Progress Report 3

Surya Shanmugaselvam | 12/7/2021

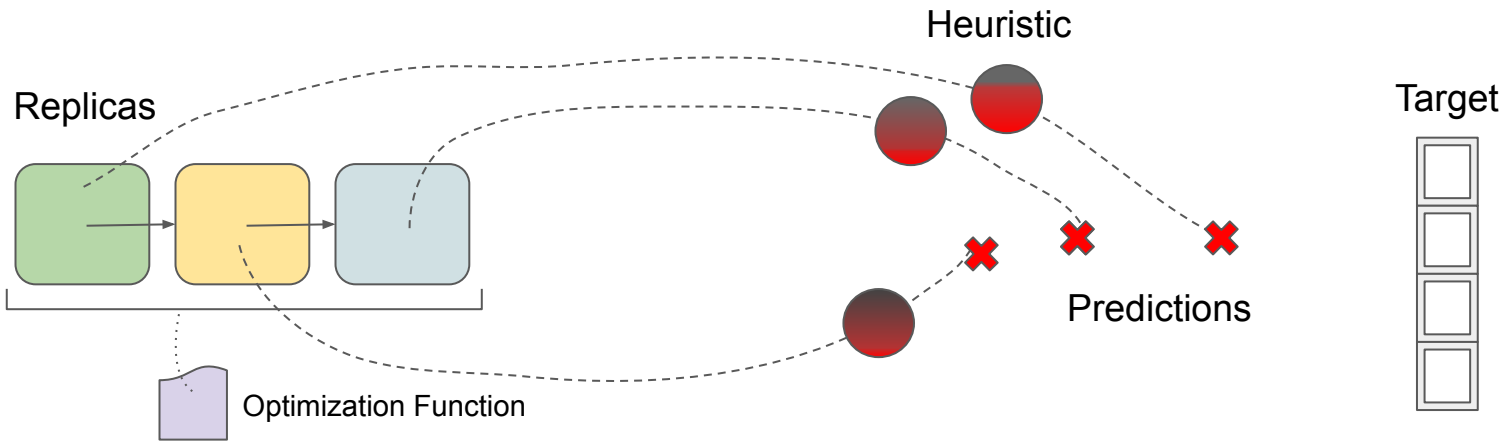
Recap of Previous Presentation Ideas

- Bayesian Search Space Optimizer (Hyperparameter tuning)
- LSTM Layer for intermediate layer(s) - effective backpropagation by “forgetting” any “un-improving” weights

- Weight Averaging - allowing best models to pave the way for newer models
- Reinforcement and Heuristic Learning approach (AlphaZero inspired probabilistic models and/or Pheromone Optimization)

Concept: Heuristic Learning and “Minion” Optimization

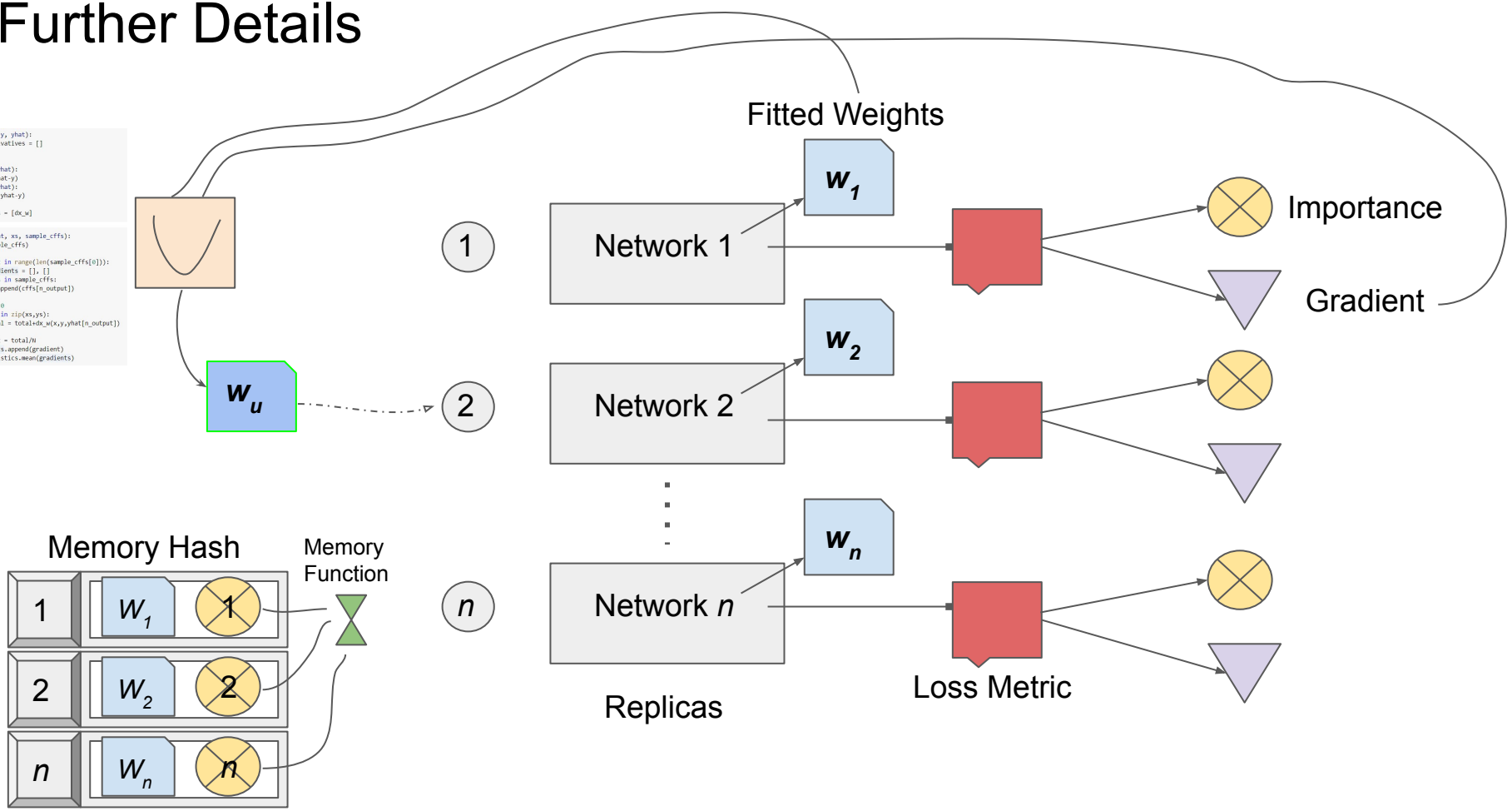
- Band of “minion” networks constructed to subsample of fraction of the larger dataset and validate their predictions
- Short-term memory of the the weights for each network replica and its performance against the actual dataset
- Probabilistic optimization function to find next weights to start the subsequent “minion”/replica model at:



Further Details

```
def derivate(x, y, yhat):
    partial_derivatives = []


def dx_b(x, y, yhat):
    return 2*(yhat-y)
def dx_w(x, y, yhat):
    return 2*x*(yhat-y)

derivative_funcs = [dx_w]
```

```
def gradient(yhat, xs, sample_cffs):
    N = len(sample_cffs)

    for n_output in range(len(sample_cffs[0])):
        ys, gradients = [], []
        for cffs in sample_cffs:
            ys.append(cffs[n_output])

        total = 0
        for x,y in zip(xs,ys):
            total = total+dx_w(x,y,yhat[n_output])

        gradient = total/N
        gradients.append(gradient)
    return statistics.mean(gradients)
```



Further Details to Approach

- *n* number of replicas are created for each subsampling of the BVCS dataset and each replica is trained over a number of *e* epochs
  - n=30, e=1000
- Each replica is fitted and Compton Form Factors are extracted from the layer with error and magnitude of distance from the target computed.
- Gradient Descent Optimizer determines the gradient of the cost function and determines the next set of weights to start the next iteration
  - Compute the residual for each Compton Form Factor prediction
  - The sum of the squares of the residuals acts as our cost function $C(<e_{ReH},e_{ReE},e_{ReHt}>)$
  - Partial derivative of our cost function for each CFF provides a gradient. Magnitude of gradient provides the expected rate of improvement in loss.
  - Learning Rate (0.01 chosen) multiplied by the rate of improvement provides the factor for weight update.
- Distance from target determines the “importance” → “importance” determine which weights in previous iterations to hop to if cost function’s gradient does not improve over 5 iterations.

Future Work

- Get distributions using new algorithm - already ran algorithm and convergence of error is occurring but had runtime disconnects during the week
- Implement more robust gradient descent-based optimizer for starting weight update.
- Implement process-based parallelism in which “importance” function can act more like a pheromone concentration function to direct models to improve upon previous models that have gotten closer to the target CFFs.