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

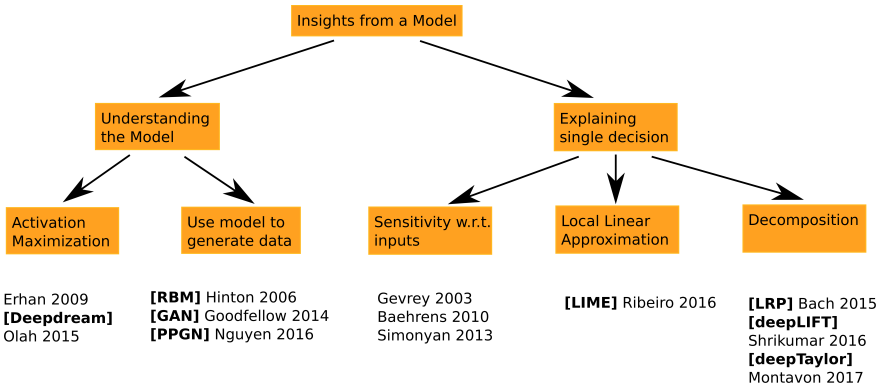
- What questions?
- model interpretation
  - t-SNE embeddings (pytorch + scikit learn) of features from a dataset
- decision interpretation (one unlabeled test sample as compared to a dataset)
  - Gradient-based (and its deficiencies) in pytorch
  - LIME
  - guided backprop in keras+tf+innvestigate
  - 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:
  - 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 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
- 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 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 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

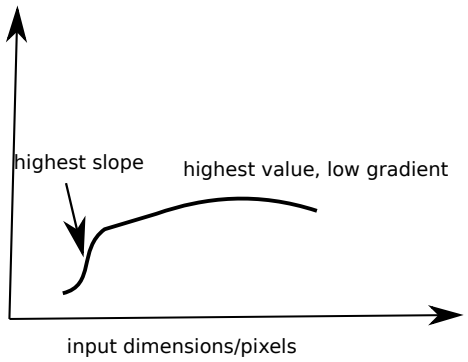
The main drawback:

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

Key takeaways

- Today two better approaches for explaining a single decision:

(1) guided backprop

(2) LRP

- applications of LRP and other single-sample explanations methods:
  - finding biases in training data
  - selecting samples from unsupervised datasets for annotation and for iterative growing of datasets, – when annotations are not cheap (medicine, physics)
  - explaining decisions when the why matters
- evaluating the quality of an explanation itself!

Plus some messing with keras and tensorflow using guided backprop and LRP.

We had 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

and LIME

- sample datapoints z around your point of interest and the value of f on them $f(z)$
- train sparse linear approximation $A(x) = \sum_d w_d x_d + b$ on $\{(z, f(z))\}$
- use the linear component contributions $R_d = w_d x_d$ for an explanation

(+) conceptually clear

(-) need to train for every input sample

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

3.1 Guided backpropagation

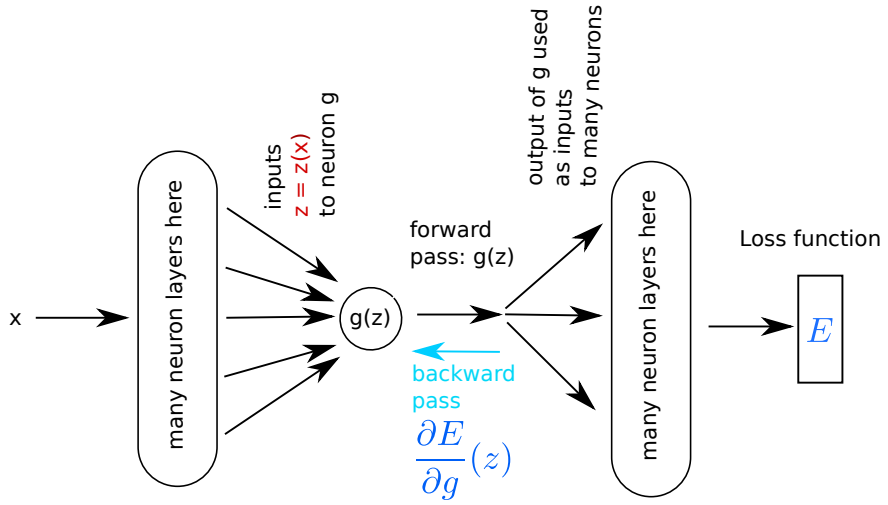
Key takeaways for guided backprop

The computation rule.

backpropagation with a heuristic to cancel out parts of the backpropagated scores.

Consider a relu activation neuron $g(z) = relu(w \cdot z + b)$ (more general: activation function).

- this receives in the forward pass the value vector $z = z(x)$ from the layers below.
- In the backward pass it received the derivative with respect to itself $\frac{\partial E}{\partial g}(z)$ from the layers above.



Guided backprop says: do backprop but **zero out incoming gradient** $\frac{\partial E}{\partial g}(z)$ when passing to through $g()$ if:

- the activation of the neuron is negative $g(z) < 0$
- the gradient arriving at this neuron is negative $\frac{\partial E}{\partial g}(z) < 0$

Why?

- if the activation of the neuron is negative $g(z) < 0$, then $g(z)$ is a suppressing neuron. rule: Ignore gradients from suppressing neurons, pass through only gradient signal from firing neurons
- if the gradient arriving at this neuron is negative. rule: ignore gradients which decrease the function value, look only at gradients which increase the prediction

- its a heuristic to look only at one end of effects. Look only at activating signals and gradients

In class: test it with keras.

(-) its a heuristic, no theoretical underpinning. Not really sure what it does.

(+) look only at one sign of signals and gradients gives often clean heatmaps

3.2 LRP

Key takeaways for LRP

- LRP as decomposition of a prediction: $f(x) \approx \sum_d r_d(x)$ with additional constraints
- Given a neuron $y = g(\sum_d w_d x_d)$, LRP distributes a quantity, relevance, top-down: from a neuron output y, onto the the inputs x_d of that neuron.
- LRP results in different rules which can be applied to different types of neural network layers.
- Example: β-rule as proportionality.

  Given a neuron $y = g(\sum_d w_d x_d + b)$, and R_y being the relevance of its output, β-rule says:

$$
\begin{aligned}
r_d(x) &= R_y \, h(w_d x_d) \\
h(w_d x_d) &= (1+\beta)\,(w_d x_d)_+ \, C_+ - \beta\,(w_d x_d)_- \, C_- \\
C_+ &= \sum_{d'} (w_{d'} x_{d'})_+ + b_+ \\
C_- &= \sum_{d'} (w_{d'} x_{d'})_- + b_-
\end{aligned}
$$

  that is: the relevance $r_d(x)$ of an input x_d is computed as the top-down relevance R_y times (the positive part $(w_d x_d)_+$ of $w_d x_d$ with some normalizing constant C_+ + the negative part $(w_d x_d)_-$ of $w_d x_d$ with some normalizing constant C_-). no need to memorize C_+, C_-
- Example: ϵ-rule as alternative for fully connected layer as proportionality

$$
\begin{aligned}
r_d(x) &= R_y \, h(w_d x_d) \\
h(w_d x_d) &= w_d x_d \, C \\
C &= \left(\sum_{d'} w_{d'} x_{d'} + b \right) + \epsilon \, \text{sign} \left(\sum_{d'} w_{d'} x_{d'} + b \right)
\end{aligned}
$$

- LRP related to neuron-wise Taylor decomposition

Motivation: Linear models are easily interpretable. Suppose one has a classification

with prediction thresholded at zero

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

$$pred = \begin{cases} 1 & \text{if } f(x) > 0 \\ 0 & \text{if } f(x) <= 0 \end{cases}$$

We know that if $f(x) > 0$, then dimension d with $w_d x_d > 0$ supports the prediction. If $f(x) > 0$, then dimension d with $w_d x_d < 0$ contains evidence against the prediction $f(x) > 0$.
Also, if $w_d x_d > 0$ and $w_d x_d \gg w_{d_0} x_{d_0} > 0$, then dimension d has a stronger contribution to $f(x) > 0$ than d_0.

Thus is natural to define as an explanation for the contribution of dimension d:

$$f(x) = \sum_d w_d x_d \Rightarrow R_d = w_d x_d$$

When asking: how much does dimension d contribute to a classification prediction, then the gradient gets even a linear model wrong:

$$\frac{\partial f}{\partial x_d} = w_d$$

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

Ignores whether sign(x_d) ==sign(w_d), ignores scale of x_d completely ... gradient just asks which dim is most sensitive!?

What do we want ?

- **Decomposition of a prediction:** Given a classifier f which predicts label= 1 if $f(x) > 0$ over a sample $x = (x_1, \ldots, x_D)$: Want a score $r_d(x)$ for the input dimension x_d which tells us how much x_d contributes to $f(x)$
- goal: sum of relevances $r_d(x)$ reconstruct $f(x)$ (approximatively):

  $$f(x) \approx \sum_d r_d(x)$$

  subject to constraints, so that this decomposition becomes meaningful – the constraint are in place to avoid meaningless decompositions like that uniform one $r_d(x) = f(x)/D$
- Want a decomposition which, when applied to a linear layer, is at least for some parameter values consistent with the natural decomposition of a linear model

  $$f(x) = \sum_d w_d x_d \Rightarrow R_d = w_d x_d$$

- How does LRP solve that?
- What for can an explanation of neural nets be used?

see uploaded talk slides