50.039 – Theory and Practice of Deep Learning

Alex

Week 10: Interpretability of Models and Predictions I

Interpretability in Machine Learning and Deep Learning. The following is a program over 2 lectures

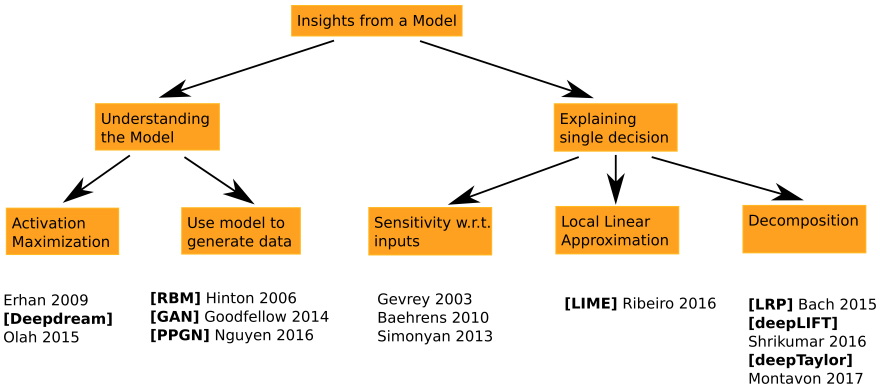
- What questions?
- model interpretation
  - feature visualization by activation maximization
  - t-SNE embeddings (pytorch + scikit learn) of features from a dataset
- decision interpretation (one unlabeled test sample as compared to a dataset)
  - Gradient-based (and its deficiencies) in pytorch
  - LIME
  - guided backprop in keras+tf+innvestigate
  - LRP and Deep Taylor in keras+tf+innvestigate
  - application to finding biases in your data, to identify systematic failcases

Key takeaways

- understanding the three basic approaches:
  - maximize activations of neurons and look at what maximizes them
  - use a neural network to generate data (e.g. as encoder part in an auto-encoder)
  - visualize similarities between the features over a dataset, extracted from some layer
  - explaining what in an input (image, sentence as sequence, audio file) contributes to a prediction
- be able to explain the basic idea of feature visualization by activation maximization
- be able to explain what for t-sne is good for: namely that it computes a low-dimensional embedding y_i for each sample x_i, which tries to preserve local similarities between the original samples. where similarity is a gaussian over the distance
- The basic steps how it works (symmetrical pairwise interaction probability of input samples based on gaussians, symmetrical pairwise interaction probability of embedded samples based on heavytailed distributed, optimize embedded sample location by minimizin KL-Div)
- visualize a decision using gradient
- be able to explain the drawbacks when visualizing a decision using gradient
- visualize a decision using LIME
- be able to explain the drawbacks of LIME

Trained a model on a dataset. Have some performance 83.7% accuracy. What does one want to know else ?

- What features has it learned (convolutional filters?) what inputs activate a feature?
- what input samples does the model consider as similar to each other?
- Does it predict for the right reasons ? Does it predict the existence of A because it recognizes B and it has learned that B co-occurs often with A? (Pascal VOC)
- ???



1 Interpret models

1.1 Analysing learned features by synthetising outputs

One simple way to visualize learned features is to ask what inputs maximize a feature map?
For a simple neuron that would be – without any regularization:

$$y = g(w \cdot x + b)$$
$$x = \text{argmax}_x y$$

Obviously this has a flaw, which one ?

Typically one adds a certain regularization, to restrict the possible samples to lie in some kind of bounded space:

$$y = g(w \cdot x + b)$$
$$x = \text{argmax}_{x:\|x\|_p \leq 1} y$$

In general a layer channel $z[:, c, :, :]$ or one (i, j)-element of a layer channel $z[:, c, i, j]$ of a neural network is not a simple neuron, but the same idea applies.
The direct way is

$$\hat{x} = \text{argmax}_{x:\|x\|_p \leq 1} \sum_{h,w} z[:, c, h, w](x)$$

How to implement $\sum_{h,w} z[:, c, h, w]$ in pytorch ?
If we do not care about whether the activation is positive or negative, then one can resort to

$$\hat{x} = \text{argmax}_{x:\|x\|_p \leq 1} \sum_{h,w} z[:, c, h, w]^2(x) = \text{argmax}_{x:\|x\|_p \leq 1} \| \, z[:, c, \underbrace{:, :}_{\text{norm taken over this}}](x) \, \|_2^2$$

Unfortunately often using this natively results in high frequency noise of the type which is added to create adversarial attacks. Note: this is one legitimate outcome.

If one wants visually appealing low frequency shapes, then one can use either the approach in Nguyen et al with parametrized shapes, or one applies blur and pixel-translation jitter to an image after every step.
In the end one uses regularizers to see something else than noise

$$\hat{x} = \text{argmax}_{x:\|x\|_p \leq 1} \sum_{h,w} z[:, c, h, w](x) + R(x)$$

$R(x)$ can be for example an approximation to the total variation, for example

$$TV(x) = \sqrt{\sum_{ij} (x_{i,j+1} - x_{i,j})^2 + (x_{i+1,j} - x_{i,j})^2} \approx \sum_{ij} |x_{i,j+1} - x_{i,j}| + |x_{i+1,j} - x_{i,j}|$$

`https://distill.pub/2017/feature-visualization/` gives a good overview over tricks used.

(-) unclear scientific value – how to use it to understand what a model does or how to improve it ??

(-) in terms of model understanding: a preference to pretty shapes to be generated over ugly noise is human bias for certain aesthetics??

(+) artistic value!

1.2 Analysing learned features by dataset evaluation

A data-driven variant of the above considers those images/ input samples which activate a neuron most

$$\hat{x} = \text{argmax}_{x \in \text{Data}} \sum_{h,w} z[:, c, h, w](x)$$

It does not show what a filter really learned, but it shows what it focuses on in your given dataset.

1.3 visualize similarity of samples from a learned model by a 2d embedding

Different idea: suppose I have feature maps of 200 images, how to visualize the similarities between these feature maps ?

One way is to visualize the similarities between samples by embedding all these in 2 dimensions – according to their similarities and look at them. One in CS circles well known way is t-sne.
`http://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf`
Working principle: Given high dimensional data points $D_n = (x_1, \ldots, x_n)$, goal is to create n 2-dimensional data points $y_1, \ldots, y_n$ which have similar distances to each other as the set D_n.

- step 1 compute the probability that i would vote for j as being his neighbor based on a gaussian model which is centered on x_i as mean

$$p_{j|i} \propto \exp(-\|x_i - x_j\|^2 / 2\sigma^2)$$
$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma^2)}{\sum_{k:k \neq i} \exp(-\|x_i - x_k\|^2 / 2\sigma^2)}$$

  (sums up over j to 1)

- symmetrize

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}, p_{ii} = 0$$

  Reason? $\sum_i p_{ij} > \frac{1}{2n}$, so each point, even an outlier has some large interactions to his neighbors. Otherwise points i which are very far outliers may contribute little to the embedding because $p_{i|j} \approx 0$.

- learn a similar, but heavy-tailed distribution model of y_i voting for y_j as neigbor:

$$q_{ij} \propto (1 + \|y_i - y_j\|^2)^{-1}$$
$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k,l:k \neq l} (1 + \|y_k - y_l\|^2)^{-1}}$$

- how to optimize for q_{ij}? Minimize Kullback-Leibler-Divergence:

$$\{q_{ij}, ij\} = \text{argmin}_{\{q_{ij}, ij\}} KL(P||Q) = \text{argmin}_{\{q_{ij}, ij\}} \sum_{ij} p_{ij} \log\left(\frac{p_{ij}}{q_{ij}}\right)$$

- minimize for y_i by computing the gradient of $KL(P||Q)$ with respect to y_i (Q depends on y_i)

Why for the y_i use a heavy tailed probability?

`https://lvdmaaten.github.io/publications/papers/JMLR_2008.pdf`:
"This allows a moderate distance in the high-dimensional space to be faithfully modeled by a much larger distance in the map and, as a result,it eliminates the unwanted attractive forces between map points that represent moderately dissimilar datapoints."

Idea is the following: in high dimensional spaces many points can have intermediate distance to each other, resulting in an intermediate interaction probability p_{ij}.

What means choosing a heavy tailed probability for the model of the y_i? A heavy tailed probability assigns a relatively high probability to points far away. Therefore those points x_i, x_j with intermediate distance in the original space can be assigned to points y_i, y_j in the 2-d model which are far away (and still result in an intermediate-valued q_{ij} which fits well to the intermediate p_{ij}.)

By this t-sne can focus on putting only those points i and j close in the embedding y_i, y_j for which the original points x_i and x_j are really close.
See: `https://distill.pub/2016/misread-tsne/` – what one gets out from t-sne, depends a lot on the perplexity parameter choice, and it may be very different from the original distances. Its a visualization, not some kind of truth. Results need to be validated.

2 Interpret Single Decisions I: Gradient

- compute gradient
- uses as score for the relevance of an input dimension x_d:

$$r_d(x) = \left(\frac{\partial f}{\partial x_d}(x)\right)^2$$

(+) easy to implement

(+) fast to compute

(-) see below what sensitivity explains, often not the question you wanted to ask

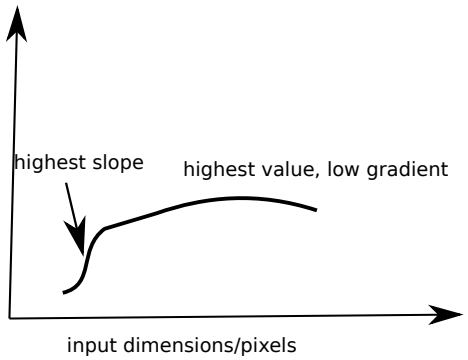
The main drawback:

What the gradient explains

- The gradient does **not** explain which pixels **are most contributing to the prediction** of a cat.
- The gradient explains which pixels **are most sensitive to change the prediction** of a cat.

Most sensitive to change ≠ most contributing

Compare: where a function has highest value vs where a function has highest slope.



3 Interpret Single Decisions II: Lime

`https://arxiv.org/pdf/1602.04938.pdf` Ribeiro et al, ICML 2016

The first thing to understand: the decisions made by a linear model are easily explainable

$$g(x) = \sum_d w_d x_d$$

$$r_d := w_d x_d$$

The contribution of a single decision can be explained easily. If it has a bias, there is an unexplained component thought – the bias b, which cannot be naturally assigned into contributions of single dimensions d:

$$g(x) = \sum_d w_d x_d + b$$

$$r_d := w_d x_d$$

The idea of Lime: given a test sample x learn a locally linear approximation to f around x.

$$f(x) \approx A(x) = \sum_d w_d x_d$$

$$r_d(x) = w_d x_d$$

How to get to the locally linear approximation $A(x)$?

Algorithm 1 Sparse Linear Explanations using LIME

Require: Classifier f, Number of samples N
Require: Instance x, and its interpretable version x'
Require: Similarity kernel π_x, Length of explanation K
  $\mathcal{Z} \leftarrow \{\}$
  for $i \in \{1, 2, 3, ..., N\}$ **do**
    $z'_i \leftarrow sample\_around(x')$
    $\mathcal{Z} \leftarrow \mathcal{Z} \cup \langle z'_i, f(z_i), \pi_x(z_i) \rangle$
  end for
  $w \leftarrow$ K-Lasso$(\mathcal{Z}, K)$ ▷ with z'_i as features, $f(z)$ as target
  return w

K-Lasso

- train lasso

$$w \leftarrow \text{argmin}_w \frac{1}{2n} \sum_i (f(z_i) - w \cdot z_i) + \lambda \|w\|_1$$

- ℓ_1-norm makes many weights to be zero
- select K dimensions with highest weights
- train a linear/ridge regression with only those dimensions to obtain $A(x)$
- parameters: sampling radius size, K, λ

A large sampling radius allows to learn correlations between neighboring data points more than just the gradient.

One thing to be taken care is: for a too small sampling radius LIME converges to the gradient – which answers a different question.

(+) conceptually clear

(-) need to train for every input sample

(-) explanation sensitive to radius parameter – need to test with these parameters

4 In class tasks

4.1 gradient

Quick:

- take any pretrained neural net
- take a few images, compute the sensitivity as squared gradient,
- visualize the sensitivity by:
  - summing them over all subpixels of a pixel $h(p) = \sum_{c \in r,g,b} (\frac{\partial f}{\partial x_{p,c}})^2$, then normalize it so that $\max_p abs(h) = 1$ and $h(p) \in [-1, 1]$
  - then transform it, so that the values lie in $[0, 1]$ and $h(p) = 0$ is mapped onto 0.5
  - then use a matplotlib colormap of your choice `https://matplotlib.org/gallery/color/colormap_reference.html` `https://matplotlib.org/users/colormaps.html` to turn h into a heatmap

```
cmap_type="seismic"
cmap = plt.cm.get_cmap(cmap_type)

maxabsval=np.amax(np.abs(hmimg))
hm2=hmimg/maxabsval/2.0+0.5  #[-1,+1] (after divide by maxabsval)  /2.0
                             #--->  [-0.5,+0.5]  +0.5 ---> [0,1]

#colormaps take values in [0,1] as input
coloredhm=cmap(hm2)*255.0

imhmf= .......
PIL.Image.fromarray(coloredhm.astype(np.uint8)).save(imhmf)
```

4.2 t-sne over feature sets

- take any pretrained neural net, compute features for $\approx 2 \times 250$ images from two classes. example: 2 distinctive flowers
- compute a t-sne for two 2 dimensions `https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html`
- visualize it with images as points (see code in dropbox)
- then redo this after you trained the same neural net to discriminate these two classes