CS 224 Fall 2023 Scribes: Itai Shapira
09/13/2023

Lecture 3: Iterative methods for tensor decomposition

1 The Limitations of the Jennrich Algorithm

In the previous class, we discussed a simple algorithm called Jennrich's algorithm for tensor decomposition. To briefly recall, let T be a rank-k tensor in $\mathbb{R}^{d\times d\times d}$, expressible as

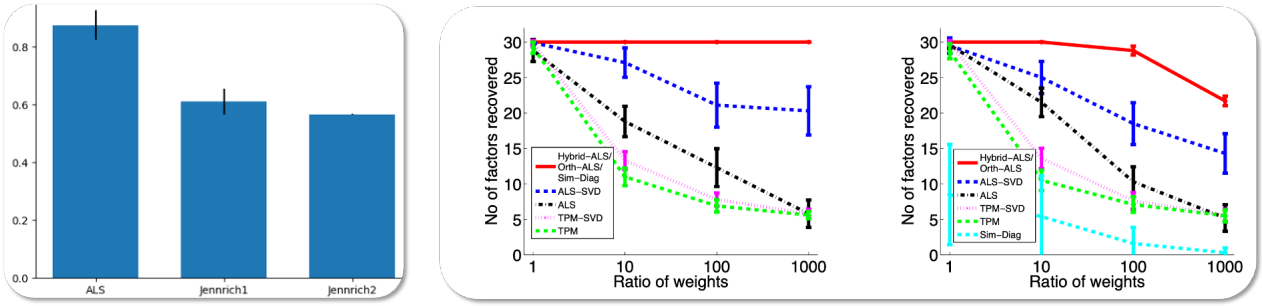
$$T = \sum_{i=1}^{k} \lambda_i \mathbf{u}_i^{\otimes 3}$$

with $\{\mathbf{u}_i\}_{i=1}^k$ being linearly independent unit vectors and $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_k \geq 0$. Jennrich's algorithm decomposes the tensor and precisely recovers $\{\mathbf{u}_i\}_{i=1}^k$.

However, Jennrich's algorithm is not commonly used in practice for two primary reasons:

1. **Not Noise Robust:** Despite its theoretical guarantees in a noiseless setting, Jennrich's algorithm is numerically finicky when it comes to handling noise. When the tensor deviates even slightly from its low-rank form, the algorithm's performance declines sharply in terms of accuracy (see Figure 1)

2. **Computational Overhead:** The algorithm is dominated by dense matrix operations such as matrix multiplications and pseudoinversion. These operations become the bottleneck, particularly when only partial contractions of the tensor are needed in applications and we only need to know how the tensor "acts" on individual vectors.



(a) (b)

Figure 1: (a) A comparison between two implementations of Jennrich's algorithm and ALS. The Y-axis represents a measure of accuracy in recovering a $25 \times 25 \times 25$ tensor with added noise ([Unk21]) (b) The second plot presents a comprehensive comparison of various algorithms, where we pay special attention to a curve labeled "SimDiag," which represents Jennrich's algorithm. In a perfect noiseless setting, Jennrich's algorithm accurately recovers all tensor components. However, upon the introduction of noise, its performance is the worst among the algorithms considered ([SV17])

.

2 Iterative Algorithms

In practice, practitioners commonly resort to three types of formally-interrelated iterative algorithms: Gradient Ascent, Power Iteration and Alternating Least Squares.

For the remainder of this discussion, let us assume that tensor T can be represented as: $T = \sum_{i=1}^{k} \lambda_i \mathbf{u}_i^{\otimes 3}$ where $\{\mathbf{u}_i\}_{i=1}^{k}$ are orthonormal vectors. The rationale for this assumption will be addressed later (see Section 2.4).

Every tensor T can be associated with a polynomial $p(\mathbf{x})$ of degree three with coefficients that correspond to the entries of T:

$$p(\mathbf{x}) = \sum_{a,b,c} T_{abc} x_a x_b x_c = T(\mathbf{x}, \mathbf{x}, \mathbf{x}) = \sum_{i=1}^{k} \lambda_i \langle \mathbf{u}_i, \mathbf{x} \rangle^3$$

For low-rank tensors with orthogonal components, it can be shown that the local maximizers of $p(\mathbf{x})$ over the unit sphere are precisely the set $\{\mathbf{u}_i\}_{i=1}^{k}$. Consequently, our optimization problem simplifies to:

$$\max_{\|x\|=1} p(\mathbf{x}) = \max_{\|\mathbf{x}\|=1} \sum_{i=1}^{k} \lambda_i \langle \mathbf{u}_i, \mathbf{x} \rangle^3 \tag{1}$$

To solve the optimization problem 1, we will discuss several approaches. Both Gradient Ascent and Tensor Power Method aim to identify a single local optimum, while our ultimate goal is to find all the local optima. One naive approach to overcome this limitation, sometimes referred to as *deflation*, is that once we have found a vector $\hat{\mathbf{u}} \approx \mathbf{u}_i$, we can evaluate the polynomial at $\mathbf{u}_i$ to get $\lambda_i = p(\mathbf{u}_i)$. Subsequently, we subtract this component from the original tensor T, effectively reducing the tensor to one with $k-1$ components. This allows us to focus on the remaining components in subsequent iterations:

$$T - p(\hat{\mathbf{u}})\hat{\mathbf{u}}^{\otimes 3} \approx \sum_{j \neq i} \lambda_j \mathbf{u}_j^{\otimes 3}$$

However, this method is not widely adopted due to the compounding errors that can occur in each step. Even if the original tensor had no noise, the modified tensor could introduce an error ϵ, which can exponentially accumulate in successive iterations.

An alternative and more practical approach is to run those methods multiple times with different, randomly chosen initializations. By doing so, we increase the likelihood of converging to different local optima. Once we have obtained a set of vectors $\hat{\mathbf{u}}$, we can cluster them to estimate the actual components of the tensor.

2.1 Gradient Ascent

To solve 1, we use a variant of gradient ascent tailored for constrained optimization. The standard gradient ascent update rule is

$$\mathbf{x}^{(t)} = \mathbf{x}^{(t-1)} + \eta \cdot \nabla p(\mathbf{x}^{(t-1)}) = \mathbf{x}^{(t-1)} + 3\eta \cdot T(:, \mathbf{x}^{(t-1)}, \mathbf{x}^{(t-1)})$$

where η is the step size. However, we need to ensure that our iterations lie on the unit sphere. To enforce this constraint, the update rule is modified to:

$$\mathbf{x}^{(t+1)} = \text{proj}\left(\mathbf{x}^{(t)} + 3\eta \cdot \left(\text{Id} - \mathbf{x}^{(t)}(\mathbf{x}^{(t)})^T\right) \cdot T(:, \mathbf{x}^{(t)}, \mathbf{x}^{(t)})\right) \tag{2}$$

where $(\text{Id} - xx^T)$ is the projection to the tangent space at x and proj is the projection to the sphere $y \mapsto y/||y||$. This update rule involves a projection onto the tangent space as movement orthogonal to the tangent space at a given point is wasted during projection back to the sphere.

Expanding 2, we get:

$$\begin{aligned}\mathbf{x}^{(t+1)} &= \text{proj}\left(\mathbf{x}^{(t)} + 3\eta \cdot \left(T(:, \mathbf{x}^{(t)}, \mathbf{x}^{(t)}) - \mathbf{x}^{(t)}(\mathbf{x}^{(t)})^T T(:, \mathbf{x}^{(t)}, \mathbf{x}^{(t)})\right)\right) \\ &= \text{proj}\left(\mathbf{x}^{(t)} + 3\eta \cdot \left(T(:, \mathbf{x}^{(t)}, \mathbf{x}^{(t)}) - \mathbf{x}^{(t)} T(\mathbf{x}^{(t)}, \mathbf{x}^{(t)}, \mathbf{x}^{(t)})\right)\right) \\ &= \text{proj}\left(\mathbf{x}^{(t)} + 3\eta \cdot \left(T(:, \mathbf{x}^{(t)}, \mathbf{x}^{(t)}) - \mathbf{x}^{(t)} p(\mathbf{x}^{(t)})\right)\right)\end{aligned} \tag{3}$$

2.2 Tensor Power Method

An appealing choice of step size in 3 is

$$\eta = \frac{1}{3p(\mathbf{x}^{(t)})}$$

This adaptive step size, inversely proportional to the value of the polynomial $p(x)$, takes larger steps if the objective function is small and vice versa. More importantly, it allows us to cancel out terms and simplify the update rule in 3 to:

$$\mathbf{x}^{(t+1)} = \text{proj}\left(\frac{T(:, \mathbf{x}^{(t)}, \mathbf{x}^{(t)})}{p(\mathbf{x}^{(t)})}\right) = \text{proj}\left(T(:, \mathbf{x}^{(t)}, \mathbf{x}^{(t)})\right) \tag{4}$$

Equation 4 is a tensor generalization of the *matrix power method*. The matrix power method is an efficient algorithm for finding the top eigenvector of a matrix, and is notably faster than methods based on solving the roots of the characteristic polynomial. If T were a matrix $T = \sum_{i=1}^{k} \lambda_i \mathbf{u}_i \mathbf{u}_i^T$, the algorithm iteratively applies T to an initial vector $\mathbf{x} = \sum_i c_i \mathbf{u}_i$, and then normalizes $\mathbf{x}$. The matrix T acts on $\mathbf{x}$ as: $T\mathbf{x} = \sum_i \lambda_i c_i \mathbf{u}_i$, and this vector is then normalized by its norm:

$$\text{proj}\left(T(:, \mathbf{x})\right) = \sum_i \frac{\lambda_i c_i}{\left(\sum_j \lambda_j^2 c_j^2\right)^{1/2}} \mathbf{u}_i$$

.

Assume without loss of generality that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_k$. Then intuitively, the algorithm inherently focuses more on the first coordinate. To see this, consider the ratio $r_i = \frac{c_i}{c_1}$. After a single iteration, r_i transforms to $\frac{\lambda_i}{\lambda_1} r_i$; and after one more iteration: $(\frac{\lambda_i}{\lambda_1})^2 r_i$. If λ_1 is distinctively larger than $\lambda_2, \ldots, \lambda_k$, the ratios $r_2, \ldots, r_k$ will asymptotically approach zero. This ensures that the vector converges to the top eigenvector over successive iterations.

For tensors, the analysis is analogous: the tensor T acts on $\mathbf{x}$ to produce $T(:,\mathbf{x},\mathbf{x}) = \sum_i \lambda_i c_i^2 \mathbf{u}_i$. This is then projected back to the unit sphere, resulting in the new coefficients as

$$\text{proj}\left(\sum_i \lambda_i c_i^2 \mathbf{u}_i\right) = \sum_i \frac{\lambda_i c_i^2}{\left(\sum_j \lambda_j^2 c_j^4\right)^{1/2}} \mathbf{u}_i.$$

Consider again how the ratio between coordinates $r_i = \frac{c_i}{c_1}$ evolve during this procedure:

$$r_i = \frac{c_i}{c_1} \mapsto \frac{\lambda_i c_i}{\lambda_1 c_1} r_i$$

If we assume without loss of generality that the first coordinate in our initial $(c_1, .., c_k)$ is such that

$$\rho := \max_{i \neq 1} \frac{\lambda_i c_i}{\lambda_1 c_1} < 1$$

then r_i decays and in the next step ρ transforms as:

$$\max_{i \neq 1} \frac{\lambda_i (\lambda_i c_i^2)}{\lambda_1 (\lambda_1 c_1^2)} = \max_{i \neq 1} \left(\frac{\lambda_i c_i}{\lambda_1 c_i}\right)^2 = \rho^2 < 1$$

and

$$r_i \mapsto \rho r_i \mapsto \rho^2 r_i \mapsto \rho^4 r_i \mapsto \rho^8 r_i$$

This implies that the tensor power method offer faster convergence than its matrix counterpart and can achieve double exponential convergence rate to the eigenvector with the largest $\lambda_i c_i$. The particular eigenvector to which the algorithm converges to depends on the initialization vector $\mathbf{x}$.

2.3 Alternating Least Squares (ALS)

The key advantage of ALS over other approaches is that it doesn't suffer from issues of deflation or require restarting the algorithm multiple times. Instead, it is capable of learning all components of the tensor simultaneously.

Given a tensor T, we aim to find optimal component vectors by solving a specific optimization problem. At each iteration t, we maintain current estimates $\{\mathbf{u}_i^t\}_{i=1}^k$. The optimization problem for the next iterate $\mathbf{u}_i^{t+1}$ is defined as:

$$\mathbf{u}_i^{t+1} = \arg\min_{\hat{\mathbf{u}}_i} ||T - \sum_{i=1}^k \hat{\mathbf{u}}_i \otimes \mathbf{u}_i^t \otimes \mathbf{u}_i^t||_F^2$$

Here, the objective function is linear in terms of the optimization variable. This linearity essentially reduces the problem to one of least-squares regression, making it computationally efficient.

It is worth noting that the Tensor Power Method can essentially be considered as a "rank-1" ALS, while ALS serves as a parallelized rank-k extension of the Tensor Power Method.

Despite its complexity from a theoretical standpoint, ALS has garnered significant attention for its robustness and computational efficiency in various real-world applications.

2.4 On The Orthogonality Assumption

We now discuss theoretical justifications for the orthogonality assumption.

Let us assume we have a rank-k tensor T in $\mathbb{R}^{d\times d\times d}$ expressible as $T = \sum_{i=1}^{k} \lambda_i \mathbf{u}_i^{\otimes 3}$, where $\mathbf{u}_i$ are linearly independent but not necessarily orthogonal unit vectors. In many practical scenarios, such as Gaussian mixture models, we can also obtain the associated matrix $M = \sum_{i=1}^{k} \lambda_i \mathbf{u}_i \mathbf{u}_i^T$.

We propose a 'whitening' transformation, using M, that will help us retrieve a tensor consisting solely of orthogonal components. First, we perform eigen-decomposition of M as:

$$M = VDV^T$$

where V is an orthogonal $d\times k$ matrix and D is a $k\times k$ diagonal matrix containing the non-zero eigenvalues. We then define a new matrix $W = VD^{-1/2}$, which normalizes our data in the following sense:

$$W^T MW = (D^{-1/2}V^T)(VDV^T)(VD^{-1/2}) = Id \tag{5}$$

The transformation induced by W helps us define new vectors

$$\tilde{\mathbf{u}}_i := \lambda_i^{1/2} W^T \mathbf{u}_i$$

$\{\tilde{\mathbf{u}}_i\}$ forms an orthogonal basis as confirmed by Equation 5 and:

$$Id = W^T MW = \sum_{i=1}^{k} \lambda_i (W^T\mathbf{u}_i)(W^T\mathbf{u}_i)^T = \sum_{i=1}^{k} (\lambda_i^{1/2} W^T\mathbf{u}_i)(\lambda_i^{1/2} W^T\mathbf{u}_i)^T = \sum_{i=1}^{k} \tilde{\mathbf{u}}_i \tilde{\mathbf{u}}_i{}^T$$

Let's now construct a new tensor, denoted by T', and defined by its contractions:

$$T'(\mathbf{x},\mathbf{y},\mathbf{z}) = T(W\mathbf{x}, W\mathbf{y}, W\mathbf{z})$$

By the definition of tensor contraction, this is equivalent to:

$$
\begin{aligned}
T'(W\mathbf{x}, W\mathbf{y}, W\mathbf{z}) &= \sum_i \lambda_i \langle W\mathbf{x}, \mathbf{u}_i\rangle \langle W\mathbf{y}, \mathbf{u}_i\rangle \langle W\mathbf{z}, \mathbf{u}_i\rangle \\
&= \sum_i \lambda_i \mathbf{u}_i^T W\mathbf{x}\mathbf{u}_i^T W\mathbf{y}\mathbf{u}_i^T W\mathbf{z} \\
&= \sum_i \lambda_i^{-1/2} \langle \mathbf{x}, \tilde{\mathbf{u}}_i\rangle \langle \mathbf{y}, \tilde{\mathbf{u}}_i\rangle \langle \mathbf{z}, \tilde{\mathbf{u}}_i\rangle
\end{aligned}
$$

Therefore, we can express the new tensor T' in terms of orthogonal components:

$$
T' = \sum_i \lambda_i^{-1/2} \tilde{\mathbf{u}}_i^{\otimes 3}
$$

The main takeaway is that by using this matrix W, we are able to extract a transformation that standardizes the data. Once the data is standardized, the effect of applying this transformation is to convert $\mathbf{u}_i$ into $\tilde{\mathbf{u}}_i$. By taking W and reshaping the original tensor T, we obtain a new tensor T' whose components are exactly the orthogonal $\tilde{\mathbf{u}}_i$. This reduces the problem back to the simpler case of orthogonal tensors, which is often why practitioners prefer to work with the orthogonal case.

2.5 What if we don't / can't whiten?

When $u_1, \ldots, u_k$ are not orthogonal but merely linearly independent, analyzing these algorithms becomes much more challenging.

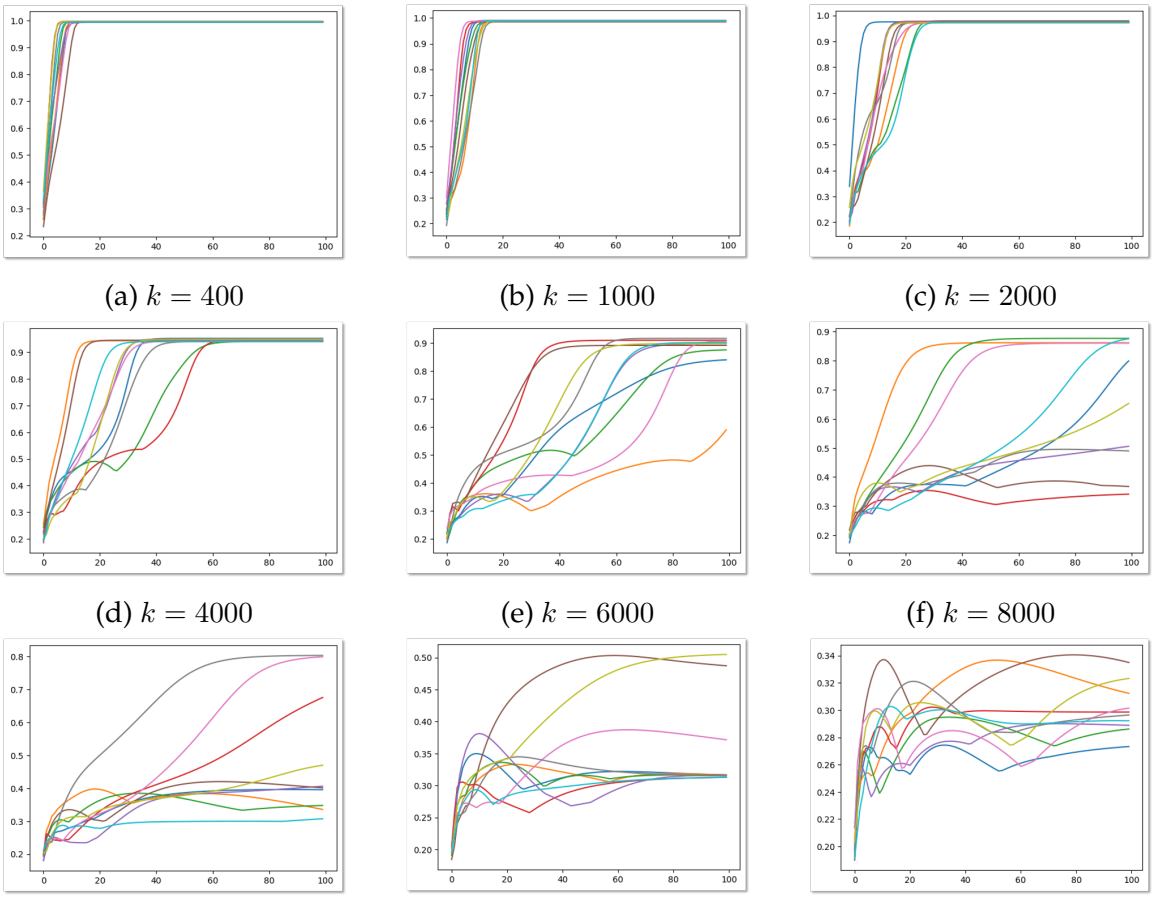
Theorem 1 ([SV17]). *Given $T = \sum_{i=1}^k u_i^{\otimes 3}$ for "incoherent" unit vectors $u_1, \ldots, u_k$, i.e., satisfying*

$$
\left|\langle u_i, u_j\rangle\right| \le c_{max} \le \frac{1}{k^{1+\epsilon}},
$$

$O(\log k + \log\log d)$ iterations of tensor power method starting from random initialization yields a vector $\hat{u}$ that is $O\left(k^{1/2}\max(c_{max}, 1/d)\right)$-close to some u_i, with high probability.

3 Empirical Mysteries

In this section, we explore the empirical behavior of tensor decomposition algorithms, emphasizing the limitations of our current theoretical understanding. This underscores the inherent difficulties in analyzing non-convex optimization problems, such as those encountered in neural networks.



(a) $k = 400$ (b) $k = 1000$ (c) $k = 2000$

(d) $k = 4000$ (e) $k = 6000$ (f) $k = 8000$

(g) $k = 10000$ (h) $k = 12000$ (i) $k = 15000$

Figure 2: each plot visualizes the behavior of tensor power methods for tensors of varying ranks k and $d = 400$. The x-axis represents the number of iterations of the tensor power method, while the y-axis quantifies the correlation between the output of the algorithm and the closest among the original tensor components. Different colors in each plot signify different training runs, initialized randomly. Across these plots, peculiar non-monotonic behavior can be observed, particularly when k is high, revealing occasional dips in the correlation value. This suggests that the algorithm might be switching its convergence target between different tensor components. As k increases beyond the dimensionality of the tensor (400) the performance deteriorates significantly, reflecting the limitations of the tensor power methods and hinting at an empirically observed *critical threshold* at which the algorithm fails.

Our focus is on understanding the behavior of the tensor power method across different training runs and dimensions. In Figure 2, we plot the behavior of the

tensor power method across different ranks k. Each graph illustrates the algorithm's output correlation with the closest original tensor component against the number of iterations. Different colors within each plot represent separate training runs with random initializations.

We find that the algorithm converges within roughly 10 iterations for low levels of k. However, peculiar behaviors arise when we exceed the boundary $k < d$ and attempt to decompose tensors with more than 400 components. Even at 1000 or 2000 components, the tensor power method generally works, albeit requiring more iterations for convergence.

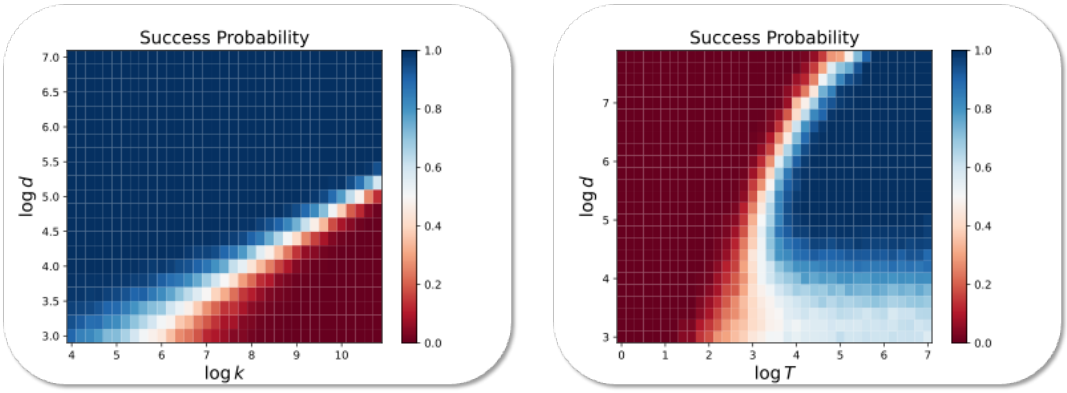


Figure 3: Success probability of tensor power iteration for varying k and d. The white line marks the threshold $k = d^{l/2}$, beyond which the success probability sharply declines. Blue regions indicate high success rates, while red regions point to low success rates. The phase transition around $\log k = 2 \log d$ suggests that tensor power iteration is likely to succeed when $k \ll d^2$ and fail when $k \gg d^2$. We also observe that polynomially many steps are necessary for tensor power iteration to converge, since $\log T$ scales linearly with $\log d$ around the success/failure boundary. The experiment involves running the tensor power method for 1000 iterations, repeated 1000 times independently for each (k, d) pair.

Interestingly, as we increase the number of components, the correlation exhibits non-monotonic behavior, indicating a complex landscape of vectors that the algorithm is navigating. It suggests that during training, the algorithm may switch its focus among different vectors, leading to drops in the correlation. This is especially challenging to analyze due to this non-monotonic behavior. At around 8000 components, we hit a computational limit, which correlates with $k = d^{3/2}$. This threshold is thought to be a boundary not only for tensor power methods but also potentially for any polynomial-time algorithm. It is noteworthy that this computational threshold is far below the information-theoretical limit of $k = d^2$, indicating gaps in our

current understanding of tensor decomposition algorithms. More generally, there exists a conjecture suggesting that the critical number of components k at which tensor decomposition algorithms will begin to fail is captured by $k = d^{\ell/2}$ for ℓ the dimension of the tensor. This is illustrated in Figure 3 by recent work from [WZ23] for $\ell = 4$. Specifically, the tensor power iteration is likely to succeed when $k \ll d^2$ and likely to fail when $k \gg d^2$.

4 Summary

While Jennrich's algorithm is theoretically viable, it is generally ill-suited for practical use due to its numerical instability and the computational burden of executing dense matrix multiplications.

Instead, iterative methods like gradient ascent and tensor power method stand out as more robust alternatives. These approaches are closely related, and are difficult to analyze formally.

The existing theory falls short of fully explaining the algorithms' empirical behavior. This gap in our understanding sets the stage for our next lecture, where we will explore provable overcomplete tensor decomposition by going beyond worst-case analysis.

References

[SV17] Vatsal Sharan and Gregory Valiant. Orthogonalized als: A theoretically principled tensor decomposition algorithm for practical use. In *International Conference on Machine Learning*, pages 3095–3104. PMLR, 2017.

[Unk21] Unknown. Will the real jennrich's algorithm please stand up?, 2021. https://www.mathsci.ai/post/jennrich/.

[WZ23] Yuchen Wu and Kangjie Zhou. Lower bounds for the convergence of tensor power iteration on random overcomplete models. In *The Thirty Sixth Annual Conference on Learning Theory*, pages 3783–3820. PMLR, 2023.