Near-Universal Multiplicative Updates for Nonnegative Einsum Factorization

John Hood
University of Chicago

Aaron Schein
University of Chicago

ICML
International Conference
On Machine Learning

Motivation: Large Family of Tensor Methods

Many powerful algorithms/models/tensor methods:

but extending/modifying/changing is hard/model-specific.

Tensor Decomposition as Einsum Factorization

Simple matrix factorization assumes:

$$y_{i,j} \approx \sum_{q=1}^{Q} \psi_{iq} \beta_{jq}$$

Indices appearing on the RHS but not LHS are **contracted**

observed indices vs latent indices

Written more compactly in terms of only the indices:

$$ij \leftarrow iq,\ jq$$

Tensor Contractions are Einsum Factorizations

$$y^{(t)}_{i \xrightarrow{a} j} \approx \sum_{c=1}^{C} \sum_{d=1}^{C} \sum_{k=1}^{K} \sum_{r=1}^{R} \psi^{(\rightarrow)}_{ic} \, \psi^{(\leftarrow)}_{jd} \, \phi_{ak} \, \theta_{t,r} \, \lambda^{(r)}_{c \xrightarrow{k} d}$$

$$ijat \leftarrow ic,\ jd,\ ak,\ tr,\ cdkr$$

4-mode Tucker: Contraction of 4 matrices, 1 tensor

$$ijat \leftarrow ic,\ jd,\ ak,\ tr,\ cp, dp, kp, rp, p$$

4-mode Tucker with low-rank core tensor

$$ijat \leftarrow itc,\ jtd,\ ak,\ tr,\ cdkr$$

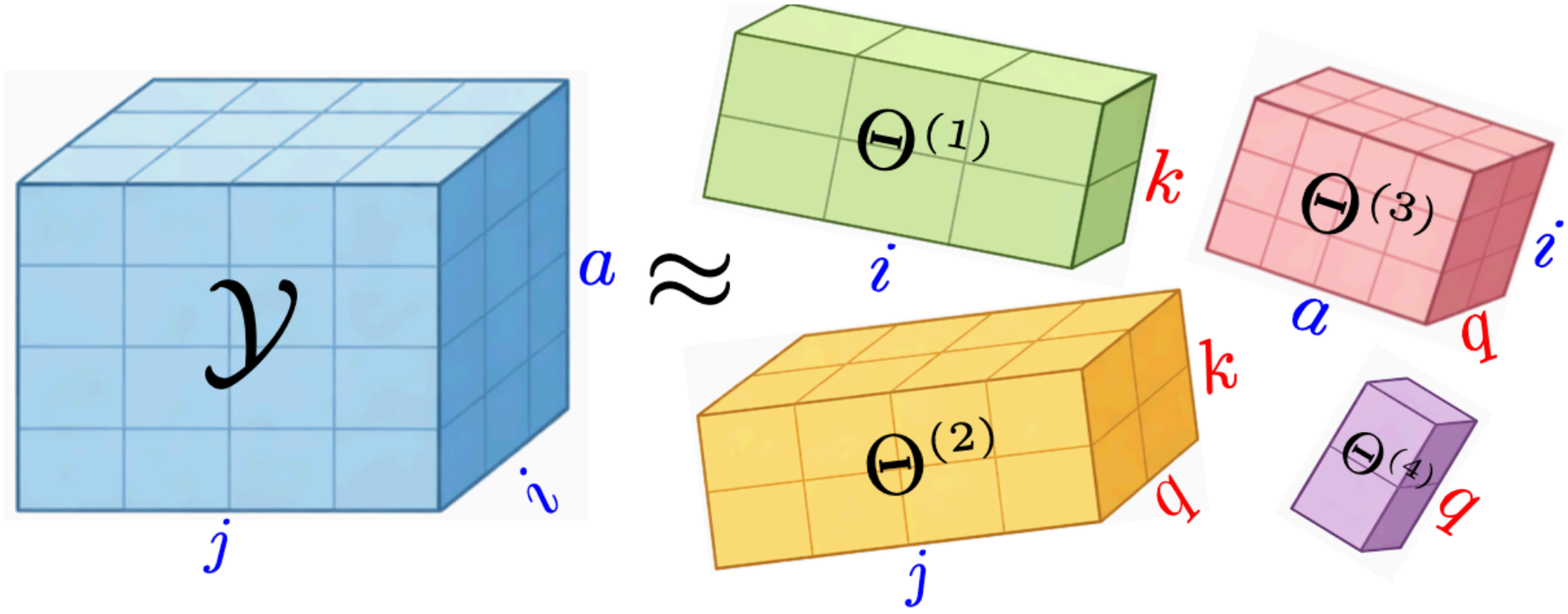
4-mode Tucker with time-varying factor matrices

…

This Work: Nonnegative Einsum Factorization (NNEinFact)

```
from einfact import NNEinFact

model = NNEinFact(
    model_str='ik,jqk,aiq,q->ija',
    loss='alpha-beta', α=0.1, β=1.1)

model.fit(data=Y_ija)

θ_ik, θ_jqk, θ_aiq, θ_q = model.get_params()
```

NMF	`ik, jk → ij`
CP	`ik, jk, ak → ija`
Tucker	`ic, jd, ak, cdk → ija`
Tensor train	`ic, jcd, ad → ija`
CANDELINC	`ic, jd, ak, cq, dq, kq → ija`
Custom	`ic, jd, ack, kd, k → ija`



Loss examples

Euclidean distance
Forward KL divergence
Reverse KL divergence
Itakura-Saito divergence
Hellinger distance
Neyman χ^2 divergence
Pearson χ^2 divergence
α-divergence
β-divergence
(α, β)-divergence

One Multiplicative Update Algorithm for All

model

$$\widehat{Y} \leftarrow \text{einsum}(\text{model\_str}, \{\Theta^{(\ell)}\}_{\ell=1}^{L})$$

loss

$$\min_{\Theta^{(1)},\ldots,\Theta^{(L)}} \mathscr{L}(Y, \widehat{Y}) = \sum_{i} \mathscr{L}(y_i, \hat{y}_i), \quad \Theta^{(\ell)} \geq \varepsilon$$

"near-universal" multiplicative update rule

$$\Theta^{(\ell)} \leftarrow \max\left(\varepsilon, \Theta^{(\ell)} \odot g^{-1}\left(\frac{\sum_i [\nabla_{\Theta^{(\ell)}} \hat{y}_i] a(y_i, \hat{y}_i)}{\sum_i [\nabla_{\Theta^{(\ell)}} \hat{y}_i] b(y_i, \hat{y}_i)}\right)\right)$$

implementation just involves calls to einsum

while not converged **do**
  for $\ell = 1, \ldots, L$ **do**
    $\widehat{Y} \leftarrow \text{einsum}(\text{model\_str}, \{\Theta^{(\ell)}\}_{\ell=1}^{L})$
    $\text{A} \leftarrow \text{einsum}(\text{einstr}_\ell, \Theta^{(1)}, \ldots, \Theta^{(\ell-1)}, a(Y, \widehat{Y}), \Theta^{(\ell+1)}, \ldots, \Theta^{(L)})$
    $\text{B} \leftarrow \text{einsum}(\text{einstr}_\ell, \Theta^{(1)}, \ldots, \Theta^{(\ell-1)}, b(Y, \widehat{Y}), \Theta^{(\ell+1)}, \ldots, \Theta^{(L)})$
    $\Theta^{(\ell)} \leftarrow \max(\varepsilon, \Theta^{(\ell)} \odot g^{-1}\left(\frac{\text{A}}{\text{B}}\right))$

Theory and Computation

Theory

- Convergence of the objective values via MM
- Convergence of the iterates $\Theta^{(1)}, \ldots, \Theta^{(L)}$ to a stationary point via BCD theory

Computation

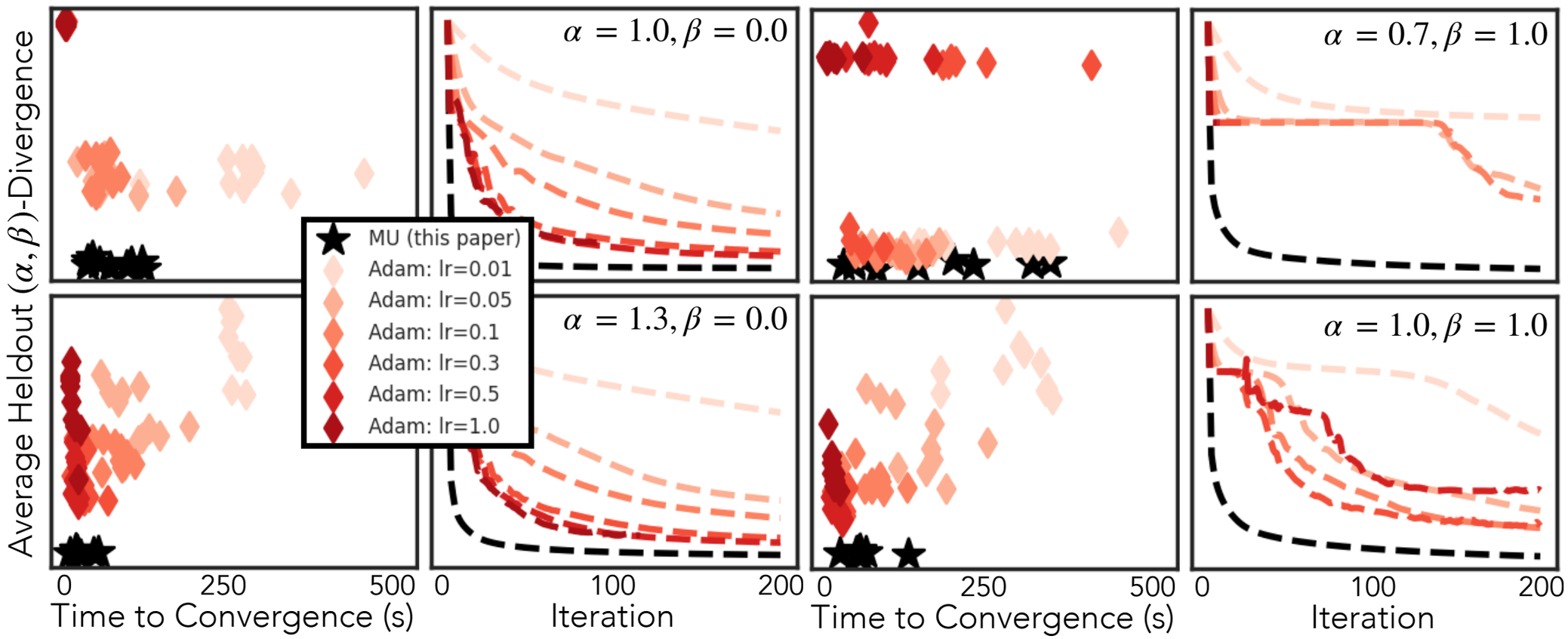
- einfact.py file is less than 200 lines

```
from einfact import NNEinFact

model = NNEinFact(
    model_str='ik,jqk,aiq,q->ija',
    loss='alpha-beta', α=0.1, β=1.1)

model.fit(data=Y_ija)

θ_ik, θ_jqk, θ_aiq, θ_q = model.get_params()
```

- Factorization structure
  - To change the model, change the string
  - Rapid iteration and refinement
- Loss function
  - α and β: tunable parameters that shape the loss
  - $0 < \alpha < 1$: shrinks large entries y_i — robustness to outliers
  - $\alpha > 1$: magnifies large entries — robustness to missing data input as zeros
  - $\beta = 1$: least-squares, $\beta = 0$: KL divergence

Fast Convergence of Multiplicative Updates

To fit custom models, the only alternative options are automatic-differentiation gradient-based methods, such as Adam.

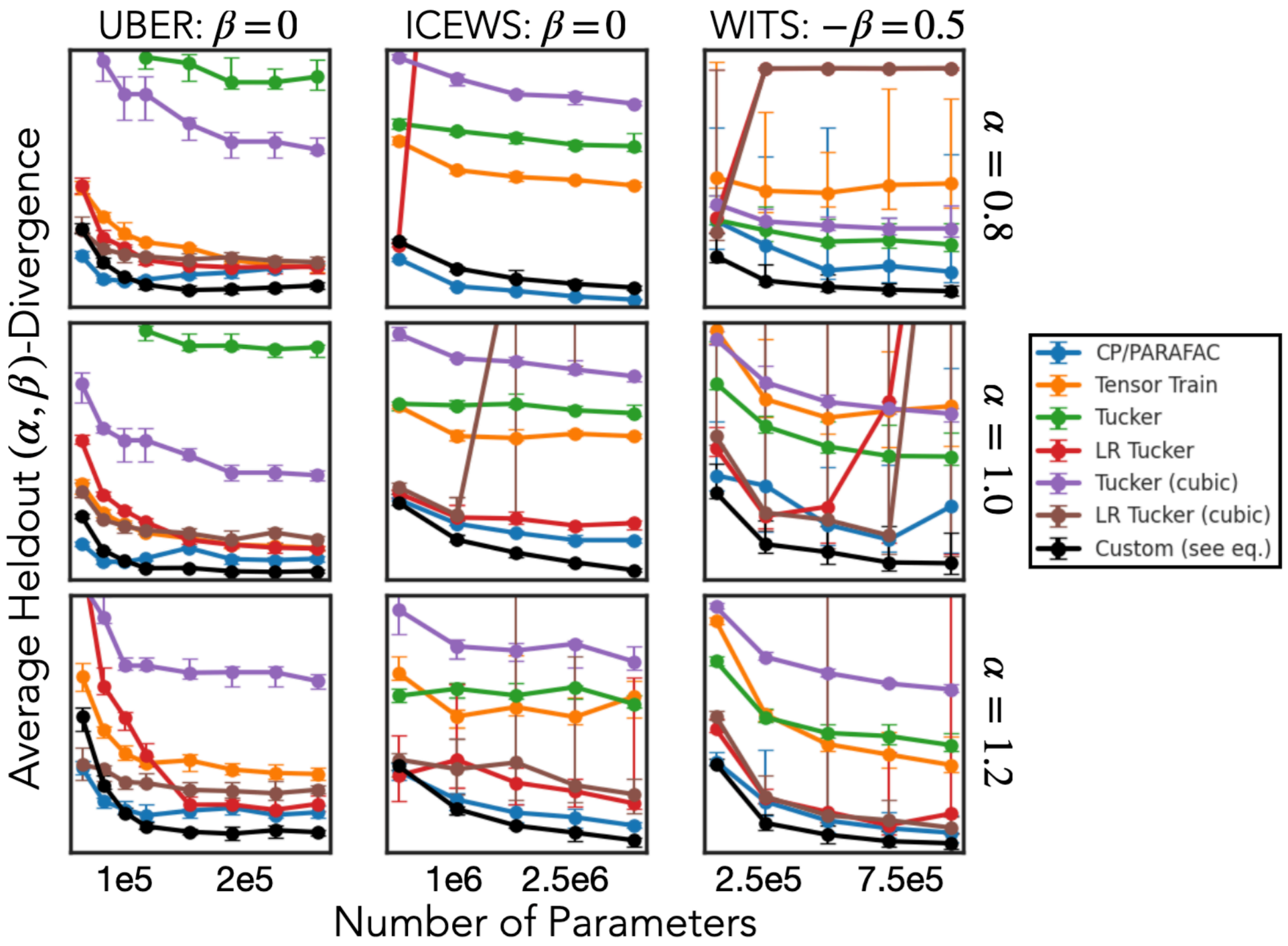


Compared to Adam, NNEinFact achieves:

- Faster convergence
- Lower heldout loss (better fit to the data)

Custom Models Often Fit the Data Better

Custom models $\Longrightarrow$ more parameter-efficient representations of the data



Flexible Parts-Based Scientific Discovery

Case study: New York City Uber pickup data, April to September of 2014

$$\mathcal{Y} \in \mathbb{N}_0^{27\times7\times24\times400\times400}$$

5-mode tensor:

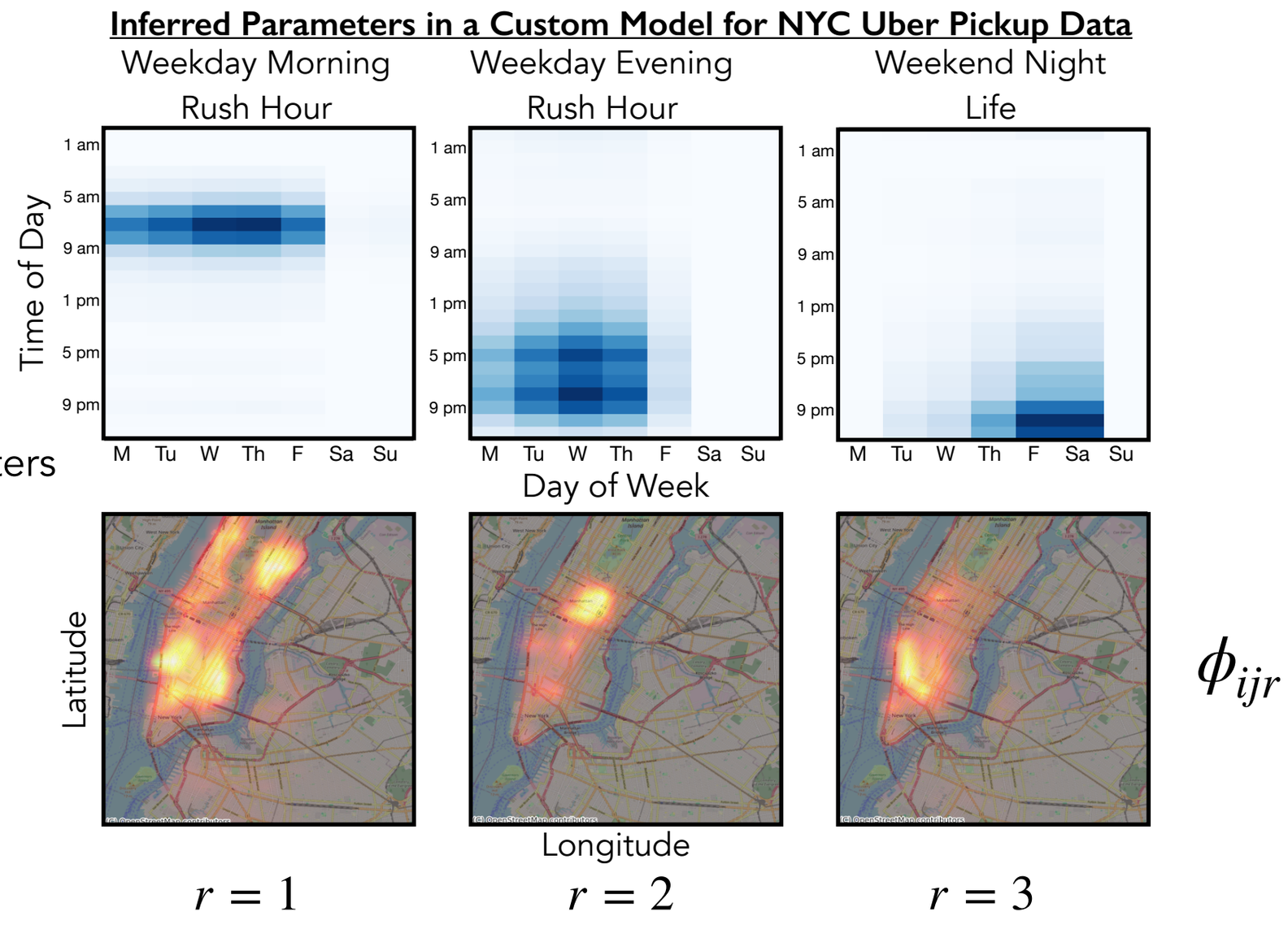
- week
- day of week
- hour of day
- latitude
- longitude

Model: CP-like, except extra parameters for spatial structure

$$\hat{y}_{wdhij} = \sum_{r=1}^{R} \theta^{(1)}_{wr}\theta^{(2)}_{dr}\theta^{(3)}_{hr} \underbrace{\sum_{k=1}^{K} \theta^{(4)}_{irk}\theta^{(5)}_{jrk}}_{\phi_{ijr}}$$

$$\updownarrow$$

wr,dr,hr,ikr,jkr → wdhij



NNEinFact: Summary

Einsum representation: Any tensor network, decomposition, or factorization that can be represented as a tensor contraction can be represented as an einsum factorization.

Multiplicative update algorithm: NNEinFact is a general-purpose multiplicative update algorithm for *any* nonnegative einsum factorization under a wide family of loss functions (e.g., Gaussian likelihood, Poisson likelihood, robust losses, …).

- Guaranteed to converge
- Flexible tool with simple, user-friendly code

Empirical advantages:

- Converges faster than gradient-based approaches to lower loss values
- Allows for expressive, parameter-efficient factorizations tailored to your data