EE-559 – Deep learning

2. Learning, capacity, basic embeddings

François Fleuret
https://fleuret.org/dlc/
February 15, 2018

Learning from data

The general objective of machine learning is to capture regularity in data to make predictions.

In our regression example, we modeled age and blood pressure as being linearly related, to predict the latter from the former.

There are multiple types of inference that we can roughly split into three categories:

- Classification (*e.g.* object recognition, cancer detection, speech processing),
- regression (*e.g.* customer satisfaction, stock prediction, epidemiology), and
- density estimation (*e.g.* outlier detection, data visualization, sampling/synthesis).

The standard formalization considers a measure of probability

$$\mu_{X,Y}$$

over the observation/value of interest, and i.i.d. training samples

$$(x_n, y_n), \ n = 1, \ldots, N.$$

Intuitively, for classification it can often be interpret as

$$\mu_{X,Y}(x,y) = \mu_{X|Y=y}(x)\,\mathbb{P}(Y=y)$$

that is, draw Y first, and given its value, generate X.

Here

$$\mu_{X|Y=y}$$

stands for the population of the observable signal for class y (*e.g.* "sound of an /ē/", "image of a cat").

For regression, one would interpret the joint law more naturally as

$$\mu_{X,Y}(x,y) = \mu_{Y|X=x}(y)\,\mu_X(x)$$

which would be: first, generate X, and given its value, generate Y.

In the simple cases

$$Y = f(X) + \epsilon$$

where f is the deterministic dependency between x and y, and ϵ is a random noise.

With such a model, we can more precisely define the three types of inferences we introduced before:

Classification,

- (X, Y) random variables on $\mathcal{X} = \mathbb{R}^D \times \{1, \ldots, C\}$,
- we want to estimate $\text{argmax}_y \mathbb{P}(Y = y \mid X = x)$.

Regression,

- (X, Y) random variables on $\mathcal{X} = \mathbb{R}^D \times \mathbb{R}$,
- we want to estimate $\mathbb{E}(Y \mid X = x)$.

Density estimation,

- X random variable on $\mathcal{X} = \mathbb{R}^D$,
- we want to estimate μ_X.

The boundaries between these categories are fuzzy:

- Regression allows to do classification through class scores.
- Density models allow to do classification thanks to Bayes' law.

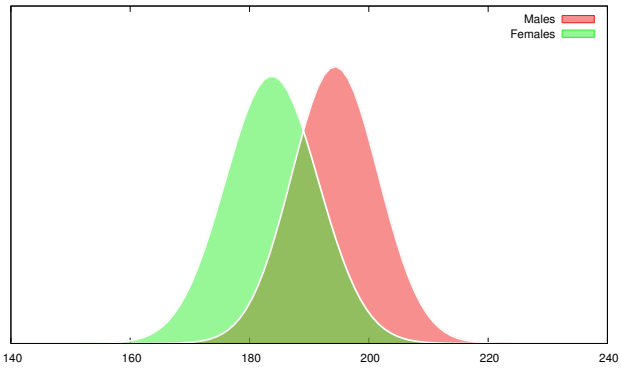
etc.

We call **generative** classification methods with an explicit data model, and **discriminative** the ones bypassing such a modeling .

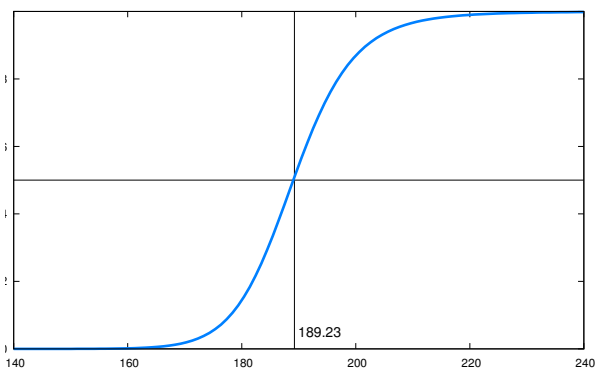
Example: Can we predict a Brazilian basketball player's gender G from his/her height H?

Females:	190	182	188	184	196	173	180	193	179	186	185	169
Males:	192	190	183	199	200	190	195	184	190	203	205	201

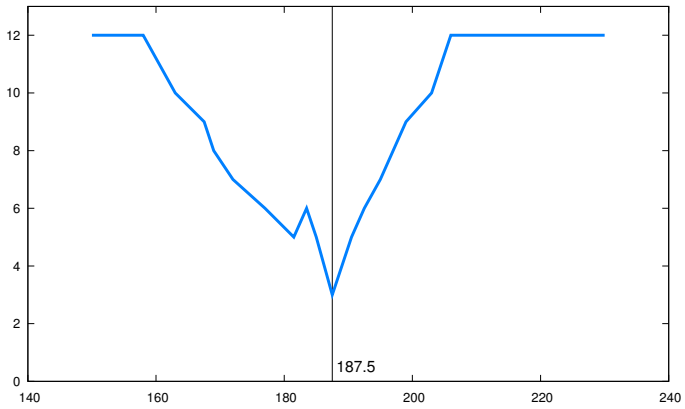
In the **generative** approach, we model $\mu_{H|G=g}(h)$



and use Bayes's law $\mathbb{P}(G = g \mid H = h) = \frac{\mu_{H|G=g}(h)\mathbb{P}(G=g)}{\mu_H(h)}$



In the **discriminative** approach we directly pick the threshold that works the best on the data:



Note that it is harder to design a confidence indicator.

Risk, empirical risk

Learning consists of finding in a set $\mathscr{F}$ of functionals a "good" f^* (or its parameters' values) usually defined through a loss

$$\ell : \mathscr{F} \times \mathscr{Z} \to \mathbb{R}$$

such that $\ell(f, z)$ increases with how wrong f is on z. For instance

- for classification:
$$\ell(f, (x, y)) = \mathbf{1}_{\{f(x) \neq y\}},$$
- for regression:
$$\ell(f, (x, y)) = (f(x) - y)^2,$$
- for density estimation:
$$\ell(q, z) = -\log q(z).$$

The loss may include additional terms related to f itself.

We are looking for an f with a small **expected risk**

$$R(f) = \mathbb{E}_Z \left(\ell(f, Z) \right),$$

which means that our learning procedure would ideally choose

$$f^* = \operatorname*{argmin}_{f \in \mathscr{F}} R(f).$$

Although this quantity is unknown, if we have i.i.d. training samples

$$\mathscr{D} = \{Z_1, \ldots, Z_N\},$$

we can compute an estimate, the **empirical risk**:

$$\hat{R}(f; \mathscr{D}) = \hat{\mathbb{E}}_{\mathscr{D}}(\ell(f, Z)) = \frac{1}{N} \sum_{n=1}^{N} \ell(f, Z_n).$$

We have

$$\begin{aligned}
\mathbb{E}_{Z_1,\dots,Z_N}\left(\hat{R}(f;\mathscr{D})\right) &= \mathbb{E}_{Z_1,\dots,Z_N}\left(\frac{1}{N}\sum_{n=1}^{N}\ell(f,Z_n)\right) \\
&= \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}_{Z_n}\left(\ell(f,Z_n)\right) \\
&= \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}_{Z}\left(\ell(f,Z)\right) \\
&= \mathbb{E}_{Z}\left(\ell(f,Z)\right) \\
&= R(f).
\end{aligned}$$

The empirical risk is an **unbiased estimator** of the expected risk.

Finally, given $\mathscr{D}$, $\mathscr{F}$, and ℓ, "learning" aims at computing

$$f^* = \operatorname*{argmin}_{f \in \mathscr{F}} \hat{R}(f;\mathscr{D}).$$

- Can we bound $R(f)$ with $\hat{R}(f,\mathscr{D})$?

  Yes if f is not chosen using $\mathscr{D}$. Since the Z_n are independent, we just need to take into account the variance of $\hat{R}(f,\mathscr{D})$.

- Can we bound $R(f^*)$ with $\hat{R}(f^*,\mathscr{D})$?

  ⚠ Unfortunately not simply, and not without additional constraints on $\mathscr{F}$.

  For instance if $|\mathscr{F}| = 1$, we can!

Note that in practice, we call "loss" both the functional

$$\ell : \mathscr{F} \times \mathscr{Z} \to \mathbb{R}$$

and the empirical risk minimized during training

$$\mathscr{L}(f) = \frac{1}{N} \sum_{n=1}^{N} \ell(f, z_n).$$

Over and under-fitting, intuition

You want to hire someone, and you evaluate candidates by asking them ten technical yes/no questions.

Would you feel confident if you interviewed one candidate and he makes a perfect score?

What about interviewing ten candidates and picking the best?

What about one thousand?

With

$$Q_k^n \sim \mathscr{B}(0.5), \ \ n = 1, \ldots, 1000, \ \ k = 1, \ldots, 10,$$

independent, we have

$$\forall n, \ \ \mathbb{P}(\forall k, Q_k^n = 1) = \frac{1}{1024}$$

and

$$\mathbb{P}(\exists n, \forall k, Q_k^n = 1) \simeq 0.62.$$

There is 62% chance that among $1,000$ candidates answering completely at random, one will score perfectly.

It is quite intuitive that selecting a candidate based on a statistical estimator biases the said estimator for that candidate.

Over and under-fitting, capacity. K-nearest-neighbors

A simple classification procedure is the "K-nearest neighbors."

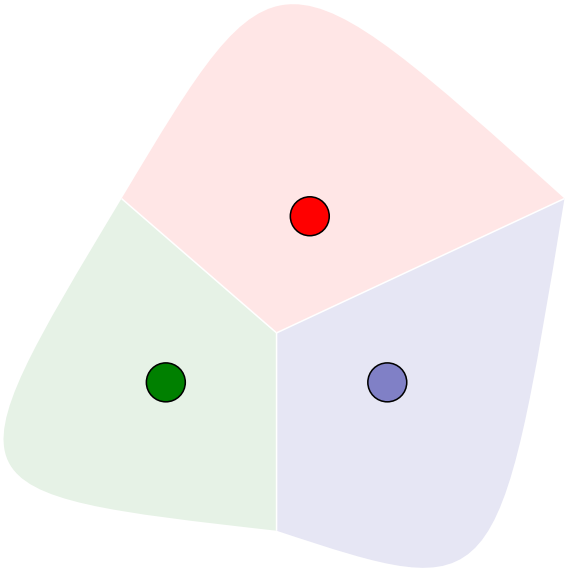
Given

$$(x_n, y_n) \in \mathbb{R}^D \times \{1, \dots, C\}, \ n = 1, \dots, N$$

to predict the y associated to a new x, take the y_n of the closest x_n:

$$\begin{aligned} n^*(x) &= \operatorname*{argmin}_n \|x_n - x\| \\ f^*(x) &= y_{n^*(x)}. \end{aligned}$$

This recipe corresponds to $K = 1$, and makes the empirical training error zero.

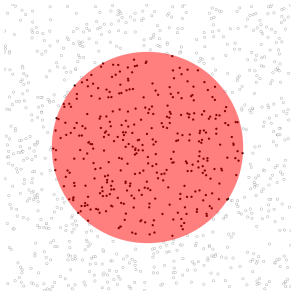


K = 1

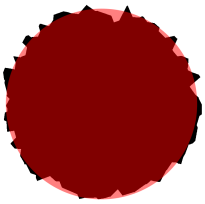
Under mild assumptions of regularities of $\mu_{X,Y}$, for $N \to \infty$ the asymptotic error rate of the 1-NN is less than twice the (optimal!) Bayes' Error rate.

It can be made more stable by looking at the $K > 1$ closest training points, and taking the majority vote.

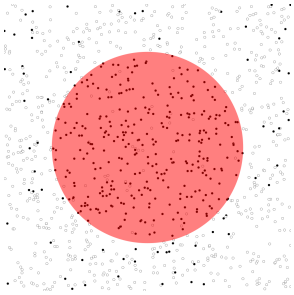
If we let also $K \to \infty$ "not too fast", the error rate is the (optimal!) Bayes' Error rate.



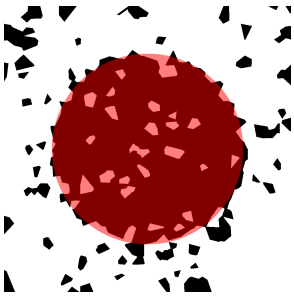
Training set



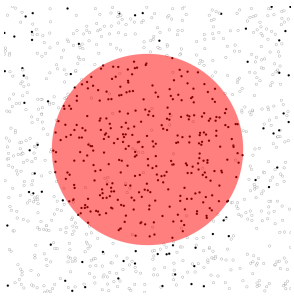
Prediction (K=1)



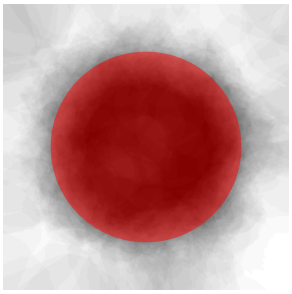
Training set



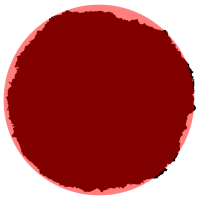
Prediction (K=1)



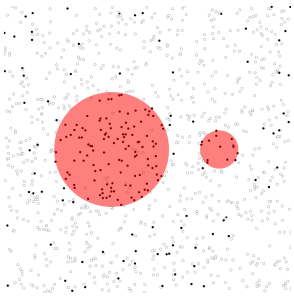
Training set



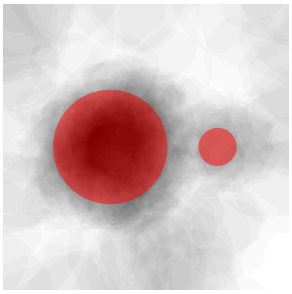
Votes (K=51)



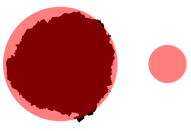
Prediction (K=51)



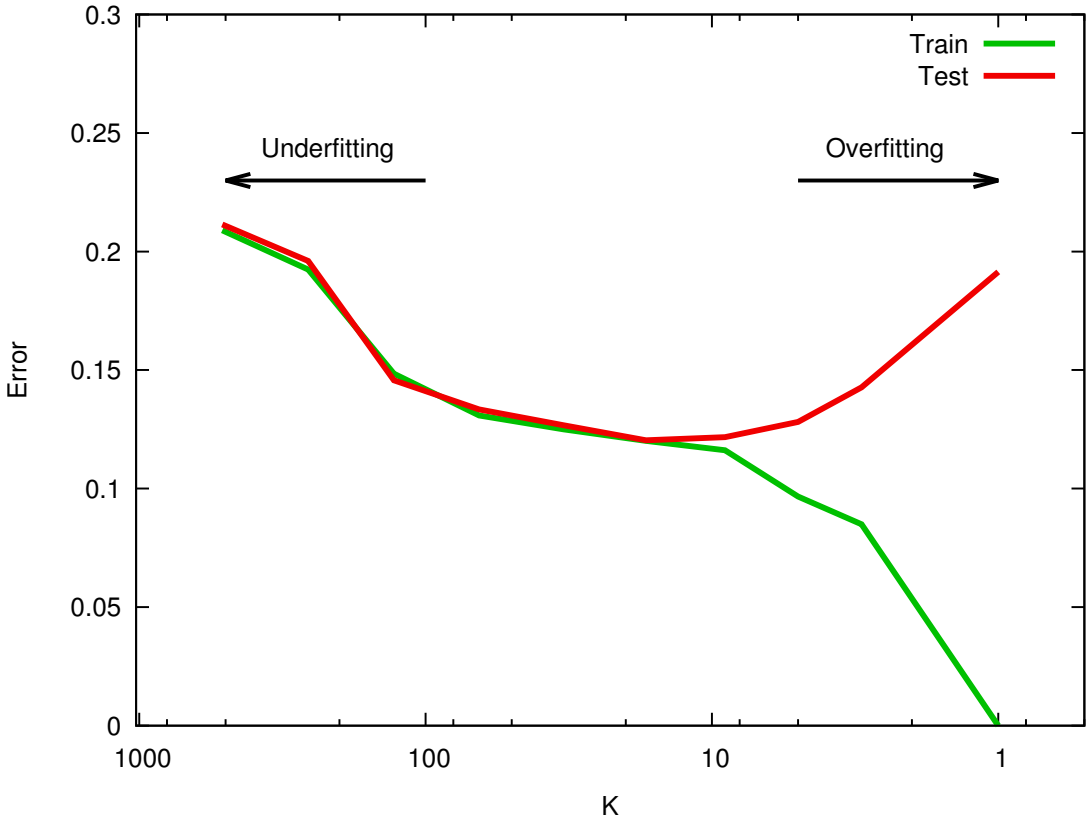
Training set



Votes (K=51)



Prediction (K=51)



Over and under-fitting, capacity, polynomials

Consider a polynomial model

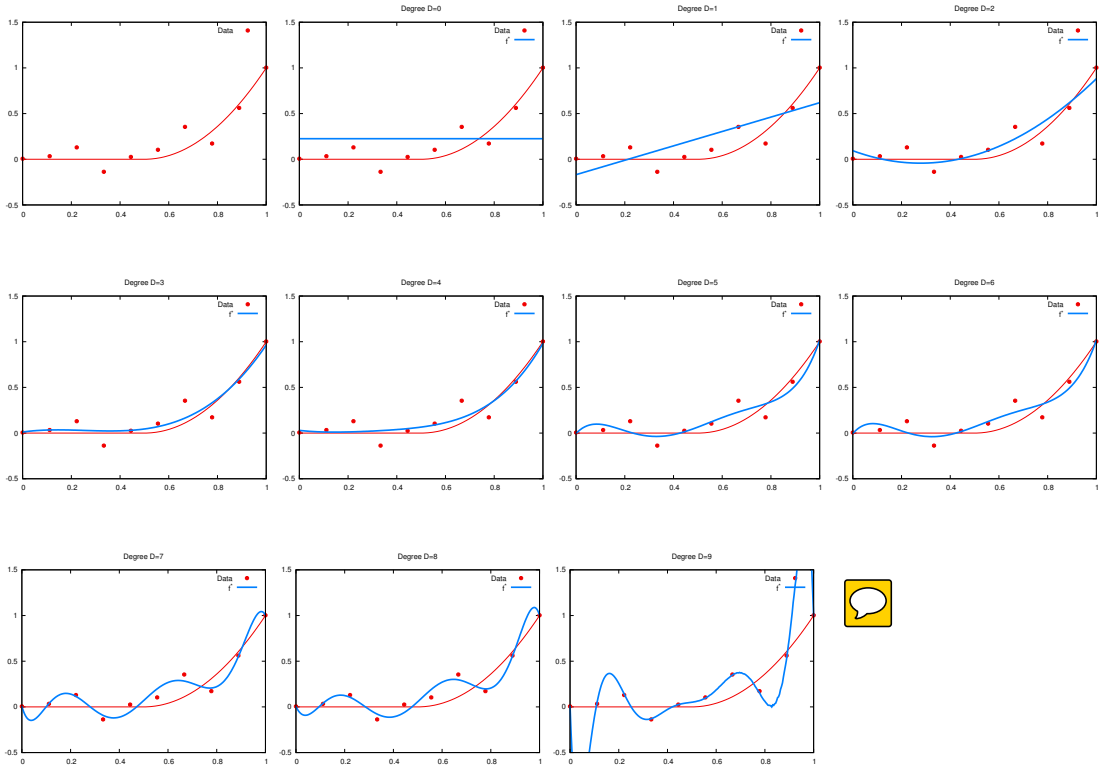
$$\forall x, \alpha_0, \ldots, \alpha_D \in \mathbb{R},\ f(x;\alpha) = \sum_{d=0}^{D} \alpha_d x^d.$$

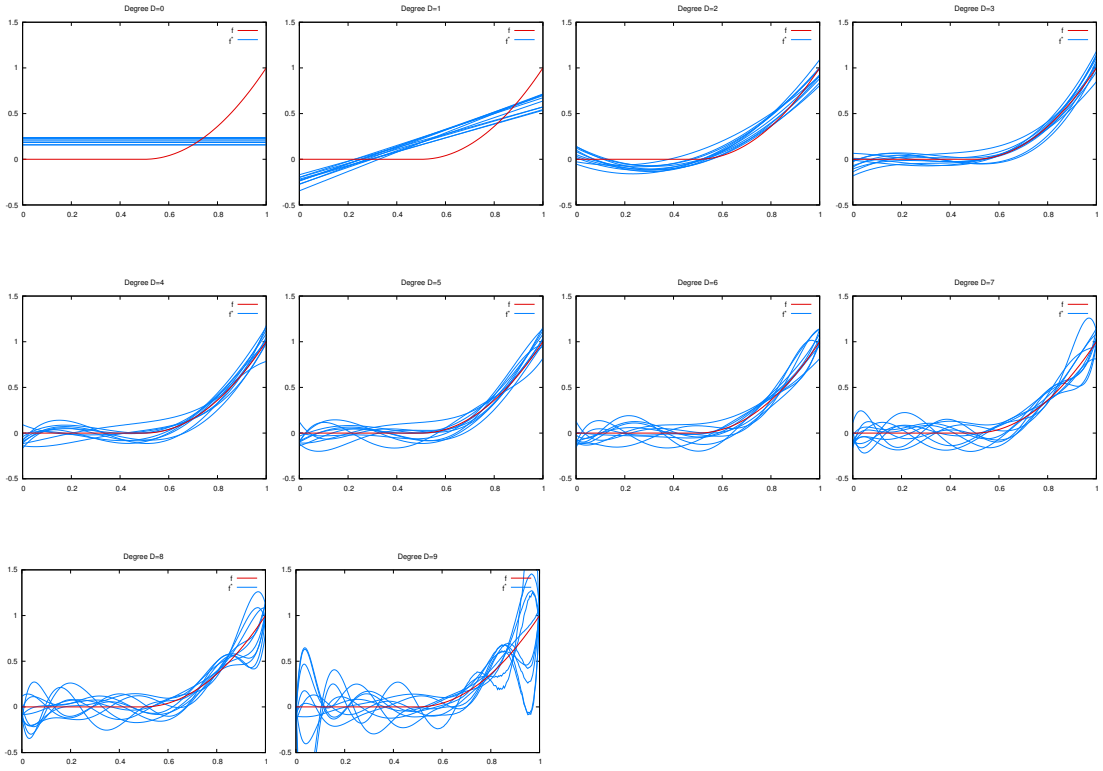
and training points $(x_n, y_n) \in \mathbb{R}^2, n = 1, \ldots, N$, minimize the quadratic loss

$$\begin{aligned}
\mathscr{L}(\alpha) &= \sum_n (f(x_n;\alpha) - y_n)^2 \\
&= \sum_n \left(\sum_{d=0}^{D} \alpha_d x_n^d - y_n \right)^2 \\
&= \left\| \begin{pmatrix} x_1^0 & \ldots & x_1^D \\ \vdots & & \vdots \\ x_N^0 & \ldots & x_N^D \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_D \end{pmatrix} - \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix} \right\|^2.
\end{aligned}$$

This is a standard quadratic problem, for which we have efficient algorithms.


```
def fit_polynomial(D, x, y):
    N = x.size(0)
    X = Tensor(N, D + 1)
    Y = Tensor(N, 1)

    # Exercise: avoid the n loop
    for n in range(N):
        for d in range(D + 1):
            X[n, d] = x[n]**d
        Y[n, 0] = y[n]

    # LAPACK's GEneralized Least-Square
    alpha, _ = torch.gels(Y, X)

    return alpha
```


Degree D=0
Degree D=1
Degree D=2
Degree D=3
Degree D=4
Degree D=5
Degree D=6
Degree D=7
Degree D=8
Degree D=9
Data
f*


Degree D=0
Degree D=1
Degree D=2
Degree D=3
Degree D=4
Degree D=5
Degree D=6
Degree D=7
Degree D=8
Degree D=9
f
f*

We can reformulate this control of the degree with a penalty

$$\mathscr{L}(\alpha) = \sum_n (f(x_n;\alpha) - y_n)^2 + \sum_d l_d(\alpha_d)$$

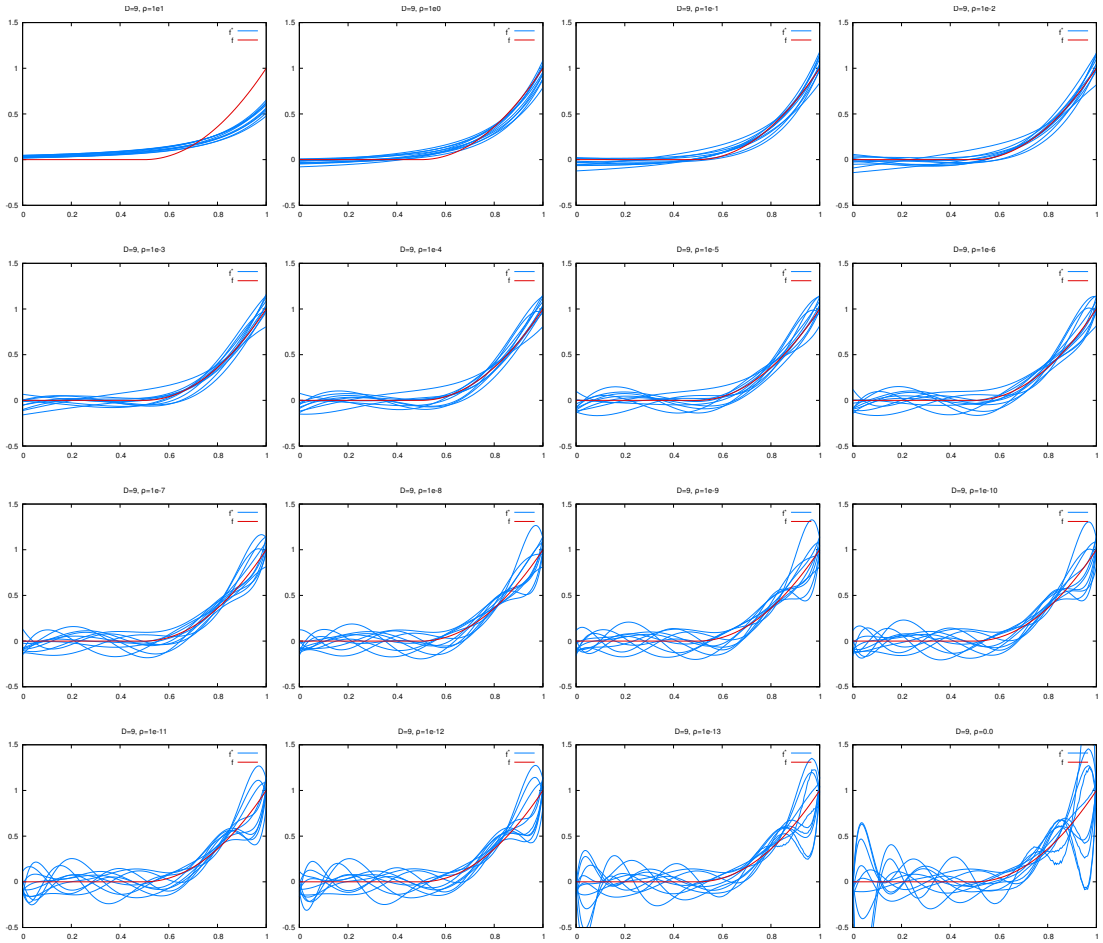
where

$$l_d(\alpha) = \begin{cases} 0 & \text{if } d \leq D \text{ or } \alpha = 0 \\ +\infty & \text{otherwise.} \end{cases}$$

Such a penalty kills any term of degree $> D$.

This motivates the use of more subtle variants. For instance, to keep all this quadratic

$$\mathscr{L}(\alpha) = \sum_n (f(x_n;\alpha) - y_n)^2 + \rho \sum_d \alpha_d^2.$$



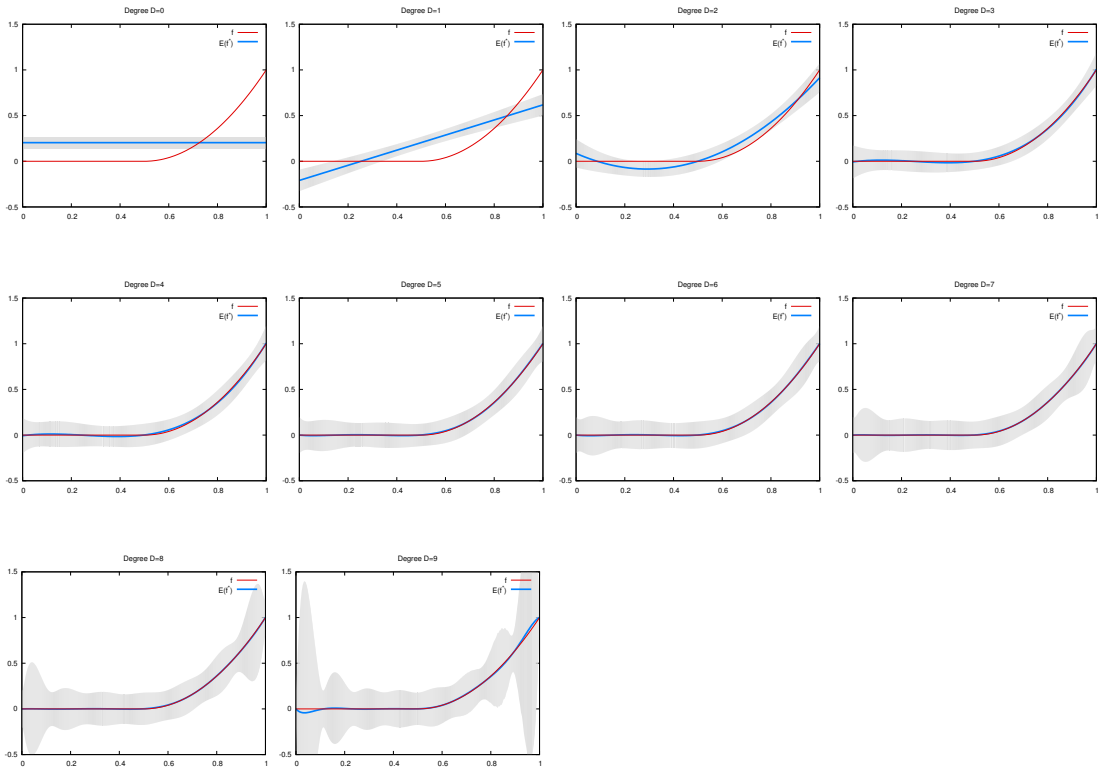
We define the **capacity** of a set of predictors as its ability to model an arbitrary functional.

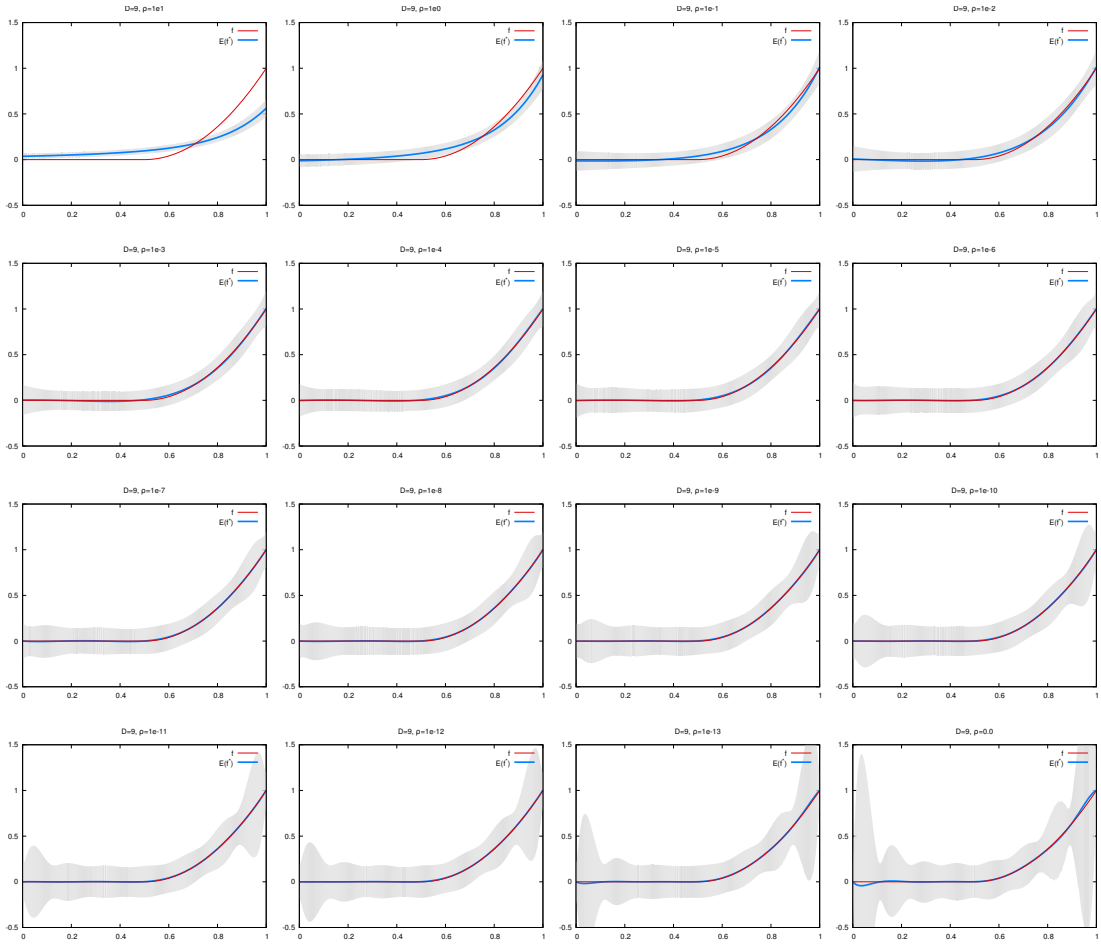
Although it is difficult to define precisely, it is quite clear in practice how to increase or decrease it for a given class of models.

So one can control over-fitting either by

- Impoverishing the space $\mathscr{F}$ (less functionals, early stopping)
- Make the choice of f^* less dependent on data (penalty on coefficients, margin maximization, ensemble methods)

Bias-variance dilemma





Let x be fixed, and $y = f(x)$ the "true" value associated to it.

Let $Y = f^*(x)$ be the value we predict.

If we consider that the training set is a random quantity, so is f^*, and consequently Y.

We have

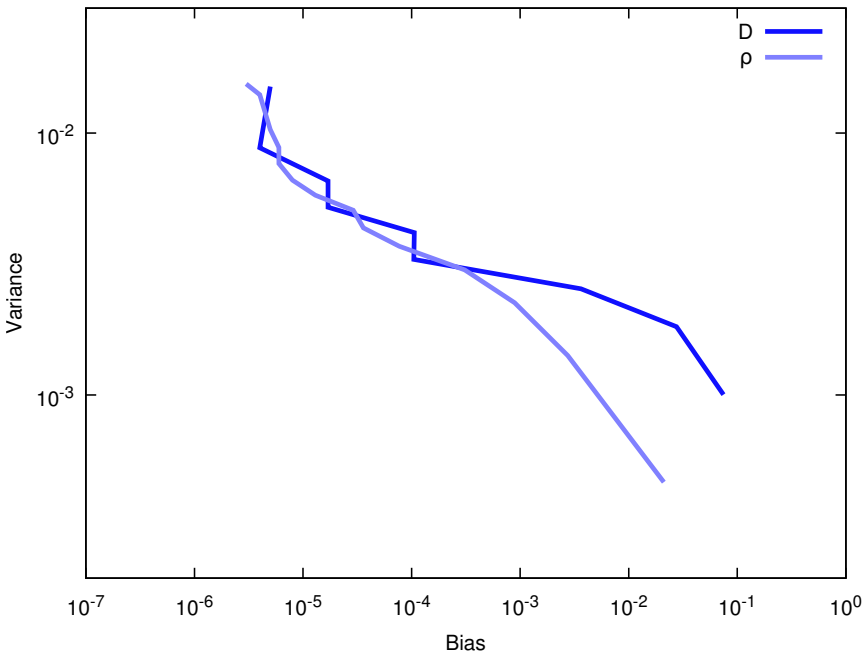
$$\begin{aligned}\mathbb{E}((Y-y)^2) &= \mathbb{E}(Y^2 - 2Yy + y^2) \\ &= \mathbb{E}(Y^2) - 2\mathbb{E}(Y)y + y^2 \\ &= \underbrace{\mathbb{E}(Y^2) - \mathbb{E}(Y)^2}_{\mathbb{V}(Y)} + \underbrace{\mathbb{E}(Y)^2 - 2\mathbb{E}(Y)y + y^2}_{(\mathbb{E}(Y)-y)^2} \\ &= \underbrace{(\mathbb{E}(Y) - y)^2}_{\text{Bias}} + \underbrace{\mathbb{V}(Y)}_{\text{Variance}}\end{aligned}$$

This is the bias-variance decomposition:

- The bias term quantifies how much the model fits on any single data-set,
- the variance term quantifies how much the model changes across data-sets.

(Geman and Bienenstock, 1992)

From this comes the **bias variance tradeoff:**



Reducing the capacity makes f^* fit the data less on average, which increases the bias term. Increasing the capacity makes f^* vary a lot with the training data, which increases the variance term.

Is all this probabilistic?

Conceptually model-fitting and regularization can be interpreted as Bayesian inference.

This approach consists of **modeling the parameters A of the model themselves as random quantities with a prior μ_A.**

By looking at the data $\mathscr{D}$, we can estimate a posterior distribution for the said parameters,

$$\mu_A(\alpha \mid \mathscr{D} = \mathbf{d}) \propto \mu_{\mathscr{D}}(\mathbf{d} \mid A = \alpha)\, \mu_A(\alpha),$$

and from that their most likely values.

So instead of a penalty term, we define a prior distribution, which is usually more intellectually satisfying.

For instance, consider the model

$$\forall d,\ A_d \sim \mathcal{N}(0, \xi),\ \forall n,\ X_n \sim \mu_X,\ \Delta_n \sim \mathcal{N}(0, \sigma)$$

all independent, and

$$\forall n,\ Y_n = \sum_{d=0}^{D} A_d\, X_n^d + \Delta_n.$$

For clarity, let $A = (A_0, \ldots, A_D)$ and $\alpha = (\alpha_0, \ldots, \alpha_D)$.

Remember that $\mathscr{D} = \{(X_1, Y_1), \ldots, (X_N, Y_N)\}$ is the (random) training set and $\mathbf{d} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ is a realization.

$$\begin{aligned}
&\log \mu_A(\alpha \mid \mathscr{D} = \mathbf{d}) \quad \text{Probability of the parameters given the data. Likelihood ?} \\
&\quad = \log \frac{\mu_{\mathscr{D}}(\mathbf{d} \mid A = \alpha)\, \mu_A(\alpha)}{\mu_{\mathscr{D}}(\mathbf{d})} \\
&\quad = \log \mu_{\mathscr{D}}(\mathbf{d} \mid A = \alpha) + \log \mu_A(\alpha) - \log Z \\
&\quad = \log \prod_n \mu(x_n, y_n \mid A = \alpha) + \log \mu_A(\alpha) - \log Z \quad \text{I.I.D} \\
&\quad = \log \prod_n \mu(y_n \mid X_n = x_n, A = \alpha)\, \underbrace{\mu(x_n \mid A = \alpha)}_{= \mu(x_n)} + \log \mu_A(\alpha) - \log Z \quad \text{Independent of a ?} \\
&\quad = \log \prod_n \mu(y_n \mid X_n = x_n, A = \alpha) + \log \mu_A(\alpha) - \log Z' \quad \text{What does mu(x) evaluate to ?} \\
&\quad = \underbrace{-\frac{1}{2\sigma^2} \sum_n \left(y_n - \sum_d \alpha_d x_n^d \right)^2}_{\text{Gaussian noise on } Y\text{s}} \underbrace{- \frac{1}{2\xi^2} \sum_d \alpha_d^2}_{\text{Gaussian prior on } A\text{s}} - \log Z''.
\end{aligned}$$

Taking $\rho = \sigma^2/\xi^2$ gives the penalty term of the previous slides.

Regularization seen through that prism is intuitive: The stronger the prior, the more evidence you need to deviate from it.

Proper evaluation protocols

Learning algorithms, in particular deep-learning ones, require the tuning of many meta-parameters.

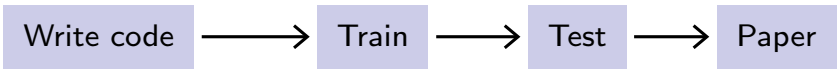
These parameters have a strong impact on the performance, resulting in a "meta" over-fitting through experiments.

We must be extra careful with performance estimation.

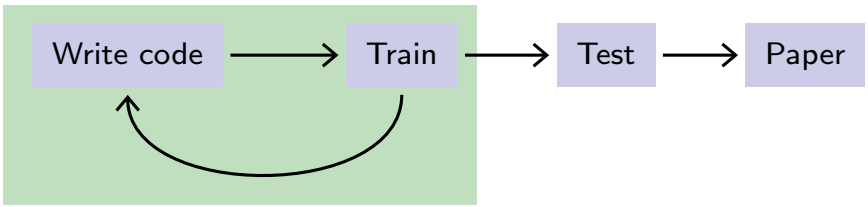
Running 100 times the same experiment on MNIST, with randomized weights, we get:

Worst	Median	Best
1.3%	1.0%	0.82%

The ideal development cycle is

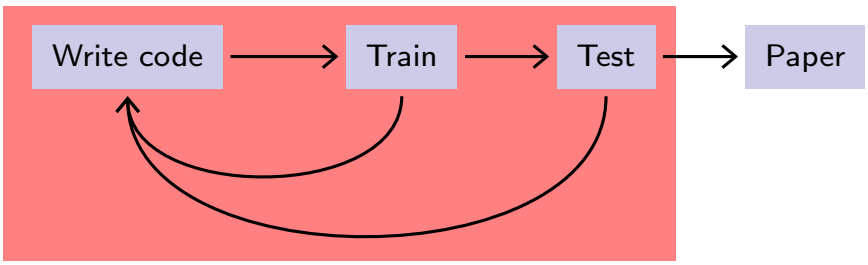


or in practice something like

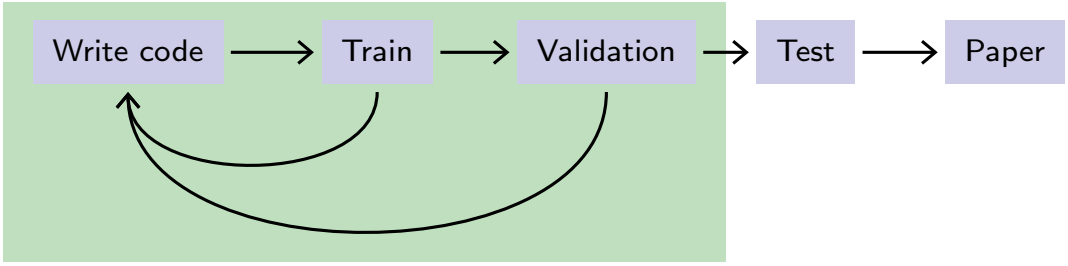


There may be over-fitting, but it does not bias the final performance evaluation.

Unfortunately, it often looks like



This should be avoided at all costs. The standard strategy is to have a separate validation set for the tuning.

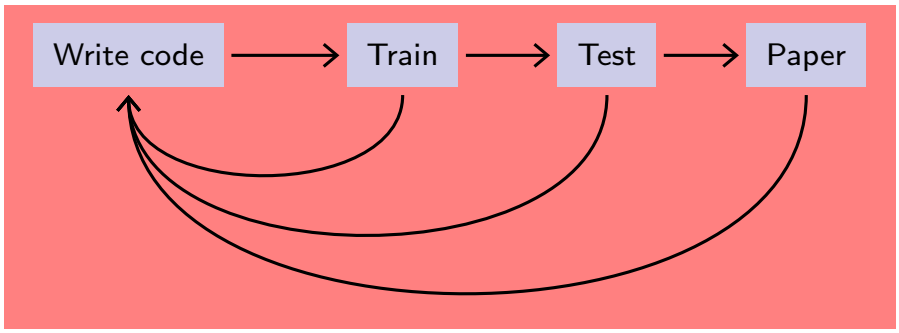


When data is scarce, one can use cross-validation: average through multiple random splits of the data in a train and a validation sets.

There is no unbiased estimator of the variance of cross-validation valid under all distributions (Bengio and Grandvalet, 2004).

Some data-sets (MNIST!) have been used by thousands of researchers, over millions of experiments, in hundreds of papers.

The global overall process looks more like



"Cheating" in machine learning, from bad to "are you kidding?":

- "Early evaluation stopping",
- meta-parameter (over-)tuning,
- data-set selection,
- algorithm data-set specific clauses,
- seed selection.

Top-tier conference are demanding regarding experiments, and are biased against "complicated" pipelines.

The community pushes toward accessible implementations, reference data-sets, leader boards, and constant upgrades of benchmarks.

Standard clustering and embedding

Deep learning models combine embeddings and dimension reduction operations.

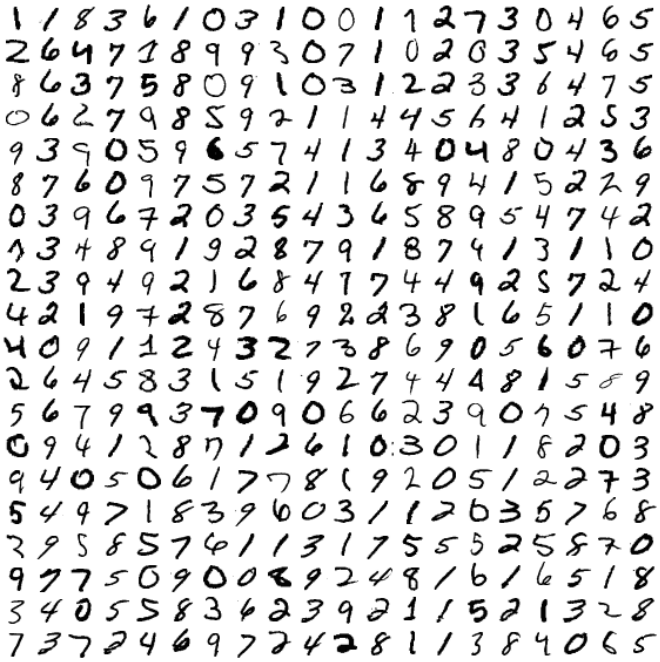
They parametrize and re-parametrize multiple times the input signal under more invariant and interpretable forms.

To get an intuition of how this is possible, we consider here two standard algorithms:

- K-means, and
- Principal Component Analysis (PCA).

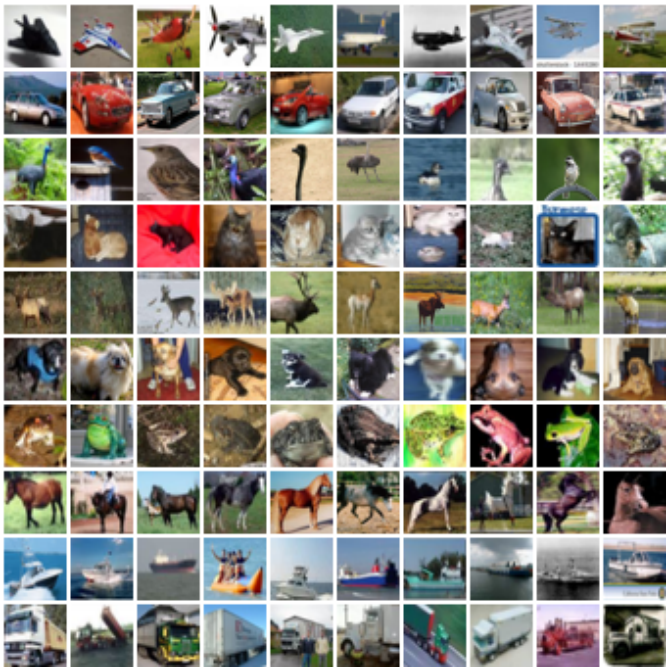
We will illustrate these methods on our two favorite data-sets.

MNIST data-set



28×28 grayscale images, 60k train samples, 10k test samples.

CIFAR10 data-set



32×32 color images, 50k train samples, 10k test samples.

(Krizhevsky, 2009, chap. 3)

Given

$$x_n \in \mathbb{R}^D, \ \ n = 1, \ldots, N,$$

and a fixed number of clusters $K > 0$, K-means tries to find K "centroids" that span uniformly the training population.

Given a point, the index of its closest centroid is a good coding.

Formally, [Lloyd's algorithm for] K-means (approximately) solves

$$\underset{c_1,\ldots,c_K \in \mathbb{R}^D}{\operatorname{argmin}} \sum_n \min_k \|x_n - c_k\|^2.$$

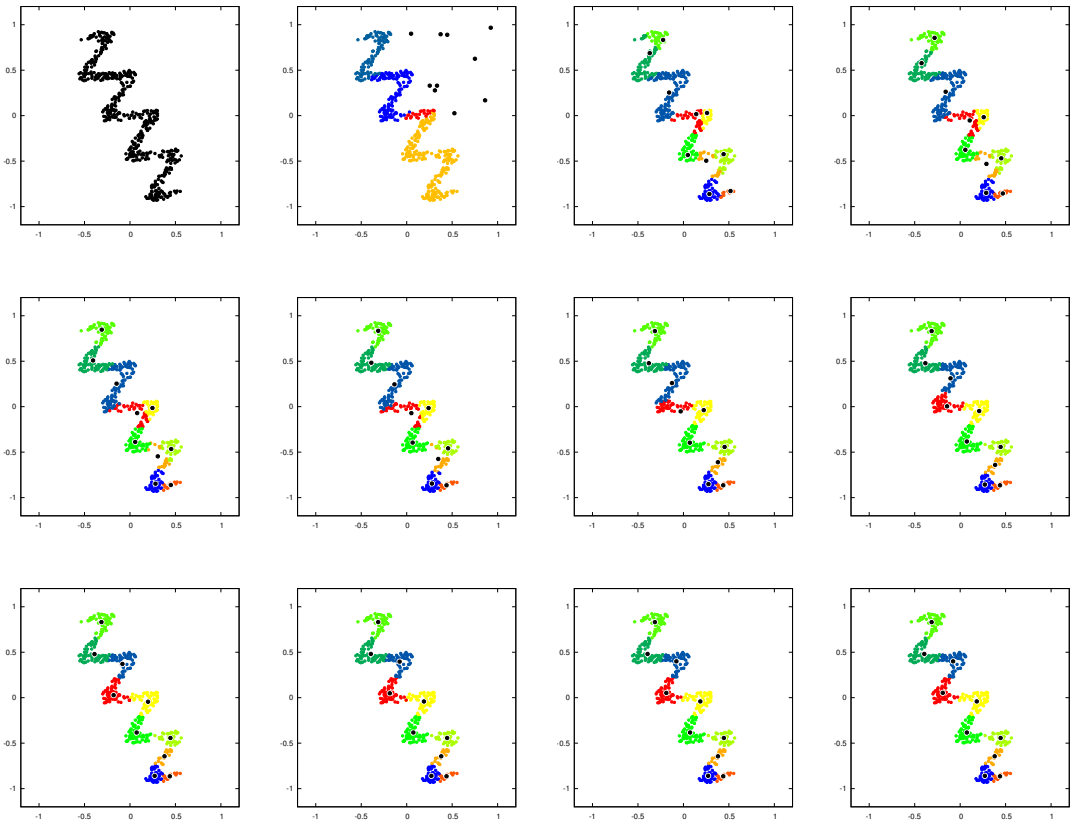
This is achieved with a random initialization of $c_1^0, \ldots c_K^0$ followed by repeating until convergence:

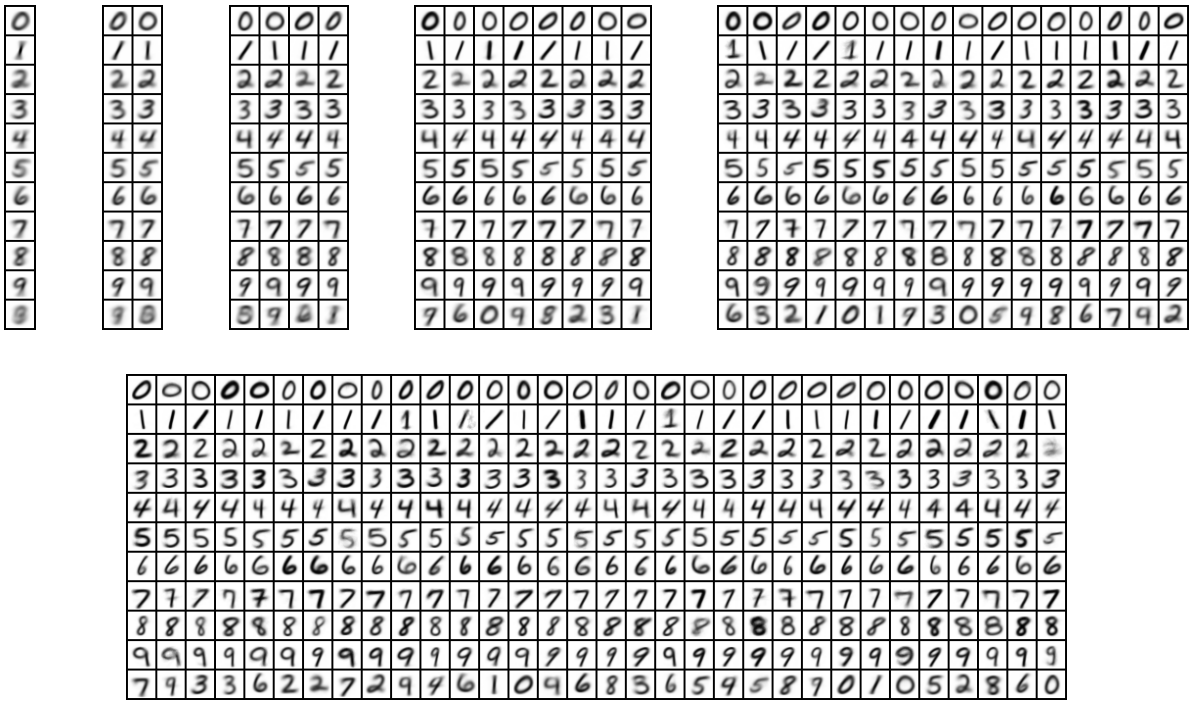
$$\forall n, k_n^t = \underset{k}{\operatorname{argmin}} \|x_n - c_k^t\| \quad (1)$$

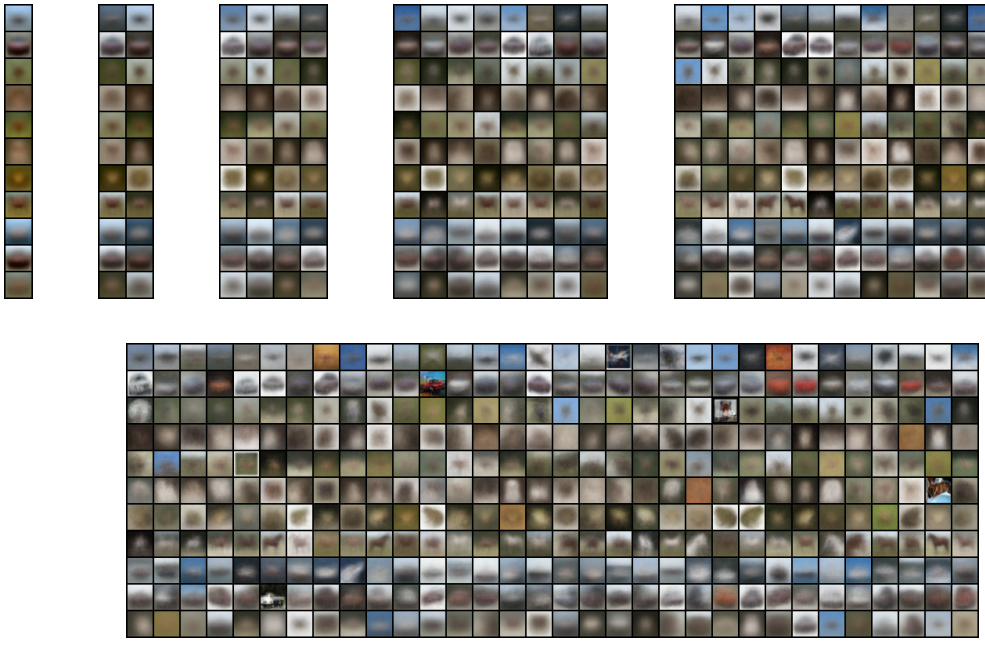
$$\forall k, c_k^{t+1} = \frac{1}{|n : k_n^t = k|} \sum_{n:k_n^t=k} x_n \quad (2)$$

At every iteration, (1) each sample is associated to its closest centroid's cluster, and (2) each centroid is updated to the average of its cluster.

lecture video mentioned how to update cluster centers without any members







The Principal Component Analysis (PCA) similarly aims at extracting an informative summary of the data, also in a L^2 sense.

However, instead of looking for clusters, it looks for an "affine subspace", *i.e.* a point and a vector basis that span the data.

The procedure consists of:

(A) center the data

and then for $t = 1, \ldots, D$,

(B) pick the direction v_t of maximum variance of the data, project data on its orthogonal space.

Formally, given data-points as row-vectors (to stack them into an array)

$$x_n \in \mathbb{R}^{1 \times D}, \ n = 1, \ldots, N$$

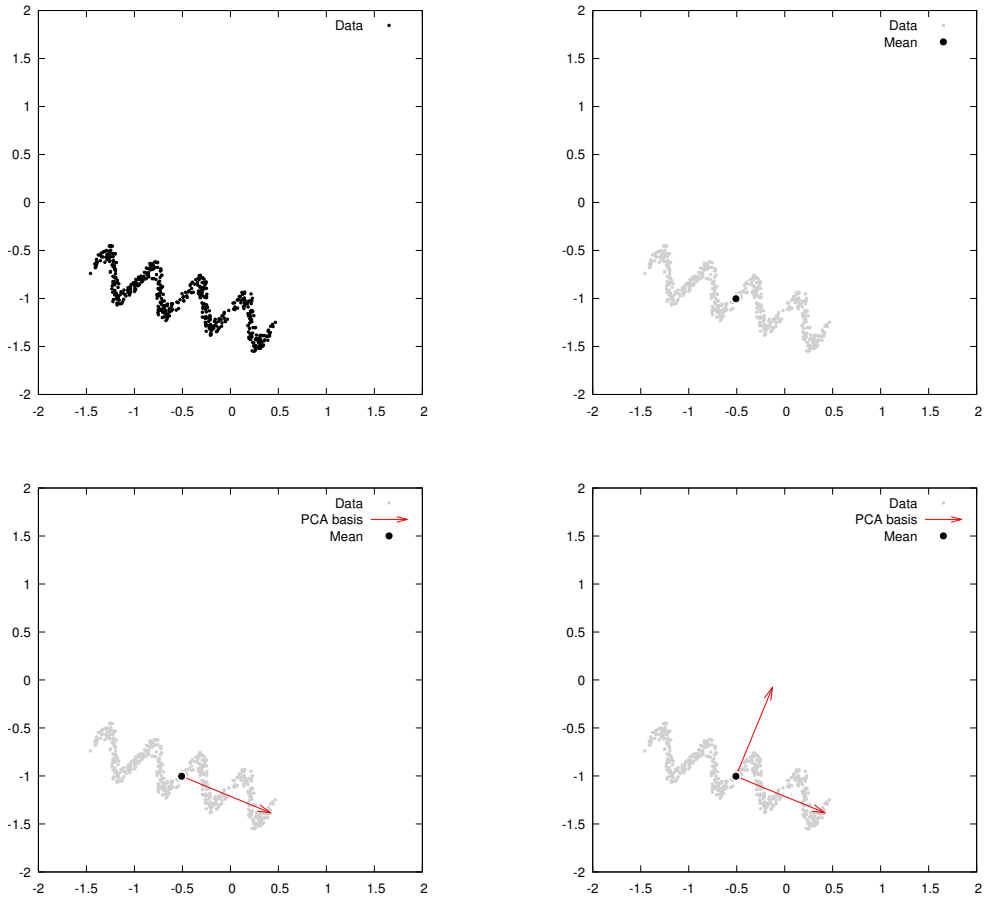
(A) compute the average and center the data

$$\begin{aligned} \bar{x} &= \frac{1}{N} \sum_n x_n \\ \forall n, \ x_n^{(0)} &= x_n - \bar{x} \end{aligned}$$

and then for $t = 1, \ldots, D$,

(B) pick the direction and project the data

$$\begin{aligned} v_t &= \operatorname*{argmax}_{\|v\|=1} \sum_n \left(v \cdot x_n^{(t-1)} \right)^2 \\ \forall n, \ x_n^{(t)} &= x_n^{(t-1)} - \left(v_t \cdot x_n^{(t-1)} \right) v_t. \end{aligned}$$



We have, $\forall v, x_1, \ldots, x_N \in \mathbb{R}^D$,

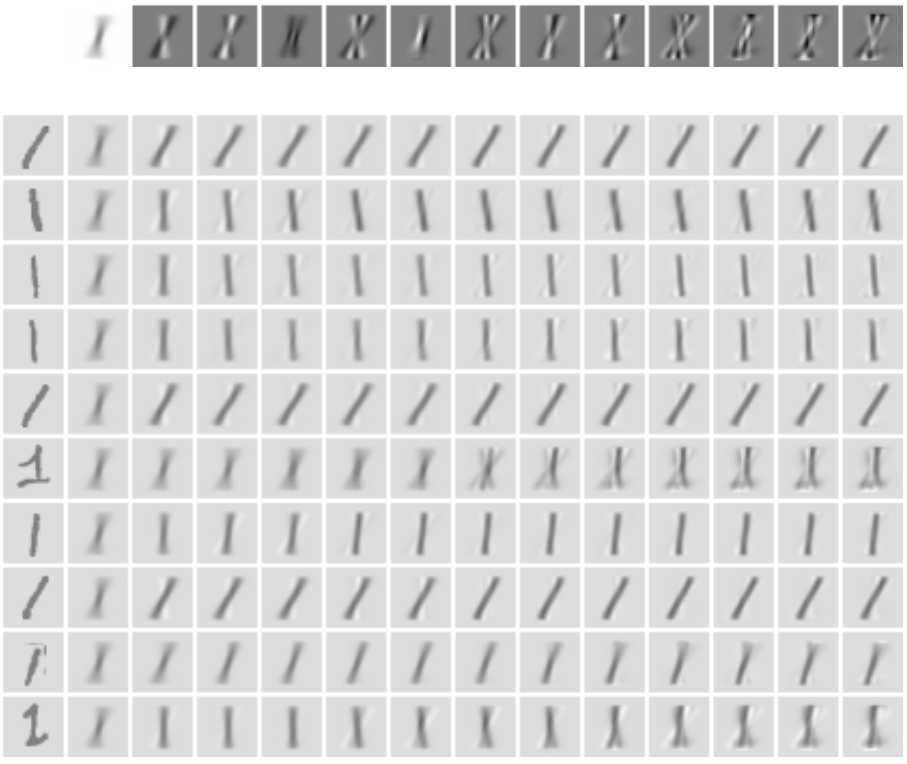
$$\begin{aligned}
\sum_n (v \cdot x_n)^2 &= \| \left((v \cdot x_1), \ldots, (v \cdot x_N) \right) \|_2^2 \\
&= \left\| v \begin{pmatrix} | & & | \\ x_1 & \ldots & x_N \\ | & & | \end{pmatrix} \right\|_2^2 \\
&= \|vX\|_2^2 \\
&= (vX^T)(vX^T)^T \\
&= (vX^T)(Xv^T) \\
&= v(X^TX)v^T.
\end{aligned}$$

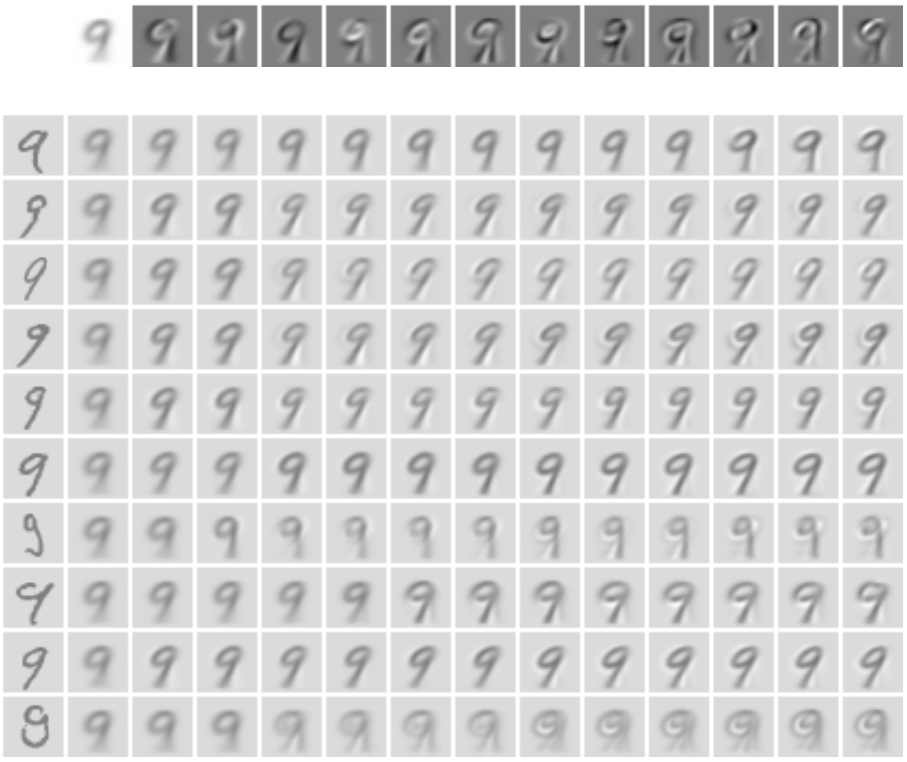
```
>>> x = Tensor(100, 10).uniform_(-10, 10)
>>> v = Tensor(1, 10).normal_()
>>> v.mm(x.t()).pow(2).sum()
34095.442095942795
>>> Sigma = x.t().mm(x)
>>> v.mm(Sigma).mm(v.t())

 34095.4414
[torch.FloatTensor of size 1x1]
```

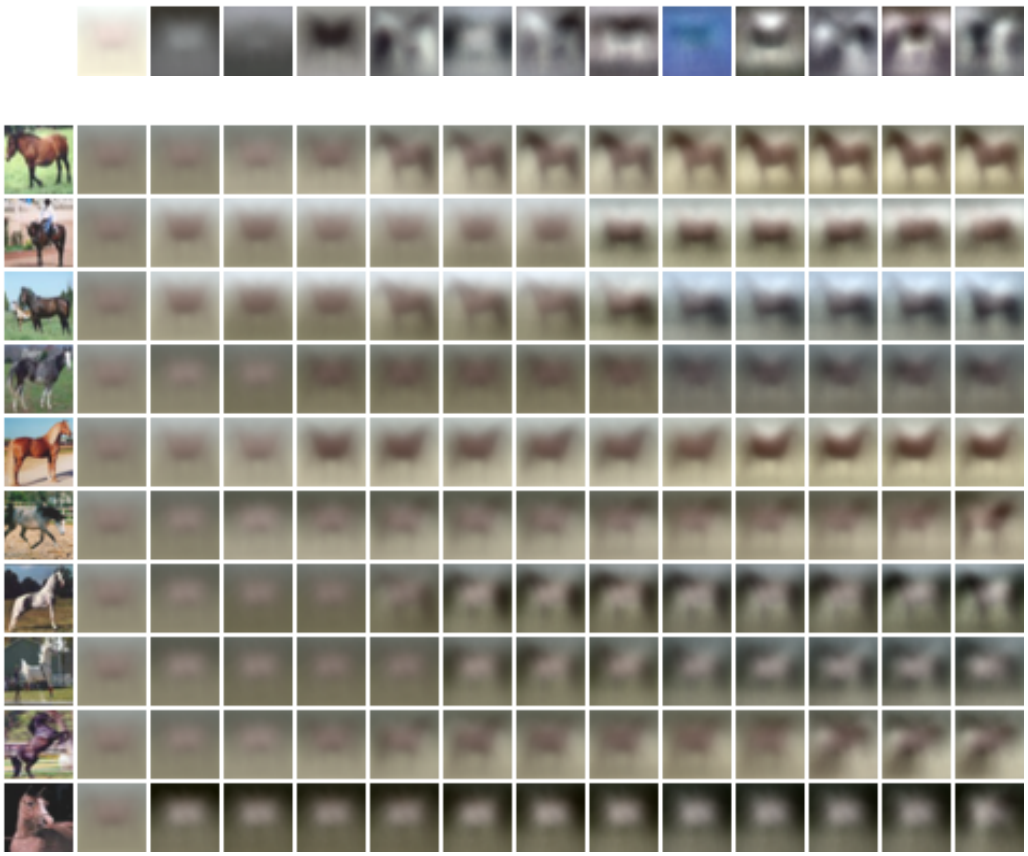
From which results that the PCA basis vectors $v_1, v_2, \ldots$ are the eigenvectors of the covariance matrix

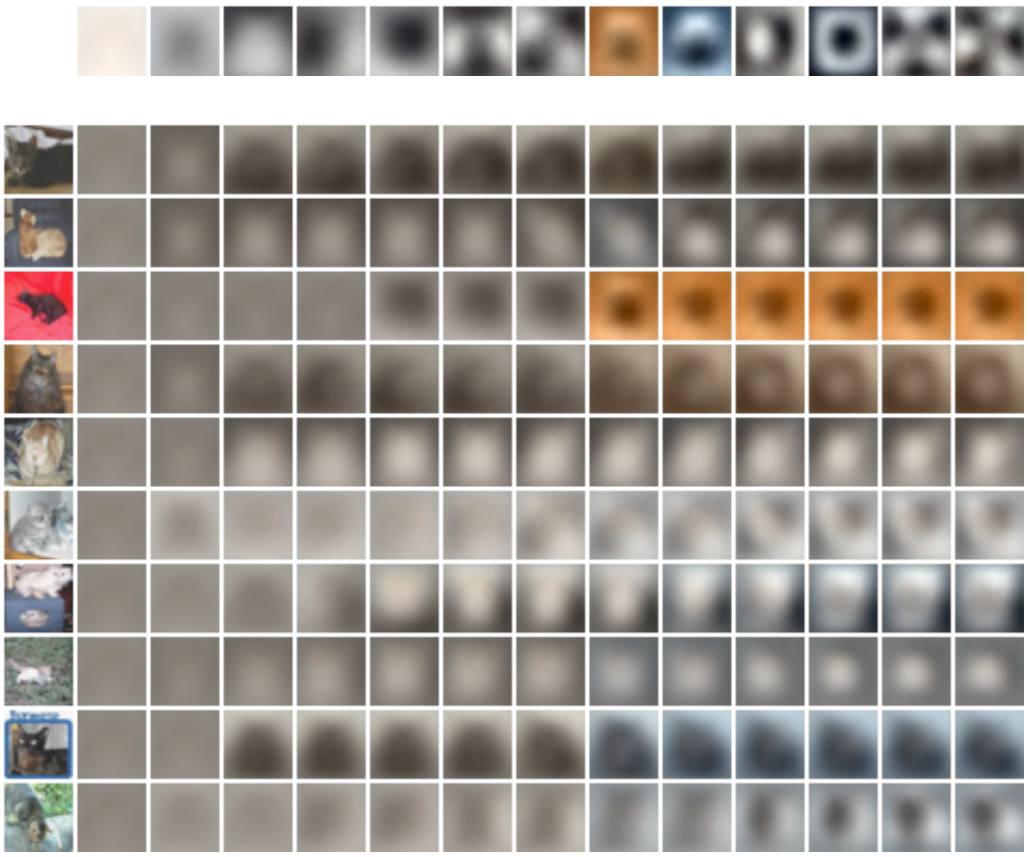
$$\Sigma = \begin{pmatrix} | & & | \\ x_1^{(0)} & \cdots & x_N^{(0)} \\ | & & | \end{pmatrix} \begin{pmatrix} — & x_1^{(0)} & — \\ & \vdots & \\ — & x_N^{(0)} & — \end{pmatrix}$$

ranked by decreasing eigenvalues.









Even crude embeddings capture something meaningful. Changes in pixel intensity as expected, but also deformations in the "indexing" space (*i.e.* the image plan).

However, translations and deformations damage the representation badly. "Composition" (*e.g.* object on background) is not handled at all.

These strengths and shortcomings provide an intuitive motivation for "deep neural networks", and the rest of this course.

We would like

- to use many encoding "of these sorts" for small local structures with limited variability,
- have different "channels" for different components,
- process at multiple scales.

Computationally, we would like to deal with large signals and large training sets, so we need to avoid super-linear cost in one or the other.

Practical session:

`https://fleuret.org/dlc/dlc-practical-2.pdf`

References

Y. Bengio and Y. Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. *Journal of Machine Learning Research (JMLR)*, 5:1089–1105, 2004.

S. Geman and E. Bienenstock. Neural networks and the bias/variance dilemma. *Neural Computation*, 4:1–58, 1992.

A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, Department of Computer Science, University of Toronto, 2009.