The Geometry of Deep Learning. Lecture 2: Deep Learning

Rita Fioresi, FaBiT, Unibo

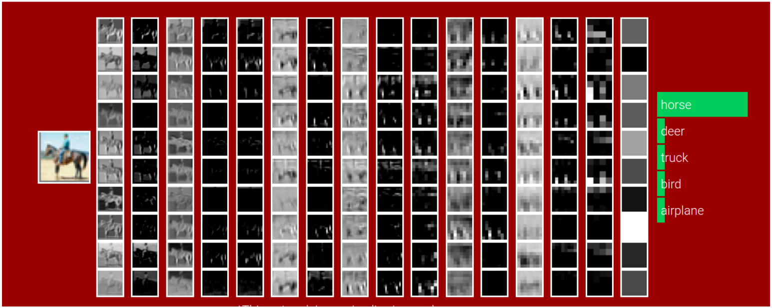
December 19, 2025

Deep Learning: CS231 Stanford

CS231n: Convolutional Neural Networks for Visual Recognition

Spring 2020

Previous Years: [Winter 2015] [Winter 2016] [Spring 2017] [Spring 2018] [Spring 2019]



Stanford CS231

- **Score function**: it is a function of the weights w

$$f(x) = W_3 max(0, W_2 max(0, W_1 x))$$

- **Loss function**: measures error (L_i loss of datum i)

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

- **Optimizer**: for weights update "minimizes" the Loss (via stochastic gradient):

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

Training

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

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

Accuracy: percentage of accurate predictions on tests set.

1. Learning process

- **Step 1**: (**Forward pass**)
  Compute the score of each image in the training set.
  The weights are inizialized randomly.
- **Step 2**: Compute the loss (it measure the "difference" between given label and correct label for each datum in training set).
- **Step 3**: (**Backpropagation**) Compute **Stochastic Gradient**.
- **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 (most optimizer now allow for automatic drop).

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.

Forward pass (green), Backpropagation (red)

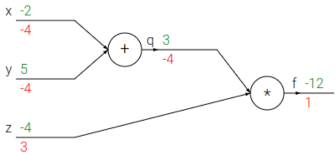
```
x = -2; y = 5; z = -4

# perform the forward pass
q = x + y # q becomes 3
f = q * z # f becomes -12

# perform the backward pass (backpropagation) in reverse order:
# first backprop through f = q * z
dfdz = q # df/dz = q, so gradient on z becomes 3
dfdq = z # df/dq = z, so gradient on q becomes -4
dqdx = 1.0
dqdy = 1.0
# now backprop through q = x + y
dfdx = dfdq * dqdx  # The multiplication here is the chain rule!
dfdy = dfdq * dqdy
```

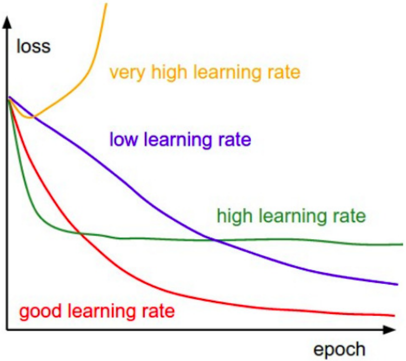
We are left with the gradient in the variables `[dfdx,dfdy,dfdz]`, which tell us the sensitivity of the variables `x,y,z` on `f`!. This is the simplest example of backpropagation. Going forward, we will use a more concise notation that omits the `df` prefix. For example, we will simply write `dq` instead of `dfdq`, and always assume that the gradient is computed on the final output.

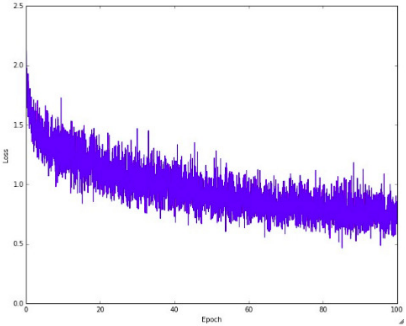
This computation can also be nicely visualized with a circuit diagram:



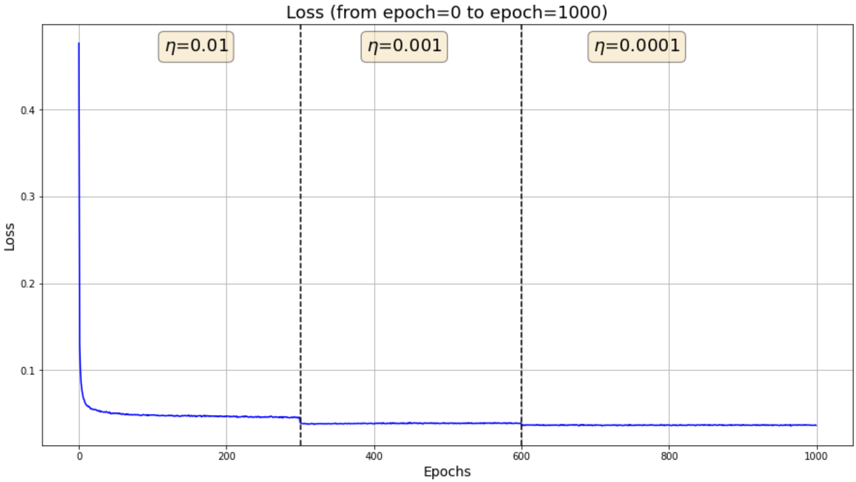
The real-valued *"circuit"* on left shows the visual representation of the computation. The **forward pass** computes values from inputs to output (shown in green). The **backward pass** then performs backpropagation which starts at the end and recursively applies the chain rule to compute the gradients (shown in red) all the way to the inputs of the circuit. The gradients can be thought of as flowing backwards through the circuit.

Loss accuracy in epochs: CIFAR10





Loss accuracy in epochs: MNIST



2. 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)

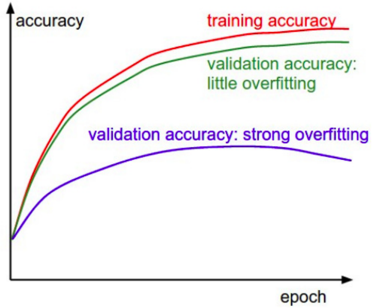
We vary hyperparameters giving some values:

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

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

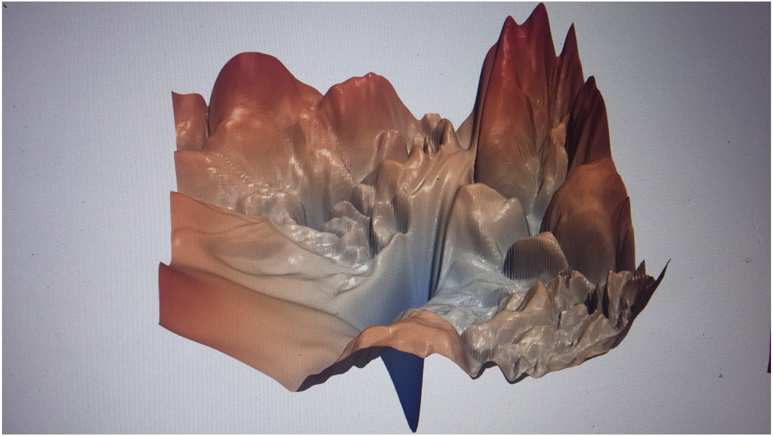
3. Test

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



Validation technique: cross validation=rotation of the training set (useful for small datasets, e.g. biological ones).

Loss Landscape



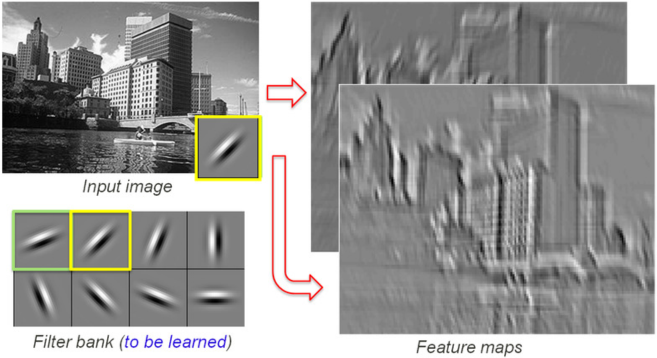
Loss (projection) as function of weights.

The Deep Learning Algorithm in more detail

1. **Score function**
   (convolutions)
2. **Loss function**
   (ex.: cross entropy with softmax)
3. **Optimizer**
   (example: stochastic gradient)

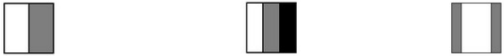
1. SCORE function: Convolutional Neural Networks

Convolutions: extract *features* from images.
Represent the mathematical operation of discrete convolutions via kernels (here called filters).



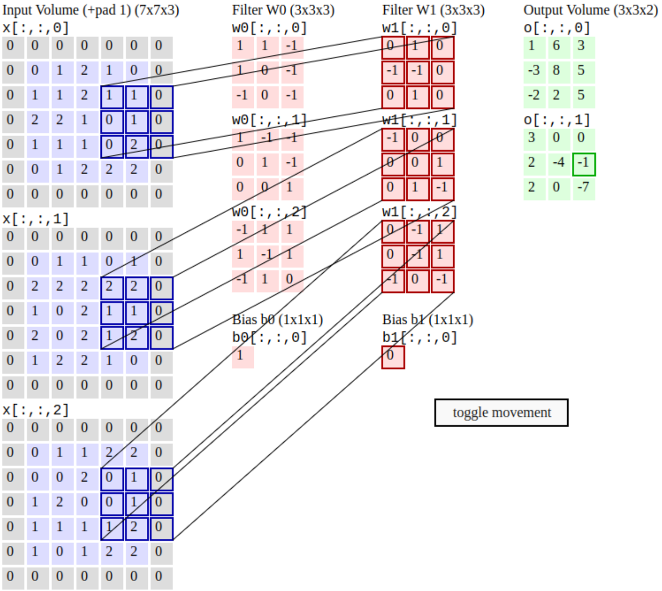
Convolutions and “edge detection”

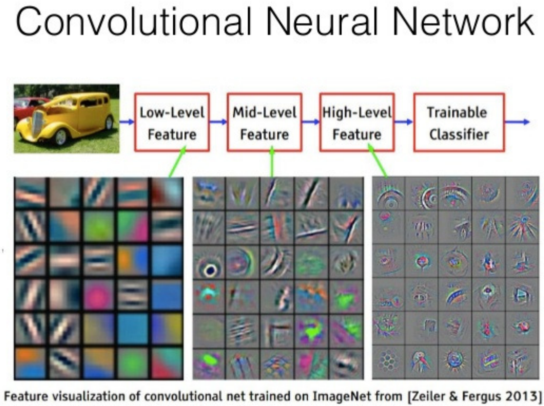
$$\underbrace{\begin{array}{|c|c|c|c|c|c|}\hline 10&10&10&0&0&0\\\hline 10&10&10&0&0&0\\\hline 10&10&10&0&0&0\\\hline 10&10&10&0&0&0\\\hline 10&10&10&0&0&0\\\hline 10&10&10&0&0&0\\\hline\end{array}}_{6\times 6} * \underbrace{\begin{array}{|c|c|c|}\hline 1&0&-1\\\hline 1&0&-1\\\hline 1&0&-1\\\hline\end{array}}_{3\times 3} = \underbrace{\begin{array}{|c|c|c|c|}\hline -0&30&30&0\\\hline 0&30&30&0\\\hline 0&30&30&0\\\hline 0&30&30&0\\\hline\end{array}}_{4\times 4}$$



Convolutions

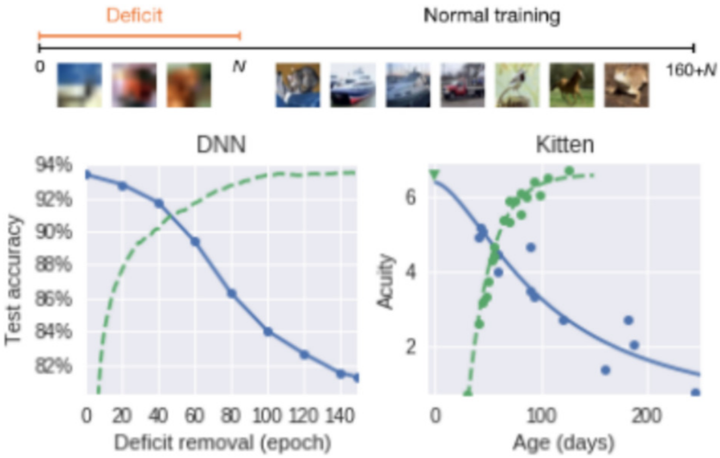
Filters/1


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 1 2 1 0 0
0 1 1 2 1 1 0
0 2 2 1 0 1 0
0 1 1 1 0 2 0
0 0 1 2 2 2 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 1 1 0 1 0
0 2 2 2 2 2 0
0 1 0 2 1 1 0
0 2 0 2 1 2 0
0 1 2 2 1 0 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 1 1 2 2 0
0 0 0 2 0 1 0
0 1 2 0 0 1 0
0 1 1 1 1 2 0
0 1 0 1 2 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 1 -1
1 0 -1
-1 0 -1
w0[:,:,1]
1 -1 -1
0 1 -1
0 0 1
w0[:,:,2]
-1 1 1
1 -1 1
-1 1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
0 1 0
-1 -1 0
0 1 0
w1[:,:,1]
-1 0 0
0 0 1
0 1 -1
w1[:,:,2]
0 -1 1
0 -1 1
-1 0 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
1 6 3
-3 8 5
-2 2 5
o[:,:,1]
3 0 0
2 -4 -1
2 0 -7
toggle movement



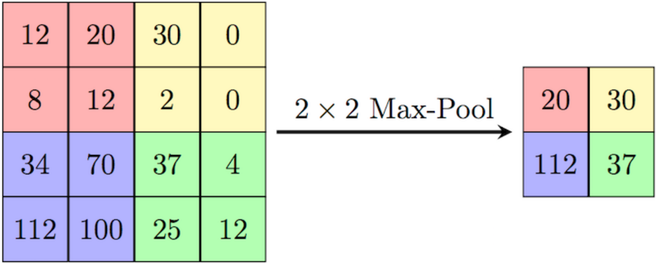
The algorithm **learns** filters!
https://arxiv.org/abs/1711.08856

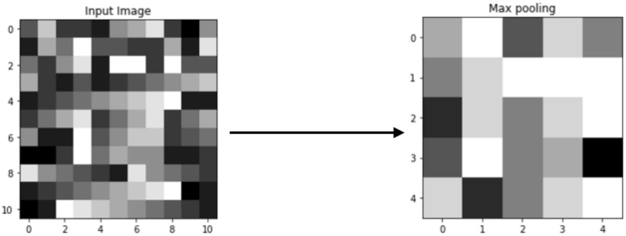
SCORE: DL filters and mammals visual system



Filters are **learned** as for mammals visual cortex! (Critical periods...)

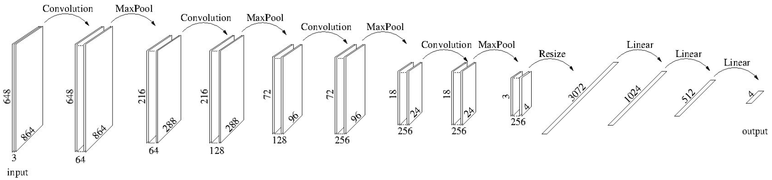
Maxpool filter





Example of Deep Learning Neural Network

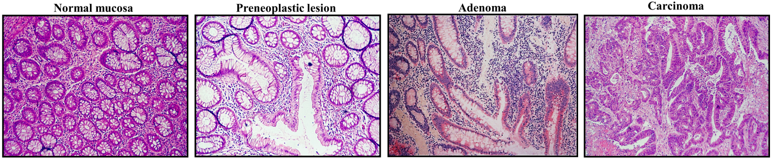
A typical network structure (alexnet):



Classification accuracy:

Training	Exact match	Nearest match
Short	$93.79 \pm 0.76\%$	$99.85 \pm 0.11\%$
Long	$95.10 \pm 0.19\%$	$99.84 \pm 0.05\%$

Optical images with Leica IRBE, CCD Rising Tech, 20X magnification.
200 patients (1998-2008).



Data	Normal	Preneo	Adenoma	Cancer	Total
Train	1616	1628	2736	2968	8048
Validate	200	204	344	256	1004
Test	200	204	340	256	1000

Deep Learning uses the Cross Entropy Loss:

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

$L(x)$: loss for image x, s the score function.

More generally:

- **Amari loss**: $I(x,w) = -\log(p(y|x,w))$
- **Information loss**: $IL(x,w) = \mathbb{E}_{y\sim p}[-\log(p(y|x,w))]$, where

$$IL(x,w) = \mathbb{E}_{y\sim p}[-\log(p(y|x,w))] = -\sum_i p_i(y|x,w)\log(p_i(y|x,w)$$

- **Empirical loss**: $L(x,w) = \mathbb{E}_{y\sim q}[-\log(p(y|x,w))]$

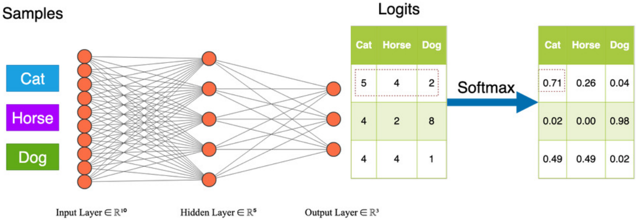
$$L(x,w) = \mathbb{E}_{y\sim q}[-\log(p(y|x,w))] = -\sum_i q_i(y|x,w)\log(p_i(y|x,w))$$

In Deep Learning $p(y|x,w)$ is obtained (usually) via Softmax:

$$p(y|x,w) = (p_i(y|x,w)) = \left(\frac{e^{s_i(x)}}{\sum_j e^{s_j(x)}}\right)$$

Softmax and Cross Entropy Loss/1

Softmax



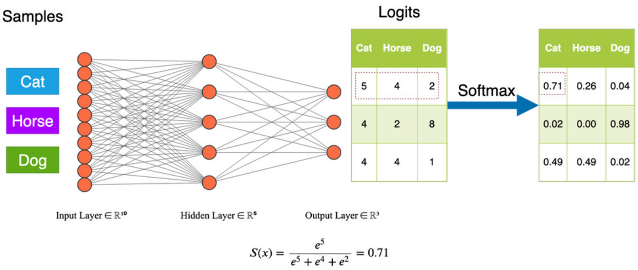
$$S(x) = \frac{e^5}{e^5 + e^4 + e^2} = 0.71$$

Hence:

$$p(\mathrm{cat}) = \left(\frac{e^5}{e^5+e^4+e^2}, \frac{e^4}{e^5+e^4+e^2}, \frac{e^2}{e^5+e^4+e^2}\right)$$

$$= (0.71, 0.26, 0.04)$$

$Loss(\mathrm{cat}) = -\log(p(cat)) = 0.34$ choose correct label (here "cat")!



Loss of one image:
$L(x) = -\log[e^{s_{\mathrm{label}(x)}}(x)/(e^{s_1(x)} + \cdots + e^{s_N(x)})]$: loss for image x.
Total loss: sum over all images

$$\begin{aligned} Loss &= -\log(p(cat)) - \log(p(horse)) - \log(p(dog)) = \\ &= -\log(0.71) - \log(0.002) - \log(0.02) = 0.34 + 6 + 3.91 = 10.2 \end{aligned}$$

KL Divergence

Kullback-Leibler divergence measures the "difference" between the correct distribution $q(x)$ and the empirical one $p(x)$:

$$KL(p(x)\|q(x)) = \mathbf{E}_q[p] = \sum q_i(x)\log(p_i(x)) =$$

$$= q_1(x)\log(p_1(x)) + q_2(x)\log(p_2(x)) + \cdots + q_n(x)\log(p_n(x))$$

Previous example:

$$x = \mathrm{cat} \longrightarrow p(x) = (0.71, 0.26, 0.04), \qquad q(x) = (1, 0, 0)$$

$$KL(p(x)\|q(x)) = 1 \cdot \log(0.71) + 0 \cdot \log(0.002) + 0 \cdot \log(0.02) = 0.34$$

This is the loss function (for image="cat")!

Loss and KL Divergence

The loss function is the Kullback-Leibler divergence up to a constant:

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

$q(y|x)$: true distribution (ex.: mass density distribution)
$p(y|x, w)$: empirical distribution

Notice: $L(x, w) = \mathrm{KL}(q(y|x) \| p(y|x, w))$,
when $q(y, x)$ is the mass density distribution i.e.
$q(y|x) = (0, 0, \ldots, 1, \ldots, 0, 0)$

3. OPTIMIZER

Principal optimizers:

- Stochastic Gradient
- SGD with Nesterov momentum
- ADAM (more popular)

Careful: read all options and make sure you understand them!

Regularization: add regularization to the loss to keep your parameters from getting too large (default is zero!).
Example: L^2 regularization, λ regularization parameter.

$$L(w) = \frac{1}{N} \sum_i L(x_i, w) + \lambda \sum_j |w_j|^2$$

Key point. The regularization will avoid the parameters to grow too large.

The Fisher Information matrix

Information Geometry: use of differential geometry to study probability and statistics.

Key idea: the parameters of a probability distribution are a manifold and possess a natural metric (Fisher).

$$F(x,w) = \mathbb{E}_{y\sim p}[\nabla_w \log p(y|x,w) \cdot (\nabla_w \log p(y|x,w))^T]$$

References

- Shun-ichi Amari, *Natural Gradient Works Efficiently in Learning*, 1998.
- F. Nielsen, *An Elementary Introduction to Information Geometry*, Entropy, 2020.
- J. Martens. *New insights and perspectives on the natural gradient method.* Journal of Machine Learning Research, 21(146):1-76, 2020.

Proposition. F is the covariance matrix of the gradient of the Amari loss.

Proof. In fact, the Amari loss is $l(x,w) = -\log p(y|x,w)$, its gradient is

$$\nabla_w l(x,w) = -\frac{\nabla_w p(y|x,w)}{p(y|x,w)}$$

Notice:

$$\mathbb{E}_{y\sim p}(\nabla_w l) = \sum p_i \frac{\nabla_w p_i}{p_i} =$$

$$= \sum_i \nabla_w p_i = \nabla_w (\sum_i p_i) = 0$$

The covariance matrix of $\nabla_w l(x,w)$ is (by definition):

$$\mathrm{Cov}(l) = \mathbb{E}_{y\sim p}[(\nabla_w l - \mathbb{E}_{y\sim p}(\nabla_w l))^t(\nabla_w l - \mathbb{E}_{y\sim p}(\nabla_w l))] =$$

$$= \mathbb{E}_{y\sim p}[(\nabla_w l)^t(\nabla_w l)] = F(x,w)$$

Hessian Geometry

Proposition. $F = \mathbb{E}_{y\sim p}[\mathbb{H}(l)]$, $l = -\log p(y|x,w)$.

Proof. In fact (write $p = p(y|x,w)$):

$$\mathbb{H}[l] = -\mathrm{Jac}\left[\frac{\nabla_w p}{p}\right] = -\left[\mathbb{H}(p)\cdot p + \nabla_w p \cdot \nabla_w p\right]\frac{1}{p^2}$$

Take the expected value:

$$\mathbb{E}_{y\sim p}[\mathbb{H}[l]] = -\sum_i p_i \frac{\mathbb{H}(p_i)}{p_i} + \mathbb{E}_{y\sim p}\left[\frac{\nabla_w p}{p}\cdot\frac{\nabla_w p}{p}\right] = F$$

where $\sum_i \mathbb{H}(p_i) = \mathbb{H}(\sum_i p_i) = 0$.

The Fisher matrix

The Fisher matrix for a minibatch of ONE sample image:

$$F(x,w) \quad = \mathbb{E}_{y\sim p}[\nabla_w \log p(y|x,w) \cdot (\nabla_w \log p(y|x,w))^T]$$

Key Facts:

$$\mathrm{KL}(p(y|x,w+\delta w)||p(y|x,w)) \quad \cong \frac{1}{2}(\delta w)^T F(x,w)(\delta w) + \mathcal{O}(||\delta w||^3)$$

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

$$\mathrm{rank}(F) < \text{number of classes}$$

The metric is neither Riemannian nor subriemannian.
Not constant rank either!

Data matrix

Idea. Treat weights w and images x on the same ground:

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

Key Facts:

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

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

For the Fisher the difference in size and rank is larger!

Data manifold

Result (F.-Grementieri, 2021). 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.

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

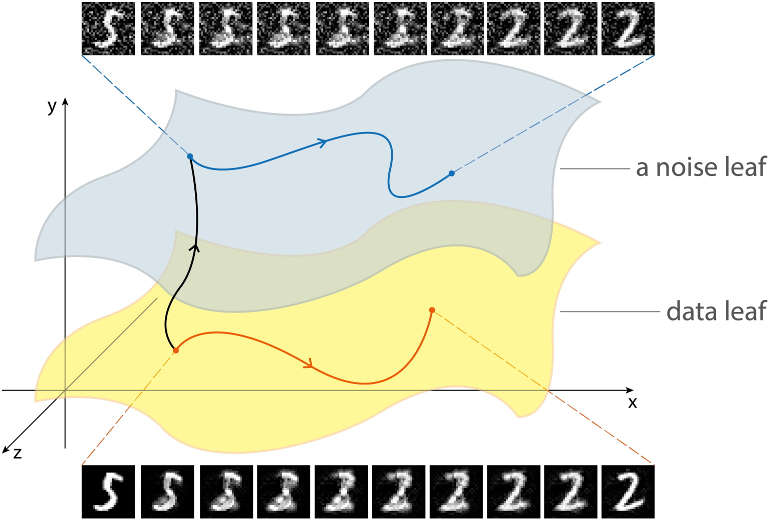
1. At each point in the dataset in $\mathbb{R}^n$, $\ker G(x,w)^{\perp}$ is tangent to a submanifold (**data leaf**) of dimension $\mathrm{rank}\ G(x,w) < C$
2. G defines a foliation on $\mathbb{R}^n$ of rank at most $C-1$ (**Frobenius Thm**).

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).

Data manifold



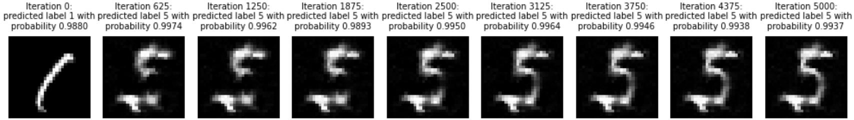
Facts

- 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.
- Not all points on the data leaf are in the data set, but they represent *symbols*.

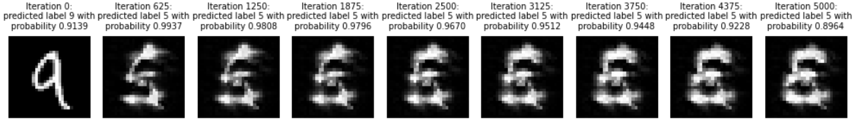
Moving on the data leaf: MNIST

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

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

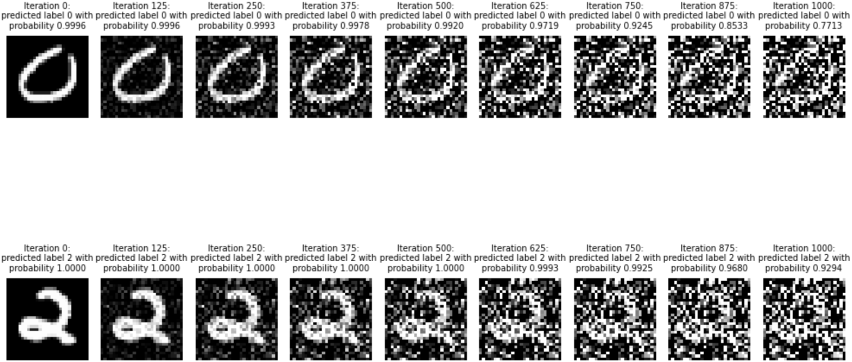


We can connect a digit from MNIST to a symbol **not** in MNIST moving on the **data leaf**:



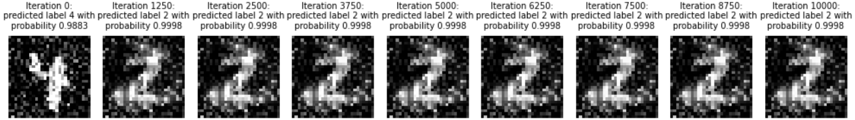
Moving away from the data leaf: MNIST

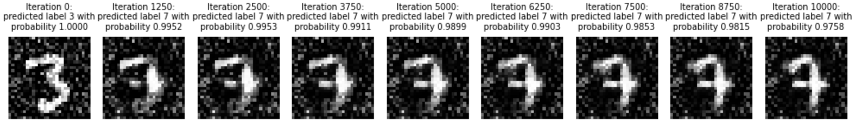
When moving **away** from a given data leaf, noise is added, but the accuracy is high.



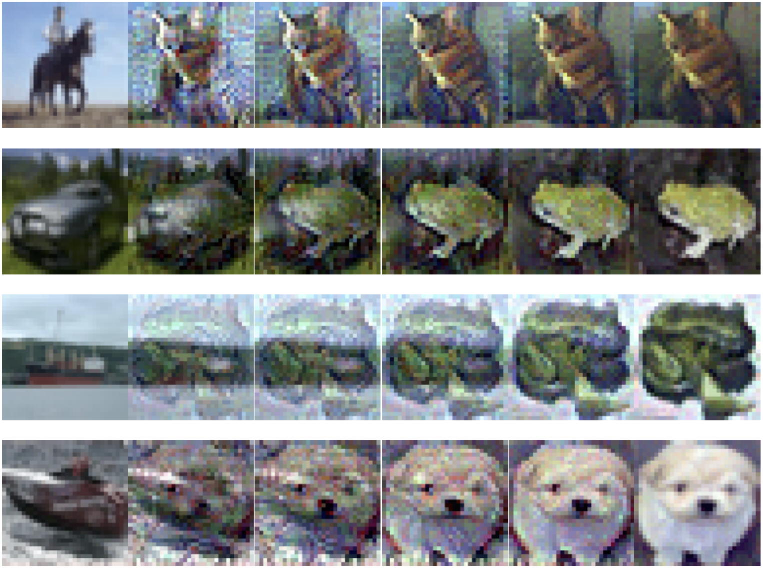
Moving on a noisy leaf: MNIST

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





Moving on the data manifold: CIFAR10



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 J. Imag. Sciences, https://arxiv.org/abs/2104.13289, 2021.
- Fioresi R., Tron E.
  Cartan moving frames and the data manifolds 2024, Information Geometry, https://arxiv.org/abs/2409.12057

Appendix: Detour on conjugate connections

Conjugate connections

Consider a statistical manifold M with the Fisher information metric g.

> *Two connections are said conjugate if their covariant derivatives ∇, ∇^* are torsion-free and satisfy*
>
> $$Xg(Y,Z) = g(\nabla_X Y, Z) + g(Y, \nabla^*_X Z)$$
>
> *for all $X, Y, Z \in \mathfrak{X}(M)$.*

Given a connection ∇ on (M,g), there exists a unique conjugate connection. The Levi-Civita connection is self-conjugate.

Conjugate connections and divergence

Given a divergence measure D on probability distributions parametrized by the points of M, we define a pair of conjugate connections by specifying their Christoffel symbols, thanks to the asymmetry of D.

$$\Gamma^i_{jk} = -g^{ih}\partial_{jk}\partial'_i D(\theta,\theta')\Big|_{\theta=\theta'}$$

$$\Gamma^i_{jk}{}^* = -g^{ih}\partial'_{jk}\partial_i D(\theta,\theta')\Big|_{\theta=\theta'} .$$

By interpolation we actually produce a family of pairs of conjugate connections.

The properties of conjugate connections are one of the main subjects of study of information geometry.

Courtesy of Prof. S. Mongodi