Statistical Physics and Machine learning 101, (part 2)

Florent Krzakala

Institut Universitaire de France

Empirical Risk Minimisation

$$(\mathbf{X}_i \in \mathbb{R}^d, y_i \in \mathbb{R}), i = 1, \ldots, n \qquad \mathscr{R} = \frac{1}{n} \sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{X}_i))$$

Ex: linear models

Model: $f_\theta(\mathbf{X}) = \theta \cdot \mathbf{X}$

Loss : $\mathscr{L}(y, h)$ **Square Loss** $L(f(\vec{x}), y) = (1 - yf(\vec{x}))^2$

Logistic loss/Cross-entropy $L(f(\vec{x}), y) = \frac{1}{\ln 2} \ln(1 + e^{-yf(\vec{x})})$

Empirical Risk Minimisation

$$(\mathbf{X}_i \in \mathbb{R}^d, y_i \in \mathbb{R}), i = 1, \ldots, n \qquad \mathscr{R} = \frac{1}{n} \sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{X}_i))$$

Ex: neural networks models

Model: $$f_\theta(\mathbf{X}) = \eta^{(0)} \left(\mathbf{W}^{(0)} \eta^{(1)} \left(\mathbf{W}^{(1)} \ldots \eta^{(L)} \left(\mathbf{W}^{(L)} \cdot \mathbf{X} \right) \right) \right)$$

$$\theta = \{\mathbf{W}^{(0)}, \mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}\}$$

Loss : $\mathscr{L}(y, h)$ **Square Loss** $L(f(\vec{x}), y) = (1 - yf(\vec{x}))^2$

Logistic loss/Cross-entropy $L(f(\vec{x}), y) = \frac{1}{\ln 2} \ln(1 + e^{-yf(\vec{x})})$

Statistical learning 101

Supervised Binary classification

- Dataset, *m* examples $\{y^{(\mu)}, \vec{x}^{(\mu)}\}_{\mu=1}^{m}$
- Function class $f_{\vec{w}} \in \mathcal{F}$

Rademacher complexity

Theorem: Uniform convergence

With probability at least 1-δ,

$$\forall f_{\vec{w}} \in \mathcal{F}, \quad \epsilon_{\text{gen}}(f_{\vec{w}}) - \epsilon_{\text{train}}^{m}(f_{\vec{w}}) \leq \mathfrak{R}_m(\mathcal{F}) + \sqrt{\frac{\log(1/\delta)}{m}}$$

Generalization error **Train error**

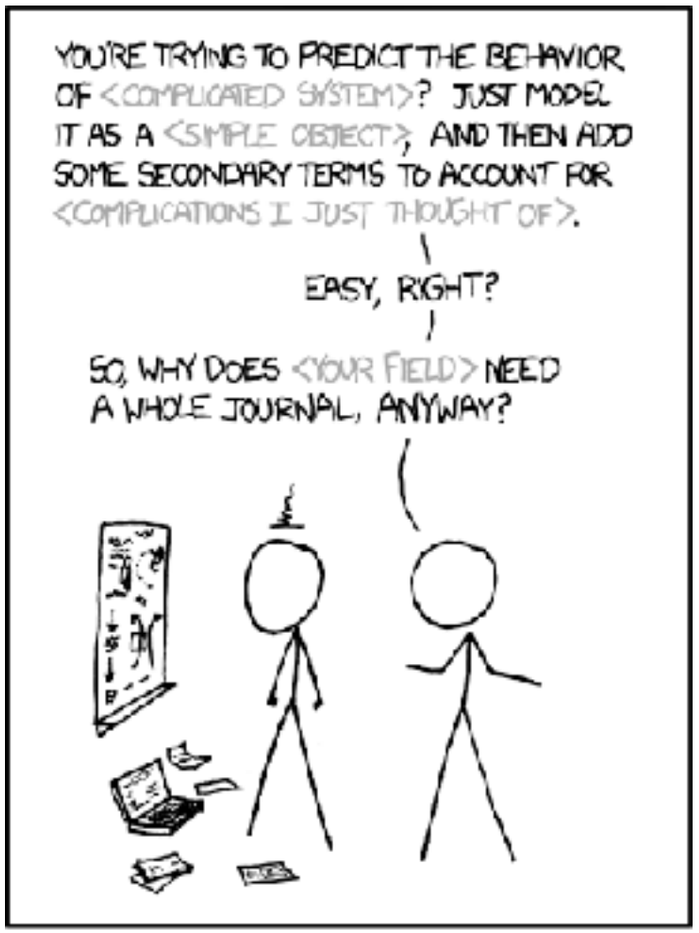
Classical result

Rademacher Complexity is bounded by VC dimension for some constant C

$$\mathfrak{R}_m(\mathcal{F}) \leq C\sqrt{\frac{d_{\text{VC}}(\mathcal{F})}{m}}$$

Physicists like models

Instead of worst case analysis, we could instead study models of data



credit: XKCD

Teacher - Student Framework

[P. Carnevali & S. Patarnello (1987)
N. Tishby, E. Levin, & S. Solla (1989)
E. Gardner, B. Derrida (1989)]

1

Results on Linear Regressions Physics-style

A simple teacher-student model

Teacher - Student Framework

[P. Carnevali & S. Patarnello (1987)
N. Tishby, E. Levin, & S. Solla (1989)
E. Gardner, B. Derrida (1989)]

Simplest version

⭐ ***Data*** $\begin{cases} \vec{x}^{(\mu)} \in \mathbb{R}^d, \mu = 1 \dots m \\ P_X(\vec{x}) = \mathcal{N}(0, \mathbf{1}_d) \end{cases}$

⭐ ***Labels*** $\begin{cases} y_\mu \equiv \text{sign}(\vec{x}_\mu \cdot \vec{W}^*) \\ \vec{W}^* \sim \mathcal{N}(0, \mathbf{1}_d) \end{cases}$

High-dimensional limit n, d $\to \infty$,
with $\alpha = n/d$ fixed

Physics literature

[…, Opper Kinzel ’90, Kleinz, Seung ‘90, Opper, Haussler ’91, Seung Sompolinsky, Tishby ’92, Watkin Rau & Biehl ’93, Opper, Kinzel ’95,…]

Rigorous proofs

[…., Barbier, **FK**, Macris, Miolane, Zdeborova ’17
Trampoulidis, Abbasi, Hassbi ’18
Montanari, Ruan, Sohn, Yan ’20
Aubin, **FK**, Lu, Zdeborova ’20,
Gerbelot, Abbara, **FK** ‘20….]

Rigorous solution

For binary classification ($y \in \pm 1$), the generalization error of ERM is given by

$$e_g^{\text{erm}}(\alpha) = \frac{1}{\pi} acos(\sqrt{\eta}) \quad \text{with} \quad \eta \equiv \frac{m^2}{\rho_{w^\star} q}$$

and $\rho_{w^\star} \equiv \frac{1}{d}\mathbb{E}\left[\|\mathbf{w}^\star\|_2^2\right]$.

Theorem (Gordon's minimax - ℓ_2 - classification)

As $n, d \to \infty$ with $n/d = \alpha = \Theta(1)$, for ℓ_2 regularization $r(w) = \frac{\lambda}{2}w^2$:

$$m \underset{d\to\infty}{\longrightarrow} \sqrt{\rho_{w^\star}}\mu^*, \qquad q \underset{d\to\infty}{\longrightarrow} (\mu^*)^2 + (\delta^*)^2, \qquad \Sigma \underset{d\to\infty}{\longrightarrow} \tau^\star$$

where parameters μ^* and δ^* are solutions of

$$(\mu^*, \delta^*) = \underset{\mu,\delta\geq 0}{\arg\min} \sup_{\tau>0} \left\{ \frac{\lambda(\mu^2+\delta^2)}{2} - \frac{\delta^2}{2\tau} + \alpha \mathbb{E}_{g,s} \mathcal{M}_\tau[\delta g + \mu s y] \right\},$$

The saddle point equations yield

$$\Rightarrow \begin{cases} \mu^* & = \frac{\alpha}{\lambda\tau^* + \alpha}\mathbb{E}_{g,s}[s \cdot y \cdot \mathcal{P}_{\tau^*}(\delta^* g + \mu^* s y)], \\ \delta^* & = \frac{\alpha}{\lambda\tau^* + \alpha - 1}\mathbb{E}_{g,s}[g \cdot \mathcal{P}_{\tau^*}(\delta^* g + \mu^* s y)], \\ (\delta^*)^2 & = \alpha \mathbb{E}_{g,s}[((\delta^* g + \mu^* s y) - \mathcal{P}_{\tau^*}(\delta^* g + \mu^* s y))^2] \end{cases}$$

with $y = \varphi_{\text{out}^\star}(\sqrt{\rho_{w^\star}} s)$, $\mathcal{M}$ and $\mathcal{P}$ the Moreau-Yosida regularization and the proximal map.

[Aubin, **FK**, Lu, Zdeborova '20]

Generalization error in high-dimensional perceptrons: Approaching Bayes error with convex optimization

Benjamin Aubin[†], Florent Krzakala[*], Yue M. Lu[°], Lenka Zdeborová[†]

[†] *Université Paris-Saclay, CNRS, CEA, Institut de physique théorique, 91191, Gif-sur-Yvette, France.*

[*] *Laboratoire de Physique Statistique, CNRS & Sorbonnes Universités, École Normale Supérieure, PSL University, Paris, France.*

[°] *John A. Paulson School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138, USA*

Abstract

We consider a commonly studied supervised classification of a synthetic dataset whose labels are generated by feeding a one-layer neural network with random iid inputs. We study the generalization performances of standard classifiers in the high-dimensional regime where $\alpha = n/d$ is kept finite in the limit of a high dimension d and number of samples n. Our contribution is three-fold: First, we prove a formula for the generalization error achieved by ℓ_2 regularized classifiers that minimize a convex loss. This formula was first obtained by the heuristic replica method of statistical physics. Secondly, focussing on commonly used loss functions and optimizing the ℓ_2 regularization strength, we observe that while ridge regression performance is poor, logistic and hinge regression are surprisingly able to approach the Bayes-optimal generalization error extremely closely. As $\alpha \to \infty$ they lead to Bayes-optimal rates, a fact that does not follow from predictions of margin-based generalization error bounds. Third, we design an optimal loss and regularizer that provably leads to Bayes-optimal generalization error.

L2 loss

$$f_\theta(\mathbf{X}) = \theta \cdot \mathbf{X}$$

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \|y_i - \theta X_i\|_2^2$$

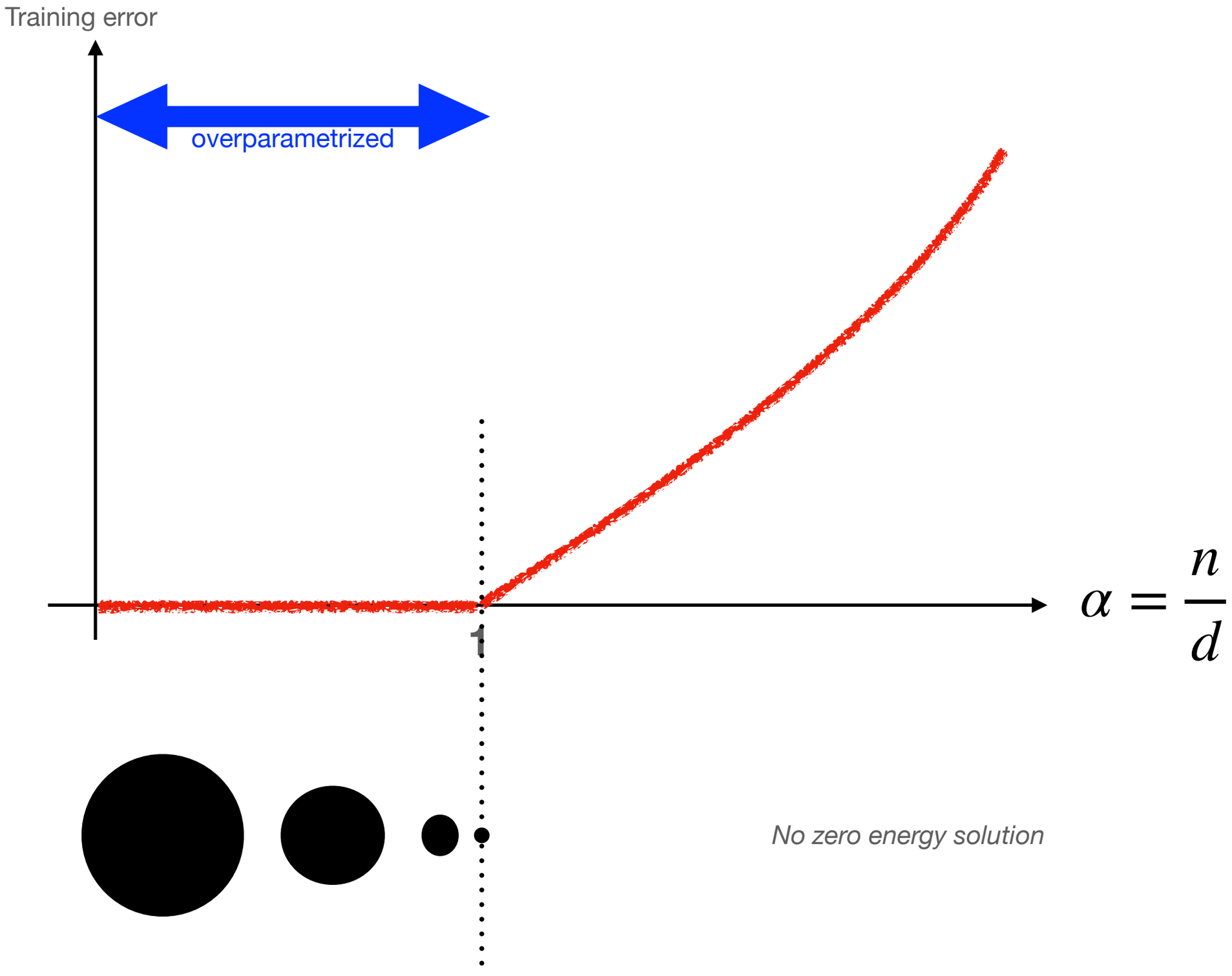
Simplest version

⭐ ***Data*** $\begin{cases} \overrightarrow{x}^{(\mu)} \in \mathbb{R}^d, \mu = 1 \ldots m \\ P_X(\overrightarrow{x}) = \mathcal{N}(0, \mathbf{1}_d) \end{cases}$

⭐ ***Labels*** $\begin{cases} y_\mu \equiv \text{sign}(\overrightarrow{\text{x}}_\mu \cdot \overrightarrow{\text{W}}^*) \\ \overrightarrow{W}^* \sim \mathcal{N}(0, \mathbf{1}_d) \end{cases}$

High-dimensional limit n, d $\to \infty$,
with $\alpha = n/d$ fixed

L2 loss: what to expect?



Ordinary least square

$$\hat{\theta} = \text{argmin}(||\mathbf{Y} - \mathrm{A}\theta||_2^2)$$

$$||\mathbf{Y} - A\theta||_2^2 = (\mathbf{Y} - A\theta)^T(\mathbf{Y} - A\theta) = \mathbf{Y}^T\mathbf{Y} + \theta^T A^T A\theta - 2\mathbf{Y}^T A\theta$$

Taking the extremum yields the normal equations:

d x d d x n

$$A^T A\theta = A^T\mathbf{Y}$$

d x 1 n x 1

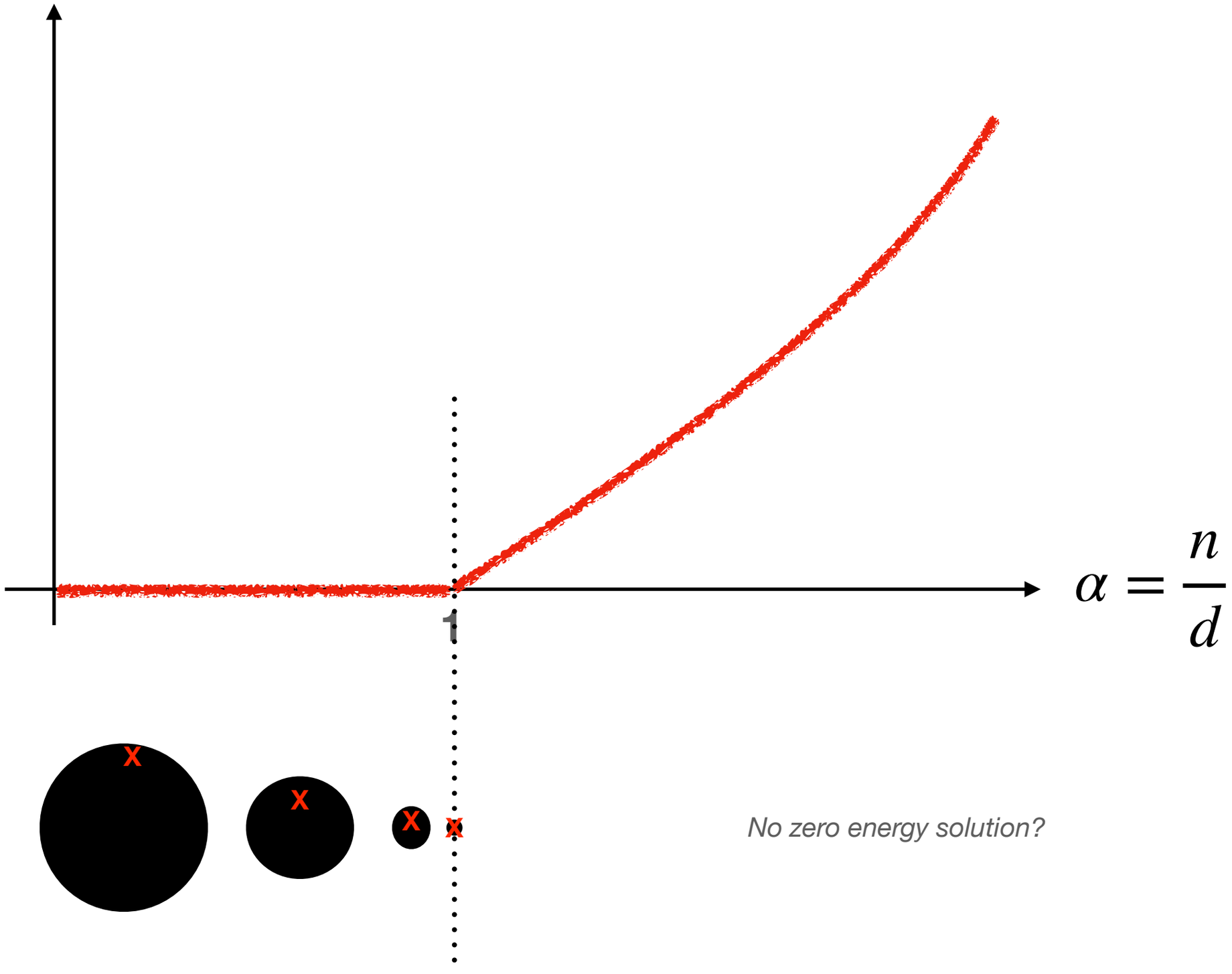
Unique solution if A^TA is full rank
This requires (*at least*) n>p
(no more unknown than datapoints)

$$\hat{\theta} = (A^T A)^{-1} A^T\mathbf{Y}$$

Otherwise, many solutions may exists.
A popular choice leads
to the least-norm (*in l2 norm*) solution:

$$\hat{\theta}_{\text{ln}} = A^T(AA^T)^{-1}\mathbf{Y}$$

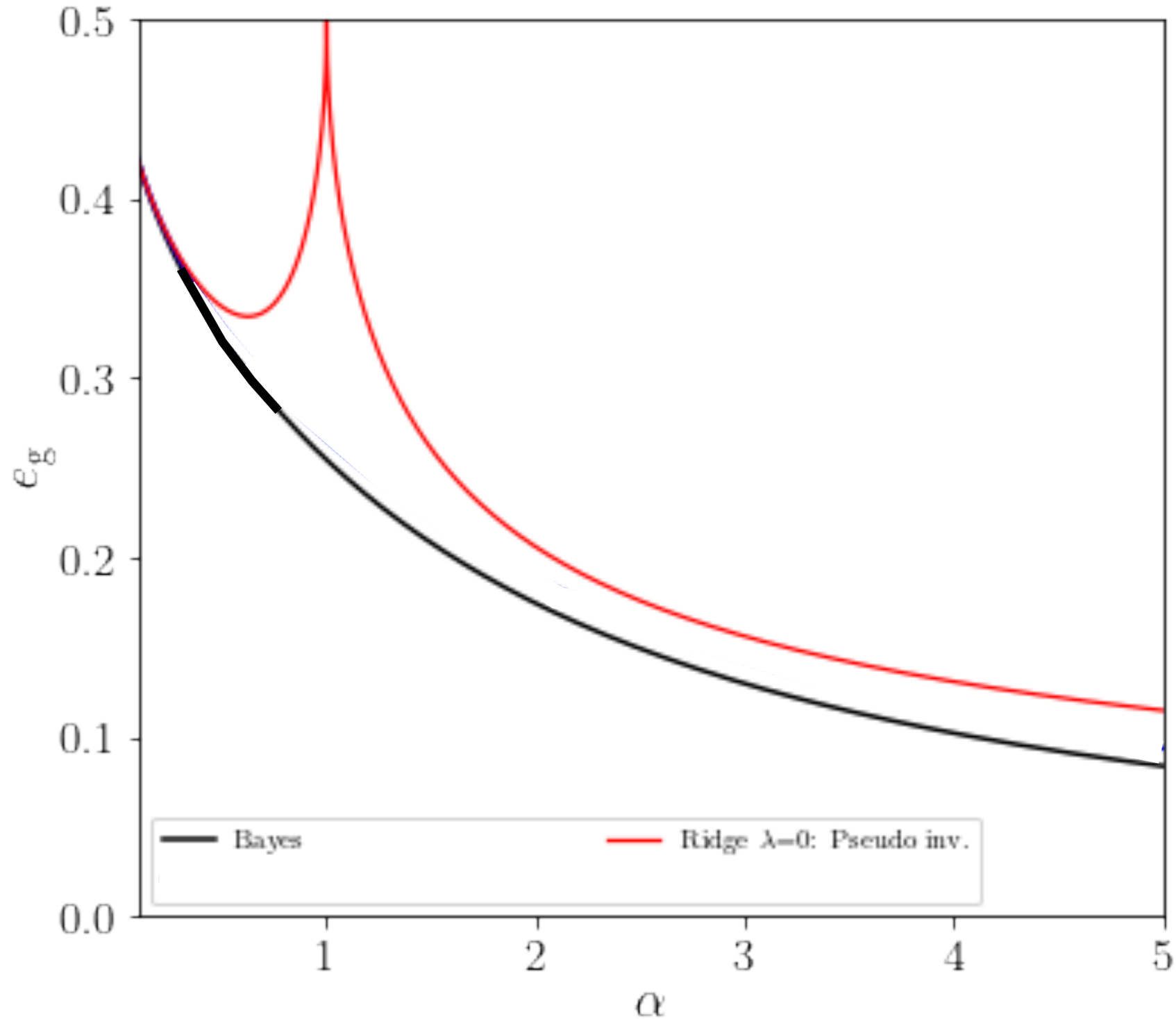
Least norm solution



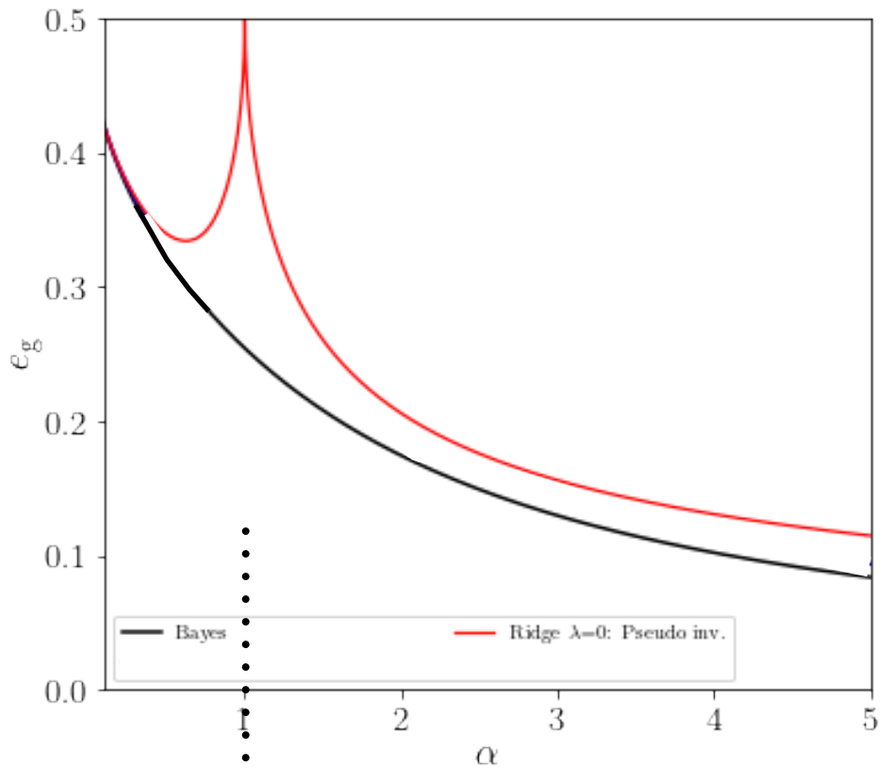
Zero energy Solutions

No zero energy solution?

Least norm solution: “double descent”

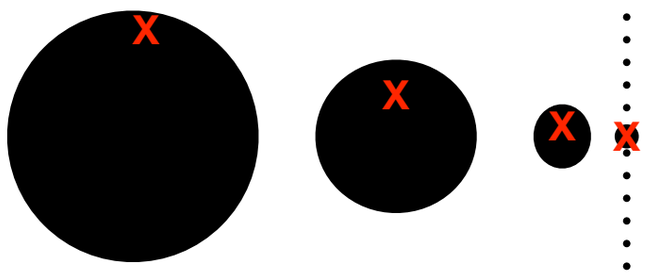


Least norm solution: “double descent”



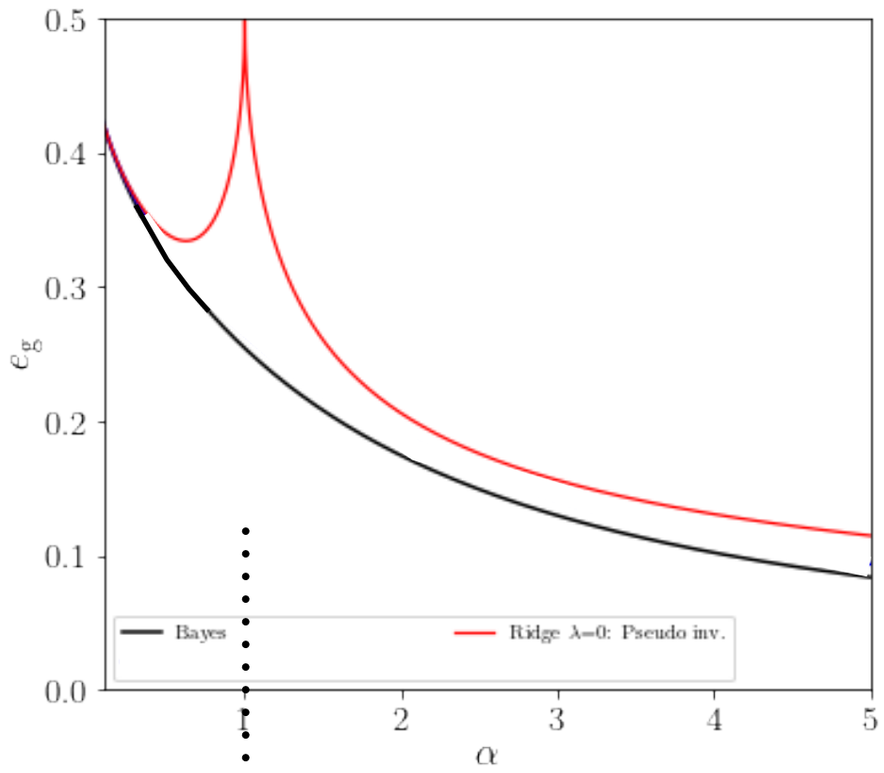
Biasing to low l2 norm solutions is good

Zero energy Solutions



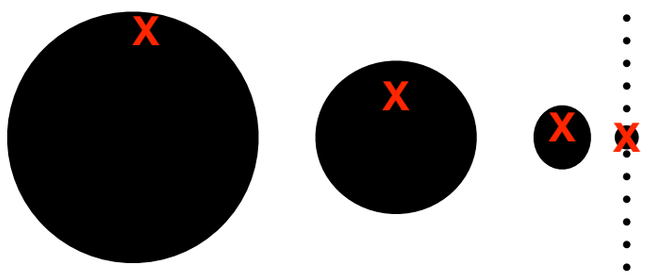
No zero energy solution?

What if I do gradient descent?



Biasing to low l2 norm solutions is good

Zero energy Solutions



No zero energy solution?

Linear models

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{X}_i))$$

$$f_\theta(\mathbf{X}) = \theta \cdot \mathbf{X}$$

Gradient descent

$$\theta \in \mathbb{R}^d$$

Gradient flow

$$\theta^t = \theta^{t-1} - \eta \nabla_\theta \mathscr{R}$$

$$\dot{\theta}^t = -\nabla_\theta \mathscr{R}$$

Representer theorem

For any loss function such that

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n}\mathscr{L}(y_i, \theta \cdot \mathbf{x}_i)$$

$$\hat{\theta} = \text{argmin}\mathcal{R}(\theta)$$

We can always write the minimiser as:

$$\hat{\theta} = \sum_{i=1}^{n}\beta_i \mathbf{X}_i$$

Proof

$$\hat{\theta} \in \mathbb{R}^d \qquad \mathbf{X}_i \in \mathbb{R}^d \forall i \qquad \mathbb{R}^d = \text{span}(\{X\}) + \text{null}(\{X\})$$

So we can write:

$$\hat{\theta} = \sum_{i=1}^{n}\beta_i \mathbf{X}_i + \overrightarrow{\mathscr{N}}$$

But:

$$\hat{\theta} \cdot \mathbf{X}_j = \left(\sum_{i=1}^{n}\beta_i \mathbf{X})_i\right) \cdot \mathbf{X}_j + \overrightarrow{\mathscr{N}} \cdot \mathbf{X}_j = \left(\sum_{i=1}^{n}\beta_i \mathbf{X})_i\right) \cdot \mathbf{X}_j$$

So we might as well write:

$$\hat{\theta} = \sum_{i=1}^{n}\beta_i \mathbf{X}_i \quad \blacksquare$$

Linear models

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{X}_i))$$

$$f_\theta(\mathbf{X}) = \theta \cdot \mathbf{X}$$

Gradient descent

$$\theta \in \mathbb{R}^d$$

Gradient flow

$$\theta^t = \theta^{t-1} - \eta \nabla_\theta \mathscr{R}$$

$$\dot{\theta}^t = -\nabla_\theta \mathscr{R}$$

Linear models, dual…

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\beta(\mathbf{X}_i))$$

$$f_\beta(\mathbf{X}) = \sum_{j=1}^{n} \beta_j \mathbf{X}_j \cdot \mathbf{X}$$

Gradient descent

$$\beta \in \mathbb{R}^n$$

Gradient flow

$$\beta^t = \beta^{t-1} - \eta \nabla_\beta \mathscr{R}$$

$$\dot{\beta}^t = -\nabla_\beta \mathscr{R}$$

A lesson from representer theorem

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n}\mathscr{L}(y_i, \theta \cdot \mathbf{x}_i)$$

$$\mathbb{R}^d = \text{span}(\{X\}) + \text{null}(\{X\})$$

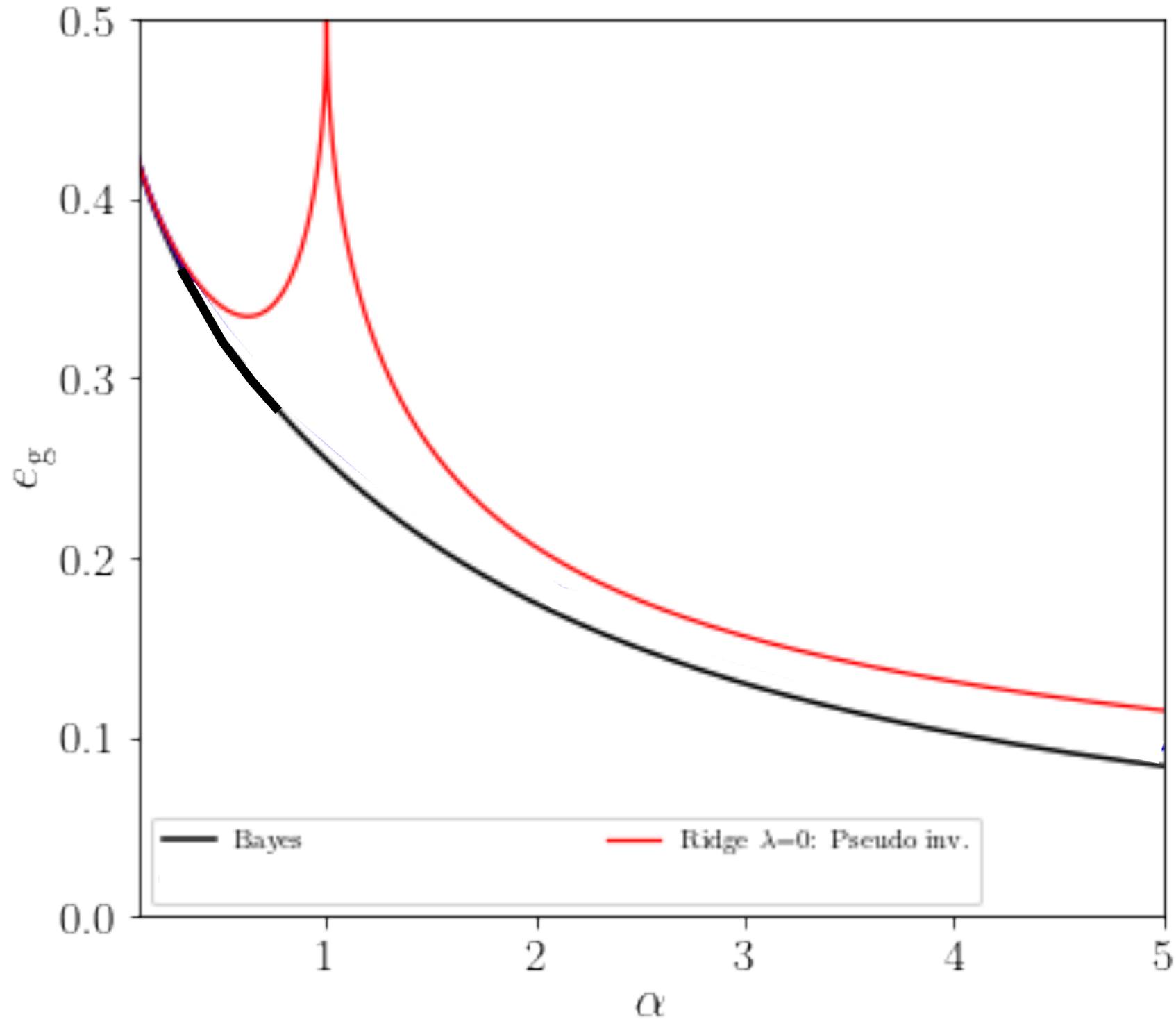
$$\theta = \sum_{i=1}^{n}\beta_i \mathbf{X}_i + \overrightarrow{\mathscr{N}}$$

If you do gradient descent, $\overrightarrow{\mathscr{N}}$ is never updated!

Solution(s):

1. Use Representer theorem (equivalent to least norm solution)
2. Initizalize weight close to zero [Saxe, Advani ‘17]

Least norm solution: non monotonous generalisation



Ridge loss

$$f_\theta(\mathbf{X}) = \theta \cdot \mathbf{X}$$

Simplest version

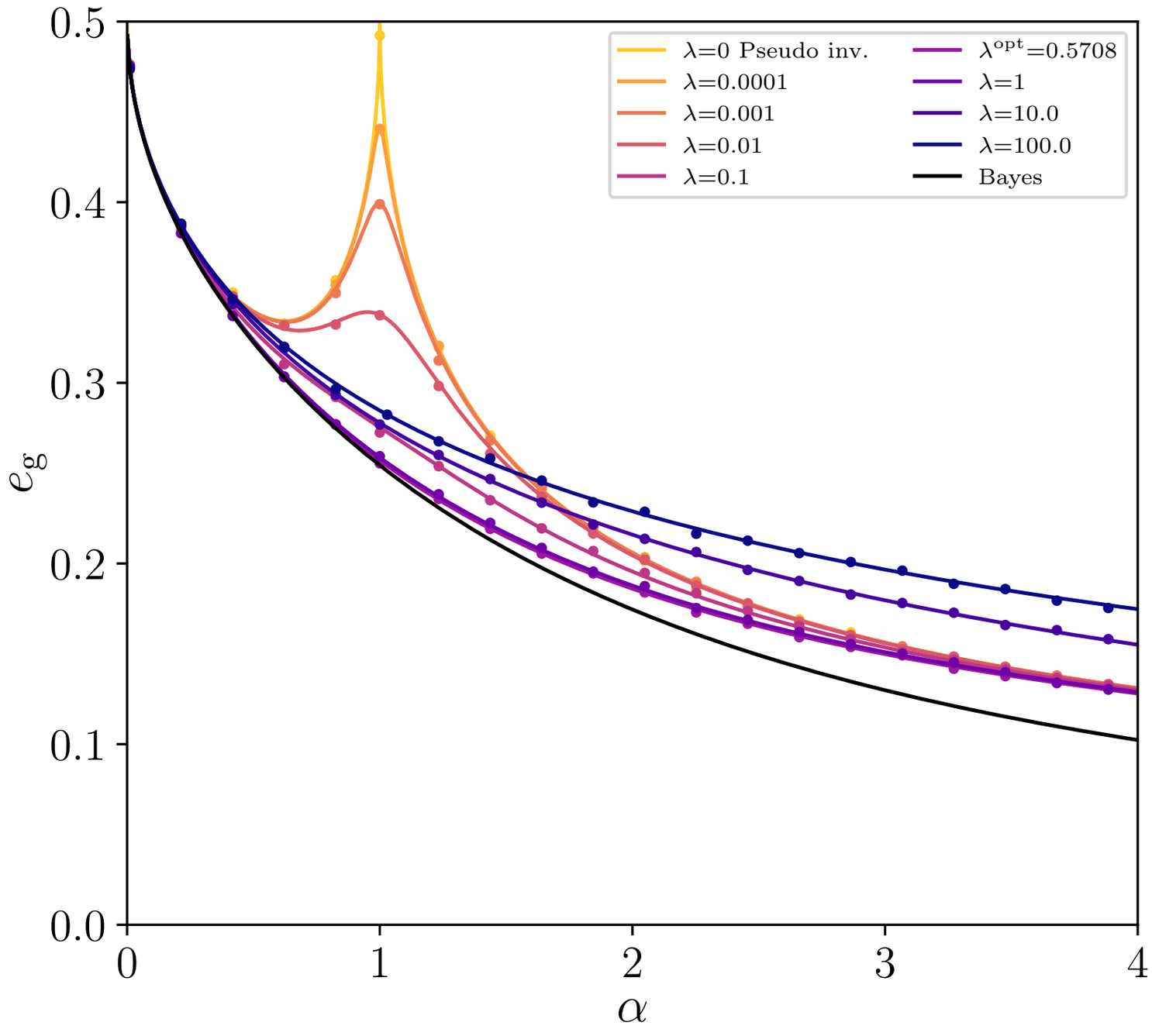
⭐ ***Data*** $\begin{cases} \overrightarrow{x}^{(\mu)} \in \mathbb{R}^d, \mu = 1 \ldots m \\ P_X(\overrightarrow{x}) = \mathcal{N}(0, \mathbf{1}_d) \end{cases}$

⭐ ***Labels*** $\begin{cases} y_\mu \equiv \text{sign}(\overrightarrow{\text{x}}_\mu \cdot \overrightarrow{\text{W}}^*) \\ \overrightarrow{W}^* \sim \mathcal{N}(0, \mathbf{1}_d) \end{cases}$

High-dimensional limit n, d $\to \infty$,
with $\alpha = n/d$ fixed

$$\mathscr{R} = \frac{1}{n} \sum_{i=1}^{n} \|y_i - \theta X_i\|_2^2 + \lambda \|\theta\|_2^2$$

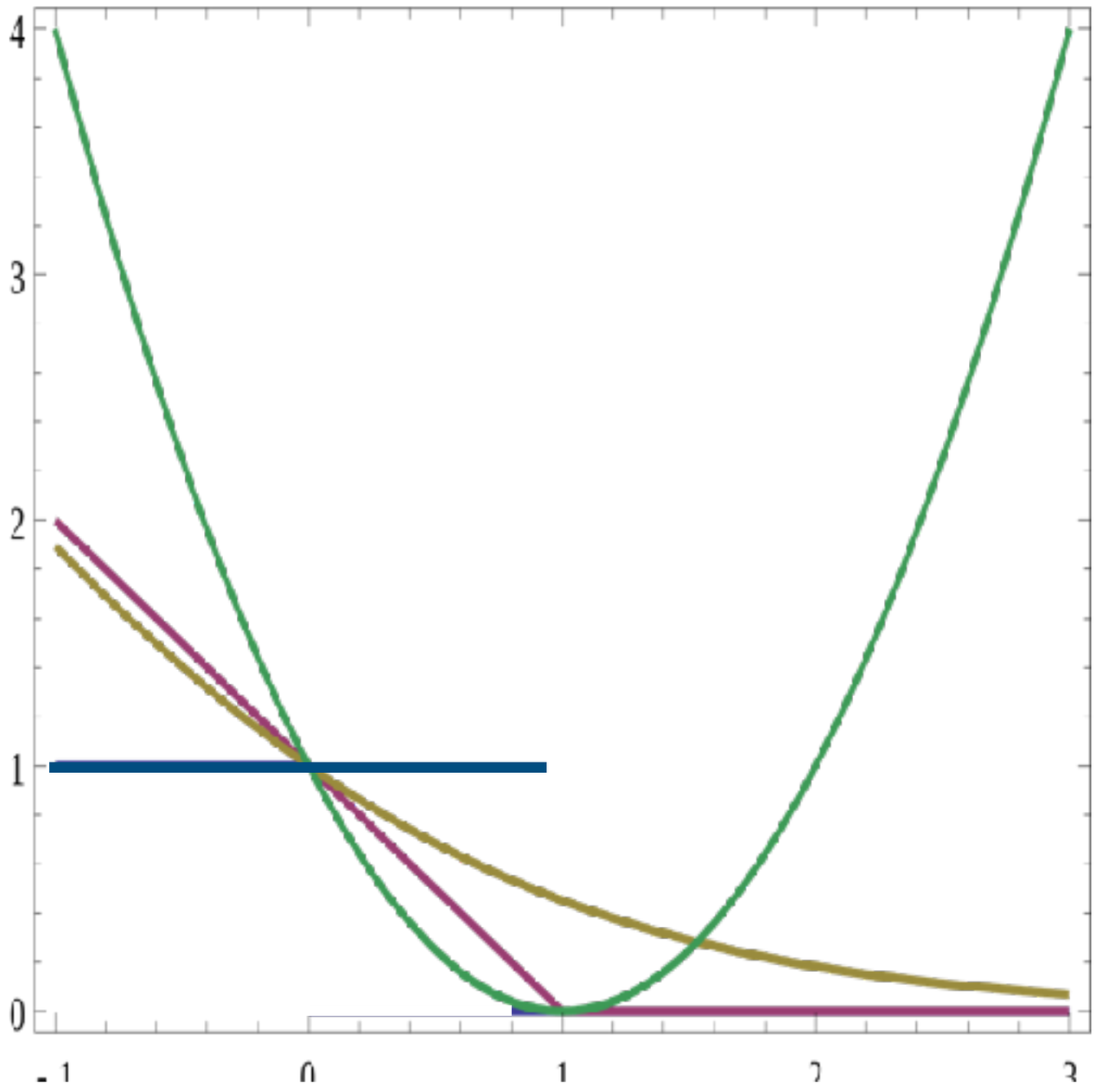
Ridge loss



[Aubin, Krzakala, Lu, Zdeborova'20]

Cover you Losses!

$$f(\vec{x}) = \theta \cdot \vec{x} + \alpha$$



Square Loss

$$L(f(\vec{x}), y) = (1 - yf(\vec{x}))^2$$

Hard margin

$$L(f(\vec{x}), y) = \mathbf{1}(yf(\vec{x}) > 1)$$

Hinge loss

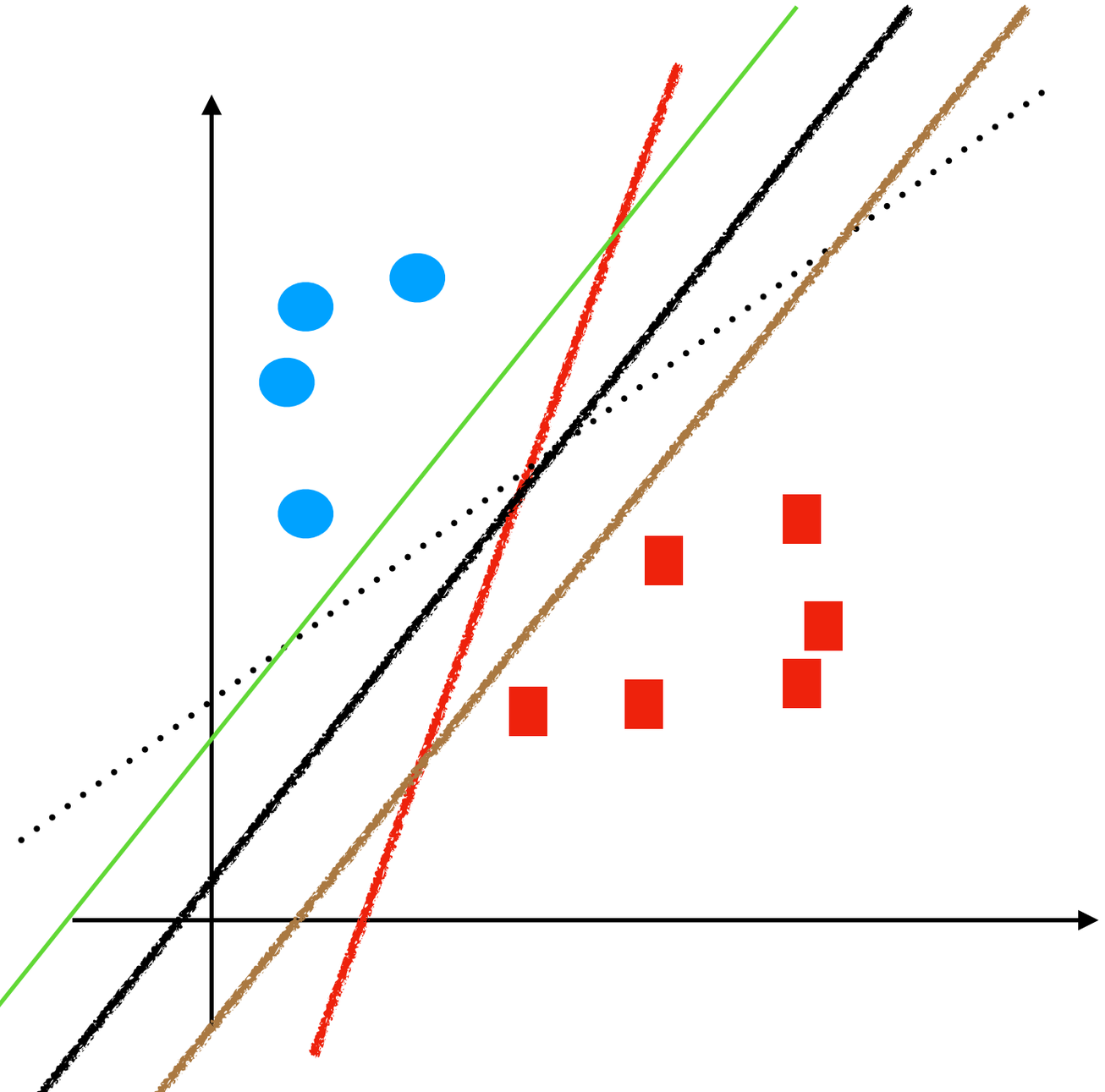
$$L(f(\vec{x}), y) = \max(0, 1 - yf(\vec{x}))$$

Logistic loss/Cross-entropy

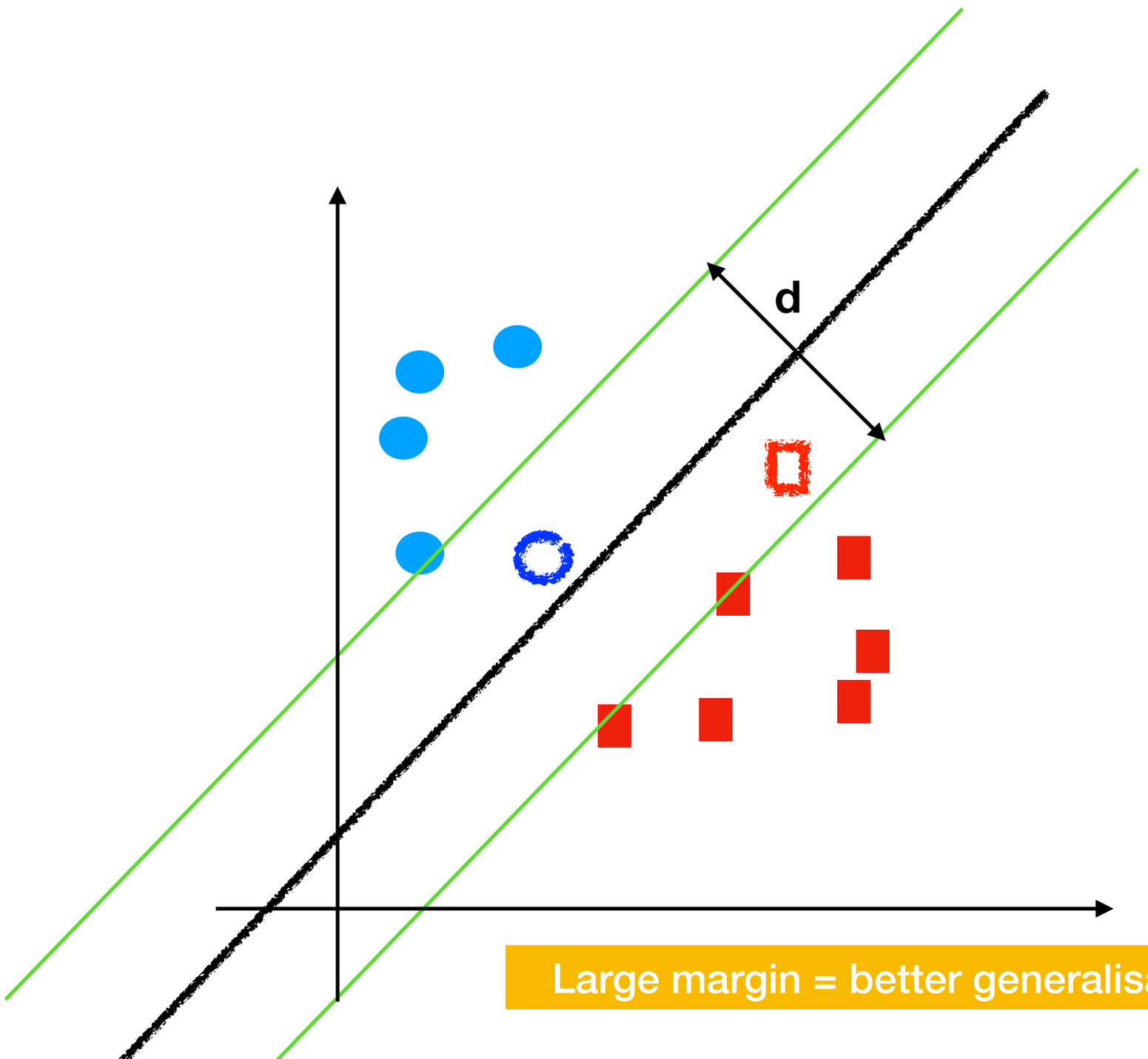
$$L(f(\vec{x}), y) = \frac{1}{\ln 2} \ln(1 + e^{-yf(\vec{x})})$$

Pushing the boundaries

Which frontier should we choose?



Pushing the boundaries



Large margin = better generalisation properties!

History!

comments

By Isabelle Guyon.

The invention of Support Vector Machines using the kernel trick in 1991 is my most important contribution.

I had the chance of working in the 1990's in Larry Jackel's group at Bell Labs with great people, some of whom became famous (Vladimir Vapnik, Yann LeCun, Leon Bottou, Yoshua Bengio, and Corinna Cortes). This was a very stimulating and competitive environment.

The story starts in Paris in 1989, when I benchmarked neural networks against kernel methods (Parzen windows, potential functions, and least-square kernel regression from Tommy Poggio's work) with my PhD advisors Gerard Dreyfus and Leon Personnaz. The same year, two physicists working close-by (Marc Mezard and Werner Krauth) published a paper on an optimal margin algorithm called 'minover,' which attracted my attention. But it was not until I joined Bell Labs that I put things together.

Support Vector Machines

L2 regularization and max-margin classification

$$f(\vec{x}) = \theta \cdot \vec{x} + \alpha$$

Hinge Loss

$$L(f(\vec{x}), y) = \max(0, 1 - yf(\vec{x}))$$

$$\theta \cdot \vec{x} + \alpha = 1$$

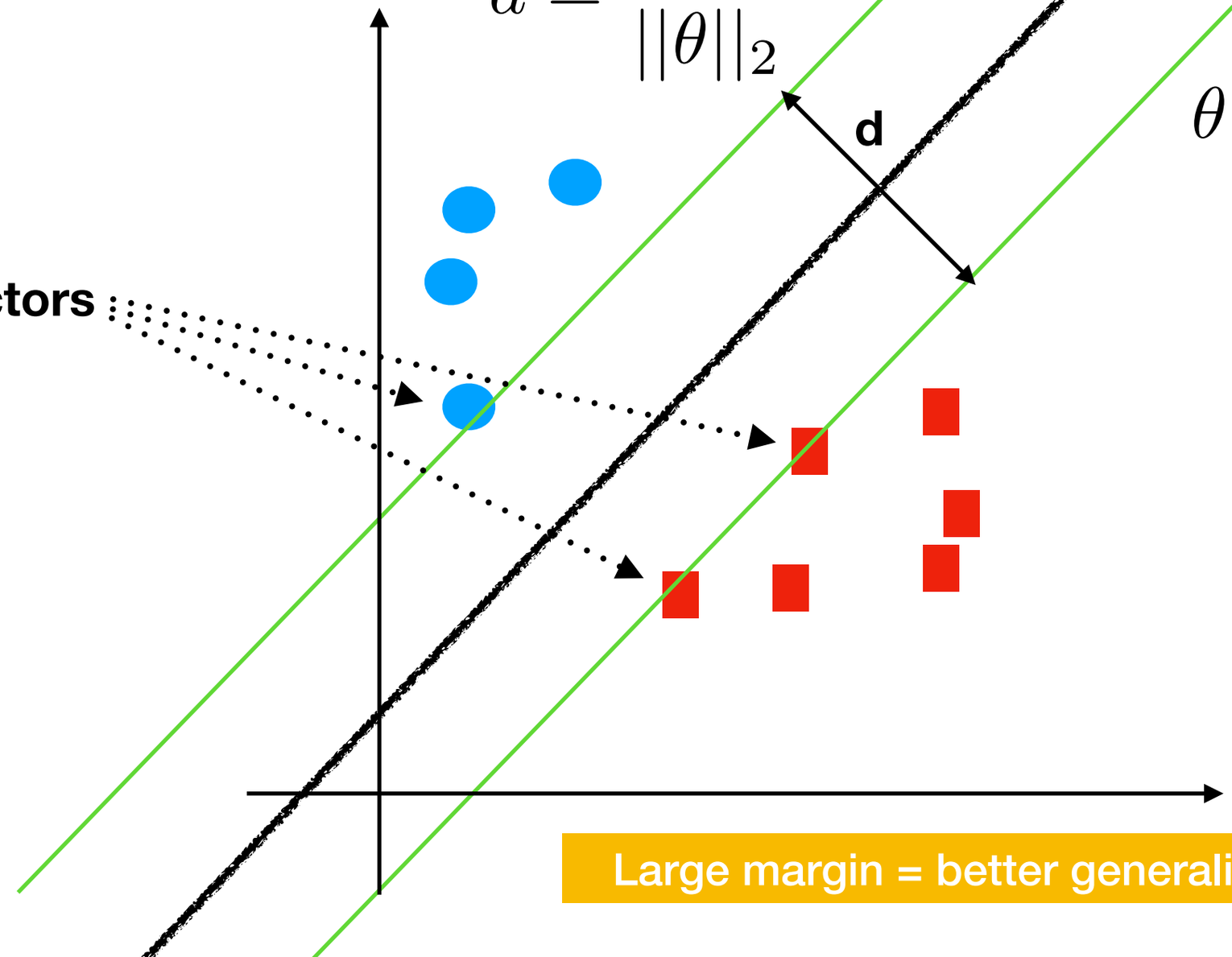
$$\theta \cdot \vec{x} + \alpha = 0$$

$$\theta \cdot \vec{x} + \alpha - 1$$

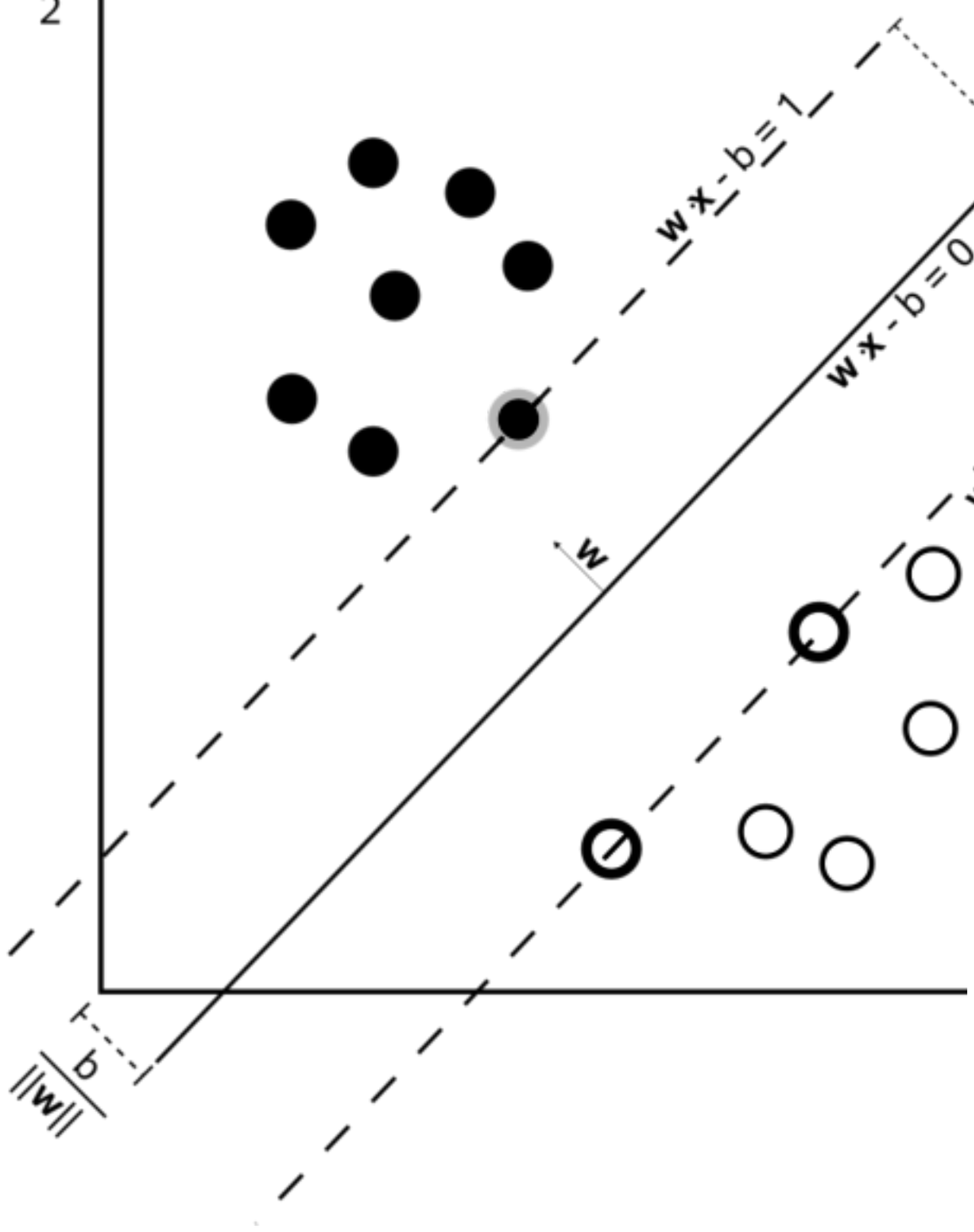
$$d = \frac{2}{||\theta||_2}$$

d

Support vectors



Large margin = better generalisation properties!



Let $\mathbf{x}_0$ be a point in the hyperplane $\mathbf{wx} - b = -1$, i.e., $\mathbf{wx}_0 - b = -1$. To measure the distance between hyperplanes $\mathbf{wx} - b = -1$ and $\mathbf{wx} - b = 1$, we only need to compute the perpendicular distance from $\mathbf{x}_0$ to plane $\mathbf{wx} - b = 1$, denoted as r.

Note that $\frac{\mathbf{w}}{\|\mathbf{w}\|}$ is a unit normal vector of the hyperplane $\mathbf{wx} - b = 1$. We have

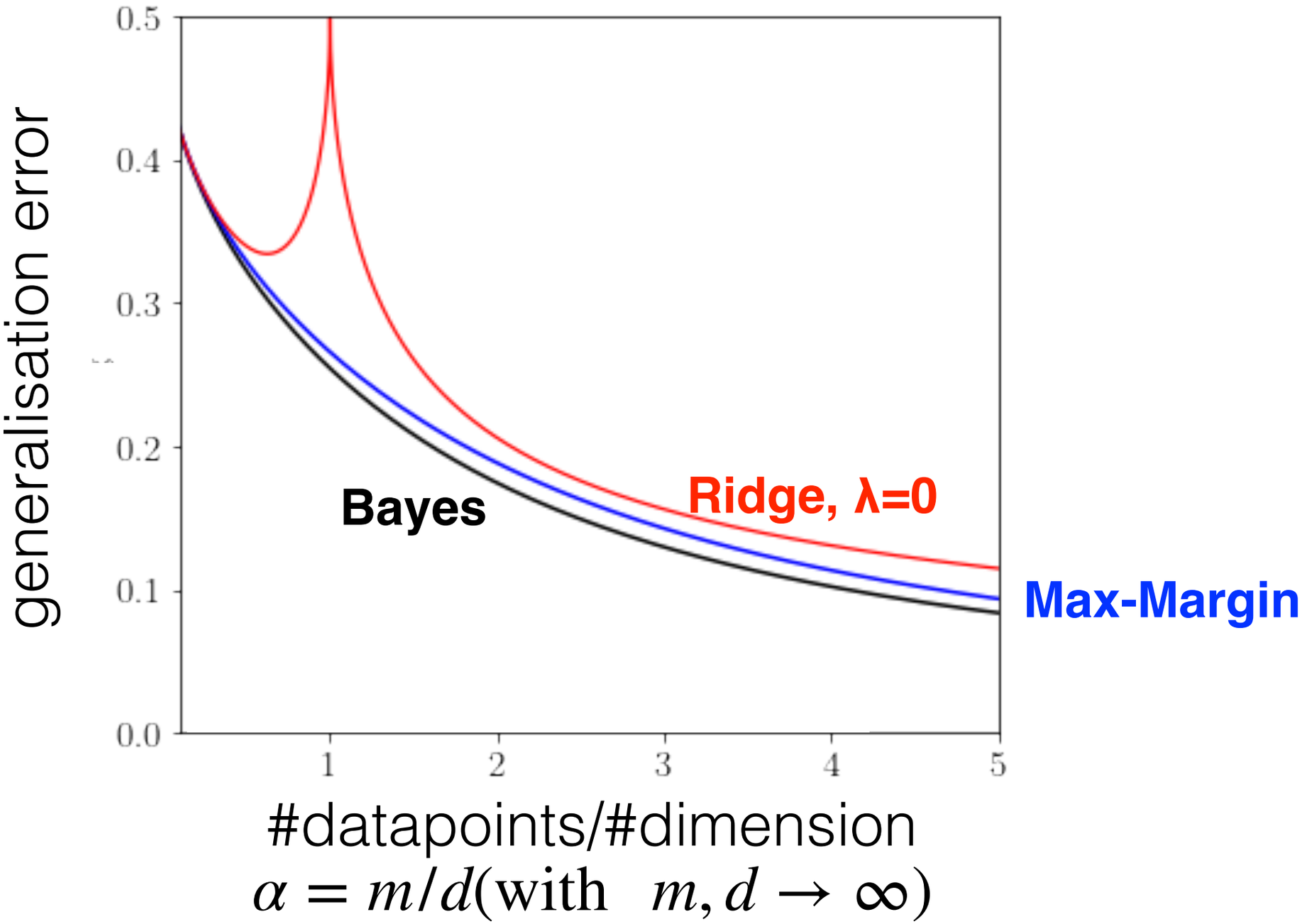
$$\mathbf{w}(\mathbf{x}_0 + r\frac{\mathbf{w}}{\|\mathbf{w}\|}) - b = 1$$

since $\mathbf{x}_0 + r\frac{\mathbf{w}}{\|\mathbf{w}\|}$ should be a point in hyperplane $\mathbf{wx} - b = 1$ according to our definition of r.

Expanding this equation, we have

$$\begin{aligned}
&\mathbf{wx}_0 + r\frac{\mathbf{ww}}{\|\mathbf{w}\|} - b = 1 \\
\Longrightarrow\ &\mathbf{wx}_0 + r\frac{\|\mathbf{w}\|^2}{\|\mathbf{w}\|} - b = 1 \\
\Longrightarrow\ &\mathbf{wx}_0 + r\|\mathbf{w}\| - b = 1 \\
\Longrightarrow\ &\mathbf{wx}_0 - b = 1 - r\|\mathbf{w}\| \\
\Longrightarrow\ &-1 = 1 - r\|\mathbf{w}\| \\
\Longrightarrow\ &r = \frac{2}{\|\mathbf{w}\|}
\end{aligned}$$

Max-margin is good



[Aubin, Krzakala, Lu, Zdeborova'20]

Implicit regularization

Margin Maximizing Loss Functions

2006

Saharon Rosset
Watson Research Center
IBM
Yorktown, NY, 10598
srosset@us.ibm.com

Ji Zhu
Department of Statistics
University of Michigan
Ann Arbor, MI, 48109
jizhu@umich.edu

Trevor Hastie
Department of Statistics
Stanford University
Stanford, CA, 94305
hastie@stat.stanford.edu

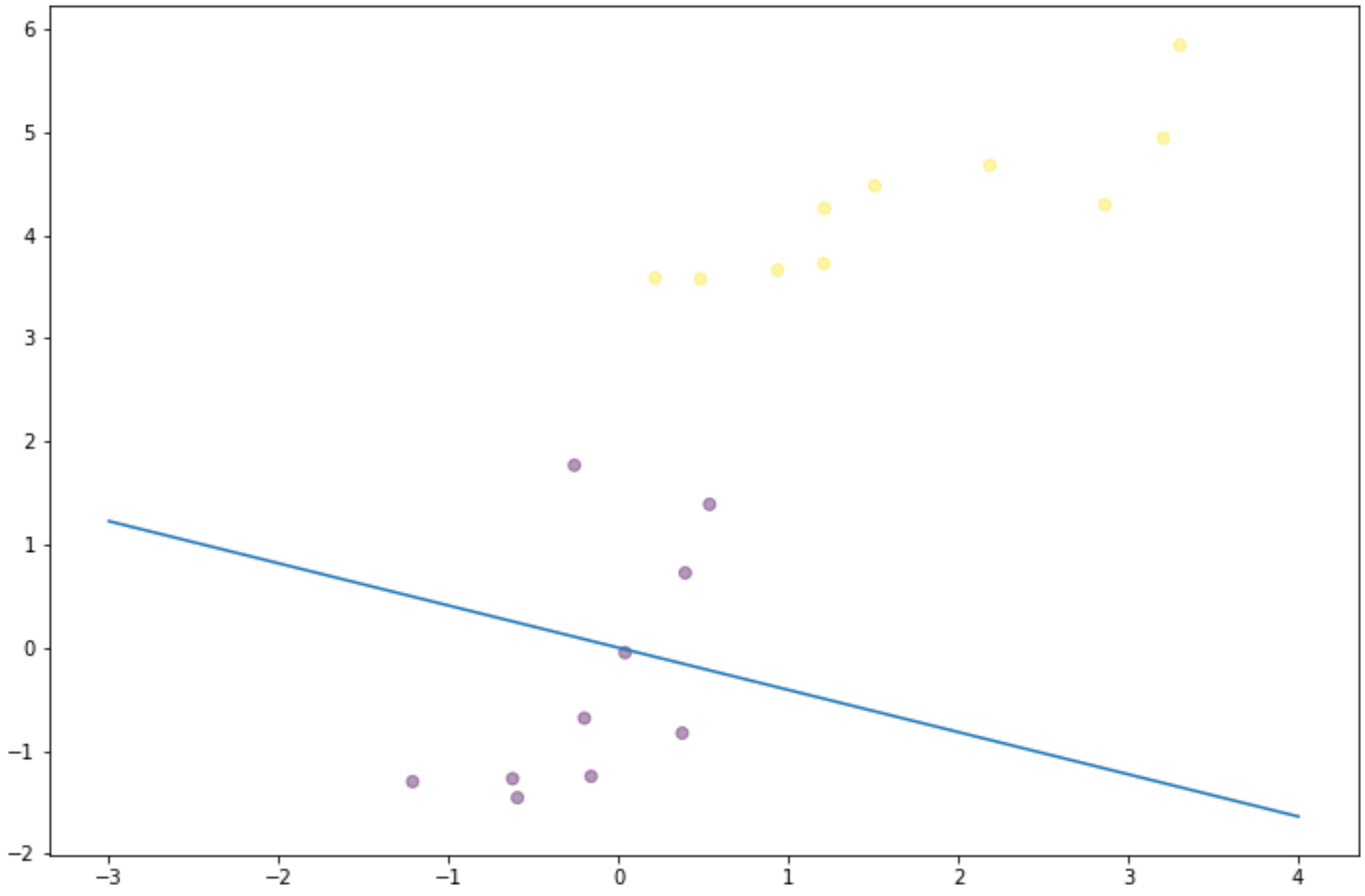
Abstract

Margin maximizing properties play an important role in the analysis of classi£-cation models, such as boosting and support vector machines. Margin maximization is theoretically interesting because it facilitates generalization error analysis, and practically interesting because it presents a clear geometric interpretation of the models being built. We formulate and prove a suf£cient condition for the solutions of regularized loss functions to converge to margin maximizing separators, as the regularization vanishes. This condition covers the hinge loss of SVM, the exponential loss of AdaBoost and logistic regression loss. We also generalize it to multi-class classi£cation problems, and present margin maximizing multi-class versions of logistic regression and support vector machines.

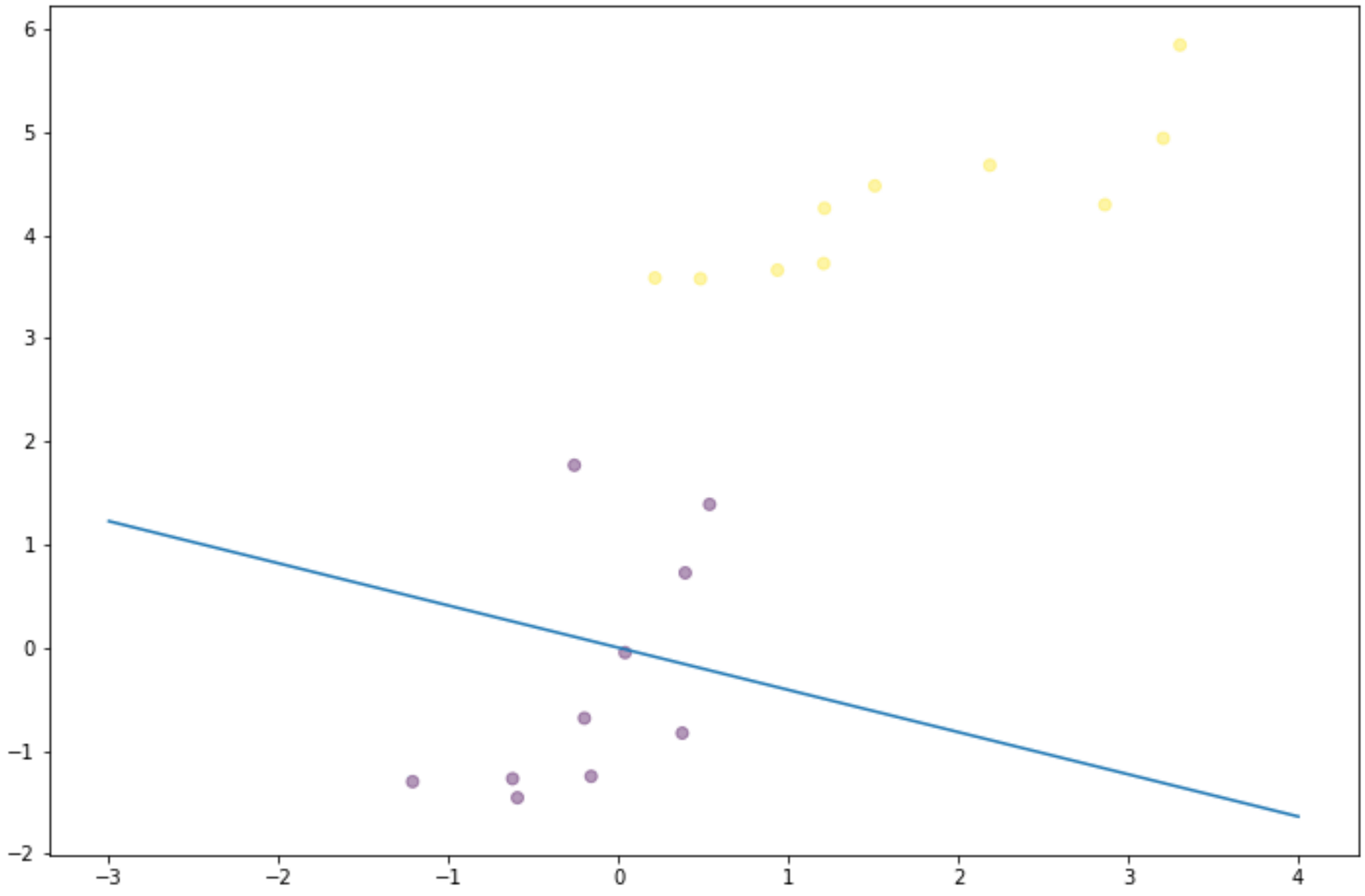
As λ goes to zero, many losses give the max-margin solution!

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n}\mathscr{L}(y_i, f_\theta(\mathbf{X}_i)) + \lambda\|\theta\|_2^2$$

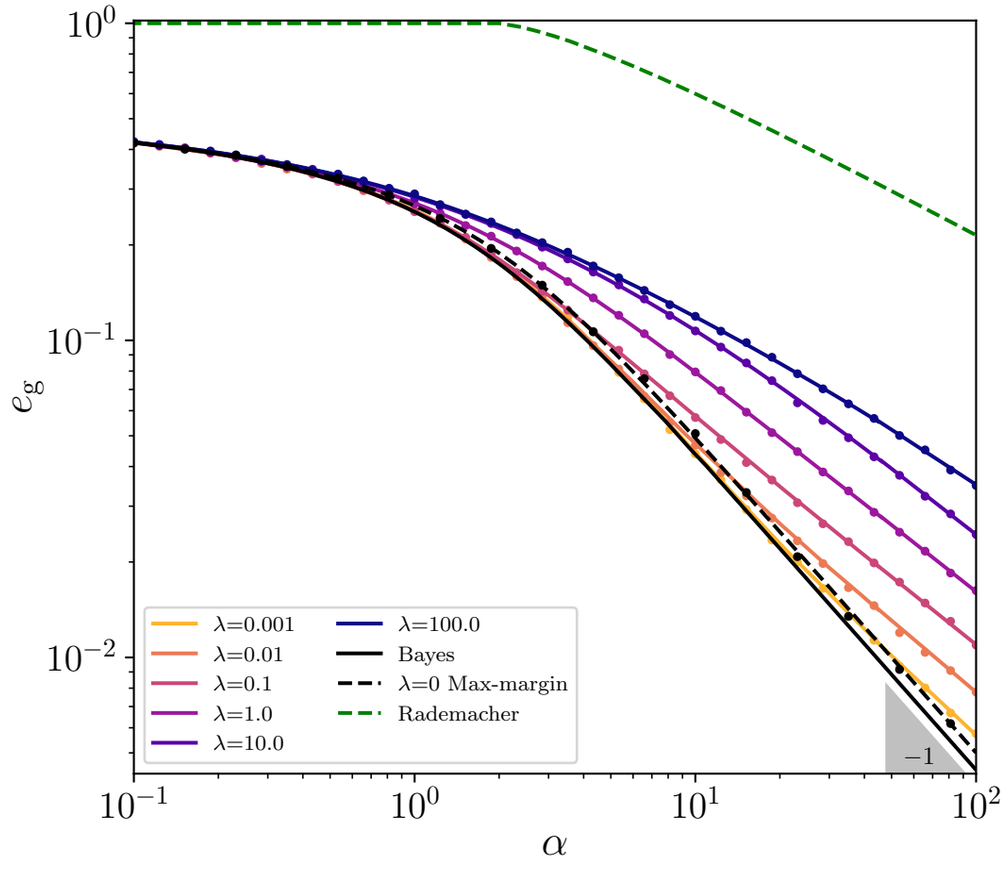
Ex: logistic LOSS

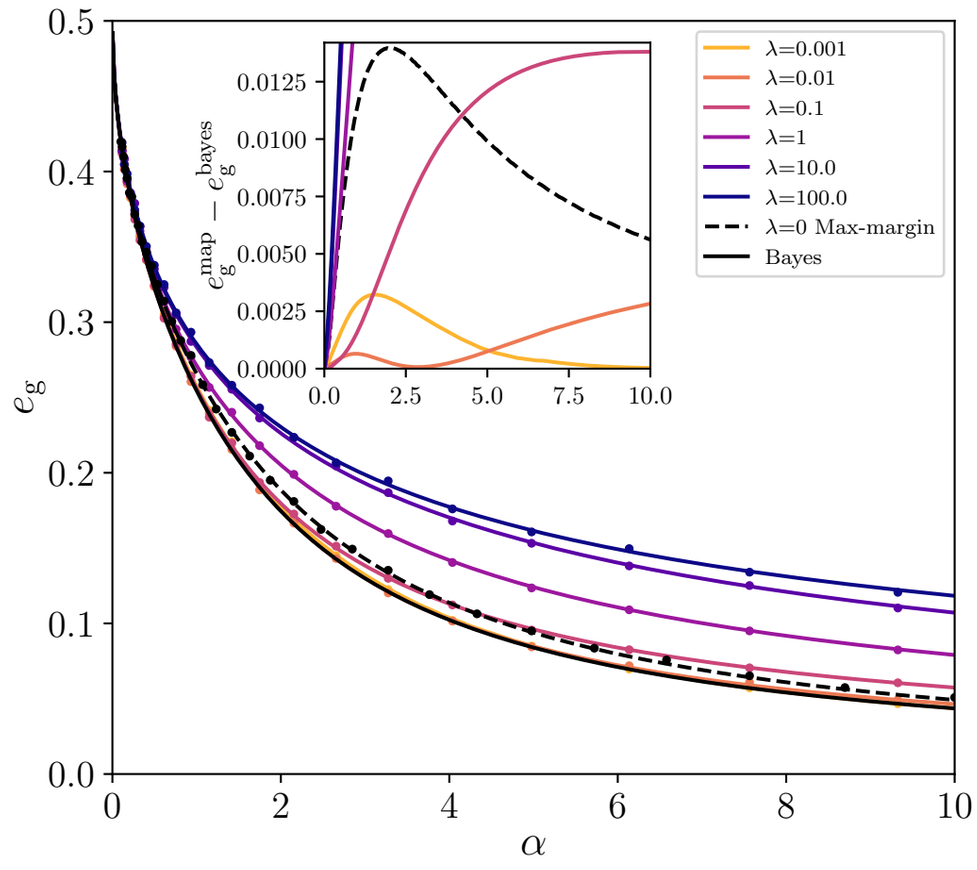


Ex: l2 LOSS



Chasing the Bayes optimal result





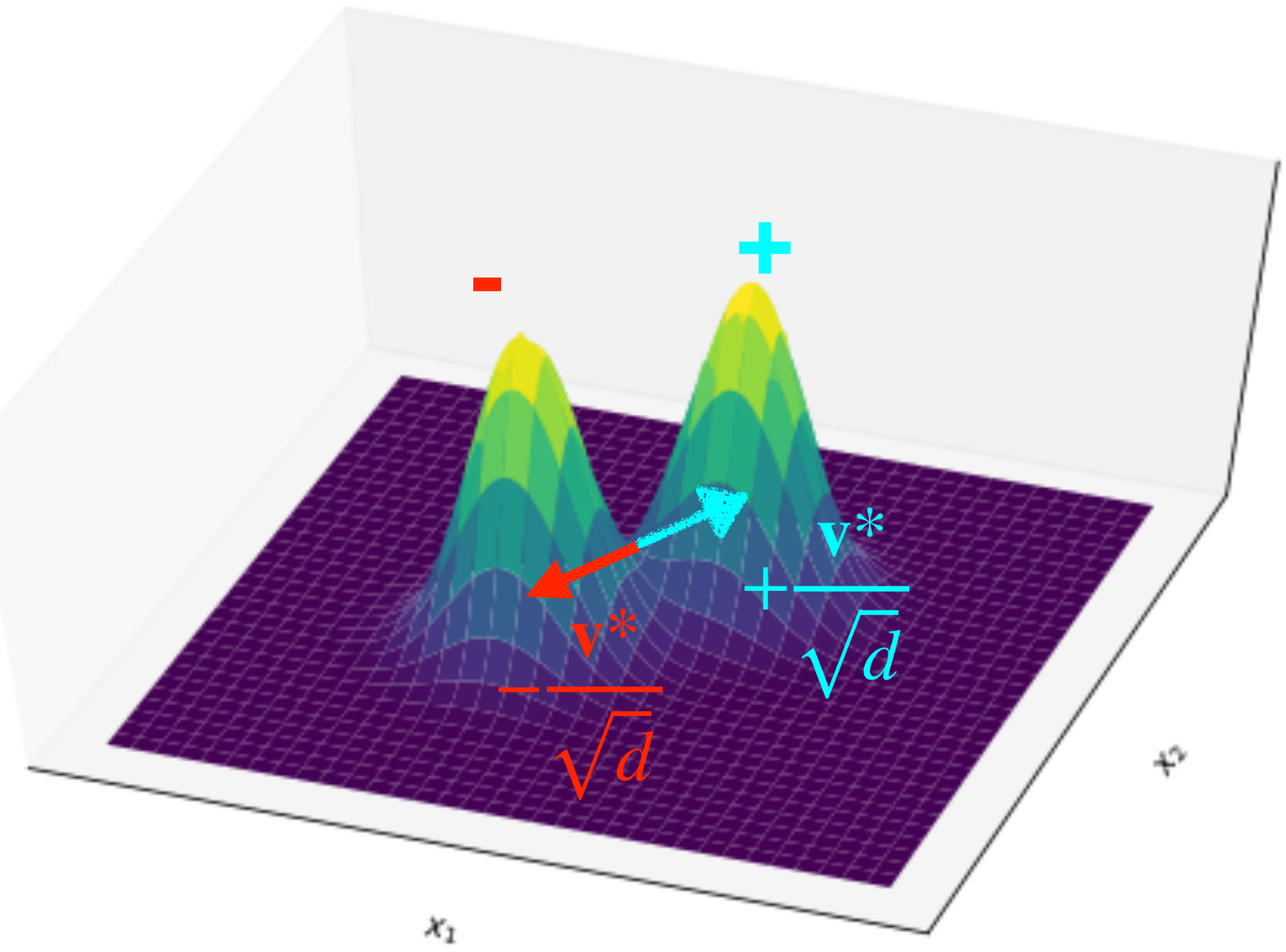
Regularized logistic losses (almost) achieve Bayes optimal results!

[Aubin, Krzakala, Lu, Zdeborova'20]

2

Linear Regressions on mixture of gaussian

Learning a high-d Gaussian Mixture



$$\mathbf{x}_i \sim \mathcal{N}\left(\frac{y_i \mathbf{v}^*}{\sqrt{d}}, \Delta\right)$$

$$\mathbf{x}_i \in \mathbb{R}^d, i = 1, \ldots, n$$

$$\alpha \equiv \frac{n}{d}$$

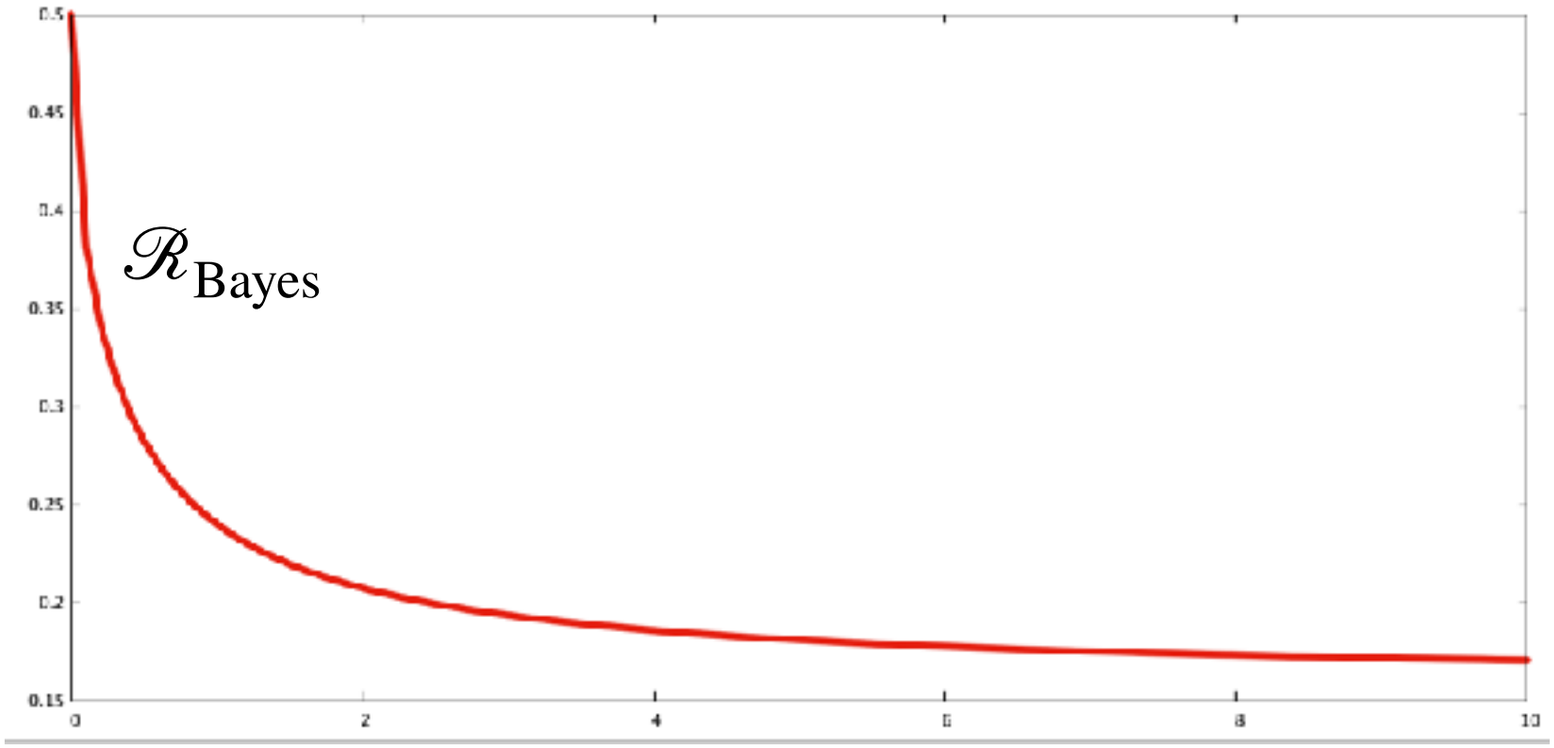
What is the best generalisation error one can obtain for the classification problem?

$$\mathscr{R}^{\text{gen}} \equiv \text{fraction of misclassified sample}$$

Learning a high-d Gaussian Mixture

Bayes optimal error

$$\mathbf{x}_i \sim \mathcal{N}\left(\frac{y_i \mathbf{v}^*}{\sqrt{d}}, \Delta\right)$$



$$\mathbf{x}_i \in \mathbb{R}^d, i = 1, \ldots, n$$

$$\alpha \equiv \frac{n}{d}$$

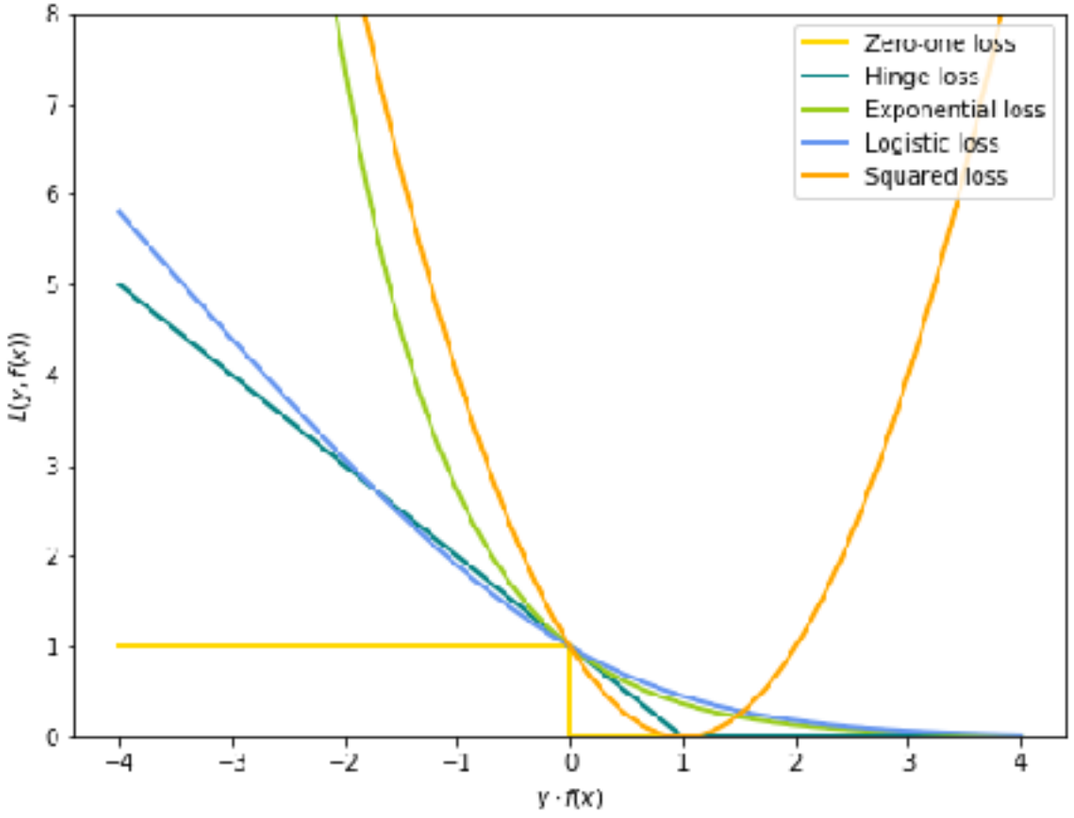
$$\mathscr{R}_{\text{Bayes}} = \frac{1}{2}\left(1 - \text{erf}\left(\frac{1}{\sqrt{2\Delta}}\sqrt{\frac{\alpha}{\Delta + \alpha}}\right)\right)$$

[E. Dobriban, S. Wager '18, L. Miolane, M. Lelarge '19]

In practice: Empirical Risk Minimization

Convex regularised regression

$$\mathscr{L} = \sum_{i=1}^{n} \ell(y\mathbf{x}_i \cdot \theta) + \frac{1}{2}\lambda\|\theta\|_2^2$$



$$\mathbf{x}_i \sim \mathcal{N}\left(\frac{y_i \mathbf{v}^*}{\sqrt{d}}, \Delta\right)$$

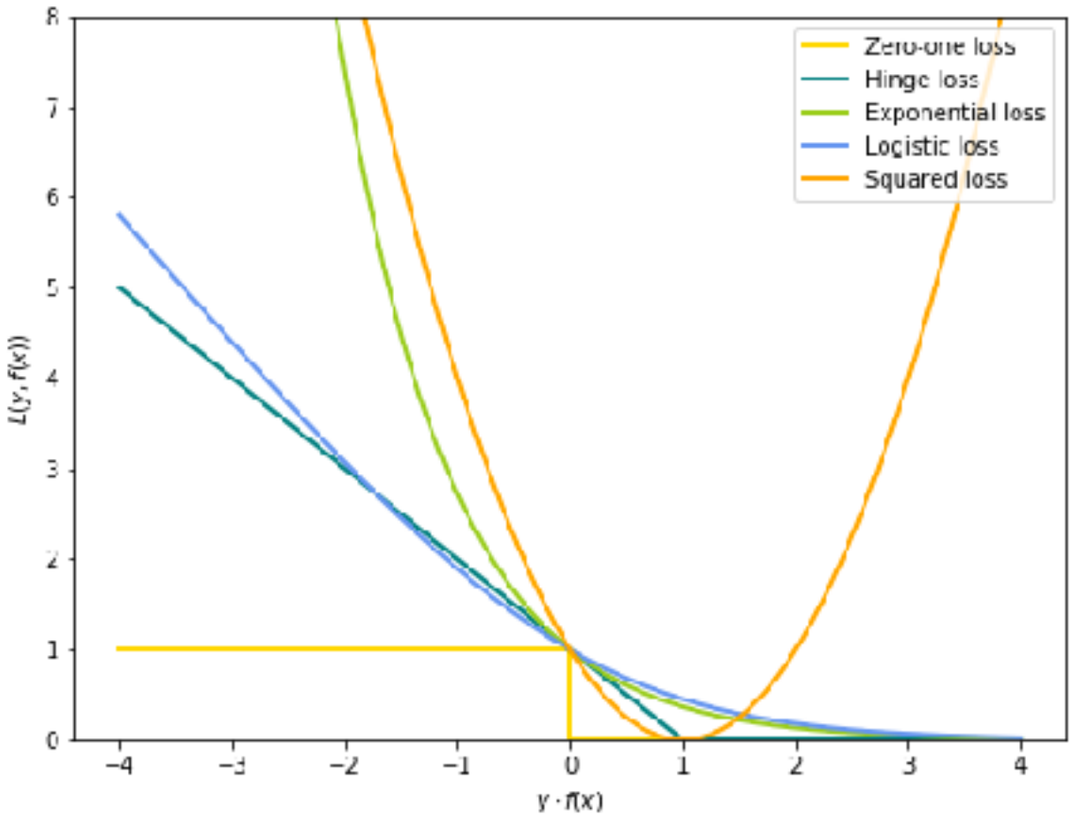
$$\mathbf{x}_i \in \mathbb{R}^d, i = 1, \ldots, n$$

$$\alpha \equiv \frac{n}{d}$$

In practice: Empirical Risk Minimization

Convex regularised regression

$$\mathscr{L} = \sum_{i=1}^{n} \ell(y\mathbf{x}_i \cdot \theta) + \frac{1}{2}\lambda\|\theta\|_2^2$$



Theorem

Definitions:

Let q and m be the unique fixed point of:

$$q = \frac{\hat{m}}{\lambda + \hat{V}} \qquad m = \frac{\hat{q} + \hat{m}^2}{\lambda + \hat{V}}$$

$$V = \frac{\Delta}{\lambda + \hat{V}}$$

$$\hat{m} = \frac{\alpha}{V}\mathbb{E}_{y,\omega}\left[yx^*(y,\omega,V) - \omega\right]$$

$$\hat{q} = \frac{\alpha\Delta}{V^2}\mathbb{E}_{y,\omega}\left[(yx^*(y,\omega,V) - \omega)^2\right]$$

$$\hat{V} = \frac{\alpha\Delta}{V}\left(1 - \mathbb{E}_{y,\omega}\left[\partial_\omega x^*(y,\omega,V)\right]\right)$$

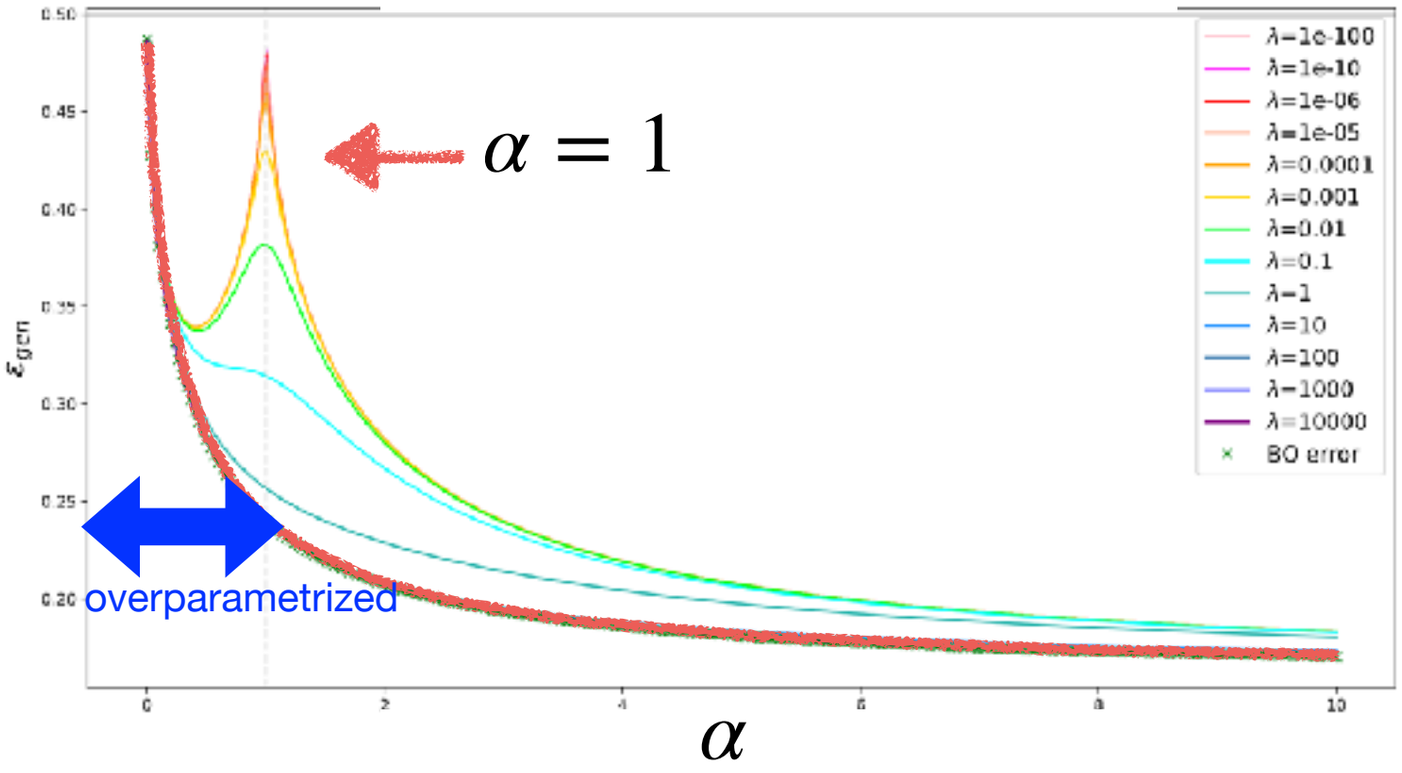
$$\alpha = n/d, \omega \sim \mathcal{N}(my, \Delta q) \quad x^* \equiv \operatorname{argmin}_x \frac{(x-\omega)^2}{2V} + \ell(yx)$$

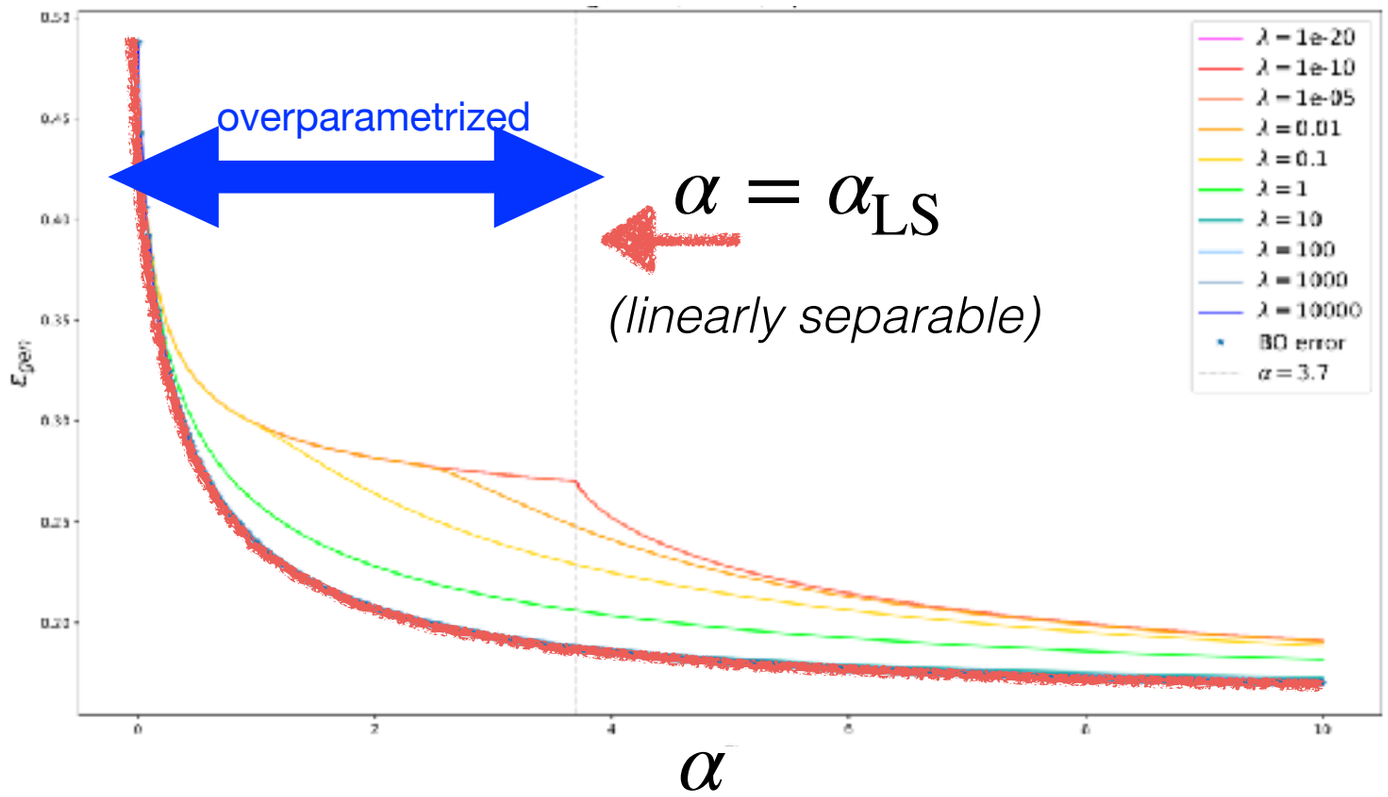
Then:

$$\mathscr{R}^{\text{gen}} = \frac{1}{2}\left(1 - \operatorname{erf}\left(\frac{m}{\sqrt{2\Delta q}}\right)\right)$$

[Mai, Liao, Couillet '19; Deng, Kammoun, Thrampoulidis '19; Mignacco, **FK**, Urbani, Zdeborova '20]

In practice: Empirical Risk Minimization





Ridge Regression

Bayes Optimal

Hinge+l2 penalty

Regularized optimisation reach Bayes Optimal results In learning a mixture of two Gaussians *for <u>any</u> convex loss*

Kernels & random projections

A tutorial

Representer theorem

For any loss function such that

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, \theta \cdot \mathbf{x}_i)$$

$$\hat{\theta} = \underset{n}{\operatorname{argmin}} \mathcal{R}(\theta)$$

We can always write the minimiser as:

$$\hat{\theta} = \sum_{i=1}^{n} \beta_i \mathbf{X}_i$$

Template matching

$$f_\beta(\mathbf{X}^{\text{new}}) = \sum_{j=1}^{n} \beta_j \mathbf{X}_j \cdot \mathbf{X}^{\text{new}}$$

Prediction for new sample are made just by comparing the scalar product of the new sample with those of the entire dataset!

Funes the memorious

As in KNN, linear models just “memorise” everything…
… no « understanding/learning » whatsoever!

‘Not only was it difficult for him to understand that the generic term ‘dog’ could embrace so many disparate individuals of diverse size and shapes, it bothered him that the dog seen in profile at 3:14 would be called the same dog at 3:15 seen from the front.’

Without effort, he had learned English, French, Portuguese, Latin. I suspect, nevertheless, that he was not very capable of thought. To think is to forget a difference, to generalize, to abstract. In the overly replete world of Funes there were nothing but details, almost contiguous details.

Jorge Louis Borges, « *Funes the Memorious* » *1942*

Template matching

$$f_\beta(\mathbf{X}^{\text{new}}) = \sum_{j=1}^{n} \beta_j \mathbf{X}_j \cdot \mathbf{X}^{\text{new}}$$

Prediction for new sample are made just by comparing the scalar product of the new sample with those of the entire dataset!

If this is a good idea, why just using a scalar product as the “similarity” ?

Template matching

$$f_\beta(\mathbf{X}) = \sum_{j=1}^{n} \beta_j K(\mathbf{X}_j, \mathbf{X}^{\text{new}})$$

Prediction for new sample are made just by comparing the scalar product of the new sample with those of the entire dataset!

If this is a good idea, why just using a scalar product as the “similarity” ?

Kernel methods

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\beta(\mathbf{X}_i))$$

$$f_\beta(\mathbf{X}) = \sum_{j=1}^{n} \beta_j K(\mathbf{X}_j, \mathbf{X})$$

Gradient descent

$$\beta \in \mathbb{R}^n$$

Gradient flow

$$\beta^t = \beta^{t-1} - \eta \nabla_\beta \mathscr{R}$$

$$\dot{\beta}^t = -\nabla_\beta \mathscr{R}$$

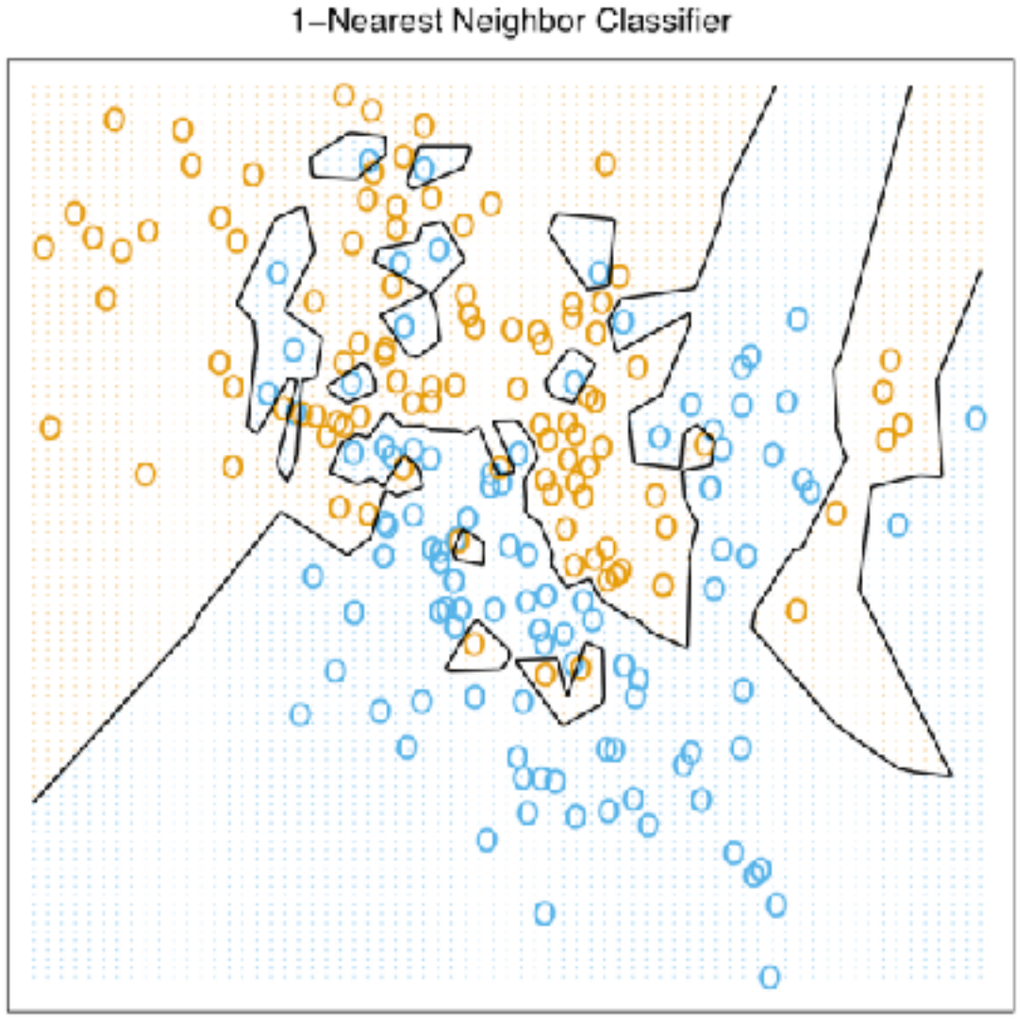
Depends only on the Gram matrix

$$K_{ij} = K(\mathbf{X}_i, \mathbf{X}_j)$$

Ex: Gaussian Kernel

$$K(\mathbf{X}_i, \mathbf{X}_j) = e^{-\beta \|\mathbf{X}_i - \mathbf{X}_j\|_2^2}$$

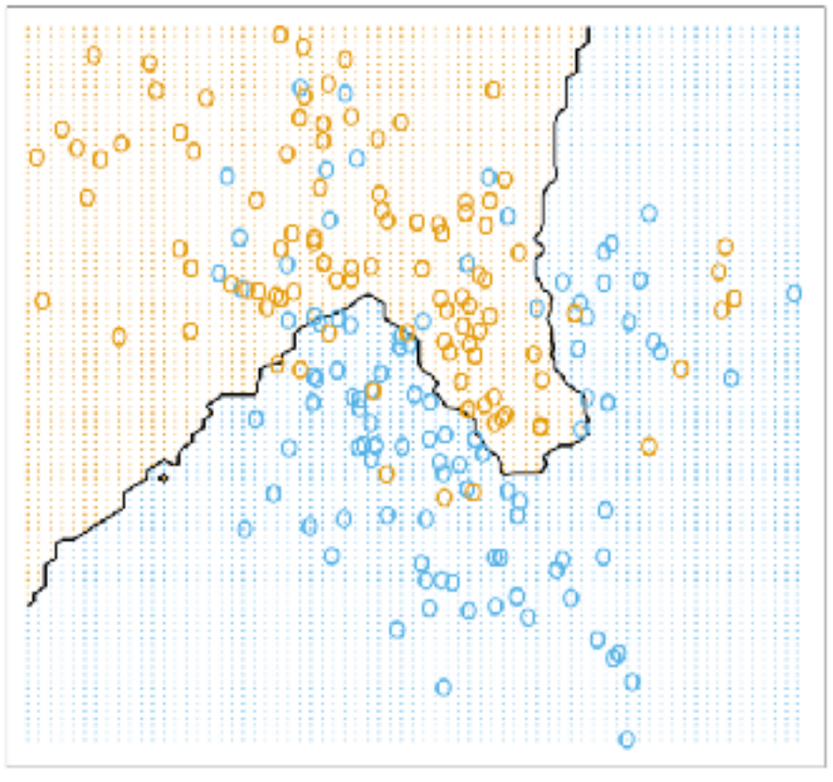
As $\beta \to \infty$ converges to 1NN methods

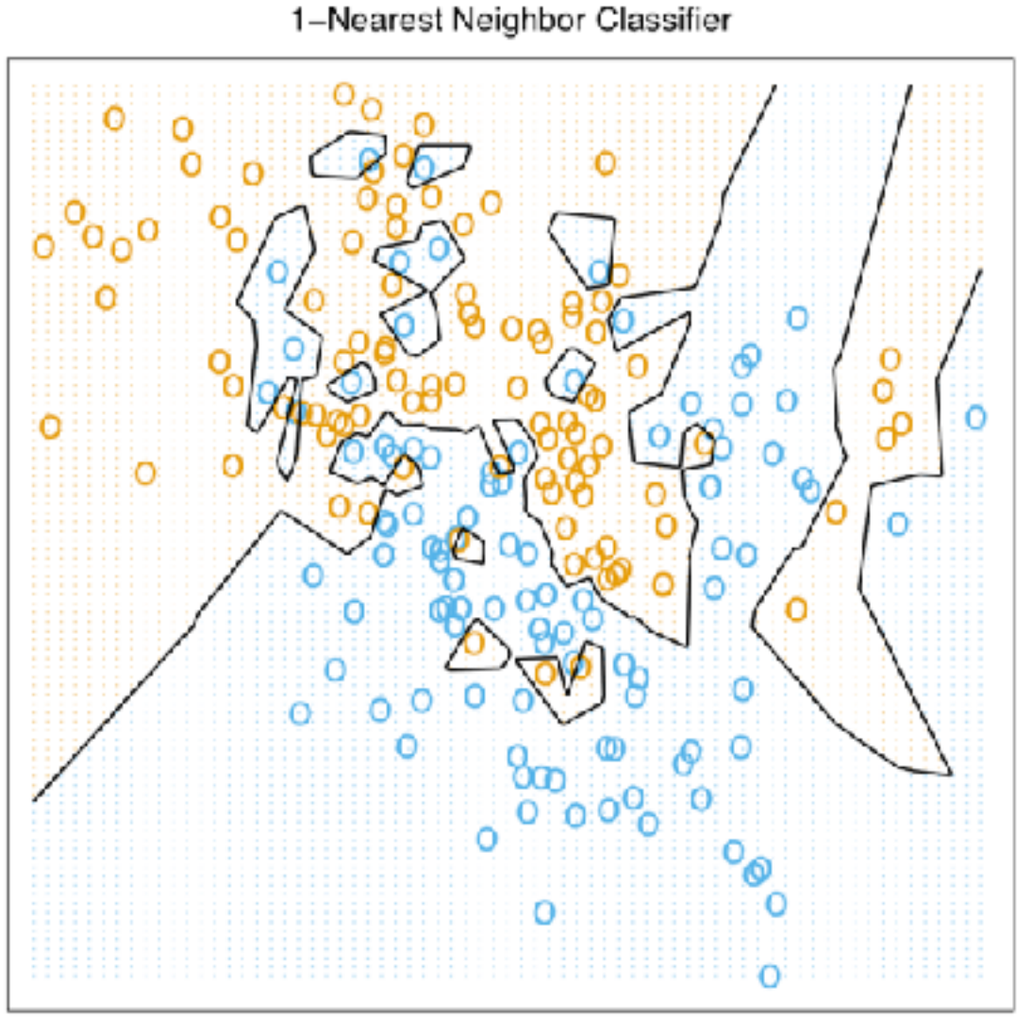


Ex: Gaussian Kernel

$$K(\mathbf{X}_i, \mathbf{X}_j) = e^{-\beta \|\mathbf{X}_i - \mathbf{X}_j\|_2^2}$$

For lower values, interpolate between neighbours





More clever kernels?

$$K(\mathbf{X}_i, \mathbf{X}_j) = ?$$

Ideally, would like to implement a meaningful notion of similarity, and invariance for relevant symmetries

---

Convolutional Kernel Networks

Julien Mairal, Piotr Koniusz, Zaid Harchaoui, and Cordelia Schmid
Inria*
firstname.lastname@inria.fr

Abstract

An important goal in visual recognition is to devise image representations that are invariant to particular transformations. In this paper, we address this goal with a new type of convolutional neural network (CNN) whose invariance is encoded by a reproducing kernel. Unlike traditional approaches where neural networks are learned either to represent data or for solving a classification task, our network learns to approximate the kernel feature map on training data.

Such an approach enjoys several benefits over classical ones. First, by teaching CNNs to be invariant, we obtain simple network architectures that achieve a similar accuracy to more complex ones, while being easy to train and robust to overfitting. Second, we bridge a gap between the neural network literature and kernels, which are natural tools to model invariance. We evaluate our methodology on visual recognition tasks where CNNs have proven to perform well, *e.g.*, digit recognition with the MNIST dataset, and the more challenging CIFAR-10 and STL-10 datasets, where our accuracy is competitive with the state of the art.

Method	[12]	[27]	[18]	[13]	[4]	[17]	[32]	CKN-GM	CKN-PM	CKN-CO
CIFAR-10	82.0	82.2	**88.32**	79.6	NA	83.96	84.87	74.84	78.30	82.18
STL-10	60.1	58.7	NA	51.5	**64.5**	62.3	NA	60.04	60.25	62.32

Table 2: Classification accuracy in % on CIFAR-10 and STL-10 without data augmentation.

Neural Tangent Kernels

$$K(\mathbf{X}_i, \mathbf{X}_j) = ?$$

Ideally, would like to implement a meaningful notion of similarity, and invariance for relevant symmetries

Sanjeev Arora @prfsanjeevarora · 5 nov.

Matching Alexnet performance (89%) on CIFAR10 using kernel method. Excluding deep nets, previous best was 86% (Mairal NIPS'16). Key ideas: convolutional NTK + Coates-Ng random patches layer + way to fold data augmentation into kernel defn arxiv.org/abs/1911.00809

47 237

More on such ideas (and related ones) in Matthieu Wyart & Max Welling lectures

Enhanced Convolutional Neural Tangent Kernels*

Zhiyuan Li† Ruosong Wang‡ Dingli Yu§ Simon S. Du¶ Wei Hu‖
Ruslan Salakhutdinov** Sanjeev Arora††

Abstract

Recent research shows that for training with ℓ_2 loss, convolutional neural networks (CNNs) whose width (number of channels in convolutional layers) goes to infinity correspond to regression with respect to the CNN Gaussian Process kernel (CNN-GP) if only the last layer is trained, and correspond to regression with respect to the Convolutional Neural Tangent Kernel (CNTK) if all layers are trained. An exact algorithm to compute CNTK [Arora et al., 2019] yielded the finding that classification accuracy of CNTK on CIFAR-10 is within 6-7% of that of the corresponding CNN architecture (best figure being around 78%) which is interesting performance for a fixed kernel.

Here we show how to significantly enhance the performance of these kernels using two ideas. (1) Modifying the kernel using a new operation called *Local Average Pooling* (LAP) which preserves efficient computability of the kernel and inherits the spirit of standard data augmentation using pixel shifts. Earlier papers were unable to incorporate naive data augmentation because of the quadratic training cost of kernel regression. This idea is inspired by *Global Average Pooling* (GAP), which we show for CNN-GP and CNTK is equivalent to full translation data augmentation. (2) Representing the input image using a pre-processing technique proposed by Coates et al. [2011], which uses a single convolutional layer composed of random image patches.

On CIFAR-10, the resulting kernel, CNN-GP with LAP and horizontal flip data augmentation,

Mercer's Theorem & the feature map

If K(s,t) is symmetric and positive-definite, then there is an orthonormal basis $\{e_i\}_i$ of $L^2[a, b]$ consisting of « **eigenfunctions** » such that the corresponding sequence of eigenvalues $\{\lambda_i\}_i$ is nonnegative.

$$K(s,t) = \sum_{j=1}^{\infty} \lambda_j \, e_j(s) \, e_j(t)$$

All symmetric positive-definite Kernels can be seen as a projection in an infinite dimensional space

Orignal space (data space) dimension d

Feature map

$$\Phi = g(X)$$

Features space (After projection) dimension D (*possibly infinite*)

X_i

$$K(\mathbf{X}_i, \mathbf{X}_j) = \mathbf{\Phi}_i \cdot \mathbf{\Phi}_j$$

$$\Phi_i = \begin{pmatrix} \sqrt{\lambda_1} e_1(X_i) \\ \sqrt{\lambda_2} e_2(X_i) \\ \sqrt{\lambda_3} e_4(X_i) \\ \cdots \\ \sqrt{\lambda_D} e_D(X_i) \end{pmatrix}$$

Example: Gaussian Kernel, 1D

$$K(\mathbf{X}_i, \mathbf{X}_j) = e^{-\frac{1}{2\sigma^2}\|\mathbf{X}_i - \mathbf{X}_j\|_2^2}$$

$$e^{\frac{-1}{2\sigma^2}(x_i - x_j)^2} = e^{\frac{-x_i^2 - x_j^2}{2\sigma^2}} \left(1 + \frac{2x_i x_j}{1!} + \frac{(2x_i x_j)^2}{2!} + \ldots\right) \tag{1.25}$$

$$= e^{\frac{-x_i^2 - x_j^2}{2\sigma^2}} \left(1 \cdot 1 + \sqrt{\frac{2}{1!}}x_i \cdot \sqrt{\frac{2}{1!}}x_j + \sqrt{\frac{(2)^2}{2!}}(x_i)^2 \cdot \sqrt{\frac{(2)^2}{2!}}(x_j)^2 + \ldots\right)$$

$$= \phi(x_i)^T \phi(x_j)$$

$$\text{where, } \phi(x) = e^{\frac{-x^2}{2\sigma^2}} \left(1, \sqrt{\frac{2}{1!}}x, \sqrt{\frac{2^2}{2!}}x^2, \ldots\right)$$

Infinite dimensional feature (polynomial) map!

Kernel methods (i)

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\beta(\mathbf{X}_i))$$

$$f_\beta(\mathbf{X}) = \sum_{j=1}^{n} \beta_j K(\mathbf{X}_j, \mathbf{X})$$

Gradient descent

$$\beta \in \mathbb{R}^n$$

Gradient flow

$$\beta^t = \beta^{t-1} - \eta \nabla_\beta \mathscr{R}$$

$$\dot{\beta}^t = -\nabla_\beta \mathscr{R}$$

Depends only on the Gram matrix

$$K_{ij} = K(\mathbf{X}_i, \mathbf{X}_j)$$

Kernel methods (ii)

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\beta(\mathbf{\Phi}_i))$$

$$f_\beta(\mathbf{\Phi}) = \sum_{j=1}^{n} \beta_j \mathbf{\Phi}_j, \mathbf{\Phi}_i$$

Gradient descent

$$\beta \in \mathbb{R}^n$$

Gradient flow

$$\beta^t = \beta^{t-1} - \eta \nabla_\beta \mathscr{R}$$

$$\dot{\beta}^t = -\nabla_\beta \mathscr{R}$$

Depends only on the Gram matrix

$$K_{ij} = \mathbf{\Phi}_i \cdot \mathbf{\Phi}_j$$

Kernel methods (iii)
Linear model (in feature space!)

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{\Phi}_i))$$

$$f_\theta(\mathbf{\Phi}) = \theta \cdot \mathbf{\Phi}$$

$$\theta \in \mathbb{R}^{D} \quad (D = \infty)$$

Gradient descent

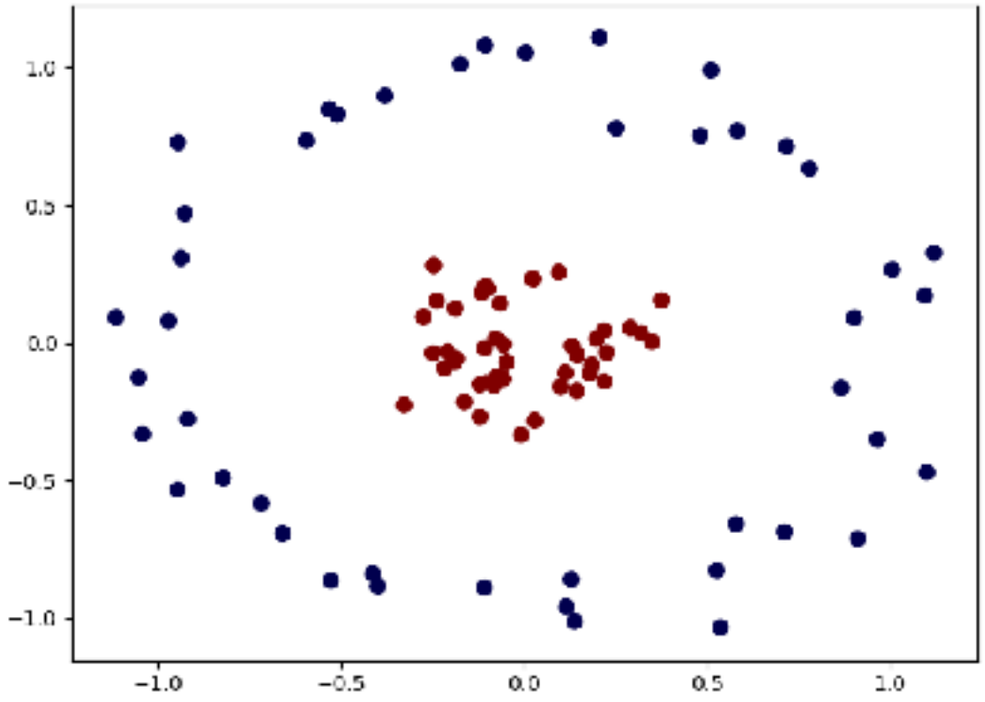
$$\theta^t = \theta^{t-1} - \eta \nabla_\theta \mathscr{R}$$

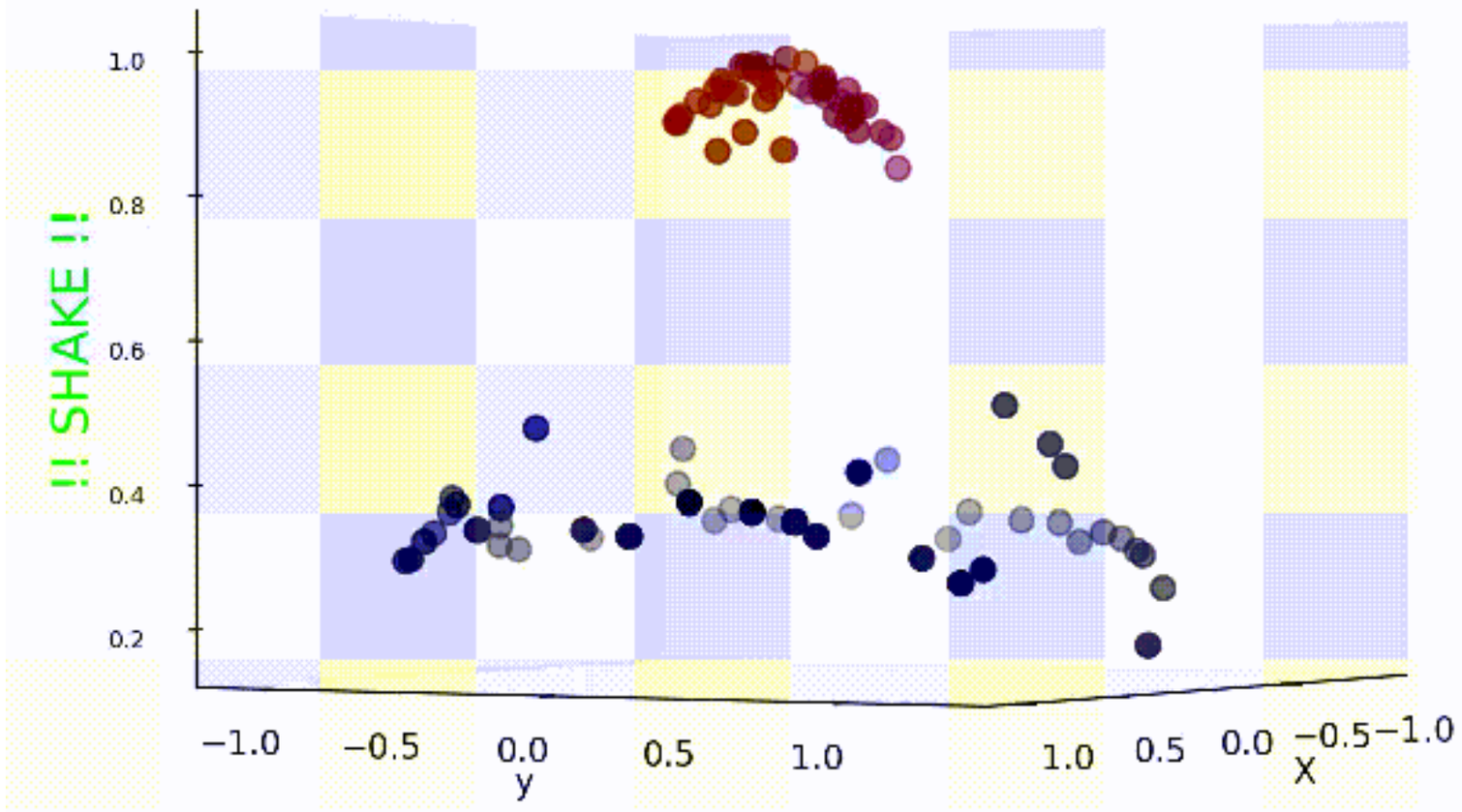
Gradient flow

$$\dot{\theta}^t = -\nabla_\theta \mathscr{R}$$

Mapping to large dimension

Can separate arbitrary complicated functions!





The Kernel Trick!

The problem with Kernels (& the solution)

Kernel methods

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\beta(\mathbf{X}_i))$$

$$f_\beta(\mathbf{X}) = \sum_{j=1}^{n} \beta_j K(\mathbf{X}_j, \mathbf{X}) \quad \beta \in \mathbb{R}^n$$

Gradient descent

$$\beta^t = \beta^{t-1} - \eta \nabla_\beta \mathscr{R}$$

Gradient flow

$$\dot{\beta}^t = -\nabla_\beta \mathscr{R}$$

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{\Phi}_i))$$

Feature map $\Phi = g(X)$

$$f_\theta(\mathbf{\Phi}) = \theta \cdot \mathbf{\Phi} \qquad \theta \in \mathbb{R}^{D=\infty}$$

Gradient descent

$$\theta^t = \theta^{t-1} - \eta \nabla_\theta \mathscr{R}$$

Gradient flow

$$\dot{\theta}^t = -\nabla_\theta \mathscr{R}$$

$$K(X_i, X_j) = \Phi_i \cdot \Phi_j$$

$$\mathbf{K} = \begin{pmatrix} K(X^1, X^1) & K(X^1, X^2) & \dots & K(X^1, X^N) \\ K(X^2, X^1) & K(X^2, X^2) & \dots & K(X^2, X^N) \\ \dots & \dots & \dots & \dots \\ K(X^N, X^1) & K(X^N, X^2) & \dots & K(X^N, X^N) \end{pmatrix}$$

Say you have one million examples….

Kernel methods

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\beta(\mathbf{X}_i))$$

$$f_\beta(\mathbf{X}) = \sum_{j=1}^{n} \beta_j K(\mathbf{X}_j, \mathbf{X}) \qquad \beta \in \mathbb{R}^n$$

Gradient descent

$$\beta^t = \beta^{t-1} - \eta \nabla_\beta \mathscr{R}$$

Gradient flow

$$\dot{\beta}^t = -\nabla_\beta \mathscr{R}$$

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{\Phi}_i))$$

Feature map $\Phi = g(X)$

$$f_\theta(\mathbf{\Phi}) = \theta \cdot \mathbf{\Phi} \qquad \theta \in \mathbb{R}^{D=\infty}$$

Gradient descent

$$\theta^t = \theta^{t-1} - \eta \nabla_\theta \mathscr{R}$$

Gradient flow

$$\dot{\theta}^t = -\nabla_\theta \mathscr{R}$$

Kernel methods

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{\Phi}_i)) \qquad f_\theta(\mathbf{\Phi}) = \theta \cdot \mathbf{\Phi}$$

Feature map $\Phi = g(X)$

$$\theta \in \mathbb{R}^{D=\infty}$$

Gradient descent

$$\theta^t = \theta^{t-1} - \eta \nabla_\theta \mathscr{R}$$

Gradient flow

$$\dot{\theta}^t = -\nabla_\theta \mathscr{R}$$

Idea 1: truncate the expansion of the feature map (e.g. polynomial features)

Idea 2: approximate the feature map by sampling

Random Fourrier features

(Recht-Rahimi '07)

$$\mathbf{\Phi} = g(\mathbf{x})$$

$$\mathbf{\Phi} \in \mathbb{R}^D \qquad \mathbf{x} \in \mathbb{R}^d$$

$$\mathbf{\Phi} = \frac{1}{\sqrt{D}} e^{iF\mathbf{x}}$$

F a Dxd matrix,
Coefficients i.i.d. random from P(F)

$$\Phi_i = \frac{1}{\sqrt{\mathbf{D}}} \begin{pmatrix} e^{i2\pi \vec{F}_1 \cdot X_i} \\ e^{i2\pi \vec{F}_2 \cdot X_i} \\ \cdots \\ e^{i2\pi \vec{F}_D \cdot X_i} \end{pmatrix}$$

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{\Phi}_i \cdot \mathbf{\Phi}_j = \frac{1}{D} \sum_{k=1}^{D} e^{i\vec{F}_k(\mathbf{x_i} - \mathbf{x_j})} \xrightarrow[D\to\infty]{\lim} \mathbb{E}_{\vec{F}}\left[e^{i\vec{F}(\mathbf{x_i}-\mathbf{x_j})}\right] = \int d\vec{F} P(\vec{F}) e^{i\vec{F}(\mathbf{x_i}-\mathbf{x_j})}$$

$$K(X_i, X_j) = K($$

Kernel Name	k
Gaussian	e
Laplacian	e
Cauchy	Γ

Ali Rahimi - NIPS 2017 Test-of-Time Award presentation

Random erf features

(Williams '07)

$$\Phi_i = \frac{1}{\sqrt{\mathbf{D}}} \begin{pmatrix} \mathrm{erf}(\tilde{\mathrm{F}}_1 \cdot \mathrm{X_i}) \\ \mathrm{erf}(\tilde{\mathrm{F}}_2 \cdot \mathrm{X_i}) \\ \cdots \\ \mathrm{erf}(\tilde{\mathrm{F}}_\mathrm{D} \cdot \mathrm{X_i}) \end{pmatrix}$$

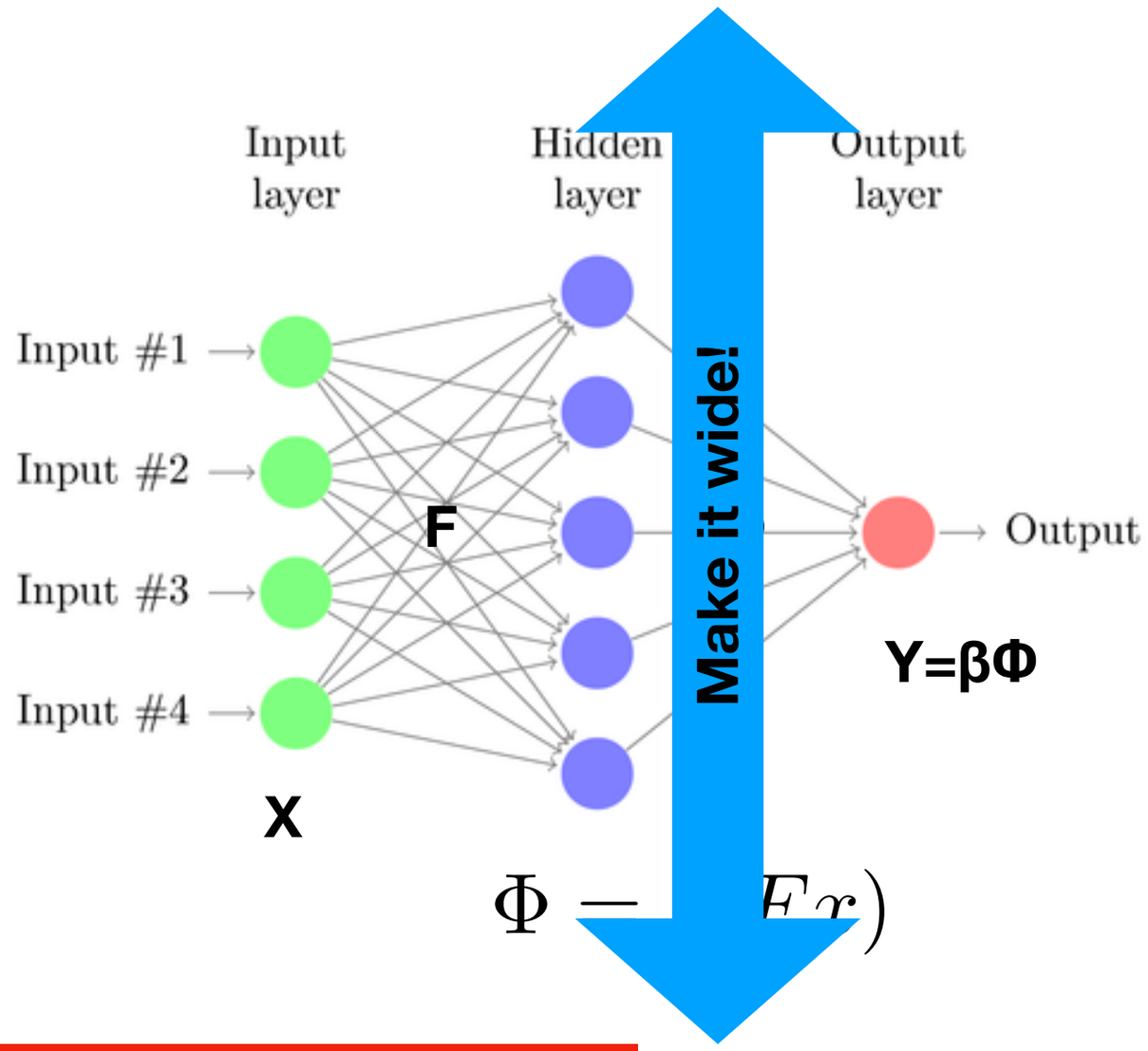
$$\lim_{D \to \infty} \Phi_i \cdot \Phi_j \ \textbf{?}$$

After a bit of work (Williams, 98)

$$K(X_i, X_j) = \frac{2}{\pi} \arcsin \left(\frac{2X_i \cdot X_j}{\sqrt{1 + 2X_i \cdot X_i}\sqrt{1 + 2X_j \cdot X_j}} \right)$$

Here the kernel depends on the angle….

Equivalent representation:
A WIIIIIIIDE random 2-layer neural network



Fix the « weights » in the first layer randomly…
… and to learn only the weights in the second layer

Infinitly wide neural net with random weights converges to kernel methods
(Neal ’96, Williams 98, Recht-Rahimi ‘07)

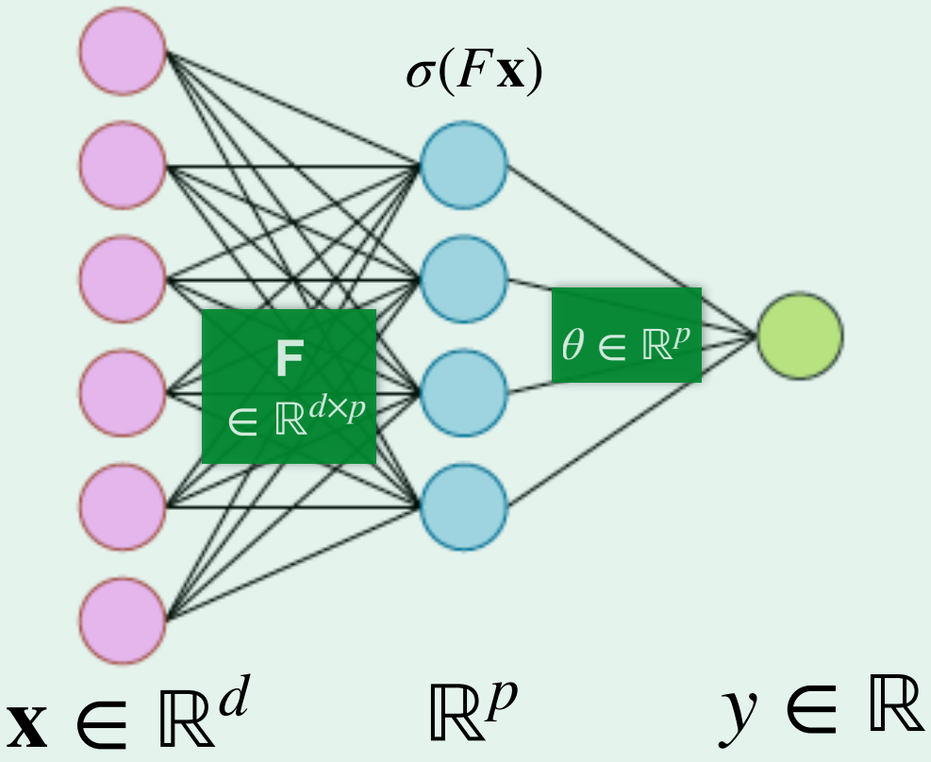
Deep connection with genuine neural networks in the “Lazy regime”
[Jacot, Gabriel, Hongler ’18; Chizat, Bach ’19; Geiger et al. ’19]

4

Results on Random Projections Physics-style

Random features neural net

Architecture: Two-layers neural network with *fixed* first layer **F**

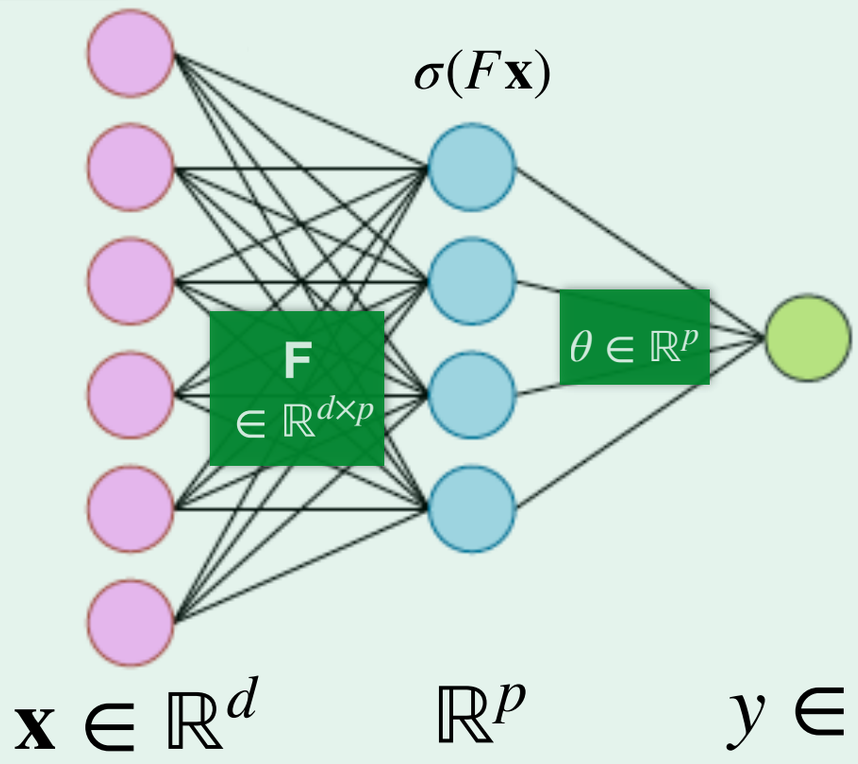


Random feature model…

Dataset:

- n vector $\mathbf{x}_i \in \mathbb{R}^d$, drawn randomly from $\mathcal{N}(0, \mathbf{1}_d)$
- n labels y_i given by a function $y_i^0 = f^0(\mathbf{x} \cdot \theta^*)$

Architecture: Two-layers neural network with fixed first layer **F**



Cost function:

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, y_i^0) + \lambda \|\theta\|_2^2 \qquad \ell(\,.\,) = \begin{matrix} \textit{Logistic loss} \\ \textit{Hinge loss} \\ \textit{Square loss} \\ \dots \end{matrix}$$

What is the training error & the generalisation error in the high dimensional limit $(d, p, n) \to \infty$?

… and its solution

Mention COUILLET HERE

[Loureiro, Gerace, **FK**, Mézard, Zdeborova, '20]

Definitions:

Consider the unique fixed point of the following system of equations

$$\begin{cases}
\hat{V}_s = \frac{\alpha}{\gamma}\kappa_1^2 \mathbb{E}_{\xi,y}\left[\mathscr{Z}\left(y,\omega_0\right)\frac{\partial_\omega \eta(y,\omega_1)}{V}\right], \\
\hat{q}_s = \frac{\alpha}{\gamma}\kappa_1^2 \mathbb{E}_{\xi,y}\left[\mathscr{Z}\left(y,\omega_0\right)\frac{\left(\eta(y,\omega_1)-\omega_1\right)^2}{V^2}\right], \\
\hat{m}_s = \frac{\alpha}{\gamma}\kappa_1 \mathbb{E}_{\xi,y}\left[\partial_\omega \mathscr{Z}\left(y,\omega_0\right)\frac{(\eta(y,\omega_1)-\omega_1)}{V}\right], \\
\hat{V}_w = \alpha\kappa_\star^2 \mathbb{E}_{\xi,y}\left[\mathscr{Z}\left(y,\omega_0\right)\frac{\partial_\omega \eta(y,\omega_1)}{V}\right], \\
\hat{q}_w = \alpha\kappa_\star^2 \mathbb{E}_{\xi,y}\left[\mathscr{Z}\left(y,\omega_0\right)\frac{\left(\eta(y,\omega_1)-\omega_1\right)^2}{V^2}\right],
\end{cases}
\qquad
\begin{cases}
V_s = \frac{1}{\hat{V}_s}\left(1 - z\ g_\mu(-z)\right), \\
q_s = \frac{\hat{m}_s^2+\hat{q}_s}{\hat{V}_s}\left[1-2zg_\mu(-z)+z^2 g'_\mu(-z)\right] \\
\quad -\frac{\hat{q}_w}{(\lambda+\hat{V}_w)\hat{V}_s}\left[-zg_\mu(-z)+z^2 g'_\mu(-z)\right], \\
m_s = \frac{\hat{m}_s}{\hat{V}_s}\left(1 - z\ g_\mu(-z)\right), \\
V_w = \frac{\gamma}{\lambda+\hat{V}_w}\left[\frac{1}{\gamma}-1+zg_\mu(-z)\right], \\
q_w = \gamma\frac{\hat{q}_w}{(\lambda+\hat{V}_w)^2}\left[\frac{1}{\gamma}-1+z^2 g'_\mu(-z)\right], \\
\quad +\frac{\hat{m}_s^2+\hat{q}_s}{(\lambda+\hat{V}_w)\hat{V}_s}\left[-zg_\mu(-z)+z^2 g'_\mu(-z)\right],
\end{cases}
\qquad
\begin{cases}
\eta(y,\omega) = \underset{x\in\mathbb{R}}{\text{argmin}}\left[\frac{(x-\omega)^2}{2V} + \mathscr{\ell}(y,x)\right] \\
\mathscr{Z}(y,\omega) = \int \frac{dx}{\sqrt{2\pi V^0}} e^{-\frac{1}{2V^0}(x-\omega)^2}\delta\left(y - f^0(x)\right)
\end{cases}$$

where $V = \kappa_1^2 V_s + \kappa_\star^2 V_w$, $V^0 = \rho - \frac{M^2}{Q}$, $Q = \kappa_1^2 q_s + \kappa_\star^2 q_w$, $M = \kappa_1 m_s$, $\omega_0 = M/\sqrt{Q}\xi$, $\omega_1 = \sqrt{Q}\xi$ and g_μ is the Stieltjes transform of FF^T

$\kappa_0 = \mathbb{E}\left[\sigma(z)\right], \kappa_1 \equiv \mathbb{E}\left[z\sigma(z)\right], \kappa_\star \equiv \mathbb{E}\left[\sigma(z)^2\right] - \kappa_0^2 - \kappa_1^2$, and $\vec{z}^\mu \sim \mathcal{N}(\vec{0}, \mathbf{I_p})$

Then in the high-dimensional limit:

$$\epsilon_{gen} = \mathbb{E}_{\lambda,\nu}\left[(f^0(\nu) - \hat{f}(\lambda))^2\right]$$

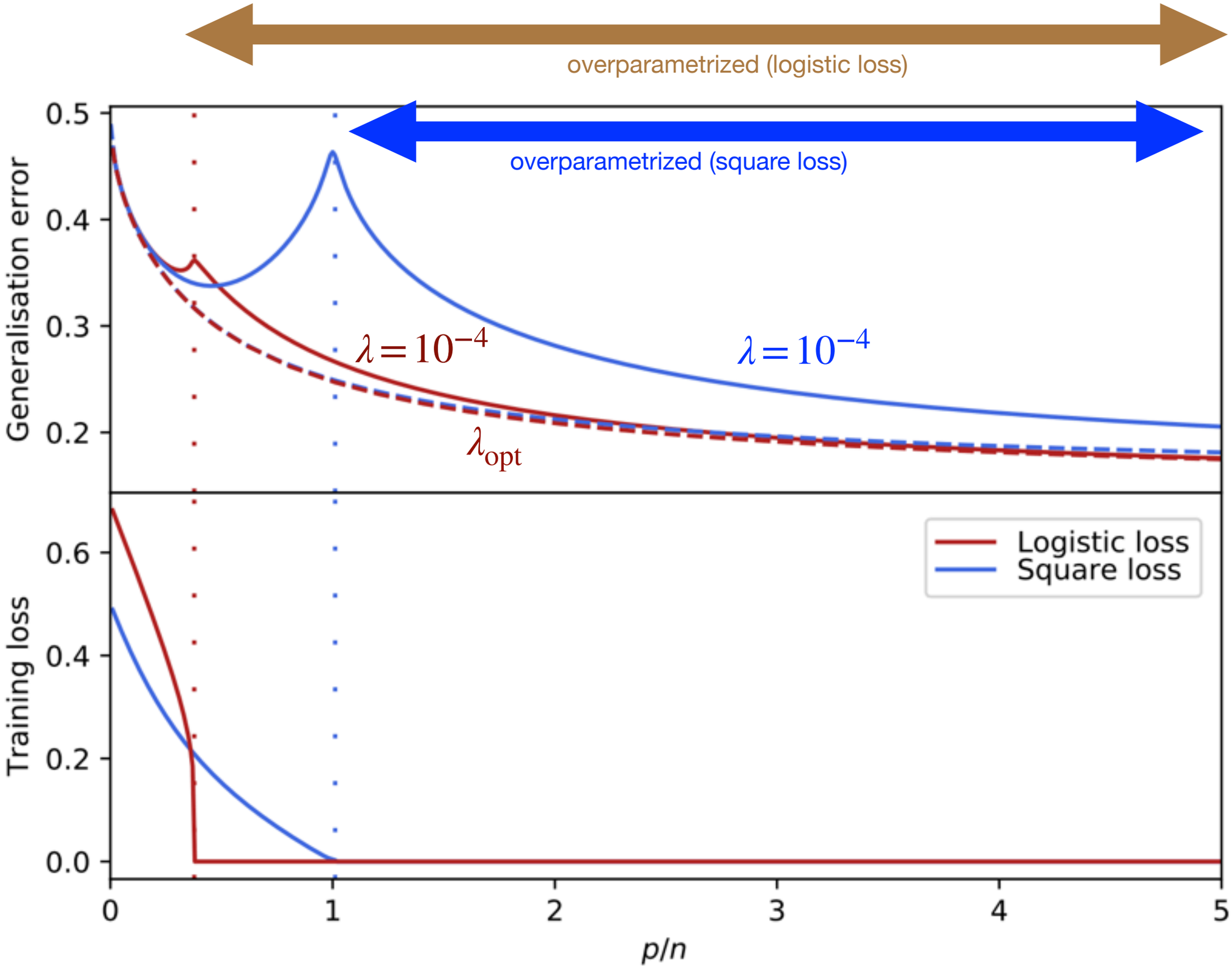
with $(\nu,\lambda) \sim \mathcal{N}\left(\begin{pmatrix}0\\0\end{pmatrix}, \begin{pmatrix}\rho & M^\star \\ M^\star & Q^\star\end{pmatrix}\right)$

$$\mathscr{L}_{\text{training}} = \frac{\lambda}{2\alpha}q_w^\star + \mathbb{E}_{\xi,y}\left[\mathscr{Z}\left(y,\omega_0^\star\right)\ell\left(y,\eta(y,\omega_1^\star)\right)\right]$$

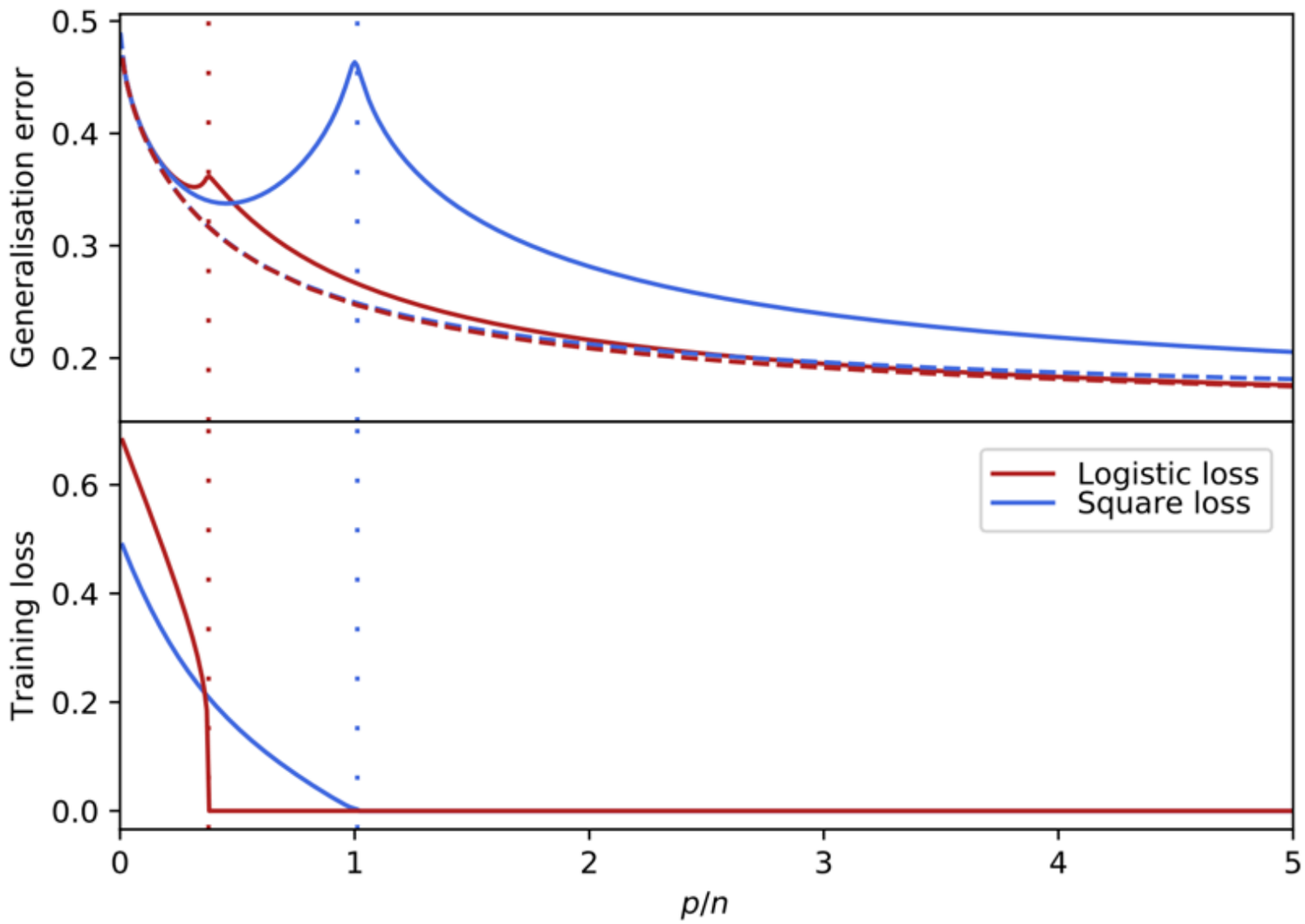
with $\omega_0^\star = M^\star/\sqrt{Q^\star}\xi, \omega_1^\star = \sqrt{Q^\star}\xi$

Agrees with **[Louart , Liao , Couillet'18 & Mei-Montanari '19]** who solved a particular case using random matrix theory: linear function f^0, $\mathscr{\ell}(x,y) = \|x-y\|_2^2$ & Gaussian random weights **F**,

A classification task



A classification task



Implicit regularisation of gradient descent [Rosset, Zhy, Hastie, '04] [Neyshabur, Tomyoka, Srebro, '15]

As $\lambda \to 0$*, in the overparametrized regime,*
Logistic converges to max-margin, ℓ_2 *converges to least norm*

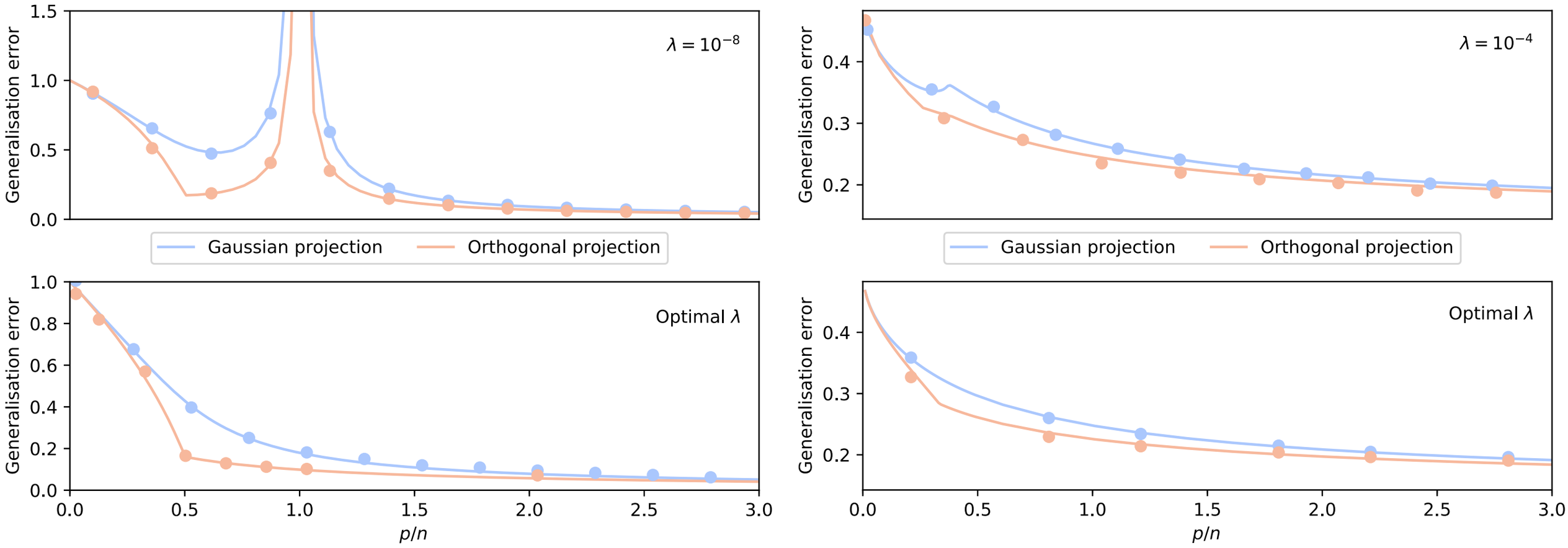
Asymptotics accurate even at d=200!

Regression task

ℓ_2 loss

Classification task

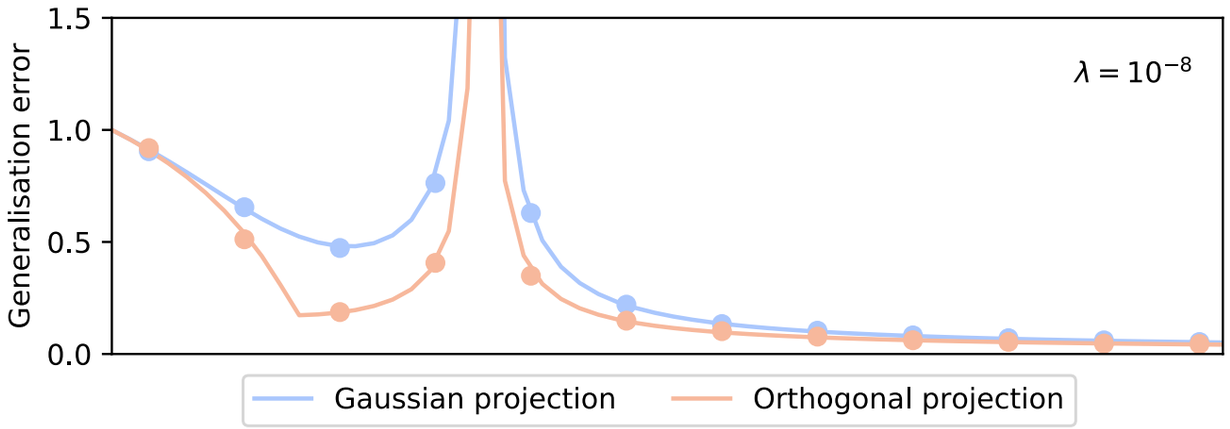
logistic loss

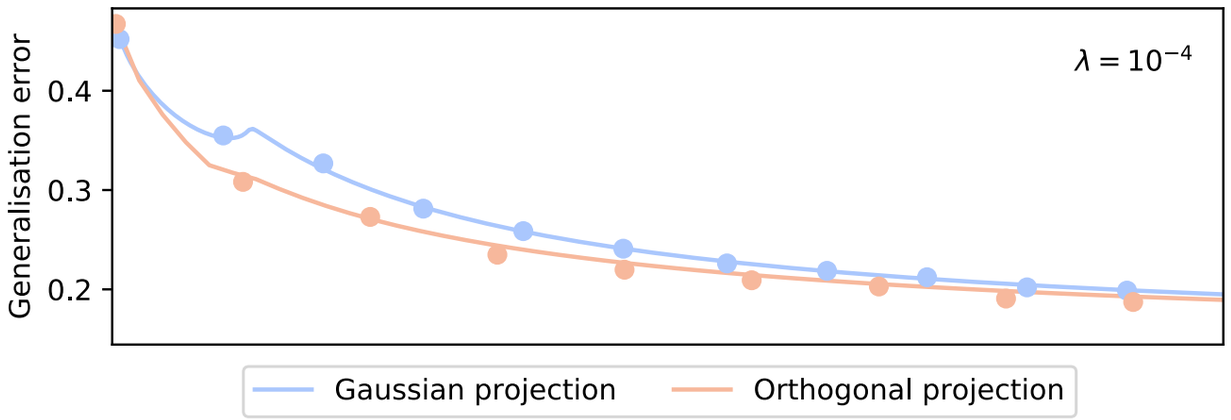


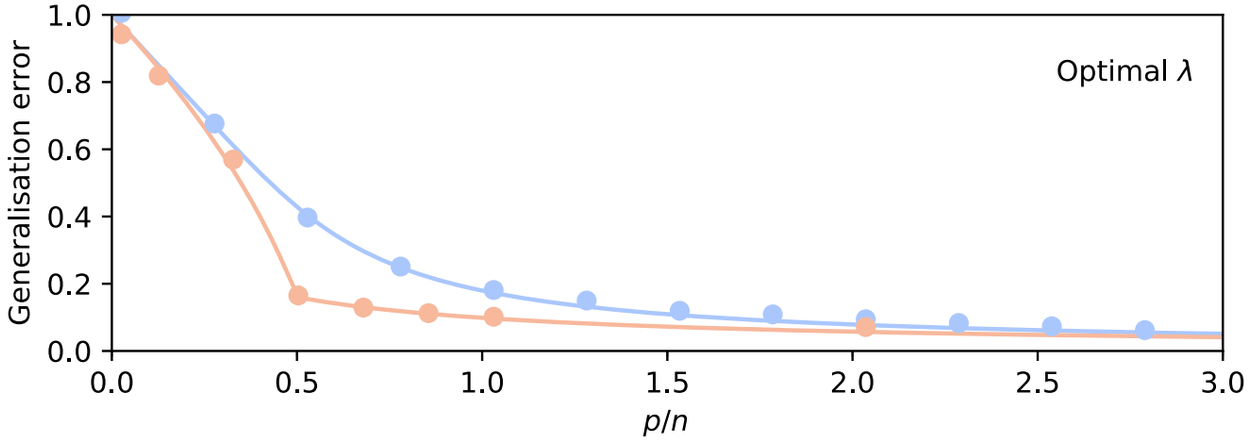
First layer: random Gaussian Matrix

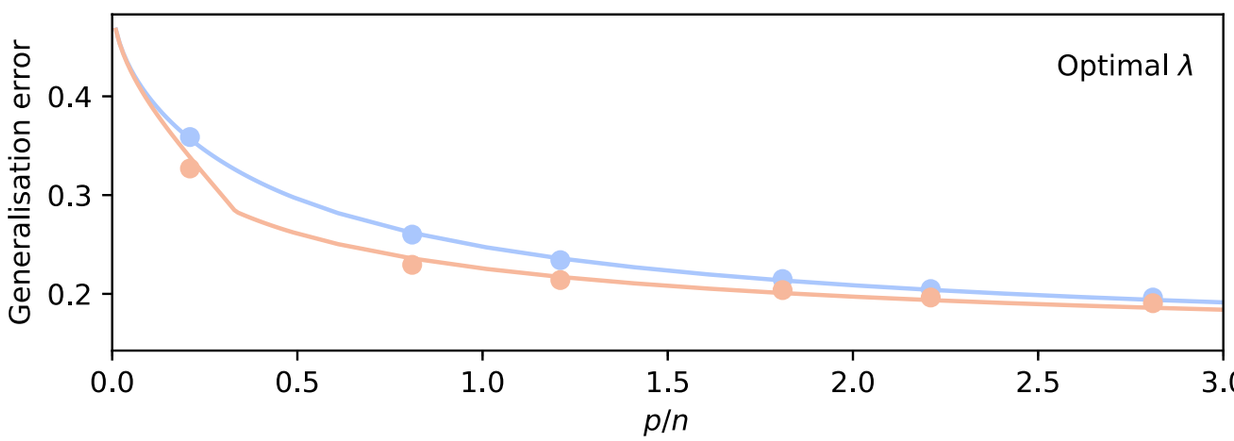
First layer: subsampled Fourier matrix

Regularisation & different First Layer









The Unreasonable Effectiveness of Structured Random Orthogonal Embeddings

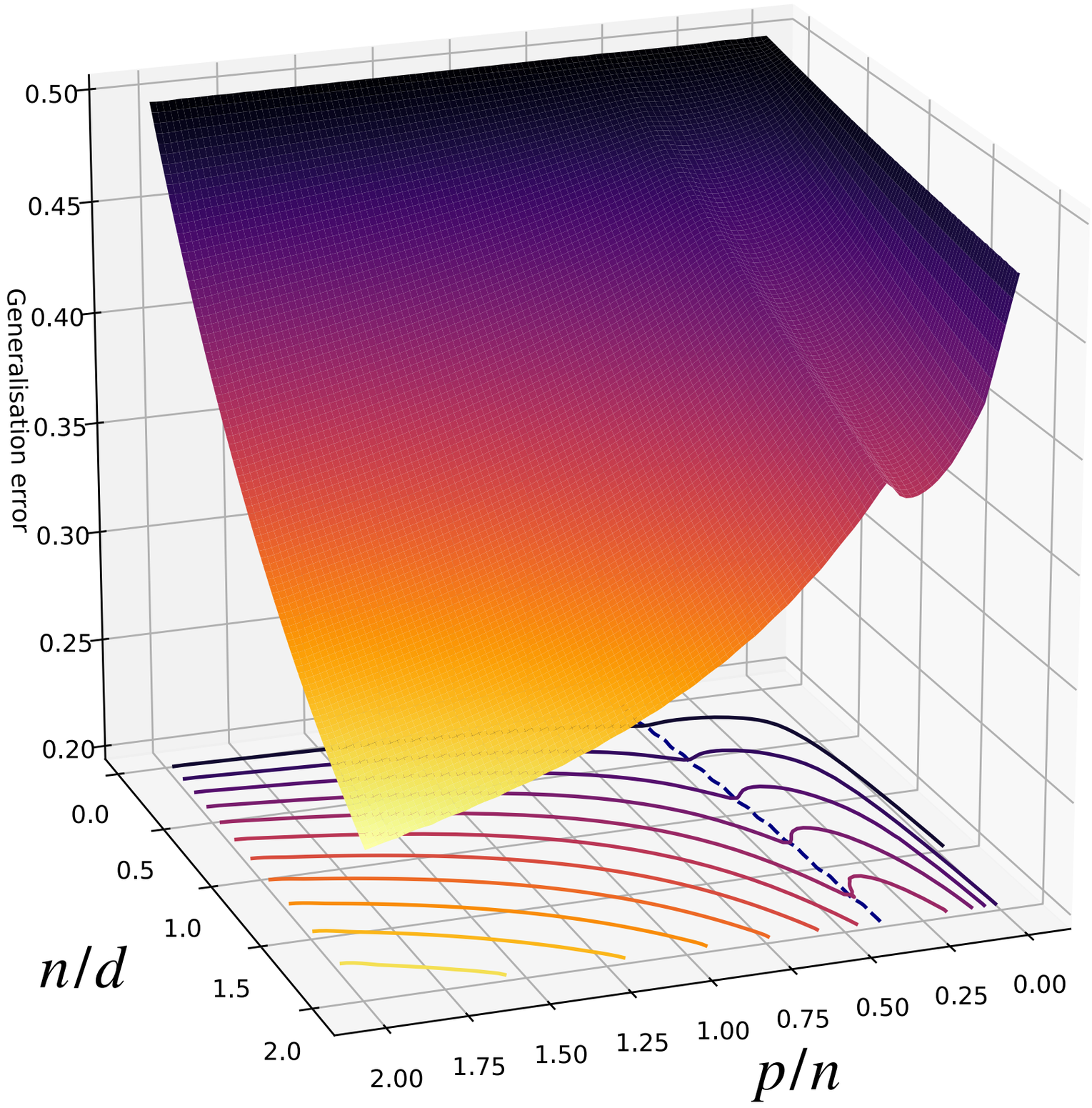
Krzysztof Choromanski *
Google Brain Robotics
kchoro@google.com

Mark Rowland *
University of Cambridge
mr504@cam.ac.uk

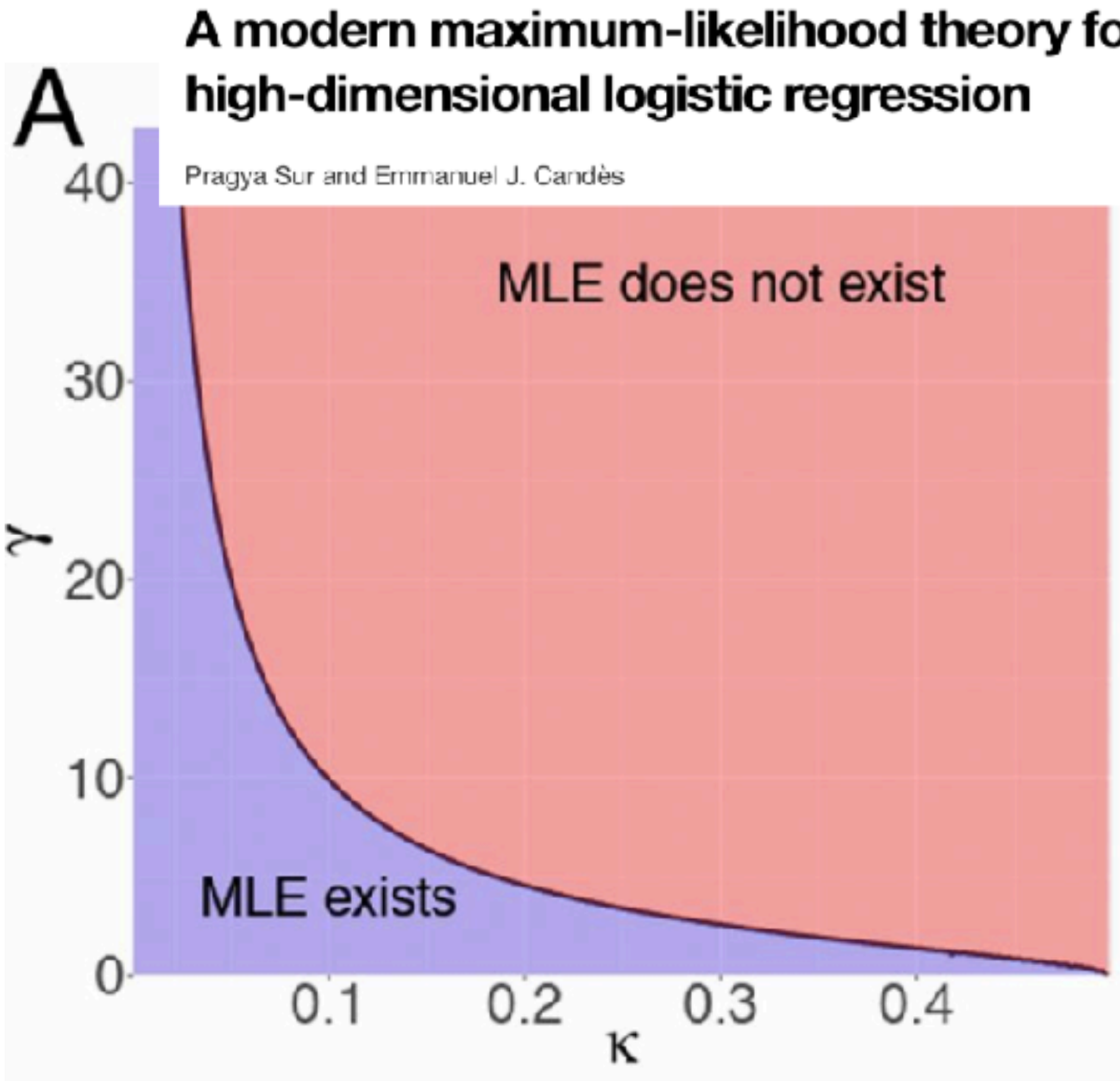
Adrian Weller
University of Cambridge and Alan Turing Institute
aw665@cam.ac.uk

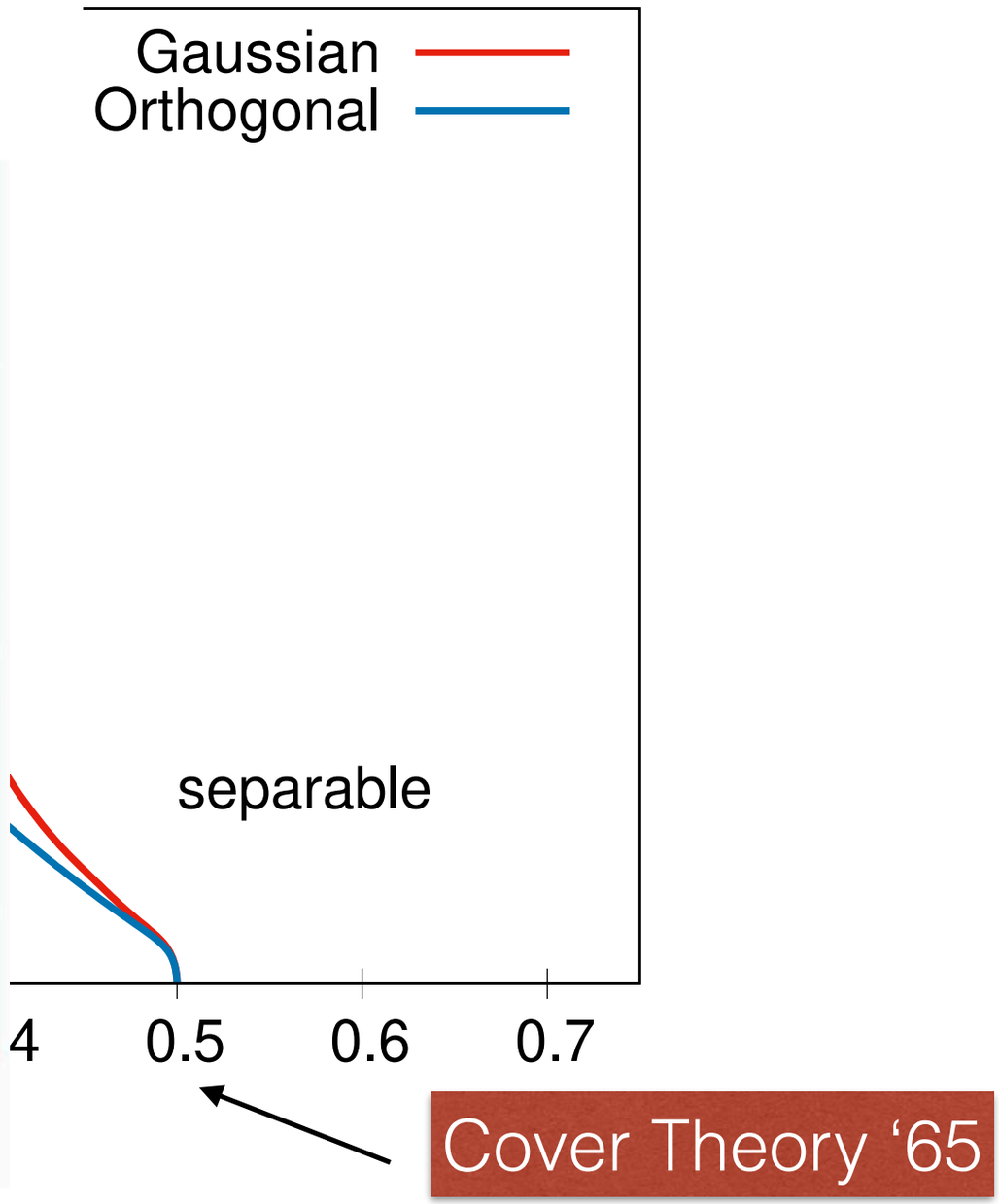
Nips '17

Logistic loss, no regularisation



Phase transition of perfect separability





Generalize a phase transition discussed
by [Cover ’65; Gardner ’87; Sur & Candes, ’18]

Kernels, random projections
(and lazy training as well) are very limited
when not enough data are available

Phase retrieval with teacher & student

$$y_i = |\mathbf{a}_i \cdot \mathbf{w}| \; for \; i = 1,\ldots,n \qquad \mathbf{w} \in \mathbb{R}^d \text{ or } \mathbb{C}^d \qquad \alpha = \frac{n}{d}$$

Both $\mathbf{a}_i$ and $\mathbf{w}$ are random i.i.d. Gaussian

Method 1:

* Use kernel method to predict y for a new vector **a** given a training set
* Kernel method cannot beat a random guess for any finite α=n/d!
* This is a consequence of the Gaussian Equivalence Theorem (and of the curse of dimensionality), see S. Goldt's lecture next week
* Open problem: does it work with $n=O(d^2)$?

[Loureiro, Gerace, **FK**, Mézard, Zdeborova, '20]

Method 2:

* Use a neural network to predict y for a new vector **a** given a training set
* A 2-layer NN can give good results starting from α=2

[Sarao Mannelli, Vanden-Eijnden, Zdeborová, '20]

“Apparté”

The Neural Tangent Kernel

Empirical Risk Minimisation

$$(\mathbf{X}_i \in \mathbb{R}^d, y_i \in \mathbb{R}), i = 1, \ldots, n$$

$$\mathscr{R} = \frac{1}{n} \sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{X}_i))$$

Ex: neural networks models

Model: $$f_\theta(\mathbf{X}) = \eta^{(0)} \left(\mathbf{W}^{(0)} \eta^{(1)} \left(\mathbf{W}^{(1)} \ldots \eta^{(L)} \left(\mathbf{W}^{(L)} \cdot \mathbf{X} \right) \right) \right)$$

$$\theta = \{\mathbf{W}^{(0)}, \mathbf{W}^{(1)}, \ldots, \mathbf{W}^{(L)}\}$$

Loss : $\mathscr{L}(y, h)$

Empirical Risk Minimisation

$$(\mathbf{X}_i \in \mathbb{R}^d, y_i \in \mathbb{R}), i = 1, \ldots, n$$

$$\mathscr{R} = \frac{1}{n} \sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{X}_i))$$

$$\frac{d}{dt} f_\theta(\mathbf{X}) = (\nabla_\theta f_\theta(\mathbf{X})) \cdot \frac{d\theta}{dt}$$

$$\dot{\theta}^t = -\nabla_\theta \mathscr{R}$$

$$\frac{d}{dt} f_\theta(\mathbf{X}) = -(\nabla_\theta f_\theta(\mathbf{X})) \cdot \left[\frac{1}{n} \sum_{i=1}^{n} \mathscr{L}'(y_i, f_\theta(\mathbf{X}_i)) \nabla_\theta f_\theta(\mathbf{X}_i) \right]$$

$$\frac{d}{dt} f_\theta(\mathbf{X}) = -\left[\frac{1}{n} \sum_{i=1}^{n} \Omega^t(\mathbf{X}, \mathbf{X}_i) \mathscr{L}'(y_i, f_\theta(\mathbf{X}_i)) \right]$$

$$\Omega^t(\mathbf{X}, \mathbf{Y}) = \nabla_\theta f_\theta(\mathbf{X}) \cdot \nabla_\theta f_\theta(\mathbf{Y})$$

Empirical Risk Minimisation

$$(\mathbf{X}_i \in \mathbb{R}^d, y_i \in \mathbb{R}), i = 1,\ldots,n$$

$$\mathscr{R} = \frac{1}{n}\sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{X}_i))$$

$$\frac{d}{dt} f_\theta(\mathbf{X}) = (\nabla_\theta f_\theta(\mathbf{X})) \cdot \frac{d\theta}{dt}$$

$$\dot{\theta}^t = -\nabla_\theta \mathscr{R}$$

$$\frac{d}{dt} f_\theta(\mathbf{X}) = -(\nabla_\theta f_\theta(\mathbf{X})) \cdot \left[\frac{1}{n}\sum_{i=1}^{n} \mathscr{L}'(y_i, f_\theta(\mathbf{X}_i))\nabla_\theta f_\theta(\mathbf{X}_i)\right]$$

$$\frac{d}{dt} f_\theta(\mathbf{X}) = -\left[\frac{1}{n}\sum_{i=1}^{n} \Omega^t(\mathbf{X}, \mathbf{X}_i)\mathscr{L}'(y_i, f_\theta(\mathbf{X}_i))\right] = -\left[\frac{1}{n}\sum_{i=1}^{n} \Omega^t(\mathbf{X}, \mathbf{X}_i)\dot{\beta}_i^t\right]$$

$$\Omega^t(\mathbf{X}, \mathbf{Y}) = \nabla_\theta f_\theta(\mathbf{X}) \cdot \nabla_\theta f_\theta(\mathbf{Y})$$

Empirical Risk Minimisation

$$(\mathbf{X}_i \in \mathbb{R}^d, y_i \in \mathbb{R}), i = 1, \ldots, n$$

$$\mathscr{R} = \frac{1}{n} \sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{X}_i))$$

Assume that $\Omega^t(\mathbf{X}, \mathbf{Y})$ **converges to** $\Omega^*(\mathbf{X}, \mathbf{Y})$ **during training reminds equal to**

$$\Omega^*(\mathbf{X}, \mathbf{Y}) = \Omega^{t=0}(\mathbf{X}, \mathbf{Y})$$

$$\frac{d}{dt} f_\theta(\mathbf{X}) = -\left[\frac{1}{n} \sum_{i=1}^{n} \Omega^t(\mathbf{X}, \mathbf{X}_i) \mathscr{L}'(y_i, f_\theta(\mathbf{X}_i))\right] = -\left[\frac{1}{n} \sum_{i=1}^{n} \Omega^t(\mathbf{X}, \mathbf{X}_i) \dot{\beta}_i^t\right]$$

$$\Omega^t(\mathbf{X}, \mathbf{Y}) = \nabla_\theta f_\theta(\mathbf{X}) \cdot \nabla_\theta f_\theta(\mathbf{Y})$$

Empirical Risk Minimisation

$$(\mathbf{X}_i \in \mathbb{R}^d, y_i \in \mathbb{R}), i = 1, \ldots, n$$

$$\mathscr{R} = \frac{1}{n} \sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{X}_i))$$

Assume that $\Omega^t(\mathbf{X}, \mathbf{Y})$ **converges to** $\Omega^*(\mathbf{X}, \mathbf{Y})$ **during training**

reminds equal to

$$\Omega^*(\mathbf{X}, \mathbf{Y}) = \Omega^{t=0}(\mathbf{X}, \mathbf{Y})$$

$$\frac{d}{dt} f_\theta(\mathbf{X}) = -\left[\frac{1}{n} \sum_{i=1}^{n} \Omega^*(\mathbf{X}, \mathbf{X}_i) \dot{\beta}_i^t\right]$$

$$\Omega^t(\mathbf{X}, \mathbf{Y}) = \nabla_\theta f_\theta(\mathbf{X}) \cdot \nabla_\theta f_\theta(\mathbf{Y})$$

Empirical Risk Minimisation

$$(\mathbf{X}_i \in \mathbb{R}^d, y_i \in \mathbb{R}), i = 1, \ldots, n \qquad \mathscr{R} = \frac{1}{n} \sum_{i=1}^{n} \mathscr{L}(y_i, f_\theta(\mathbf{X}_i))$$

$$\frac{d}{dt} f_\theta(\mathbf{X}) = -\left[\frac{1}{n} \sum_{i=1}^{n} \Omega^*(\mathbf{X}, \mathbf{X}_i) \dot{\beta}_i^t \right]$$

$$\Omega^t(\mathbf{X}, \mathbf{Y}) = \nabla_\theta f_\theta(\mathbf{X}) \cdot \nabla_\theta f_\theta(\mathbf{Y})$$

Remember that for Kernel we had: $$f_\beta(\mathbf{X}) = \sum_{j=1}^{n} \beta_j K(\mathbf{X}_j, \mathbf{X})$$

NTK is a random features net

$$\Phi = \nabla_\theta f_\theta(\mathbf{X})$$

$$f_\theta(\mathbf{X}) = \sum_i a_i \sigma\left(\mathbf{b_i X}\right)$$

$$\Phi^{(1)} = \begin{pmatrix} \sigma(\mathbf{b}_1\mathbf{X}) \\ \sigma(\mathbf{b}_2\mathbf{X}) \\ \cdots \\ \sigma(\mathbf{b}_K\mathbf{X}) \end{pmatrix} = \sigma(B\mathbf{X})$$

$$\Phi = \begin{pmatrix} \Phi^{(1)} \\ \Phi^{(2)} \end{pmatrix}$$

$$\Phi^{(2)} = \begin{pmatrix} a_1\sigma'(\mathbf{b}_1\mathbf{X})x_1 \\ a_1\sigma'(\mathbf{b}_1\mathbf{X})x_2 \\ \cdots \\ a_i\sigma'(\mathbf{b}_i\mathbf{X})x_j \\ \cdots \\ a_k\sigma'(\mathbf{b}_K\mathbf{X})x_d \end{pmatrix} = \begin{pmatrix} a_1\sigma'(\mathbf{b}_1\mathbf{X})\mathbf{x} \\ a_2\sigma'(\mathbf{b}_2\mathbf{X})\mathbf{x} \\ \cdots \\ a_K\sigma'(\mathbf{b}_K\mathbf{X})\mathbf{x} \end{pmatrix}$$

REM1: NTK yields a random feature models with correlated features. For a given Kernel, i.d.d. random features will always approximate the kernel better

REM2: Note that NTK has has many features that the original neural net, so no real advantage in training complexity! The advantage is that now the objective is convex!