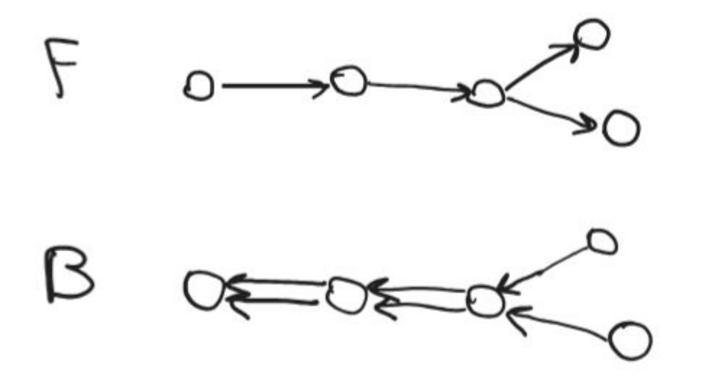
Fast Backprop for RNNs

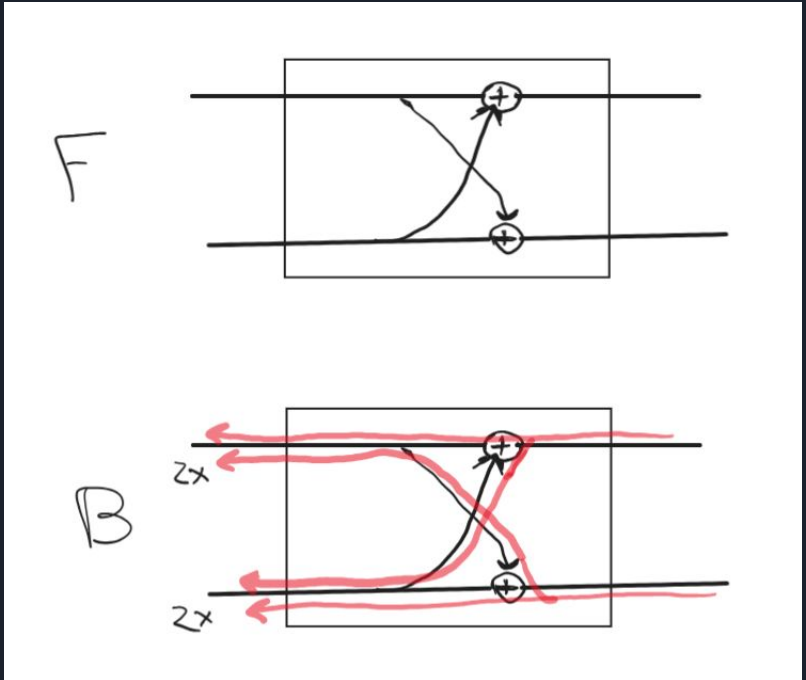
Amir Akhundzianov

Problem statement

Current Backprop algorithm – triggers all ancestors -> All possible paths are used



In RNNs – exponential complexity



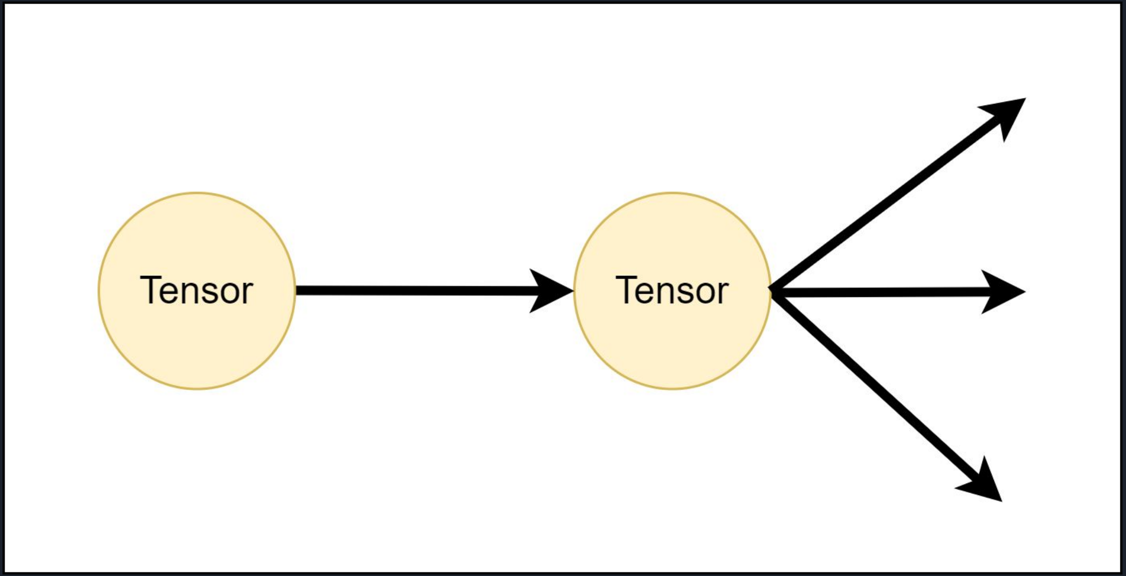
Solution targets

- minimal code changes on user side
- fair gradient computing
- optimality of asymptotics

Solution Algorithm

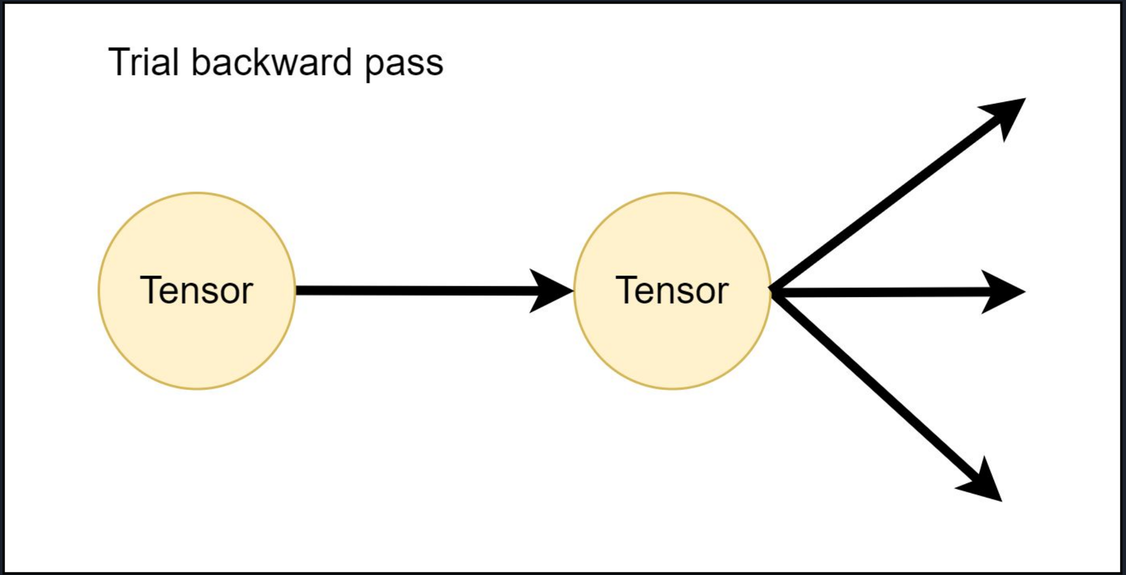
2 backward passes: trial and real

Example of Computational Graph

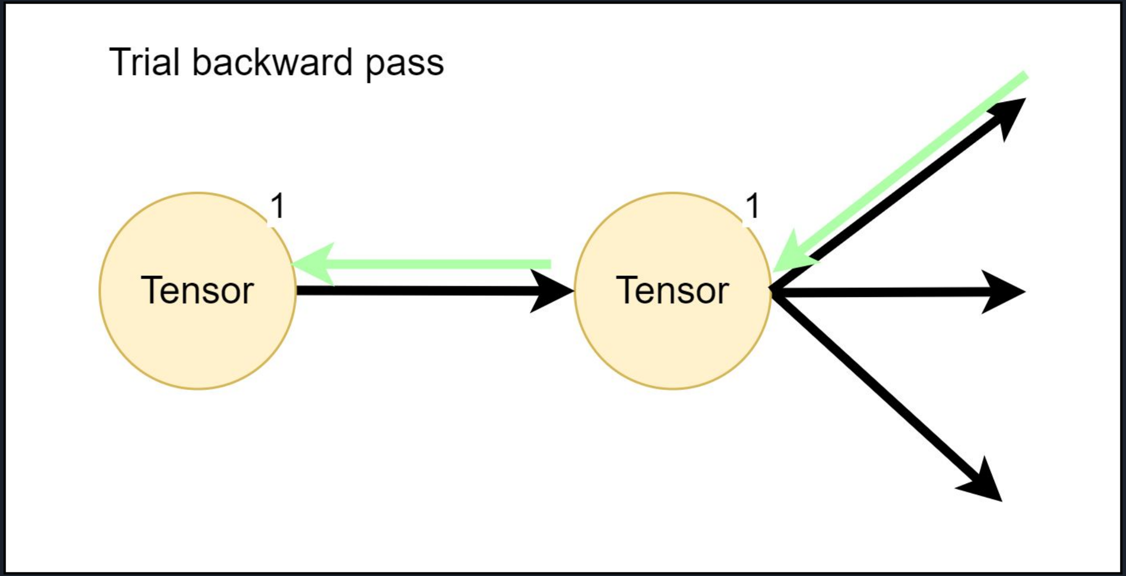


Solution Algorithm

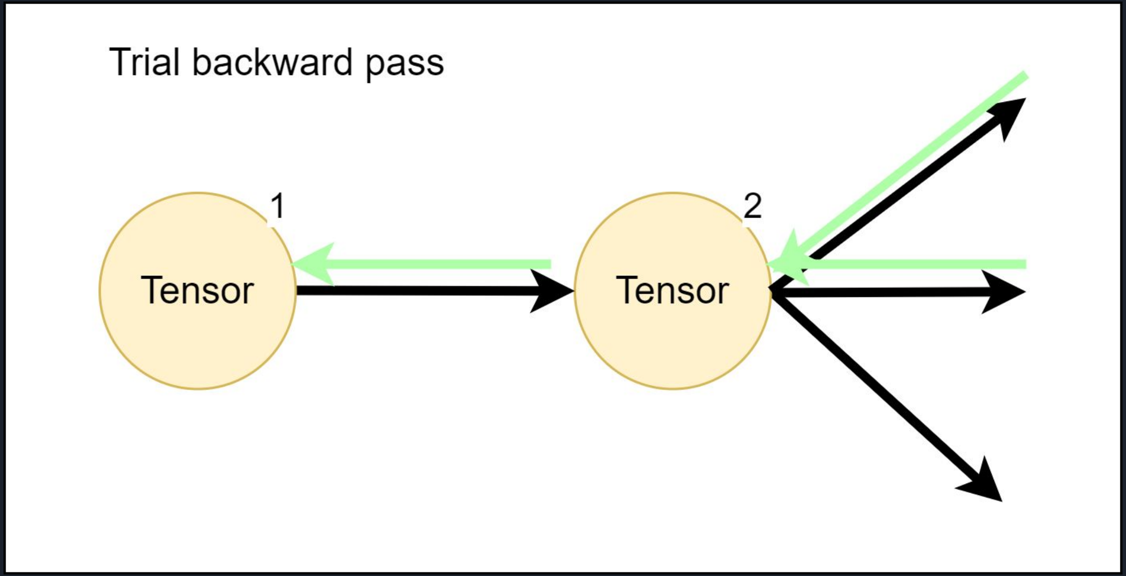
On trial, only forward connections are counted. No gradients passing



Solution Algorithm

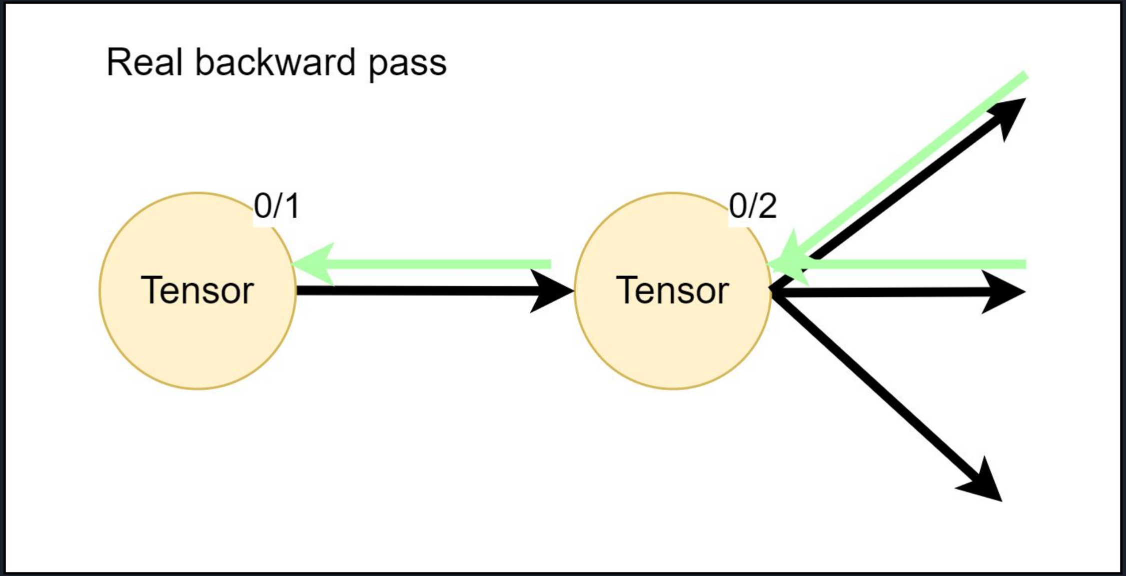


Solution Algorithm

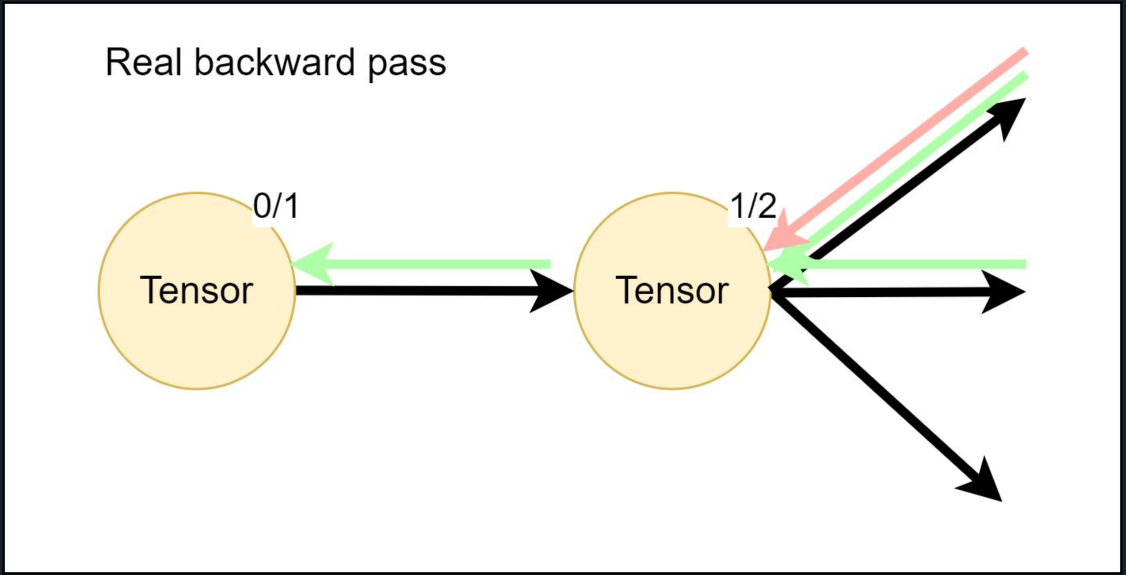


Solution Algorithm

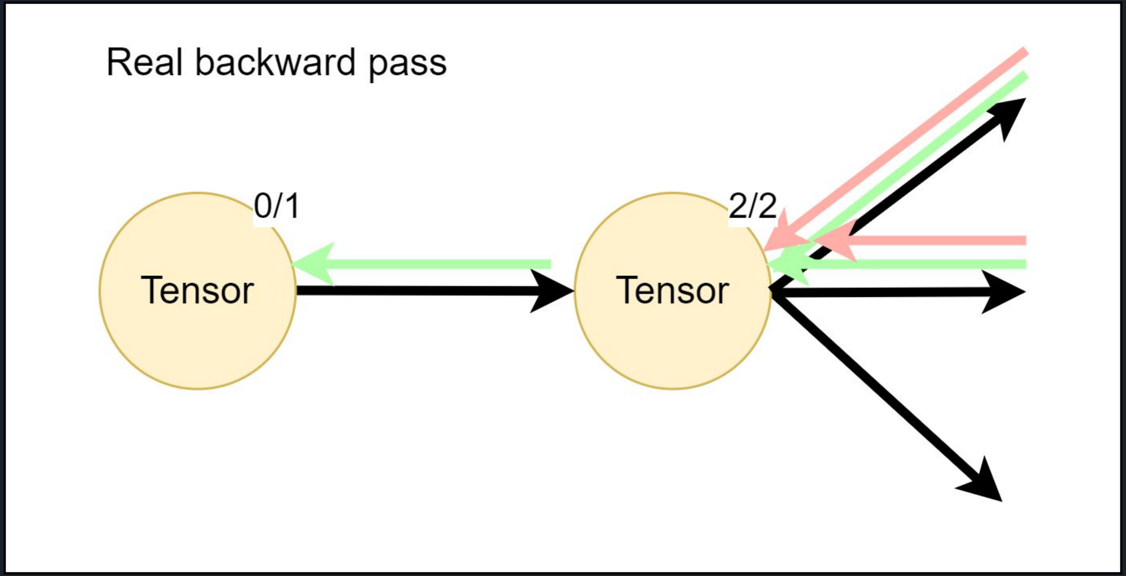
On real pass, gradients are summed until saturation.



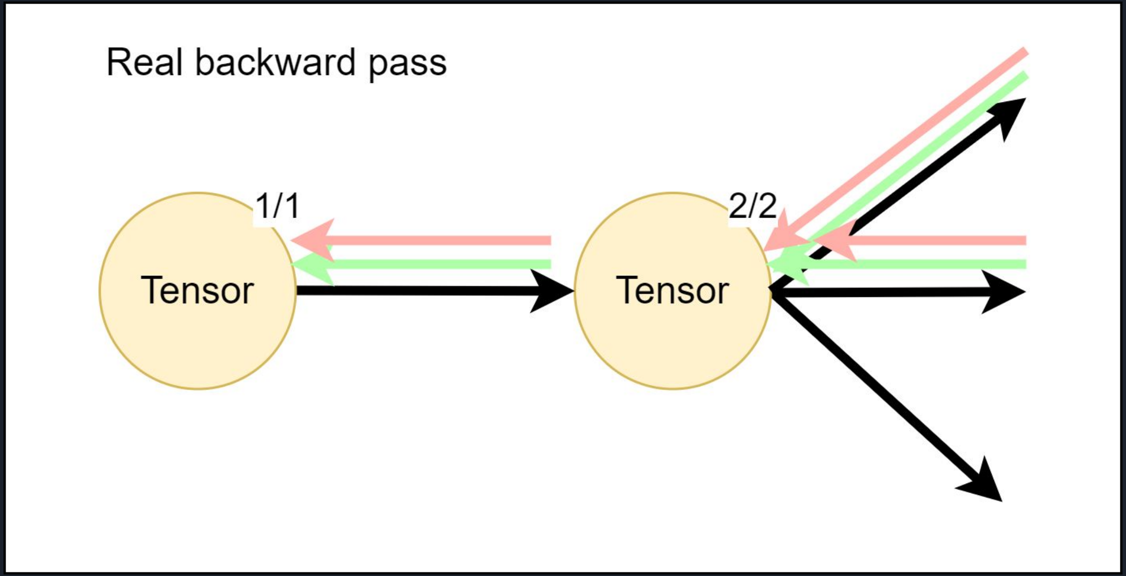
Solution Algorithm



Solution Algorithm



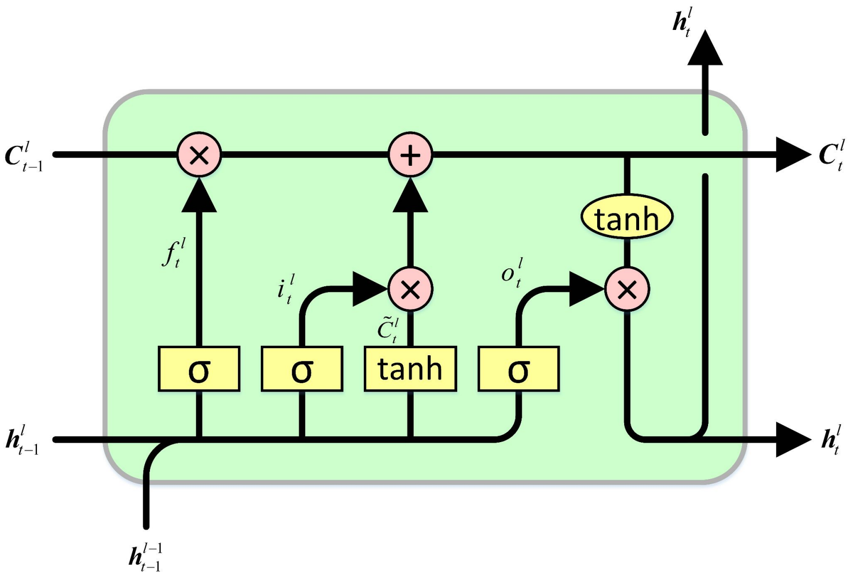
Solution Algorithm

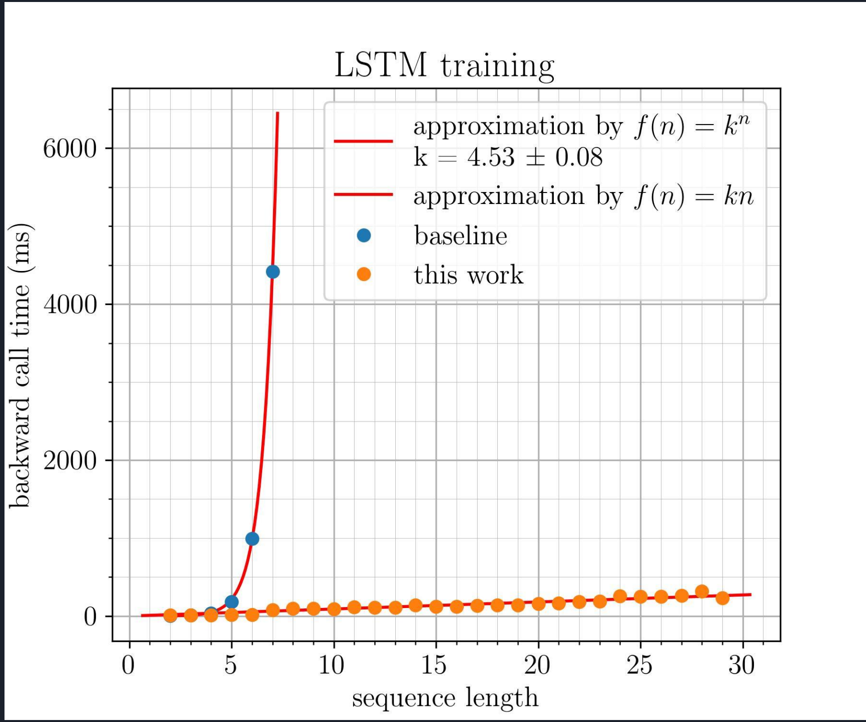


Code modification

```
class Linear(Function):
+    def backward(self, grad: np.ndarray, trial_pass):
         dW = np.dot(self.x.data.T, grad)
         db = grad.sum(axis=0)
         grad = np.dot(grad, self.W.data.T)
+        self.W.backward(dW.reshape(self.W.shape), trial_pass)
+        self.b.backward(db.reshape(self.b.shape), trial_pass)
+        self.x.backward(grad.reshape(self.x.shape), trial_pass)
```

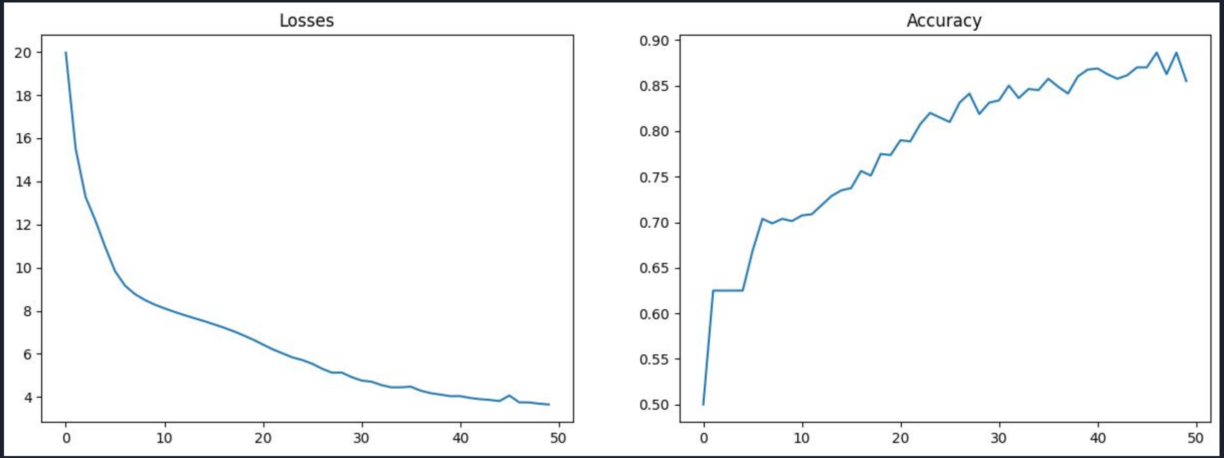
Experimental Performance

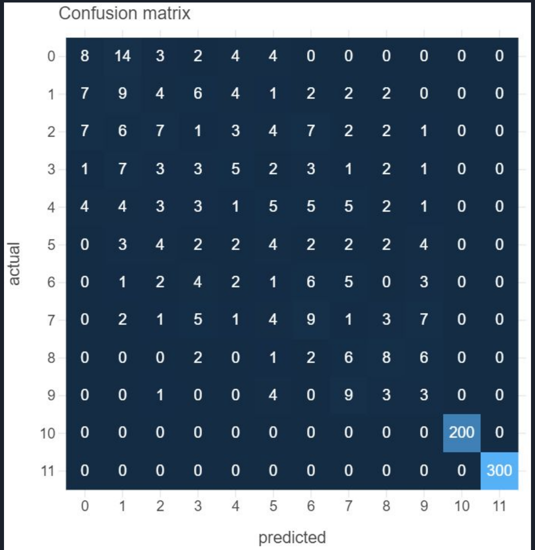




Correctness Check

1. Gradients checks pass
2. LSTM trains on list sorting task





Conclusion

New fast Deep Learning Framework is available in PyPI (yet another)

```
pip install fast-deep-rnn==0.0.2
```