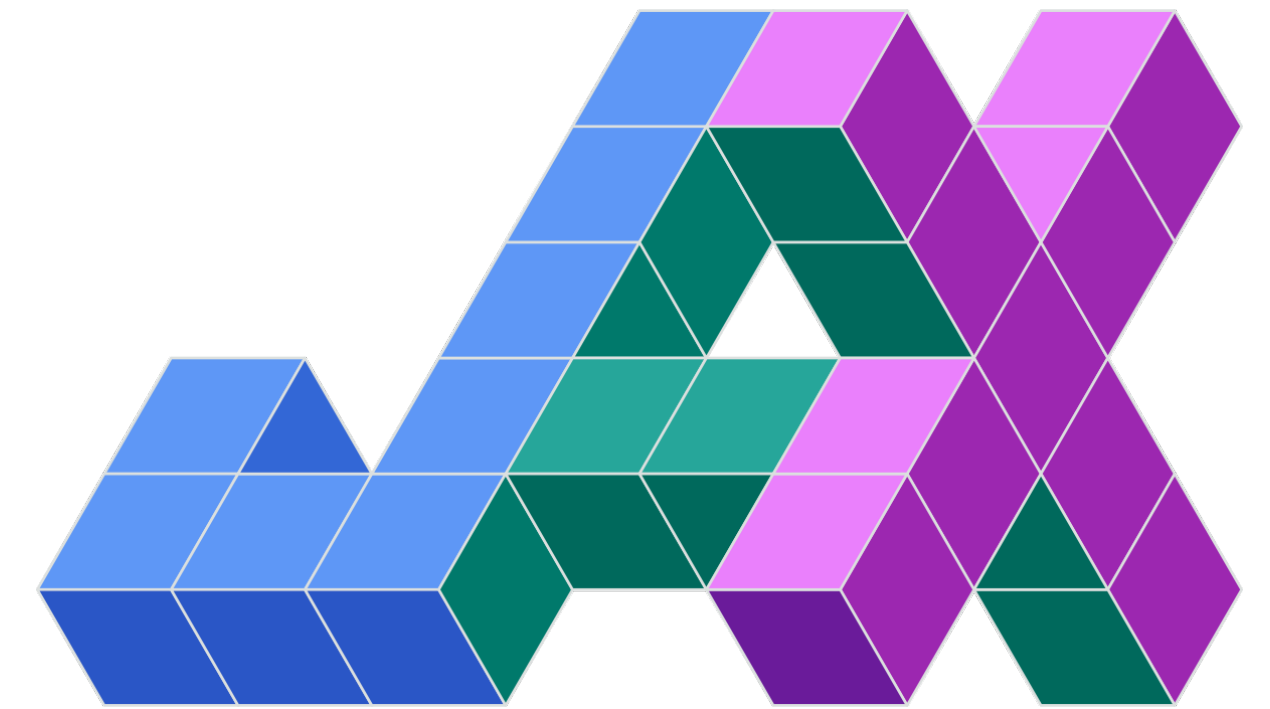


Introduction to Jax from the lens of computational biology

CCB Skills Seminar

Martin Kim (Yosef Lab, UC Berkeley)

11 May 2023



Link for following along

Colab notebook: <https://tinyurl.com/ccb-intro-to-jax>

Plan for today

What is Jax? Overview of its main features

Exploring its main features with an extended example

Why use Jax? Its ecosystem and some projects

What is Jax?

- Open-source machine learning library in Python developed by Google

What is Jax?

- Open-source machine learning library in Python developed by Google
- A scalable and flexible set of numerical **transformations** and **compositions** on arrays

What is Jax?

- Open-source machine learning library in Python developed by Google
- A scalable and flexible set of numerical **transformations** and **compositions** on arrays
 - Composable **gradient** computations

What is Jax?

- Open-source machine learning library in Python developed by Google
- A scalable and flexible set of numerical **transformations** and **compositions** on arrays
 - Composable **gradient** computations
 - Efficient **linear algebra** in multiple backends (CPU, GPU, TPU)

What is Jax?

- Open-source machine learning library in Python developed by Google
- A scalable and flexible set of numerical **transformations** and **compositions** on arrays
 - Composable **gradient** computations
 - Efficient **linear algebra** in multiple backends (CPU, GPU, TPU)
 - Flexible code **compilation**

What is Jax?

- Open-source machine learning library in Python developed by Google
- A scalable and flexible set of numerical **transformations** and **compositions** on arrays
 - Composable **gradient** computations
 - Efficient **linear algebra** in multiple backends (CPU, GPU, TPU)
 - Flexible code **compilation**
- Interface inspired by **NumPy**

What is Jax?

- Open-source machine learning library in Python developed by Google
- A scalable and flexible set of numerical **transformations** and **compositions** on arrays
 - Composable **gradient** computations
 - Efficient **linear algebra** in multiple backends (CPU, GPU, TPU)
 - Flexible code **compilation**
- Interface inspired by **NumPy**
- Emphasis on **functional programming**

Familiar API

```
import numpy as np  
import jax.numpy as jnp
```

Familiar API

```
import numpy as np  
import jax.numpy as jnp
```

```
z = np.dot(X, y)
```

```
z = jnp.dot(X, y)
```

Familiar API

```
import numpy as np
import jax.numpy as jnp
```

```
z = np.dot(X, y)
z = jnp.dot(X, y)
```

```
dist = np.linalg.norm(z - y)
dist = jnp.linalg.norm(z - y)
```

Familiar API

```
import numpy as np
import jax.numpy as jnp
```

```
z = np.dot(X, y)
z = jnp.dot(X, y)
```

```
dist = np.linalg.norm(z - y)
dist = jnp.linalg.norm(z - y)
```

```
a = np.reshape(X, (-1,))
a = jnp.reshape(X, (-1,))
```

Familiar API

```
import numpy as np
import jax.numpy as jnp
```

```
z = np.dot(X, y)
z = jnp.dot(X, y)
```

```
dist = np.linalg.norm(z - y)
dist = jnp.linalg.norm(z - y)
```

```
a = np.reshape(X, (-1,))
a = jnp.reshape(X, (-1,))
```

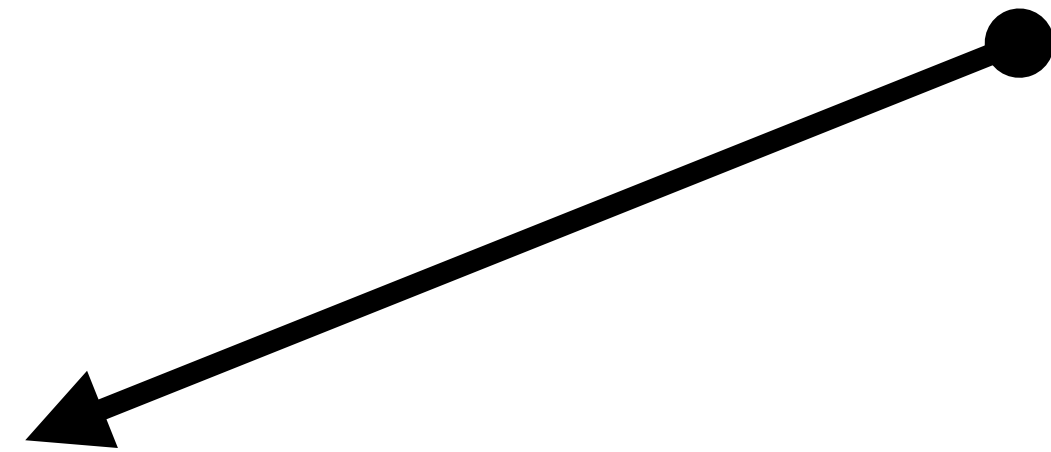
numpy.ndarray



jax.numpy.ndarray

JAX

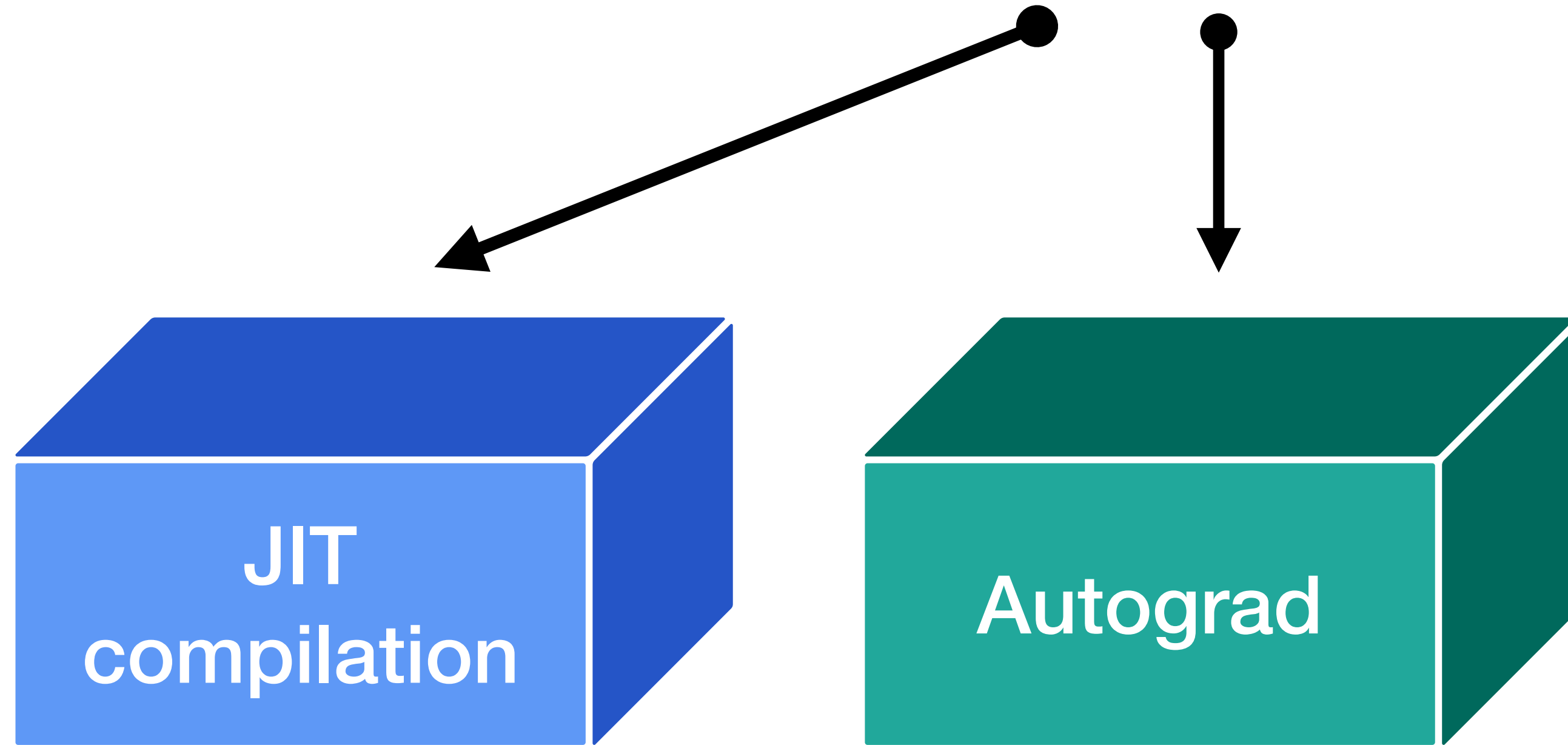
JAX



Compiles user-written code
using various optimizations

```
from jax import jit
```

JAX



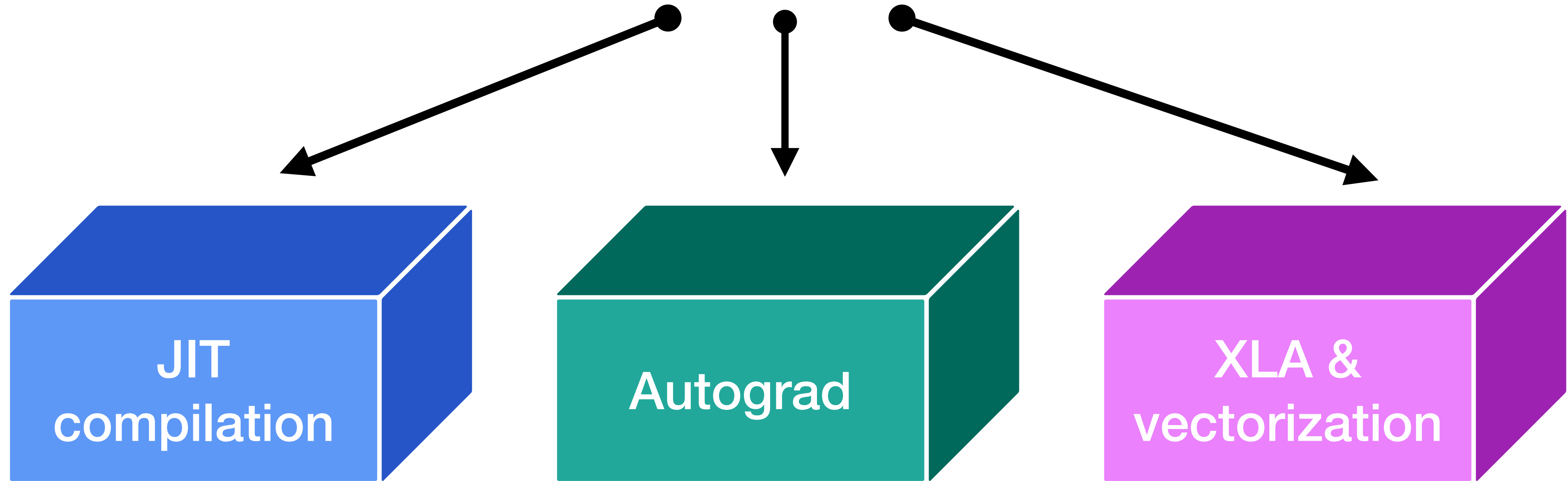
Compiles user-written code
using various optimizations

```
from jax import jit
```

Computes gradients through
Jax and native Python code

```
from jax import grad
```

JAX



Compiles user-written code using various optimizations

```
from jax import jit
```

Computes gradients through Jax and native Python code

```
from jax import grad
```

Provide efficient kernels and automatic vectorization

```
from jax import vmap
```

Extended example for today

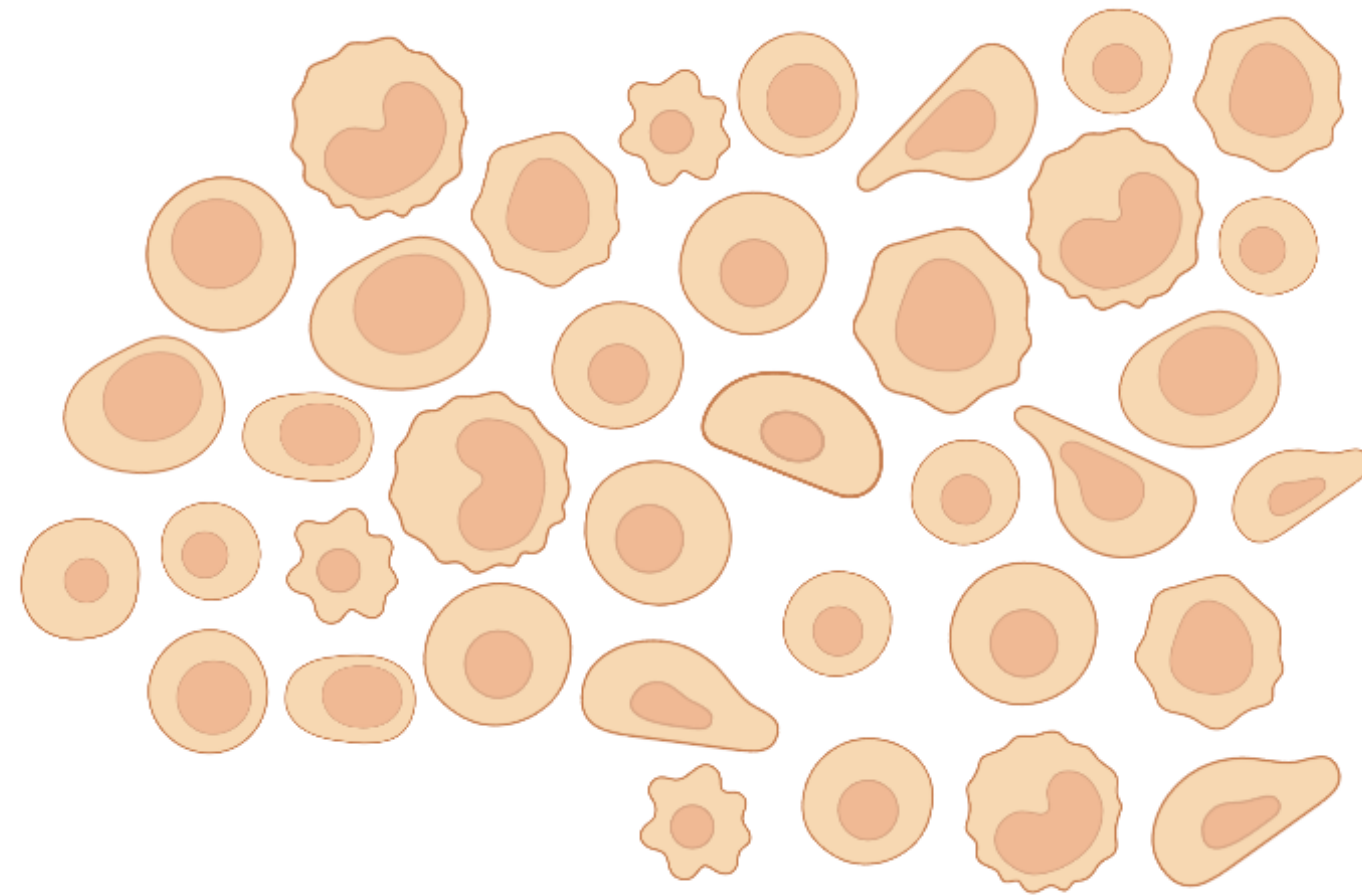
Exploratory analysis of single-cell RNA-seq data

Subset of the human lung cell atlas (3000 cells x 2000 genes)

Extended example for today

Exploratory analysis of single-cell RNA-seq data

Subset of the human lung cell atlas (3000 cells x 2000 genes)



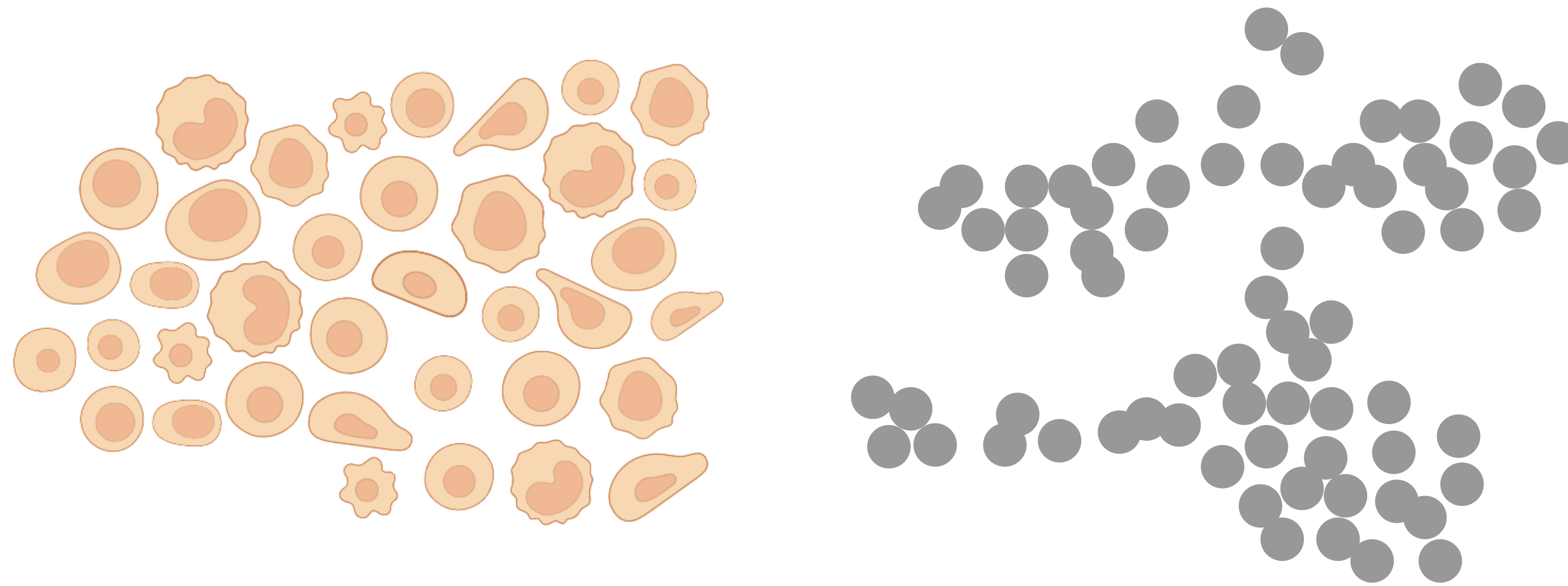
Single-cell RNA-seq data

Count matrix (cells by
genes)

Extended example for today

Exploratory analysis of single-cell RNA-seq data

Subset of the human lung cell atlas (3000 cells x 2000 genes)



Single-cell RNA-seq data

Count matrix (cells by
genes)

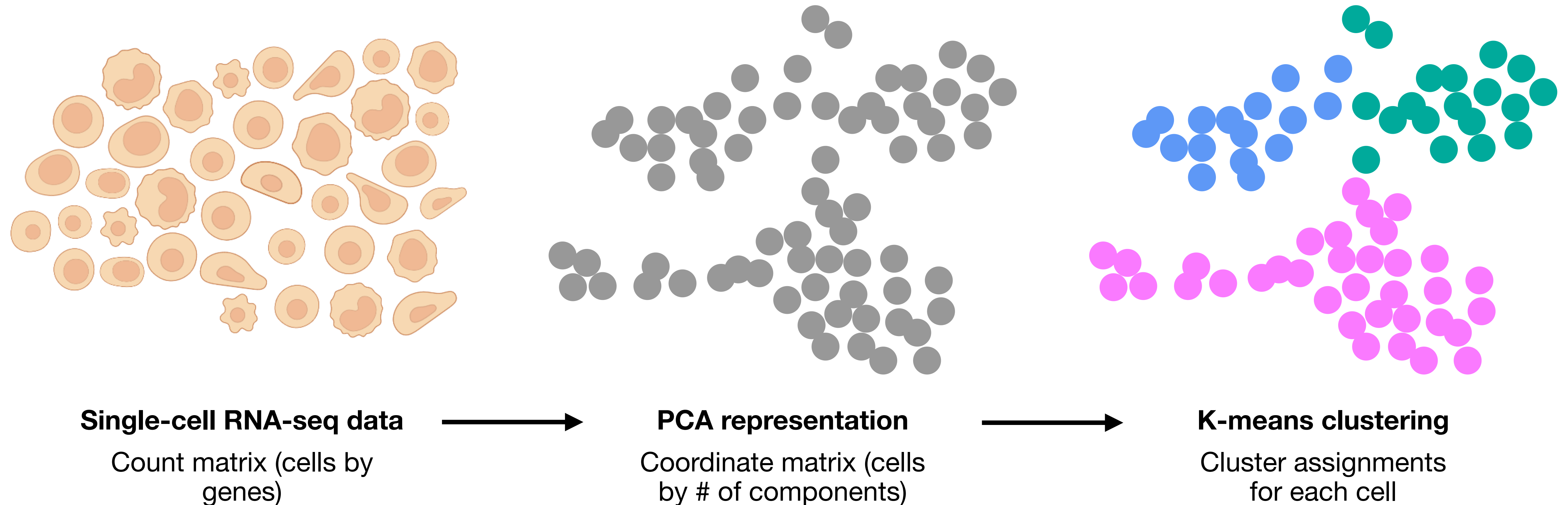
PCA representation

Coordinate matrix (cells
by # of components)

Extended example for today

Exploratory analysis of single-cell RNA-seq data

Subset of the human lung cell atlas (3000 cells x 2000 genes)



JIT compilation

Just-in-time compilation: Jax can compile code *during* the execution of a Python program and cache functions

JIT compilation

Just-in-time compilation: Jax can compile code *during* the execution of a Python program and cache functions

```
jit_my_func = jax.jit(my_func)
```

JIT compilation

Just-in-time compilation: Jax can compile code *during* the execution of a Python program and cache functions

```
jit_my_func = jax.jit(my_func)
```

Applies optimizations such as:

JIT compilation

Just-in-time compilation: Jax can compile code *during* the execution of a Python program and cache functions

```
jit_my_func = jax.jit(my_func)
```

Applies optimizations such as:

- Operation fusions

JIT compilation

Just-in-time compilation: Jax can compile code *during* the execution of a Python program and cache functions

```
jit_my_func = jax.jit(my_func)
```

Applies optimizations such as:

- Operation fusions
- Removing redundant memory allocations

JIT compilation

Just-in-time compilation: Jax can compile code *during* the execution of a Python program and cache functions

```
jit_my_func = jax.jit(my_func)
```

Applies optimizations such as:

- Operation fusions
- Removing redundant memory allocations

How it works: Jax drops an **abstract array** into the function and traces how it is affected in order to optimize the function

JIT compilation: demo

Logistic regression cost function

JIT compilation

Works best when the function is:

JIT compilation

Works best when the function is:

- **Complex with many sequential array operations**
 - More opportunities for optimizations

JIT compilation

Works best when the function is:

- Complex with many sequential array operations
 - More opportunities for optimizations
- **Called multiple times on inputs with the same shape and datatype**
 - Spreads out the initial overhead cost

JIT compilation

Works best when the function is:

- Complex with many sequential array operations
 - More opportunities for optimizations
- Called multiple times on inputs with the same shape and datatype
 - Spreads out the initial overhead cost
- ***e.g.* forward pass or a gradient update to the parameters in an neural net**

Example: PCA

Goal: Compute the PCA representation of our data for visualization and efficient clustering

Example: PCA

Goal: Compute the PCA representation of our data for visualization and efficient clustering

Find the n directions that capture the most variance in our data.

Example: PCA

Goal: Compute the PCA representation of our data for visualization and efficient clustering

Find the n directions that capture the most variance in our data.

1. $U, \Sigma, V^T \leftarrow \text{SVD}(X)$

Example: PCA

Goal: Compute the PCA representation of our data for visualization and efficient clustering

Find the n directions that capture the most variance in our data.

1. $U, \Sigma, V^T \leftarrow \text{SVD}(X)$

2. principal coordinates $\leftarrow \sigma \odot U$, $\text{diag}(\sigma) = \Sigma$

Example: PCA

Goal: Compute the PCA representation of our data for visualization and efficient clustering

Find the n directions that capture the most variance in our data.

1. $U, \Sigma, V^T \leftarrow \text{SVD}(X)$
2. principal coordinates $\leftarrow \sigma \odot U$, $\text{diag}(\sigma) = \Sigma$
3. take first n principal coordinates

JIT compilation: demo

Principal components analysis

XLA & vectorization

Accelerated Linear Algebra (XLA) is the compiler that dispatches Jax Python code to device-specific kernels (CPU, GPU, TPU)

XLA & vectorization

Accelerated Linear Algebra (XLA) is the compiler that dispatches Jax Python code to device-specific kernels (CPU, GPU, TPU)

`jax.vmap` allows us to take advantage of XLA without writing complicated batched code by automatically vectorizing array operations

XLA & vectorization

Accelerated Linear Algebra (XLA) is the compiler that dispatches Jax Python code to device-specific kernels (CPU, GPU, TPU)

`jax.vmap` allows us to take advantage of XLA without writing complicated batched code by automatically vectorizing array operations

`my_func` **# complex function only implemented for vectors**

XLA & vectorization

Accelerated Linear Algebra (XLA) is the compiler that dispatches Jax Python code to device-specific kernels (CPU, GPU, TPU)

`jax.vmap` allows us to take advantage of XLA without writing complicated batched code by automatically vectorizing array operations

```
my_func # complex function only implemented for vectors
```

```
X = np.random.randn(1000, 1000)
```

XLA & vectorization

Accelerated Linear Algebra (XLA) is the compiler that dispatches Jax Python code to device-specific kernels (CPU, GPU, TPU)

jax.vmap allows us to take advantage of XLA without writing complicated batched code by automatically vectorizing array operations

```
my_func # complex function only implemented for vectors
```

```
X = np.random.randn(1000, 1000)
```

```
vmap_my_func = jax.vmap(my_func, in_axes=0)
```

XLA & vectorization

Accelerated Linear Algebra (XLA) is the compiler that dispatches Jax Python code to device-specific kernels (CPU, GPU, TPU)

`jax.vmap` allows us to take advantage of XLA without writing complicated batched code by automatically vectorizing array operations

```
my_func # complex function only implemented for vectors
```

```
X = np.random.randn(1000, 1000)
```

```
vmap_my_func = jax.vmap(my_func, in_axes=0)
```

```
result = vmap_my_func(X)
```

XLA & vectorization: demo

Pairwise Euclidean distances

Example: k-means clustering

Goal: Cluster our data using k-means for visualization and labeling

Example: k-means clustering

Goal: Cluster our data using k-means for visualization and labeling

Given data $X \in \mathbb{R}^{N \times D}$ and fixed number of clusters k :

Find μ_i that minimize
$$\sum_{i=1}^k \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \mu_i\|_2^2$$

Example: k-means clustering

Goal: Cluster our data using k-means for visualization and labeling

Given data $X \in \mathbb{R}^{N \times D}$ and fixed number of clusters k :

Find μ_i that minimize $\sum_{i=1}^k \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \mu_i\|_2^2$

Expectation step: $y_j \leftarrow \arg \min \|\mathbf{x}_j - \mu_i\|_2^2$

Example: k-means clustering

Goal: Cluster our data using k-means for visualization and labeling

Given data $X \in \mathbb{R}^{N \times D}$ and fixed number of clusters k :

Find μ_i that minimize $\sum_{i=1}^k \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \mu_i\|_2^2$

Expectation step: $y_j \leftarrow \arg \min \|\mathbf{x}_j - \mu_i\|_2^2$

Maximization step: $\mu_i \leftarrow \frac{1}{|S_i|} \sum_{\mathbf{x}_j \in S_i} \mathbf{x}_j$

XLA & vectorization: demo

K-means clustering with Lloyd's method

Autograd

Jax provides automatic differentiation as part of its core functionality:

```
grad_my_func = jax.grad(my_func)
```

```
derivative = grad_my_func(1.5)
```

Autograd

Jax provides automatic differentiation as part of its core functionality:

```
grad_my_func = jax.grad(my_func)
```

```
derivative = grad_my_func(1.5)
```

It is composable with all other core functions we've seen:

```
composed = jax.grad(jax.vmap(jax.jit(my_func)))
```

Autograd

Jax provides automatic differentiation as part of its core functionality:

```
grad_my_func = jax.grad(my_func)
```

```
derivative = grad_my_func(1.5)
```

It is composable with all other core functions we've seen:

```
composed = jax.grad(jax.vmap(jax.jit(my_func)))
```

Similar to JIT compilation, Jax **traces** the operations that affect a particular value and then applies derivative rules.

Autograd: demo

Plotting simple functions and their derivatives

Example: Newton's method

Goal: Use Newton's method to optimizer our loss function

Example: Newton's method

Goal: Use Newton's method to optimizer our loss function

Newton's method uses second-order partial derivatives (Hessian matrices) to characterize the curvature of the loss function.

Example: Newton's method

Goal: Use Newton's method to optimizer our loss function

Newton's method uses second-order partial derivatives (Hessian matrices) to characterize the curvature of the loss function.

$$\mathcal{L} = \sum_{i=1}^k \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \mu_i\|_2^2$$

Expectation step: $y_j \leftarrow \arg \min \|\mathbf{x}_j - \mu_i\|_2^2$

Example: Newton's method

Goal: Use Newton's method to optimizer our loss function

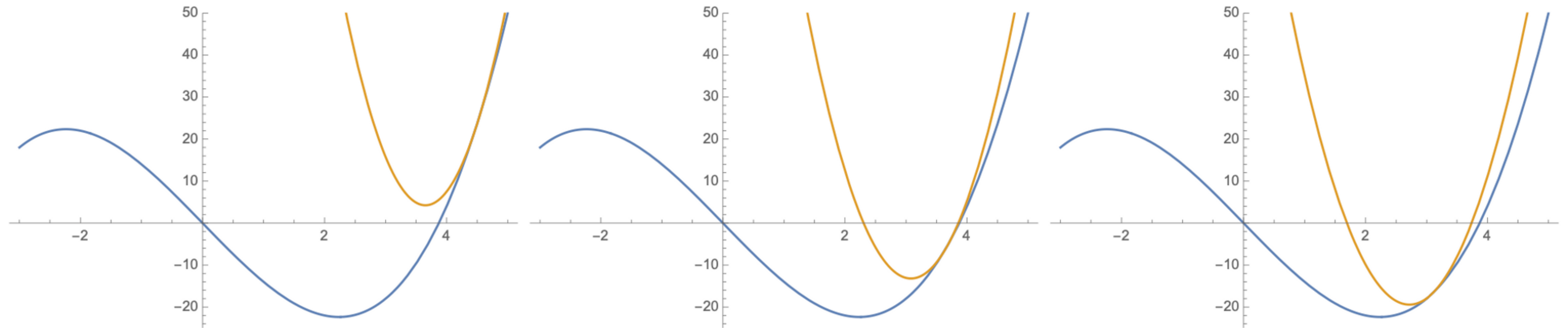
Newton's method uses second-order partial derivatives (Hessian matrices) to characterize the curvature of the loss function.

$$\mathcal{L} = \sum_{i=1}^k \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \mu_i\|_2^2$$

Expectation step: $y_j \leftarrow \arg \min \|\mathbf{x}_j - \mu_i\|_2^2$

Maximization step: $\mu_i \leftarrow \mu_i - \mathbb{H}^{-1} \frac{\partial \mathcal{L}}{\partial \mu_i}$

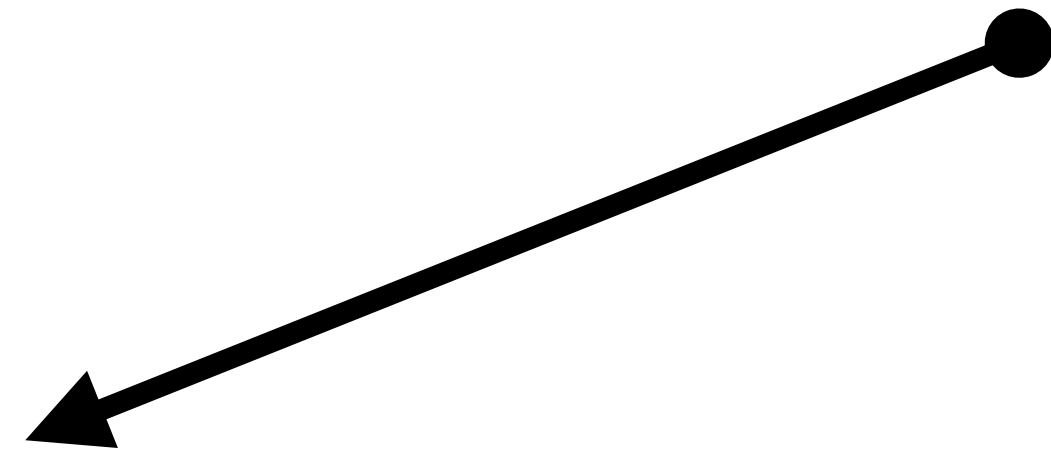
Example: Newton's method



Autograd: demo

K-means clustering with Newton's method

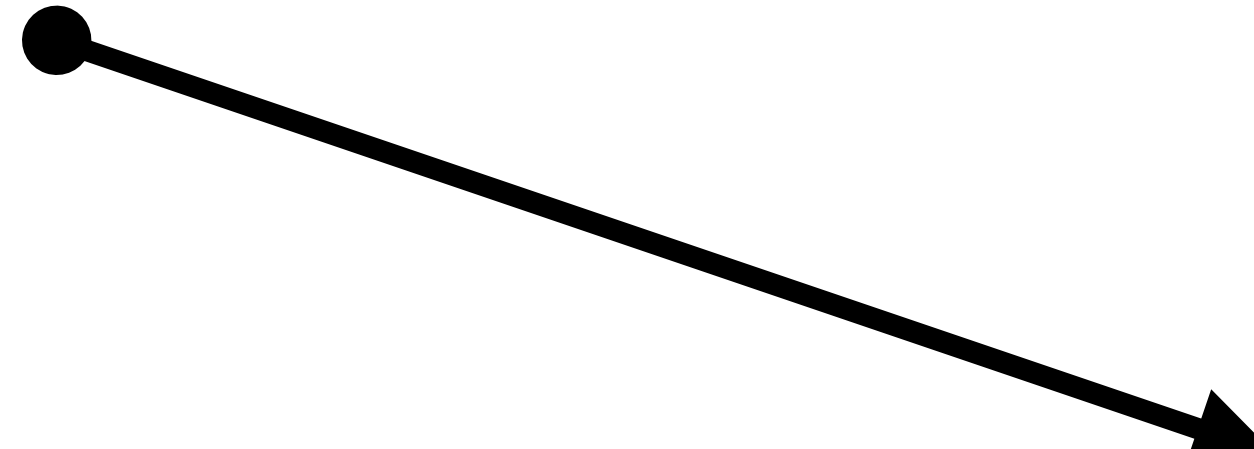
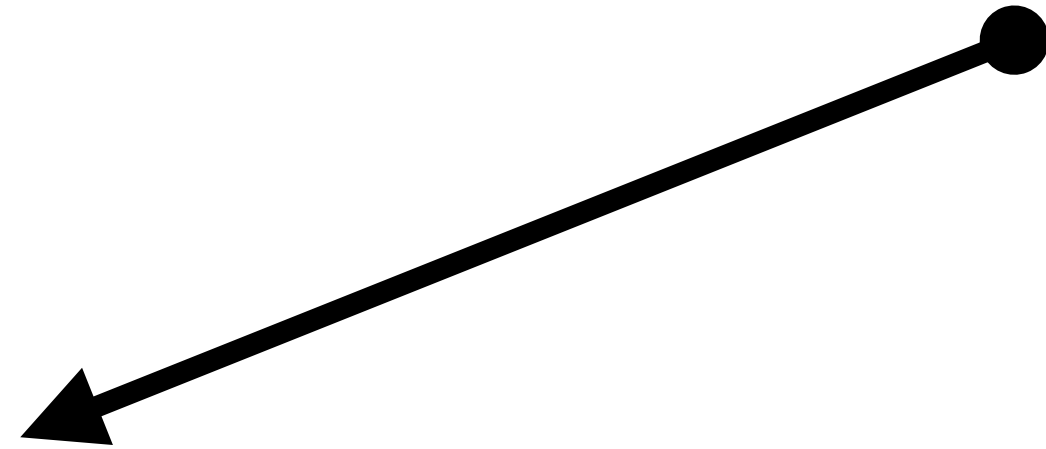
JAX



**Binary cross-entropy loss for
logistic regression**

Principal component analysis

JAX



Binary cross-entropy loss for logistic regression

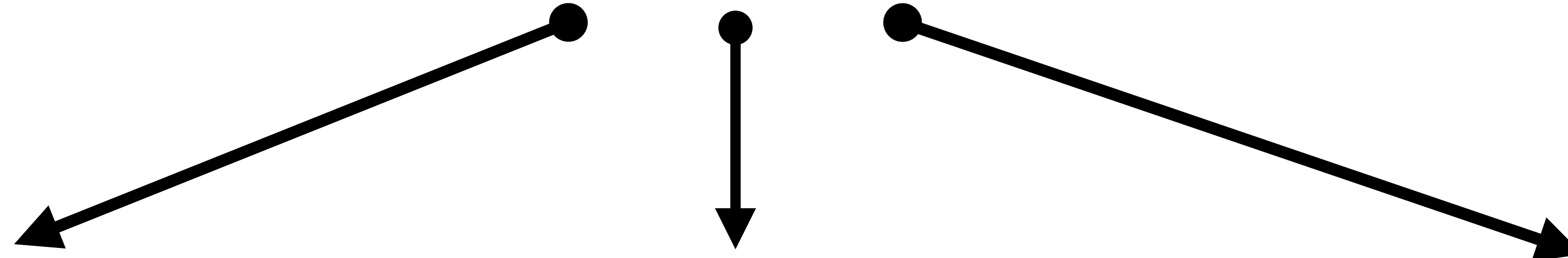
Principal component analysis



Pairwise Euclidean distances between vectors

K-means clustering

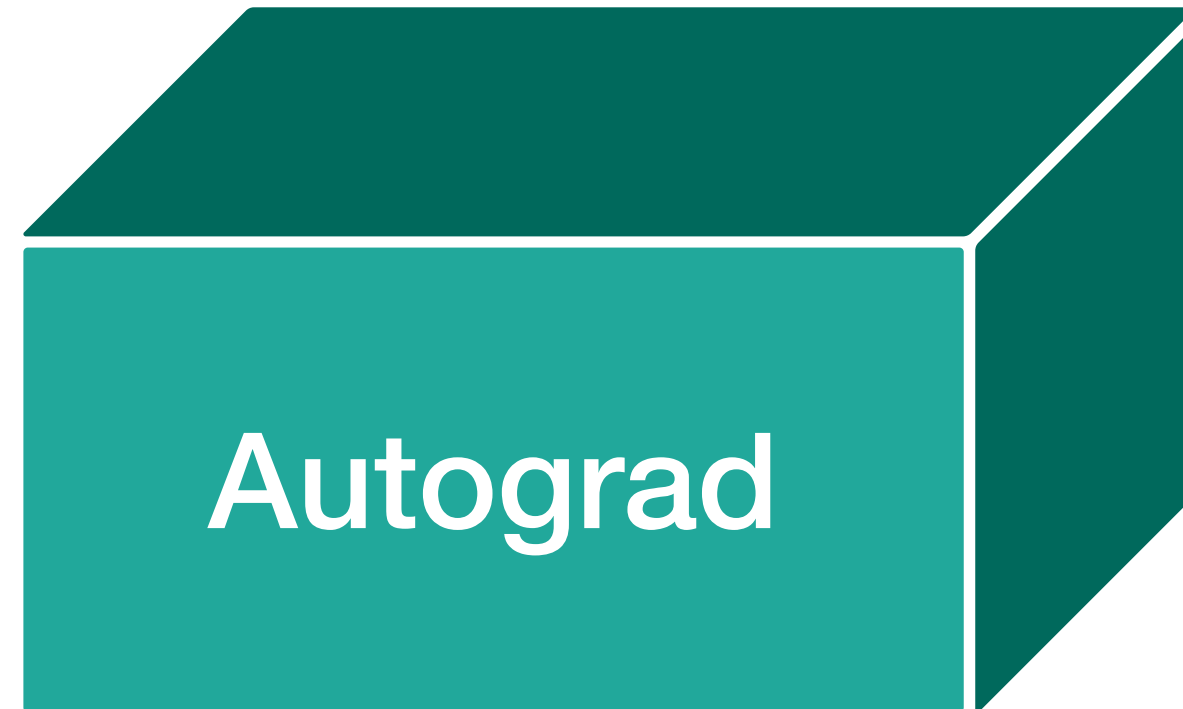
JAX



**JIT
compilation**

**Binary cross-entropy loss for
logistic regression**

Principal component analysis



Autograd

**Derivatives of polynomials
and piecewise functions**

Newton's method



**XLA &
vectorization**

**Pairwise Euclidean distances
between vectors**

K-means clustering

Other topics not covered here

- `jax.scipy`: SciPy API
- `jax.pmap`: parallel execution on multiple devices
- `jax.lax`: lower-level functions for more custom operations
- **PyTrees**: an essential data structure in Jax
- Random number generation
- Ahead-of-time compilation
- Forward- and reverse-mode autodiff

Why use Jax? Its ecosystem

- **Flax:** Neural network library
- **Optax:** Gradient processing and optimizers
- **Jraph:** Graph neural networks
- **NumPyro:** Probabilistic programming
- **HuggingFace:** Pretrained models

Why use Jax?

Why use Jax?

JAX, M.D.

A Framework for Differentiable Physics

Samuel S. Schoenholz
Google Research: Brain Team
schsam@google.com

Ekin D. Cubuk
Google Research: Brain Team
cubuk@google.com

Molecular dynamics

Why use Jax?

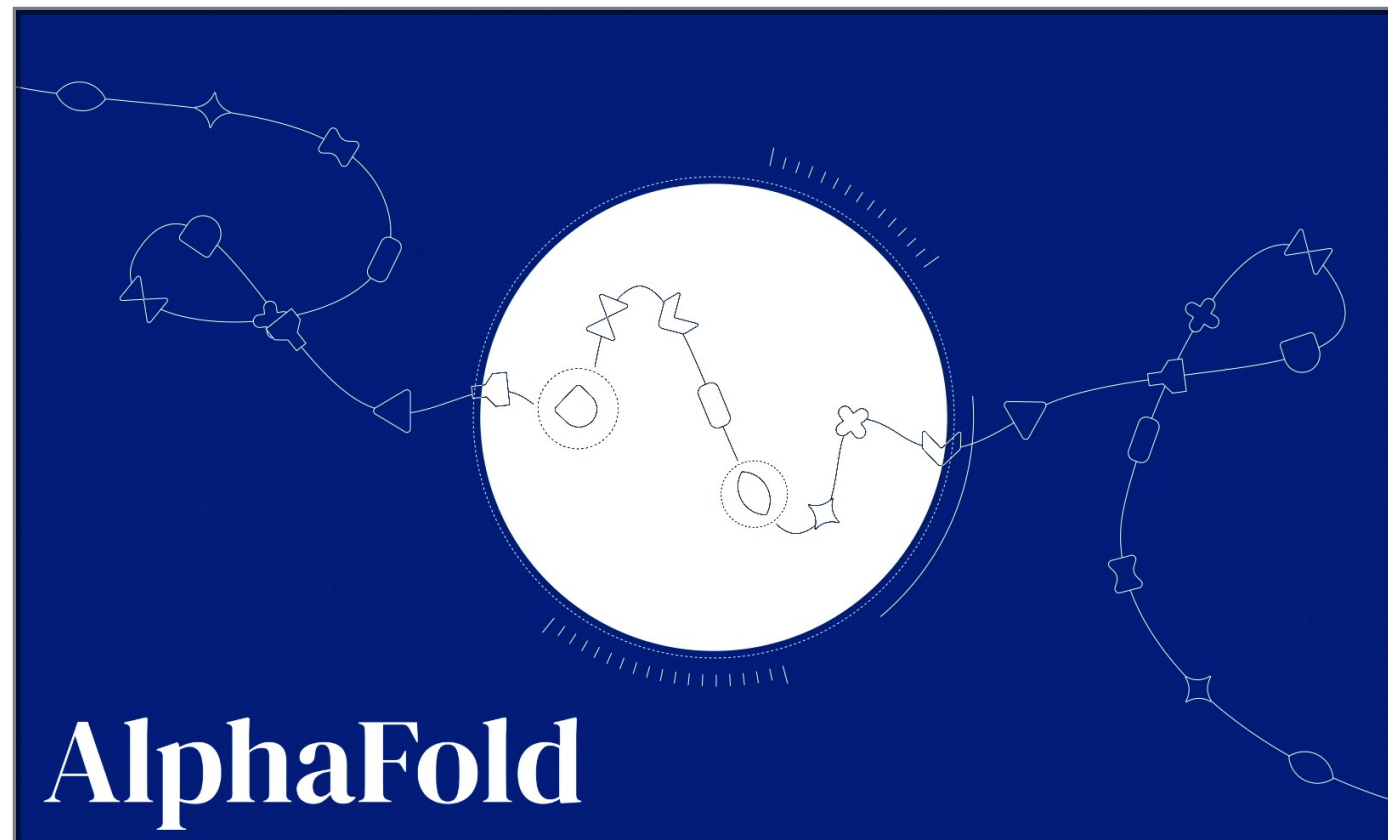
JAX, M.D.

A Framework for Differentiable Physics

Samuel S. Schoenholz
Google Research: Brain Team
schsam@google.com

Ekin D. Cubuk
Google Research: Brain Team
cubuk@google.com

Molecular dynamics



Structure prediction

Why use Jax?

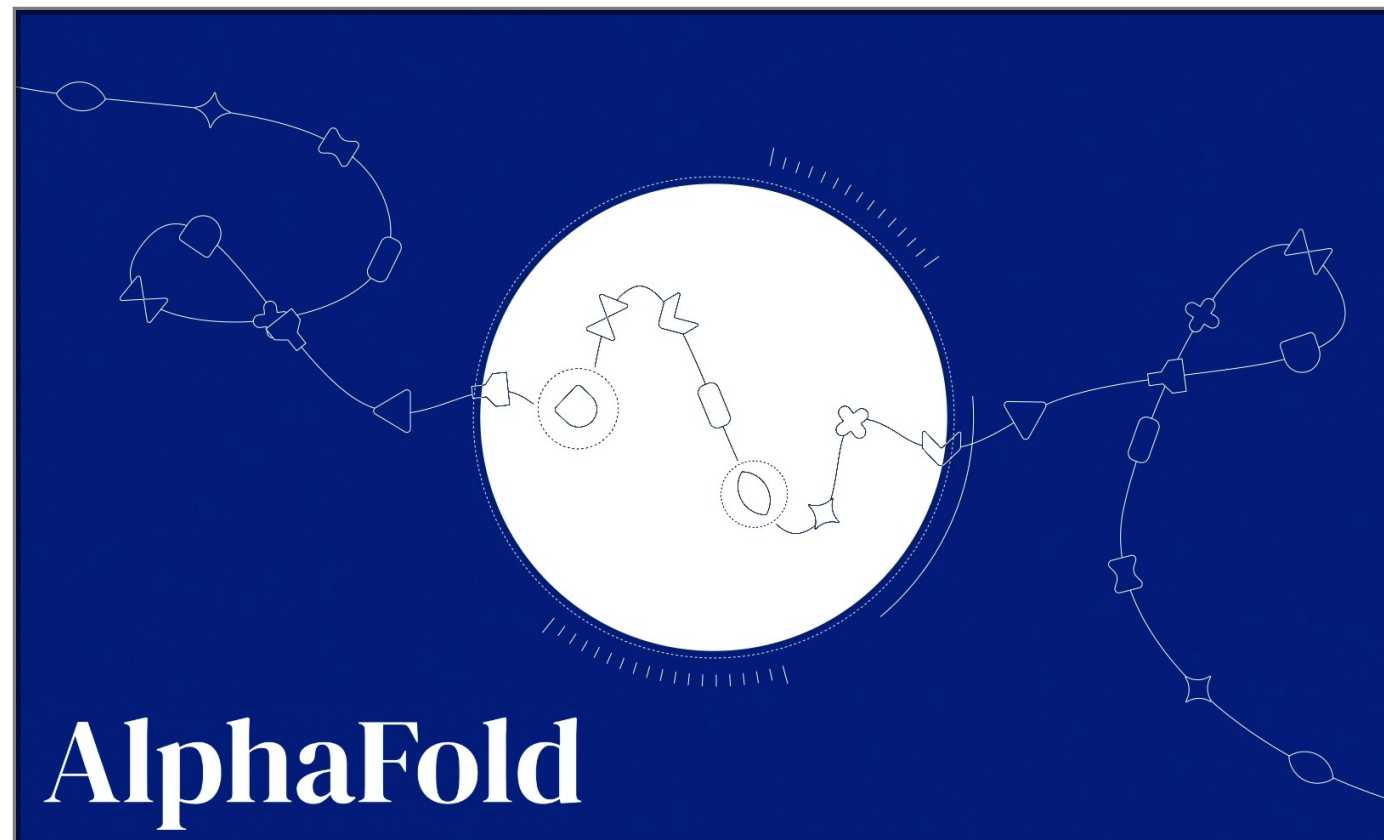
JAX, M.D.

A Framework for Differentiable Physics

Samuel S. Schoenholz
Google Research: Brain Team
schsam@google.com

Ekin D. Cubuk
Google Research: Brain Team
cubuk@google.com


Molecular dynamics



Structure prediction

JOURNAL ARTICLE

End-to-end learning of multiple sequence alignments with differentiable Smith–Waterman

Samantha Petti, Nicholas Bhattacharya, Roshan Rao, Justas Dauparas, Neil Thomas, Juannan Zhou, Alexander M Rush, Peter Koo, Sergey Ovchinnikov 

Bioinformatics, Volume 39, Issue 1, January 2023, btac724, <https://doi.org/10.1093/bioinformatics/btac724>

[/bioinformatics/btac724](https://doi.org/10.1093/bioinformatics/btac724)

Published: 10 November 2022 [Article history](#) 

Differentiable multiple sequence alignment

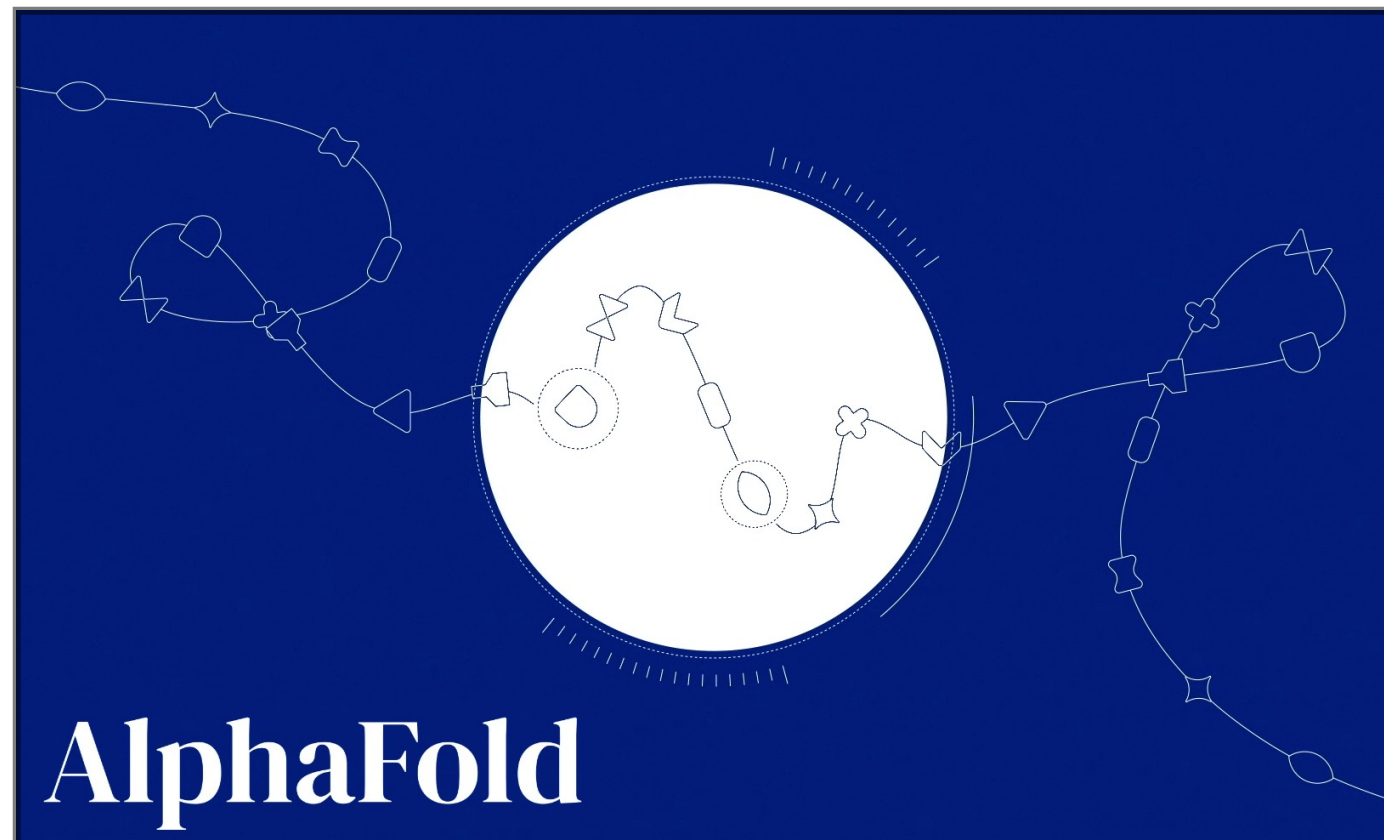
Why use Jax?

JAX, M.D. A Framework for Differentiable Physics

Samuel S. Schoenholz
Google Research: Brain Team
schsam@google.com

Ekin D. Cubuk
Google Research: Brain Team
cubuk@google.com


Molecular dynamics



Structure prediction

JOURNAL ARTICLE

End-to-end learning of multiple sequence alignments with differentiable Smith–Waterman

Samantha Petti, Nicholas Bhattacharya, Roshan Rao, Justas Dauparas, Neil Thomas, Juannan Zhou, Alexander M Rush, Peter Koo, Sergey Ovchinnikov 

Bioinformatics, Volume 39, Issue 1, January 2023, btac724, <https://doi.org/10.1093/bioinformatics/btac724>

Published: 10 November 2022 [Article history](#) ▾

Differentiable multiple sequence alignment

Biological metrics and benchmarks

scib-metrics

build passing docs passing

Accelerated and Python-only metrics for benchmarking single-cell integration outputs.

This package contains implementations of metrics for evaluating the performance of single-cell omics data integration methods. The implementations of these metrics use `jax` when possible for jit-compilation and hardware acceleration. All implementations are in Python.

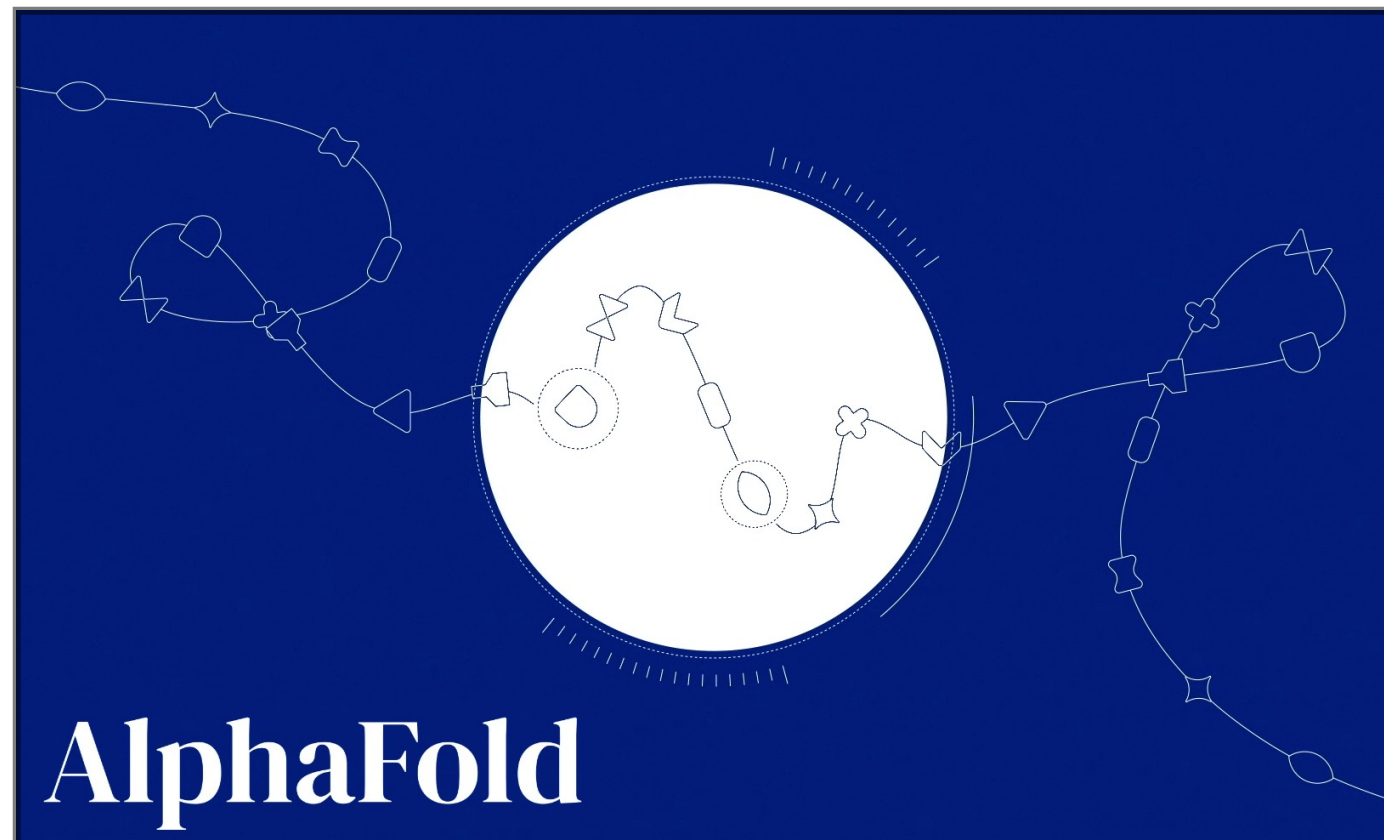
Why use Jax?

JAX, M.D. A Framework for Differentiable Physics

Samuel S. Schoenholz
Google Research: Brain Team
schsam@google.com

Ekin D. Cubuk
Google Research: Brain Team
cubuk@google.com

Molecular dynamics



Structure prediction

JOURNAL ARTICLE

End-to-end learning of multiple sequence alignments with differentiable Smith–Waterman

Samantha Petti, Nicholas Bhattacharya, Roshan Rao, Justas Dauparas, Neil Thomas, Juannan Zhou, Alexander M Rush, Peter Koo, Sergey Ovchinnikov

Bioinformatics, Volume 39, Issue 1, January 2023, btac724, <https://doi.org/10.1093/bioinformatics/btac724>

Published: 10 November 2022 Article history

Differentiable multiple sequence alignment

Biological metrics and benchmarks

scib-metrics

build passing docs passing

Accelerated and Python-only metrics for benchmarking single-cell integration outputs.

This package contains implementations of metrics for evaluating the performance of single-cell omics data integration methods. The implementations of these metrics use `jax` when possible for jit-compilation and hardware acceleration. All implementations are in Python.

Models 8,214 Filter by name new Full-text search Sort: Most Downloads

bert-base-uncased Updated Nov 16, 2022 • 68.8M • 826	jonatasgrosman/wav2vec2-large-xlsr-53-english Updated Mar 25 • 59.8M • 103
gpt2 Updated Dec 16, 2022 • 24.3M • 1.02k	xlm-roberta-base Updated Apr 7 • 23M • 280
microsoft/resnet-50 Updated Mar 10 • 14.1M • 61	openai/clip-vit-large-patch14 Updated Oct 4, 2022 • 13.9M • 381
roberta-base Updated Mar 6 • 12.2M • 160	distilbert-base-uncased Updated Nov 16, 2022 • 9.8M • 194

Large language models

Thanks for listening! Questions?

Colab notebook: <https://tinyurl.com/ccb-intro-to-jax>

Email: martinkim (at) berkeley.edu