Simplified Concrete Dropout - Improving the Generation of Attribution Masks for Fine-grained Classification

- Supplementary material -

Dimitri Korsch[1][0000−0001−7187−1151], Maha Shadaydeh[1][0000−0001−6455−2400], and Joachim Denzler[1][0000−0002−3193−3300]

Computer Vision Group, Friedrich Schiller University Jena, Jena, Germany
{dimitri.korsch,maha.shadaydeh,joachim.denzler}@uni-jena.de
https://inf-cv.uni-jena.de

Abstract. This document contains supplementary material for the paper *Simplified Concrete Dropout - Improving the Generation of Attribution Masks for Fine-grained Classification*. In Sect. S1, we show a Python snippet of our implementation of the Concrete Dropout layer using the PyTorch [2] framework. In Sect. S2, we provide the results of an additional experiment about the improved gradient stability of our proposed implementation.

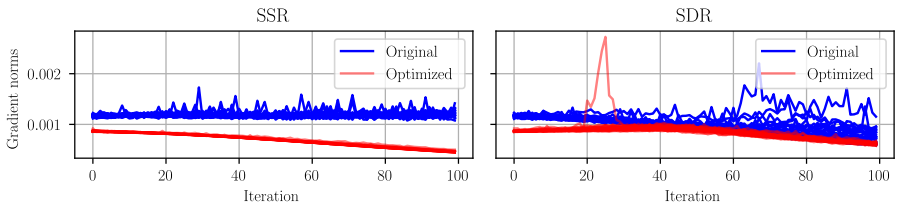
S1 Implementation

In the following, we report the Python code for our improved implementation using the PyTorch [2] framework. Contrary to the original implementation, the inputs of our presented function are directly the logits of the dropout probabilities. Mathematically, our implementation and the implementation of Gal *et al.* [1][1] are identically, but our implementation improves the stability of the estimated gradients, as we show in Sect. S2.

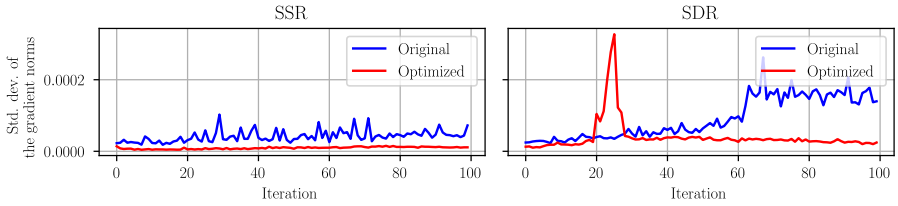
The implementation presented below is part of our FIDO implementation that can be found on GitHub: https://github.com/cvjena/fido-pytorch.

[1] https://github.com/yaringal/ConcreteDropout

```python
import numpy as np
import torch as th

def concrete_dropout(logit_p, temp: float = 0.1, u = None):
    return ConcreteDropout.apply(logit_p, temp, u)

class ConcreteDropout(th.autograd.Function):

    @staticmethod
    def forward(ctx, logit_p, temp: float = 0.1, u = None):
        """
          our proposed simplification
        """
        eps = 1e-7 # small constant for stability
        temp = th.scalar_tensor(temp)

        if u is None:
            u = th.zeros_like(logit_p).uniform_()

        noise = ((u + eps) / (1 - u  + eps)).log()

        # see Eq. (8) in the paper
        logit_p_temp = (logit_p + noise) / temp
        res = logit_p_temp.sigmoid()

        ctx.save_for_backward(res, logit_p_temp, temp)
        return res

    @staticmethod
    def backward(ctx, grad_output):
        """
          Gradient of the output w.r.t logit_p is
        1/temp * sigmoid(logit_p_temp)^2 * e^(-logit_p_temp)
          which is equivalent to
        1/temp * output^2 * e^(-logit_p_temp)
        """
        res, logit_p_temp, temp = ctx.saved_tensors
        grad = th.zeros_like(res)

        # we need this masking trick for stability reasons
        mask = res != 0

        g0 = res[mask]**2
        g1 = (-logit_p_temp[mask]).exp()
        grad[mask] = g0 * g1 / temp

        return grad * grad_output, None, None
```



(a) L2-norms of the gradients during the optimization process.



(b) Standard deviation of the L2-norms for 30 runs.

Fig. 1: We visualized the L2-norms of the gradients during the optimization process. Using 30 randomly initialized inputs, we observed the gradients during the optimization process (a) and the standard deviation across these runs (b). Except for one outlier, our improved implementation results in a lower variation of the gradients, as we stated in out paper.

S2 Gradient Stability

In an additional experiment, we estimated the gradient stability of both implementations: the original implementation as it is proposed by Gal *et al.* [1] and our improved implementation presented in this paper. First, we generated one random input and estimated the dropout masks using both approaches. During the optimization process, we observed the L2-norm of the gradients w.r.t. the dropout probabilities. We repeated this setup 30 times for 100 iterations each time. In Figure 1a, we visualized the observed L2-norms of the gradients for each run. As one can see, the variance of the gradients is much higher if the original implementation is used. Our implementation, on the other hand, produces gradients with a much lower variance, as shown in Figure 1b. With this empirical evaluation, we argue that using our implementation results in gradient estimates with a lower variance, as we stated in our paper.

References

1. Gal, Y., Hron, J., Kendall, A.: Concrete dropout. Advances in neural information processing systems **30** (2017)
2. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch (2017)