# This presentation

- Intuition about linear algebra
- Matrix-vector and matrix-matrix multiplications
- Using linear algebra to express ML algorithms
- Worked example gradient descent in linear algebra

# Language of ML: linear algebra

# Language of ML: linear algebra – what is it?

- Calculations with vectors and matrices

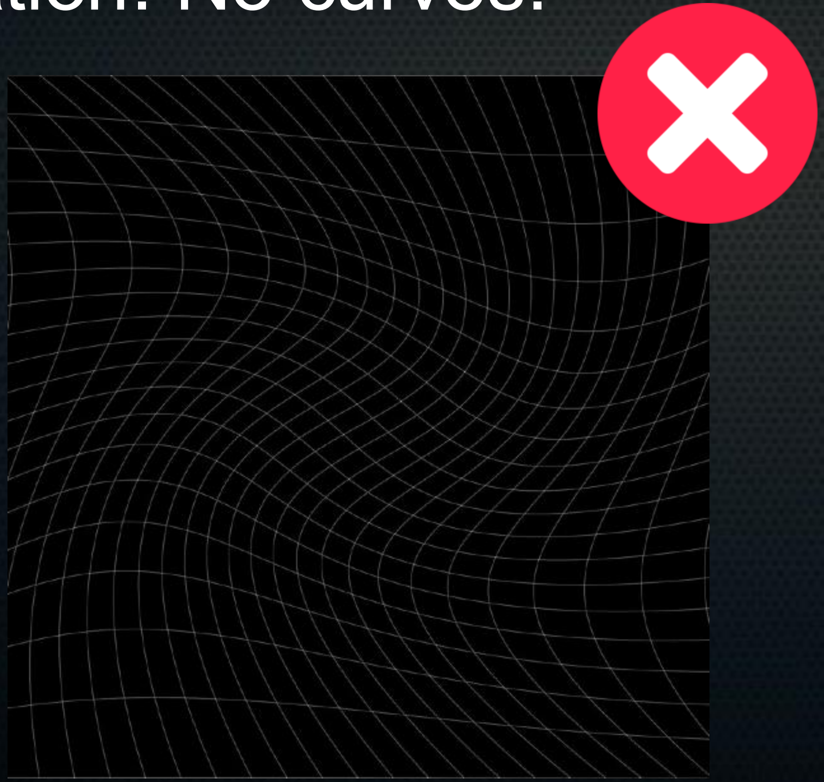
$$\begin{cases} 2x + 3y = 8 \\ x - y = 1 \end{cases}$$

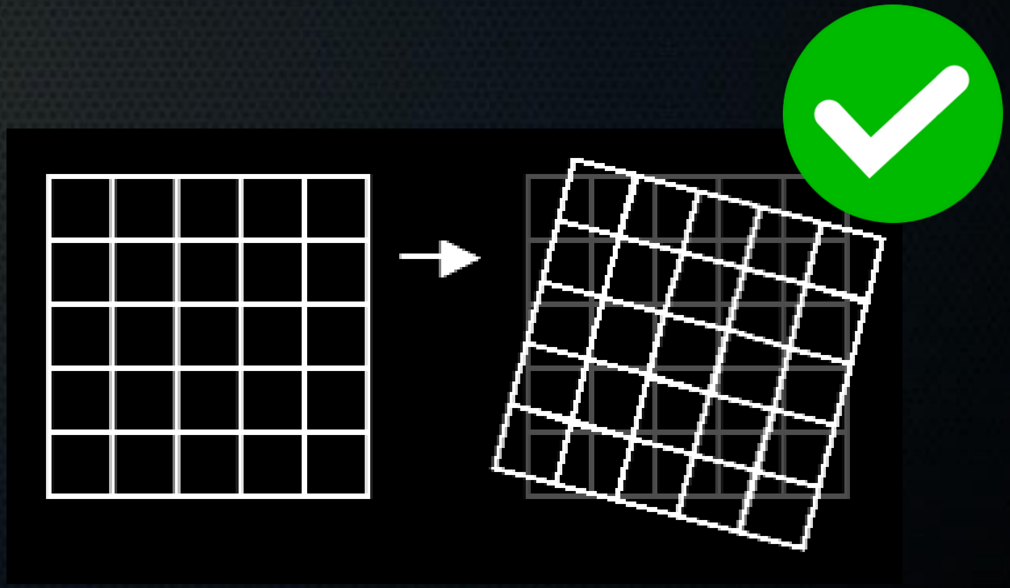
$$\downarrow$$

$$\begin{bmatrix} 2 & 3 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 8 \\ 1 \end{bmatrix}$$

# Language of ML: linear algebra – what is it?

- Calculations with vectors and matrices
- Should stay linear: just addition and scalar multiplication. No curves.





# Language of ML: linear algebra – what is it?

- Calculations with vectors and matrices

$$\begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

# Language of ML: linear algebra – what is it?

- Calculations with vectors and matrices

$$\begin{bmatrix} \underline{3} \\ \underline{4} \end{bmatrix}$$

# Language of ML: linear algebra – what is it?

- Calculations with vectors and matrices

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

# Language of ML: linear algebra – what is it?

- Calculations with vectors and matrices
- For example, scaling a vector

$$0.5 \cdot \underline{\begin{bmatrix} 3 \\ 4 \end{bmatrix}} \longrightarrow \underline{\begin{bmatrix} 1.5 \\ 2 \end{bmatrix}}$$

# Language of ML: linear algebra – what is it?

- Calculations with vectors and matrices
- For example, scaling a vector

$$-0.5 \cdot \underline{\begin{bmatrix} 3 \\ 4 \end{bmatrix}} \longrightarrow \underline{\begin{bmatrix} -1.5 \\ -2 \end{bmatrix}}$$

# Language of ML: linear algebra – what is it?

- Calculations with vectors and matrices
- Or matrix-vector multiplication


*

# Language of ML: linear algebra – what is it?

- Calculations with vectors and matrices
- Or matrix-vector multiplication

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 4 \end{bmatrix} \longrightarrow \begin{bmatrix} 11 \\ 25 \end{bmatrix}$$

# Language of ML: linear algebra – what is it?

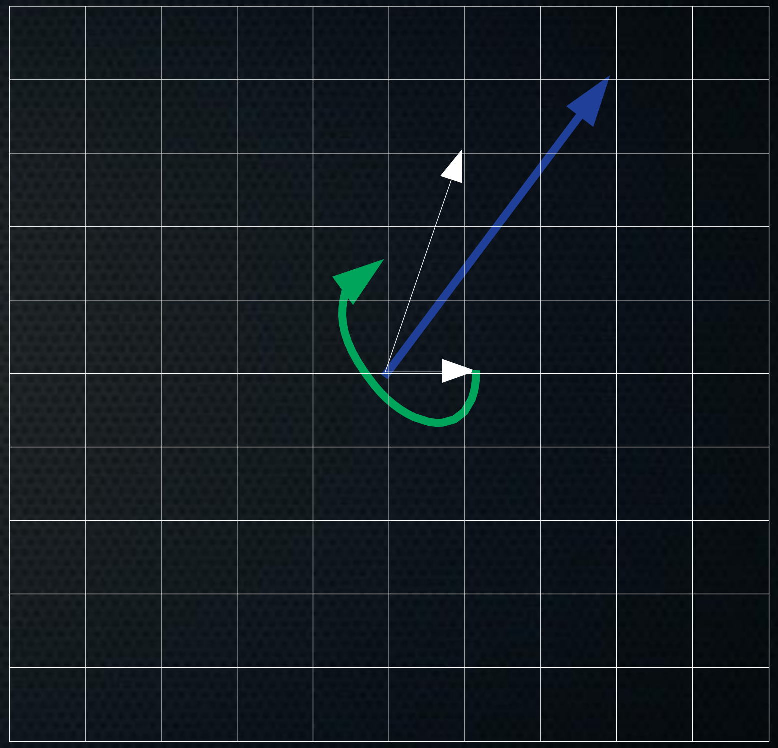
- Calculations with vectors and matrices
- Or matrix-vector multiplication
- Corresponds to a sort of squishing of space

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 4 \end{bmatrix} \longrightarrow \begin{bmatrix} 11 \\ 25 \end{bmatrix}$$

# Language of ML: linear algebra – what is it?

- Calculations with vectors and matrices
- Or matrix-vector multiplication

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 4 \end{bmatrix} \longrightarrow \begin{bmatrix} 11 \\ 25 \end{bmatrix}$$



# Language of ML: linear algebra – what is it?

- Calculations with vectors and matrices
- Or matrix-vector multiplication

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 4 \end{bmatrix} \longrightarrow \begin{bmatrix} 11 \\ 25 \end{bmatrix}$$

# Language of ML: linear algebra – what is it?

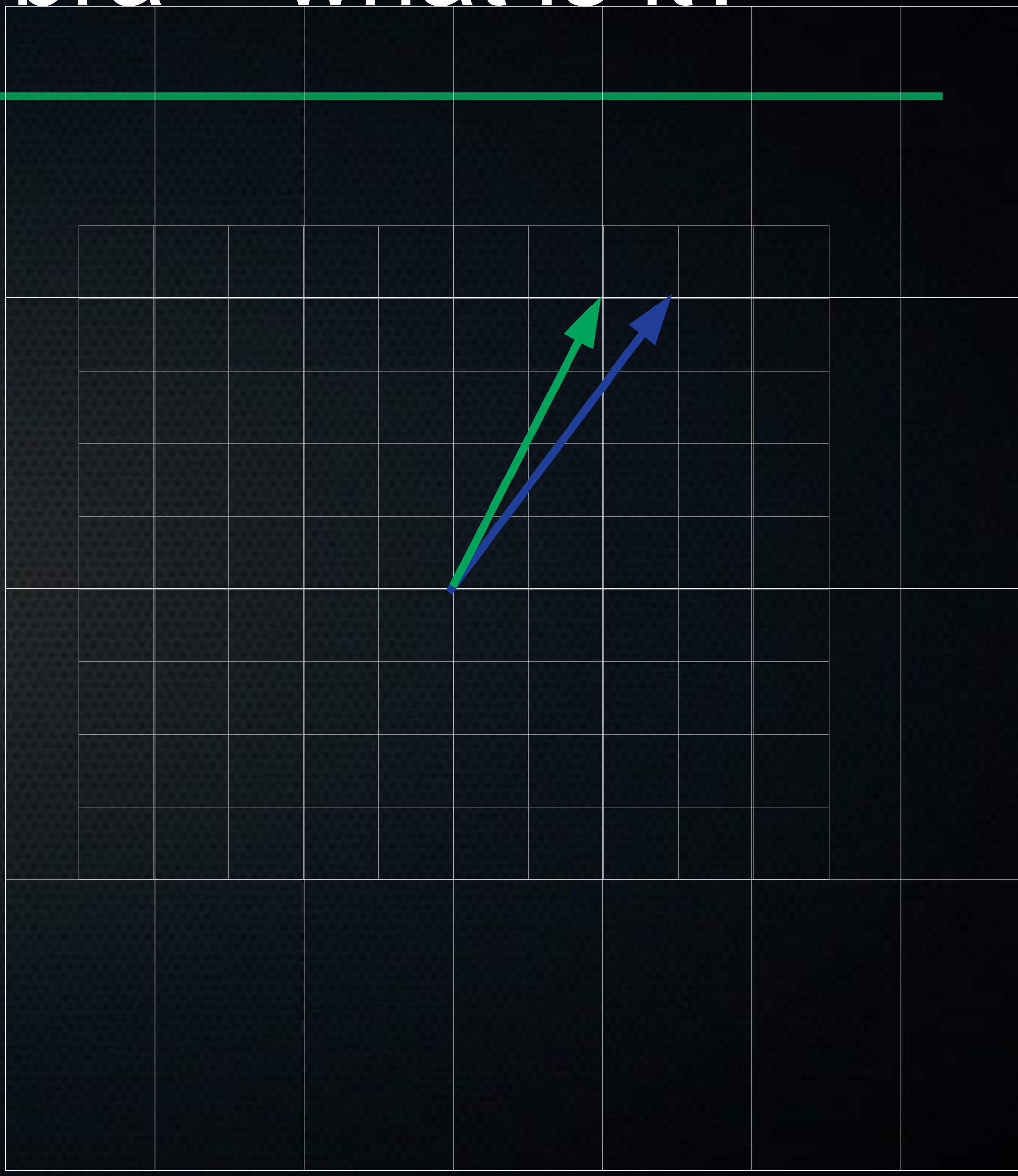
- Calculations with vectors and matrices
- Or matrix-vector multiplication

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 4 \end{bmatrix} \longrightarrow \begin{bmatrix} 11 \\ 25 \end{bmatrix}$$

# Language of ML: linear algebra – what is it?

- Calculations with vectors and matrices
- Or matrix-vector multiplication

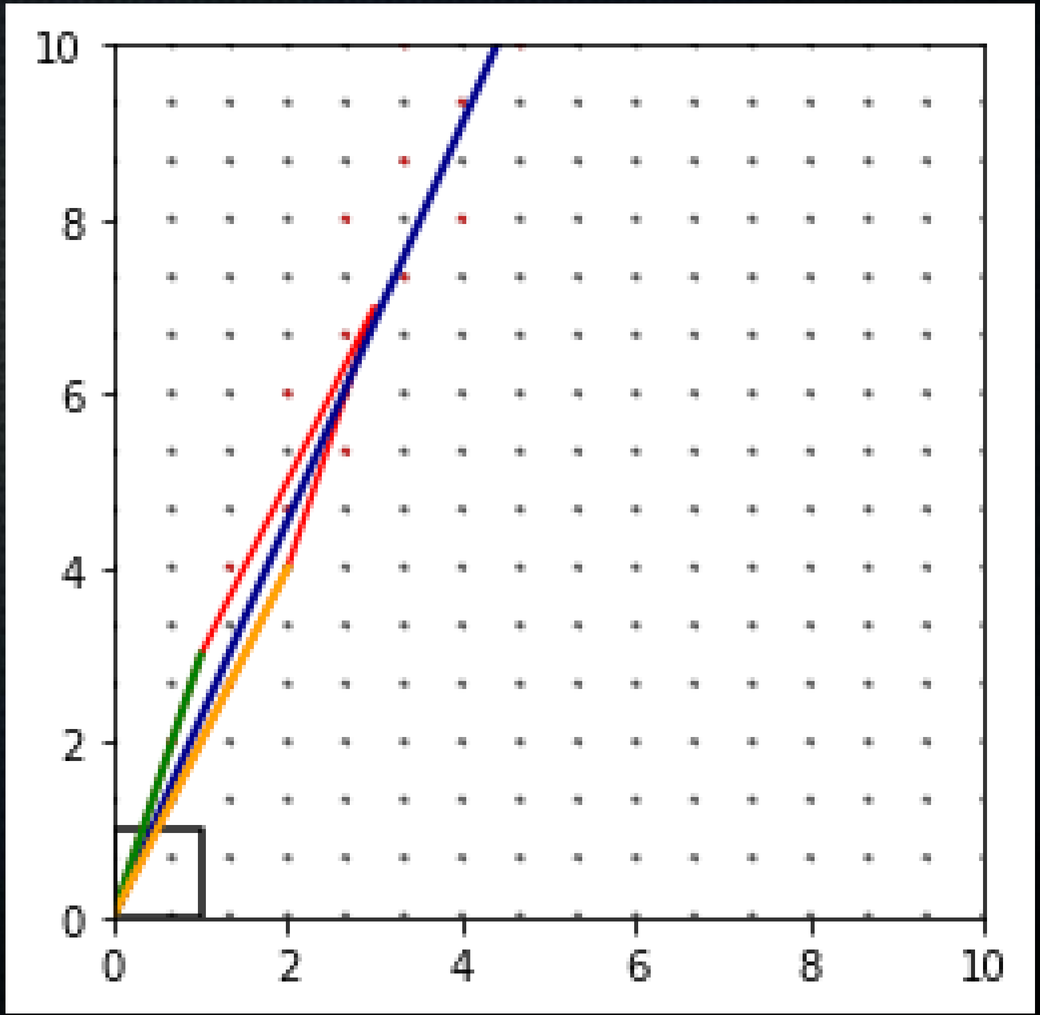
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 4 \end{bmatrix} \longrightarrow \begin{bmatrix} 11 \\ 25 \end{bmatrix}$$



# Language of ML: linear algebra – what is it?

- Calculations with vectors and matrices
- Or matrix-vector multiplication

$$\begin{bmatrix} \boxed{\begin{matrix}1\\3\end{matrix}} & \boxed{\begin{matrix}2\\4\end{matrix}} \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 4 \end{bmatrix} \longrightarrow \begin{bmatrix} 11 \\ 25 \end{bmatrix}$$



# Language of ML: linear algebra – why do we care?

- Machine learning algorithms are implemented and defined in linear algebra. Linear regression prediction:

$$\hat{Y} = X^T \hat{\beta}$$

# Language of ML: linear algebra – why do we care?

- Games require parallel calculations of many transformations of 3D vectors to rotate and show objects in 3D as you move around.
  - This has given us GPUs which are geared to do that immensely quickly and in parallel (GeForce GTX 690: ~5622 * 10^9/second)
  - And now TPUs or Tensor Processing Units which are geared more towards ML applications.
- Take advantage of that!

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}$$


# Language of ML: linear algebra – why do we care?

- Get rid of all the loops in your code *and* make it much faster. Win-win!

# Language of ML: linear algebra – vectors

- Scalar multiplication (*scales* the vector):

$$5 \cdot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 15 \\ 20 \end{bmatrix}$$

# Language of ML: linear algebra – vectors

- Scalar multiplication (*scales* the vector):

$$5 \cdot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 15 \\ 20 \end{bmatrix}$$

- Vector addition:

$$\begin{bmatrix} 3 \\ 4 \end{bmatrix} + \begin{bmatrix} 2 \\ 8 \end{bmatrix} = \begin{bmatrix} 5 \\ 12 \end{bmatrix}$$

# Language of ML: linear algebra – vectors

- Scalar multiplication (*scales* the vector):

$$5 \cdot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 15 \\ 20 \end{bmatrix}$$

- Vector addition:

$$\begin{bmatrix} 3 \\ 4 \end{bmatrix} + \begin{bmatrix} 2 \\ 8 \end{bmatrix} = \begin{bmatrix} 5 \\ 12 \end{bmatrix}$$

- Vector transpose (from column vector to row vector):

$$\begin{bmatrix} 3 \\ 4 \end{bmatrix}^T = \begin{bmatrix} 3 & 4 \end{bmatrix}$$

# Language of ML: linear algebra – matrices

- Scalar multiplication (*scales* the matrix):

$$5 \cdot \begin{bmatrix} 2 & 3 \\ 8 & 9 \end{bmatrix} = \begin{bmatrix} 10 & 15 \\ 40 & 45 \end{bmatrix}$$

- Matrix addition:

$$\begin{bmatrix} 2 & 3 \\ 8 & 9 \end{bmatrix} + \begin{bmatrix} 10 & 15 \\ 40 & 45 \end{bmatrix} = \begin{bmatrix} 12 & 18 \\ 48 & 54 \end{bmatrix}$$

# Language of ML: linear algebra – matrices

- Scalar multiplication (*scales* the matrix):

$$5 \cdot \begin{bmatrix} 2 & 3 \\ 8 & 9 \end{bmatrix} = \begin{bmatrix} 10 & 15 \\ 40 & 45 \end{bmatrix}$$

- Matrix addition:

$$\begin{bmatrix} 2 & 3 \\ 8 & 9 \end{bmatrix} + \begin{bmatrix} 10 & 15 \\ 40 & 45 \end{bmatrix} = \begin{bmatrix} 12 & 18 \\ 48 & 54 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 3 \\ 8 & 9 \end{bmatrix} + \begin{bmatrix} 10 & 15 & 1 \\ 40 & 45 & 9 \end{bmatrix} = ERROR$$

# Language of ML: linear algebra – matrices

- Scalar multiplication (*scales* the matrix):

$$5 \cdot \begin{bmatrix} 2 & 3 \\ 8 & 9 \end{bmatrix} = \begin{bmatrix} 10 & 15 \\ 40 & 45 \end{bmatrix}$$

- Matrix addition:

$$\begin{bmatrix} 2 & 3 \\ 8 & 9 \end{bmatrix} + \begin{bmatrix} 10 & 15 \\ 40 & 45 \end{bmatrix} = \begin{bmatrix} 12 & 18 \\ 48 & 54 \end{bmatrix}$$

- Matrix transpose:

$$\begin{bmatrix} 2 & 3 \\ 8 & 9 \end{bmatrix}^T = \begin{bmatrix} 2 & 8 \\ 3 & 9 \end{bmatrix}$$

# Language of ML: linear algebra – matrices

- Scalar multiplication (*scales* the matrix):

$$5 \cdot \begin{bmatrix} 2 & 3 \\ 8 & 9 \end{bmatrix} = \begin{bmatrix} 10 & 15 \\ 40 & 45 \end{bmatrix}$$

- Matrix addition:

$$\begin{bmatrix} 2 & 3 \\ 8 & 9 \end{bmatrix} + \begin{bmatrix} 10 & 15 \\ 40 & 45 \end{bmatrix} = \begin{bmatrix} 12 & 18 \\ 48 & 54 \end{bmatrix}$$

- Matrix transpose:

$$\begin{bmatrix} 2 & 3 \\ 8 & 9 \end{bmatrix}^T = \begin{bmatrix} 2 & 8 \\ 3 & 9 \end{bmatrix}$$

Note: vector special case of matrix where one dimension is 1.

# Language of ML: linear algebra – matrices

- Matrix-vector multiplication:

$$\begin{bmatrix} 2 & 3 \\ 8 & 9 \end{bmatrix} \cdot \begin{bmatrix} 10 \\ 8 \end{bmatrix} = \begin{bmatrix} 2 \cdot 10 + 3 \cdot 8 \\ 8 \cdot 10 + 9 \cdot 8 \end{bmatrix} = \begin{bmatrix} 44 \\ 152 \end{bmatrix}$$

# Language of ML: linear algebra – matrices

- Matrix-vector multiplication:

$$\boxed{\begin{bmatrix} 2 & 3 \\ 8 & 9 \end{bmatrix}} \cdot \boxed{\begin{bmatrix} 10 \\ 8 \end{bmatrix}} = \begin{bmatrix} 2 \cdot 10 + 3 \cdot 8 \\ 8 \cdot 10 + 9 \cdot 8 \end{bmatrix} = \begin{bmatrix} 44 \\ 152 \end{bmatrix}$$

-Sum of each element in the *row* of the matrix * each element in the *column* of the vector
-2 by 2 matrix times 2 by 1 vector becomes 2 by 1 vector.

# Language of ML: linear algebra – matrices

- Matrix-vector multiplication:

$$\begin{bmatrix} 2 & 3 \\ 8 & 9 \end{bmatrix} \cdot \begin{bmatrix} 10 \\ 8 \end{bmatrix} = \begin{bmatrix} 2 \cdot 10 + 3 \cdot 8 \\ 8 \cdot 10 + 9 \cdot 8 \end{bmatrix} = \begin{bmatrix} 44 \\ 152 \end{bmatrix}$$

-Sum of each element in the *row* of the matrix * each element in the *column* of the vector

-2 by $\underbrace{2 \text{ matrix times } 2}_{}$ by 1 vector becomes 2 by 1 vector.

\# of *columns* in A matches # of *rows* in B

# Language of ML: linear algebra – matrices

- Matrix-vector multiplication:

$$\begin{bmatrix}10\\8\end{bmatrix}\cdot\begin{bmatrix}2 & 3\\8 & 9\end{bmatrix} =$$


-Sum of each element in the *row* of the matrix * each element in the *column* of the vector
-2 by 1 vector times 2 by 2 matrix is undefined

# of *columns* in A **does not match**
# of *rows* in B

# Language of ML: linear algebra – matrices

- Matrix-vector multiplication:

$$\begin{bmatrix} 2 & 3 \\ 8 & 9 \end{bmatrix} \cdot \begin{bmatrix} 10 \\ 8 \end{bmatrix} = \begin{bmatrix} 2 \cdot 10 + 3 \cdot 8 \\ 8 \cdot 10 + 9 \cdot 8 \end{bmatrix} = \begin{bmatrix} 44 \\ 152 \end{bmatrix}$$

-Sum of each element in the *row* of the matrix * each element in the *column* of the vector

-2 by $\underbrace{2 \text{ matrix times } 2}_{\text{\# of columns in A matches \# of rows in B}}$ by 1 vector becomes 2 by 1 vector.

# Language of ML: linear algebra – matrices

- Matrix-vector multiplication:

$$\begin{bmatrix} \boxed{2 \quad 3} \\ 8 \quad 9 \end{bmatrix} \cdot \boxed{\begin{bmatrix} 10 \\ 8 \end{bmatrix}} = \begin{bmatrix} 2\cdot 10 + 3\cdot 8 \\ 8\cdot 10 + 9\cdot 8 \end{bmatrix} = \begin{bmatrix} 44 \\ 152 \end{bmatrix}$$

-Sum of each element in the *row* of the matrix * each element in the *column* of the vector

-2 by 2 matrix times 2 by 1 vector becomes 2 by 1 vector.

# of *rows* in matrix and number of *columns* in vector defines shape new vector

# Language of ML: linear algebra – matrices

- Matrix-vector multiplication:

$$\begin{bmatrix} 2 & 3 \\ 8 & 9 \\ 0 & 4 \end{bmatrix} \cdot \begin{bmatrix} 10 \\ 8 \end{bmatrix} = \begin{bmatrix} 2 \cdot 10 + 3 \cdot 8 \\ 8 \cdot 10 + 9 \cdot 8 \\ 0 \cdot 10 + 4 \cdot 8 \end{bmatrix} = \begin{bmatrix} 44 \\ 152 \\ 32 \end{bmatrix}$$

-Sum of each element in the *row* of the matrix * each element in the *column* of the vector

-3 by 2 matrix times 2 by 1 vector becomes 3 by 1 vector.

# of *rows* in matrix and number of *columns* in vector defines shape new vector

# Language of ML: linear algebra – matrices

- Matrix-matrix multiplication:
  - Matrix really just concatenated vector, so similar process:

$$\begin{bmatrix} \boxed{2 \quad 3} \\ 8 \quad 9 \end{bmatrix} \cdot \begin{bmatrix} \boxed{\begin{matrix} 10 \\ 8 \end{matrix}} & \begin{matrix} 2 \\ 15 \end{matrix} \end{bmatrix} = \begin{bmatrix} \boxed{2 \cdot 10 + 3 \cdot 8} \\ 8 \cdot 10 + 9 \cdot 8 \end{bmatrix} \begin{bmatrix} 2 \cdot 2 + 3 \cdot 15 \\ 8 \cdot 2 + 9 \cdot 15 \end{bmatrix} = \begin{bmatrix} \boxed{44} & ? \\ ? & ? \end{bmatrix}$$

# Language of ML: linear algebra – matrices

- Matrix-matrix multiplication:
  - Matrix really just concatenated vector, so similar process:

$$\begin{bmatrix}2 & 3\\8 & 9\end{bmatrix}\cdot\begin{bmatrix}10 & 2\\8 & 15\end{bmatrix}=\begin{bmatrix}2\cdot10+3\cdot8 & \boxed{2\cdot2+3\cdot15}\\8\cdot10+9\cdot8 & 8\cdot2+9\cdot15\end{bmatrix}=\begin{bmatrix}44 & \boxed{49}\\? & ?\end{bmatrix}$$

# Language of ML: linear algebra – matrices

- Matrix-matrix multiplication:
  - Matrix really just concatenated vector, so similar process:

$$\begin{bmatrix} 2 & 3 \\ \boxed{8 \quad 9} \end{bmatrix} \cdot \begin{bmatrix} \boxed{\begin{matrix}10 \\ 8\end{matrix}} & \begin{matrix}2 \\ 15\end{matrix} \end{bmatrix} = \begin{bmatrix} 2\cdot 10+3\cdot 8 & 2\cdot 2+3\cdot 15 \\ \boxed{8\cdot 10+9\cdot 8} & 8\cdot 2+9\cdot 15 \end{bmatrix} = \begin{bmatrix} 44 & 49 \\ \boxed{152} & ? \end{bmatrix}$$

# Language of ML: linear algebra – matrices

- Matrix-matrix multiplication:
  - Matrix really just concatenated vector, so similar process:

$$\begin{bmatrix} 2 & 3 \\ 8 & 9 \end{bmatrix} \cdot \begin{bmatrix} 10 & 2 \\ 8 & 15 \end{bmatrix} = \begin{bmatrix} 2\cdot10+3\cdot8 & 2\cdot2+3\cdot15 \\ 8\cdot10+9\cdot8 & \boxed{8\cdot2+9\cdot15} \end{bmatrix} = \begin{bmatrix} 44 & 49 \\ 152 & \boxed{151} \end{bmatrix}$$

# Language of ML: linear algebra – matrices

- Matrix-matrix multiplication:
  - Matrix really just concatenated vector, so similar process:

$$\begin{bmatrix} 2 & 3 \\ \boxed{8 \quad 9} \end{bmatrix} \cdot \begin{bmatrix} 10 & \boxed{\begin{matrix} 2 \\ 15 \end{matrix}} \\ 8 & \end{bmatrix} = \begin{bmatrix} 2 \cdot 10 + 3 \cdot 8 \\ 8 \cdot 10 + 9 \cdot 8 \end{bmatrix} \begin{bmatrix} 2 \cdot 2 + 3 \cdot 15 \\ \boxed{8 \cdot 2 + 9 \cdot 15} \end{bmatrix} = \begin{bmatrix} 44 & 49 \\ 152 & \boxed{151} \end{bmatrix}$$

-Sum of each element in the *row* of the matrix A * each element in the *column* of matrix B.
-2 by 2 matrix times 2 by 2 matrix becomes 2 by 2 matrix.

# of *columns* in A matches # of *rows* in B

# Language of ML: linear algebra – matrices

- Matrix-matrix multiplication is *non-commutative*: order matters!

$$\begin{bmatrix} 2 & 3 \\ 8 & 9 \end{bmatrix} \cdot \begin{bmatrix} 10 & 2 \\ 8 & 15 \end{bmatrix} = \begin{bmatrix} 44 & 49 \\ 152 & 151 \end{bmatrix}$$

$$2 \cdot 3 = 6$$

$$\begin{bmatrix} 10 & 2 \\ 8 & 15 \end{bmatrix} \cdot \begin{bmatrix} 2 & 3 \\ 8 & 9 \end{bmatrix} = \begin{bmatrix} 36 & 48 \\ 136 & 159 \end{bmatrix}$$

$$3 \cdot 2 = 6$$

# Language of ML: linear algebra -application

- That‘s a lot of *mathiness.* How is this useful for linear regression?

# Language of ML: linear algebra -application

- That‘s a lot of *mathiness.* How is this useful for linear regression?

$$h_\theta(x) = \theta_0 + \theta_1 \cdot Gene\,1 + \theta_2 \cdot Gene\,2 + \theta_n \cdot Gene\,n$$

$$\begin{array}{cc} & \begin{array}{ccc} \text{Gene 1} & \text{Gene 2} & \text{Gene n} \end{array} \\ \begin{array}{l} \text{Sample 1} \\ \text{Sample 2} \\ \text{Sample ...} \\ \text{Sample m} \end{array} & \begin{bmatrix} 2 & 3 & -2 \\ 8 & 9 & 1 \\ 0 & 4 & 5 \\ 5 & -2 & 2 \end{bmatrix} \end{array}$$

# Language of ML: linear algebra -application

- That's a lot of *mathiness.* How is this useful for linear regression?

$$h_\theta(x) = \theta_0 + \theta_1 \cdot Gene\,1 + \theta_2 \cdot Gene\,2 + \theta_n \cdot Gene\,n$$

$$\begin{array}{l c} & \begin{array}{ccc} \text{Gene 1} & \text{Gene 2} & \text{Gene n} \end{array} \\ \begin{array}{l} \text{Sample 1} \\ \text{Sample 2} \\ \text{Sample ...} \\ \text{Sample m} \end{array} & \begin{bmatrix} 2 & 3 & -2 \\ 8 & 9 & 1 \\ 0 & 4 & 5 \\ 5 & -2 & 2 \end{bmatrix} \end{array} \quad \begin{array}{c} \theta_0 \\ \theta_1 \\ \dots \\ \theta_n \end{array} \begin{bmatrix} 3 \\ -0.5 \\ 5 \\ -1 \end{bmatrix}$$

# Language of ML: linear algebra -application

- That‘s a lot of *mathiness.* How is this useful for linear regression?

$$h_\theta(x) = \theta_0 + \theta_1 \cdot Gene\,1 + \theta_2 \cdot Gene\,2 + \theta_n \cdot Gene\,n$$

$$\begin{array}{l} \\ \text{Sample 1} \\ \text{Sample 2} \\ \text{Sample ...} \\ \text{Sample m} \end{array} \begin{array}{c} \begin{array}{ccc} \text{Gene 1} & \text{Gene 2} & \text{Gene n} \end{array} \\ \begin{bmatrix} 2 & 3 & -2 \\ 8 & 9 & 1 \\ 0 & 4 & 5 \\ 5 & -2 & 2 \end{bmatrix} \end{array} \quad \begin{array}{c} \theta_0 \\ \theta_1 \\ \dots \\ \theta_n \end{array} \begin{bmatrix} 3 \\ -0.5 \\ 5 \\ -1 \end{bmatrix}$$

To get vector of predictions from vector of thetas and matrix of data, want to multiply them

# Language of ML: linear algebra -application

- That‘s a lot of *mathiness.* How is this useful for linear regression?

$$h_\theta(x)=\theta_0+\theta_1 \cdot Gene\,1+\theta_2 \cdot Gene\,2+\theta_n \cdot Gene\,n$$

$$\begin{array}{l} \\ \text{Sample 1} \\ \text{Sample 2} \\ \text{Sample ...} \\ \text{Sample m} \end{array}
\begin{array}{c} \begin{array}{cccc} \text{Intercept} & \text{Gene 1} & \text{Gene 2} & \text{Gene n} \end{array} \\ \begin{bmatrix} 1 & 2 & 3 & -2 \\ 1 & 8 & 9 & 1 \\ 1 & 0 & 4 & 5 \\ 1 & 5 & -2 & 2 \end{bmatrix} \end{array}
\quad
\begin{array}{cc} \theta_0 & 3 \\ \theta_1 & -0.5 \\ \dots & 5 \\ \theta_n & -1 \end{array}$$

To get vector of predictions from vector of thetas and matrix of data, want to multiply them

Dimensions don‘t match, easy fix: new feature

# Language of ML: linear algebra -application

- That's a lot of *mathiness.* How is this useful for linear regression?

$$h_\theta(x)=\theta_0+\theta_1\cdot Gene\,1+\theta_2\cdot Gene\,2+\theta_n\cdot Gene\,n$$

$$\begin{array}{c} \theta_0 \\ \theta_1 \\ \dots \\ \theta_n \end{array} \begin{bmatrix} 3 \\ -0.5 \\ 5 \\ -1 \end{bmatrix} \qquad \begin{array}{c|c} & \begin{array}{cccc} \text{Intercept} & \text{Gene 1} & \text{Gene 2} & \text{Gene n} \end{array} \\ \begin{array}{c} \text{Sample 1} \\ \text{Sample 2} \\ \text{Sample ...} \\ \text{Sample m} \end{array} & \begin{bmatrix} 1 & 2 & 3 & -2 \\ 1 & 8 & 9 & 1 \\ 1 & 0 & 4 & 5 \\ 1 & 5 & -2 & 2 \end{bmatrix} \end{array} \longrightarrow ?$$

# Language of ML: linear algebra -application

- That's a lot of *mathiness.* How is this useful for linear regression?

$$h_\theta(x)=\theta_0+\theta_1\cdot Gene\,1+\theta_2\cdot Gene\,2+\theta_n\cdot Gene\,n$$

$$\begin{array}{c} \theta_0 \\ \theta_1 \\ \dots \\ \theta_n \end{array} \begin{bmatrix} \boxed{3} \\ -0.5 \\ 5 \\ -1 \end{bmatrix} \qquad \begin{array}{l} \\ \text{Sample 1} \\ \text{Sample 2} \\ \text{Sample ...} \\ \text{Sample m} \end{array} \begin{array}{c} \begin{array}{cccc} \text{Intercept} & \text{Gene 1} & \text{Gene 2} & \text{Gene n} \end{array} \\ \begin{bmatrix} 1 & 2 & 3 & -2 \\ 1 & 8 & 9 & 1 \\ 1 & 0 & 4 & 5 \\ 1 & 5 & -2 & 2 \end{bmatrix} \end{array} \longrightarrow$$

Wrong!
# of columns A doesn't match
# rows B!

# Language of ML: linear algebra -application

- That's a lot of *mathiness.* How is this useful for linear regression?

$$h_\theta(x)=\theta_0+\theta_1\cdot Gene\,1+\theta_2\cdot Gene\,2+\theta_n\cdot Gene\,n$$

$$
\begin{array}{c}
 \\ \text{Sample 1} \\ \text{Sample 2} \\ \text{Sample ...} \\ \text{Sample m}
\end{array}
\begin{array}{c}
\begin{array}{cccc} \text{Intercept} & \text{Gene 1} & \text{Gene 2} & \text{Gene n} \end{array} \\
\begin{bmatrix} 1 & 2 & 3 & -2 \\ 1 & 8 & 9 & 1 \\ 1 & 0 & 4 & 5 \\ 1 & 5 & -2 & 2 \end{bmatrix}
\end{array}
\quad
\begin{array}{c} \theta_0 \\ \theta_1 \\ \dots \\ \theta_n \end{array}
\begin{bmatrix} 3 \\ -0.5 \\ 5 \\ -1 \end{bmatrix}
\quad
\begin{bmatrix} 1\cdot 3+2\cdot -0.5+3\cdot 5+-2\cdot -1=19 \\ 43 \\ 18 \\ -11.5 \end{bmatrix}
$$

# Language of ML: linear algebra -application

- That‘s a lot of *mathiness.* How is this useful for linear regression?

$$h_\theta(x)=\theta_0+\theta_1\cdot Gene\,1+\theta_2\cdot Gene\,2+\theta_n\cdot Gene\,n$$

$$\begin{array}{ll} & \begin{array}{cccc}\text{Intercept} & \text{Gene 1} & \text{Gene 2} & \text{Gene n}\end{array} \\ \begin{array}{l}\text{Sample 1}\\ \text{Sample 2}\\ \text{Sample ...}\\ \text{Sample m}\end{array} & \begin{bmatrix}1 & 2 & 3 & -2\\ 1 & 8 & 9 & 1\\ 1 & 0 & 4 & 5\\ 1 & 5 & -2 & 2\end{bmatrix}\end{array} \quad \begin{array}{c}\theta_0\\ \theta_1\\ \dots\\ \theta_n\end{array}\begin{bmatrix}3\\ -0.5\\ 5\\ -1\end{bmatrix} \quad \begin{bmatrix}1\cdot 3+2\cdot -0.5+3\cdot 5+-2\cdot -1=19\\ 43\\ 18\\ -11.5\end{bmatrix}$$

# Language of ML: linear algebra -application

- That's a lot of *mathiness.* How is this useful for linear regression?

$$h_\theta(x) = \theta_0 + \theta_1 \cdot Gene\,1 + \theta_2 \cdot Gene\,2 + \theta_n \cdot Gene\,n$$

$$\begin{array}{r c}
 & \begin{array}{cccc} \text{Intercept} & \text{Gene 1} & \text{Gene 2} & \text{Gene n} \end{array} \\
\begin{array}{r} \text{Sample 1} \\ \text{Sample 2} \\ \text{Sample ...} \\ \text{Sample m} \end{array} &
\begin{bmatrix} 1 & 2 & 3 & -2 \\ 1 & 8 & 9 & 1 \\ 1 & 0 & 4 & 5 \\ 1 & 5 & -2 & 2 \end{bmatrix}
\end{array}
\quad
\begin{array}{c} \theta_0 \\ \theta_1 \\ \dots \\ \theta_n \end{array}
\begin{bmatrix} 3 \\ -0.5 \\ 5 \\ -1 \end{bmatrix}
\quad
\begin{bmatrix} 1 \cdot 3 + 2 \cdot -0.5 + 3 \cdot 5 + -2 \cdot -1 = 19 \\ 43 \\ 18 \\ -11.5 \end{bmatrix}$$

# Language of ML: linear algebra -application

- That‘s a lot of *mathiness.* How is this useful for linear regression?

$$h_\theta(x) = \theta_0 + \theta_1 \cdot Gene\,1 + \theta_2 \cdot Gene\,2 + \theta_n \cdot Gene\,n$$

$$\begin{array}{cc} & \begin{matrix}\text{Intercept} & \text{Gene 1} & \text{Gene 2} & \text{Gene n}\end{matrix} \\ \begin{matrix}\text{Sample 1}\\ \text{Sample 2}\\ \text{Sample ...}\\ \text{Sample m}\end{matrix} & \begin{bmatrix}1 & 2 & 3 & -2\\ 1 & 8 & 9 & 1\\ 1 & 0 & 4 & 5\\ 1 & 5 & -2 & 2\end{bmatrix}\end{array} \quad \begin{array}{c}\theta_0\\ \theta_1\\ \dots\\ \theta_n\end{array}\begin{bmatrix}3\\ -0.5\\ 5\\ -1\end{bmatrix} \quad \begin{bmatrix}1\cdot 3 + 2\cdot -0.5 + 3\cdot 5 + -2\cdot -1 = 19\\ 43\\ 18\\ -11.5\end{bmatrix}$$

# Code comparison

```
#data setup
thetas = np.array([3, -0.5, 5, -1], np.newaxis)

featureDataFrame = pd.DataFrame({"Intercept" : [1,1,1,1],
                                 "Gene1" : [2,8,0,5],
                                 "Gene2" : [3,9,4,-2],
                                 "Gene3" : [-2, 1, 5, 2]})
featureDataFrame.index = (["Sample" + str(num) for num in range(1,5)] )

print(featureDataFrame)
print(thetas)
```

```
         Intercept  Gene1  Gene2  Gene3
Sample1          1      2      3     -2
Sample2          1      8      9      1
Sample3          1      0      4      5
Sample4          1      5     -2      2
[ 3.  -0.5  5.  -1. ]
```

```
#not vectorised
totalPredictions = []
for sample, sampleData in featureDataFrame.iterrows():
    thisPrediction = 0
    for index, feature in enumerate(sampleData):
      thisPrediction += feature * thetas[index]
    totalPredictions.append(thisPrediction)

print(totalPredictions)
```

```
[19.0, 43.0, 18.0, -11.5]
```

```
#vectorised
totalPredictionsLA = featureDataFrame @ thetas
print(totalPredictionsLA)
```

```
Sample1    19.0
Sample2    43.0
Sample3    18.0
Sample4   -11.5
dtype: float64
```

# Code comparison

```
#data setup
thetas = np.array([3, -0.5, 5, -1], np.newaxis)

featureDataFrame = pd.DataFrame({"Intercept" : [1,1,1,1],
                                 "Gene1" : [2,8,0,5],
                                 "Gene2" : [3,9,4,-2],
                                 "Gene3" : [-2, 1, 5, 2]})
featureDataFrame.index = (["Sample" + str(num) for num in range(1,5)] )

print(featureDataFrame)
print(thetas)
```

```
         Intercept  Gene1  Gene2  Gene3
Sample1          1      2      3     -2
Sample2          1      8      9      1
Sample3          1      0      4      5
Sample4          1      5     -2      2
[ 3.  -0.5  5.  -1. ]
```

```
#not vectorised
totalPredictions = []
for sample, sampleData in featureDataFrame.iterrows():
    thisPrediction = 0
    for index, feature in enumerate(sampleData):
      thisPrediction += feature * thetas[index]
    totalPredictions.append(thisPrediction)

print(totalPredictions)
```

```
[19.0, 43.0, 18.0, -11.5]
```

```
#vectorised
totalPredictionsLA = featureDataFrame @ thetas
print(totalPredictionsLA)
```

```
Sample1     19.0
Sample2     43.0
Sample3     18.0
Sample4    -11.5
dtype: float64
```

# Code comparison

```
#data setup
thetas = np.array([3, -0.5, 5, -1], np.newaxis)

featureDataFrame = pd.DataFrame({"Intercept" : [1,1,1,1],
                                 "Gene1" : [2,8,0,5],
                                 "Gene2" : [3,9,4,-2],
                                 "Gene3" : [-2, 1, 5, 2]})
featureDataFrame.index = (["Sample" + str(num) for num in range(1,5)] )

print(featureDataFrame)
print(thetas)
```

```
         Intercept  Gene1  Gene2  Gene3
Sample1          1      2      3     -2
Sample2          1      8      9      1
Sample3          1      0      4      5
Sample4          1      5     -2      2
[ 3.  -0.5  5.  -1. ]
```

```
#not vectorised
totalPredictions = []
for sample, sampleData in featureDataFrame.iterrows():
    thisPrediction = 0
    for index, feature in enumerate(sampleData):
      thisPrediction += feature * thetas[index]
    totalPredictions.append(thisPrediction)

print(totalPredictions)
```

```
[19.0, 43.0, 18.0, -11.5]
```

```
#vectorised
totalPredictionsLA = featureDataFrame @ thetas
print(totalPredictionsLA)
```

```
Sample1    19.0
Sample2    43.0
Sample3    18.0
Sample4   -11.5
dtype: float64
```

# Summary

- Linear algebra is the basis of ML: algorithms are defined in it and run quickly due to hardware optimised for matrix and vector operations
- Using linear algebra cuts down on code complexity
- You always add a „dummy“ feature that is 1 to multiply with $\theta_0$
- We covered how to multiply and add matrices and vectors, and showed that matrix multiplication is *non-commutative*: order matters!

# Summary

- In a very real way we are just using linear algebra as ✨fancy bookkeeping✨, and computer hardware is optimized for this as many operations can be expressed in terms of this bookkeeping.

# Numpy details

- During the computer lab, try to make your basic unit a column vector. That is, a 2D array that looks like this:

```
array([[0],
       [1]
       [5]])
```

- When you subset things, numpy automatically makes single lists 1D. To make a 1D array 2D again, use `array[:,np.newaxis]`
- For your thetas, too, don't use a list like so: [0.5, -0.3, 0.8], but make it 2D.
- To check whether something is the right form, you can check that `array.ndim == 2`, for instance.

# Gradient descent in linear algebra

- Goal gradient descent: take a small step in every parameter such that you get closer to the minimum of the cost. Return new theta's.

$$\theta_{0new} = \theta_0 - \frac{a}{m} \sum_{i=1}^{m} ((h_\theta(x^{(i)}) - y^{(i)}) \cdot 1)$$

$$\theta_{1new} = \theta_1 - \frac{a}{m} \sum_{i=1}^{m} ((h_\theta(x^{(i)}) - y^{(i)}) \cdot x_1^{(i)})$$

$$\theta_{2new} = \theta_2 - \frac{a}{m} \sum_{i=1}^{m} ((h_\theta(x^{(i)}) - y^{(i)}) \cdot x_2^{(i)})$$

# Gradient descent in linear algebra

**We have data, known values, and initial theta's:**

$$X = \begin{bmatrix} 1 & feat_1 val_1 & feat_2 val_1 \\ 1 & feat_1 val_2 & feat_2 val_2 \end{bmatrix}; y = \begin{bmatrix} 10.23 \\ -4 \end{bmatrix}; params = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$$

**Get predicted values:**

$$\begin{bmatrix} 1 & feat_1 val_1 & feat_2 val_1 \\ 1 & feat_1 val_2 & feat_2 val_2 \end{bmatrix} @ \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} 9.23 \\ -2.5 \end{bmatrix}$$

2 by 3 times 3 by 1 gives 2 by 1 (rows by columns)

**Get errors:**

$$errs = \begin{bmatrix} 9.23 \\ -2.5 \end{bmatrix} - y = \begin{bmatrix} 9.23 \\ -2.5 \end{bmatrix} - \begin{bmatrix} 10.23 \\ -4 \end{bmatrix} = \begin{bmatrix} -1 \\ 1.5 \end{bmatrix}$$

Where did the $^2$ go?

$$J = \frac{1}{2m} \sum (\text{pred} - y)^2$$

$$\frac{\partial J}{\partial \text{pred}} = \frac{1}{m} (\text{pred} - y)$$

# Gradient descent in linear algebra

$$errs = \begin{bmatrix} -1 \\ 1.5 \end{bmatrix}$$

$$\theta_{0new} = \theta_0 - \frac{a}{m} \sum_{i=1}^{m} \left( \boxed{\left( h_\theta(x^{(i)}) - y^{(i)} \right)} \cdot 1 \right)$$

$$\theta_{1new} = \theta_1 - \frac{a}{m} \sum_{i=1}^{m} \left( \boxed{\left( h_\theta(x^{(i)}) - y^{(i)} \right)} \cdot x_1^{(i)} \right)$$

$$\theta_{2new} = \theta_2 - \frac{a}{m} \sum_{i=1}^{m} \left( \boxed{\left( h_\theta(x^{(i)}) - y^{(i)} \right)} \cdot x_2^{(i)} \right)$$

# Gradient descent in linear algebra

**Calculate, for each feature, sum of each error times that feature:**

$$\begin{bmatrix} -1 \\ 1.5 \end{bmatrix}^T = \begin{bmatrix} -1 & 1.5 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 1.5 \end{bmatrix} @ \begin{bmatrix} 1 & feat_1val_1 & feat_2val_1 \\ 1 & feat_1val_2 & feat_2val_2 \end{bmatrix} =$$

$$\begin{bmatrix} -1 \cdot 1 + 1.5 \cdot 1 & -1 \cdot feat_1val_1 + 1.5 \cdot feat_1val_2 & -1 \cdot feat_2val_1 + 1.5 \cdot feat_2val_2 \end{bmatrix}$$

$$\boxed{errs = \begin{bmatrix} -1 \\ 1.5 \end{bmatrix}}$$

# Gradient descent in linear algebra

**Calculate, for each feature, sum of each error times that feature:**

$$\begin{bmatrix} -1 \\ 1.5 \end{bmatrix}^T = \begin{bmatrix} -1 & 1.5 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 1.5 \end{bmatrix} @ \begin{bmatrix} 1 & feat_1val_1 & feat_2val_1 \\ 1 & feat_1val_2 & feat_2val_2 \end{bmatrix} =$$

$$errs = \begin{bmatrix} -1 \\ 1.5 \end{bmatrix}$$

$$\begin{bmatrix} \boxed{-1 \cdot 1 + 1.5 \cdot 1} & \boxed{-1 \cdot feat_1val_1 + 1.5 \cdot feat_1val_2} & \boxed{-1 \cdot feat_2val_1 + 1.5 \cdot feat_2val_2} \end{bmatrix}$$

$$\theta_{0new} = \theta_0 - \frac{a}{m} \boxed{\sum_{i=1}^{m} ((h_\theta(x^{(i)}) - y^{(i)}) \cdot 1)}$$

$$\theta_{1new} = \theta_1 - \frac{a}{m} \boxed{\sum_{i=1}^{m} ((h_\theta(x^{(i)}) - y^{(i)}) \cdot x_1^{(i)})}$$

$$\theta_{2new} = \theta_2 - \frac{a}{m} \boxed{\sum_{i=1}^{m} ((h_\theta(x^{(i)}) - y^{(i)}) \cdot x_2^{(i)})}$$

# Gradient descent in linear algebra

Now all that we need to do is multiply with $\alpha$/m and subtract from our old theta's:

$$\alpha/m \cdot \begin{bmatrix} -1\cdot 1 + 1.5\cdot 1 & -1\cdot feat_1val_1 + 1.5\cdot feat_1val_2 & -1\cdot feat_2val_1 + 1.5\cdot feat_2val_2 \end{bmatrix}$$
$$= \begin{bmatrix} \frac{\alpha}{m}(-1\cdot 1 + 1.5\cdot 1) & \frac{\alpha}{m}(-1\cdot feat_1val_1 + 1.5\cdot feat_1val_2) & \frac{\alpha}{m}(-1\cdot feat_2val_1 + 1.5\cdot feat_2val_2) \end{bmatrix}$$

Transpose it:

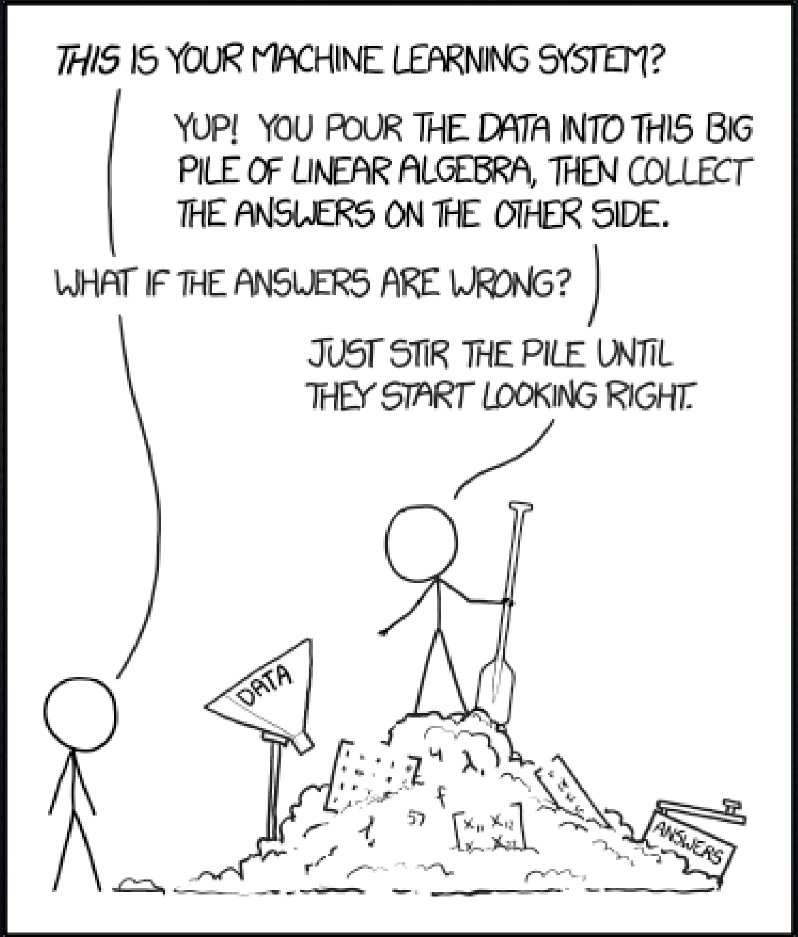
$$\begin{bmatrix} \frac{\alpha}{m}(-1\cdot 1 + 1.5\cdot 1) & \frac{\alpha}{m}(-1\cdot feat_1val_1 + 1.5\cdot feat_1val_2) & \frac{\alpha}{m}(-1\cdot feat_2val_1 + 1.5\cdot feat_2val_2) \end{bmatrix}^T = \begin{bmatrix} \frac{\alpha}{m}(-1\cdot 1 + 1.5\cdot 1) \\ \frac{\alpha}{m}(-1\cdot feat_1val_1 + 1.5\cdot feat_1val_2) \\ \frac{\alpha}{m}(-1\cdot feat_2val_1 + 1.5\cdot feat_2val_2) \end{bmatrix}$$

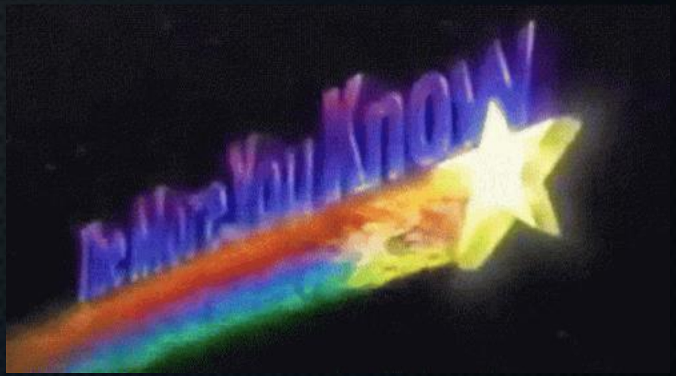
So finally:

$$\begin{bmatrix} \theta_{0old} \\ \theta_{1old} \\ \theta_{2old} \end{bmatrix} - \begin{bmatrix} \frac{\alpha}{m}(-1\cdot 1 + 1.5\cdot 1) \\ \boxed{\frac{\alpha}{m}(-1\cdot feat_1val_1 + 1.5\cdot feat_1val_2)} \\ \frac{\alpha}{m}(-1\cdot feat_2val_1 + 1.5\cdot feat_2val_2) \end{bmatrix} = \begin{bmatrix} \theta_{0new} \\ \theta_{1new} \\ \theta_{2new} \end{bmatrix}$$

$$\theta_{1new} = \theta_1 - \boxed{\frac{a}{m}\sum_{i=1}^{m}((h_\theta(x^{(i)}) - y^{(i)})\cdot x_1^{(i)})}$$

# Relevant XKCD



# Computer Lab

- Practicing vector and matrix operations with numpy
- Changing cost function, hypothesis function, and gradient descent to work with matrices and vectors
- Working with a real biological dataset

HAVE FUN!

(and/or suffer if it’s too hard… but then tell me and I shall try to ease your suffering)