An alternative approach is to look at the activations themselves.

Since the convolutional layers maintain the 2d structure of the signal, the activations can be visualized as images, where the local coding at any location of an activation map is associated to the original content at that same location.

Given the large number of channels, we have to pick a few at random.

**Since the representation is distributed across multiple channels, individual channel have usually no clear semantic.**
A MNIST character with LeNet (LeCun et al., 1998).

An RGB image with AlexNet (Krizhevsky et al., 2012).
ILSVRC12 with ResNet152 (He et al., 2015).

Yosinski et al. (2015) developed analysis tools to visit a network and look at the internal activations for a given input signal.

This allowed them in particular to find units with a clear semantic in an AlexNet-like network trained on ImageNet.
Prediction of 2d dynamics with a 18 layer residual network.

\[
G_n \quad S_n \quad R_n
\]

(Fleuret, 2016)
\[ \Psi(S_n, G_n) \]

(Fleuret, 2016)
Layers as embeddings

(Fleuret, 2016)
In the classification case, the network can be seen as a series of processings aiming as disentangling classes to make them easily separable for the final decision.

In this perspective, it makes sense to look at how the samples are distributed spatially after each layer.

The main issue to do so is the dimensionality of the signal. If we look at the total number of dimensions in each layer:

- A MNIST sample in a LeNet goes from 784 to up to 18k dimensions,
- A ILSVRC12 sample in Resnet152 goes from 150k to up to 800k dimensions.

This requires a mean to project a [very] high dimension point cloud into a 2d or 3d “human-brain accessible” representation.
We have already seen PCA and $k$-means as two standard methods for dimension reduction, but they poorly convey the structure of a smooth low-dimension and non-flat manifold.

It exists a plethora of methods that aim at reflecting in low-dimension the structure of data points in high dimension.

Given data-points in high dimension

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

the objective of data-visualization is to find a set of corresponding low-dimension points

$$\mathcal{E} = \{y_n \in \mathbb{R}^C, \ n = 1, \ldots, N\}$$

such that the positions of the $y$s “reflect” that of the $x$s.
The t-Distributed Stochastic Neighbor Embedding (t-SNE) proposed by van der Maaten and Hinton (2008) optimizes with SGD the \( y_j \)'s so that the distances to close neighbors of each point are preserved.

It actually matches for \( D_{KL} \) two distance-dependent distributions: Gaussian in the original space, and Student t-distribution in the low-dimension one.

The scikit-learn toolbox

http://scikit-learn.org/

is built around SciPy, and provides many machine learning algorithms, in particular embeddings, among which an implementation of t-SNE.

The only catch to use it in PyTorch is the conversions to and from numpy arrays.

```python
from sklearn.manifold import TSNE

# x is the array of the original high-dimension points
x_np = x.numpy()
y_np = TSNE(n_components = 2, perplexity = 50).fit_transform(x_np)
# y is the array of corresponding low-dimension points
y = torch.from_numpy(y_np)
```

\( n_{\text{components}} \) specifies the embedding dimension and \( \text{perplexity} \) states [crudely] how many points are considered neighbors of each point.
t-SNE unrolling of the swiss roll (with one noise dimension)

t-SNE for LeNet on MNIST
t-SNE for a home-baked resnet (no pooling, 66 layers) CIFAR10

References


