Training of Feedforward Networks Fails on a Simple Parity-Task

Supplementary Material

Sebastian Stabinger, David Peer, and Antonio Rodríguez-Sánchez

November 27, 2020

1 Manual Implementation of a CNN that Solves the Dataset

```
import tensorflow as tf
from tensorflow.keras import layers
import tensorflow_addons as tfa

import numpy as np


class PerfectCNN(tf.keras.Model):


    def __init__(self, config):
        super(PerfectCNN, self).__init__()
        self.config = config
        self.convs = []
        self.fcs = []

        # INIT CONV1 layer
        kernel = np.ones(shape=[3,3,1,12])
        bias = np.ones(shape=[1,1,1,12])
        for i in range(12):
            bias[:,:,:,i] = -(i+8)

        self.convs.append(tf.keras.layers.Conv2D(
            filters=12,
            kernel_size=(3, 3),
            kernel_initializer=tf.constant_initializer(kernel),
            bias_initializer=tf.constant_initializer(bias),
            strides=1,
            padding="VALID",
            activation="relu"))

        # INIT CONV2 layer
        kernel = np.zeros(shape=[1,1,12,12])
```

```
        for i in range(12):
            kernel[:,:,i,i] = -1
        bias = np.ones(shape=[1,1,1,12])
        self.convs.append(tf.keras.layers.Conv2D(
            filters=12,
            kernel_size=(1, 1),
            kernel_initializer=tf.constant_initializer(kernel),
            bias_initializer=tf.constant_initializer(bias),
            strides=1,
            padding="VALID",
            activation="relu"))

        # INIT CONV3 layer
        kernel = np.zeros(shape=[1,1,12,12])
        for i in range(11):
            w = 2 if (i+1) % 2 == 0 else 1
            kernel[:,:,i,i] = -1 * w
            kernel[:,:,i+1,i] = 1 * w
        self.convs.append(tf.keras.layers.Conv2D(
            filters=12,
            kernel_size=(1, 1),
            kernel_initializer=tf.constant_initializer(kernel),
            bias_initializer=tf.zeros_initializer(),
            strides=1,
            padding="VALID",
            activation="relu"))

        # Classify
        self.fcs.append(tf.keras.layers.Dense(
            units=1,
            activation="relu",
            use_bias=False,
            kernel_initializer=tf.ones_initializer()))

        kernel = [[1], [-1]]
        bias = [-5, 4]
        self.fcs.append(tf.keras.layers.Dense(
            units=2,
            activation="relu",
            kernel_initializer=tf.constant_initializer(kernel),
            bias_initializer=tf.constant_initializer(bias)))

        kernel = [[0, 2], [0, 2]]
        bias = [1, 0]
        self.fcs.append(tf.keras.layers.Dense(
            units=2,
            activation="linear",
            kernel_initializer=tf.constant_initializer(kernel),
            bias_initializer=tf.constant_initializer(bias)))


    def call(self, x, training=True):
        outputs = []
        batch_size = tf.shape(x)[0]

        # Conv layers
```

```
for conv in self.convs:
    x = conv(x)
    outputs.append(x)

# FC layers
x = tf.reshape(x, [batch_size, -1])
for fc in self.fcs:
    x = fc(x)
    outputs.append(x)

return outputs
```