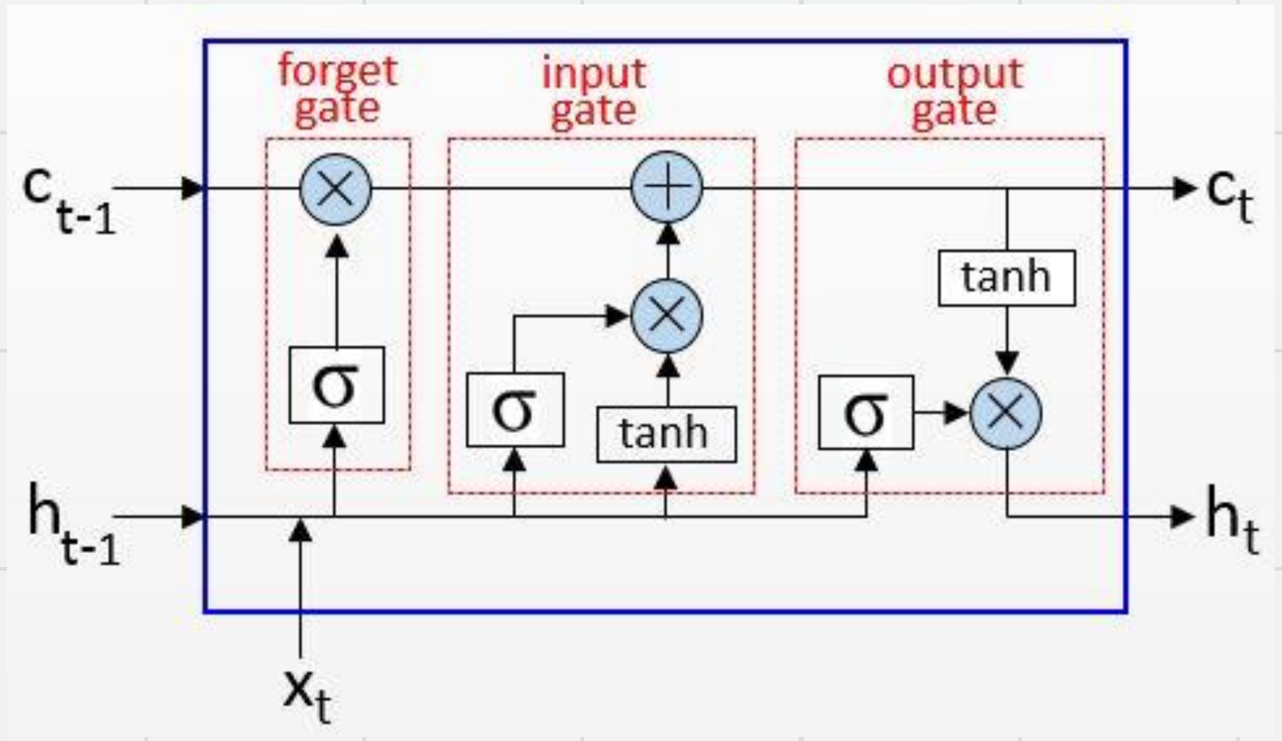
A Novel Approach to Optimizing LSTM Models Using the L-BFGS Algorithm: An Experimental Study

APPLIED MATHEMATICS AND INTELLIGENT COMPUTATION

Final project
report

Originally, this paper was about optimizing the hyperparameters of an LSTM model using genetic algorithms. That was my project for the GA class. However, to adapt this project for this class, we decided to also optimize the model parameters by substituting the Adam optimizer with L-BFGS. Our goal was to see if this algorithm could achieve at least the same performance as Adam, not in runtime but in minimizing the error during the training and validation phases.

What is a LSTM?



An LSTM, or Long Short-Term Memory network, is a type of recurrent neural network (RNN) designed to recognize patterns in sequences of data, such as time series or natural language**. Unlike standard RNNs, LSTMs can remember information for long periods of time**, which makes them particularly effective at capturing long-term dependencies.

Research Aim and Strategy

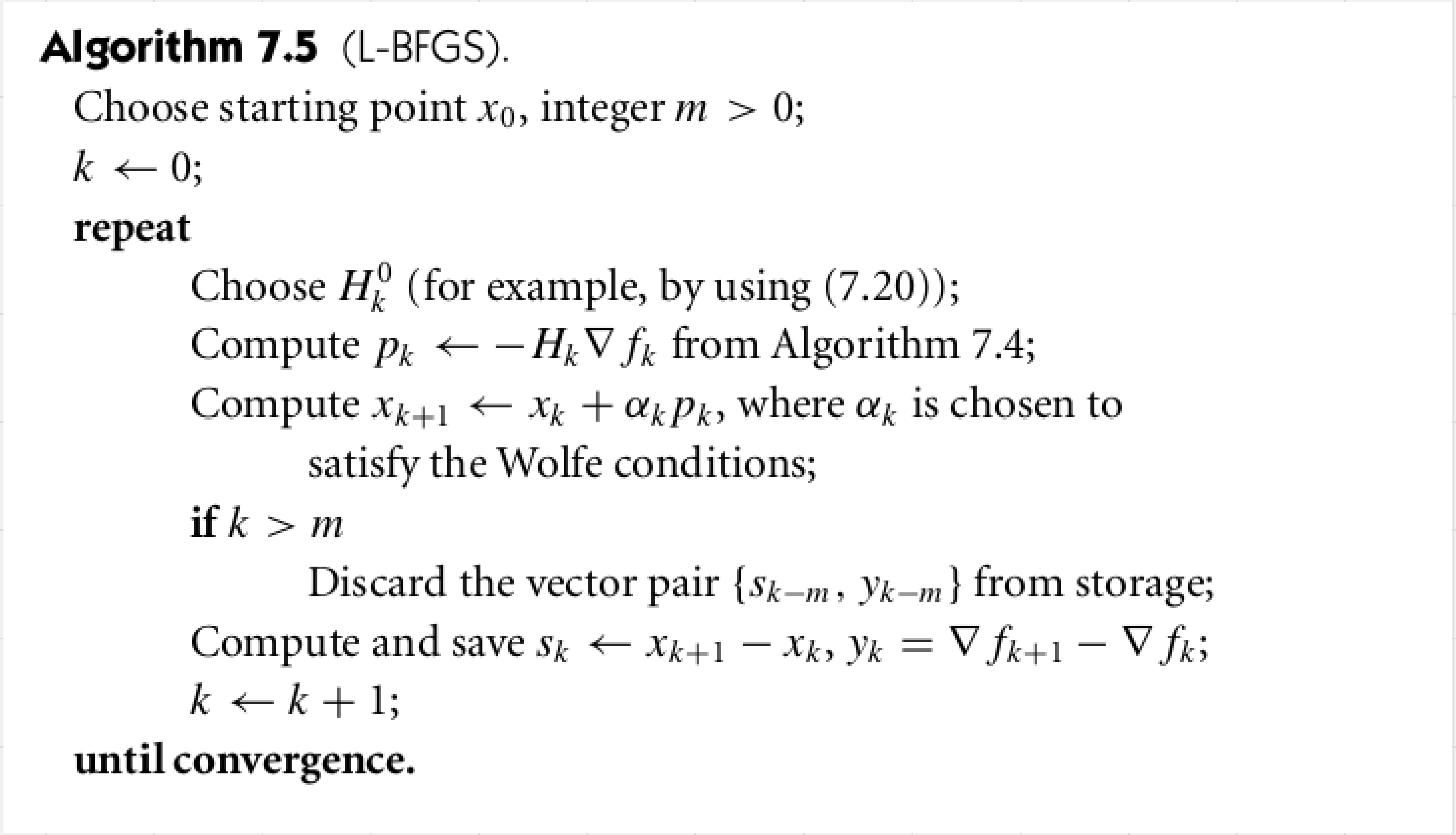
- The idea is to **substitute the Adam optimizer with L-BFGS** and see if this algorithm can achieve at least the same or better results than Adam in minimizing the error during the training and validation phases..

What is the L-BFGS Algorithm and How Does It Work?

The L-BFGS algorithm originates from the BFGS algorithm, a well-known method for solving **unconstrained optimization problems**. The BFGS algorithm is named after its developers: **Broyden, Fletcher, Goldfarb, and Shanno.** It is an iterative method that approximates the Hessian matrix (second-order derivatives) to guide the optimization process efficiently. BFGS is particularly effective because it uses both **gradient** and **curvature information** to find the minimum of a function. However, **BFGS has a significant drawback**: it requires storing and manipulating large matrices, making it impractical for high-dimensional problems where memory is limited.

L-BFGS is an extension of BFGS that addresses the memory issue by using a limited amount of memory. Instead of storing the entire Hessian matrix**, L-BFGS maintains only a few vectors that represent the approximation**. This makes L-BFGS much **more scalable** and suitable for large-scale optimization problems.

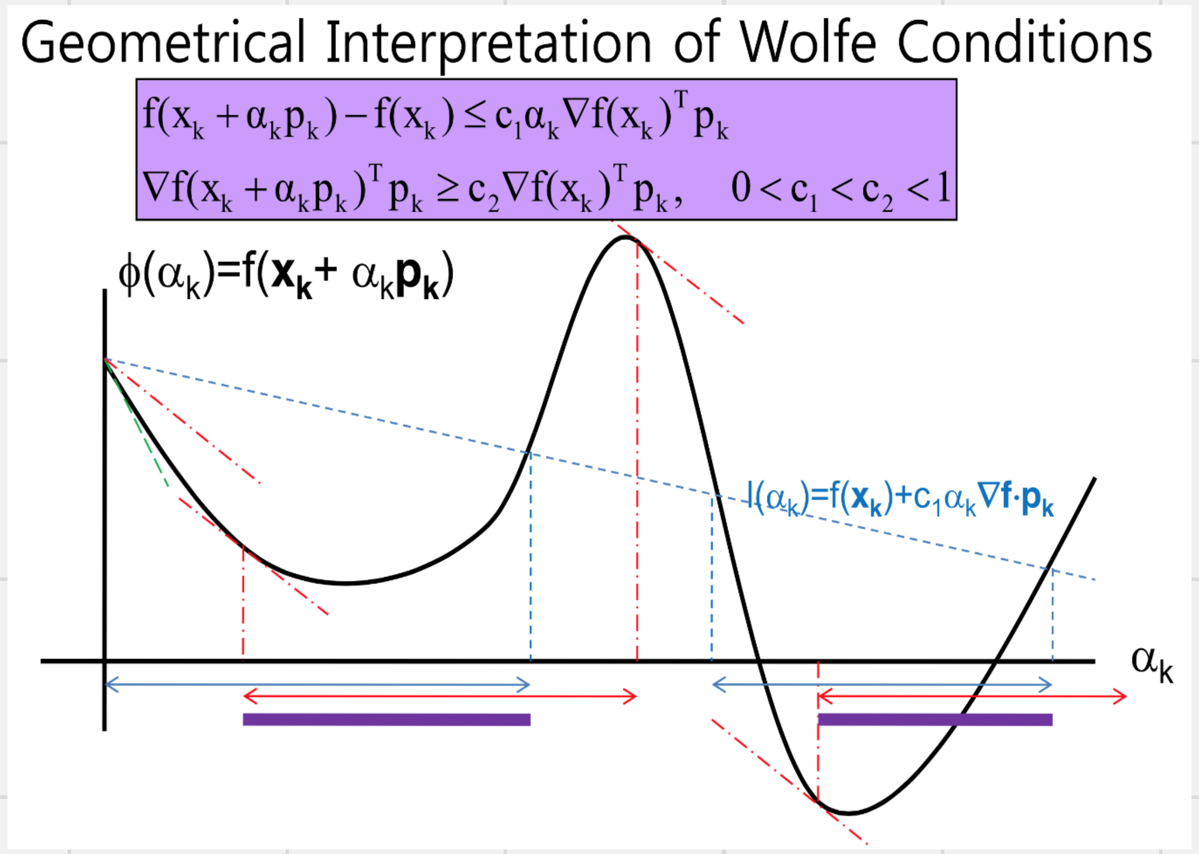
L-BFGS Pseudocode



Algorithm 7.5 (L-BFGS).

```
Choose starting point $x_0$, integer $m > 0$;
$k \leftarrow 0$;
repeat
    Choose $H_k^0$ (for example, by using (7.20));
    Compute $p_k \leftarrow -H_k \nabla f_k$ from Algorithm 7.4;
    Compute $x_{k+1} \leftarrow x_k + \alpha_k p_k$, where $\alpha_k$ is chosen to
        satisfy the Wolfe conditions;
    if $k > m$
        Discard the vector pair $\{s_{k-m}, y_{k-m}\}$ from storage;
    Compute and save $s_k \leftarrow x_{k+1} - x_k$, $y_k = \nabla f_{k+1} - \nabla f_k$;
    $k \leftarrow k + 1$;
until convergence.
```

Wolfe Conditions


Geometrical Interpretation of Wolfe Conditions
$f(x_k + \alpha_k p_k) - f(x_k) \le c_1 \alpha_k \nabla f(x_k)^T p_k$
$\nabla f(x_k + \alpha_k p_k)^T p_k \ge c_2 \nabla f(x_k)^T p_k, \quad 0 < c_1 < c_2 < 1$
$\phi(\alpha_k) = f(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$
$l(\alpha_k) = f(\mathbf{x}_k) + c_1 \alpha_k \nabla \mathbf{f} \cdot \mathbf{p}_k$
α_k

1. Armijo condition(Sufficient Decrease Condition):

This condition ensures that the step size results in a sufficient decrease in the objective function value. It prevents the algorithm from taking **too large steps** that could overshoot the minimum.

2. Curvature condition:

This condition ensures that the slope of the objective function has decreased sufficiently. It guarantees that the **step size is not too small**, thus avoiding overly cautious steps that could slow down the optimization process.

✓ By satisfying both conditions, **the algorithm adaptively adjusts the step size (α_k) in each iteration**, ensuring efficient and stable convergence to the minimum of the objective function.

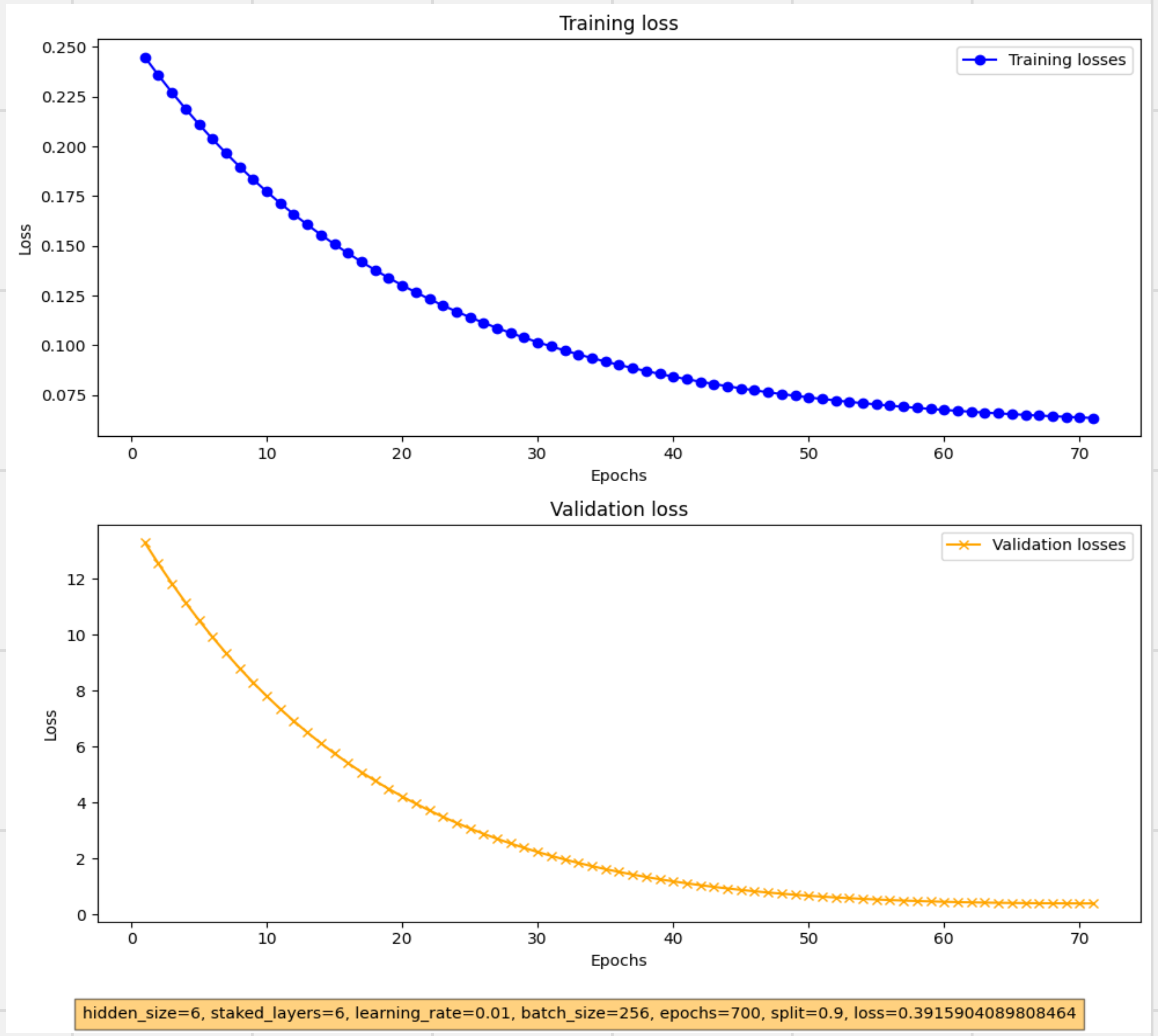
Experiments and results

✓ ***I used GA to find good hyperparameters for LSTM model***

LSTM Training hyperparameters

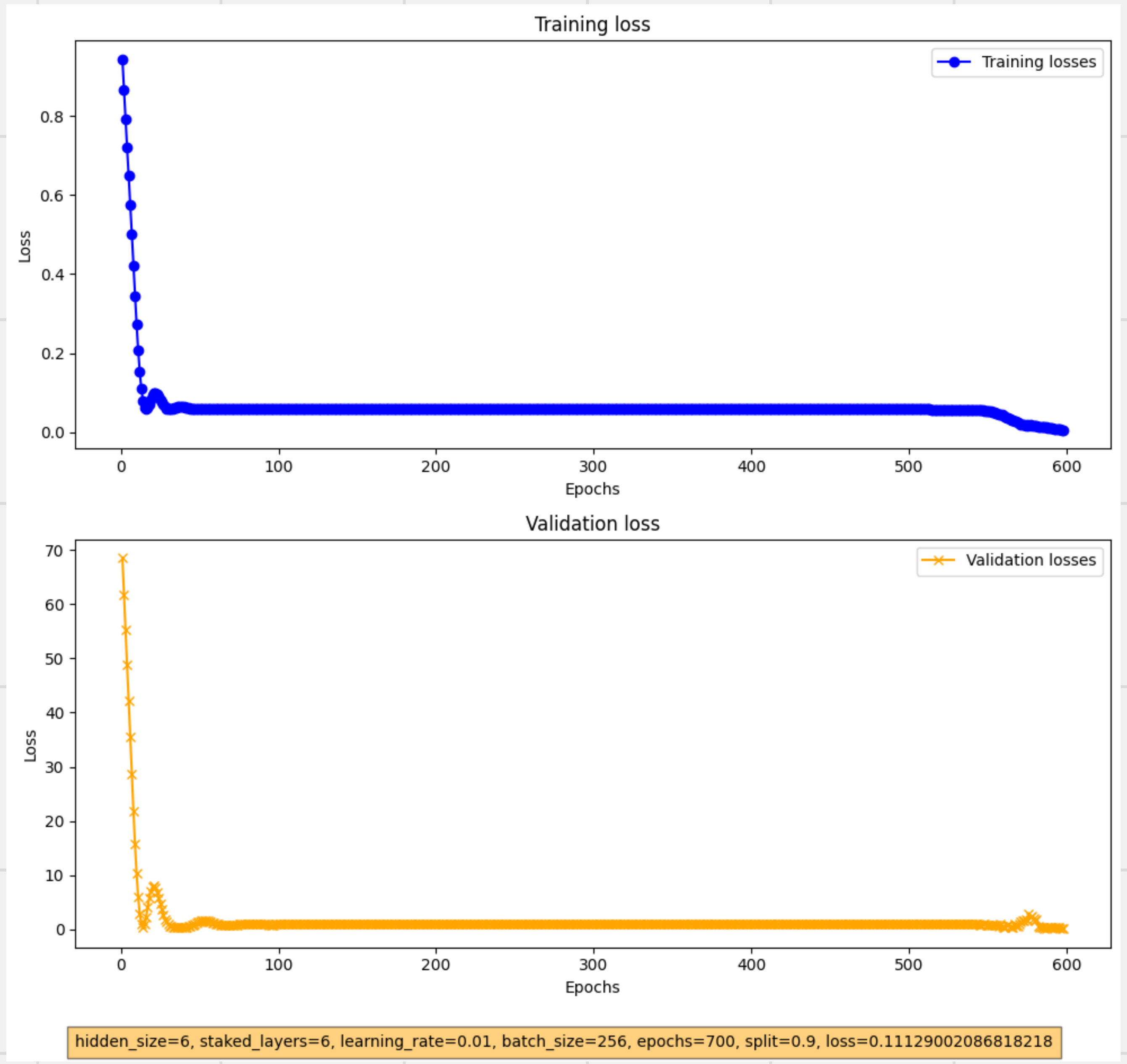
- **hidden size**=6
- **staked layers**=6
- **learning rate**=0.01
- **batch size**=256
- **epochs**=700
- **split**=0.9

L-BFGS best result in 10 runs



hidden_size=6, staked_layers=6, learning_rate=0.01, batch_size=256, epochs=700, split=0.9, loss=0.3915904089808464

Adam best result in 10 runs



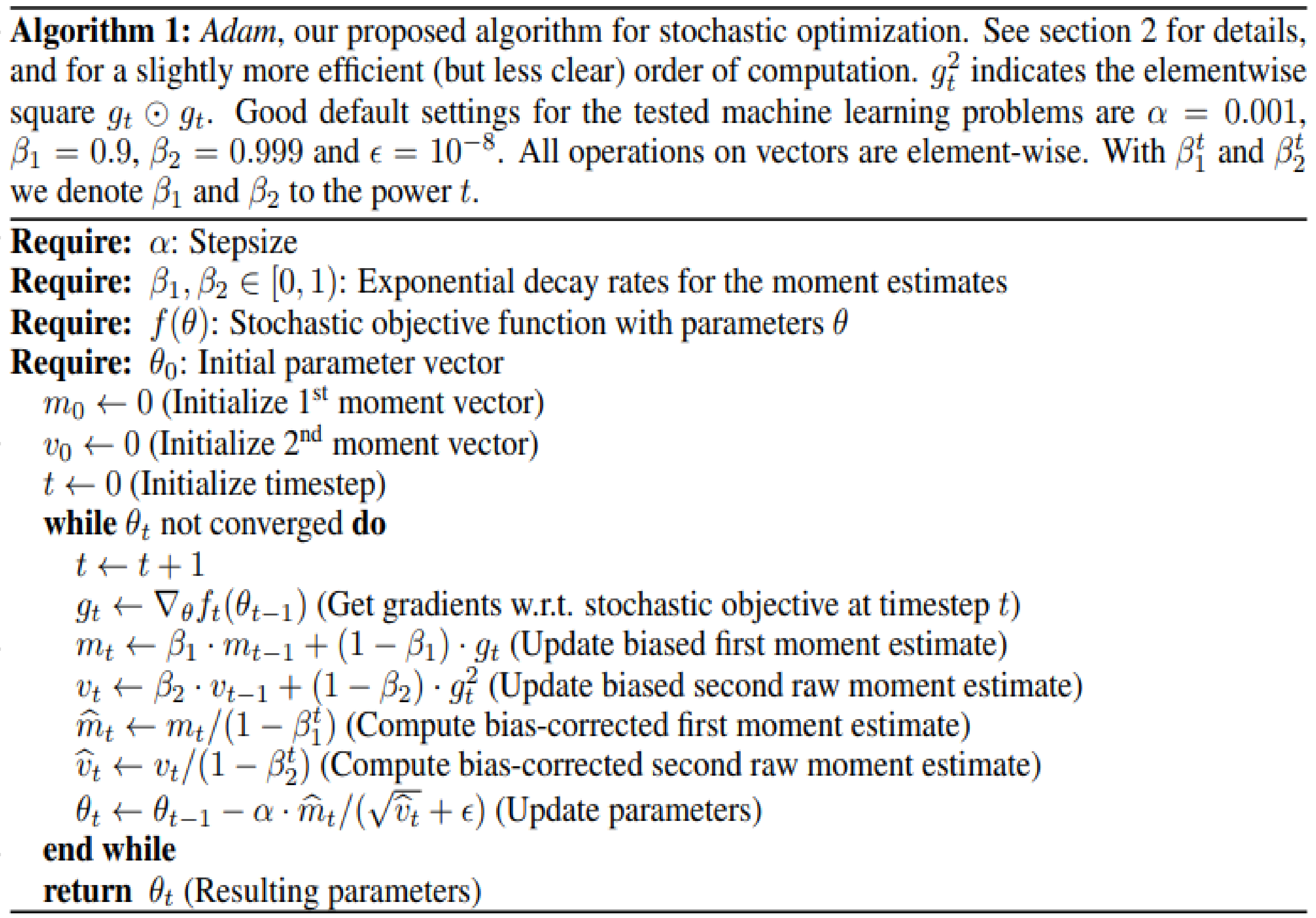
hidden_size=6, staked_layers=6, learning_rate=0.01, batch_size=256, epochs=700, split=0.9, loss=0.11129002086818218

Mean loss in 10 runs

Algorithm	Mean Loss	Standard Deviation
Adam	0.152377	0.023634
L-BFGS	0.391656	0.000047

Why Doesn't L-BFGS Perform as Well as Adam?

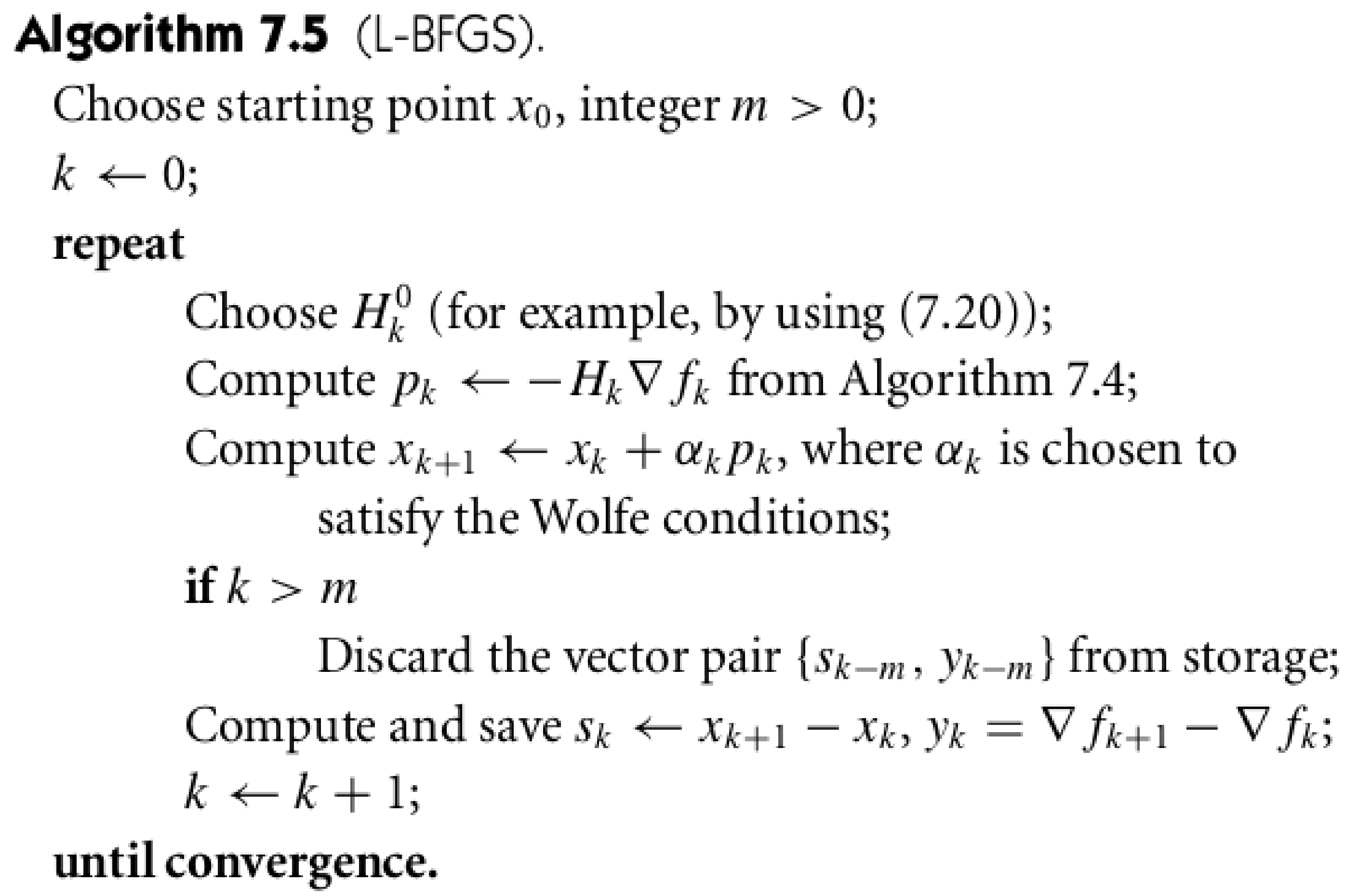
Adam



Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

L-BFGS



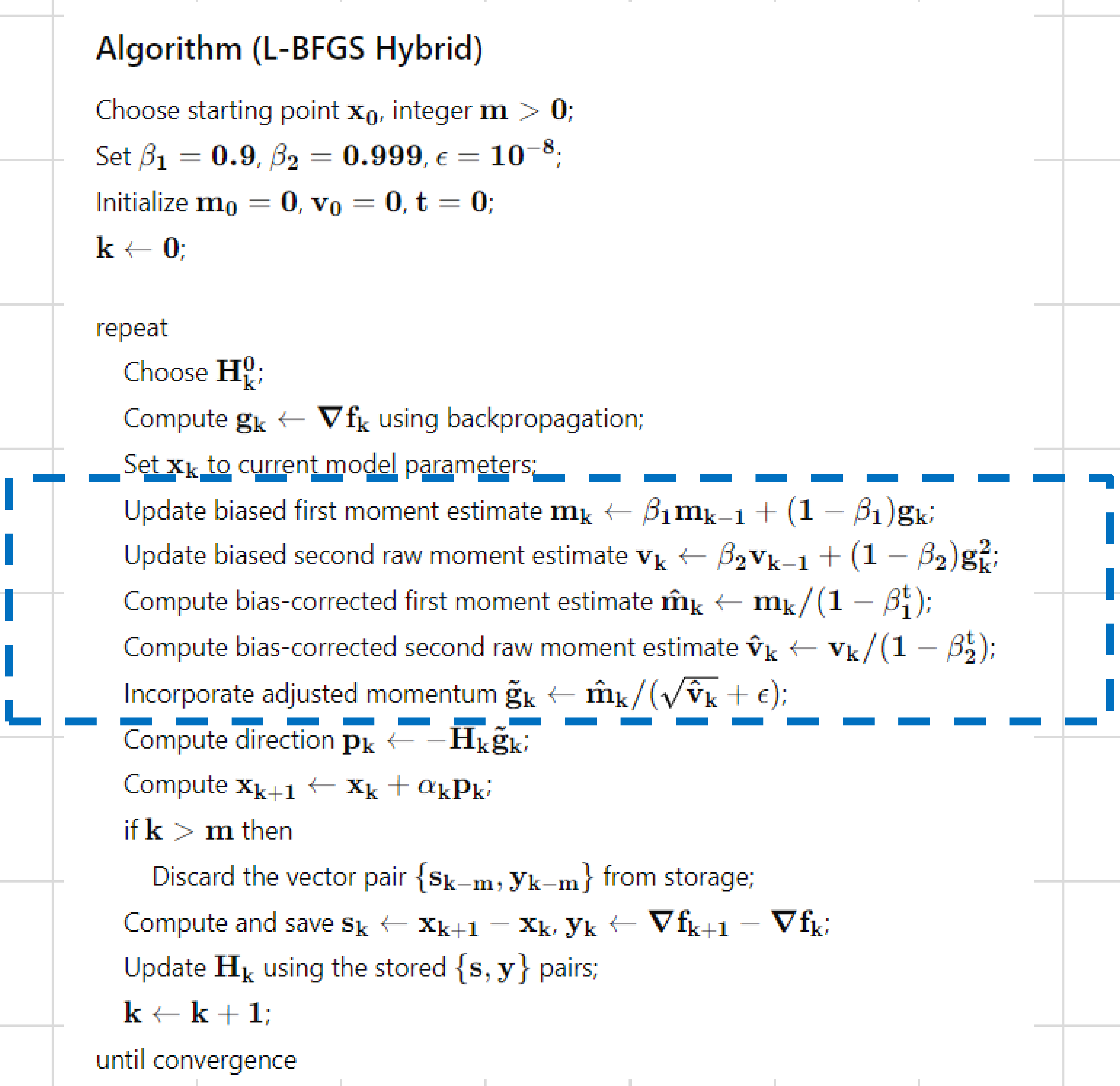
Algorithm 7.5 (L-BFGS).

```
Choose starting point $x_0$, integer $m > 0$;
$k \leftarrow 0$;
repeat
    Choose $H_k^0$ (for example, by using (7.20));
    Compute $p_k \leftarrow -H_k \nabla f_k$ from Algorithm 7.4;
    Compute $x_{k+1} \leftarrow x_k + \alpha_k p_k$, where $\alpha_k$ is chosen to
        satisfy the Wolfe conditions;
    if $k > m$
        Discard the vector pair $\{s_{k-m}, y_{k-m}\}$ from storage;
    Compute and save $s_k \leftarrow x_{k+1} - x_k$, $y_k = \nabla f_{k+1} - \nabla f_k$;
    $k \leftarrow k + 1$;
until convergence.
```

- **Adam**: Computes gradients **g_t** and updates **biased first moment estimates m_t** and **second moment estimates V_t** ***(to adapt the learning rate dynamically).***
- **Adam:** Adapts the learning rate α using **bias-corrected moment estimates** $\hat{m}_t$ and $\hat{V}_t$ ***(to adjust step sizes based on the historical gradient information).***
- **Adam**: Updates the parameters **θ_t** by subtracting the product of the learning rate α and the bias-corrected first moment estimate divided by the square root of the bias-corrected second moment estimate ***(to efficiently converge to the minimum).***

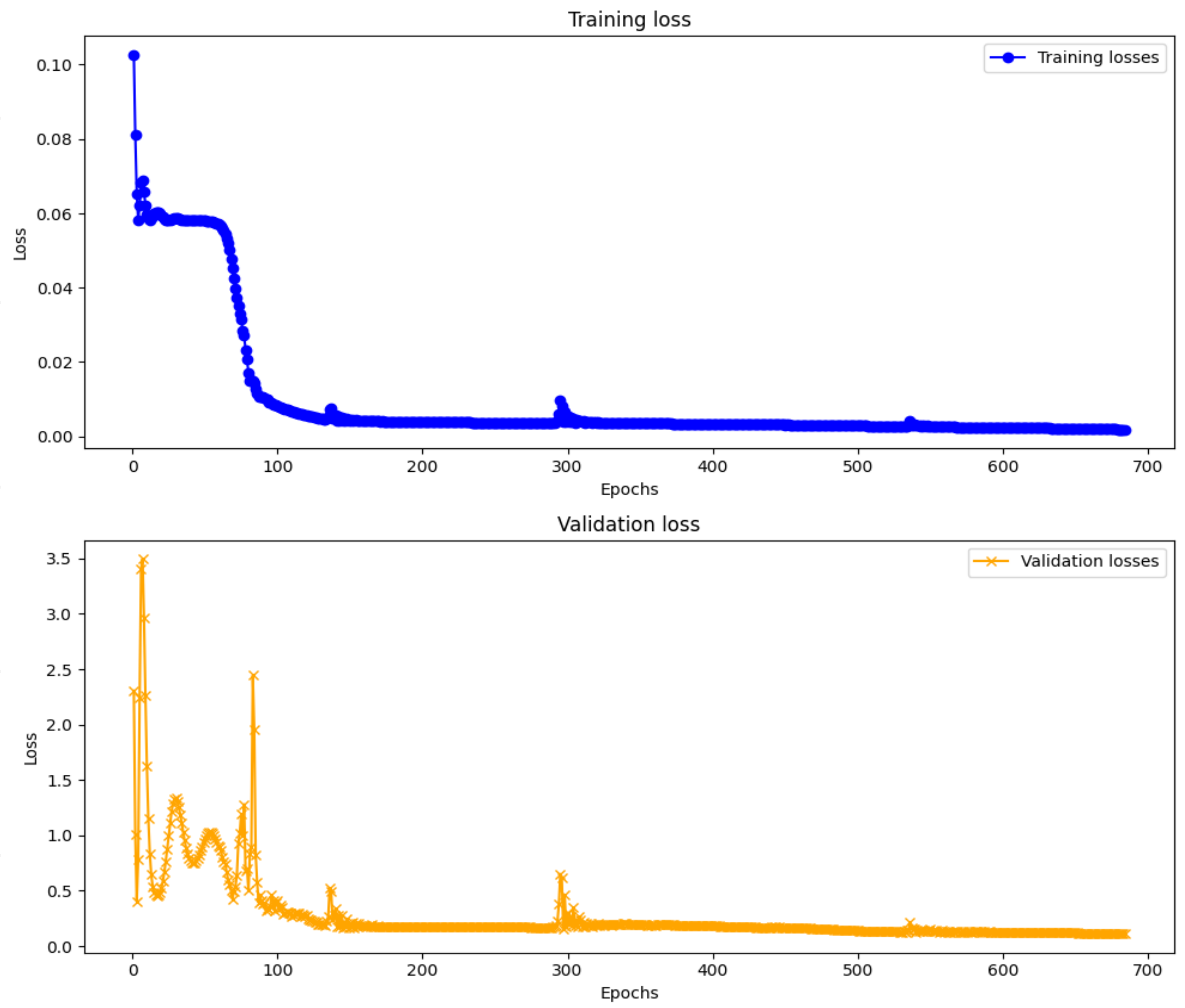
- **L-BFGS**: Computes the search direction **P_k** using an **approximation of the inverse Hessian H_k** and the gradient **∇f_k** ***(to find an optimal search direction).***
- **L-BFGS**: Selects a step size **α_k** to satisfy **Wolfe conditions** (***to ensure sufficient decrease and curvature conditions).***
- **L-BFGS**: Updates the parameters **X_{k+1}** by adding the product of the step size **α_k** and the search direction **P_k** ***(to iteratively move towards the minimum).***

L-BFGS Hybrid pseudocode



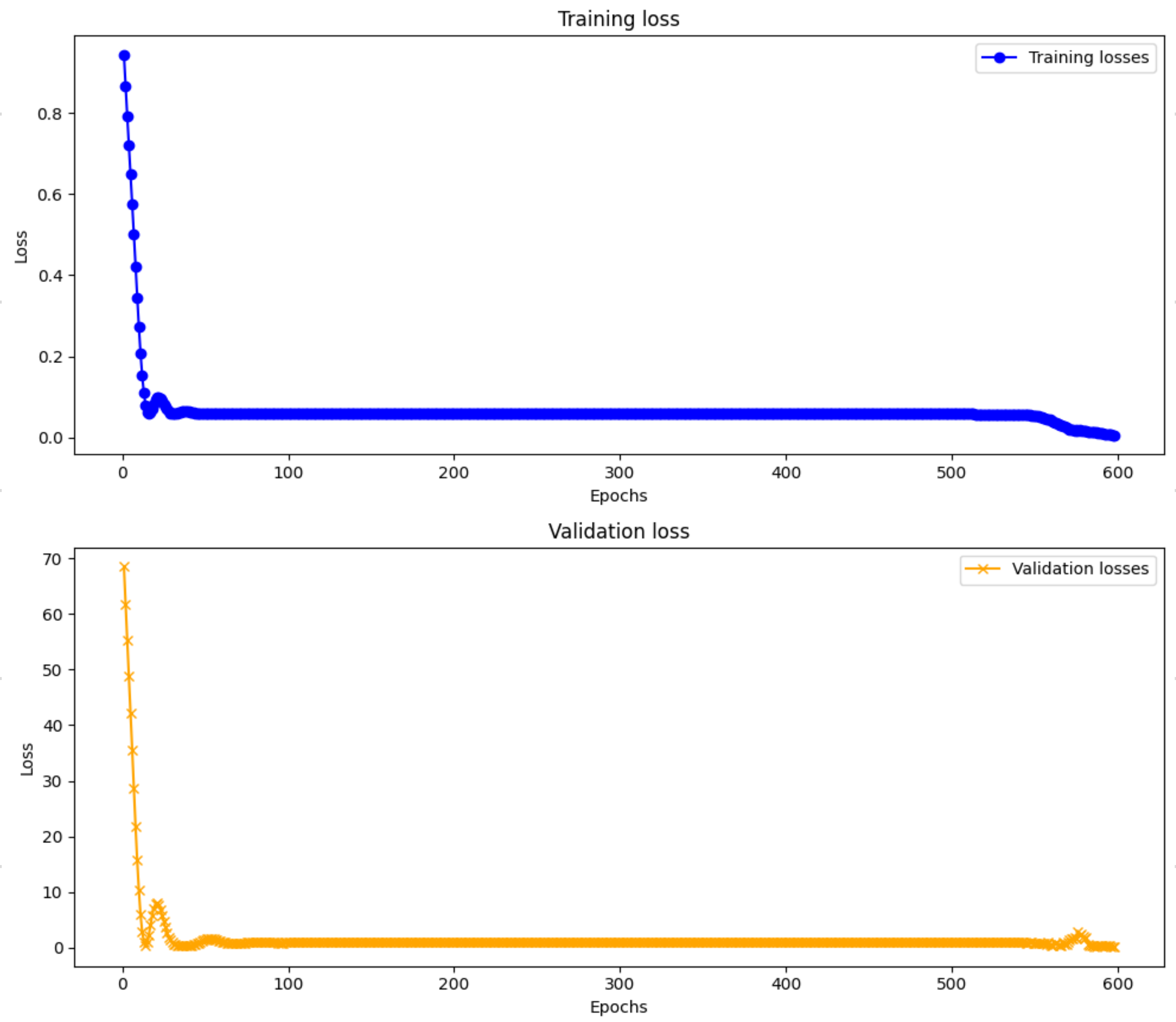
```
Algorithm (L-BFGS Hybrid)
Choose starting point $\mathbf{x_0}$, integer $\mathbf{m} > \mathbf{0}$;
Set $\beta_1 = \mathbf{0.9}$, $\beta_2 = \mathbf{0.999}$, $\epsilon = \mathbf{10^{-8}}$;
Initialize $\mathbf{m_0} = \mathbf{0}$, $\mathbf{v_0} = \mathbf{0}$, $\mathbf{t} = \mathbf{0}$;
$\mathbf{k} \leftarrow \mathbf{0}$;

repeat
    Choose $\mathbf{H_k^0}$;
    Compute $\mathbf{g_k} \leftarrow \nabla \mathbf{f_k}$ using backpropagation;
    Set $\mathbf{x_k}$ to current model parameters;
    Update biased first moment estimate $\mathbf{m_k} \leftarrow \beta_1 \mathbf{m_{k-1}} + (\mathbf{1} - \beta_1)\mathbf{g_k}$;
    Update biased second raw moment estimate $\mathbf{v_k} \leftarrow \beta_2 \mathbf{v_{k-1}} + (\mathbf{1} - \beta_2)\mathbf{g_k^2}$;
    Compute bias-corrected first moment estimate $\hat{\mathbf{m}}_\mathbf{k} \leftarrow \mathbf{m_k}/(\mathbf{1} - \beta_1^\mathbf{t})$;
    Compute bias-corrected second raw moment estimate $\hat{\mathbf{v}}_\mathbf{k} \leftarrow \mathbf{v_k}/(\mathbf{1} - \beta_2^\mathbf{t})$;
    Incorporate adjusted momentum $\tilde{\mathbf{g}}_\mathbf{k} \leftarrow \hat{\mathbf{m}}_\mathbf{k}/(\sqrt{\hat{\mathbf{v}}_\mathbf{k}} + \epsilon)$;
    Compute direction $\mathbf{p_k} \leftarrow -\mathbf{H_k}\tilde{\mathbf{g}}_\mathbf{k}$;
    Compute $\mathbf{x_{k+1}} \leftarrow \mathbf{x_k} + \alpha_\mathbf{k}\mathbf{p_k}$;
    if $\mathbf{k} > \mathbf{m}$ then
        Discard the vector pair $\{\mathbf{s_{k-m}}, \mathbf{y_{k-m}}\}$ from storage;
    Compute and save $\mathbf{s_k} \leftarrow \mathbf{x_{k+1}} - \mathbf{x_k}$, $\mathbf{y_k} \leftarrow \nabla \mathbf{f_{k+1}} - \nabla \mathbf{f_k}$;
    Update $\mathbf{H_k}$ using the stored $\{\mathbf{s}, \mathbf{y}\}$ pairs;
    $\mathbf{k} \leftarrow \mathbf{k} + \mathbf{1}$;
until convergence
```

L-BFGS Hybrid best result in 10 runs



hidden_size=6, staked_layers=6, learning_rate=0.01, batch_size=256, epochs=700, split=0.9, loss=0.11124442098662257

Adam best result in 10 runs



hidden_size=6, staked_layers=6, learning_rate=0.01, batch_size=256, epochs=700, split=0.9, loss=0.11129002086818218

Mean loss in 10 runs

Algorithm	Mean Loss	Standard Deviation
Adam	0.152377	0.023634
L-BFGS Hybrid	0.130885	0.008659

Conclusions

- Adam's performance is much better than L-BFGS in this experiment because Adam updates the parameters by **adapting the learning rate for each parameter**. It calculates the mean of the **decay of the gradients *(first moment)*** and the **variance of the decay of the gradients *(second moment)*.** In contrast, L-BFGS uses Wolfe conditions, **which in this experiment, did not prove to be an effective method for updating the learning rate in each iteration.**
- **L-BFGS Hybrid** is an adaptation that incorporates **Adam's first moment and second moment** strategies instead of **Wolfe conditions** to guide the optimization towards the global minimum.
- Although **L-BFGS Hybrid** demonstrates **good and stable results**, **it does not necessarily mean it is better than Adam**. L-BFGS Hybrid requires **additional calculations for the Hessian matrix**, making the algorithm l**ess suitable for scaling to more complex models with a higher number of parameters**. Therefore, to improve its performance, further investigations and experiments are needed.

THANK YOU