Review of Research - Paper n°5

Ahmed Yassine Chraa, Pauline Zhou, Emma de Charry

version: April 2, 2025

1 Introduction

1.1 Summary of the paper

The paper "Network Newton Distributed Optimization Methods" introduces a novel approach to distributed optimization that significantly improves upon traditional first-order methods. The authors Aryan Mokhtari, Qing Ling, and Alejandro Ribeiro address the fundamental challenge of minimizing a sum of convex functions distributed across network nodes where each node(or agent) only has access to its local function and can only communicate with neighboring nodes as we have seen in the course. The key innovation lies in incorporating second-order information through a distributed approximation of Newton steps, the approximation is achieved by truncating the Taylor series expansion of the exact Newton step. This creates a family of methods (NN-K) where K represents the number of terms kept in the approximation, with larger K values requiring communication across K-hop neighborhoods but providing closer approximations to the exact Newton step. In the paper, compared to first-order methods like Decentralized Gradient Descent (DGD), Network Newton demonstrates substantially faster convergence, particularly for ill-conditioned problems, while maintaining the distributed nature of the computation.

1.2 Problem formulation

The paper addresses the problem of distributed optimization over a network of n agents. Each agent i holds a local objective convex function $f_i : \mathbb{R}^p \to \mathbb{R}$, and the objective is to solve the global optimization problem:

$$\min_{x\in\mathbb{R}^p} f(x) = \min_{x\in\mathbb{R}^p} \sum_{i=1}^{n} f_i(x), \tag{1}$$

An alternative and equivalent formulation introduces local variables x_i for each node where the network is represented by neighborhoods $\mathcal{N}_i$ containing nodes that can communicate directly with node i

$$\min_{\{x_i\}} \sum_{i=1}^{n} f_i(x_i), \quad \text{s.t. } x_i = x_j, \quad \forall (i,j) \in \mathcal{N}_i. \tag{2}$$

The constraint $x_i = x_j$ for neighboring nodes ensures that all local variables converge to the same value in connected networks, making this formulation equivalent to the original problem.

The standard approach, **Decentralized Gradient Descent (DGD)** as we have seen in course follows the update rule:

$$x_i^{t+1} = \sum_{j\in\mathcal{N}_i} w_{ij} x_j^t - \alpha \nabla f_i(x_i^t), \tag{3}$$

where w_{ij} are weights that are positive only when j is a neighbour of i or $j = i$, α is the step size and $t = 1, 2, \ldots$ is the number of iterations The DGD update can be written in the following vectors form:

$$y_{t+1} = Zy_t - \alpha h(y_t) = y_t - [(I - Z)y_t + \alpha h(y_t)]$$

.

Where $y = [x_1; ...; x_n] \in \mathbb{R}^{np}$ concatenates all local variables, $h(y) = [\nabla f_1(x_1); ...; \nabla f_n(x_n)]$ concatenates local gradients, and Z = $W \bigotimes I$ combines the weight matrix W with the identity matrix through a Kronecker product.

Which can be seen as a Gradient Descient algorithm applied to function $\mathbf{F}(y) = \frac{1}{2}y^T(I - Z)y + \alpha \sum_{i=1}^{n} f(x_i)$

The idea **Network Newton (NN)** method is to apply a Newton's method to function $\mathbf{F}(y)$

So basically, we will follow the update rule :

$$y_{t+1} = y_t + \varepsilon d_t$$

Where $d_t = -\mathrm{H}_t^{-1}\mathrm{g}_t$ with H_t **and** g_t **are the Hessian matrix and gradient vector of** $\mathbf{F}(y)$

The main issue is that for many reasons (computational, nature of problem) the Hessian matrix is not always invertible. To overcome this problem, the authors thought about splitting the diagonal and off diagonal blocks of H_t and rely on a Taylor's expansion of the inverse.

Let's write

$$\begin{aligned} \mathrm{H}_t = \mathrm{D}_t - \mathrm{B}, \quad & \text{where :} \\ & \mathrm{D}_t = \alpha \mathrm{G}_t + 2(\mathrm{I} - \mathrm{diag}(\mathrm{Z})) = \alpha \mathrm{G}_t + 2(\mathrm{I} - \mathrm{Z}_\mathrm{d}) \\ & \mathrm{B} = \mathrm{I} - 2\mathrm{Z}_\mathrm{d} + \mathrm{Z} \end{aligned}$$

where G_t is the diagonal elements of the Hessian

And then we can factor $\mathrm{D}_t^{1/2}$ in the expression : $\mathrm{H}_t = \mathrm{D}_t^{1/2}(\mathrm{I} - \mathrm{D}_t^{-1/2}\mathrm{B}\mathrm{D}_t^{-1/2})\mathrm{D}_t^{1/2}$.

Now we can use the Taylor series expansion $(\mathrm{I} - \mathrm{X})^{-1} = \sum_{\mathrm{k}=0}^{\infty} \mathrm{X}^\mathrm{k}$:

$$\mathrm{H}_t^{-1} = \mathrm{D}_t^{1/2} \sum_{\mathrm{k}=0}^{\infty} (\mathrm{D}_t^{-1/2}\mathrm{B}\mathrm{D}_t^{-1/2})^\mathrm{k}$$

The sum converge iff matrix $\mathrm{D}_t^{-1/2}\mathrm{B}\mathrm{D}_t^{-1/2}$ *has eighenvalues in* $[-1, 1]$

The K-approximation idea in the paper is taking only K terms in the sum to approximate $\mathbf{H}_t^{-1}$

$$\mathrm{H}_t^{(\mathrm{K})^{-1}} = \mathrm{D}_t^{1/2} \sum_{\mathrm{k}=0}^{\mathrm{K}} (\mathrm{D}_t^{-1/2}\mathrm{B}\mathrm{D}_t^{-1/2})^\mathrm{k}$$

The update rule becomes then :

$$y_{t+1} = y_t + \varepsilon \mathrm{H}_t^{(\mathrm{K})^{-1}} \mathrm{g}_t = y_t + \varepsilon \mathrm{d}_\mathrm{K}$$

We can compute d_K iteratively by setting $\mathrm{d}_0 = -\mathrm{D}_t^{-1}\mathrm{g}_t$ and $\mathrm{d}_{k+1} = \mathrm{D}_t^{-1}(\mathrm{B}\mathrm{d}_\mathrm{k} - \mathrm{g}_t)$

The whole algoithm is :

Algorithm 1 Network Newton-K method in vectors format

Initialize: y_0, W
$Z = W \otimes I_m$
$Z_d = \text{diag}(Z_{1,1}, Z_{2,2}, \ldots, Z_{m,m})$
$B = I - 2Z_d + Z$
for $t = 1, 0, \ldots$ **do**
  $D_t = \alpha \nabla^2 f(y) + 2(I - Z_d)$
  $g_t = (I - Z)y + \alpha \nabla f(y)$
  $d_0 = -D_t^{-1} g_t$
  for $j = 0$ to K **do**
    $d_j = Z d_j$ ▷ Communication step
    $d_{j+1} = D_t^{-1}(Bd - g_t)$
  end for
  $y_{t+1} = y_t + \alpha d_K$
end for
return y^*

Algorithm 2 Network Newton-K method (node-wise)

Initialize: $y_{i,0}, w_{ij}$ weights for all nodes i
Define $\mathcal{N}_i$ as neighbors of node i
B matrix blocks : $\mathrm{B_{ii}} = (1 - w_{ii})I$ and $\mathrm{B_{ij}} = w_{ij}I$
for $t = 1, 0, \ldots$ **do**
  $D_{ii,t} = \alpha \nabla^2 f_i(x_{i,t}) + 2(1 - w_{ii})I$
  $g_{i,t} = (1 - w_{ii})x_{i,t} - \sum_{j \in \mathcal{N}_i} w_{ij} x_{j,t} + \alpha \nabla f_i(x_{i,t})$
  $d_0^i = -D_{i,t}^{-1} g_{i,t}$
  for $j = 0$ to $K - 1$ **do**
    Send d_j^i to all neighbors
    Receive d_j^p from all neighbors
    $\mathrm{d_{j+1}^i = D_{ii,t}^{-1}} \left[\sum_{\mathrm{p} \in \mathcal{N}_\rangle, \mathrm{p=i}} \mathrm{B_{ip} d_j^p} - \mathrm{g_{i,t}} \right]$
  end for
  $x_{i,t+1} = x_{i,t} + \alpha \mathrm{d}_K^i$
end for
return x_i^*

This approach retains second-order curvature information while remaining computationally feasible for distributed settings.

1.3 Assumptions

To derive the theoretical results, the paper relies on the following key assumptions:

1. **Network Connectivity:** The communication network is represented as a connected, undirected graph $G = (V, E)$, where each node i can exchange information only with its neighbors $j \in \mathcal{N}_i$. This ensures that all agents can collectively optimize the function.

2. **Weight Matrix Properties:** The network is characterized by a symmetric and doubly stochastic mixing matrix $W \in \mathbb{R}^{n \times n}$ with entries w_{ij}, satisfying:

$$W1 = 1, \quad 1^T W = 1^T, \quad \texttt{null}(I - W) = \texttt{span}(1). \tag{4}$$

   This ensures consensus among nodes.

3. **Convexity and Smoothness:** Each local function $f_i(x)$ is convex and has a Lipschitz-continuous gradient:

$$\|\nabla f_i(x) - \nabla f_i(y)\| \leq L_i \|x - y\|, \quad \forall x, y \in \mathbb{R}^p, \tag{5}$$

   where L_i is the Lipschitz constant.

4. **Strong Convexity (for Linear Convergence):** The global function $f(x) = \sum_{i=1}^n f_i(x)$ is strongly convex with parameter $\mu > 0$, meaning:

$$\langle \nabla f(x) - \nabla f(y), x - y \rangle \geq \mu \|x - y\|^2, \quad \forall x, y \in \mathbb{R}^p. \tag{6}$$

   This guarantees linear convergence rates.

5. **Lipschitz Continuity of the Hessian:** The Hessians $\nabla^2 f_i(x)$ are Lipschitz continuous with parameter L_H, ensuring stability of second-order approximations:

$$\|\nabla^2 f_i(x) - \nabla^2 f_i(y)\| \leq L_H \|x - y\|, \quad \forall x, y \in \mathbb{R}^p. \tag{7}$$

6. **Bounded Eigenvalues:** The Hessians satisfy:

$$0 < m \leq \lambda_{\min}(\nabla^2 f_i(x)) \leq \lambda_{\max}(\nabla^2 f_i(x)) \leq M < \infty. \tag{8}$$

   This ensures Newton-like updates remain well-conditioned.

7. **Hessian Approximation:** A fundamental assumption is that the Hessian inverse can be effectively approximated using a truncated Taylor series expansion. This approximation becomes more accurate as more terms are included (larger K values) and only possible if matrix $\mathrm{D_t^{-1/2}BD_t^{-1/2}}$ has eighenvalues in $[-1, 1]$

These assumptions collectively enable the Network Newton method to approximate Newton steps in a distributed manner. The compatibility between the network structure and the Hessian approximation's sparsity pattern is particularly important, as it allows information aggregation to remain localized within K-hop neighborhoods.

2 Result: theory and practice

2.1 Theory

The Network Newton method introduces a family of algorithms (NN-K) that approximate Newton steps through K terms of a Taylor series expansion. The theoretical analysis yields several important results:

Theorem 1 establishes that NN-K converges at a rate that is at least linear to the optimal argument of the penalized objective function. Specifically, for sufficiently small step sizes, the algorithm satisfies:

$$\|y_{t+1} - y^*\| \leq \gamma \|y_t - y^*\| \tag{9}$$

where $\gamma < 1$ is a contraction factor and y^* is the optimizer of the penalized objective. This guarantee ensures that the iterates approach the solution at a predictable rate.

A key insight from the analysis is that the error between the Hessian inverse approximation utilized by NN-K and the actual inverse Hessian decays exponentially with K:

$$\|D^{-1} - D^{-1} \sum_{k=0}^{K} (I - D^{-1}H)^k\| \leq C \cdot \rho^{K+1} \tag{10}$$

where D is a diagonal approximation of the Hessian H, C is a constant, and $\rho < 1$. This exponential decay explains why even small values of K can produce significant improvements over first-order methods.

Theorem 2 reveals that NN-K exhibits a quadratic convergence phase for iterations beyond the initial few. For iterations within a specific interval whose length depends on K, the weighted gradient norm follows:

$$\|\nabla \tilde{f}(y_{t+1})\|_H^{-1} \leq \beta \cdot (\|\nabla \tilde{f}(y_t)\|_H^{-1})^2 + \epsilon K \tag{11}$$

where β is a constant and ϵK represents the error due to the Hessian approximation, which decreases as K increases. This quadratic convergence phase is a distinctive feature of Newton-like methods and explains the superior performance of NN-K compared to DGD.

The theoretical analysis identifies a fundamental trade-off in the algorithm design: larger penalty coefficients accelerate convergence but increase the distance between the optimal solutions of the original and penalized objectives. Similarly, increasing K improves the Hessian approximation and extends the quadratic convergence phase but requires more extensive communication.

In most cases tested bu authors, the NN-K outperformed the DGD algorithm. It converges faster than DGD and sometimes gives better results.

2.2 Practice

The code we built is in the folder with the Kernel project, as we needed to use some functions from the `utils.py` file and to compare with DGD.

2.2.1 Gradient and Hessian Computation

To apply the NN-K method, we first need to derive the gradient and Hessian of our objective function. Given the quadratic nature of our problem, the Hessian matrix remains constant throughout the optimization process.

The gradient of the objective function with respect to α is:

$$\nabla f_a(\alpha) = \frac{\sigma^2}{5} K_{mm}\alpha - \sum_{i \in \mathcal{A}} K_{(i)m}^T (y_i - K_{(i)m}\alpha) + \frac{\nu}{5}\alpha \tag{12}$$

The Hessian matrix is given by:

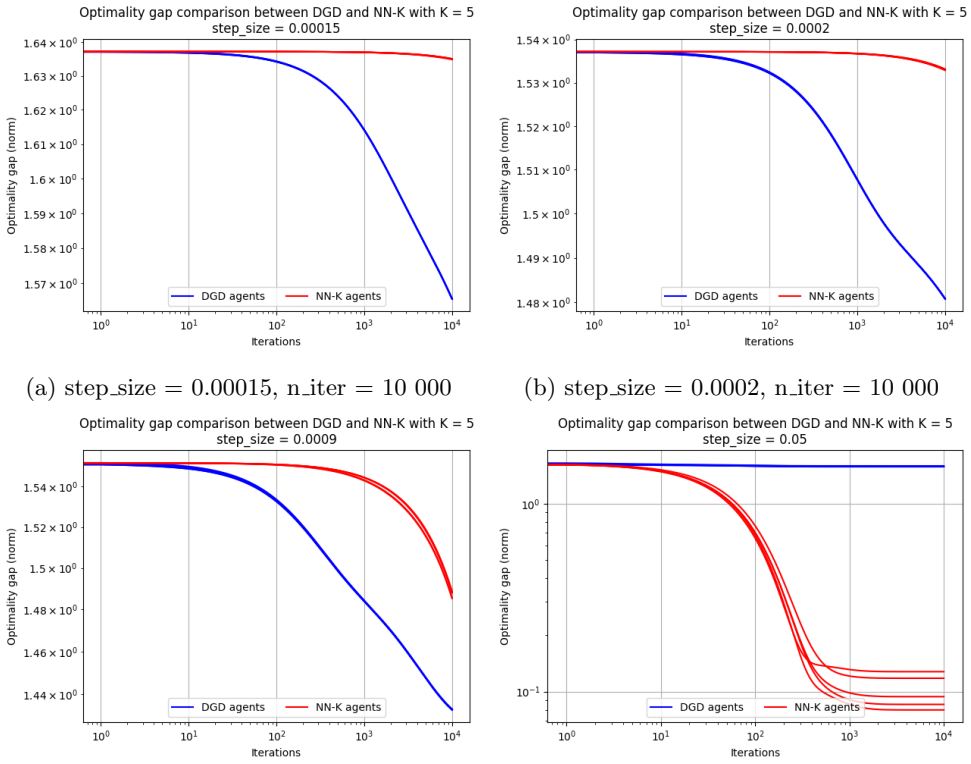
$$\nabla^2 f_a(\alpha) = \frac{\sigma^2}{5} K_{mm} + \sum_{i \in \mathcal{A}} K_{(i)m}^T K_{(i)m} + \frac{\nu}{5} I \tag{13}$$

$$\nabla f(\alpha) = [\nabla f_1(\alpha), \dots \nabla f_5(\alpha)]^T$$

$$\text{H} \quad \text{is a diagonal block matrix with blocks} \quad H_{ii} = \nabla^2 f_a(\alpha) \quad \text{for} \quad a = 1, \dots 5$$

Since our problem is quadratic, the Hessian matrix remains constant throughout the optimization process, which theoretically should benefit second-order methods like NN-K.

2.2.2 Comparaison avec DGD



(a) step_size = 0.00015, n_iter = 10 000

(b) step_size = 0.0002, n_iter = 10 000

(c) step_size = 0.0009, n_iter = 10 000

(d) step_size = 0.05, n_iter = 10 000

Figure 1: Comparaison of gaps between DGD and NN-5 for different step sizes

We implemented the NN-K algorithm as described in Algorithm 1 and applied it to our distributed kernel ridge regression problem. For comparison, we also implemented the Distributed Gradient Descent (DGD) method from the course.

The results show that at low step sizes, the NN-5 algorithm is so slow compared to the DGD and hence doesn't have the best performance before 10000 iterations and the optimality gap stays very big. But when we increase the step size to 0.05 NN-K converges faster than DGD (which seems to not converging event after 1000 iterations) and reaches better optimality gap (around 0.08)

2.2.3 Convergence Behavior of NN-K

One notable observation was that the NN-K method required significantly higher learning rates to converge efficiently. While DGD and other first-order methods typically required very small learning rates (around 0.0002) to ensure stability, NN-K performed best with much larger learning rates (0.1, 0.05, 0.2). This aligns with the theoretical insights from the original paper, which suggests that quadratic problems benefit from higher learning rates when using second-order methods.

2.2.4 Approximation K impact

After seeing the impact of changing the step size and how it make the NN-K convgerges faster. Let's analyse how the approximation depth K impacts the performance oh the NN on the kernel problem. We took **step size = 0.05** for the following figures since we found that taking higher values doesn't improve much the performance.

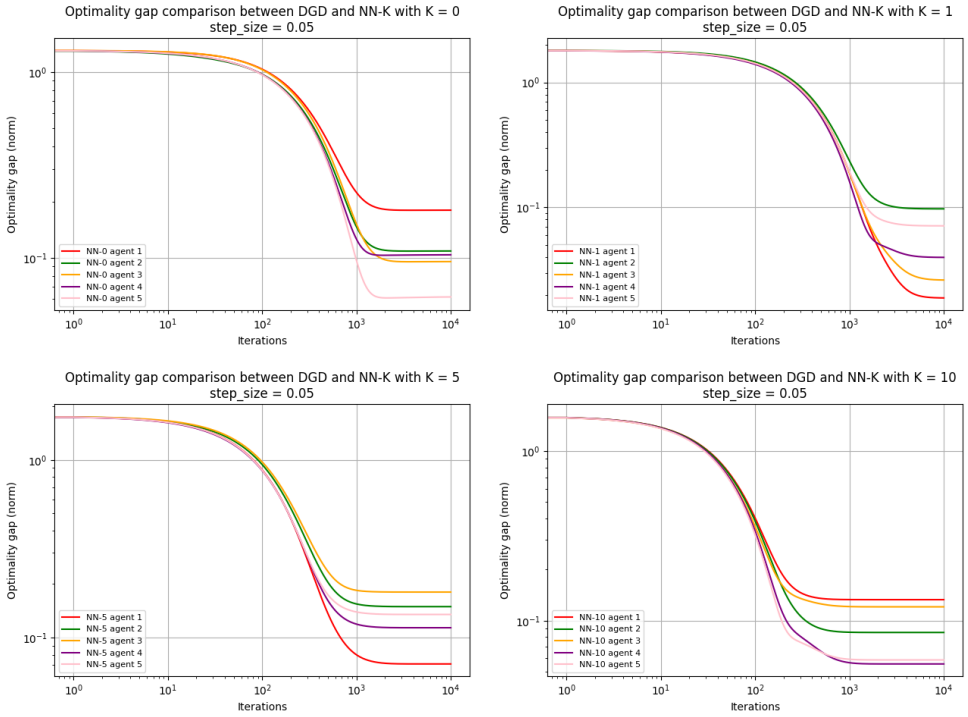


Figure 2: Optimal gap for each agent of NN-K method for different values of K

We run the algorithm on the problem with different values of K (K = 0,1,5,10). First of all we can see that **the best optimal gap is with K = 1**, and we don't really improve the gap by increasing the precision degree. Also, the convergence speed is higher when K is increased. For example with K = 1, the algorithm converged after 1000 iterations while with K = 10 it converges only after 100 iterations. This is an expected result and it was mentioned in the paper, since by increasing K with add more informations to the update step and hence it need less iterations to converge but increases the computational cost (actually in this example the number of iterations is even the same because with K= 10 we do $100 \times 10 = 1000$ iterations)

2.2.5 Other remarks

This NN-K method is still a 2nd order optimization technique. So, it incorporates the Hessian matrix, 2nd derivatives and the need to inverse some matrix. This allows the method to potentially converge faster and more accurately than first-order methods in many scenarios. However, the performance of this method can be sensitive to the choice of initial point. The starting value of α can significantly

influence the convergence behavior and the quality of the final solution. Here we were always starting from $\alpha = [0, \ldots, 0]$ so when testing with other points the method may give different results or even diverge.

3 Conclusion

We have explored the application of the Network Newton-K method to kernel regression problem. Our experiments confirm the theoretical advantages of second-order methods for quadratic problems, particularly the benefit of using higher learning rates with NN-K compared to first-order methods like DGD. The results demonstrate that NN-K achieves superior convergence when properly tuned, reaching small optimality gaps (around 0.08) with higher step sizes (0.05), still the performance of variations of NN are not significantly better than DGD. Interestingly, we found that increasing the approximation depth K beyond K=1 primarily accelerates convergence in terms of outer iterations rather than improving the final optimality gap. The NN-K method effectively balances local computation with network communication, demonstrating how second-order information can be leveraged in a distributed setting to overcome the limitations of first-order methods seen in course. However, the increased computational cost per iteration and the sensitivity to problem structure highlight the importance of algorithm selection based on specific application requirements and network constraints optimization problems.