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

- Today two better approaches for explaining a single decision:

(1) guided backprop

  - guided backprop as backprob with zome zeroing outrules: inhibiting input to activation or negative gradient at activation

(2) Grad-Cam and Guided Grad-Cam

(3) LRP

  - LRP as decomposition of a prediction with constraints
  - backpropagates relevance from a neuron to inputs of the neuron
  - relevance of any neuron x as sum of relevance messages from neurons for which x is input
  - β-rule: relevance message is proportional to relevance from output $R(y)$ times a weighted sum of positive part $(w_d x_d)_+$ of weighted input and negative part of weighted input $(w_d x_d)_-$

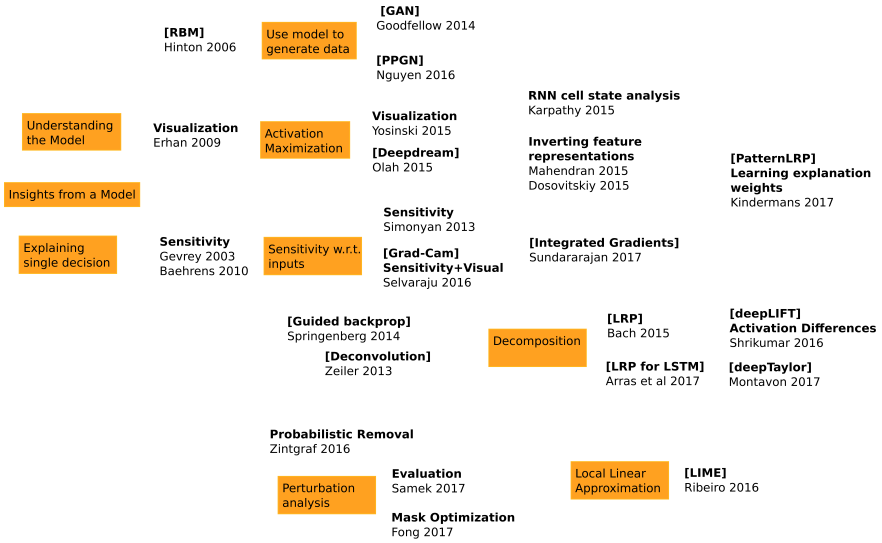
- applications of LRP and other single-sample explanations methods:

  - 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 and the latter two in GPU mode.

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



1 Interpret models

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

1.1 Guided backpropagation

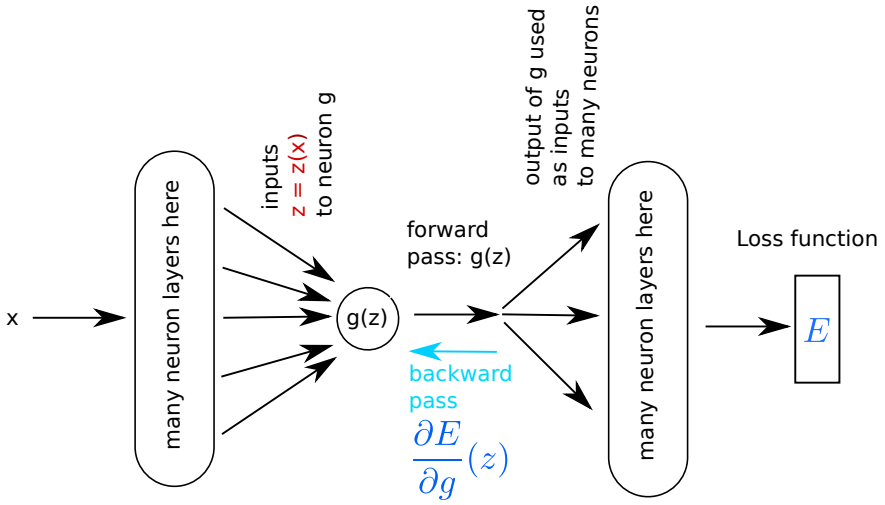
Key takeaways for guided backprop

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

1.2 LRP

Key takeaways for LRP

- LRP distributes a quantity, relevance, top-down
- β-rule as proportionality: the relevance of an input is the top-down relevance times (a constant times the positive part $(w_d x_d)_+$ of $w_d x_d$ + another constant times the negative part $(w_d x_d)_-$ of $w_d x_d$)
- ϵ-rule as alternative for fully connected layer as proportionality
- LRP as decomposition $f(x) = \sum_d r_d(x)$ with constraints
- LRP as 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:

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

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 ?

- 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 $f(x)$
- we try to achieve this in way that all these scores $r_d(x)$ 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

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

- want a decomposition that can be done for every neuron separately

2 LRP applications

Gait analysis, neuroscience

3 todos

LRP for text cnn LRP for LSTM LRP for X ?