



Linear Algebra for Matrix Memories

Matt Gormley
Lecture 5
September 12, 2018

Q&A

HEBIAN LEARNING & MATRIX MEMORIES

Storing a Pattern Pair in a Matrix

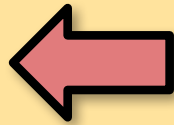
In-Class Exercise

1. Given vectors:
 - stimulus: $x = [0.5, -0.5]^T$
 - response: $y = [1, 2]^T$
2. Compute their outer product:
$$W = yx^T$$
3. Compute the predicted response:
$$r = Wx$$
4. Does the relationship you observe here between r and y always hold?

Storing a Pattern Pair in a Matrix

In-Class Exercise

1. Given vectors:
 - stimulus: $x = [0.5, -0.5]^T$
 - response: $y = [1, 2]^T$
2. Compute their outer product:
 $W = yx^T$
3. Compute the predicted response:
 $r = Wx$
4. Does the relationship you observe here between r and y always hold?



The typo in the slide last time broke this example.

It read $W = xy^T$

1. Why does the **corrected** version offer a different answer for Q4 than the **typo** version?
2. How could we change the definition $r = Wx$ such that we **recover** the desired predicted response?

LINEAR ALGEBRA

Linear Algebra: Vector Properties

Chalkboard

- Orthogonal
- Normalized
- Othonormal

HEBIAN LEARNING & MATRIX MEMORIES

Matrix Memories

Chalkboard

- Matrix Multiplication
 - examples of vector-vector, matrix-vector, matrix-matrix
 - views of matrix-matrix
- Compactly representing the sum of several outer products
- Storing several pattern pairs in a single weight matrix

Ex: Matrix Memories in Numpy

matrix_memories_1.py

```
import numpy as np

s = np.array([[1, 0, 1, -2, 2]]).T
t = np.array([[2, 1, -1]]).T
W = np.matmul(t, s.T)
r = np.matmul(W, s)
n = np.matmul(W, s) / np.matmul(s.T, s)
assert np.array_equal(n, t)

print('s =\n',s)
print('t =\n',t)
print('W =\n',W)
print('r =\n',r)
print('n =\n',n)
```

Goal: Construct an association matrix W that associates the pattern pair $[1, 0, 1, -2, 2] \rightarrow [2, 1, -1]$

(It works up to normalization!)

stdout

```
s =
[[ 1]
 [ 0]
 [ 1]
 [-2]
 [ 2]]
t =
[[ 2]
 [ 1]
 [-1]]
W =
[[ 2  0  2 -4  4]
 [ 1  0  1 -2  2]
 [-1  0 -1  2 -2]]
r =
[[ 20]
 [ 10]
 [-10]]
n =
[[ 2.]
 [ 1.]
 [-1.]]
```

Ex: Matrix Memories in Numpy

matrix_memories_2.py

```
import numpy as np

s = np.array([[.5, -.5, .5, -.5]]).T
t = np.array([[2, 1, -1]]).T
W = np.matmul(t, s.T)
r = np.matmul(W, s)
assert np.array_equal(r, t)

print('s =\n',s)
print('t =\n',t)
print('W =\n',W)
print('r =\n',r)
```

Goal: Construct an association matrix W that associates the pattern pair $[.5, -.5, .5, -.5] \rightarrow [2, 1, -1]$

stdout

```
s =
[[ 1]
 [ 0]
 [ 1]
 [-2]
 [ 2]]
t =
[[ 2]
 [ 1]
 [-1]]
W =
[[ 2  0  2 -4  4]
 [ 1  0  1 -2  2]
 [-1  0 -1  2 -2]]
r =
[[ 2.]
 [ 1.]
 [-1.]]
```

Ex: Matrix Memories in Numpy

matrix_memories_3.py

```
import numpy as np

s1 = np.array([[.5, -.5, .5, -.5]]).T
s2 = np.array([[.5, .5, -.5, -.5]]).T
t1 = np.array([[2, 1, -1]]).T
t2 = np.array([[0, -1, 0]]).T
W1 = np.matmul(t1, s1.T)
W2 = np.matmul(t2, s2.T)
W = W1 + W2
r1 = np.matmul(W, s1)
r2 = np.matmul(W, s2)
```

Goal: Construct an association matrix W that associates **two** pattern pairs:

$[.5, -.5, .5, -.5] \rightarrow [2, 1, -1]$ and
 $[.5, .5, -.5, -.5] \rightarrow [0, -1, 0]$

stdout

```
W1 =
[[ 1. -1.  1. -1. ]
 [ 0.5 -0.5  0.5 -0.5]
 [-0.5  0.5 -0.5  0.5]]
W2 =
[[ 0.  0. -0. -0. ]
 [-0.5 -0.5  0.5  0.5]
 [ 0.  0. -0. -0. ]]
W =
[[ 1. -1.  1. -1. ]
 [ 0. -1.  1.  0. ]
 [-0.5  0.5 -0.5  0.5]]
r1 =
[[ 2.]
 [ 1.]
 [-1.]]
r2 =
[[ 0.]
 [-1.]
 [ 0.]]
```

Ex: Matrix Memories in Numpy

matrix_memories_4.py

```
import numpy as np

# Each s-pattern is a column
S = np.array([[0.5, 0.5, 0.5],
              [0.5, -0.5, -0.5],
              [-0.5, 0.5, -0.5],
              [-0.5, -0.5, 0.5]])

# All s-patterns are orthonormal
dot = np.matmul(S.T, S)

# Each t-pattern is a column
U = np.array([[1, 3, 5],
              [2, 4, 6]])

# Construct and sum the weight matrices
W = np.matmul(U, S.T)

# Compute responses
R = np.matmul(W, S)
```

stdout

```
dot=
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

W =
[[ 4.5 -3.5 -1.5  0.5]
 [ 6.  -4.  -2.   0. ]]

R =
[[1. 3. 5.]
 [2. 4. 6.]]
```

Goal: Associate many three pattern pairs

Ex: Matrix Memories in Numpy

matrix_memories_4.py

```
import numpy as np

# Each s-pattern is a column
S = np.array([[0.5, 0.5, 0.5],
              [0.5, -0.5, -0.5],
              [-0.5, 0.5, -0.5],
              [-0.5, -0.5, 0.5]])

# All s-patterns are orthonormal
dot = np.matmul(S.T, S)

# Each t-pattern is a column
U = np.array([[1, 3, 5],
              [2, 4, 6]])

# Construct and sum the weight matrices
W = np.matmul(U, S.T)

# Compute responses
R = np.matmul(W, S)
```


Goal: Associate many three pattern pairs

stdout

```
dot=
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]

W =
[[ 4.5 -3.5  1.5  0.5]
 [ 6.  -4.  -1.  0. ]]

R =
[[1. 3. 5.]
 [2. 4. 6.]]
```



Claim: This approach works for any stimulus matrix S and response matrix U where the columns of S are orthonormal!

Proof: Left as an exercise for Homework 1