

# Straight Through Estimator

## Motivation

The key to training a Neural Network

- is Gradient Descent
- implemented through Back Propagation

But there are conditions in which Back Propagation breaks down.

Let's illustrate the issue.

For notational consistency

- we will explain Back Propagation by equating operators and layers
- the operator for layer  $i$  maps input  $y_{(i-1)}$  to output  $\mathbf{y}_{(i)}$
- this works for operators organized in non-layered architecture as well

Our eventual goal is to compute the gradient of the Loss with respect to the weights  $\mathbf{W}_i$  of each layer  $l$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_l}$$

Back Propagation achieves this

- through repeated use of the Chain Rule
- starting from the deepest layer (head) and proceeding to the shallowest layer (input)

We define the Loss Gradient for layer  $i$  as

$$\mathcal{L}'_{(l)} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{(l)}}$$

It is the derivative of  $\mathcal{L}$  with respect to the output of layer  $l$ , i.e.,  $\mathbf{y}_{(l)}$ .

Back propagation inductively updates the Loss Gradient from the output of layer  $l$  to its inputs (e.g., prior layer's output  $\mathbf{y}_{(l-1)}$ )

- Given  $\mathcal{L}'_{(l)}$
- Compute  $\mathcal{L}'_{(l-1)}$
- Using the chain rule

$$\begin{aligned}\mathcal{L}'_{(l-1)} &= \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{(l-1)}} \\ &= \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{(l)}} \frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}} \\ &= \mathcal{L}'_{(l)} \frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}}\end{aligned}$$

The loss gradient "flows backward", from  $\mathbf{y}_{(L+1)}$  to  $\mathbf{y}_{(1)}$ .

That is:

- the upstream Loss Gradient  $\mathcal{L}'_{(l)}$
- is modulated by the local gradient  $\frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}}$
- where the "layer" is the operation transforming input  $\mathbf{y}_{(l-1)}$  to output  $\mathbf{y}_{(l)}$

The problematic issue for Back-Propagation is the local gradient term

$$\frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}}$$

that relates the output of an operation (layer) to its input.

What happens when the operation

- is non-differentiable
- or has zero derivative almost everywhere
- is non-deterministic (e.g., `tf.argmax` when two inputs are identical)

Either

- the backward gradient flow is killed (zero derivative)
- or can't be computed (non-differentiable or non-deterministic) **analytically**

## Solution: Straight Through Estimator (STE)

Suppose for the purpose of Back Propagation only we replace the problematic

$$\frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}}$$

with the gradient of a *proxy function*  $\tilde{\mathbf{y}}$

$$\frac{\partial \tilde{\mathbf{y}}_{(l)}}{\partial \mathbf{y}_{(l-1)}}$$

That is, for the purpose of computing gradients

- we treat the operator as mapping  $\mathbf{y}_{(l-1)}$  to  $\tilde{\mathbf{y}}_{(l)}$  rather than  $\mathbf{y}_{(l)}$



A common choice for the proxy function is the *identity function*

- operator implements  $\mathbf{y}_{(l)} = \mathbf{y}_{(l-1)}$
- hence the formerly problematic gradient

$$\frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}}$$

is replaced by

$$\begin{aligned} \frac{\partial \mathbf{y}_{(l)}}{\partial \mathbf{y}_{(l-1)}} &= \frac{\partial \mathbf{y}_{(l-1)}}{\partial \mathbf{y}_{(l-1)}} \quad \text{since proxy implements } \mathbf{y}_{(l)} = \mathbf{y}_{(l-1)} \\ &= 1 \end{aligned}$$

The Loss Gradient flows through the operator unchanged !

This solution is called a *Straight Through Estimator*.

A consequence of using a Straight Through Estimator is that

- Back propagation does not compute the true gradient

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_l}$$

- but results in a value  $g_i$  (referred to as the *coarse gradient*)
  - not clear whether  $g_i$  is the gradient of any true function

This seems disturbing at first.

But recall that the purpose of computing the true gradient

- is to update weights  $\mathbf{W}_l$

$$\mathbf{W}_l = \mathbf{W}_l - \alpha * \frac{\partial \mathcal{L}}{\partial \mathbf{W}_l}$$

- **As long as the direction  $\mathbf{g}_i$  has a non-zero correlation with  $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_l}$** 
  - the weight update step will move in the direction of reducing the Loss

Hence, the Straight Through Estimator can be used for the purpose of Gradient Descent.

## Proxy functions

Under certain assumptions (<https://arxiv.org/pdf/1903.05662.pdf#page=5>).

- a non-zero correlation can be established between the true and coarse gradients
- for some specific proxy functions

Interestingly enough: the Identity function is **not** one of those functions (<https://arxiv.org/pdf/1903.05662.pdf#page=8>) !

- ReLU and Clipped ReLU *are* such functions
  - these are more commonly used as Activation Functions
  - but here they just map an input to an output

Nonetheless: the Identity function is commonly used as the proxy

- it may initially lead to decreased Loss
- but will may stop decreasing the loss as it approaches a local minimum

# Implementing a Straight Through Estimator in TensorFlow

## Stop Gradient operator

The *Stop Gradient* operator `sg` in TensorFlow

- acts as the identity operator on the Forward Pass

$$\text{sg}(\mathbf{x}) = \mathbf{x}$$

- But on the Backward Pass of Backpropagation: *it stops the gradient* from flowing backwards

$$\frac{\partial \text{sg}(\mathbf{x})}{\partial \mathbf{y}} = 0 \text{ for all } \mathbf{y}$$

We can use the Stop Gradient operator

- to prevent the computation of a gradient for a problematic operation

# Implementing Straight Through Estimation using Stop Gradient

Consider the implementation of an operator `ProblemOp`

- taking input `in`
- and producing output `out`
- defined by

```
class ProblemOp(layers.Layer):  
    def call(self, in):  
        # Computation of out  
        result = ...  
  
        # Straight-through estimator.  
        out = in + tf.stop_gradient(result - in)  
  
        return out
```

n.b., the `call` method is what implements the Forward Pass in TensorFlow.

The line of code

```
out = in + tf.stop_gradient(result - in)
```

seems odd: mathematically, it just copies `result` to `out`

But

- it connects the output of the operator (the LHS of the assignment)

$$\mathbf{y}_{(l)} = \text{out}$$

- to the input of the operator (appearing on the RHS of the assignment)

$$\mathbf{y}_{(l-1)} = \text{in}$$

- causing the Loss Gradient to flow from `out` to `in` during Back Propagation
  - unchanged

Thus, a Straight Through Estimator using an Identity proxy function has been created.

Note too that the line of code

- **also** connects the output (LHS of assignment) to `result` (appearing on RHS of assignment)
- but, because of Stop Gradient
  - no gradient flows from `out` to `result`

Putting the problematic operation (calculation of `result`) within a Stop Gradient

- solves the potential issue of computing a problematic gradient



In [2]: `print("Done")`

Done

