Geometric structure of Data through Deep Learning models

Rita Fioresi, FaBiT, Unibo

May 2, 2023

Plan of the Talk

- Introduction to Deep Learning

Plan of the Talk

- Introduction to Deep Learning
- Information Geometry

Plan of the Talk

- Introduction to Deep Learning
- Information Geometry
- Data manifold and dimensionality reduction

Plan of the Talk

- Introduction to Deep Learning
- Information Geometry
- Data manifold and dimensionality reduction
- Main results (joint work with Grementieri)

Plan of the Talk

- Introduction to Deep Learning
- Information Geometry
- Data manifold and dimensionality reduction
- Main results (joint work with Grementieri)
- Conclusions and future directions

https://site.unibo.it/calista/en

CaLISTA Website

Join CaLISTA CA 21109!

- Working group 1: Cartan Geometry and Representation Theory

https://e-services.cost.eu/action/CA21109/working-groups/applications

https://site.unibo.it/calista/en

CaLISTA Website

Join CaLISTA CA 21109!

- Working group 1: Cartan Geometry and Representation Theory
- Working group 2: Integrable Systems and Supersymmetry

https://e-services.cost.eu/action/CA21109/working-groups/applications

https://site.unibo.it/calista/en

CaLISTA Website

Join CaLISTA CA 21109!

- Working group 1: Cartan Geometry and Representation Theory
- Working group 2: Integrable Systems and Supersymmetry
- Working group 3: Noncommutative Geometry and Quantum Homogeneous Spaces

https://e-services.cost.eu/action/CA21109/working-groups/applications

https://site.unibo.it/calista/en

CaLISTA Website

Join CaLISTA CA 21109!

- Working group 1: Cartan Geometry and Representation Theory
- Working group 2: Integrable Systems and Supersymmetry
- Working group 3: Noncommutative Geometry and Quantum Homogeneous Spaces
- Working group 4: Vision

https://e-services.cost.eu/action/CA21109/working-groups/applications

https://site.unibo.it/calista/en

CaLISTA Website

Join CaLISTA CA 21109!

- Working group 1: Cartan Geometry and Representation Theory
- Working group 2: Integrable Systems and Supersymmetry
- Working group 3: Noncommutative Geometry and Quantum Homogeneous Spaces
- Working group 4: Vision
- Working group 5: Dissemination and Public Engagement

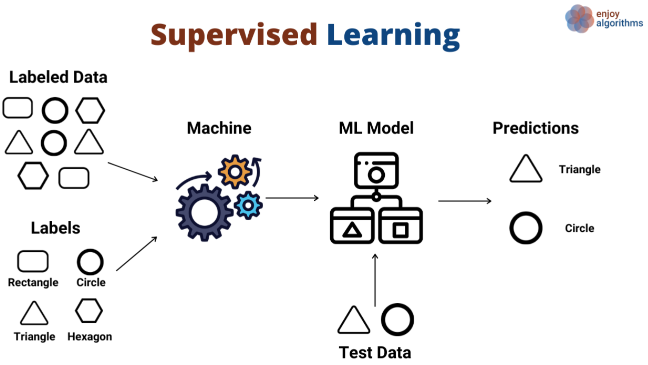
https://e-services.cost.eu/action/CA21109/working-groups/applications

- Deep Learning: Convolutional neural networks.

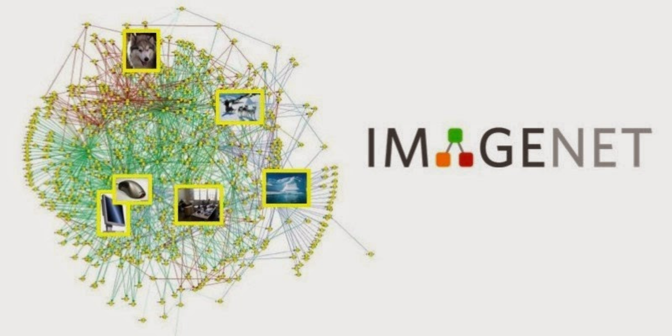
- Deep Learning: Convolutional neural networks.
- Deep Learning for Supervised Classification Tasks e.g. classification of images

- Deep Learning: Convolutional neural networks.
- Deep Learning for Supervised Classification Tasks e.g. classification of images

- Deep Learning: Convolutional neural networks.
- Deep Learning for Supervised Classification Tasks e.g. classification of images



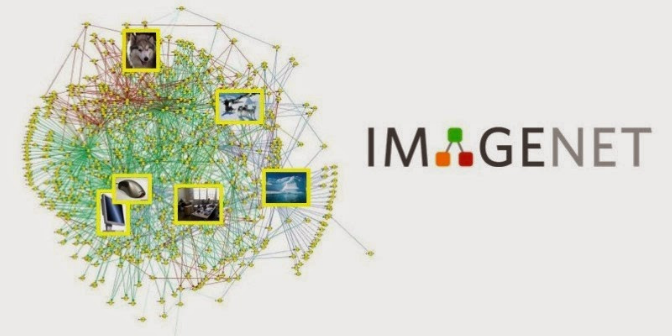
Imagenet Challenge ILSVRC: ImageNet Large Scale Visual Recognition Challenge



- **2010** 20000 images, 20 categories, 25% error.

2017: the challenge is declared won.

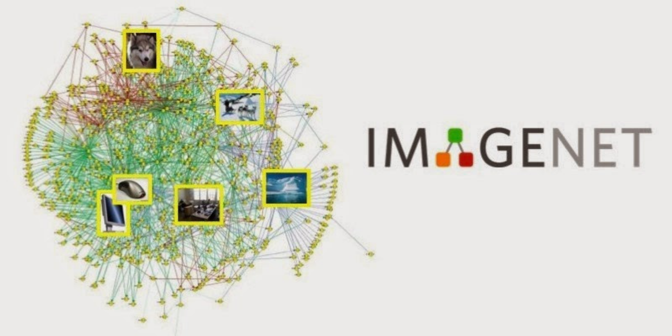
Imagenet Challenge ILSVRC: ImageNet Large Scale Visual Recognition Challenge



- **2010** 20000 images, 20 categories, 25% error.
- **2011** 1 million images, 1000 categories: 16% error.

2017: the challenge is declared won.

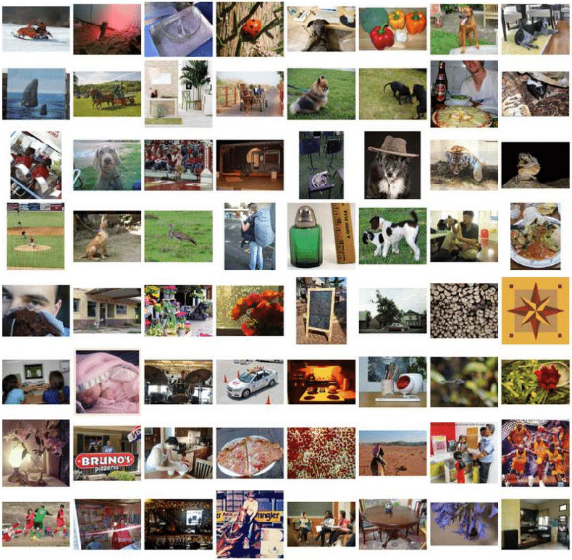
Imagenet Challenge ILSVRC: ImageNet Large Scale Visual Recognition Challenge



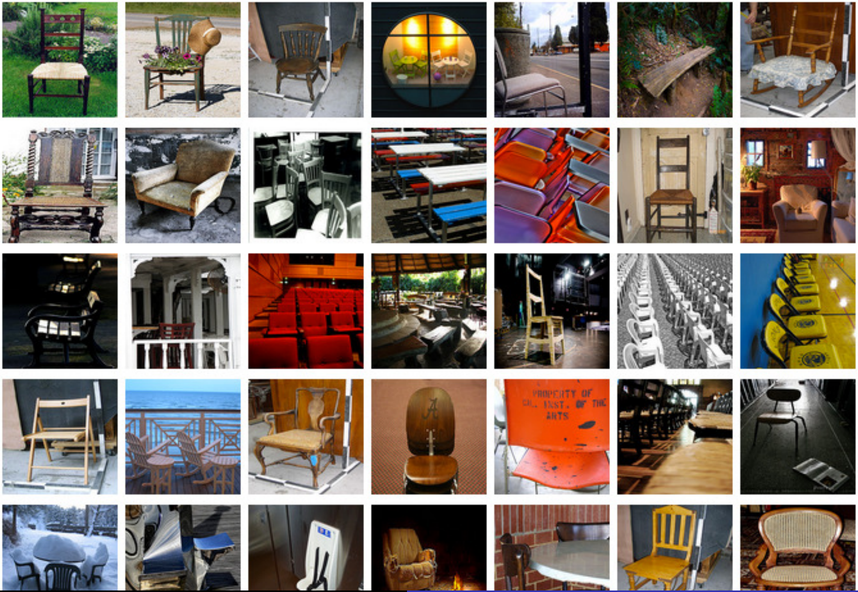
- **2010** 20000 images, 20 categories, 25% error.
- **2011** 1 million images, 1000 categories: 16% error.
- **2015** 1 million images, 1000 categories: 4% error.

2017: the challenge is declared won.

Images in Imagenet

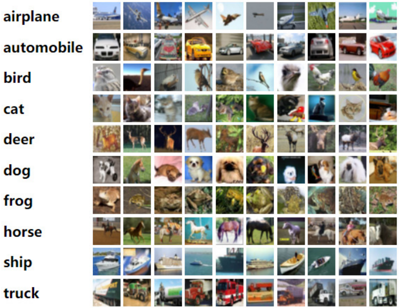


Images in Imagenet category "chair"

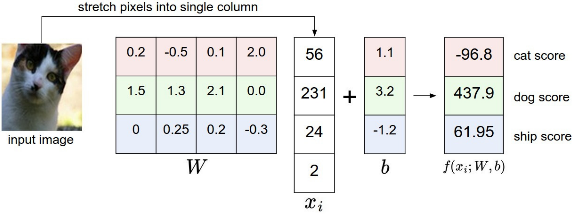


Benchmark datasets: MNIST and CIFAR10

Benchmark datasets: MNIST and CIFAR10

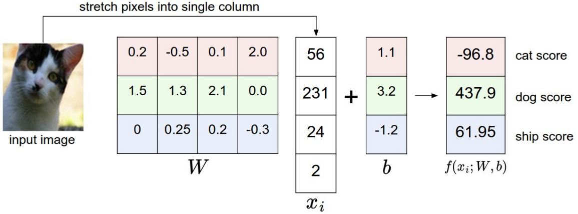


- **Score function**: it is a function of the weights w (es. linear classifier)



Ingredients for Deep Learning

- **Score function**: it is a function of the weights w (es. linear classifier)

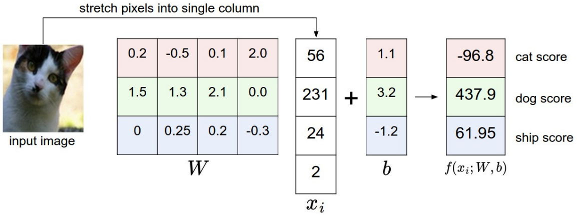


- **Loss function**: measures error
  (L_i datum i loss, y_i correct label)

$$L_i = -log \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} = -f_{y_i} + log \sum_j e^{f_j}, \qquad L = \sum_i L_i$$

Ingredients for Deep Learning

- **Score function**: it is a function of the weights w (es. linear classifier)



- **Loss function**: measures error
  (L_i datum i loss, y_i correct label)

$$L_i = -log \frac{e^{f_{y_i}}}{\sum_j e^{f_j}} = -f_{y_i} + log \sum_j e^{f_j}, \qquad L = \sum_i L_i$$

- **Optimizer**: for weights update "minimizes" the Loss

$$w_{ij}(t+1) = w_{ij}(t) - \alpha \nabla L_{\mathrm{stoc}}, \qquad \nabla L_{\mathrm{stoc}} = \sum_{i=1}^{32} \nabla L_{\mathrm{rand(i)}}$$

Training

Divide the dataset (ex. CIFAR10):
80% Data for **training**
10% Data for **validation**
10% Data for **test** (ONCE)

1. **Learning**: determine weights **parameters**

Accuracy: percentage of accurate predictions on tests set.

Divide the dataset (ex. CIFAR10):
80% Data for **training**
10% Data for **validation**
10% Data for **test** (ONCE)

1. **Learning**: determine weights **parameters**
2. **Validation**: determine net structure.
   Example: choose loss function, number of layers, learning rate
   Goal: find best hyperparameters.

Accuracy: percentage of accurate predictions on tests set.

Training

Divide the dataset (ex. CIFAR10):
80% Data for **training**
10% Data for **validation**
10% Data for **test** (ONCE)

1. **Learning**: determine weights **parameters**
2. **Validation**: determine net structure.
   Example: choose loss function, number of layers, learning rate
   Goal: find best hyperparameters.
3. **Test**: once at the end.

Accuracy: percentage of accurate predictions on tests set.

Learning process

- **Step 1**: Compute score of images in training set (**Forward pass**)
  The weights are inizialized randomly.

Learning process

- **Step 1**: Compute score of images in training set (**Forward pass**)
  The weights are inizialized randomly.
- **Step 2**: Compute the loss
  (i.e. measure the “difference” between given label and correct label for each datum in training set).

Learning process

- **Step 1**: Compute score of images in training set (**Forward pass**)
  The weights are inizialized randomly.
- **Step 2**: Compute the loss
  (i.e. measure the "difference" between given label and correct label for each datum in training set).
- **Step 3**: Compute **Stochastic Gradient**. (**Backpropagation**)

Learning process

- **Step 1**: Compute score of images in training set (**Forward pass**)
  The weights are inizialized randomly.
- **Step 2**: Compute the loss
  (i.e. measure the "difference" between given label and correct label for each datum in training set).
- **Step 3**: Compute **Stochastic Gradient**. (**Backpropagation**)
- **Step 4**: update weights.

Learning process

- **Step 1**: Compute score of images in training set (**Forward pass**)
  The weights are inizialized randomly.
- **Step 2**: Compute the loss
  (i.e. measure the “difference” between given label and correct label for each datum in training set).
- **Step 3**: Compute **Stochastic Gradient**. (**Backpropagation**)
- **Step 4**: update weights.
- **Step 5**: Repeat Step 1-2-3 up to an epoch.

Learning process

- **Step 1**: Compute score of images in training set (**Forward pass**)
  The weights are inizialized randomly.
- **Step 2**: Compute the loss
  (i.e. measure the "difference" between given label and correct label for each datum in training set).
- **Step 3**: Compute **Stochastic Gradient**. (**Backpropagation**)
- **Step 4**: update weights.
- **Step 5**: Repeat Step 1-2-3 up to an epoch.
- **Step 6**: After 150-200 epochs reduce learning rate and repeat all steps 1-5.

Learning process

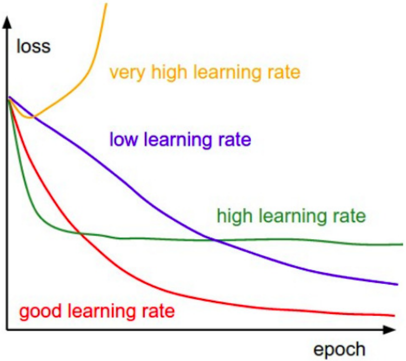
- **Step 1**: Compute score of images in training set (**Forward pass**)
  The weights are inizialized randomly.
- **Step 2**: Compute the loss
  (i.e. measure the "difference" between given label and correct label for each datum in training set).
- **Step 3**: Compute **Stochastic Gradient**. (**Backpropagation**)
- **Step 4**: update weights.
- **Step 5**: Repeat Step 1-2-3 up to an epoch.
- **Step 6**: After 150-200 epochs reduce learning rate and repeat all steps 1-5.

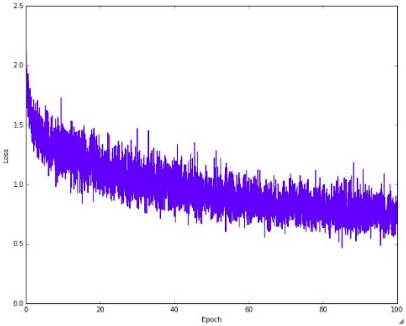
Learning process

- **Step 1**: Compute score of images in training set (**Forward pass**)
  The weights are inizialized randomly.
- **Step 2**: Compute the loss
  (i.e. measure the "difference" between given label and correct label for each datum in training set).
- **Step 3**: Compute **Stochastic Gradient**. (**Backpropagation**)
- **Step 4**: update weights.
- **Step 5**: Repeat Step 1-2-3 up to an epoch.
- **Step 6**: After 150-200 epochs reduce learning rate and repeat all steps 1-5.

Epoch$=$ $\|$Training set$\|/\|$minibatch size$\|$.
NOTE: measure accuracy every 10-20 epochs.
Example: 40000 training set (CIFAR10), 32 images in minibatch, 1 epoch$=40000/32$ updates.

Loss accuracy in epochs: CIFAR10





Validation process

Purpose of validation: Determine **hyperparameters**:

- α learning rate,

Validation process

Purpose of validation: Determine **hyperparameters**:

- α learning rate,
- $\mathcal{B}$ minibatch size,

Validation process

Purpose of validation: Determine **hyperparameters**:

- α learning rate,
- $\mathcal{B}$ minibatch size,
- optimizer (SGD, Adam),

Validation process

Purpose of validation: Determine **hyperparameters**:

- α learning rate,
- $\mathcal{B}$ minibatch size,
- optimizer (SGD, Adam),
- net structure (e.g. how many layers, parameters)

Validation process

Purpose of validation: Determine **hyperparameters**:

- α learning rate,
- $\mathcal{B}$ minibatch size,
- optimizer (SGD, Adam),
- net structure (e.g. how many layers, parameters)
- training (e.g. number of epochs)

Validation process

Purpose of validation: Determine **hyperparameters**:

- α learning rate,
- $\mathcal{B}$ minibatch size,
- optimizer (SGD, Adam),
- net structure (e.g. how many layers, parameters)
- training (e.g. number of epochs)

Validation process

Purpose of validation: Determine **hyperparameters**:

- α learning rate,
- $\mathcal{B}$ minibatch size,
- optimizer (SGD, Adam),
- net structure (e.g. how many layers, parameters)
- training (e.g. number of epochs)

We vary hyperparameters giving some values:

- e.g. $\alpha = 0.1, 00.1$ etc

We use the **validation set** to test accuracy, while searching for best hyperparameters.

Purpose of validation: Determine **hyperparameters**:

- α learning rate,
- $\mathcal{B}$ minibatch size,
- optimizer (SGD, Adam),
- net structure (e.g. how many layers, parameters)
- training (e.g. number of epochs)

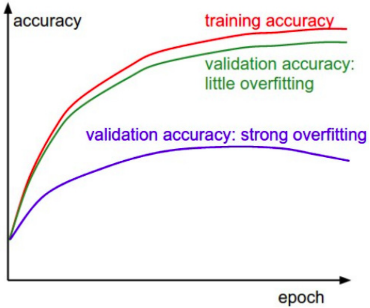
We vary hyperparameters giving some values:

- e.g. $\alpha = 0.1, 00.1$ etc
- e.g. $\mathcal{B} = 8, 16, 32$

We use the **validation set** to test accuracy, while searching for best hyperparameters.

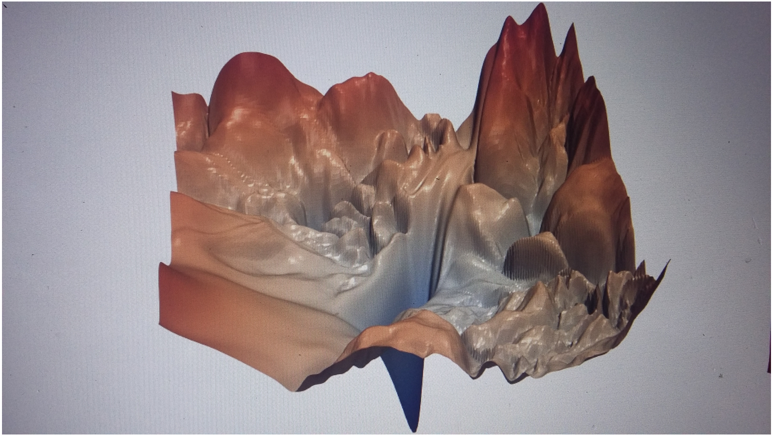
Test

ATTENTION!: use test set ONCE to avoid **overfitting**!



Validation technique: cross validation=rotation of the training set.

Loss Landscape



Loss (projection) as function of weights.

Information Geometry

Information Geometry: studies geometrical structures on manifolds in the parameter space and the data domain.

Amari, S.-I. *Natural gradient works efficiently in learning. Neural computation*, 10(2):251-276, 1998.

Information Geometry

Information Geometry: studies geometrical structures on manifolds in the parameter space and the data domain.

Amari, S.-I. *Natural gradient works efficiently in learning. Neural computation*, 10(2):251-276, 1998.

Amari Loss: $l(x, w) = -\log(p(y|x, w))$

Information Geometry

Information Geometry: studies geometrical structures on manifolds in the parameter space and the data domain.

Amari, S.-I. *Natural gradient works efficiently in learning. Neural computation*, 10(2):251-276, 1998.

Amari Loss: $l(x, w) = -\log(p(y|x, w))$

Loss function: $L(x, w) = \mathbb{E}_{y \sim q}[l(x, w)]$

Information Geometry: studies geometrical structures on manifolds in the parameter space and the data domain.

Amari, S.-I. *Natural gradient works efficiently in learning. Neural computation*, 10(2):251-276, 1998.

Amari Loss: $l(x, w) = -\log(p(y|x, w))$

Loss function: $L(x, w) = \mathbb{E}_{y\sim q}[l(x, w)]$

$$L(x, w) = \mathbb{E}_{y\sim q}[-\log(p(y|x, w))] = \mathrm{KL}(q(y|x)||p(y|x, w)) + \mathrm{constant}$$

$p(y|x, w) = (p_i(y|x, w))_{i=1,\dots,C}$: discrete probability distribution of data x

Information Geometry: studies geometrical structures on manifolds in the parameter space and the data domain.

Amari, S.-I. *Natural gradient works efficiently in learning. Neural computation*, 10(2):251-276, 1998.

Amari Loss: $l(x, w) = -\log(p(y|x, w))$

Loss function: $L(x, w) = \mathbb{E}_{y\sim q}[l(x, w)]$

$$L(x, w) = \mathbb{E}_{y\sim q}[-\log(p(y|x, w))] = \mathrm{KL}(q(y|x)||p(y|x, w)) + \mathrm{constant}$$

$p(y|x, w) = (p_i(y|x, w))_{i=1,\dots,C}$: discrete probability distribution of data x
$q(y|x)$: mass discrete probability distribution.
C: classification labels y.

Information Geometry

Information Geometry: studies geometrical structures on manifolds in the parameter space and the data domain.

Amari, S.-I. *Natural gradient works efficiently in learning. Neural computation*, 10(2):251-276, 1998.

Amari Loss: $l(x, w) = -\log(p(y|x, w))$

Loss function: $L(x, w) = \mathbb{E}_{y \sim q}[l(x, w)]$

$$L(x, w) = \mathbb{E}_{y \sim q}[-\log(p(y|x, w))] = \mathrm{KL}(q(y|x)||p(y|x, w)) + \mathrm{constant}$$

$p(y|x, w) = (p_i(y|x, w))_{i=1,\dots,C}$: discrete probability distribution of data x
$q(y|x)$: mass discrete probability distribution.
C: classification labels y.
w: parameters.

Loss Function

The empirical Loss function as expected value of the Amari Loss:

Loss Function

The empirical Loss function as expected value of the Amari Loss:

$$
\begin{aligned}
L(x,w) &= \mathbb{E}_{y\sim q}[-\log(p(y|x,w))] = \\
&= \sum_{i=1}^{C} q_i(y|x)\log\frac{q_i(y|x)}{p_i(y|x,w)} - \sum_{i=1}^{C} q_i(y|x)\log q_i(y|x) = \\
&= \mathrm{KL}(q(y|x)||p(y|x,w)) - \sum_{i=1}^{C} q_i(y|x)\log q_i(y|x). \qquad (1)
\end{aligned}
$$

Loss Function

The empirical Loss function as expected value of the Amari Loss:

$$
\begin{aligned}
L(x, w) &= \mathbb{E}_{y \sim q}[-\log(p(y|x, w))] = \\
&= \sum_{i=1}^{C} q_i(y|x) \log \frac{q_i(y|x)}{p_i(y|x, w)} - \sum_{i=1}^{C} q_i(y|x) \log q_i(y|x) = \\
&= \mathrm{KL}(q(y|x)||p(y|x, w)) - \sum_{i=1}^{C} q_i(y|x) \log q_i(y|x). \qquad (1)
\end{aligned}
$$

The Kullback-Leibler divergence measures the “difference” between the two probability distributions the “empirical distribution” p and the “true distribution” q.

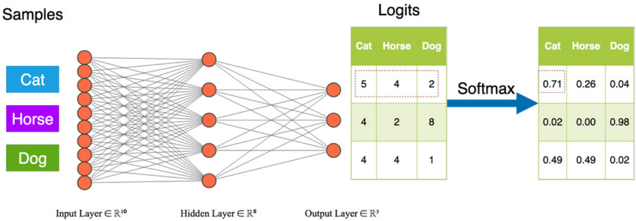
LOSS: Softmax S and Cross Entropy Loss L

$L(x, w) = -\log[S(x, w)] = -\log[e^{s_{y_j}(x)}/(e^{s_1(x)} + \cdots + e^{s_N(x)})]$
$L(x, w)$: loss of datum x with label y_j.

LOSS: Softmax S and Cross Entropy Loss L

$$L(x,w) = -\log[S(x,w)] = -\log[e^{s_{y_j}(x)}/(e^{s_1(x)} + \cdots + e^{s_N(x)})]$$

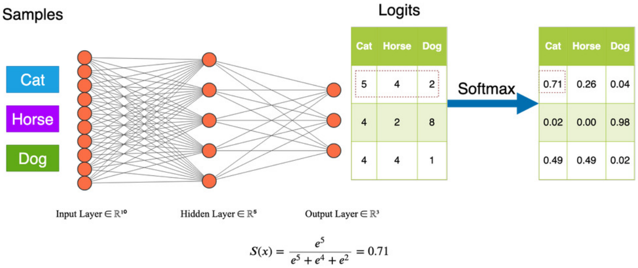
$L(x,w)$: loss of datum x with label y_j.



$$S(x) = \frac{e^5}{e^5 + e^4 + e^2} = 0.71$$

$L(x,w) = -\log[S(x,w)] = -\log[e^{s_{y_j}(x)}/(e^{s_1(x)} + \cdots + e^{s_N(x)})]$

$L(x,w)$: loss of datum x with label y_j.



$$Loss = -\log(S_{cat}) - \log(S_{horse}) - \log(S_{dog}) =$$

$$= -\log(0.71) - \log(0.002) - \log(0.02) = 0.34 + 6 + 3.91 = 10.25$$

The Fisher matrix F and the Local Data Matrix G

The Fisher matrix F and the Local Data Matrix G

$$
\begin{aligned}
F(x,w) &= \mathbb{E}_{y\sim p}[\nabla_w \log p(y|x,w)\cdot(\nabla_w \log p(y|x,w))^T] \\
G(x,w) &= \mathbb{E}_{y\sim p}[\nabla_x \log p(y|x,w)\cdot(\nabla_x \log p(y|x,w))^T].
\end{aligned}
$$

Key Facts:

The Fisher matrix F and the Local Data Matrix G

$$
\begin{aligned}
F(x,w) &= \mathbb{E}_{y\sim p}[\nabla_w \log p(y|x,w) \cdot (\nabla_w \log p(y|x,w))^T] \\
G(x,w) &= \mathbb{E}_{y\sim p}[\nabla_x \log p(y|x,w) \cdot (\nabla_x \log p(y|x,w))^T].
\end{aligned}
$$

Key Facts:

$$
\begin{aligned}
\mathrm{KL}(p(y|x,w+\delta w)||p(y|x,w)) &\cong \tfrac{1}{2}(\delta w)^T F(x,w)(\delta w) + \mathcal{O}(||\delta w||^3) \\
\mathrm{KL}(p(y|x+\delta x,w)||p(y|x,w)) &\cong \tfrac{1}{2}(\delta x)^T G(x,w)(\delta x) + \mathcal{O}(||\delta x||^3)
\end{aligned}
$$

The Fisher matrix F and the Local Data Matrix G

$$\begin{aligned} F(x,w) &= \mathbb{E}_{y\sim p}[\nabla_w \log p(y|x,w)\cdot(\nabla_w \log p(y|x,w))^T] \\ G(x,w) &= \mathbb{E}_{y\sim p}[\nabla_x \log p(y|x,w)\cdot(\nabla_x \log p(y|x,w))^T]. \end{aligned}$$

Key Facts:

$$\begin{aligned} \mathrm{KL}(p(y|x,w+\delta w)||p(y|x,w)) &\cong \tfrac{1}{2}(\delta w)^T F(x,w)(\delta w) + \mathcal{O}(||\delta w||^3) \\ \mathrm{KL}(p(y|x+\delta x,w)||p(y|x,w)) &\cong \tfrac{1}{2}(\delta x)^T G(x,w)(\delta x) + \mathcal{O}(||\delta x||^3) \end{aligned}$$

The Fisher matrix F provides a natural metric on the **parameter space** during dynamics of the stochastic gradient descent.

The Fisher matrix F and the Local Data Matrix G

$$
\begin{aligned}
F(x, w) &= \mathbb{E}_{y\sim p}[\nabla_w \log p(y|x, w) \cdot (\nabla_w \log p(y|x, w))^T] \\
G(x, w) &= \mathbb{E}_{y\sim p}[\nabla_x \log p(y|x, w) \cdot (\nabla_x \log p(y|x, w))^T].
\end{aligned}
$$

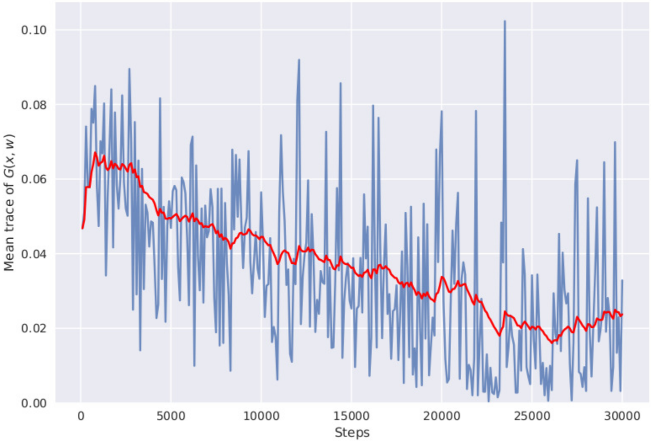
Key Facts:

$$
\begin{aligned}
\mathrm{KL}(p(y|x, w+\delta w)||p(y|x, w)) &\cong \tfrac{1}{2}(\delta w)^T F(x, w)(\delta w) + \mathcal{O}(||\delta w||^3) \\
\mathrm{KL}(p(y|x+\delta x, w)||p(y|x, w)) &\cong \tfrac{1}{2}(\delta x)^T G(x, w)(\delta x) + \mathcal{O}(||\delta x||^3)
\end{aligned}
$$

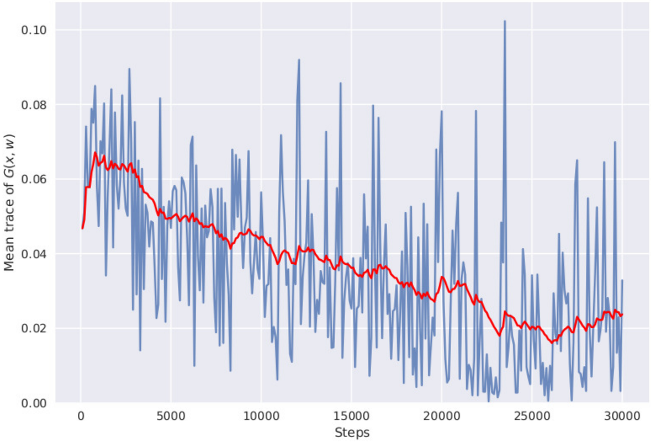
The Fisher matrix F provides a natural metric on the **parameter space** during dynamics of the stochastic gradient descent.
The Local Data matrix G provides a **natural metric on the data domain**.

The local data matrix G during optimization

The local data matrix G during optimization



The local data matrix G during optimization



This is why we do not want a fully trained model: the information is lost at equilibrium!

Properties of the Fisher matrix F and local data matrix G

1. $F(x, w)$ and $G(x, w)$ is a positive semidefinite symmetric matrix.

Properties of the Fisher matrix F and local data matrix G

1. $F(x, w)$ and $G(x, w)$ is a positive semidefinite symmetric matrix.
2. $\ker F(x, w) = (\mathrm{span}_{i=1,\dots,C}\{\nabla_w \log p_i(y|x, w)\})^{\perp}$;

Properties of the Fisher matrix F and local data matrix G

1. $F(x, w)$ and $G(x, w)$ is a positive semidefinite symmetric matrix.
2. $\ker F(x, w) = (\mathrm{span}_{i=1,\dots,C}\{\nabla_w \log p_i(y|x, w)\})^{\perp}$;
3. $\ker G(x, w) = (\mathrm{span}_{i=1,\dots,C}\{\nabla_x \log p_i(y|x, w)\})^{\perp}$.

Properties of the Fisher matrix F and local data matrix G

1. $F(x, w)$ and $G(x, w)$ is a positive semidefinite symmetric matrix.
2. $\ker F(x, w) = (\mathrm{span}_{i=1,\dots,C}\{\nabla_w \log p_i(y|x, w)\})^{\perp}$;
3. $\ker G(x, w) = (\mathrm{span}_{i=1,\dots,C}\{\nabla_x \log p_i(y|x, w)\})^{\perp}$.
4. $\mathrm{rank}\ F(x, w) < C, \quad \mathrm{rank}\ G(x, w) < C$.

Properties of the Fisher matrix F and local data matrix G

1. $F(x, w)$ and $G(x, w)$ is a positive semidefinite symmetric matrix.
2. $\ker F(x, w) = (\mathrm{span}_{i=1,\dots,C}\{\nabla_w \log p_i(y|x, w)\})^\perp$;
3. $\ker G(x, w) = (\mathrm{span}_{i=1,\dots,C}\{\nabla_x \log p_i(y|x, w)\})^\perp$.
4. $\mathrm{rank}\ F(x, w) < C, \quad \mathrm{rank}\ G(x, w) < C$.

Dataset	$G(x, w)$ size	$\mathrm{rank}\ G(x, w)$ bound
MNIST	784	10
CIFAR-10	3072	10
CIFAR-100	3072	100
ImageNet	150528	1000

C: is the number of classes for our classification task

Deep Learning and classification tasks:

Deep Learning and classification tasks:

- Data occupies a domain in $\mathbb{R}^n$

The Geometric Structure of Data

Deep Learning and classification tasks:

- Data occupies a domain in $\mathbb{R}^n$
  (e.g. MNIST in $\mathbb{R}^{784}$, $n = 784 = 28 \times 28$ pixels)

The Geometric Structure of Data

Deep Learning and classification tasks:

- Data occupies a domain in $\mathbb{R}^n$
  (e.g. MNIST in $\mathbb{R}^{784}$, $n = 784 = 28 \times 28$ pixels)
- The data domain is mostly composed of meaningless noise:

Deep Learning and classification tasks:

- Data occupies a domain in $\mathbb{R}^n$
  (e.g. MNIST in $\mathbb{R}^{784}$, $n = 784 = 28 \times 28$ pixels)
- The data domain is mostly composed of meaningless noise: data occupy a thin region of it!

Deep Learning and classification tasks:

- Data occupies a domain in $\mathbb{R}^n$
(e.g. MNIST in $\mathbb{R}^{784}$, $n = 784 = 28 \times 28$ pixels)
- The data domain is mostly composed of meaningless noise: data occupy a thin region of it!

Main result:

Deep Learning and classification tasks:

- Data occupies a domain in $\mathbb{R}^n$ (e.g. MNIST in $\mathbb{R}^{784}$, $n = 784 = 28 \times 28$ pixels)
- The data domain is mostly composed of meaningless noise: data occupy a thin region of it!

Main result:

1. A partially trained neural network decomposes the data domain in $\mathbb{R}^n$ as the disjoint union of submanifolds (the **leaves** of a foliation).

The Geometric Structure of Data

Deep Learning and classification tasks:

- Data occupies a domain in $\mathbb{R}^n$ (e.g. MNIST in $\mathbb{R}^{784}$, $n = 784 = 28 \times 28$ pixels)
- The data domain is mostly composed of meaningless noise: data occupy a thin region of it!

Main result:

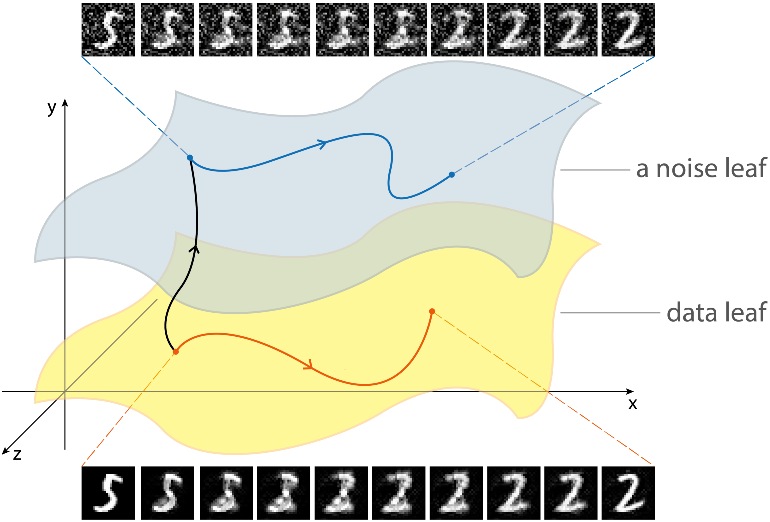
1. A partially trained neural network decomposes the data domain in $\mathbb{R}^n$ as the disjoint union of submanifolds (the **leaves** of a foliation).
2. The dimension d of every submanifold (every leaf of the foliation) is bounded by the number of classes C of our classification model: $d << n$ (e.g. MNIST $d = 9 << 784$).

Data leaf versus Noise leaf

The data domain is the disjoint union of subdomains (foliation) and the **training data are all on one leaf**.

Data leaf versus Noise leaf

The data domain is the disjoint union of subdomains (foliation) and the **training data are all on one leaf**.



Main Result/1. Let w be the weights of a deep ReLU neural network classifier, p given by softmax, $G(x, w)$ the local data matrix.

Main Result/1. Let w be the weights of a deep ReLU neural network classifier, p given by softmax, $G(x, w)$ the local data matrix.
The distribution in the data domain:

Main Result/1. Let w be the weights of a deep ReLU neural network classifier, p given by softmax, $G(x, w)$ the local data matrix.

The distribution in the data domain:

$$x \mapsto \mathcal{D}_x = (\ker G(x, w))^{\perp}$$

is involutive i.e.

Data manifold

Main Result/1. Let w be the weights of a deep ReLU neural network classifier, p given by softmax, $G(x,w)$ the local data matrix.
The distribution in the data domain:

$$x \mapsto \mathcal{D}_x = (\ker G(x,w))^{\perp}$$

is involutive i.e.

$$[X,Y] \in \mathcal{D}, \qquad \forall\ X,Y \in \mathcal{D}.$$

Main result/2.

1. At each point in the dataset in $\mathbb{R}^n$, $\ker G(x,w)^{\perp}$ is tangent to a submanifold (**data leaf**) of dimension $\mathrm{rank}\ G(x,w) < C$

Data manifold

Main Result/1. Let w be the weights of a deep ReLU neural network classifier, p given by softmax, $G(x, w)$ the local data matrix.
The distribution in the data domain:

$$x \mapsto \mathcal{D}_x = (\ker G(x, w))^{\perp}$$

is involutive i.e.

$$[X, Y] \in \mathcal{D}, \qquad \forall\, X, Y \in \mathcal{D}.$$

Main result/2.

1. At each point in the dataset in $\mathbb{R}^n$, $\ker G(x, w)^{\perp}$ is tangent to a submanifold (**data leaf**) of dimension $\mathrm{rank}\; G(x, w) < C$
2. G defines a foliation on $\mathbb{R}^n$ of rank at most $C - 1$ (**Frobenius Thm**).

Data manifold

Main Result/1. Let w be the weights of a deep ReLU neural network classifier, p given by softmax, $G(x, w)$ the local data matrix.

The distribution in the data domain:

$$x \mapsto \mathcal{D}_x = (\ker G(x, w))^\perp$$

is involutive i.e.

$$[X, Y] \in \mathcal{D}, \qquad \forall\, X, Y \in \mathcal{D}.$$

Main result/2.

1. At each point in the dataset in $\mathbb{R}^n$, $\ker G(x, w)^\perp$ is tangent to a submanifold (**data leaf**) of dimension $\mathrm{rank}\; G(x, w) < C$
2. G defines a foliation on $\mathbb{R}^n$ of rank at most $C - 1$ (**Frobenius Thm**).

Remark: This is not true for the distribution via the Fisher matrix!

$$w \mapsto \mathcal{D}'_w := (\ker F(w))^\perp$$

is **not** involutive (e.g. MNIST, lenet).

Facts

Facts

- The matrix $G(x, w)$, restricted to the subspace $(\ker G(x, w))^{\perp}$ gives a Riemannian metric to each leaf of the foliation.

Facts

- The matrix $G(x, w)$, restricted to the subspace $(\ker G(x, w))^{\perp}$ gives a Riemannian metric to each leaf of the foliation.
- All the dataset is on one leaf: the **data leaf**

Riemannian Structure on the Data Manifold

Facts

- The matrix $G(x, w)$, restricted to the subspace $(\ker G(x, w))^{\perp}$ gives a Riemannian metric to each leaf of the foliation.
- All the dataset is on one leaf: the **data leaf**
  We perform dimensionality reduction!
- We move from a point x in our dataset to any other point x' in the dataset with an with an *horizontal* path, that is a path on the data leaf.

Facts

- The matrix $G(x, w)$, restricted to the subspace $(\ker G(x, w))^{\perp}$ gives a Riemannian metric to each leaf of the foliation.
- All the dataset is on one leaf: the **data leaf**
  We perform dimensionality reduction!
- We move from a point x in our dataset to any other point x' in the dataset with an with an *horizontal* path, that is a path on the data leaf.
- Not all points on the data leaf are in the data set, but they represent *symbols*.

Moving on the data leaf: MNIST

Moving around in **on the data leaf**:

- We can connect any two data=images.

Moving on the data leaf: MNIST

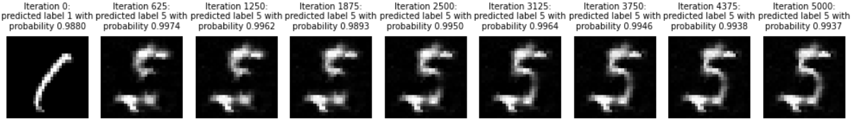
Moving around in **on the data leaf**:

- We can connect any two data=images.
- Any path starting from one image and going to another goes through data with the same level of noise.

Moving on the data leaf: MNIST

Moving around in **on the data leaf**:

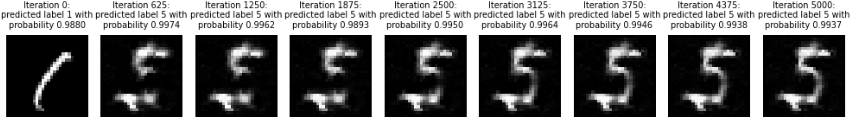
- We can connect any two data=images.
- Any path starting from one image and going to another goes through data with the same level of noise.



Moving on the data leaf: MNIST

Moving around in **on the data leaf**:

- We can connect any two data=images.
- Any path starting from one image and going to another goes through data with the same level of noise.

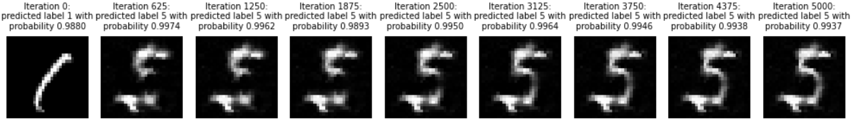


We can connect a digit from MNIST to a symbol **not** in MNIST moving on the **data leaf**:

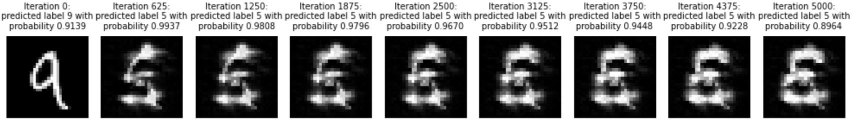
Moving on the data leaf: MNIST

Moving around in **on the data leaf**:

- We can connect any two data=images.
- Any path starting from one image and going to another goes through data with the same level of noise.



We can connect a digit from MNIST to a symbol **not** in MNIST moving on the **data leaf**:

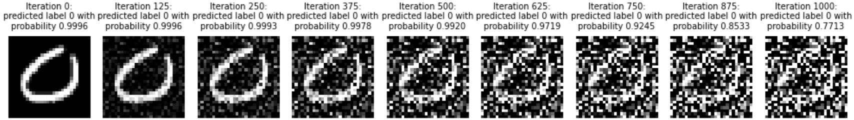


Moving away from the data leaf: MNIST

When moving **away** from a given data leaf, noise is added, but the accuracy is high.

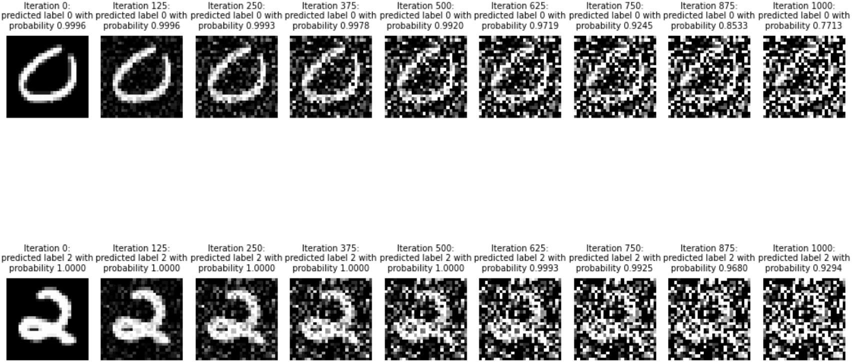
Moving away from the data leaf: MNIST

When moving **away** from a given data leaf, noise is added, but the accuracy is high.



Moving away from the data leaf: MNIST

When moving **away** from a given data leaf, noise is added, but the accuracy is high.

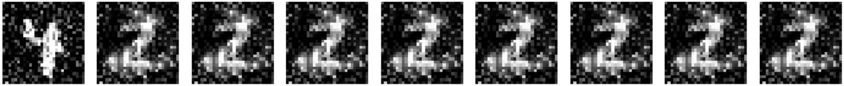


Moving on a noisy leaf: MNIST

We can connect a noisy datum with any other datum with the **same** level of noise:

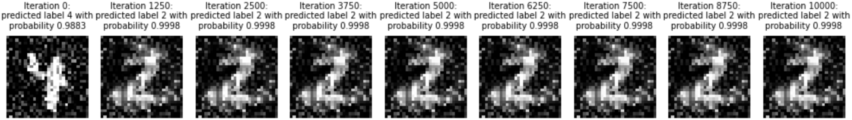
Moving on a noisy leaf: MNIST

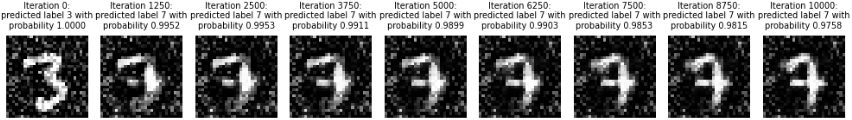
We can connect a noisy datum with any other datum with the **same** level of noise:



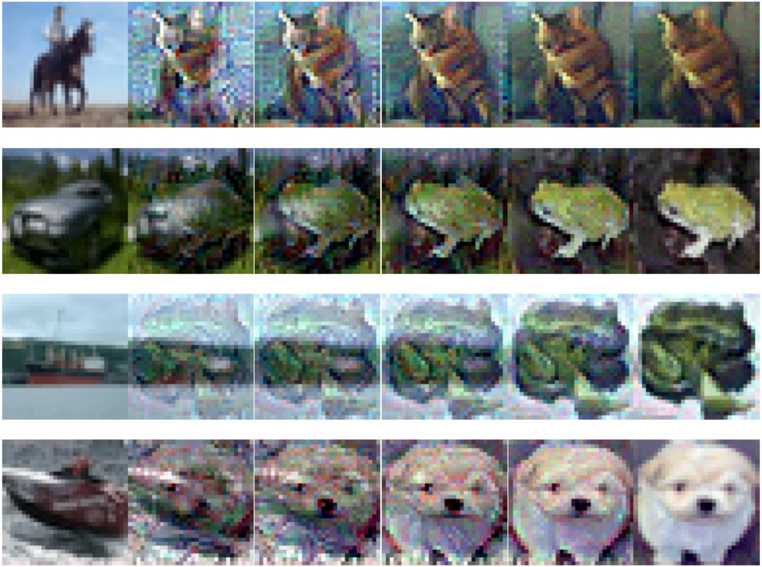
Moving on a noisy leaf: MNIST

We can connect a noisy datum with any other datum with the **same** level of noise:





Moving on the data manifold: CIFAR10



Conclusions

- Using a partially trained model we can construct a low dimensional submanifold the **data leaf** of $\mathbb{R}^n$ containing the data the model was trained with.

Conclusions

- Using a partially trained model we can construct a low dimensional submanifold the **data leaf** of $\mathbb{R}^n$ containing the data the model was trained with.
- We can navigate the data leaf and obtain either data or points with similarities to our data.

Conclusions

- Using a partially trained model we can construct a low dimensional submanifold the **data leaf** of $\mathbb{R}^n$ containing the data the model was trained with.
- We can navigate the data leaf and obtain either data or points with similarities to our data.
- Moving orthogonally to the data leaf will add noise to data, but the model will not change its accuracy.

Conclusions

- Using a partially trained model we can construct a low dimensional submanifold the **data leaf** of $\mathbb{R}^n$ containing the data the model was trained with.
- We can navigate the data leaf and obtain either data or points with similarities to our data.
- Moving orthogonally to the data leaf will add noise to data, but the model will not change its accuracy.

1. Possible Applications:

Conclusions

- Using a partially trained model we can construct a low dimensional submanifold the **data leaf** of $\mathbb{R}^n$ containing the data the model was trained with.
- We can navigate the data leaf and obtain either data or points with similarities to our data.
- Moving orthogonally to the data leaf will add noise to data, but the model will not change its accuracy.

1. Possible Applications:
   - Denoising of images: project a noisy data point on the data leaf to perform denoising.

Conclusions

- Using a partially trained model we can construct a low dimensional submanifold the **data leaf** of $\mathbb{R}^n$ containing the data the model was trained with.
- We can navigate the data leaf and obtain either data or points with similarities to our data.
- Moving orthogonally to the data leaf will add noise to data, but the model will not change its accuracy.

1. Possible Applications:
   - Denoising of images: project a noisy data point on the data leaf to perform denoising.
   - Use the distance from the data leaf to recognize out-of-distribution examples

Conclusions

- Using a partially trained model we can construct a low dimensional submanifold the **data leaf** of $\mathbb{R}^n$ containing the data the model was trained with.
- We can navigate the data leaf and obtain either data or points with similarities to our data.
- Moving orthogonally to the data leaf will add noise to data, but the model will not change its accuracy.

1. Possible Applications:
   - Denoising of images: project a noisy data point on the data leaf to perform denoising.
   - Use the distance from the data leaf to recognize out-of-distribution examples
   - GAN: generate new images with the same label, by moving on the data leaf.

Conclusions

- Using a partially trained model we can construct a low dimensional submanifold the **data leaf** of $\mathbb{R}^n$ containing the data the model was trained with.
- We can navigate the data leaf and obtain either data or points with similarities to our data.
- Moving orthogonally to the data leaf will add noise to data, but the model will not change its accuracy.

1. Possible Applications:
   - Denoising of images: project a noisy data point on the data leaf to perform denoising.
   - Use the distance from the data leaf to recognize out-of-distribution examples
   - GAN: generate new images with the same label, by moving on the data leaf.

Future Directions

We need to understand the geometry and the metric structure of the data leaf.

- It not a riemannian and not a subriemannian manifold

Future Directions

We need to understand the geometry and the metric structure of the data leaf.

- It not a riemannian and not a subriemannian manifold
- The involutive distribution defining the data leaf is not constant rank: we have a singular foliation!

Future Directions

We need to understand the geometry and the metric structure of the data leaf.

- It not a riemannian and not a subriemannian manifold
- The involutive distribution defining the data leaf is not constant rank: we have a singular foliation!
- What are the geodesics in this geometry? (proto-sub riemannian geometry)

Future Directions

We need to understand the geometry and the metric structure of the data leaf.

- It not a riemannian and not a subriemannian manifold
- The involutive distribution defining the data leaf is not constant rank: we have a singular foliation!
- What are the geodesics in this geometry? (proto-sub riemannian geometry)
- Navigating the data leaf can lead to data augmentation and efficient denoising algorithms

Bibliography

- Amari, S.-I. *Natural gradient works efficiently in learning. Neural computation*, 10(2):251–276, 1998.

Bibliography

- Amari, S.-I. *Natural gradient works efficiently in learning. Neural computation*, 10(2):251–276, 1998.
- Grementieri, L., Fioresi, R. *Model-centric Data Manifold: the Data Through the Eyes of the Model*, SIAM Journal on Imaging Sciences, pp. 1140 – 1156, 2022.

Bibliography

- Amari, S.-I. *Natural gradient works efficiently in learning. Neural computation*, 10(2):251–276, 1998.
- Grementieri, L., Fioresi, R. *Model-centric Data Manifold: the Data Through the Eyes of the Model*, SIAM Journal on Imaging Sciences, pp. 1140 – 1156, 2022.
- Bergomi, M. G., Frosini, P., Giorgi, D., and Quercioli, N. *Towards a topological–geometrical theory of group equivariant non-expansive operators for data analysis and machine learning*. Nature Machine Intelligence, 1 (9):423–433, 2019.

Bibliography

- Amari, S.-I. *Natural gradient works efficiently in learning. Neural computation*, 10(2):251–276, 1998.
- Grementieri, L., Fioresi, R. *Model-centric Data Manifold: the Data Through the Eyes of the Model*, SIAM Journal on Imaging Sciences, pp. 1140 – 1156, 2022.
- Bergomi, M. G., Frosini, P., Giorgi, D., and Quercioli, N. *Towards a topological–geometrical theory of group equivariant non-expansive operators for data analysis and machine learning*. Nature Machine Intelligence, 1 (9):423–433, 2019.
- Sommer, S. and Bronstein, A. M. *Horizontal flows and manifold stochastics in geometric deep learning*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2020.