ICLR

Fraunhofer IOSB

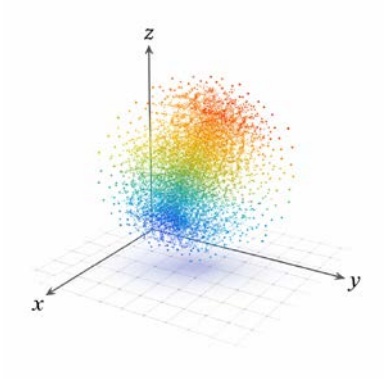
It's All Just Vectorization: einx, a Universal Notation for Tensor Operations

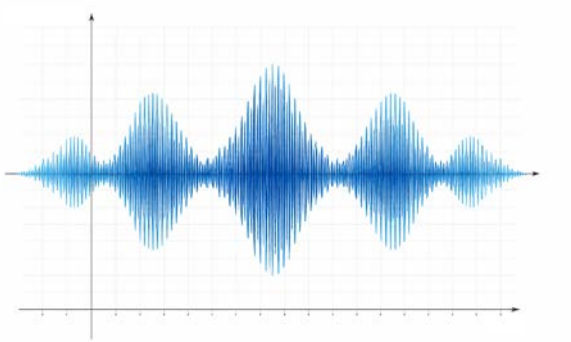
Florian Fervers, Sebastian Bullinger,
Christoph Bodensteiner, Michael Arens

Website: https://github.com/fferflo/einx

Introduction

Tensor: Medium for data

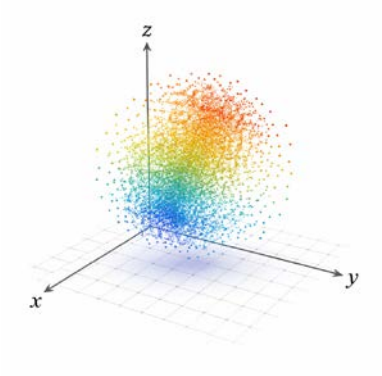


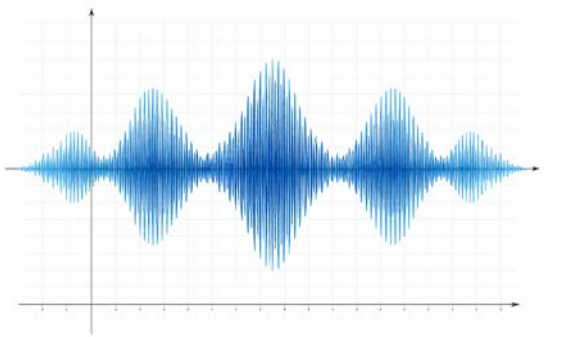


$$A = \begin{bmatrix} 12 & 7 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Introduction

Tensor: Medium for data





$$A = \begin{bmatrix} 12 & 7 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

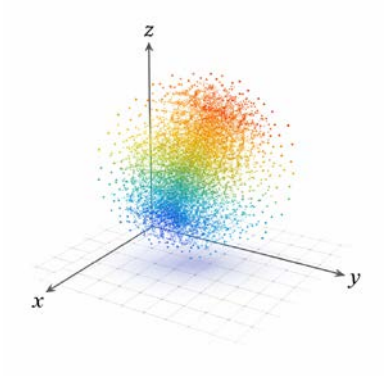
Tensor operation: Manipulation of data

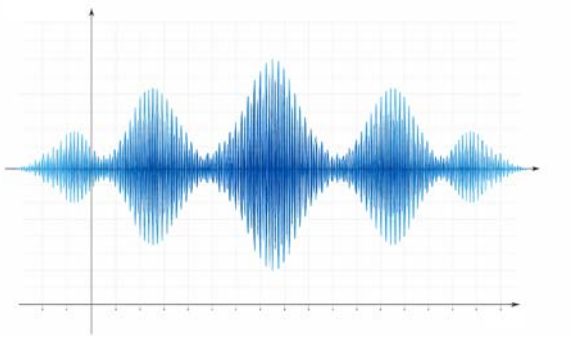
```
x[:, np.newaxis, :] + y[:, :, np.newaxis]
np.sum(x, axis=1, keepdims=True)
np.transpose(x, (2, 0, 1, 3))
np.stack([x, y], axis=0)
np.concatenate([x, y], axis=0)
torch.take(x, y)
torch.gather(x, 0, y)
```

```
torch.take_along_dim(x, y, dim=0)
torch.index_select(x, 1, y)
np.matmul(x, y)
np.dot(x, y)
np.tensordot(x, y, axes=(0, 1))
np.inner(x, y)
```

Introduction

Tensor: Medium for data





$$A = \begin{bmatrix} 12 & 7 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

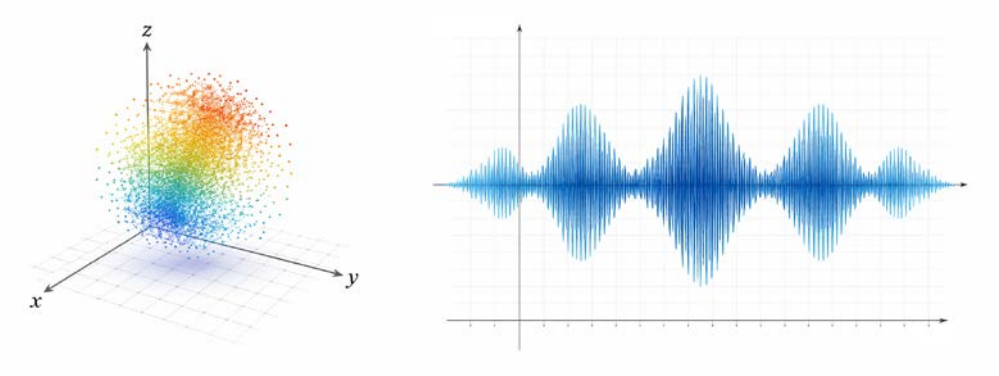
Tensor operation: Manipulation of data

```
x[:, np.newaxis, :] + y[:, :, np.newaxis]
np.sum(x, axis=1, keepdims=True)
np.transpose(x, (2, 0, 1, 3))
np.stack([x, y], axis=0)
np.concatenate([x, y], axis=0)
torch.take(x, y)
torch.gather(x, 0, y)
```

```
torch.take_along_dim(x, y, dim=0)
torch.index_select(x, 1, y)
np.matmul(x, y)
np.dot(x, y)
np.tensordot(x, y, axes=(0, 1))
np.inner(x, y)
np.einsum("ab,bc->ac", x, y)
```

Introduction

Tensor: Medium for data



$$A = \begin{bmatrix} 12 & 7 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Tensor operation: Manipulation of data

```
x[:, np.newaxis, :] + y[:, :, np.newaxis]
np.sum(x, axis=1, keepdims=True)
np.transpose(x, (2, 0, 1, 3))
np.stack([x, y], axis=0)
np.concatenate([x, y], axis=0)
torch.take(x, y)
torch.gather(x, 0, y)
```

```
torch.take_along_dim(x, y, dim=0)
torch.index_select(x, 1, y)
np.matmul(x, y)
np.dot(x, y)
np.tensordot(x, y, axes=(0, 1))
np.inner(x, y)
np.einsum("ab,bc->ac", x, y)
```

→ How do we read and write tensor operations? How do we *think* tensor operations?

Overview: einx notation

Full operation = Elementary operation + Vectorization

Example: Matrix multiplication

Example: Matrix multiplication

Matrix multiplication = Dot-product + Vectorization

Example: Matrix multiplication

Matrix multiplication = Vectorized dot-product

Example: Matrix multiplication

Matrix multiplication = Vectorized dot-product

loop notation

```
for a in range(...):
    for c in range(...):
        z[a, c] = dot(x[a, :], y[:, c])
```

Example: Matrix multiplication

Matrix multiplication = Vectorized dot-product

loop notation

```
for a in range(...):
    for c in range(...):
        z[a, c] = dot(x[a, :], y[:, c])
```

Example: Matrix multiplication

Matrix multiplication = Vectorized dot-product

loop notation

```
for a in range(...):
    for c in range(...):
        z[a, c] = dot(x[a, :], y[:, c])
```

Example: Matrix multiplication

Matrix multiplication = Vectorized dot-product

loop notation

```
for a in range(...):
    for c in range(...):
        z[a, c] = dot(x[a, :], y[:, c])
```

Example: Matrix multiplication

Matrix multiplication = Vectorized dot-product

loop notation

```
for a in range(...):
    for c in range(...):
        z[a, c] = dot(x[a, :], y[:, c])
```

Yields the
same output

einx notation

Example: Matrix multiplication

Matrix multiplication = Vectorized dot-product

loop notation

```
for a in range(...):
    for c in range(...):
        z[a, c] = dot(x[a, :], y[:, c])
```

einx notation

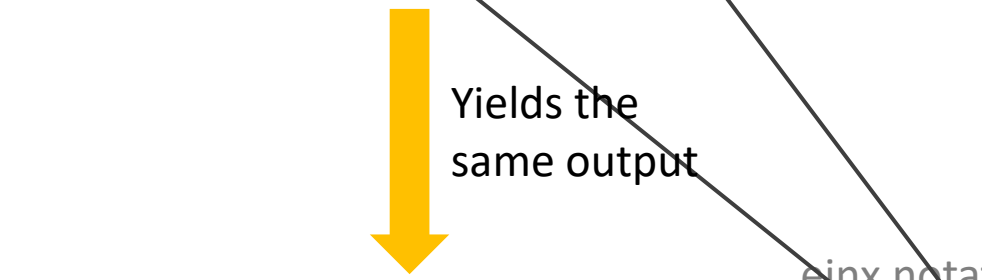
```
einx.dot
```

Example: Matrix multiplication

Matrix multiplication = Vectorized dot-product

loop notation

```
for a in range(...):
    for c in range(...):
        z[a, c] = dot(x[a, :], y[:, c])
```



einx notation

```
z = einx.dot(                                  x, y)
```

Example: Matrix multiplication

Matrix multiplication = Vectorized dot-product

loop notation

```
for a in range(...):
    for c in range(...):
        z[a, c] = dot(x[a, :], y[:, c])
```

Yields the same output

einx notation

```
z = einx.dot("a [b], [b] c -> a  c", x, y)
```

einx expression

Example: Matrix multiplication

Matrix multiplication = Vectorized dot-product

loop notation

```
for a in range(...):
    for c in range(...):
        z[a, c] = dot(x[a, :], y[:, c])
```

Yields the same output

einx notation

```
z = einx.dot("a [b], [b] c -> a  c", x, y)
```

```
x[a, :] y[:, c] z[a, c]
```

Example: Matrix multiplication

Matrix multiplication = Vectorized dot-product

loop notation

```
for a in range(...):
    for c in range(...):
        z[a, c] = dot(x[a, :], y[:, c])
```

Yields the
same output

einx notation

```
z = einx.dot("a [b], [b] c -> a  c", x, y)
```

```
x[a, :] y[:, c] z[a, c]
```

Example: Matrix multiplication

Matrix multiplication = Vectorized dot-product

loop notation

```
for a in range(...):
    for c in range(...):
        z[a, c] = dot(x[a, :], y[:, c])
```

Yields the same output

einx notation

```
z = einx.dot("a [b], [b] c -> a  c", x, y)
```

```
x[a, :] y[:, c] z[a, c]
```

Example: Matrix multiplication

Matrix multiplication = Vectorized dot-product

loop notation

```
for a in range(...):
    for c in range(...):
        z[a, c] = dot(x[a, :], y[:, c])
```

einx notation

```
z = einx.dot("a [b], [b] c -> a  c", x, y)
```

```
x[a, :] y[:, c] z[a, c]
```

Example: Matrix multiplication

Matrix multiplication = Vectorized dot-product

loop notation

```
for a in range(...):
    for c in range(...):
        z[a, c] = dot(x[a, :], y[:, c])
```

Yields the
same output

einx notation

```
z = einx.dot("a [b], [b] c -> a  c", x, y)
```

Example: Softmax

“Softmax along an axis” = Vectorized softmax

```
y = einx.softmax("a [b] -> a [b]", x)
```

Example: Softmax

"Softmax along an axis" = Vectorized softmax

```
y = einx.softmax("a [b] -> a [b]", x)
```

Example: Outer product

Outer product = Vectorized scalar multiplication

```
z = einx.multiply("a, b -> a b", x, y)
```

Example: Axis permutation

Axis permutation =

Example: Axis permutation

Axis permutation = Vectorized identity map

```
y = einx.id("a b c -> c b a", x)
```

Example: Axis permutation

Axis permutation = Vectorized identity map

```
y = einx.id("a b c -> c b a", x)
```

Example: Broadcasting

Broadcasting = Vectorized identity map

```
y = einx.id("a -> b a", x, b=42)
```

Elementary Operation ⊥ Vectorization

Broadcasting:

```
einx.id("a -> b a", x, b=42)
```

Axis permutation:

```
einx.id("a b c -> c b a", x)
```

Elementary Operation ⊥ Vectorization

Numpy-like notation

```
np.broadcast_to(x, (42, 5))
```

```
np.transpose(x, (2, 1, 0))
```

einx notation

```
einx.id("a -> b a", x, b=42)
```

```
einx.id("a b c -> c b a", x)
```

Elementary Operation ⊥ Vectorization

Numpy-like notation	einx notation
`np.broadcast_to(x, (42, 5))`	`einx.id("a -> b a", x, b=42)`
`np.transpose(x, (2, 1, 0))`	`einx.id("a b c -> c b a", x)`
`np.squeeze(x, axis=1)`	`einx.id("a 1 -> a", x)`
`np.reshape(x, (2, -1))`	`einx.id("a b c -> a (b c)", x)`
`np.concatenate([x, y], axis=-1)`	`einx.id("a b, a c -> a (b + c)", x)`
`np.stack([x, y], axis=-1)`	`einx.id("a, a -> a (1 + 1)", x)`
`np.meshgrid(x, y, indexing="ij")`	`einx.id("a, b -> a b, a b", x, y)`

Elementary Operation ⊥ Vectorization

Numpy-like notation	einx notation
`np.broadcast_to(x, (42, 5))`	`einx.id("a -> b a", x, b=42)`
`np.transpose(x, (2, 1, 0))`	`einx.id("a b c -> c b a", x)`
`np.squeeze(x, axis=1)`	`einx.id("a 1 -> a", x)`
`np.reshape(x, (2, -1))`	`einx.id("a b c -> a (b c)", x)`
`np.concatenate([x, y], axis=-1)`	`einx.id("a b, a c -> a (b + c)", x)`
`np.stack([x, y], axis=-1)`	`einx.id("a, a -> a (1 + 1)", x)`
`np.meshgrid(x, y, indexing="ij")`	`einx.id("a, b -> a b, a b", x, y)`

Elementary Operation ⊥ Vectorization

Numpy-like notation	einx notation
`np.broadcast_to(x, (42, 5))`	`einx.id("a -> b a", x, b=42)`
`np.transpose(x, (2, 1, 0))`	`einx.id("a b c -> c b a", x)`
`np.squeeze(x, axis=1)`	`einx.id("a 1 -> a", x)`
`np.reshape(x, (2, -1))`	`einx.id("a b c -> a (b c)", x)`
`np.concatenate([x, y], axis=-1)`	`einx.id("a b, a c -> a (b + c)", x)`
`np.stack([x, y], axis=-1)`	`einx.id("a, a -> a (1 + 1)", x)`
`np.meshgrid(x, y, indexing="ij")`	`einx.id("a, b -> a b, a b", x, y)`

– Names don’t communicate behavior

Elementary Operation ⊥ Vectorization

Numpy-like notation

```
np.broadcast_to(x, (42, 5))
np.transpose(x, (2, 1, 0))
np.squeeze(x, axis=1)
np.reshape(x, (2, -1))
np.concatenate([x, y], axis=-1)
np.stack([x, y], axis=-1)
np.meshgrid(x, y, indexing="ij")
```

einx notation

```
einx.id("a -> b a", x, b=42)
einx.id("a b c -> c b a", x)
einx.id("a 1 -> a", x)
einx.id("a b c -> a (b c)", x)
einx.id("a b, a c -> a (b + c)", x)
einx.id("a, a -> a (1 + 1)", x)
einx.id("a, b -> a b, a b", x, y)
```

– Names don’t communicate behavior

+ Names communicate behavior of elementary operation

Elementary Operation ⊥ Vectorization

Numpy-like notation

```
np.broadcast_to(x, (42, 5))
np.transpose(x, (2, 1, 0))
np.squeeze(x, axis=1)
np.reshape(x, (2, -1))
np.concatenate([x, y], axis=-1)
np.stack([x, y], axis=-1)
np.meshgrid(x, y, indexing="ij")
```

einx notation

```
einx.id("a -> b a", x, b=42)
einx.id("a b c -> c b a", x)
einx.id("a 1 -> a", x)
einx.id("a b c -> a (b c)", x)
einx.id("a b, a c -> a (b + c)", x)
einx.id("a, a -> a (1 + 1)", x)
einx.id("a, b -> a b, a b", x, y)
```

– Names don’t communicate behavior

+ Names communicate behavior of elementary operation
+ Explicit vectorization

Elementary Operation ⊥ Vectorization

Numpy-like notation

```
np.broadcast_to(x, (42, 5))
np.transpose(x, (2, 1, 0))
np.squeeze(x, axis=1)
np.reshape(x, (2, -1))
np.concatenate([x, y], axis=-1)
np.stack([x, y], axis=-1)
np.meshgrid(x, y, indexing="ij")
```

einx notation

```
einx.id("a -> b a", x, b=42)
einx.id("a b c -> c b a", x)
einx.id("a 1 -> a", x)
einx.id("a b c -> a (b c)", x)
einx.id("a b, a c -> a (b + c)", x)
einx.id("a, a -> a (1 + 1)", x)
einx.id("a, b -> a b, a b", x, y)
```

– Names don’t communicate behavior

+ Names communicate behavior of elementary operation
+ Explicit vectorization

Elementary Operation ⊥ Vectorization

Numpy-like notation

```
np.broadcast_to(x, (42, 5))
np.transpose(x, (2, 1, 0))
np.squeeze(x, axis=1)
np.reshape(x, (2, -1))
np.concatenate([x, y], axis=-1)
np.stack([x, y], axis=-1)
np.meshgrid(x, y, indexing="ij")
```

einx notation

```
einx.id("a -> b a", x, b=42)
einx.id("a b c -> c b a", x)
einx.id("a 1 -> a", x)
einx.id("a b c -> a (b c)", x)
einx.id("a b, a c -> a (b + c)", x)
einx.id("a, a -> a (1 + 1)", x)
einx.id("a, b -> a b, a b", x, y)
```

– Names don’t communicate behavior
– Superfluous concepts

+ Names communicate behavior of elementary operation
+ Explicit vectorization

Elementary Operation ⊥ Vectorization

Numpy-like notation

```
np.broadcast_to(x, (42, 5))
np.transpose(x, (2, 1, 0))
np.squeeze(x, axis=1)
np.reshape(x, (2, -1))
np.concatenate([x, y], axis=-1)
np.stack([x, y], axis=-1)
np.meshgrid(x, y, indexing="ij")
```

einx notation

```
einx.id("a -> b a", x, b=42)
einx.id("a b c -> c b a", x)
einx.id("a 1 -> a", x)
einx.id("a b c -> a (b c)", x)
einx.id("a b, a c -> a (b + c)", x)
einx.id("a, a -> a (1 + 1)", x)
einx.id("a, b -> a b, a b", x, y)
```

– Names don’t communicate behavior
– Superfluous concepts
– Inconsistent interfaces

+ Names communicate behavior of elementary operation
+ Explicit vectorization

Elementary Operation ⊥ Vectorization

Numpy-like notation

```
np.broadcast_to(x, (42, 5))
np.transpose(x, (2, 1, 0))
np.squeeze(x, axis=1)
np.reshape(x, (2, -1))
np.concatenate([x, y], axis=-1)
np.stack([x, y], axis=-1)
np.meshgrid(x, y, indexing="ij")
```

einx notation

```
einx.id("a -> b a", x, b=42)
einx.id("a b c -> c b a", x)
einx.id("a 1 -> a", x)
einx.id("a b c -> a (b c)", x)
einx.id("a b, a c -> a (b + c)", x)
einx.id("a, a -> a (1 + 1)", x)
einx.id("a, b -> a b, a b", x, y)
```

– Names don’t communicate behavior
– Superfluous concepts
– Inconsistent interfaces

+ Names communicate behavior of elementary operation
+ Explicit vectorization
+ Universal notation across operations

Elementary Operation ⊥ Vectorization

Numpy-like notation

```
np.matmul(x, y)
np.dot(x, y)
np.tensordot(x, y, axes=(0, 1))
np.inner(x, y)
```

einx notation

```
einx.dot("a [b], [b] c -> a c", x, y)
einx.dot("x [a], y [a] b -> x y b", x, y)
einx.dot("[a] b, c [a] -> b c", x, y)
einx.dot("x [a], y [a] -> x y", x, y)
```

– Names don’t communicate behavior
– Superfluous concepts
– Inconsistent interfaces

+ Names communicate behavior of elementary operation
+ Explicit vectorization
+ Universal notation across operations

Elementary Operation ⊥ Vectorization

Numpy-like notation	einx notation
`torch.take(x, y)`	`einx.get_at("[x], ... -> ... ", x, y)`
`torch.gather(x, 0, y)`	`einx.get_at("[x] b c, i b c -> i b c", x, y)`
`torch.index_select(x, 1, y)`	`einx.get_at("a [x] c, i -> a i c", x, y)`
`tf.gather_nd(x, y)`	`einx.get_at("[...], b [i] -> b", x, y)`
`tf.gather_nd(x, y, batch_dims=1)`	`einx.get_at("a [...], a b [i] -> a b", x, y)`
`x[y[:, 0], y[:, 1]]`	`einx.get_at("[x y], a [2] -> a", x, y)`

– Names don’t communicate behavior
– Superfluous concepts
– Inconsistent interfaces

+ Names communicate behavior of elementary operation
+ Explicit vectorization
+ Universal notation across operations

Implementation

einx operations are compiled to function calls in a given framework:

```
einx.add("a b, b c -> c b a", x, y)
```

Implementation

einx operations are compiled to function calls in a given framework:

```
einx.add("a b, b c -> c b a", x, y)
```


```
import numpy as np
def op(a, b):
    a = np.transpose(a, (1, 0))
    a = np.reshape(a, (1, 3, 2))
    b = np.transpose(b, (1, 0))
    b = np.reshape(b, (4, 3, 1))
    c = np.add(a, b)
    return c
```

Implementation

einx operations are compiled to function calls in a given framework:

```
einx.add("a b, b c -> c b a", x, y, backend="jax.vmap")
```

is compiled to

```
import jax.numpy as jnp
import jax
def op(a, b):
    c = jax.vmap(jnp.add, in_axes=(None, 0), out_axes=0)
    c = jax.vmap(c, in_axes=(0, 0), out_axes=1)
    c = jax.vmap(c, in_axes=(0, None), out_axes=2)
    d = c(a, b)
    return d
```

Supported Operations

1. **Built-in operations:** `einx.{id|add|dot|multiply|logical_and|softmax|…}`

Supported Operations

1. Built-in operations: `einx.{id|add|dot|multiply|logical_and|softmax|…}`

2. User-defined operations:

```
def foo(x, y):
    # ...
    return z
```

Supported Operations

1. Built-in operations: `einx.{id|add|dot|multiply|logical_and|softmax|…}`

2. User-defined operations:

```
def foo(x, y):
    # ...
    return z

einfoo = einx.jax.adapt_with_vmap(foo)
```

Supported Operations

1. Built-in operations: `einx.{id|add|dot|multiply|logical_and|softmax|…}`

2. User-defined operations:

```
def foo(x, y):
    # ...
    return z

einfoo = einx.jax.adapt_with_vmap(foo)

z = einfoo("[c d] a, b -> a [c] b", x, y)
```

Conclusion

einx, a novel notation for tensor programming:

- Universal
- Self-documenting
- Interpretable
- Better mental model

einx, a Python library:

- Compatible with Numpy, PyTorch, Jax, Tensorflow, …
- Supports built-in and user-defined operations

Conclusion

einx, a novel notation for tensor programming:

- Universal
- Self-documenting
- Interpretable
- Better mental model

einx, a Python library:

- Compatible with Numpy, PyTorch, Jax, Tensorflow, …
- Supports built-in and user-defined operations

```
pip install einx
```

```
import einx
import torch

x = torch.rand(10, 20)

y = einx.sum("a [b]", x)
```

Conclusion

einx, a novel notation for tensor programming:

- Universal
- Self-documenting
- Interpretable
- Better mental model

einx, a Python library:

- Compatible with Numpy, PyTorch, Jax, Tensorflow, …
- Supports built-in and user-defined operations

```
pip install einx
```

```
import einx
import torch

x = torch.rand(10, 20)

y = einx.sum("a [b]", x)
```

```
# Multi-head attention:
a = einx.dot("b q (h [c]), b k (h [c]) -> b q k h", q, k, h=8)
a = einx.softmax("b q [k] h", a * s)
x = einx.dot("b q [k] h, b [k] (h c) -> b q (h c)", a, v)
```