Study on Activation Patterns

Lucas Swiniarski lucas.swiniarski@nyu.edu
Joan Bruna bruna@cims.nyu.edu
Kyunghyun Cho kyunghyun.cho@nyu.edu

Abstract

Given a neural network and an input image, we can define the Activation pattern of this image as a $0-1$ vector representing which activations of the neural network are activated. This study focus on understanding the so-called "activation pattern" of an image. We look at general statistics of those activation patterns, how those statistics are affected by various normalization techniques (BatchNorm, LayerNorm, etc.). We define a distance between images as the number of difference between their activation patterns (we call it Hamming distance of activation patterns), and study the meaningfulness of this metric, such as appearance of clusters with this metric or shortest path between two images. Finally, noting the the activation pattern definition uniquely define a linear region of a ReLU neural network, we look at counting the number of feasible linear regions in a neural network as well as understanding the complexity of a linear region (polytope).

Contents

1 Introduction

Let's define a neural network with p layers, each layers having n_i hidden units. x input with dimension n_0.

$$x \in \mathbb{R}^{n_0} \to f_1(x) \in \mathbb{R}^{n_1} \to f_2(x) \in \mathbb{R}^{n_2} \cdots \to f_p(x) \in \mathbb{R}^{n_p} \tag{1}$$

$$f_j(x) = \rho(A^j f_{j-1}(x) + b^j) \tag{2}$$

$$A^j \in \mathbb{R}^{n_j \times n_{j-1}}, b^j \in \mathbb{R}^{n_j} \tag{3}$$

$$\rho(x) = \max(0, x) \text{ ReLU function.} \tag{4}$$

$$a_j(x) = 1_{f_j(x)} \in \{0, 1\}^{n_j} \tag{5}$$

We can see $a_j(x)$ as an approximation of a given layer, easier to study. The goals we set from the beginning are :

- How much of an approximation $a_j(x)$ is compared to $f_j(x)$?
- What can we say about the probability to activate at each layer, that is :

$$P_X((f_j(x))_i > 0) \text{ the probability of activation } i \text{ of layer } j \text{ to activate.} \tag{6}$$

  Furthermore, we can look at a given layer j, at the distribution of probability of activation of all activations i. What is the effect of learning on those.
- Use the approximation $a_j(x)$ to demonstrate the behaviours at hand.
- Assess the usefulness of a layer with a distance between the distributions of activation patterns, before and after the layer.

We can also declare a new distance between activation patterns : the hamming distance between $a_j(x_1)$ and $a_j(x_2)$, x_1, x_2 being two different input.

$$d_{Ham,j}(x_1, x_2) = \|a_j(x_1) - a_j(x_2)\|_1 \text{ the hamming distance at layer } j \text{ between two images.} \tag{7}$$

We can then ask :

- Does this distance has some semantic meaning ?
- Can we compare it to L_2 distance between $f_j(x_1)$ and $f_j(x_2)$?
- What can we say about the shortest path in the sense of this distance between two images ?
- Can we use this distance as an indicator of how useful is a layer. If we define :

$$M_{dist,k} = (d_{Ham,k}(x_i, x_j))_{i,j} \text{ matrix of hamming distance on the training set at layer } k. \tag{8}$$

  Can we compare $M_{dist,k-1}$ and $M_{dist,k}$ in a meaning-full way such that it correlate with the effect of layer k on the final accuracy ?

2 Exploration

All the exploration is done on Cifar10 with fully connected neural networks, or AlexNet for convnets.

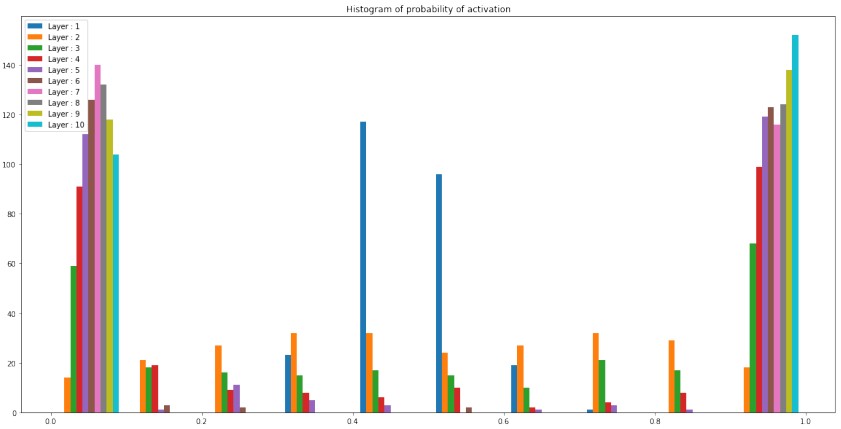


Figure 1: Probability of activation - FC 10 layers, 256 hidden units per layers.

2.1 Activation's probability

Let's look at the probability of activations of a 10 layer fully connected neural network. The experiment is done by initializing the network, forwarding all the training set and looking at each activation, counting the number of time the activation is none zero compared to zero.

Observations :

- First layer all probability of activation is close to 0.5 : Most activations activate half the time.
- Second layer, the probability of activation is close to uniform.
- Starting after the 3rd layer, we see that the probability of activation is almost a Bernoulli(0, 1, p=0.5) : Either an activation always activate or never activate. It seems that this saturation behaviour cause the network to be hard to train.
- This network is do not train, the training loss is not minimized, maybe some tuning would improve it though. If we add BatchNormalization between each layer however, the probability to activate at each layer is close to 0.5 and we can train the network. Maybe when the probability to activate is saturated - meaning all have probability 0 or 1 - is an indicator to the difficulty to train a network.

2.2 Hamming Distance and the effect of learning

In this experiment, we train a 3 layer fully connected neural network with 256 hidden units at each layer. At each epoch, we select 10000 random pairs of images, and look at the distribution of $d_{Ham,k}(x_i, x_j)$ for all 3 layers.

Observations :

- Before training, there is a big difference in distribution between layers : This is correlated to the distribution of probability of activation. If an activation i is always 1 or always 0, then $(a_j(x))_i == (a_j(y))_i, \forall x, y$ images and this unit is irrelevant in the distance count.
- It seems that during training, all activations learn to have a probability of activation of 0.5. Thus initializing so that each probability of activation is 0.5 from the beginning should help the learning process - This is what Batch Norm does in effect.
- As learning goes, we see the distance between two elements on each layer being very close to $256/2$: Do we end up having random bernouilli vectors or no ? Do we have constellations ?

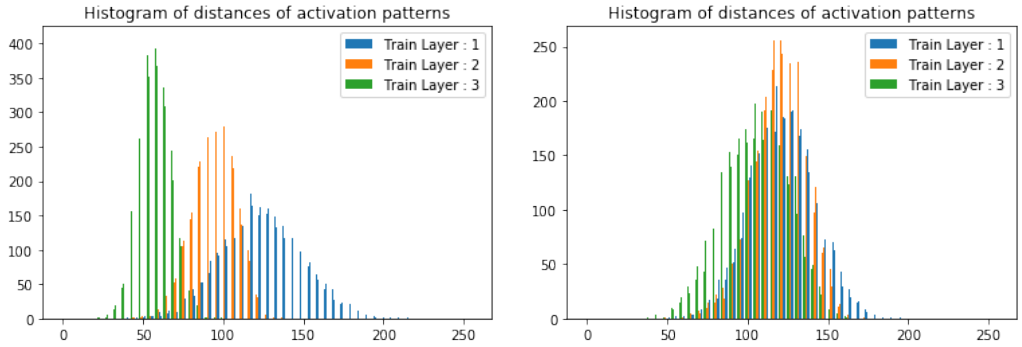


Figure 2: Histogram of Hamming distance of the 3 layers, left before training, right after 16 epochs (54 % accuracy)

2.3 Are there patterns appearing ?

2.3.1 are the vectors $a_j(x)$ randomly distributed with a Bernoulli $\mathcal{B}(p = 0.5)$?

The network is a fully connected network with one hidden layer and 256 hidden units.

Test statistic : Mean distance between elements. We look at different epoch of training at : given 10000 pairs of images, look at the empirical mean over 10000 elements of $d(a_1(x_{P(k,1)}, x_{P(k,2)}))$ where $P(k, i)$ is the i-th ($i \in \{1, 2\}$) element of the k-th pair.

Now we draw 10000 pairs of random Bernoulli vectors of size 256, and look at the mean distance. We repeat the draw 1000 times. We then look at the p-value : The probability given the underlying distributing is random Bernoulli vectors, to see the mean distance reported.

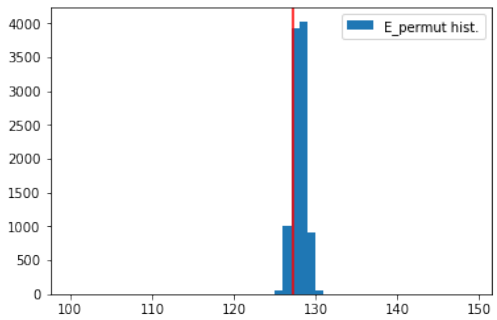


Figure 3: Red : Mean distance at 0 epochs, p-value : 18 %

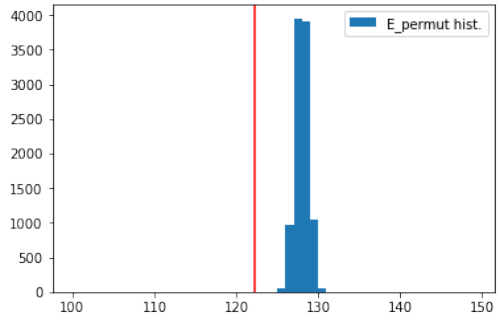


Figure 4: Red : Mean distance after 5 epochs, p-value : 0 %

Figure 5: Distribution of mean distance following random Bernoulli.

Observations :

- At the beginning, we are close to random Bernoulli behaviours.
- After 5 epochs, the mean distance is lower enough to be statistically significantly not random Bernoulli behaviour. During learning, we learn to deviate from random behaviour.

2.3.2 Do we see constellations ?

We define a constellation as a group of images whose activation patterns are close together, and far from others. As this definition is vague let's something usable :

We define a constellation as a group of images whose maximum distance between two elements of the same cluster is below a threshold, and the minimum distance between an element of the cluster

and any other element is above this threshold. In order to use this definition, we use hierarchical clustering, and create clusters out of 16000 elements, then count the number of unique clusters at a certain threshold. In order to not be dependent of a threshold we look at the curve made by moving the threshold from 1 to n_j. The experiment is done with the same network as before

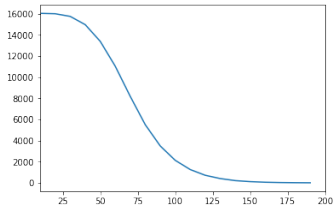


Figure 6: Epoch 0

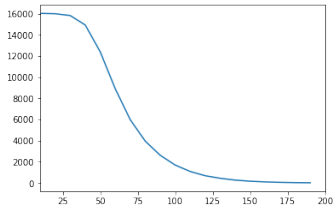


Figure 7: Epoch 6

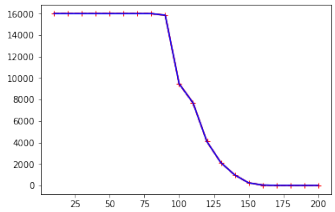


Figure 8: Random Bernouilli vectors

Observations :

- On the metric : In order to have a "constellation score" that we could keep track of, using the integral of the above curve should be better, the lower the integral the more constellations are created.
- On the curve itself : We would actually want a certain number of clusters to appear quickly and then don't change much as the threshold grow for a while. This way it would mean that clusters are far away from each other and would have semantic meaning.

2.4 Semantic Meaning of Hamming Distance

Let's train a one hidden layer fully connected network with 256 hidden units and look at the nearest neighbours with Hamming Distance.

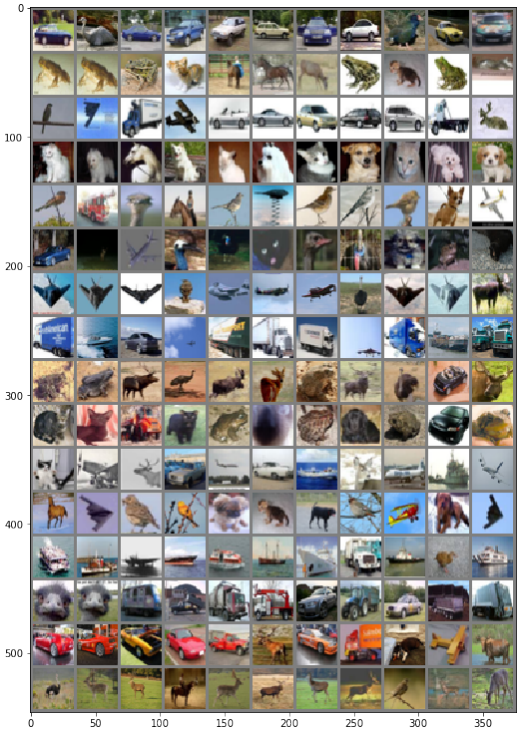


Figure 9: First column is the input image, then on the same row are 10 ranked nearest neighbours.

Observations :

- While there clearly are some weird nearest neighbours, overall there is some meaning in this distance.

3 Normalization techniques and effects on activations' statistics

Normalizing the distributions within a neural network is useful because if your distribution's variance explode or collapse after a couple of layers, the end signal is not trainable ie 10 fully connected layers with no normalization doesn't train.

- Xavier Initialization : We want to propagate the signal without changing its scale.
  If we set $Var(W) = \frac{1}{n_{in}}$ then $y = \sum w_i x_i$ has same variance as x under independence assumption.
  During back-prop we need $Var(W) = \frac{1}{n_{out}}$. As we can't have both most of the time, last version of Xavier initialization is $(W) = \frac{2}{n_{in}+n_{out}}$.
  Drawback :
  We don't account for the loss of variance due to the non-linearity, and we see some collapse (PyTorch initialize Linear layers like this) and stacking linear layers ends up making a highly untrainable network. Also it's worth noting that it's very difficult to see a training error going to 0 in that setting : Cifar10 usually stick to 1.5 training loss and validation loss.
- Batch Normalization : For each activation, keep track of its running mean and standard deviation (the rate of update is an hyper-parameter) and normalize each activation with the running mean/standard deviation.
- Weight Normalization : Reparametrization of the weights : $W \rightarrow \frac{V}{\|V\|_2} G$ and special initialization on the first batch of data.
- Layer Normalization : Given all the activations of a layer $a_1, ..., a_n$ compute the mean and standard deviation and normalize $\frac{a_1-\mu}{\sigma}, ..., \frac{a_1-\mu}{\sigma}$: Normalization of the activations of a single input, instead of normalizing each activation with regard to the batch(s).
- Self Regularizing Neural Networks - SELU :

$$selu(x) = \lambda \begin{cases} x & x > 0 \\ \alpha \exp(x) - \alpha & x \leq 0 \end{cases} \tag{9}$$

  You can choose compute $\alpha_{01}, \lambda_{01}$ so that the outputted distribution has 0 mean and unit deviation.

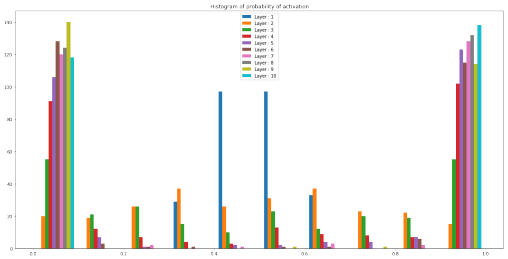


Figure 10: Epoch 0

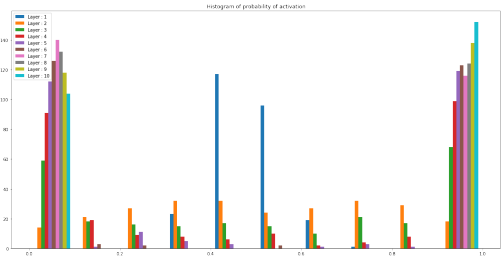


Figure 11: Epoch 5 : 10 % accuracy

Figure 12: Xavier Initialization

Commentaries :

- Layer Normalization / Weight Normalization are easy to train, and as a concequence might lead to overfitting. Training loss on both methods end up close to 0.5 with the same accuracy or poorer than other methods. Train accuracy generally is close to 1.5.
- Batch Norm seems too restrictive as the activation probability is restricted to be close to 0.5.

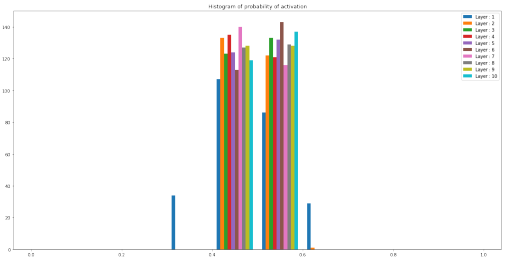


Figure 13: Epoch 0

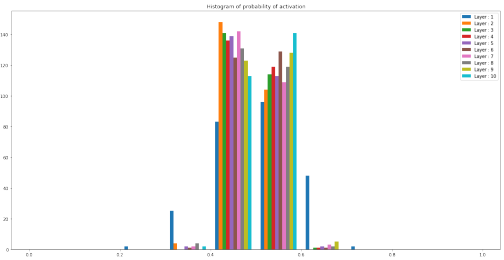


Figure 14: Epoch 13 : 52 % accuracy

Figure 15: Batch Normalization

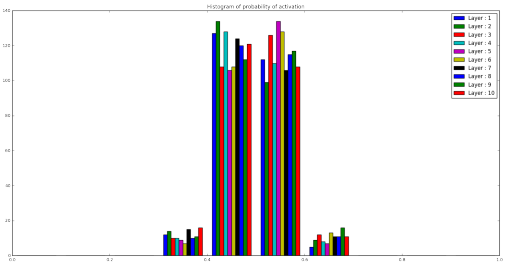


Figure 16: Epoch 0

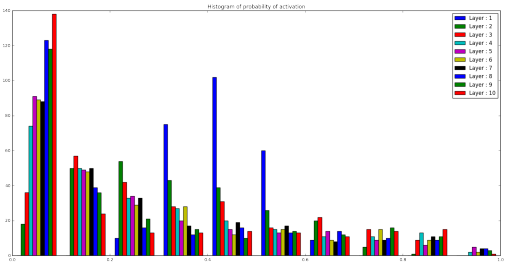


Figure 17: Epoch 20 : 53 % accuracy

Figure 18: Weight Normalization

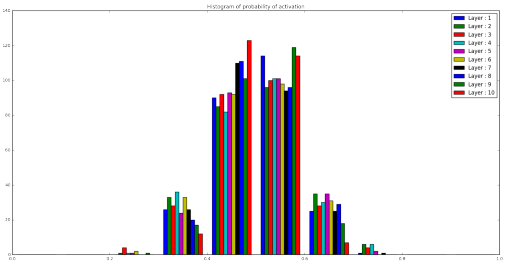


Figure 19: Epoch 0

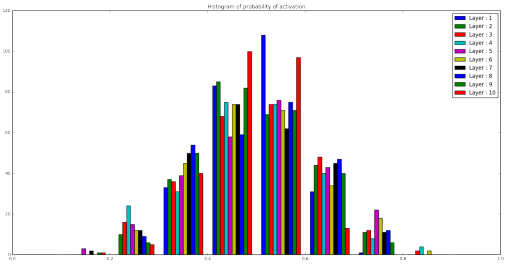


Figure 20: Epoch 20 : 41 % accuracy

Figure 21: Layer Normalization

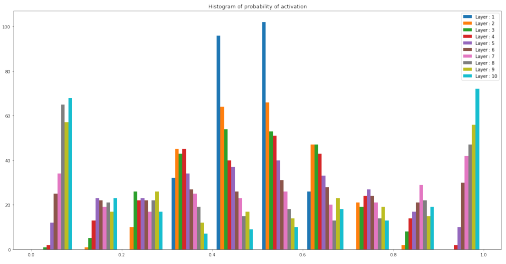


Figure 22: Epoch 0

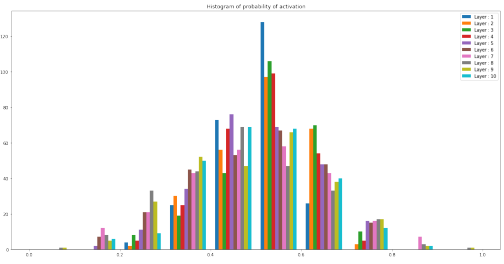


Figure 23: Epoch 20 : 50 % accuracy

Figure 24: Self-Regularizing Neural Networks

4 Modeling

4.1 Sampling Hyper planes

What does the hyper planes we sample following the initialization of a neural network looks like ? At layer j, one activation is initialized as :

$$W \in \mathbb{R}^{n_j} \sim \mathcal{U}([\frac{-1}{\sqrt{n_j}}, \frac{1}{\sqrt{n_j}}]) \tag{10}$$

$$b \in \mathcal{R} \sim \mathcal{U}([\frac{-1}{\sqrt{n_j}}, \frac{1}{\sqrt{n_j}}]) \tag{11}$$

$$\mathcal{H} = \{X \in \mathbb{R}^{n_j} | \langle X, W \rangle + b = 0\} \tag{12}$$

$$d(\mathcal{H}, 0) = \min_{x \in \mathcal{H}} \|x\|_2 = \frac{b}{\|W\|_2} \tag{13}$$

Let's look at the shape of the distribution of $d(\mathcal{H}, 0)$:

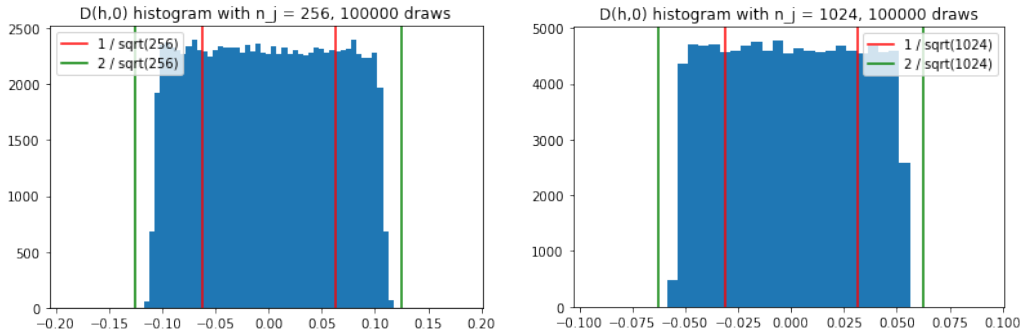


Figure 25: Distribution of $d(\mathcal{H}, 0)$, left $n_j = 256$, right $n_j = 1024$

Observations :

- All hyper planes drawn with this technique end up very close being very close to the origin of the vector space.
- Drawing hyper planes near 0 is a good strategy if distributions are centered, otherwise there are more elements on one side of the hyper-plane than on the other.
- Note that what would actually like to follow is the median not the mean, we want to cut our distribution at the median point in order for the hyper-plane to have equal opportunity for a point to be above or below.

4.2 Modelling of neural networks first layer

Let's assume our input is i.i.d., which is what we tend to have when we whiten the input space.

$$X \in \mathbb{R}^{n_0} \sim \mathcal{N}(0, I) \tag{14}$$

$$W \in \mathbb{R}^{n_0} \sim \mathcal{U}([\frac{-1}{\sqrt{n_0}}, \frac{1}{\sqrt{n_0}}]) \tag{15}$$

$$b \in \mathcal{R} \sim \mathcal{U}([\frac{-1}{\sqrt{n_0}}, \frac{1}{\sqrt{n_0}}]) \tag{16}$$

$$\tag{17}$$

What does $A = \rho(\langle W, X \rangle + b)$ distribution look like ? See fig. 4.3.1

Observations :

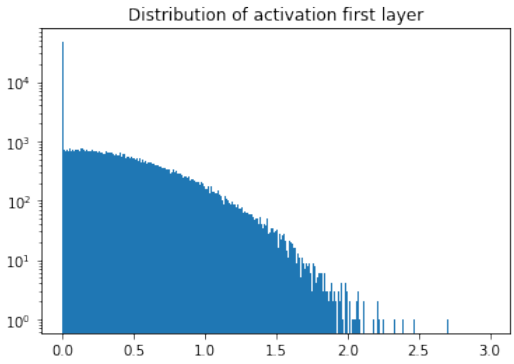


Figure 26: Distribution of $\rho(\langle W, X\rangle + b)$ sampling X. ($\mu = 0.25, \sigma = 0.35$)

- There is a log scale because in 0 there are as many elements as the rest of the space, thus we would not see anything without the log scale.
- The shape of the distribution when we are not in 0 looks like a centered gaussian with strictly positive elements only : a folded normal distribution.

We can model : $A = B|Y|$ with $B \sim \mathcal{B}(0.5), Y \sim \mathcal{N}(\mu, \sigma^2)$. We can assume $\mu = 0$, what is sigma here then ? We know $(A|A > 0)|(B|Y||B = 1) = |Y|$ and :

$$\mu_{|Y|} = \sigma\sqrt{\frac{2}{\pi}}\exp(-\frac{\mu^2}{2\sigma^2}) + \mu(1 - 2\Phi(-\frac{\mu}{\sigma})) \tag{18}$$

$$\sigma_{|Y|} = \mu^2 + \sigma^2 - \mu^2_{|Y|} \tag{19}$$

With $\mu = 0$:

$$\mu_{|Y|} = \sigma\sqrt{\frac{2}{\pi}} \tag{20}$$

$$\sigma_{|Y|} = \sigma^2 - \mu^2_{|Y|} \tag{21}$$

From an experiment we have : $\mu_{|Y|} = 0.4685, \sigma_{|Y|} = 0.3489$ so $\sigma = 0.5872$ from first equation and $\sigma = 0.5842$ from second equation.

A last model we can do is look at the statistics of $|Y| * C \simeq Y$ where $C \sim \mathcal{B}(p = 0.5, -1, 1)$ it yields mean 0 and standard deviation 0.5842.

Observations :

- The simplification we proposed at the beginning is equal to saying that activations follows Bernoulli distribution instead of Bernoulli multiplied by a folded normal distribution.
- It's interesting to see that overall the standard deviation of the activation is lower than the standard deviation of X : It would be interesting to have an equation of the form $\sigma_{Y/A} = f(\sigma_X, \sigma_W, b)$.
- There is a link with SVM in the sense that we look for hyper-planes and do a very harsh classification $0 - 1$ classification afterwards, maybe we could do something more forgiving - as an SVM.

4.3 $0 - 1$ activations model

Let's put ourselves in the same setting as the last section : We have activations that have a distribution of the type $B|Y|$ with $B \sim \mathcal{B}(0.5), Y \sim \mathcal{N}(0, \sigma^2)$.

4.3.1 Batch Normalization visualization

First, we see that with this type of distribution, it is clearly not a normalized distribution : 4.3.1.

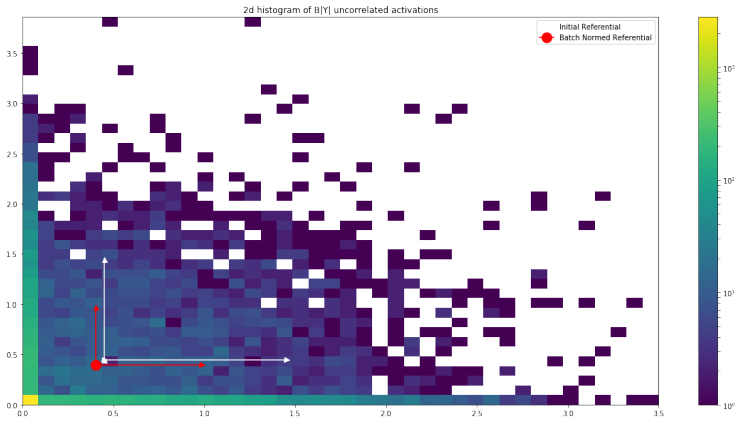


Figure 27: 2 activations of the first layer : in red we see the new referential batch norm gives.

In this setting, batch normalization is actually changing the Cartesian system so that distribution are normalized again in this new referential.

But we can actually integrate this part in our network if we want to, given we know the entry distribution is more or less Gaussian.

Technically, a Layer can learn to normalize its input as the normalization part is $\frac{x_i - \mu_i}{\sigma_i}$ then you have your new layer. During training it's not done if not explicitly enforced because of SGD/non convex minimization ... We can say that batch normalization is leveraging the prior information that the best way a layer (or dictionary of hyper-planes) can classify/extract features is to cut a ball.

4.3.2 Can we approximate an activation value by $0-1$?

By approximation, we mean : what is the probability that approximating a layer $f_j(x)$ by $a_j(x)$ it's $0-1$ values, a new layer W, b will classify on the same side of the hyperplane $\langle f_j(x), w\rangle + b$ and $\langle a_j(x), w\rangle + b$. So :

$$P_W((\langle f_j(x), w\rangle + b) \times (\langle a_j(x), w\rangle + b) \geq 0) \tag{22}$$

First, simulating the result : Always very close to 79%.

Based on this simulation, if you approximate the first layer, the second layer will have 20% of activations that are wrongly placed, and so on. Might not be a good idea after all...

4.3.3 $0-1$ activations and future layer - Work in progress

We can use (normally) the $0-1$ simplification to demonstrate that we indeed have this uniform probability of activation in the second layer. Proof in work for now.

Given :

$$x \in \mathcal{R}^n, x \sim \mathcal{B}(0.5) \tag{23}$$

$$w \in \mathcal{R}^n, w, w_0 \sim \mathcal{U}([-m, m]), m = \frac{1}{\sqrt{n}} \tag{24}$$

$$P_W(\langle w, x\rangle + w_0 > 0) = P_X(\langle d, x\rangle + \frac{u}{0.577\sqrt{n}} > 0) \tag{25}$$

$$d \in \mathcal{U}(\mathcal{B}(0, 1)) \tag{26}$$

$$u \in \mathcal{U}([-1, 1]) \tag{27}$$

$$\|w\|_2 \sim 0.577 \text{ with very low variance.} \tag{28}$$

$$\tag{29}$$

We have :

$$P(P_a = 0/1) = \frac{1}{2^{n+1}} \tag{30}$$

$$P(P_A = \frac{k}{2^n}) = \frac{1}{2^n} \tag{31}$$

True for $n = 2, 3$, what we can try :

- Proof by recurrence : It might be doable to understand what happens when we add only one dimension ?
- Maybe we can prove that $P(P_A = \frac{k}{2^n}) = P(P_A = \frac{k+1}{2^n})$ by some rotation invariance ? Basically each $k/2^n$ has its own cone of hyperplane ?

4.4 Variance of the first layer - Work in progress

We are in the case :

$$X_k \in \mathbb{R}^{n_0} \sim \mathcal{N}(0, I) \tag{32}$$

$$X \in \mathbb{R}^{n \times n_0} \tag{33}$$

$$W \in \mathbb{R}^{n_0} \sim \mathcal{U}([\frac{-1}{\sqrt{n_0}}, \frac{1}{\sqrt{n_0}}]) \tag{34}$$

$$b \in \mathcal{R} \sim \mathcal{U}([\frac{-1}{\sqrt{n_0}}, \frac{1}{\sqrt{n_0}}]) \tag{35}$$

$$\tag{36}$$

We have :

$$\begin{aligned} {}_X(\langle w, x\rangle + b) =& {}_X\ (\sum_{i=1}^{n_0} w_i x_i) && (37)\\ =& \sum_{i,j \in [1,n_0]^2} w_i w_j {}_X(x_i, x_j) && (38)\\ =& \frac{1}{n} W^T X^T X W && (39)\\ =& \frac{1}{n} \|XW\|_2^2 && (40) \end{aligned}$$

So if we have $\|XW\|_2^2 = f(\sigma_X, \sigma_W)$ then we have the variance of the next layer, so we can initialize the layer in such a way that it's batch normalized already.

From Self-Normalizing Neural Networks we have, with independent assumption :

$$_X(\langle w, x\rangle) = (x)\sum_i w_i^2 \tag{41}$$

5 Shortest path

Definition Given two input images x_1, x_2, the vectors $(a_1(x_1), \cdots, a_p(x_1)), (a_1(x_2), \cdots, a_p(x_2))$ is a representation of the linear region which classify x_1, x_2 because :

$$f_1(x) = a_1(x) \otimes (A^1 x + b^1) \tag{42}$$
$$f_2(x) = a_2(x) \otimes (A^2 f_1(x) + b^2) \tag{43}$$

And so on.

Notes

- Two different sets of activation patterns $a(x_1) = (a_1(x_1), \cdots, a_p(x_1))$ can yield in the end the same matrix, for now we treat this event as unlikely.
- All activations patterns might not be attainable - or at least not in the input space $[-1, 1]^N$.
- Given two images x_1, x_2 that have m differences in their activation patterns : $d_H(x_1, x_2) = \|a(x_1) - a(x_2)\|_0 = m$, there is $m!$ possible paths from x_1 to x_2.

5.1 First algorithm

Using the geodesic algorithm of [Olivier J. Hénaff et al], which is this algorithm :

$$E[\gamma] = \sum_{i=1}^{n-1} \|x_{i+1} - x_i\|_2^2 \tag{44}$$
$$E[f(\gamma)] = \sum_{i=1}^{n-1} \|f(x_{i+1}) - f(x_i)\|_2^2 \tag{45}$$
$$\tag{46}$$

Note that f in $E[f(\gamma)]$ can be any layer you want, or more than one layer with a weighting for each layer (a layer with twice as many activations would have more impact otherwise).

l_0 vs. l_2 Note that we need to minimize l_0 norm, which cannot be directly optimized because of 0 gradients, here l_2 norm is in the convex hull of l_0 norm. Note also that l_1 norm might be better as it's a tighter convex hull of the l_0 norm.

Results : Here we compute the hamming distance on the 3rd and last convolution. The network is AlexNet for cifar10 (3 convolutions / max pooling and one linear layer).

In this example we don't end up having the a cumulative hamming distance $d_{H,cum} = \sum_i d_H(x_{i+1}, x_i)$ equals to $d_H(x_1, x_{N+1})$ but we have the general behaviour of this algorithm : We end up pushing on a few pixels to change the activation pattern and satisfy our constraints.

Algorithm 1 Conditional geodesic computation

Require: f : continuous mapping
Require: x_0, x_N initial and final images
Require: N number of steps along geodesic path ($N = 10$ usually)
Require: λ gradient descent step size
Ensure: $\gamma = \{x_n; n = 0, ..., N\}$ minimizes $E[\gamma]$ conditionned on minimizing $E[f(\gamma)]$
  $x_n \leftarrow \frac{N-n}{N} x_0 + \frac{n}{N} x_N, n \in [0, ..., N]$
  minimize $E[f(\gamma)]$
  while γ has not converged **do**
    $d_r \leftarrow \nabla_\gamma E[f(\gamma)]$
    $d_p \leftarrow \nabla_\gamma E[\gamma]$
    $\hat{d_p} \leftarrow d_p - \frac{\langle d_r, d_p \rangle}{\|d_r\|_2^2} d_r$
    $\gamma \leftarrow \gamma - \lambda \hat{d_p}$
    minimize $E[f(\gamma)]$
  return γ

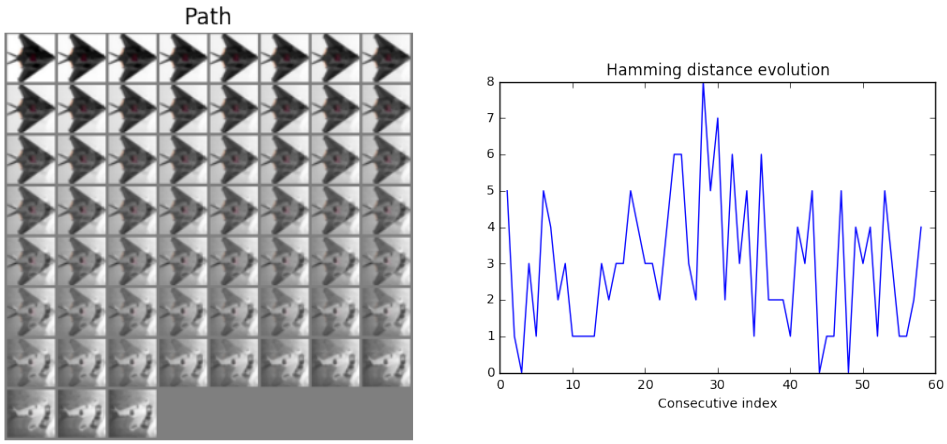


Figure 28: Interpolation path : $d_H(x_1, x_2) = 116$, cumulative hamming distance 174

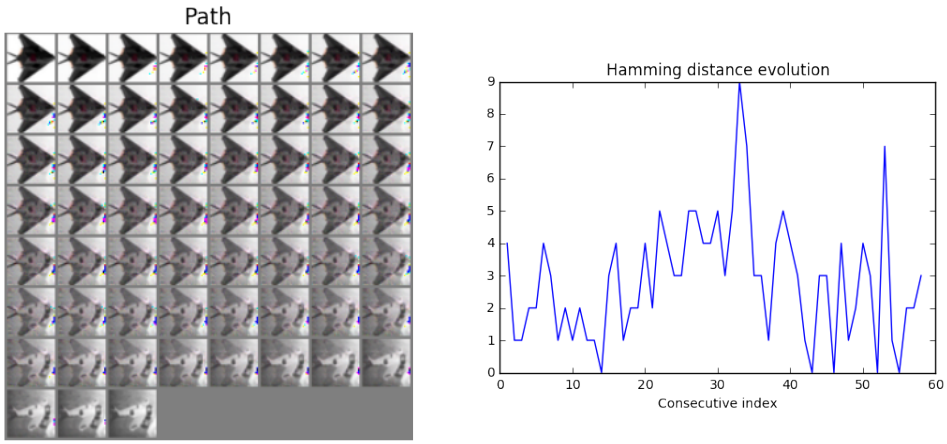


Figure 29: After training : $d_H(x_1, x_2) = 116$, cumulative hamming distance 164

5.2 Follow-up algorithms

Behaviours that we don't like with the first algorithm :

- Minimizing the l_2 norm is a tighter constraint than the l_0 norm, l_1 norm could be better.
- We are looking for a path with 1 hamming distance between two consecutive images of the path, here we tend to minimize the cumulative sum, without penalizing a big consecutive distance.
- This algorithm requires 2 optimization procedures : First $E[f(\gamma)]$ then $E[\gamma] s.t. E[f(\gamma)]$. Maybe we can speed this up with an optimization procedure like $\min_\gamma E[\gamma] + \alpha E[f(\gamma)]$. Also two optimization procedures means having to find a stopping criterion for the first one, and results are dependent on the stopping criterion a lot.
- Given two images at hamming distance d, if we try to find $d-1$ minimizing the cumulative distance with the algorithm above, we end up with a lot of images at distance 0 and some at a big distance - which is not the desired property.
- For now, it is not necessary to minimize the l_2 distance in image space, we want the shortest path in the activation space first.

In order to solve the problems stated above, the sub-task we need to solve is :

$$\text{Given : } x_0, x_1, d_H(x_0, x_1) = d \tag{47}$$

$$\text{Find : } z, d_H(x_0, z) + d_H(z, x_1) = d \text{ and approximately } d_H(z, x_1) = d_H(z, x_0) = \frac{d}{2} \tag{48}$$

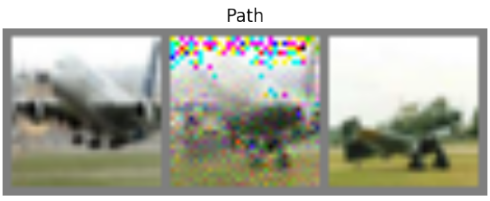
If we are able to find a suitable middle point, we can use dynamic programming to solve the more general problem of path solving.

5.2.1 Algorithm 1

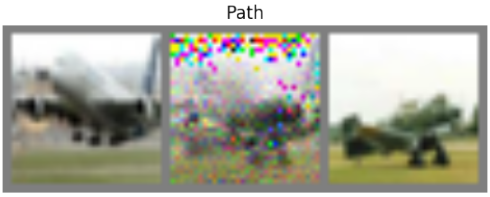
Let's find z solution of :

$$\min_z \sum_{\text{layer}l} \|f_l(z) - f_l(x_0)\|_2^2 + \|f_l(x_1) - f_l(z)\|_2^2 \tag{49}$$

l_1 **norm :** $d_H(x0, x1)$: 2596 $d_H(x0, z)$: 1210, $d_H(z, x1)$: 1460 (cumulative : 2670)



l_2 **norm :** $d_H(x0, x1)$: 2596 $d_H(x0, z)$: 1452, $d_H(z, x1)$: 1776 (cumulative : 3228)



Notes : The point found is roughly in the middle, but it is very strange to find that the l_2 norm works not that well, compared to the l_1 norm.

5.2.2 Algorithm 2

Let's find z solution of :

$$\min_z \sum_{\text{layer}l} |a_l(z) - a_l(x_0)| \otimes f_l(z) - f_l(x_0)\|_2^2 + \||a_l(z) - a_l(x_1)| \otimes f_l(x_1) - f_l(z)\|_2^2 \quad (50)$$

l_1 **norm :** $d_H(x0, x1)$: 2596 $d_H(x0, z)$: 1846, $d_H(z, x1)$: 1640 (cumulative : 3486)

l_2 **norm :** $d_H(x0, x1)$: 2596 $d_H(x0, z)$: 2072, $d_H(z, x1)$: 2326 (cumulative : 4398)

5.2.3 Algorithm 3 - work in progress

Let's solve the problem backward : Given 2 images, let's find an activation pattern that is in-between the activation pattern of x_0 and x_1 (There are $d_H(x_0, x_1)$ different such activation patterns). Given this activation pattern, let's try to find a z in the input space with this activation pattern. If we are able to, we just found a good candidate.

For now it clearly does not work - the hamming distance to the chosen activation pattern stay big. Issues might come from the fact that a randomly chosen activation pattern might not be reachable - there might not be an image yielding this particular activation pattern.

5.3 Conclusion

- There is always a big trouble to find the optimal path when we consider all layers. If we consider only a few layer - and deep layers - we tend to get good results. But when we consider all layers, and also the first one it does not work. This is also closely related to the hamming distance at hand : the deeper the layer, the less activations it has, the closer elements are (because less activations), the better the algorithms works.
- Observation : We do end up with an l_2 interpolation path in the activation space. Here I've taken a geodesic path and plotted the last layer (10 activations - 10 class prediction) values of activations throughout the path (10 elements on the path) :

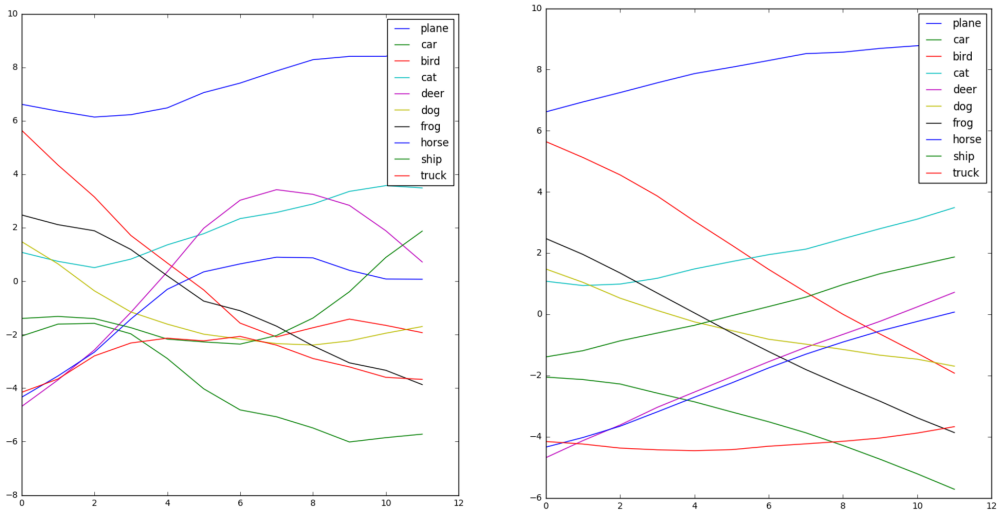


Figure 30: Plotting the last layer values throughout the l_2 initialization path (left) and activation l_2 minimization path (right)

- l_1 globally works better than l_2, we constantly have better paths.

6 Properties of Linear regions classifier

A neural network can be seen as locally linear. Given an activation and an image, we have $a = \langle x, W_{a,x} \rangle + b_{a,x}$ and we can have access to $W_{a,x}$ by forwarding x through the network up until this activation, and back-proping at this activation.

Empirical analysis show that the above technique to get $W_{a,x}$ is noisy. There is a way to get it without noise. For a linear neural network it looks like :

$$M^l \in \mathcal{M}_{n_{l-1},n_l}(\mathbb{R}) \text{ matrix of each linear layer.} \tag{51}$$

$$\text{Given activation } i \text{ at layer } L \text{ and } x \text{ with activation patterns } a^1(x), ..., a^l(x), a^j(x) \in \{0,1\}^{n_j} \tag{52}$$

$$\text{Take the row vector of : } M^{L-1} \text{ at row index } i, v_{l-1} \tag{53}$$

$$v_{l-1} \leftarrow v_{l-1} \otimes a^{l-1}(x) \tag{54}$$

$$v_{l-2} \leftarrow (M^l)^T v_{l-1} \text{ and so on until input space.} \tag{55}$$

Some properties to look at :

- $W_{a,x}$ is actually the size of an image, so we can plot it, what does it look like ?
- If we sample $z_1, \cdots, z_n \in \mathbb{R}^n$ Noise - or real images - in the input space, what is the rank of the matrix $(W_{a,z_1}, \cdots, W_{a,z_n})$?
- How much does $W_{a,x}$ differ from $W_{a,x+\epsilon}$, ϵ being a small noise ?

6.1 Plotting $W_{a,x}$

Notes We need to normalize the image vector used for classification so it can be seen as an image, otherwise elements have a low range.

Last layer (10 activations corresponding to 10 classes) :

classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

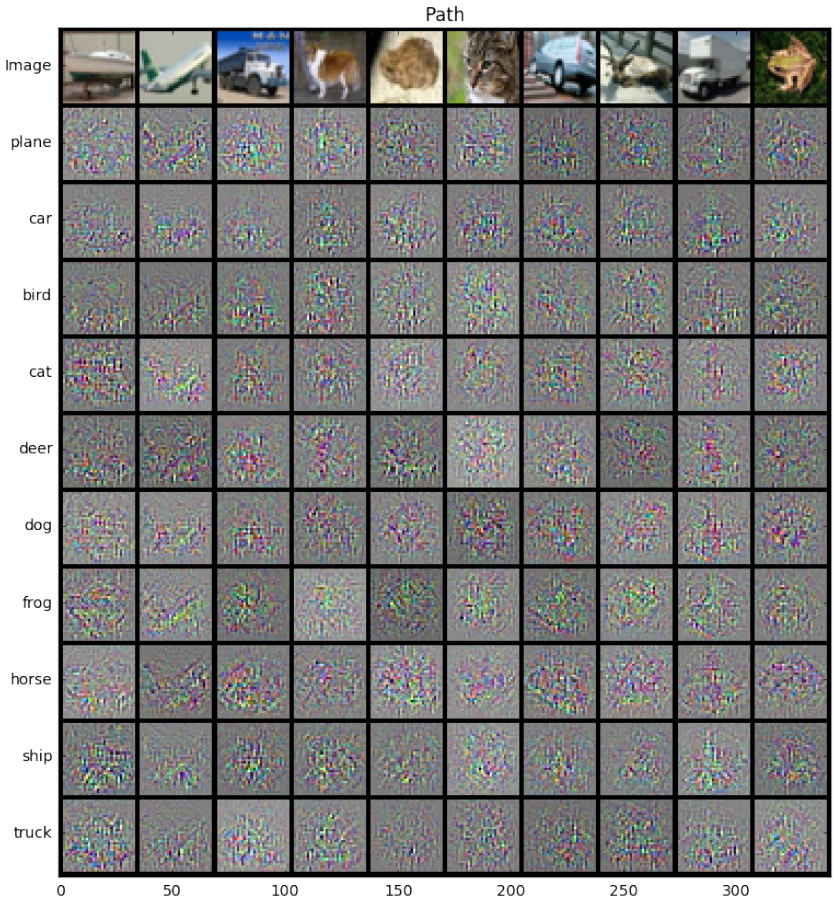


Figure 31: Last layer, ten images and their classification image for all 10 activations.

6.2 Rank of W_x vectors - Can we get an idea of the complexity ?

The first observation is that the .backward() command is noisy - if you forward a batch of same images and call .backward(), the gradient on the input differs slightly.

That being said, given an activation, which is made by $\langle x, W_x \rangle + b_x$, we can sample points and look at how much W_x differs : looking at the rank of the matrix made by $W_{x_1}, \cdots, W_{x_n}$.

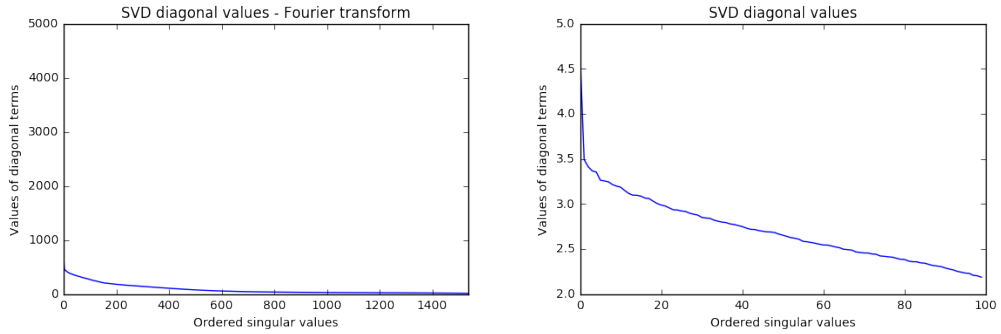


Figure 32: SVD on 50k image and fft (left) and 10k images without fft, first 100 singular values (right)

50000 **real images - the training set**

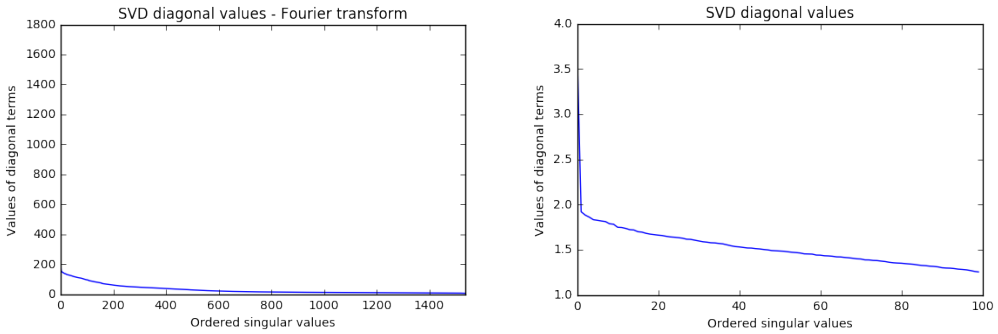


Figure 33: SVD on 50k image and fft (right) and 10k images without fft, first 100 singular values (left)

50000 **noise points drawn in** $\mathcal{U}([-1, 1])$

6.3 Orthogonal decomposition of classification image

Because of the small noise the classification hyperplanes (W_x) are subject to, it's possible that it inflate the rank of the matrix. What we can try to do instead is decompose an image classification hyperplane W_x onto the classification hyperplanes of the whole training set :

Given an activation a, compute for the whole training set $(x_1, ..., x_n)$ the classification hyperplanes $W_{x_1}, \cdots, W_{x_n}$. Then given an image in the validation set x decompose W_x onto $W_{x_1}, \cdots, W_{x_n}$ by doing recursively :

$$\text{Compute : } \forall i \in [1, n], \cos(W_{x_i}) \tag{56}$$

$$\text{Get the best projection : } j = \underset{i \in [1,n]}{|\cos(W_{x_i})|} \tag{57}$$

$$\text{Compute the remainder : } W_x \leftarrow W_x - \frac{\langle W_x, W_{x_j} \rangle}{\|W_{x_j}\|_2^2} W_{x_j} \tag{58}$$

Experiments : First image : image from the validation set, following images are the consecutive projections with W_{x_j} visualization below. The network predicts a plane here.

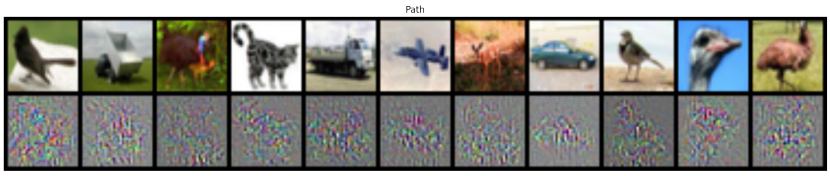


Figure 34: Bird activation (last layer).

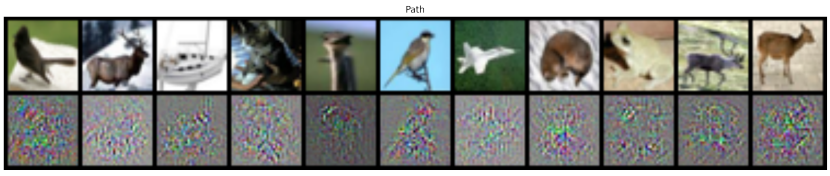


Figure 35: Plane activation (last layer).

Notes

- We decompose onto 20000 images of the training set, because lack of memory.
- The first 10 projections clearly do not approximate well W_x. Here I've plotted $\frac{\|W_{x,\text{step i}}\|_2}{\|W_{x,\text{step 0}}\|_2}$. It shows that the first 10 decomposition approximate 9% of the initial hyperplane. A test with 100 hyperplanes approximate 40% of the initial hyperplane.

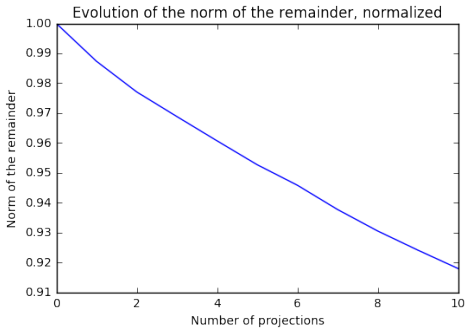


Figure 36: Bird activation remainder norm evolution

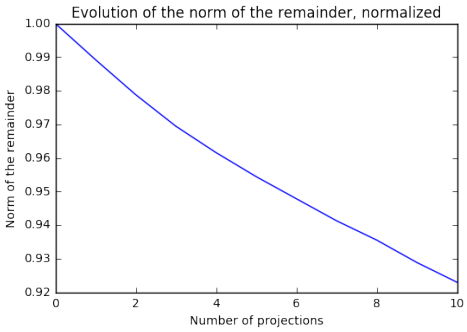


Figure 37: Plane activation remainder norm evolution

6.4 $W_{a,x}$ differ from $W_{a,x+\epsilon}$

Given an activation a and an input x, look at the angle between $W_{a,x}$ and $W_{a,x+\epsilon}$, $\epsilon \sim \mathcal{N}(0, \sigma^2)$:

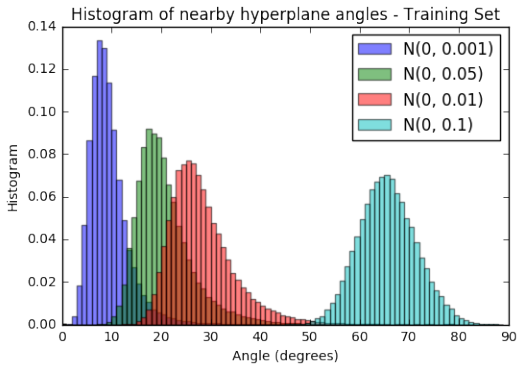


Figure 38: Training set angles for different noise.

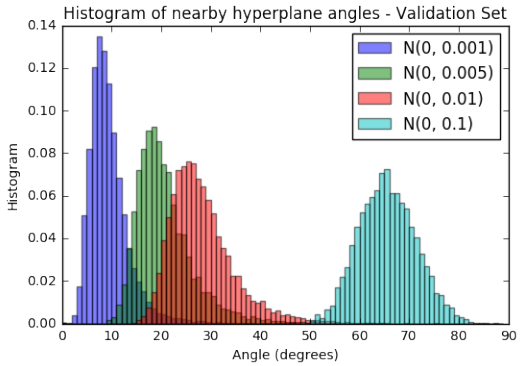


Figure 39: Validation set angles for different noise.

6.5 Exploring polytops

The goal here is to get an idea of how large a neighborhood of an image can be and still be in the same linear region. Let's proceed as follow :

- Input : An image and we compute its activation pattern. N the number of images we seek to find in the same linear region.
- Initialize N images equals to the input.
- Initialize σ uniform noise range.
- Repeat :
  - Add uniform noise to the N images.
  - forward all images, compute their activation pattern.
  - Compare to initial activation pattern, drop all images that have a different activation pattern.
  - If there is less than 20% images remaining, divide σ by 2. If there is no more images, re-initialize N images equals to the input.
  - Recreate images as a copy of remaining images so we have N images.
  - Compute the l_2 distances between all images, drop the 10% images that have the lowest cumulative l_2 distance to promote diversity.
- Print N images.

Only first layer Final noise scale : $1e-4.00421$

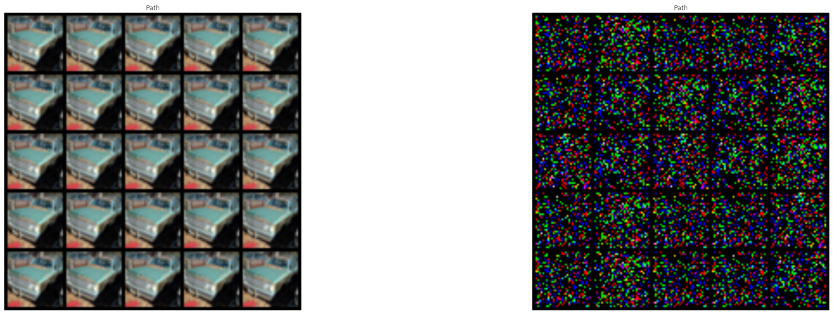


Figure 40: Layer 1 only geodesic + noise plotting

Only Second layer Final noise scale : $1e-3.51966$

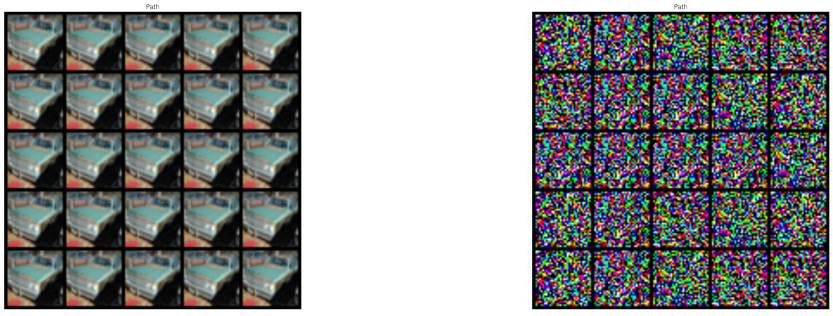


Figure 41: Layer 2 only geodesic + noise plotting

Only Third layer Final noise scale : $1e-3.22893$

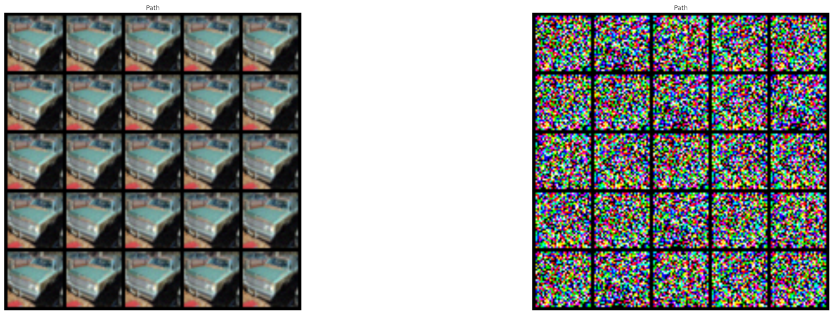


Figure 42: Layer 3 only geodesic + noise plotting

All 3 layers Final noise scale : $1e - 4.29494$

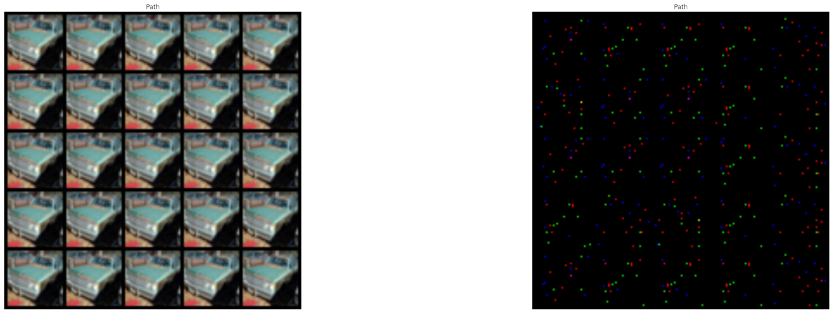


Figure 43: All 3 layers images and only noise

Notes

- The linear regions are extremely tiny : we can't see any differences between the images.

7 Intuitions and Future work ideas

7.1 Number of linear regions created by a linear layer

Let's call n the number of activations and r the rank of the matrix of the linear layer, $f(n, r)$ the number of linear regions. Then we can show that :

$$f(n, n) = 2^n \tag{59}$$
$$f(n, 1) = n + 1 \tag{60}$$

My intuition is that we also have something like (based on small numbers of n, r computation that we can make) :

$$f(n + 1, r + 1) = f(n, r + 1) + f(n, r) \tag{61}$$

Notes

- Given that what I wrote is accurate, then we could regularize a network based on the rank of the matrix, which is possible with convolution as well (controlling the rank of the kernel matrix).
- Maybe we can study the conditional number of linear regions, i.e. : If you have two layers, how many linear regions are they in the network given the first layer has a fixed activation pattern. Possibly there are only a few number of linear regions left ?

7.2 Trying to count the number of linear regions

Idea 1 If we have an algorithm that given an activation pattern finds an image in it, we could sample random activation patterns, see if they are reachable and deduce the probability for an activation pattern to be reachable, hence having the order of magnitude of the number of linear regions.

Idea 2 Given an activation pattern, try to have an idea of its width, the volume it occupies. If you know the mean width of a polytopes you can deduce the number of polytopes.

Idea 3 Let's try to compute for 2 layers the boundaries. We can also look at projection cones : If I'm here (x) and I look toward there v vector, I'll look toward this decision boundary.

7.3 Conditional Activation pattern

The idea is to say : Let's take layer $l-1$ and enforce an activation pattern from layer 1 to $l-1$. How many activation patterns can layer l then take ? The intuition is to say only a handful, thus making the count of activation patterns easier ?

7.4 Adversarial Example

How does an adversarial example fit in this work? An adversarial example is a very close region in the L_2 sense, the question is now: Is an adversarial example very far in terms of Activation pattern or still very close, and how can this information help us to anihilate adversarial examples?

If it's very close, then two near activation patterns can have drastically different classifier and thus the goal is to regularize how much a classifier change from two nearby activation pattern. On the other hand, there is many linear regions in a very small L_2 distance, and thus we would need to focus on minimizing the number of linear regions.