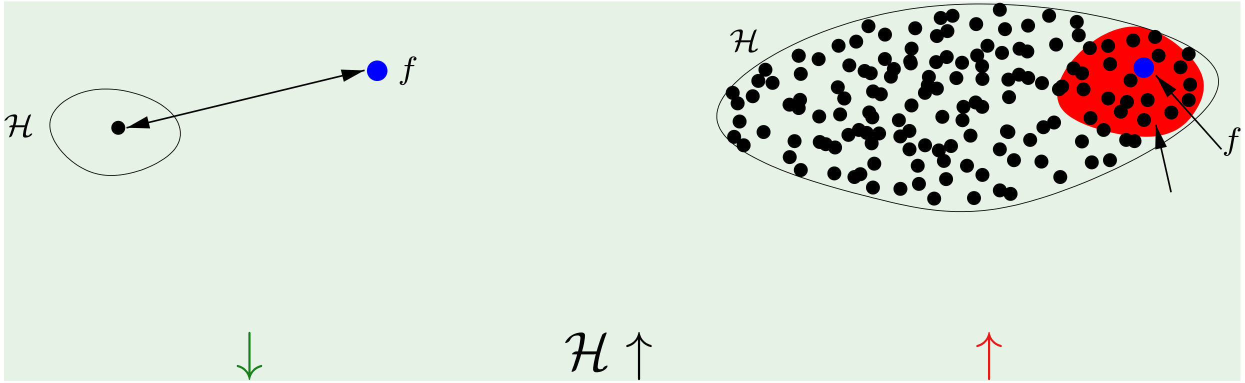


Figure 3.1 of "On the Bias-Variance Tradeoff: Textbooks Need an Update" by B. Neal.

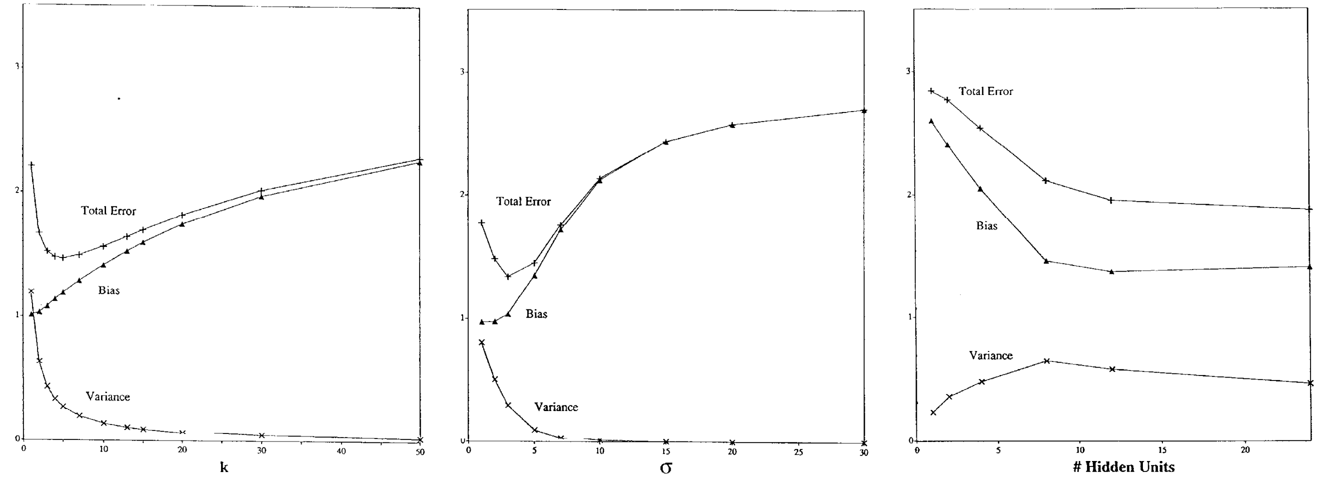
Bias-Variance Trade-off

For a k-NN search, when we consider an expectation over all possible labelings of a fixed training set, the MSE decomposes as

$$\mathbb{E}\Big[\big(y(\boldsymbol x) - t(\boldsymbol x)\big)^2\Big] = \left(t(\boldsymbol x) - \frac{1}{K}\sum_{k=1}^K t\big(N_k(\boldsymbol x)\big)\right)^2 + \frac{\sigma^2}{K} + \sigma^2,$$

where $N_k(\boldsymbol x)$ is the $k^{\textrm{th}}$ nearest neighbor of $\boldsymbol x$ and σ^2 is the irreducible error.

Bias-Variance Trade-off



(a) K-nearest neighbor (KNN) (higher k is *less* complexity)

(b) Kernel regression (higher kernel width σ is *less* complexity)

(c) Single hidden layer neural network (higher "# Hidden Units" is *more* complexity)

Figure 3.3 of "On the Bias-Variance Tradeoff: Textbooks Need an Update" by B. Neal.

Quoting from *Neural Networks and the Bias/Variance Decomposition* by S. Geman, 1992:

> *The basic trend is what we expect: bias falls and variance increases with the number of hidden units. The effects are not perfectly demonstrated (notice, for example, the dip in variance in the experiments with the largest numbers of hidden units), presumably because the phenomenon of overfitting is complicated by convergence issues and perhaps also by our decision to stop the training prematurely.*

Bias-Variance Trade-off

However, in past years, neural networks with increasing capacity have performed better and better.

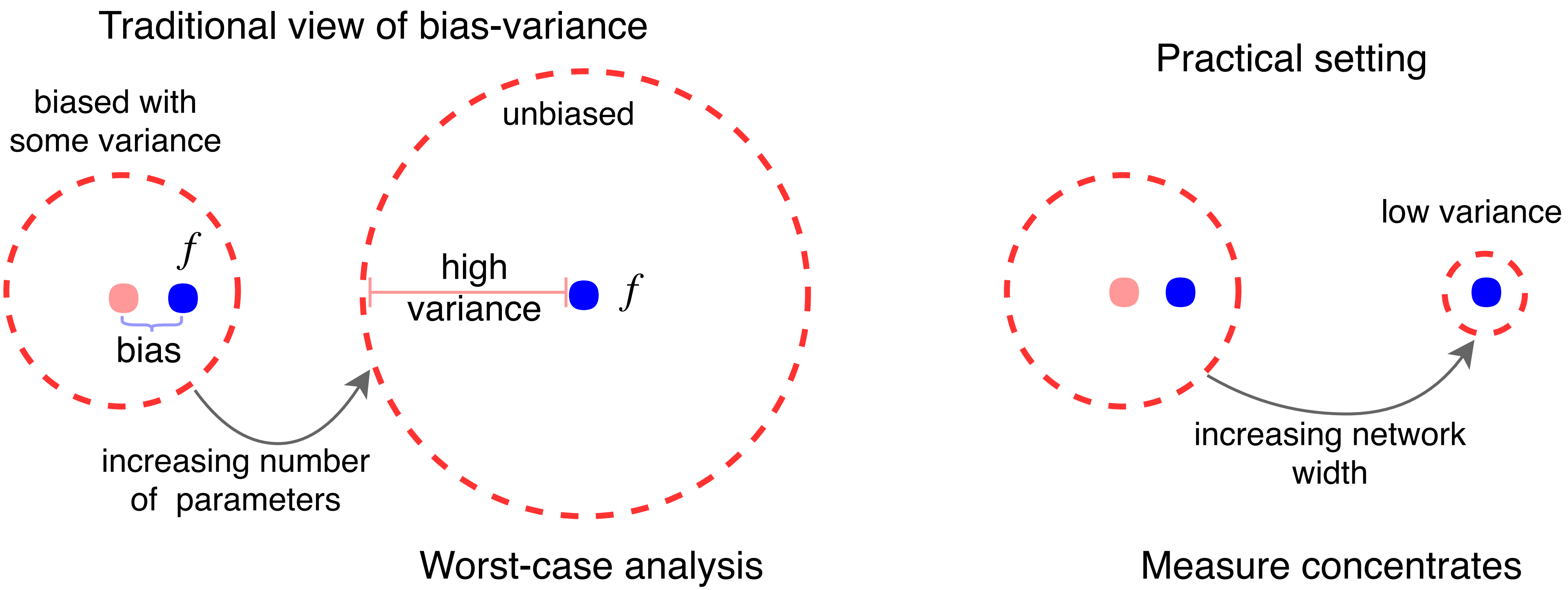


Figure 4.1 of "On the Bias-Variance Tradeoff: Textbooks Need an Update" by B. Neal.

Double Descent

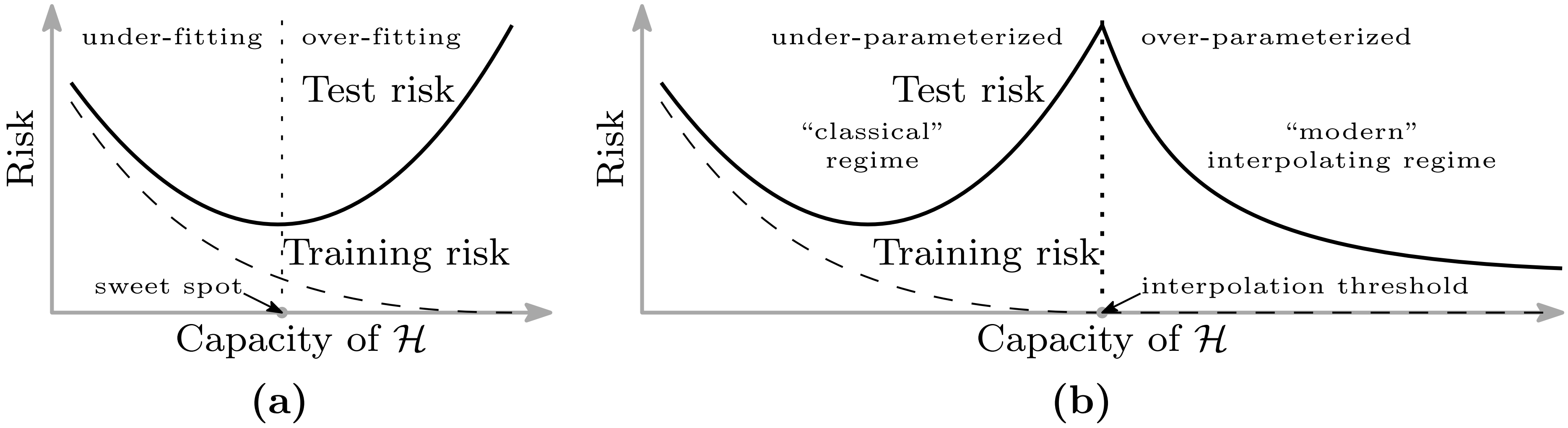


Figure 1: **Curves for training risk (dashed line) and test risk (solid line).** (**a**) The classical *U-shaped risk curve* arising from the bias-variance trade-off. (**b**) The *double descent risk curve*, which incorporates the U-shaped risk curve (i.e., the "classical" regime) together with the observed behavior from using high capacity function classes (i.e., the "modern" interpolating regime), separated by the interpolation threshold. The predictors to the right of the interpolation threshold have zero training risk.

Figure 1 of "Reconciling modern machine learning practice and the bias-variance trade-off" by M. Belkin et al.

Double Descent – Overparametrized with Minimum L2

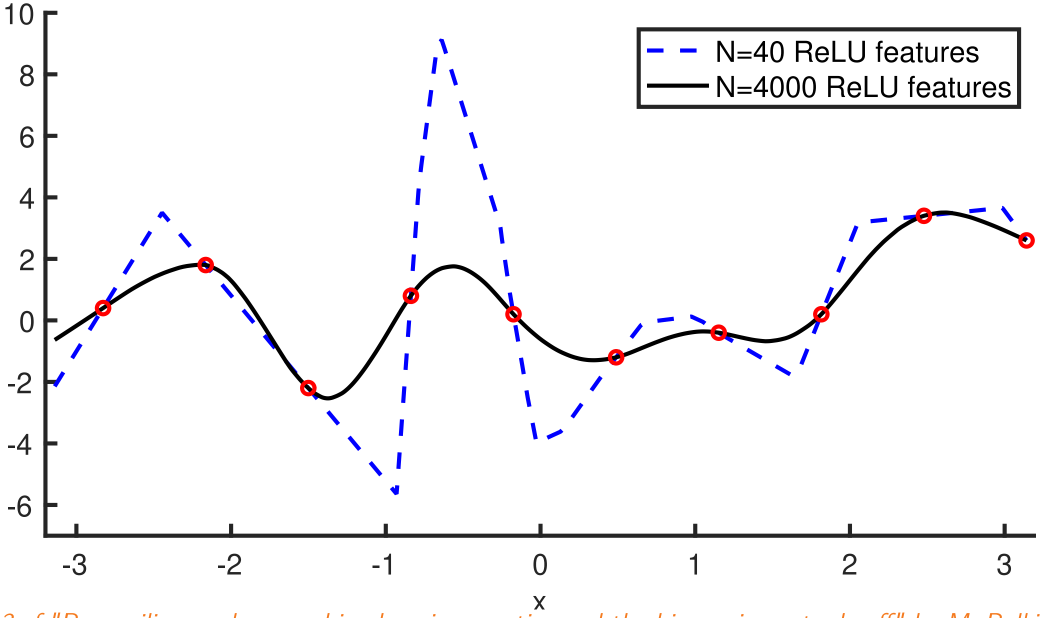


Figure 3 of "Reconciling modern machine learning practice and the bias-variance trade-off" by M. Belkin et al.

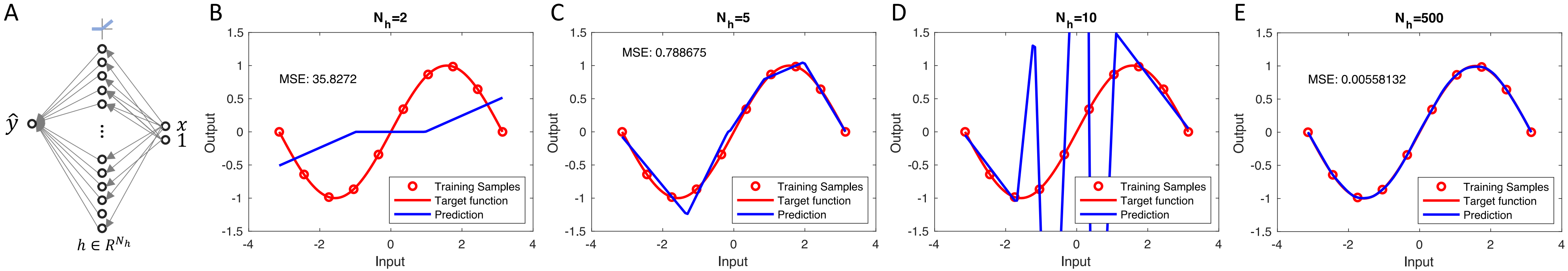


Figure 1 of "Minnorm training: an algorithm for training over-parameterized deep neural networks", https://arxiv.org/abs/1806.00730

Double Descent – RFF on MNIST

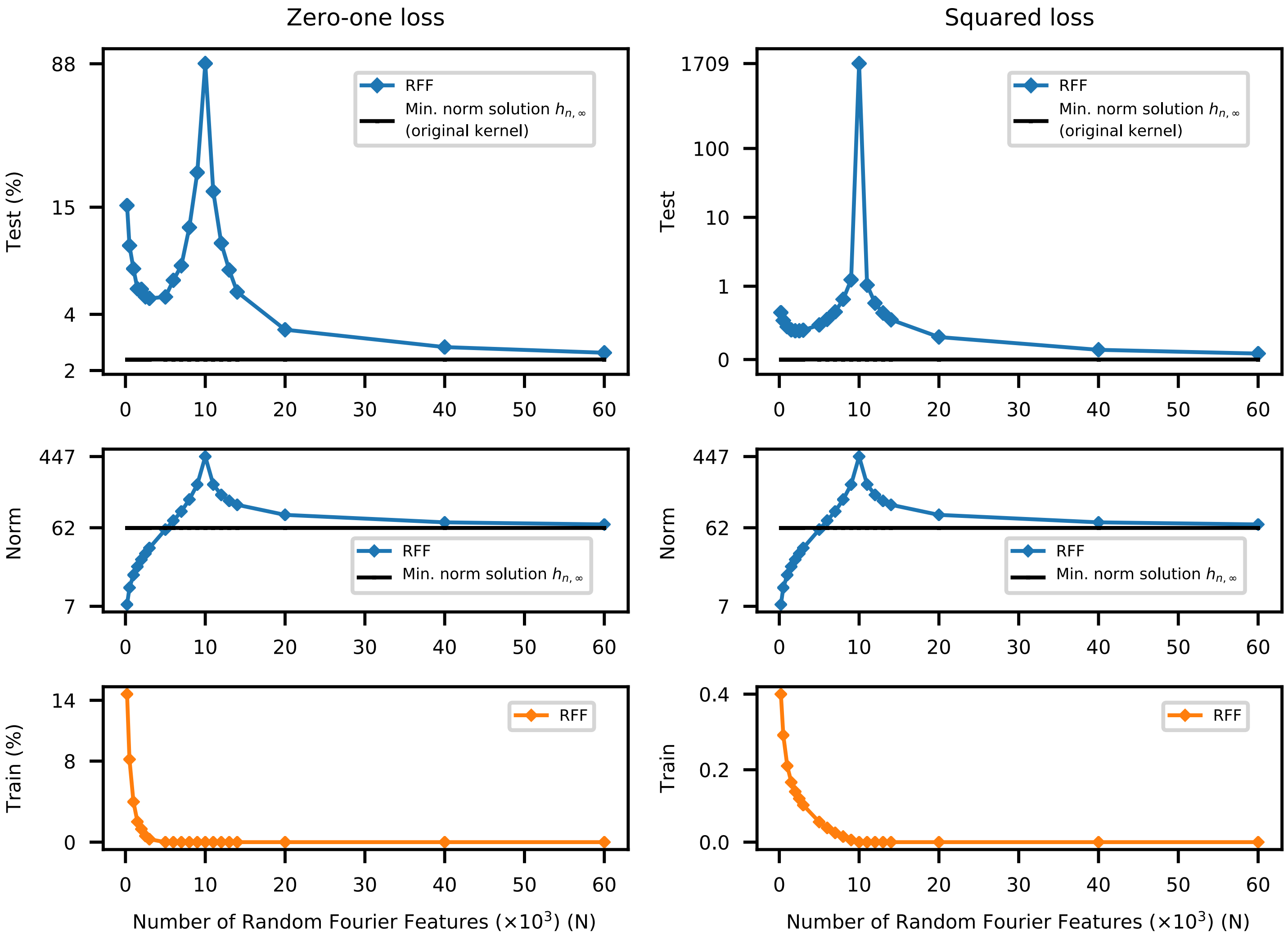


Figure 2 of "Reconciling modern machine learning practice and the bias-variance trade-off" by M. Belkin et al.

Double Descent – MLP and RF on MNIST

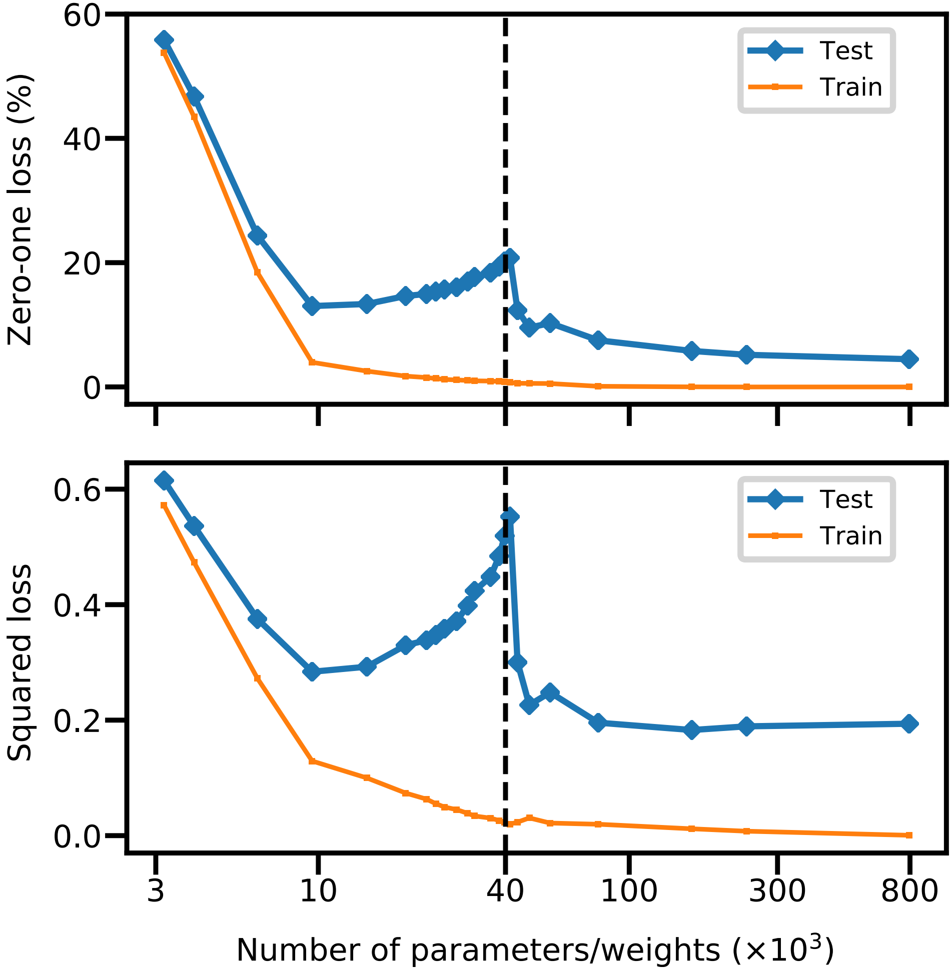


Figure 4 of "Reconciling modern machine learning practice and the bias-variance trade-off" by M. Belkin et al.

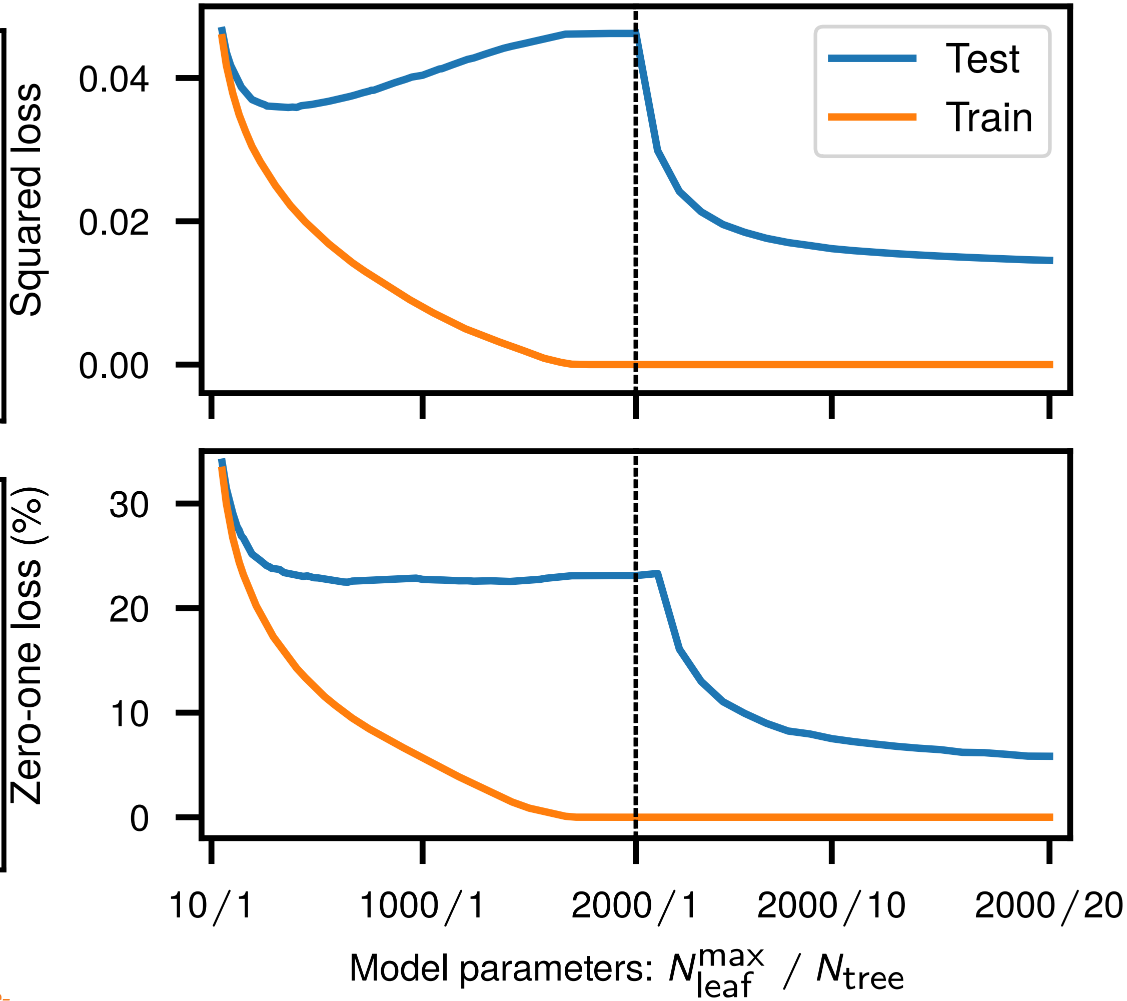


Figure 5 of "Reconciling modern machine learning practice and the bias-variance trade-off" by M. Belkin et al.

Deep Double Descent

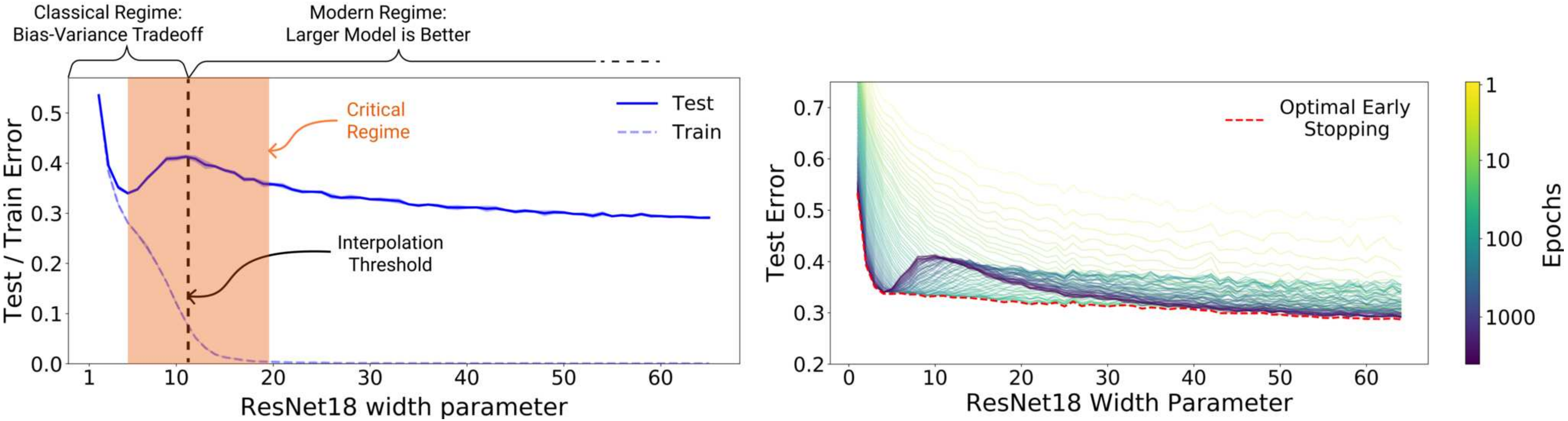


Figure 1: **Left:** Train and test error as a function of model size, for ResNet18s of varying width on CIFAR-10 with 15% label noise. **Right:** Test error, shown for varying train epochs. All models trained using Adam for 4K epochs. The largest model (width 64) corresponds to standard ResNet18.

Figure 1 of "Deep Double Descent: Where Bigger Models and More Data Hurt" by P. Nakkiran et al.

Deep Double Descent – Effective Model Complexity

The authors define the **Effective Model Complexity** (**EMC**) of a training procedure $\mathcal{T}$ with respect to distribution $\mathcal{D}$ and parameter $\varepsilon > 0$ as

$$\mathrm{EMC}_{\mathcal{D},\varepsilon}(\mathcal{T}) \stackrel{\text{def}}{=} \max\left\{n \,\middle|\, \mathbb{E}_{\mathrm{S}\sim\mathcal{D}^n}\left[\mathrm{Error}_S(\mathcal{T}(S))\right] \leq \varepsilon\right\},$$

where $\mathrm{Error}_S(M)$ is the mean error of a model M on the train samples S.

Hypothesis: For any natural data distribution $\mathcal{D}$, neural-network-based training procedure $\mathcal{T}$, and small $\varepsilon > 0$, if we consider the task of predicting labels based on n samples from $\mathcal{D}$, then:

- **Under-parametrized regime.** If $\mathrm{EMC}_{\mathcal{D},\varepsilon}(\mathcal{T})$ is sufficiently smaller than n, any perturbation of $\mathcal{T}$ that increases its effective complexity will decrease the test error.
- **Over-parametrized regime.** If $\mathrm{EMC}_{\mathcal{D},\varepsilon}(\mathcal{T})$ is sufficiently larger than n, any perturbation of $\mathcal{T}$ that increases its effective complexity will decrease the test error.
- **Critically parametrized regime.** If $\mathrm{EMC}_{\mathcal{D},\varepsilon}(\mathcal{T}) \approx n$, then a perturbation of $\mathcal{T}$ that increases its effective complexity might decrease **or increase** the test error.

Deep Double Descent

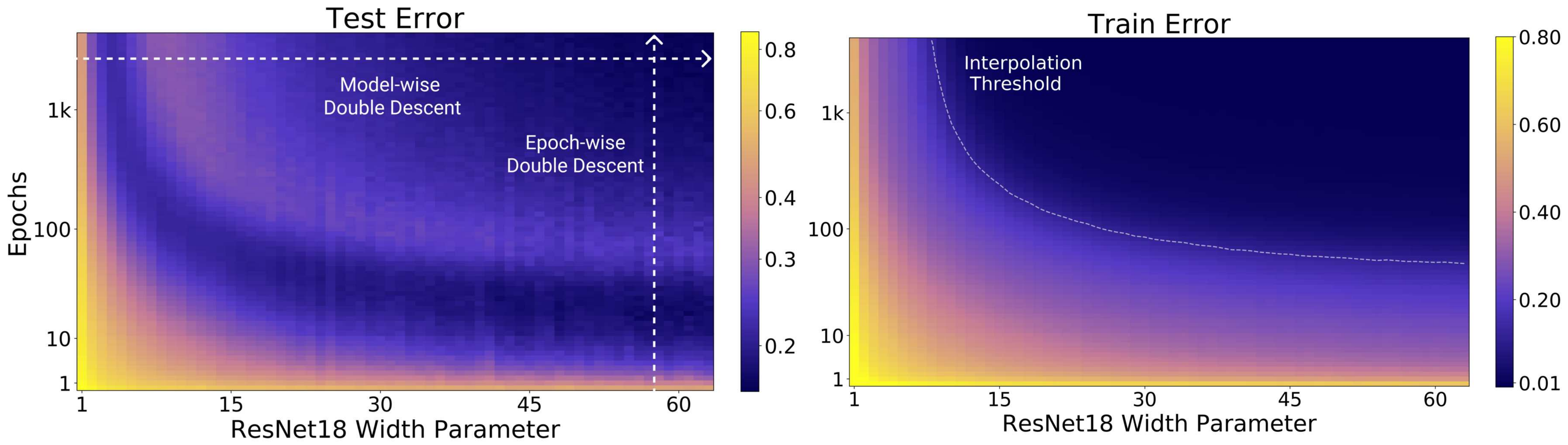
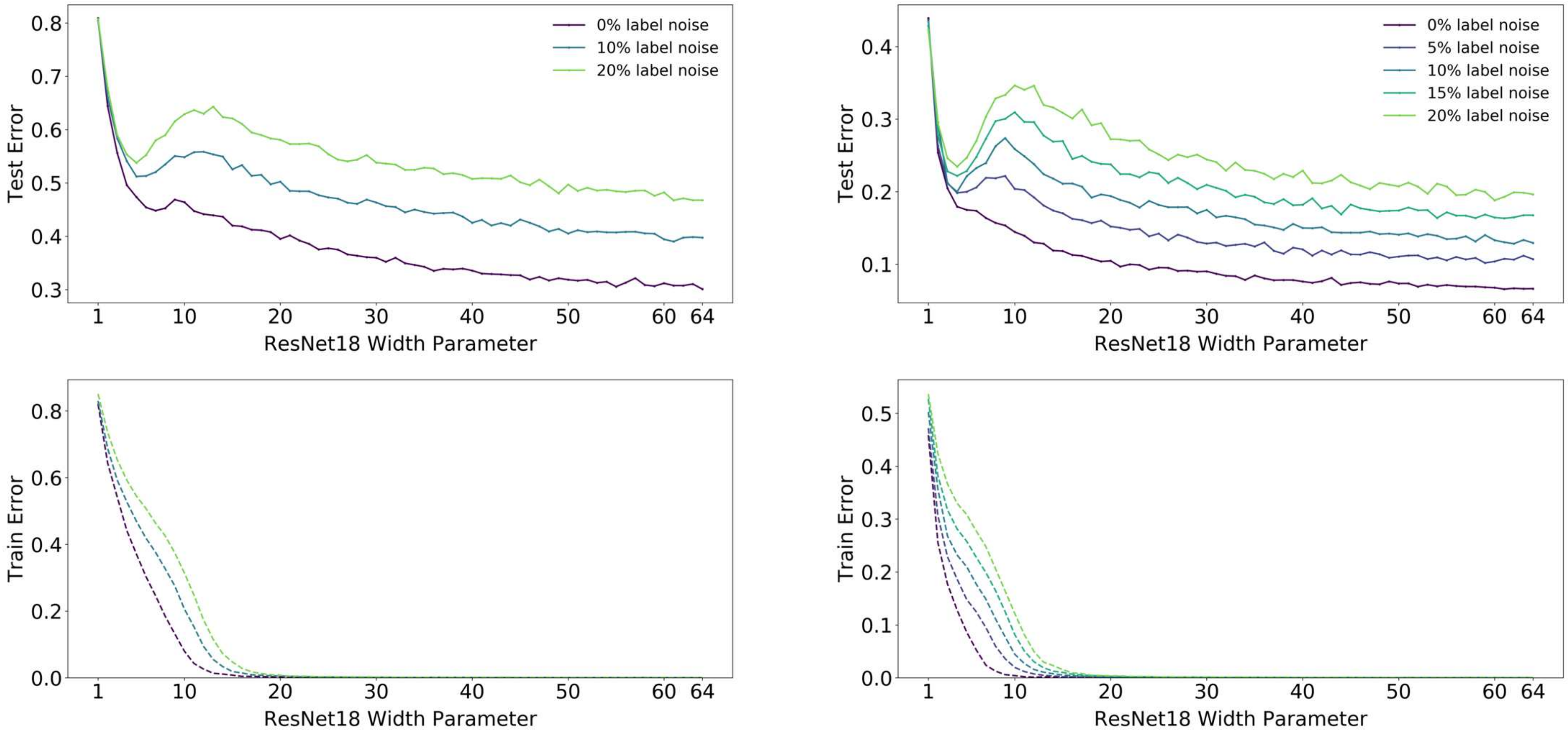


Figure 2: **Left:** Test error as a function of model size and train epochs. The horizontal line corresponds to model-wise double descent–varying model size while training for as long as possible. The vertical line corresponds to epoch-wise double descent, with test error undergoing double-descent as train time increases. **Right** Train error of the corresponding models. All models are Resnet18s trained on CIFAR-10 with 15% label noise, data-augmentation, and Adam for up to 4K epochs.

Figure 2 of "Deep Double Descent: Where Bigger Models and More Data Hurt" by P. Nakkiran et al.

Deep Double Descent



(a) **CIFAR-100.** There is a peak in test error even with no label noise.

(b) **CIFAR-10.** There is a “plateau” in test error around the interpolation point with no label noise, which develops into a peak for added label noise.

Figure 4 of "Deep Double Descent: Where Bigger Models and More Data Hurt" by P. Nakkiran et al.