Notes on Canonization for Resnets and Densenets

Alexander Binder

July 31, 2020

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

Credits

Code inspired by Philipp Seegerer (TU Berlin).
With important ideas from Gregoire Montavon (TU Berlin), Philipp Seegerer and Sebastian Lapuschkin (Fraunhofer HHI).

Recap on LRP rules

understanding properties of rules: LRP-ϵ

LRP-ϵ:

$$R_{i \leftarrow k}(\mathbf{x}) \propto R_k h(w_i x_i) \tag{1}$$

$$z = \sum_{i'} w_{i'} x_{i'} + b \tag{2}$$

$$R_{i \leftarrow k}(\mathbf{x}) = R_k \left(\frac{w_i x_i}{z + \epsilon \mathrm{sign}(z)} \right) \tag{3}$$

- may produce $R_{i \leftarrow k}$ with $|R_{i \leftarrow k}| \gg |R_k|$
- no control over relevance scale!
- ϵ - dampens redistribution differences
- $\epsilon \to \infty$ convergence to flat redistribution

understanding properties of rules: LRP-β

LRP-β:

$$R_{i \leftarrow k}(\mathbf{x}) \propto R_k h(w_i x_i) \tag{4}$$

$$R_{i \leftarrow k}(\mathbf{x}) = R_k \left((1+\beta) \frac{(w_i x_i)_+}{\sum_{i'} (w_{i'} x_{i'})_+ + b_+} - \beta \frac{(w_i x_i)_-}{\sum_{i'} (w_{i'} x_{i'})_- + b_-} \right) \tag{5}$$

- β controls ratio of negative to positive evidence.
- bounded relevance scale: $|R_{i \leftarrow k}| \leq (1+\beta)|R_k|$
- negative to positive evidence: $\frac{\beta}{1+\beta}$,
- negative to total evidence: $\frac{\beta}{1+2\beta} \to 0.5$,
  It is fixed independent of network inputs(!).

which rule for which layer?

Name	Formula	layers
LRP-ϵ	$\sum_k R_k \left(\frac{x_i w_{ik}}{\sum_i x_i w_{ik} + b + \epsilon \mathrm{sign}(z)} \right)$	linear
LRP-$\beta = 0$	$\sum_k R_k \left(\frac{(x_i w_{ik})_+}{\sum_i (x_i w_{ik})_+ (b)_+} \right)$	conv
LRP-$\alpha - \beta$	$\sum_k R_k \left(\frac{(1+\beta)(x_i w_{ik})_+}{\sum_i (x_i w_{ik})_+ + (b)_+} - \beta \frac{(x_i w_{ik})_-}{\sum_i (x_i w_{ik})_- + (b)_-} \right)$	conv
LRP-z_β	$\sum_k R_k \left(\frac{x_i w_{ik} - l_i (w_{ij})_+ + h_i (w_{ij})_-}{\sum_i x_i w_{ik} + b - l_i (w_{ij})_+ + h_i (w_{ij})_-} \right)$	first conv layer
LRP-w^2	$\sum_k R_k \frac{w_{ik}^2}{\sum_i w_{ik}^2}$	same 1. conv
LRP-γ	$\sum_k R_k \frac{x_i w_{ik} + \gamma (x_i w_{ik})_+}{\sum_i (x_i w_{ik} + b + \gamma (x_i w_{ik})_+ \gamma (b)_+)}$	conv

Biases in denominators can be omitted during the backward pass for better attributions

step I: canonization resnet example

make model usable for custom backward explanations

- create a modified copy with parameters from trained source model
- technical issue: need to replace the $+$ in a residual connection $x + ConvConv(x)$ by an operator implementing $+$.
- LRP-issue: fuse Conv-BatchNorm chains into a Conv-Layer

The conv-BN fusion is due to an LRP-issue:

$\boxed{\text{Adebayo et al: LRP is not implementation invariant.}}$

step I: canonization resnet example

Why conv-bn-fusion? Adebayo et al: LRP is not implementation invariant.

$\boxed{\text{Why LRP then at all and not Gradient/ Grad-CAM ?}}$

- Gradient estimates an often suboptimal measure:
  a single-pixel sensitivity instead of contributions which account for interactions between larger regions.
- Gradient: +high noise from gradient shattering in ReLU nets.
- For a comparison of gradients against guided back prop
  in a medical context see eg. Eitel et.al. MICCAI 2019
  https://link.springer.com/chapter/10.1007/978-3-030-33850-3_1
  https://arxiv.org/abs/1909.08856
- for NLP: Poerner et al. ACL 2018,
  https://www.aclweb.org/anthology/P18-1032.pdf

step I: canonization resnet example

Fail in Implementation-invariance can be managed.

Fail in measures u can?

step I: canonization resnet example

manual step!

- resnet: need to replace the $+$ in a residual connection $x + ConvConv(x)$ by an operator implementing $+$.
  Why we do not overload the backward passes for $+$ in general?

Check the code example for `copy_resnet_onlycopy_v2.py`

- create a `nn.Module`-derived class `sum_stacked2` ,
- create a derived bottleneck, basicblock, resnet classes (easy).
- replace the shortcut by `sum_stacked2`

step I: canonization resnet example

- fuse Conv-BatchNorm chains into a Conv-Layer. Resnet has the following chain: Conv $\rightarrow$ BN

$$\text{conv-layer: } y = w_{conv} \cdot x + b_{conv,c} \tag{6}$$

$$\text{bn-layer: } z = w_c\,(y - \mu_{bn})/s_c + bn_c, \qquad s_c = (\sigma_{bn,c} + \epsilon_{bn})^{0.5} \tag{7}$$

$$\text{bn} \rightarrow conv : z = w_c/s_c\,(y - \mu_{bn}) + bn_c \tag{8}$$

$$= (w_c/s_c)\,(w_{conv} \cdot x + b_{conv,c} - \mu_{bn}) + bn_c \tag{9}$$

$$= \alpha_c \cdot x + \beta_c \tag{10}$$

$$\Rightarrow \alpha_c = (w_c/s_c) w_{conv} \tag{11}$$

$$\Rightarrow \beta_c = (w_c/s_c)(b_{conv,c} - \mu_{bn,c}) + bn_c \tag{12}$$

step I: canonization resnet example

Check the code example for `copy_resnet_onlycopy_v2.py`

$$\alpha_c = (w_c/s_c)w_{conv} \tag{13}$$

$$\beta_c = (w_c/s_c)(b_{conv,c} - \mu_{bn,c}) + bn_c \tag{14}$$

```
def bnafterconv_overwrite_intoconv(conv,bn):
    s = (bn.running_var+bn.eps)**.5
    w = bn.weight
    b = bn.bias
    m = bn.running_mean

    conv.weight = torch.nn.Parameter(conv.weight * (w / s).reshape(-1, 1,
    if conv.bias is None:
      conv.bias = torch.nn.Parameter((0 - m) * (w / s) + b)
    else:
      conv.bias = torch.nn.Parameter(( conv.bias - m) * (w / s) + b)
    return conv
```

the `.reshape(-1,1,1,1)` due to the structure of the conv-weight as: $weight[c_{out}, c_{in}, h, w]$, bn-weight: $w[c_{out}]$

step I: canonization resnet example

`copy_resnet_onlycopy_v1.py`

in derived class create routine for:

- copy layers with parameters from pretrained model + process all layers

```
def copyfromresnet(self,net, ...):
```

- conv-bn-fusion:
  - if detect conv-layer, stash it (next will be a BN!).
  - if detect bn, (1) fuse bn into stashed conv, (2) overwrite stashed conv in model, (3) reset BN stats, so that it is the identity
- which layers need to be copied from the trained model?
  - Conv2d, BatchNorm(reset), nn.Linear

step I: canonization resnet example

You got code for canonizing resnets.
`copy_resnet_onlycopy_v2.py`

```
def copyfromresnet(self,net, ...):
```

TODO:

- verify that forward passes of original and canonized model are matching!
- which layers need to be wrapped for backward pass?
  (next step, when LRP rules are implemented)
  - Conv2d, BatchNorm(reset), nn.Linear + ReLU, adaptiveavgpool, maxpool

implementation principles: how to treat biases?

- bias as constant-value firing legitimate neuron?
- bias as nuisance term onto which relevance dissipates?

!!BUG: You absolutely must not zero out biases in the forward pass!!

- if you do that, you explain a different predictor than your original model
  - predicted class gets wrong.
  - inputs x_i used to distribute relevances towards layer inputs getting wrong
  - You can zero out biases in the LRP-backward pass

canonization II: densenet-121 example

3.30 am random thought: pytorch is the communist movement in deep learning?

`copy_densenet_onlycopy.py` manual steps:

- create derived class
- replace in classifier head calls to `F.function(...)` by `nn.Module` equivalents in the derived class
  - adapt classifier head to use these equivalents: `self.toprelu`, `self.toppool`
- `def copyfromdensenet(self,net):` to copy trainable parameters from trained net
  - nn.Linear in the classifier head, nn.Conv2d at the start and denseblocks
  - canonization different from resnets!!
  - have: BN $\rightarrow$ ReLU $\rightarrow$ Conv blocks. Need to deal with this structure

canonization II: densenet-121 example

have: BN $\rightarrow$ ReLU $\rightarrow$ Conv blocks.

- step 1: swap the BN $\rightarrow$ ReLU
- step 2: fuse BN $\rightarrow$ Conv.
- result: ThreshRELU $\rightarrow$ tensorbiasedConv.
  step 2 will result in a convolution layer which cannot be represented by `nn.Conv2D` anymore, because it will have a bias which is spatially varying.

canonization II: densenet-121 example

step 1: swap the BN $\rightarrow$ ReLU

A theorem

$$\text{given } BN(x) = \frac{w_{bn}}{\sigma_{bn}} x - \frac{w_{bn}\mu_{bn}}{\sigma_{bn}} + b_{bn} \tag{15}$$

The following commutation holds for any $w_{bn} \neq 0$:

$$ReLU(BN(x)) = BN(ThreshReLU(x)) \text{ with} \tag{16}$$

$$ThreshReLU(x) = \begin{cases} x & \text{if } x - t > 0 \text{ and } w_{bn} > 0 \\ x & \text{if } x - t < 0 \text{ and } w_{bn} < 0 \\ t & \text{else} \end{cases} \tag{17}$$

$$\text{for } t = \mu_{bn} - \frac{b_{bn}\sigma_{bn}}{w_{bn}} \tag{18}$$

$$= t + (x - t)\{\mathbf{1}[x - t > 0]\mathbf{1}[w_{bn} > 0] + \mathbf{1}[x - t < 0]\mathbf{1}[w_{bn} < 0]\} \tag{19}$$

canonization II: densenet-121 example

step 1: swap the BN $\rightarrow$ ReLU $\rightarrow$ Conv2d

- replace it by ThreshReLU $\rightarrow$ BN $\rightarrow$ Conv2d
- code: `get_clamplayer` in the code computes the ThreshReLU

step 2: now can fuse the BN into the Conv2d

canonization II: densenet-121 example

step 2: now can fuse the BN into the Conv2d

$$\begin{aligned} conv(BN(x)) &= conv[w](\alpha x + \beta) + b \qquad (20)\\ &= conv[w\alpha](x) + conv[w](broadcast(\beta)) + b\\ conv.w.shape &= (n_{out}, n_{in}, ksize, ksize),\ \alpha.shape = n_{in}\\ (w\alpha)[o,c,h,w] &:= w[o,c,h,w]\alpha[c] \text{ and}\\ broadcast(\beta).shape &= (n_{in}, ksize, ksize)\\ broadcast(\beta)[c,h,w] &= \beta[c]\\ conv[w](broadcast(\beta)).shape &= (n_{out}, f, f) \end{aligned}$$

The point here is: if *conv* is using any padding, then

- $conv(broadcast(\beta))$ is not constant the spatial dimensions h, w in $[:, h, w]$
- for $kernelsize = 3$, $pad = 1$ the value on the fringe indices $h = 0$ and $h = f - 1$ will be different from $h \in [1, f - 2]$
- thats why defining the class `tensorbiased_convlayer` for densenets.

canonization II: densenet-121 example

- step 2: now can fuse the BN into the Conv2d: `def convafterbn_returntensorbiasedconv(conv,bn)` implements this
- need a similar trick in the classifier head:

$$
\begin{aligned}
&\text{BN(norm5)}\rightarrow\text{relu(toprelu)}\rightarrow\text{adaptiveAvgPool(toppool)}\rightarrow\text{linear(classifier)}\\
=\ &\text{ThresReLU} \rightarrow \text{BN} \rightarrow\text{adaptiveAvgPool(toppool)}\rightarrow\text{linear(classifier)}\\
=\ &\text{ThresReLU} \rightarrow\text{adaptiveAvgPool(toppool)}\rightarrow \text{BN} \rightarrow\text{linear(classifier)}\\
=\ &\text{ThresReLU} \rightarrow\text{adaptiveAvgPool(toppool)}\rightarrow \text{tensorbiasedlinear(classifier)}
\end{aligned}
$$