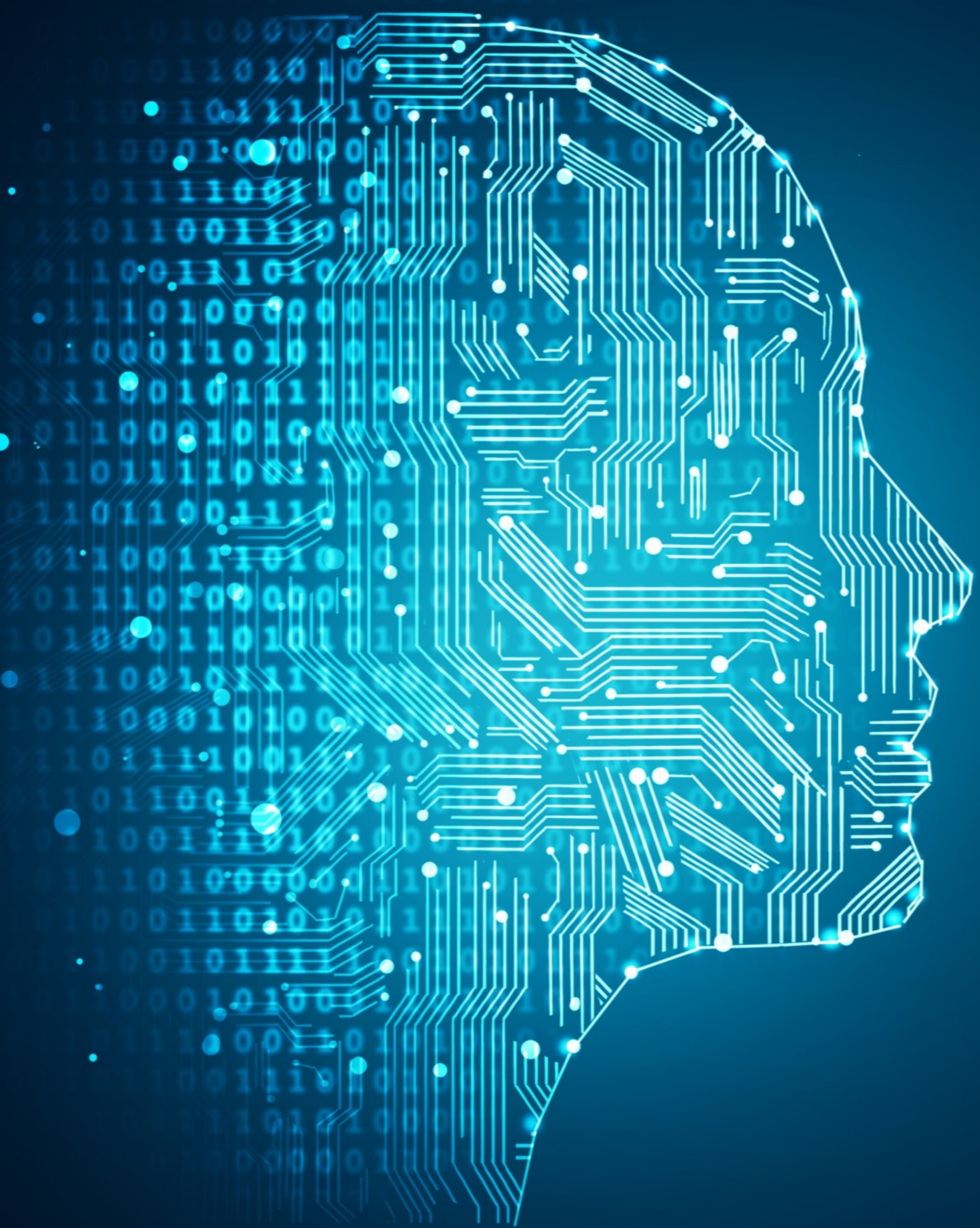


AI6103 Things Not Required for the Quiz

Boyang Li, Albert

School of Computer Science and Engineering

Nanyang Technological University

Linear Regression: An example

- y is selling price of house in $\$1000$ (in some location, over some period)
- regressor is

  $$x = (\text{house area, \# bedrooms})$$

  (house area in 1000 sq.ft.)
- regression model weight vector and offset are

  $$\beta = (148.73, -18.85), \qquad v = 54.40$$
- we'll see later how to guess β and v from sales data

House	x_1 (area)	x_2 (beds)	y (price)	$\hat{y}$ (prediction)
1	0.846	1	115.00	161.37
2	1.324	2	234.50	213.61
3	1.150	3	198.00	168.88
4	3.037	4	528.00	430.67
5	3.984	5	572.50	552.66

$$\begin{bmatrix} 0.846 & 1 \\ 1.324 & 2 \\ 1.150 & 3 \\ 3.037 & 4 \\ 3.984 & 5 \end{bmatrix} \begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix} = \begin{bmatrix} 155.00 \\ 243.50 \\ 198.00 \\ 528.00 \\ 527.50 \end{bmatrix}$$

No inverse for the 5×2 matrix. No exact solutions.

We can find the so-called "pseudo-inverse"

Python Programming

- Python is a dynamic-typed language.
  - No static type-checking
  - Flexible, succinct, though less efficient and error-prone
- Python has many numerical libraries
  - They call highly efficient C/C++ libraries behind the curtain
  - LAPACK, BLAS, etc.
- Great support for deep learning
  - PyTorch, Tensorflow, Jax, Peddle, etc.

Basics of a Programming Language

- Basic syntax
- Data types
- Control structure
- Exceptions
- Threads
- Performance

Basics of a Programming Language

- Basic syntax
- Data types
- ~~Control structure~~
- ~~Exceptions~~
- Threads
- Performance

Basic Syntax

- Indentation and colon mark code blocks
- Space-based indentations and tab-based are different!
- Can be super hard to debug!

```
sum = 0
for i in range(10):
    sum += i
print(sum)
```

<u>sum is printed only once!</u>

```
sum = 0
for i in range(10):
    sum += i + i**2 + \\
    i**3
print(sum)
```

Basic Syntax

- The return character marks end of line
- To continue the same line of code, use the line continuation character ‘\’
- # is the beginning of a comment

```
sum = 0
for i in range(10):
    sum += i + i**2 #no
continuation
    +i**3
print(sum) # 330
----------------------------------------
sum = 0
for i in range(10):
    sum += i + i**2 \
    +i**3
print(sum) # 2355
# good practice to end line on the
operator ‘+’
```

Data Type

- Integers have no limits on their value.

```
num = 17**320
#5541850949295918202405081540365965
79297285297717255852930128420561689
97165580793547643430275803715662642
27284283224328943216544902369965718
41142458684669230453829660776514954
68000637704311015847106327664574761
87127432980846740321915053107552729
32560408186056080555416180145335951
36336836482778399338370133645413199
50089537137346381263775423227192683
61430812049042445124742754151760840
8025625601
```

Data Type

- Integers have no limits on their value.
- Floating point numbers, however, do have limits

```
num = 1.8e308
# inf
```

- 64-bit max $\approx 1.8 \times 10^{308}$
- 32-bit max $\approx 3.4 \times 10^{38}$
- 16-bit max $= 65{,}504$

Danger of overflow and underflow during mixed precision training

Data Type

- Linked lists are first-class citizens

```
nl = list()
nl.append(3)
nl.append(“hello”)
nl.append(8.1)
#[1, 'hello', 0.33]

nl = []
nl = [3, 0.2]
```

Data Type

- Dictionaries are first-class citizens

```
dd = {1:0, ‘n’:0.3, “big”:
“better”}

print(dd)
#{1: 0, 'n': 0.3, 'big': 'good'}

print(dd[“big”])
#good
```

Data Type

- Objects are not encapsulated
- Python does not have the private keyword.
- Based on convention, anything that starts with two underscores are private.

```
class Robot(object):
   def __init__(self):
      self.a = 123
      self._b = 223
      self.__c = 323

obj = Robot()
print(obj.a)
print(obj._b)
print(obj.__c) # will not work
print(obj.__dict__['_Robot__c’])
# this works!
# does allow for hacks
```

Functions

- Python has some aspects of a functional language
  - You can assign a function to a variable

```
def addone(x):
  return x+1

def addtwo(x):
  return x+2

f = addone
print(f(1)) # result is 2
f = addtwo
print(f(1)) # result is 3
```

Functions

- Python has some aspects of a functional language
  - You can assign a function to a variable
  - Python provides higher-order functions like Lisp

```
import functools
def add_one_more(x, y):
  return x+y+1

f = 
functools.partial(add_one_more,1)
# this creates a partial function 
whose first argument is known but 
the second is not.

print(f(2)) # result is 1+2+1=4
```

Functions

- Python has some aspects of a functional language
  - You can assign a function to a variable
  - Python provides higher-order functions like Lisp
    - `map()`, `filter()`, `reduce()`, etc.

```
def add_one(x):
  return x+1

l1 = [1, 2, 3, 4]
l2 = list(map(add_one, l1))
print(l2) # [2, 3, 4, 5]

l3 = [add_one(x) for x in l1]
print(l3) # [2, 3, 4, 5]

# the two results are the same but
many prefer the second style
```

Multi-threading

- You can create multiple threads in Python but they don’t work as you expect.
- The Global Interpreter Lock (GIL) ensures that only one thread is executing at a time.
- It is created as a simple fix for thread safety.
- Multi-threading leads to well-known problems of race conditions and dead locks.
- Python chose a simple solution: get rid of multi-threading
- Today, there is just too much path dependency to do anything about it.

Multi-processing

- We can circumvent the GIL by using the multiprocessing package of Python.
- A process has more overhead than a thread.
- All multi-threading problems, like race conditions and deadlocks, still need to be handled by the programmer.
- Those problems are inherent to any memory-sharing programs running in parallel.
- Don't over-simplify problems.

Performance

- Python is an interpreted, dynamically typed language.
- You can do a lot of unexpected things in Python
- Automatic optimization is hard because the program cannot predict what you will do.
- The result in slow execution
- To improve performance, avoid things like for loops and lists.
- Use packages like numpy, which calls fast libraries written in C.

Probability Theory

The Need for Probability

- **Laziness**: It is too much work to list the complete set of exceptional cases like exceptions to a rule or extraordinary categories of images.
- **Theoretical ignorance**: We do not have a complete theory for most scientific and engineering domains. Thus, we cannot make predictions that are 100% accurate.
- **Practical ignorance**: Even if we know all the rules, we might be uncertain about a particular patient because we may not have enough resources (money, time, manpower, etc.) to run all tests.
- **Artificial ignorance**: In the case of games, the game rules prevent us from learning all facts such as the sequence of cards in a deck.

Advanced Topics That are not Covered

- **Probabilistic Graphic Models**
  - Directed graphs
  - Undirected graphs
  - Factor graphs
- The graph representations encode independence relations between variables
- Central Question: How to find the distribution of unknown variables given the observed or known variables?
- These models were very popular before the era of deep learning

The Terminology "Moment"

- The first moment

$$E_{P(x)}[x] = \int x \, P(x) dx$$

- The second moment

$$E_{P(x)}[x^2] = \int x^2 \, P(x) dx$$

- The centered second moment

$$Var(x) = E[(x - E[x])^2] = E[x^2] - (E[x])^2$$

The derivation of bias and variance for $\hat{\mu}_{\mathrm{MLE}}$

- $\hat{\mu}_{\mathrm{MLE}} = \frac{1}{N}\sum_{i=1}^{N} X^{(i)}$
- $E[\hat{\mu}_{\mathrm{MLE}}] = E\left[\frac{1}{N}\sum_{i=1}^{N} X^{(i)}\right] = \frac{1}{N}\sum_{i=1}^{N} E[X^{(i)}] = \frac{1}{N}\sum_{i=1}^{N} \boxed{\mu}$ mean of the real distribution of X

The derivation of bias and variance for $\hat{\mu}_{\mathrm{MLE}}$

$$Var(\hat{\mu}_{\mathrm{MLE}}) = Var\left(\frac{1}{N}\sum_{i=1}^{N} X^{(i)}\right) = \frac{1}{N^2} Var\left(\sum_{i=1}^{N} X^{(i)}\right)$$

$$= \frac{1}{N^2}\sum_{i=1}^{N} Var\left(X^{(i)}\right) + \boxed{\sum_{i=1}^{N}\sum_{j=1}^{N} Cov(X^{(i)}, X^{(j)})}$$

Independent variables have zero covariance

$$= \frac{1}{N^2} N\sigma^2 = \frac{\sigma^2}{N}$$

Advanced Topics That are Not Covered

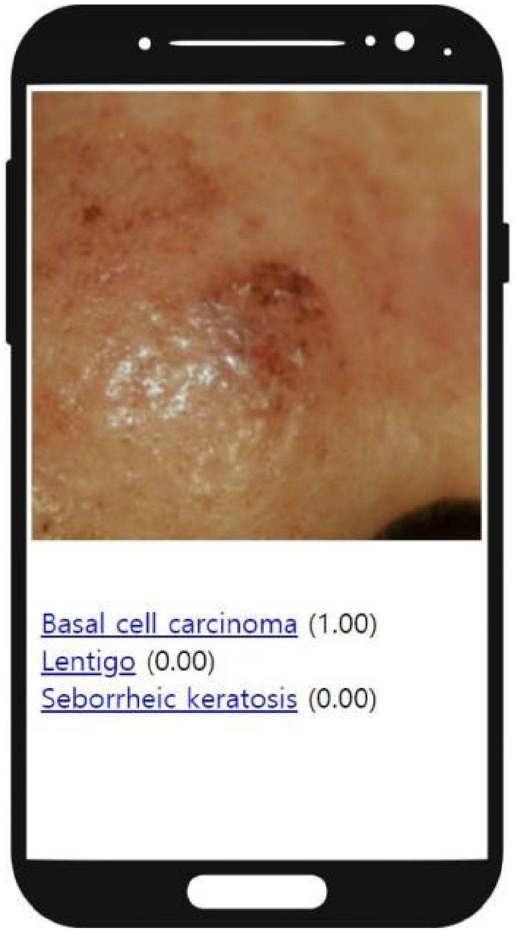
- Other Commonly Used Distributions
  - Poisson Distribution
  - Exponential Distribution
  - Beta Distribution
  - Dirichelet Distribution
  - Fat-tailed Distributions, and many more
- Mixture of Distributions
  - Gaussian Mixtures, often used as a clustering model
- Bayesian Estimates
  - Incorporate "prior" into the estimates
- Estimators and Variance reduction
  - Variance is often a problem (for example, in stochastic gradient descent)
  - Ways to reduce the variance of an estimator

Information Theory

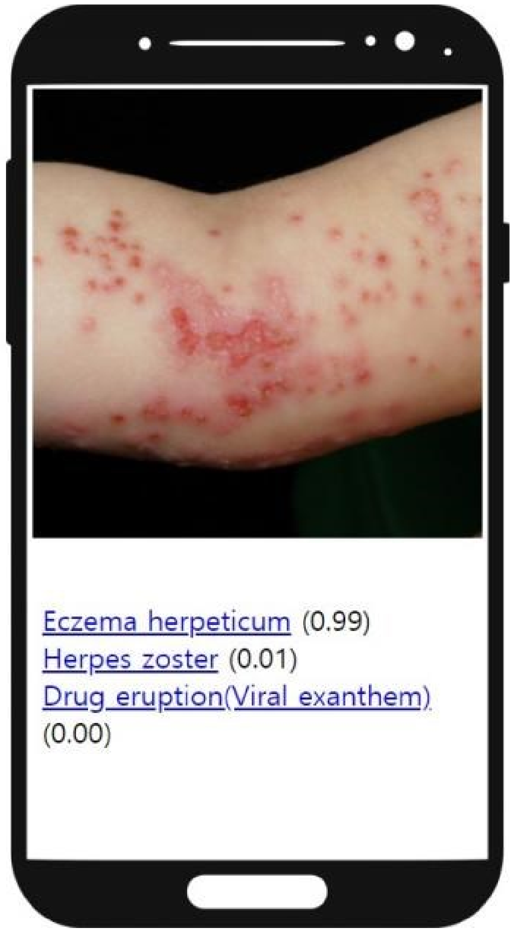
Entropy: Relation to Event Encoding

- If we observe 26 letters with equal probability, we can use $\log_2 26 = -\log_2 \frac{1}{26}$ bits to encode each character.
- No fractional bits, so $\lceil \log_2 26 \rceil = 5$
- A = 00000, B = 00001, C=00010, D=00011, etc.
- However, if one letter is more common than others, we can design the encoding such that we use fewer bits for more frequent letters.
- A = 001, B=0001, C=10100, D=10011, etc.
- BAD=00010010011 (12 bits)
- Using fewer bits in expectation!

Classification for Skin Problems



Basal cell carcinoma (1.00)
Lentigo (0.00)
Seborrheic keratosis (0.00)



Eczema herpeticum (0.99)
Herpes zoster (0.01)
Drug eruption(Viral exanthem) (0.00)

Maximum A Posteriori Estimation for Gaussian

- Data: $y^{(1)}, \dots, y^{(N)}$
- In MAP, we look for

$$\theta^* = P(\theta|\boldsymbol{y}, \alpha) = \frac{P(\boldsymbol{y}|\theta,\alpha)P(\theta|\alpha)}{p(\boldsymbol{y})}$$

Likelihood distribution $P(\boldsymbol{y}|\theta,\alpha)$
Prior distribution $P(\theta|\alpha)$
Normalization constant $p(\boldsymbol{y})$

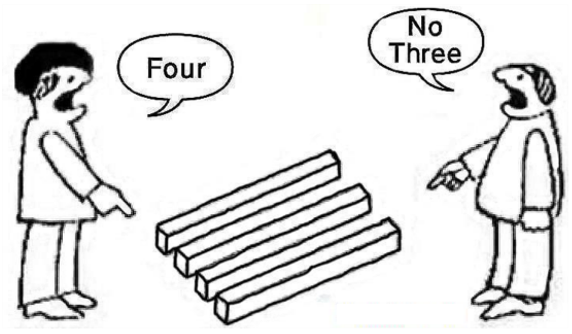
- The joint probability is

$$P(\boldsymbol{y}, \boldsymbol{\beta}, \boldsymbol{\mu}', \sigma') = \prod_{i=1}^{N} P\left(y^{(i)} \middle| f_{\boldsymbol{\beta}}(\boldsymbol{x}), \sigma\right) P(\boldsymbol{\beta}|\boldsymbol{\mu}', \sigma') = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} \exp - \frac{\left(y^{(i)} - f_{\boldsymbol{\beta}}(\boldsymbol{x})\right)^2}{\sigma^2} \frac{1}{\sqrt{2\pi}\sigma'} \exp - \frac{(\boldsymbol{\beta} - \boldsymbol{\mu}')^2}{\sigma'^2}$$

- $p(\boldsymbol{y})$ does not involve μ, so it can be ignored in the optimization

$$\boldsymbol{\beta}^* = \underset{\boldsymbol{\beta}}{\operatorname{argmin}}\, P(\boldsymbol{y}, \boldsymbol{\beta}, \boldsymbol{\mu}', \sigma')$$

Plugging in …

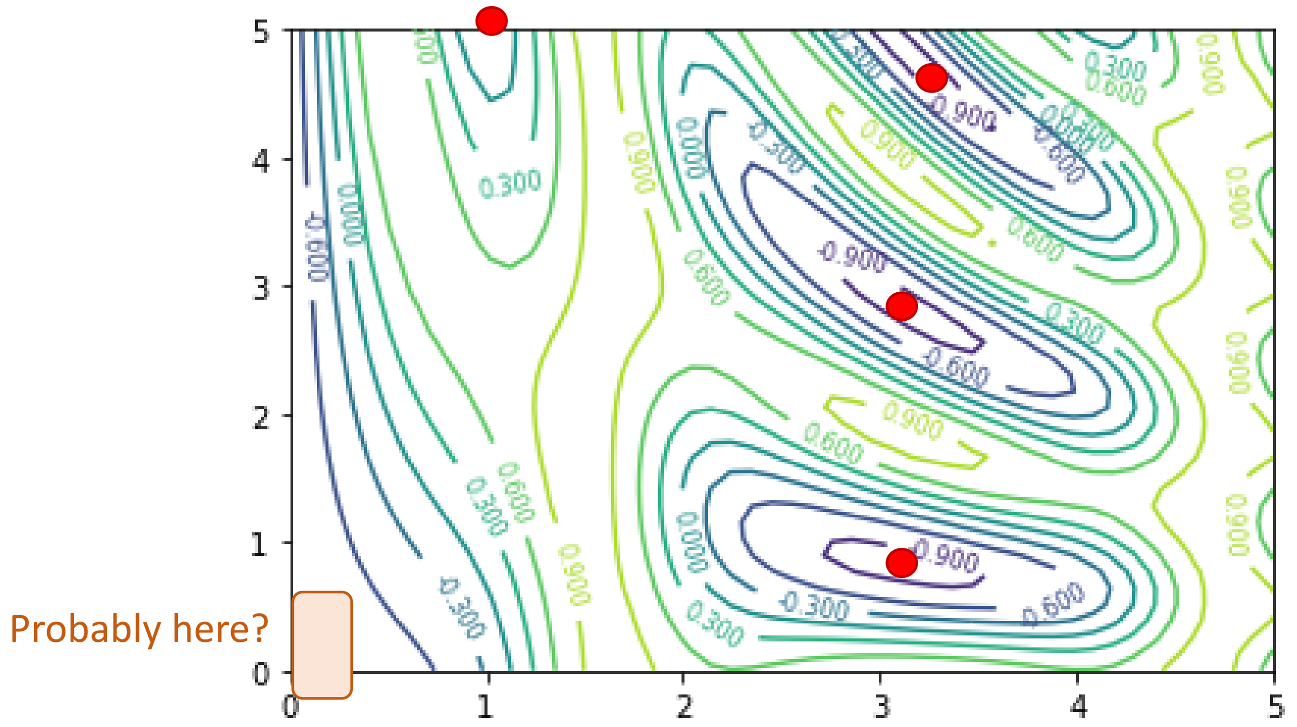


- $\mu' = 0, \sigma'^2 = \frac{1}{\lambda}$

$$\boldsymbol{\beta}^* = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \sum_{i=1}^{N} \left(y^{(i)} - f_{\boldsymbol{\beta}}\left(\boldsymbol{x}^{(i)}\right)\right)^2 + \lambda \boldsymbol{\beta}^2$$

Ridge regression can be understood as Bayesian maximum a posteriori (MAP) estimation with a Gaussian prior $\mathcal{N}(0, \frac{1}{\lambda})$ for the model parameters $\boldsymbol{\beta}$**.**

All local minima on the diagram



Vanishing Gradient: Math Derivation

We can understand vanishing gradient using singular value decomposition. Slides 17 to 24 are not required for AI6103.

Singular Value Decomposition

- It turns out that every matrix can be written as the combination of such operations.
- Any $m \times n$ real matrix M can be written as

$$M = U\Sigma V^\top$$

- where U is an $m \times m$ orthonormal matrix, V is an $n \times n$ orthonormal matrices, and Σ is a $m \times n$ rectangle diagonal matrix with nonnegative values.
- The values on the diagonal of Σ are singular values
- $$\Sigma = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & & \sigma_n \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & & 0 \end{bmatrix}$$

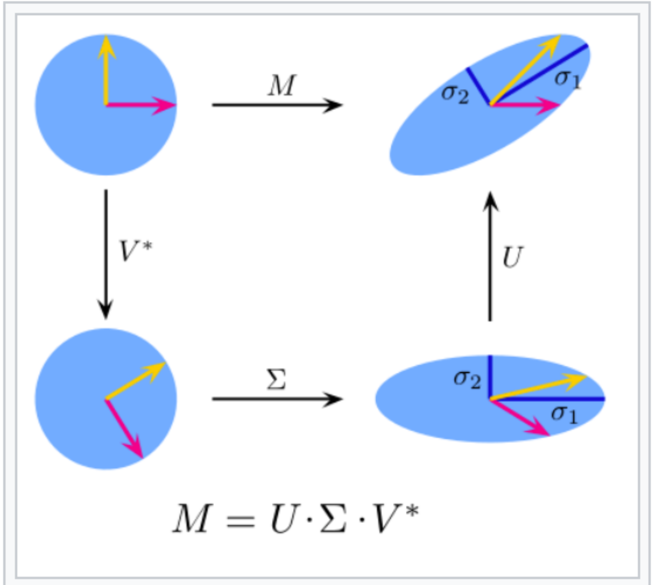
Illustration of the singular value decomposition **UΣV*** of a real 2×2 matrix **M**.

Top: The action of **M**, indicated by its effect on the unit disc D and the two canonical unit vectors e_1 and e_2.

Left: The action of **V***, a rotation, on D, e_1, and e_2.

Bottom: The action of **Σ**, a scaling by the singular values σ_1 horizontally and σ_2 vertically.

Right: The action of **U**, another rotation.



The order of singular values

- Any $m \times n$ real matrix M can be written as

$$M = U\Sigma V^\top = [u_1\ u_2\ u_3\ \cdots u_n] \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & & \sigma_n \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & & 0 \end{bmatrix} \begin{bmatrix} v_1^\top \\ v_2^\top \\ v_3^\top \\ \vdots \\ v_m^\top \end{bmatrix}$$

$$= \sum_{i=1}^{\min(m,n)} \sigma_i u_i v_i^\top$$

This sum does not care about the ordering of σ

- where u_i are the columns of U and v_i are the columns of V.
- We can freely switch the positions of singular values, as long as we keep (σ_i, u_i, v_i) together
- Typically, we sort them in either descending or ascending order

SVD for Low-rank Approximation

- Any $m \times n$ real matrix M can be written as

$$M = U\Sigma V^{\top} = [u_1\, u_2\, u_3\, \cdots u_n] \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & & \sigma_n \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & & 0 \end{bmatrix} \begin{bmatrix} v_1^{\top} \\ v_2^{\top} \\ v_3^{\top} \\ \vdots \\ v_m^{\top} \end{bmatrix}$$

$$= \sum_{i=1}^{\min(m,n)} \sigma_i u_i v_i^{\top}$$

- where u_i are the columns of U and v_i are the columns of V.
- Since u_i and v_i are of unit length. If σ_i is really small, then $\sigma_i u_i v_i^{\top}$ does not matter much.
- Thus, we can approximate M using K largest σ_i and the corresponding sum.
- Reducing space complexity from O(MN) to O(K(M+N))

SVD for Low-rank Approximation

- Any $m \times n$ real matrix M can be written as

$$M = U\Sigma V^\top = \sum_{i=1}^{\min(m,n)} \sigma_i u_i v_i^\top$$

- Thus, we can approximate M using K largest σ_i and the corresponding sum.
- Reducing space complexity from O(MN) to O(K(M+N))
- For the image on the right, # pixels = 59,000
- Using 14 singular values, # values stored = 6,804
- More examples: http://timbaumann.info/svd-image-compression-demo/

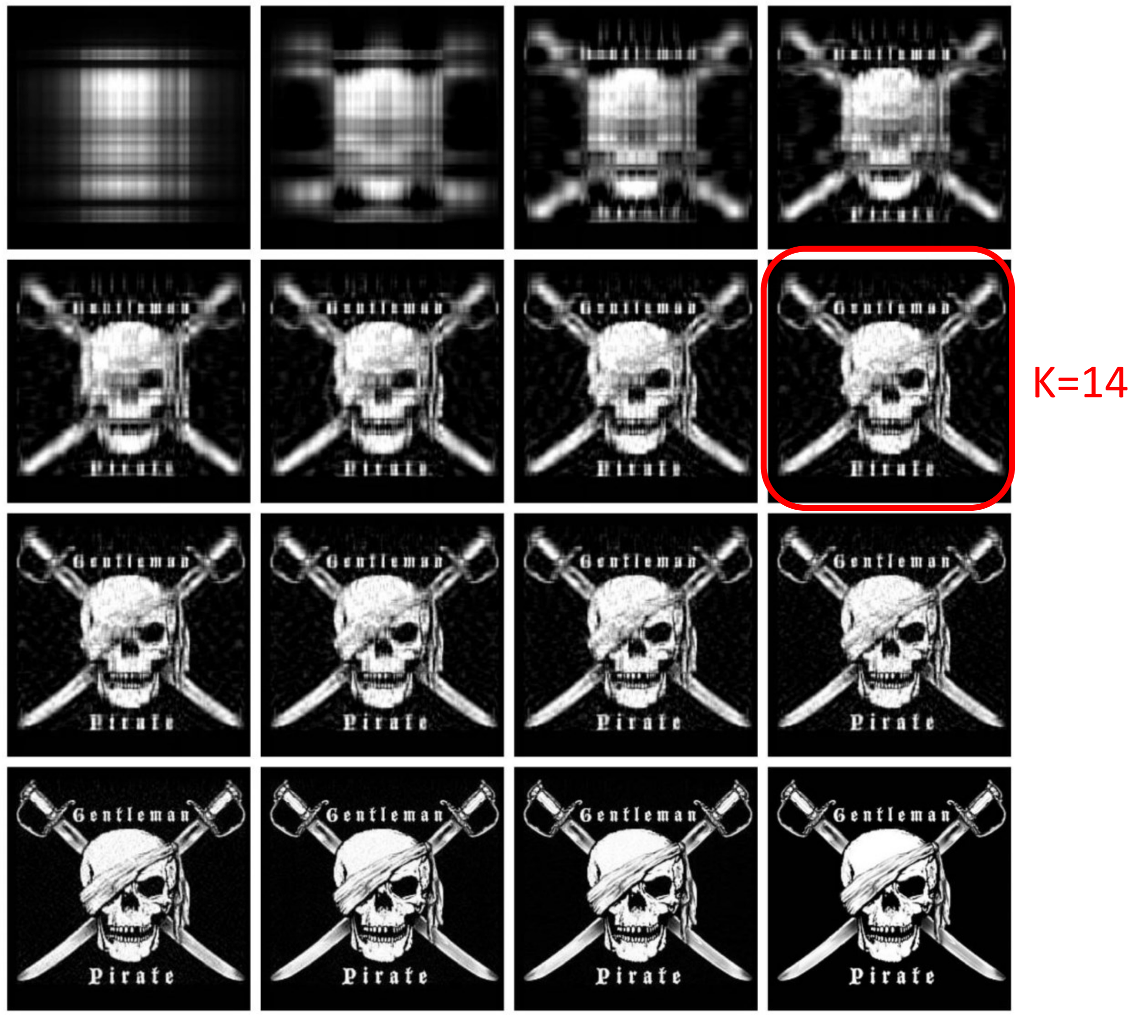


Figure 3.1: Image size 250x236 – modes used {{1,2,4,6},{8,10,12,14},{16,18,20,25},{50,75,100,original image}}

Brady Mathews. Image Compression using Singular Value Decomposition (SVD) 2014

Vanishing Gradient

- The gradient is computed as matrix multiplications. How does that affect gradients in deep networks?
- Compare $\|A\boldsymbol{x}\|$ and $\|\boldsymbol{x}\|$
- By SVD, $A = U\Sigma V^\top$ where U and V are orthogonal matrices. The singular values $\sigma_1 > \cdots > \sigma_n > 0$.
- $\|V^\top\boldsymbol{x}\| = \|\boldsymbol{x}\|$ because rotation, permutation, and reflection do not change vector length.

- Let $\|\boldsymbol{x}\| = c$. Assuming A is full rank, we can write

$$\boldsymbol{x} = c(a_1\boldsymbol{v}_1 + a_2\boldsymbol{v}_2 + \cdots + a_n\boldsymbol{v}_n)$$
$$\boldsymbol{x}^\top\boldsymbol{x} = c^2(a_1^2 + a_2^2 + \cdots + a_n^2)$$

- We know that $\boldsymbol{x}^\top\boldsymbol{x} = c^2$
- So $a_1^2 + a_2^2 + \cdots + a_n^2 = 1$

$$\Sigma V^\top\boldsymbol{x} = \Sigma\begin{bmatrix}\boldsymbol{v}_1^\top\boldsymbol{x}\\ \boldsymbol{v}_2^\top\boldsymbol{x}\\ \vdots\\ \boldsymbol{v}_n^\top\boldsymbol{x}\end{bmatrix} = c\Sigma\begin{bmatrix}a_1\\ a_2\\ \vdots\\ a_n\end{bmatrix}$$

Vanishing Gradient

- We want to compare $\|A\boldsymbol{x}\|$ and $\|\boldsymbol{x}\|$
- By SVD, $A = U\Sigma V^\top$ where U and V are orthogonal matrices. The singular values $\sigma_1 > \cdots > \sigma_n > 0$.
- Let $\|\boldsymbol{x}\| = c$. Assuming A is full rank, we can write

$$\boldsymbol{x} = c(a_1\boldsymbol{v}_1 + a_2\boldsymbol{v}_2 + \cdots + a_n\boldsymbol{v}_n)$$
$$\boldsymbol{x}^\top\boldsymbol{x} = c^2(a_1^2 + a_2^2 + \cdots + a_n^2)$$

- We know that $\boldsymbol{x}^\top\boldsymbol{x} = c^2$
- So $a_1^2 + a_2^2 + \cdots + a_n^2 = 1$

$$\Sigma V^\top\boldsymbol{x} = \Sigma\begin{bmatrix}\boldsymbol{v}_1^\top\boldsymbol{x}\\ \boldsymbol{v}_2^\top\boldsymbol{x}\\ \vdots\\ \boldsymbol{v}_n^\top\boldsymbol{x}\end{bmatrix} = c\Sigma\begin{bmatrix}a_1\\ a_2\\ \vdots\\ a_n\end{bmatrix}$$

$$\Sigma V^\top\boldsymbol{x} = c\begin{bmatrix}a_1\sigma_1\\ a_2\sigma_2\\ \vdots\\ a_n\sigma_n\end{bmatrix}$$

$$\|\Sigma V^\top\boldsymbol{x}\|^2 = c^2(a_1^2\sigma_1^2 + a_2^2\sigma_2^2 + \cdots a_n^2\sigma_n^2)$$

- Therefore, $\|\Sigma V^\top\boldsymbol{x}\| \leq c\sigma_1$
- Equality holds when $a_1 = 1, a_2 = \cdots = a_n = 0$

Vanishing Gradient

- We want to compare $\|A\boldsymbol{x}\|$ and $\|\boldsymbol{x}\|$
- By SVD, $A = U\Sigma V^\top$ where U and V are orthogonal matrices. The singular values $\sigma_1 > \cdots > \sigma_n > 0$.
- $\|A\boldsymbol{x}\| \leq \sigma_1 \|\boldsymbol{x}\|$
- Equal sign is taken when $\boldsymbol{x} = \boldsymbol{v}_1 \|\boldsymbol{x}\|$

- What happens when multiple matrices all have maximum singular values less than 1?

$$A_L \ldots A_3 A_2 A_1 \boldsymbol{x}$$

- The magnitude of the product will diminish as L increases.

Vanishing Gradient

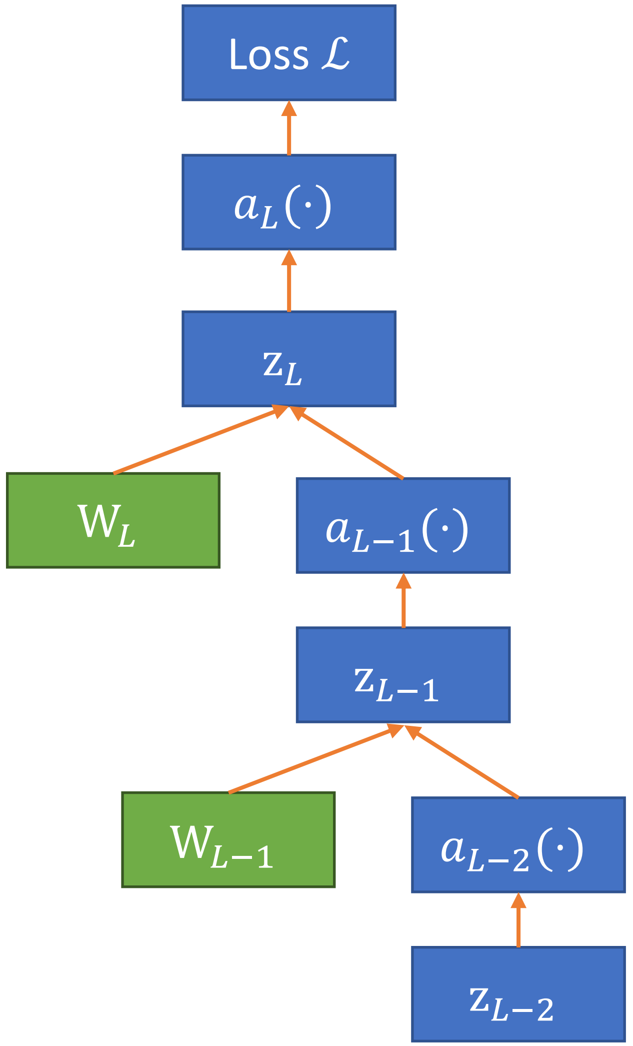
- What happens when multiple matrices all have maximum singular values less than 1?

$$A_L \dots A_3 A_2 A_1 \boldsymbol{x}$$

- The magnitude of the product will diminish as L increases.

$$\frac{\partial \mathcal{L}}{\partial W_{L-1}} = \frac{\partial \mathcal{L}}{\partial a_L(z_L)} \frac{\partial a_L(z_L)}{\partial z_L} \frac{\partial z_L}{\partial a_{L-1}(z_{L-1})} \frac{\partial a_{L-1}(z_{L-1})}{\partial z_{L-1}} \frac{\partial z_{L-1}}{\partial W_{L-1}}$$

- $\frac{\partial a_L(z_L)}{\partial z_L}$ is a diagonal matrix with component-wise derivative on the diagonal. For both sigmoid and tanh functions, the maximum derivative values < 1.
- Thus, the gradient diminishes
- Impossible to train very deep networks.



PyTorch AutoGrad

- PyTorch records the operations performed on Tensors as the program executes.
- This is known as Dynamic Graph, which is a directed acyclic graph.
- At every training iteration, the graph is built from scratch.
- This allows arbitrary Python control flow and extreme flexibility.
- Contrast this with the static graph approach, which has to build the graph before any input can be fed to it.

PyTorch AutoGrad

- PyTorch must keep every tensor with a reference in memory, even if you never use it afterwards.
- A drawback of the dynamic graph approach
- In static graphs, it is easy to see what tensors are no longer needed.

```
A = func_a() # computes A
B = func_b() # computes B
C = torch.mm(A, B)
# A and B are kept in memory even
if you never use them again.
```

PyTorch AutoGrad

- In order to save precious GPU memory, minimize named references in your code.
- The `del` statement tells Python that a variable or memory reference is no longer needed.
- When there are no references pointing to a memory chunk, it can be recycled.

```
A = func_a() # computes A
B = func_b() # computes B
C = torch.dot(A, B)
# delete A and B
del A, B
# another example
A = func_a()
B = A
del A
# the memory chuck is still being
referenced by B
```

PyTorch AutoGrad

- In order to save precious GPU memory, minimize named references in your code.
- PyTorch will keeps track of variable updates so that gradients can be computed properly.
- A lot of things happening behind the curtain, which may be controlled by the user.

```
A = func_a() # computes A
A = func_b() + A # updates A
A = torch.mm(A, B)
# updates A again
# this pattern avoids creating
multiple references altogether
```

Freeze Parameters

- Set the requires_grad attribute to False will prevent a tensor from being updated with SGD.
- Local control of autograd.

```
A = Tensor.empty([2, 2])
A.requires_grad_(False)
```

`no-grad` Mode

- In the *`torch.no_grad()`* context, no autograd will be performed.
- The result of every computation will have *`requires_grad=False`*, even when the inputs have *`requires_grad=True`*.
- Accelerates execution

```
x = torch.tensor([1], requires_grad=True)
with torch.no_grad():
    y = x * 2
print(y.requires_grad) # False
```

`inference` Mode

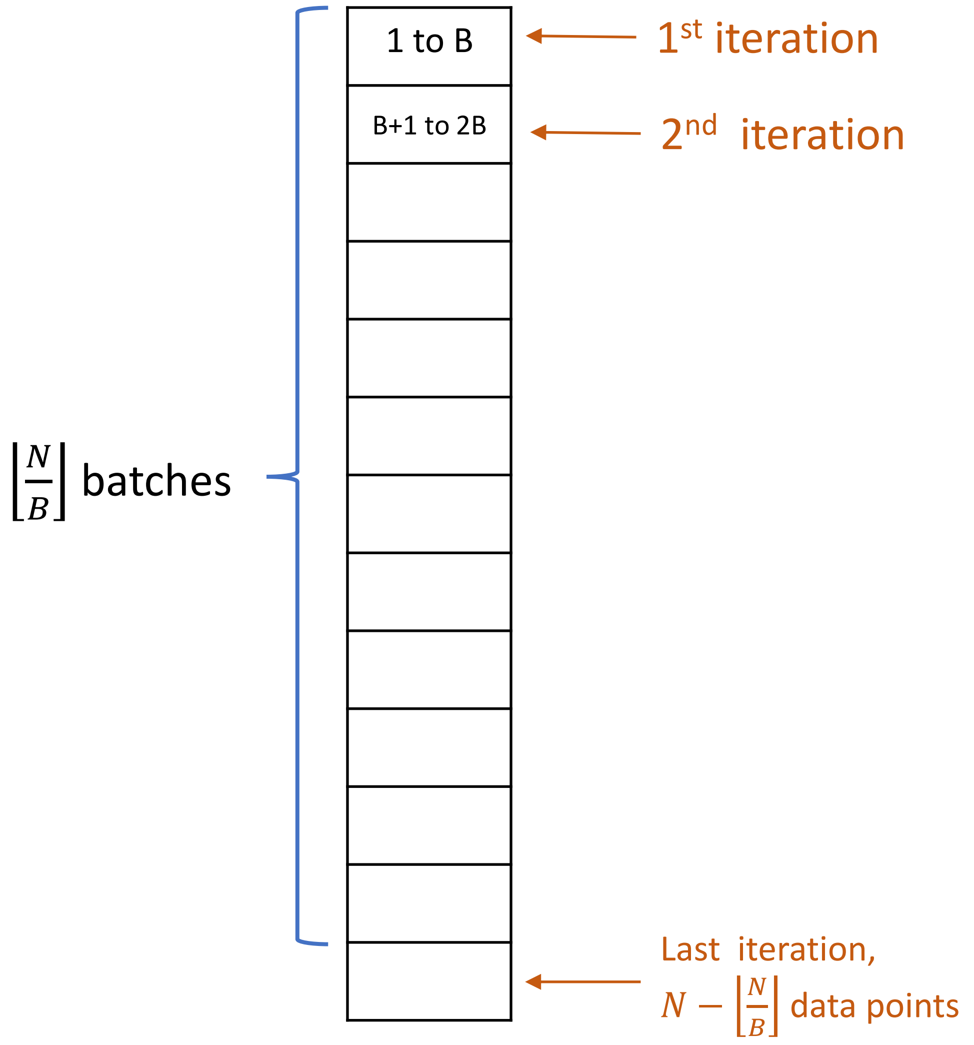
- The *`torch.inference_mode()`* context provides even more aggressive removal of autograd
- The result of every computation will have *`requires_grad=False`*, even when the inputs have *`requires_grad=True`*.
- These tensors cannot be used in the autograd mode even if it is enabled later on.

```
x = torch.tensor([1], requires_grad=True)
with torch.inference_mode():
    y = x * 2
print(y.requires_grad) # False
```

Implementation

1. Shuffle all training data into a random permutation.
2. Start from the beginning of the shuffled sequence.
3. In every iteration, take the next B data points, which form a mini-batch, and perform training on the mini-batch.
4. When the sequence is exhausted, go back to Step 1. This is one epoch of training.

Random shuffling is necessary for unbiased gradient estimates.


1 to B
B+1 to 2B
1st iteration
2nd iteration
$\left\lfloor \frac{N}{B} \right\rfloor$ batches
Last iteration, $N - \left\lfloor \frac{N}{B} \right\rfloor$ data points

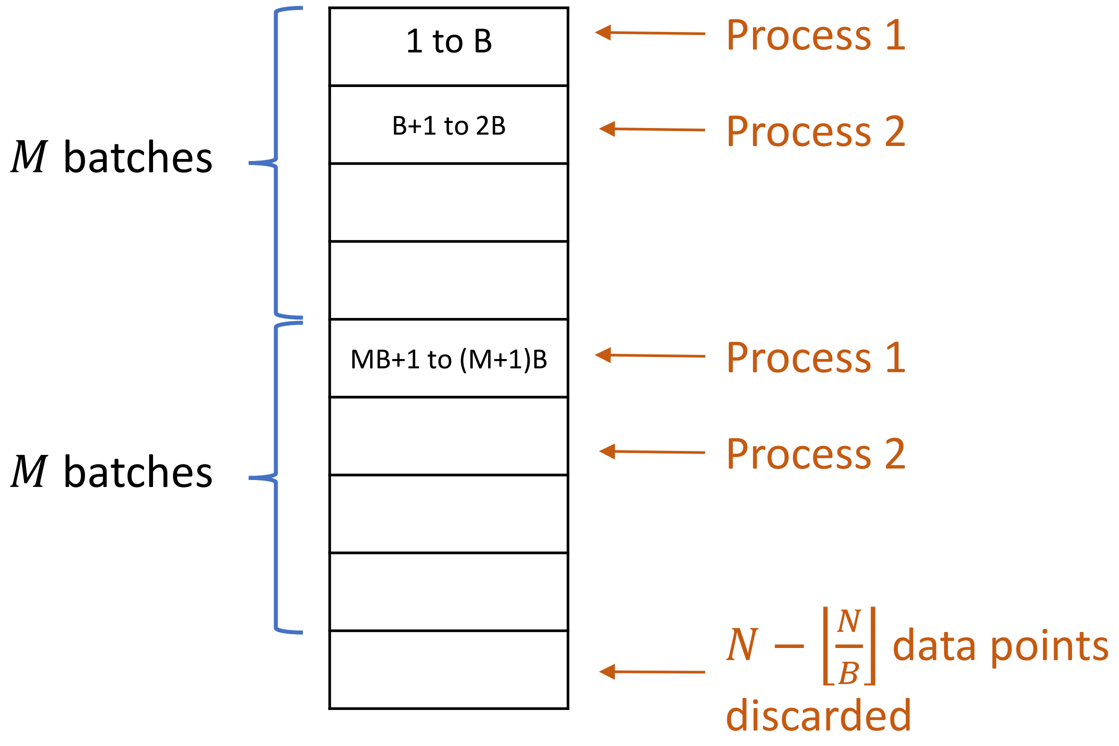
The last batch with less than B data points may be discarded if small batches cause problems (e.g., for BatchNorm)

Pseudo-random Number Generator (PNG)

- Pseudo-random number generators are not really "random".
  - What is really random? A philosophical discussion that we will not get into.
- It is a completely deterministic function of the random seed.
  - Always generates the same sequence if the seed is the same.
  - A.k.a deterministic random bit generator
- However, the sequence of numbers satisfy certain statistical properties that they can be considered random if the seed is not known.
  - Do not change seeds if you want good randomness. Use the sequence.
- If we give the same seed to two generators w/ the same algorithm, they will create the same sequence.
  - We can align multiple processes or multiple training sessions.

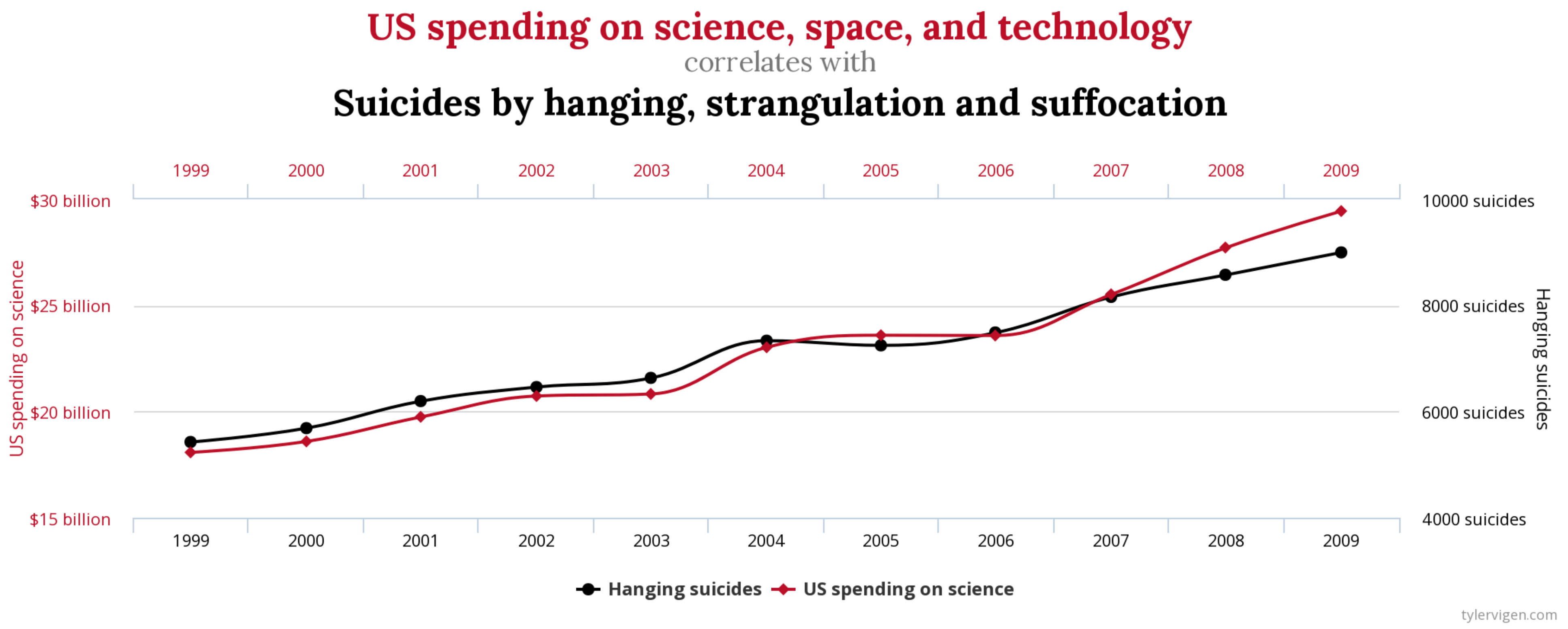
Synchronous Training on Multiple GPUs / Nodes

- Multiple processes (total number = M) use PNGs sharing the same seed when shuffling the dataset.
- Based on its process id, a process uses one batch of size B every M batches.
- The gradients from M processes are averaged and the model is updated once (synchronization).
- Equivalent to a global batch size of MB



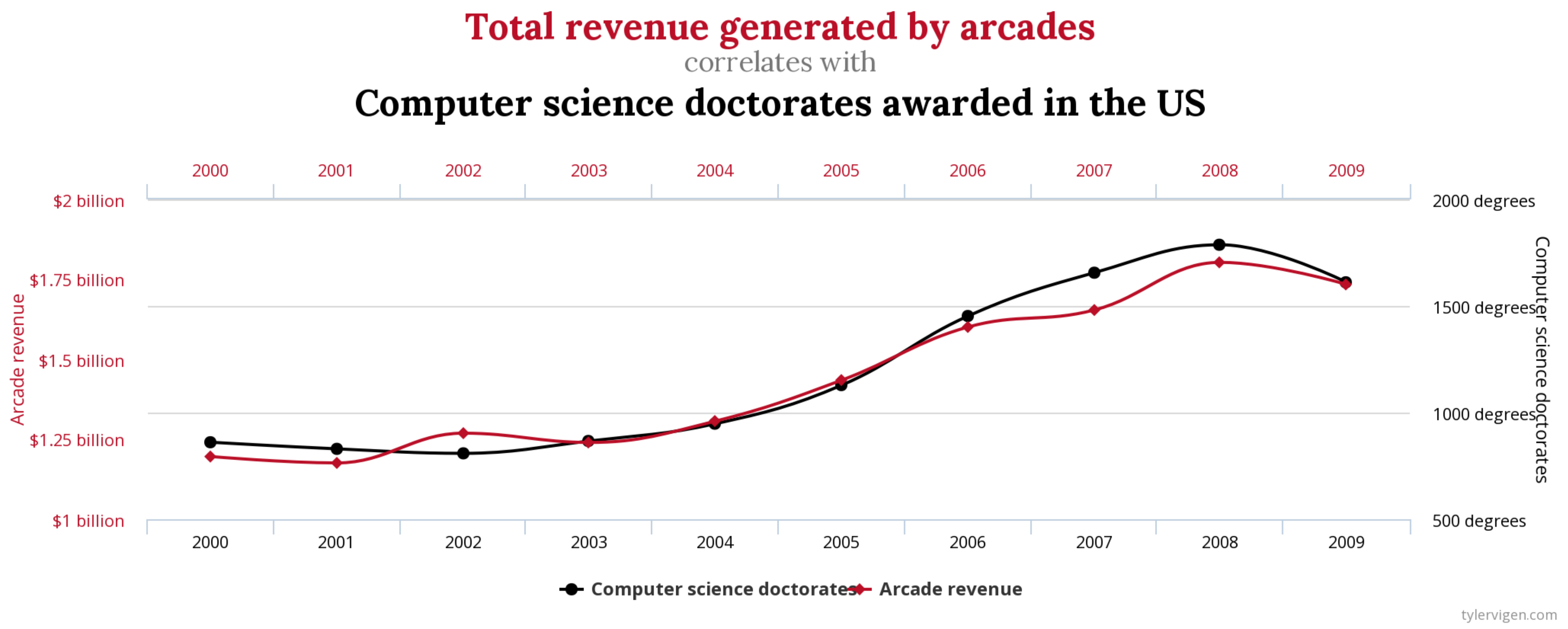
Representation Learning and Invariance

Bad Feature (correlation 99.79%)



Guaranteed: low training error, high test error on future data

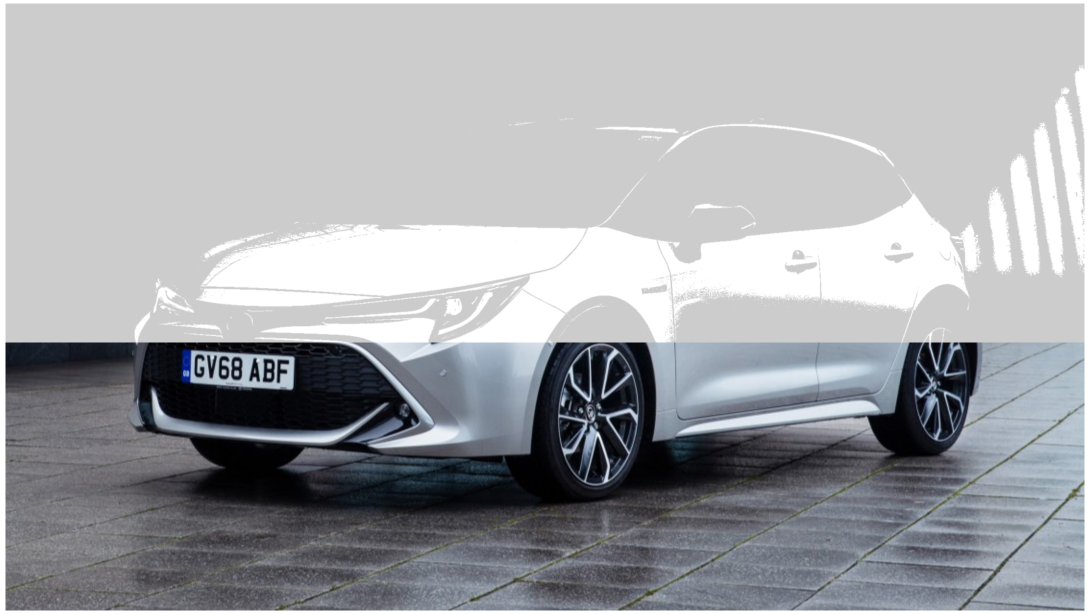
Bad Feature (correlation 98.51%)

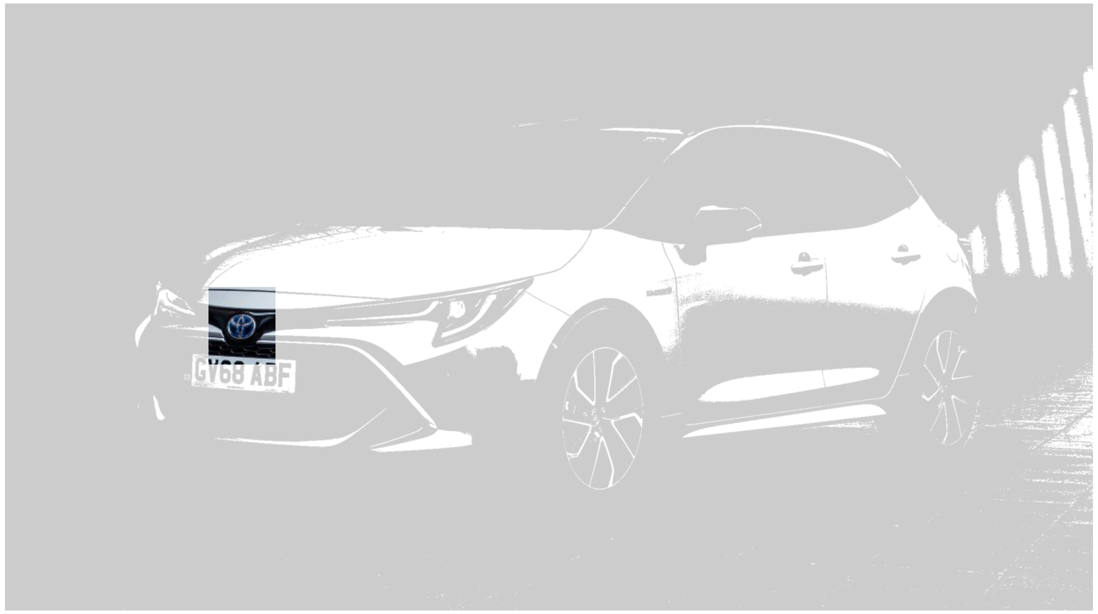


Guaranteed: low training error, high test error on future data

Feature Extraction: Which is more robust?

- What is the make of the car?

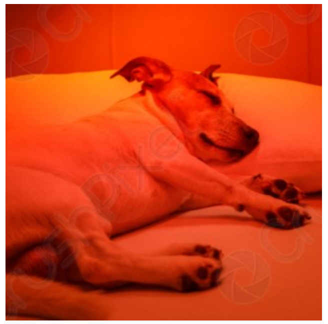




If the dataset comes from a single year, these features can be very effective, but they will stop working if Toyota ever changes its design.

Same object, different pixel values

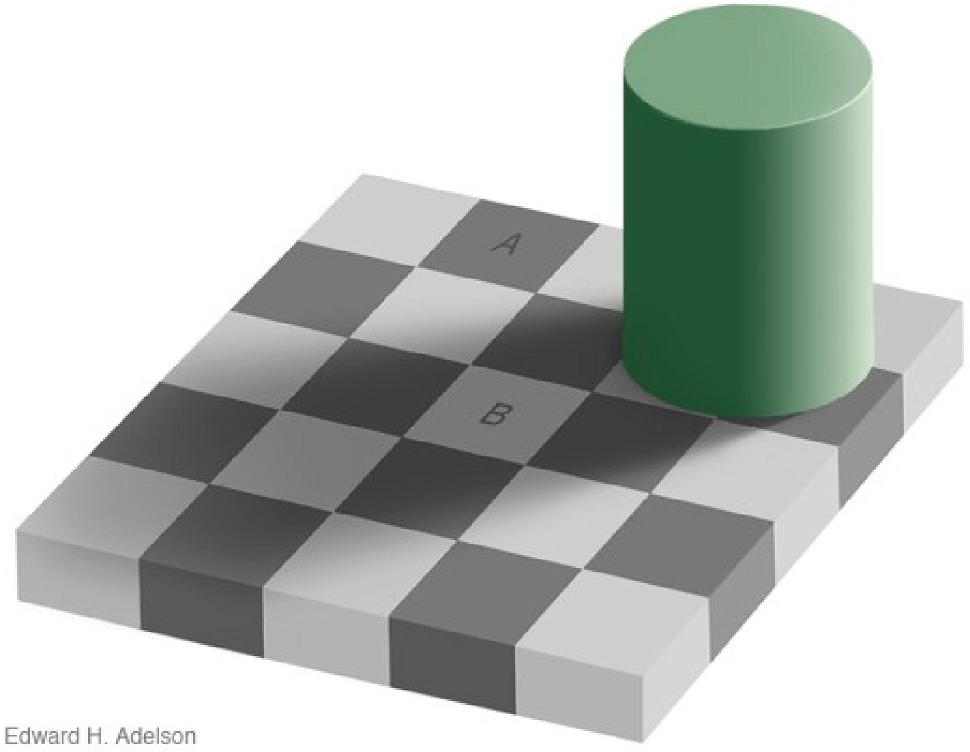
Black and white,
Low light

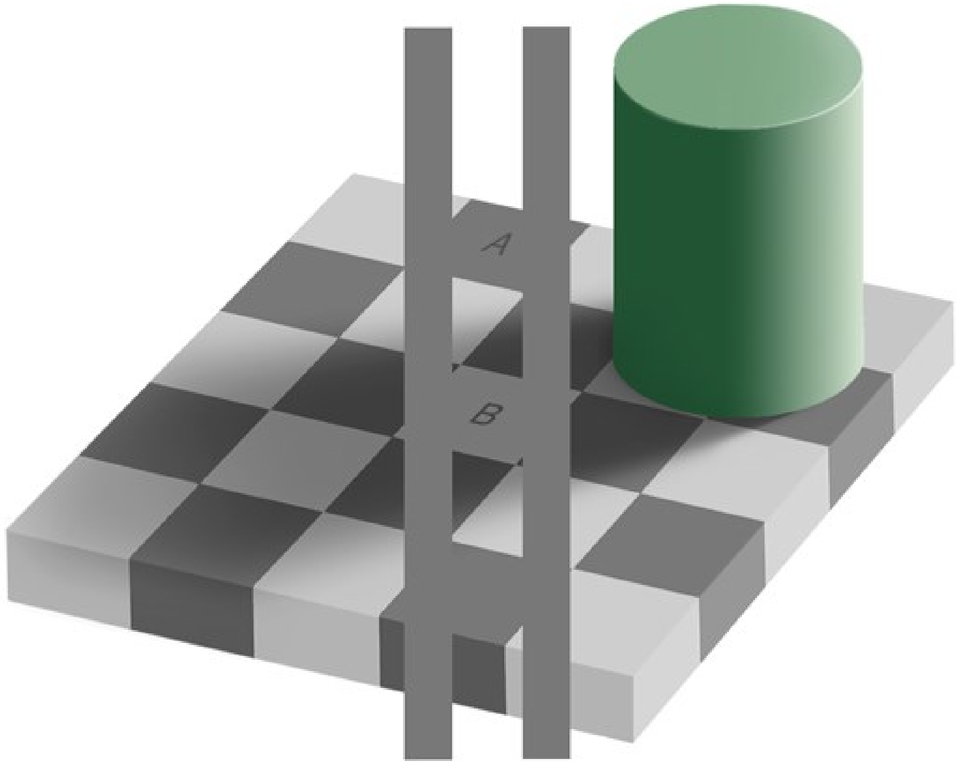

Red light

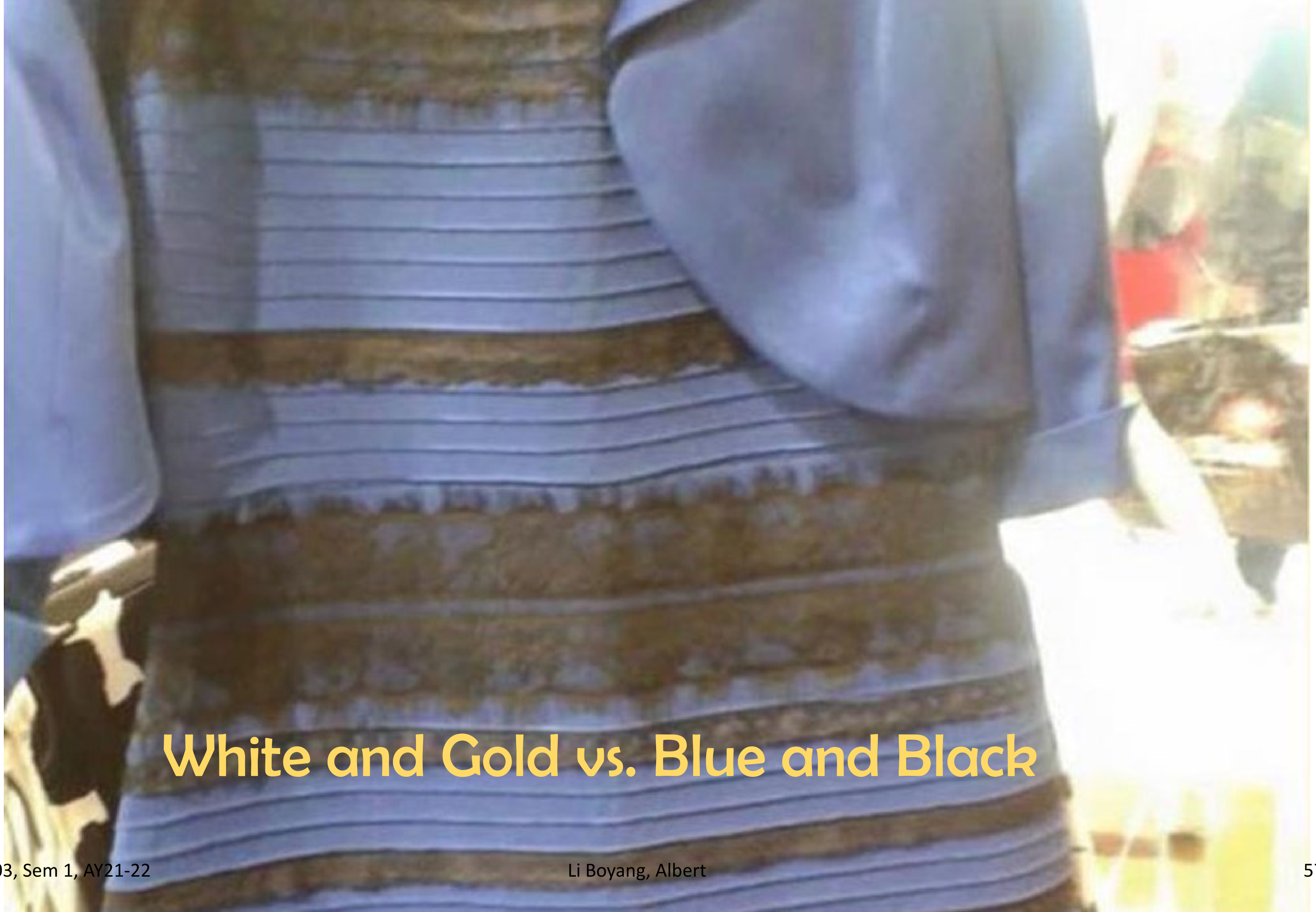
Viewpoint
change

Uncommon
Background

Lighting Invariance in Human Perception







White and Gold vs. Blue and Black

The human vision system is highly invariant

- It's built-in deep in our brain
- It offers evolutionary advantages
- It's hard to consciously override it.
- We are wired to not see certain things!

Invariance in Speech Perception

- Clearly, using visual information to guide speech perception is an advantage, especially when you don't hear too clearly due to background noise or other reasons.
- This can be used against us, too, creating the illusion.
- You are not even consciously aware of your brain's processing!

Information processing in our mind often happens at the subconscious level.

That is also why we can't write them down as rules using self-inspection.

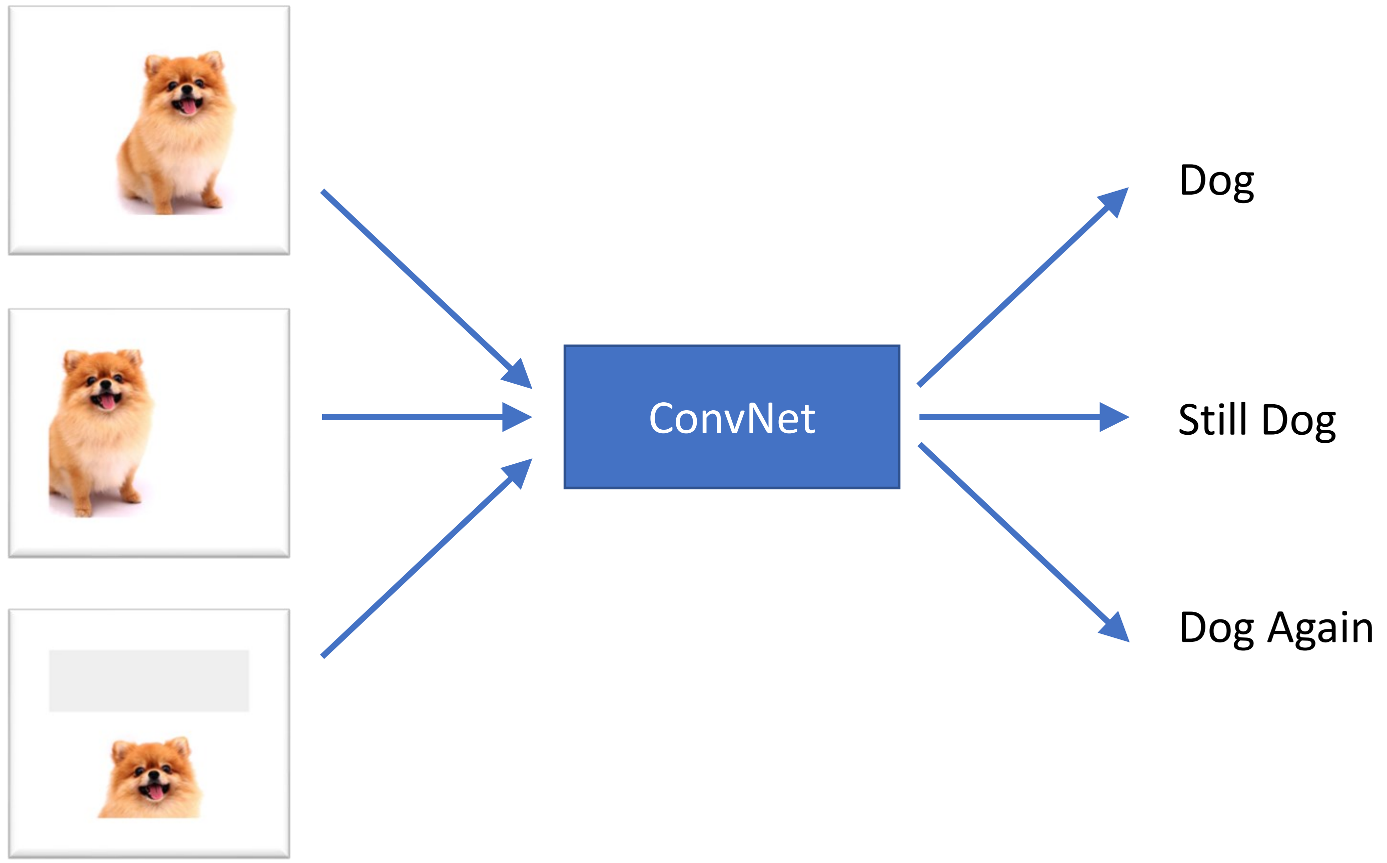
Invariance vs. Sensitivity

- We look for good vector representations of your input (image, text, audio, etc.)
- Good representations should be sensitive to the factors we care about for the task at hand
    - For animal classification, the animal species
    - For audio transcription, the phonemes (smallest unit of speech distinguishing words)
- Good representations should be invariant to factors irrelevant to the task.
    - For animal classification, it should be invariant to location of the animal, lighting, background color, hats, sunglasses, etc.
    - For location of lighting source, it should be sensitive to lighting

Invariance

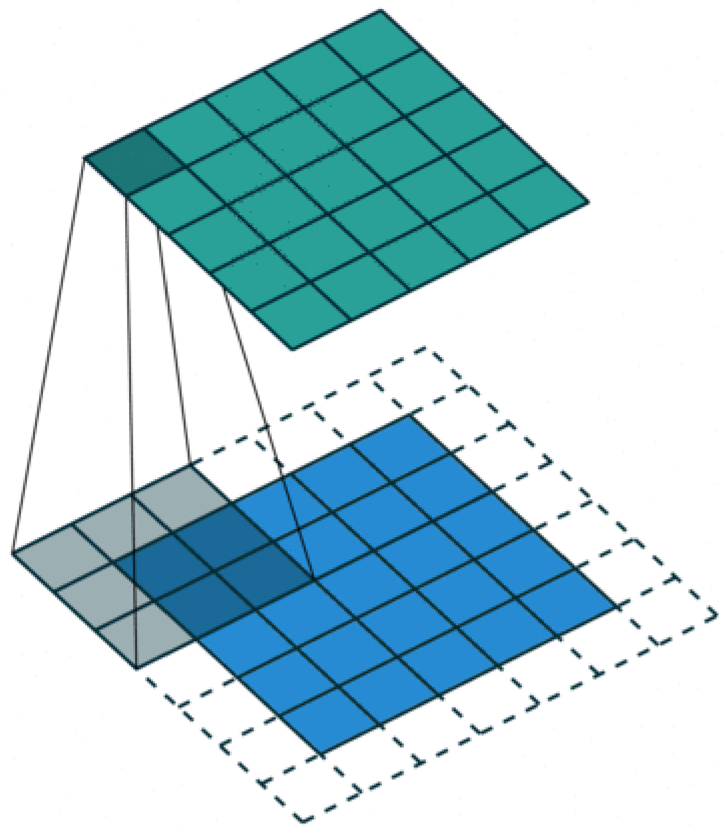
- How to create feature representations that are invariant to confounding factors is a crucial consideration in network design.
- Confounding factors = things that affect the input but should not affect the output.
- By definition, good invariance is application dependent.

Translational Invariance



Translational Invariance

- The convolution operation is inherently invariant to translation.
- Every image pixel, no matter its location, go through the same transformation.

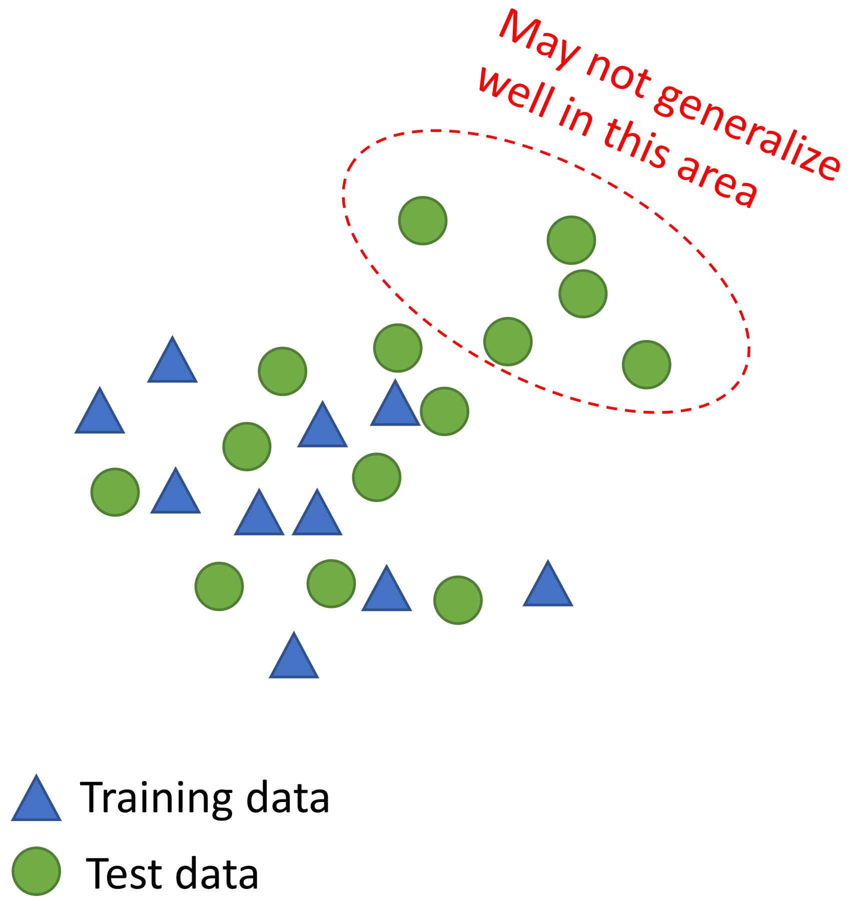


Inductive Bias

- Induction ≈ learning from data
- Inductive bias refers to the phenomenon that models and learning algorithms prefer to learn some types of functions over others.
- Could be achieved by
  - Model architecture, such as CNN
  - Optimization algorithm, such SGD
  - Regularization techniques

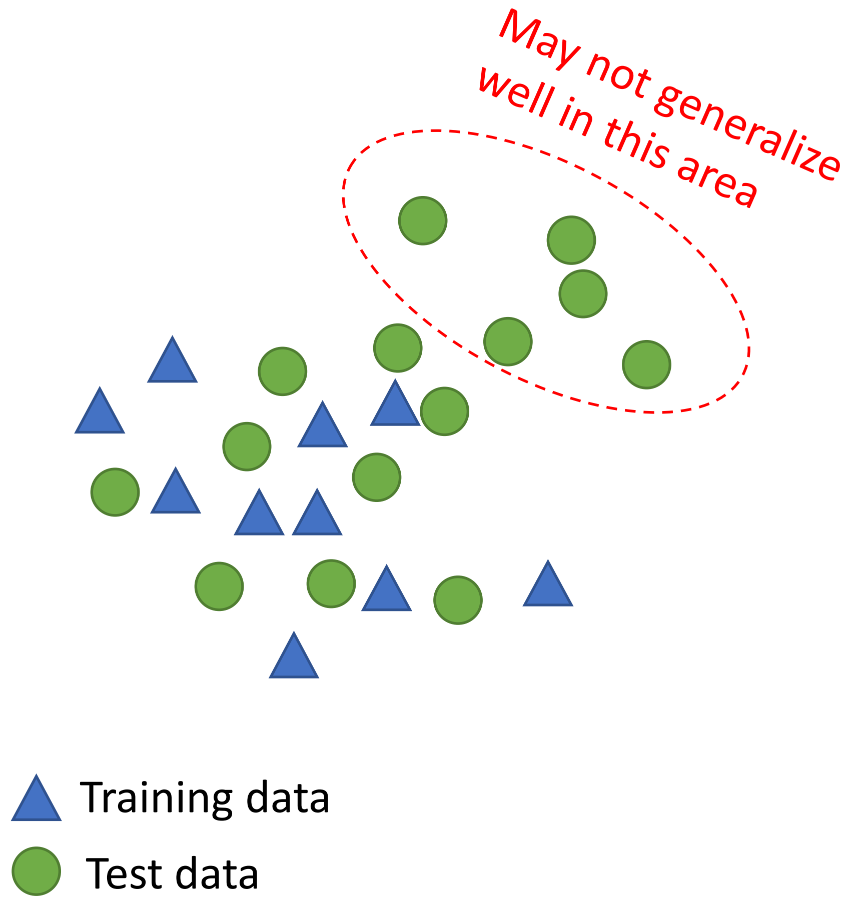
Another Perspective

- Models generalize well when the training data and test data are all drawn from the same distribution $P(X, Y)$
- This is perfectly satisfied when you have synthetic data
- In real-world data, this is hard to achieve.
    - Data-generating processes are very complex, and the observed data are only small samples in comparison



Another Perspective

- What if we can slightly modify training data so that they cover more volume of $P(X, Y)$?
- Answer: Data augmentation


May not generalize well in this area
Training data
Test data