CS 453X: Class 4

Jacob Whitehill

Linear regression

Linear regression: matrix notation

- Let's define a matrix **X** to contain all the training images:

$$\mathbf{X} = \left[\begin{array}{ccc} | & & | \\ \mathbf{x}^{(1)} & \dots & \mathbf{x}^{(n)} \\ | & & | \end{array} \right]$$

  - In statistics, **X** is called the **design matrix.**

- Let's define vector **y** to contain all the training labels:

$$\mathbf{y} = \left[\begin{array}{c} y^{(1)} \\ \vdots \\ y^{(n)} \end{array} \right]$$

Linear regression: matrix notation

- Using summation notation, we derived:

$$\mathbf{w} = \left(\sum_{i=1}^{n} \mathbf{x}^{(i)} \mathbf{x}^{(i)\top} \right)^{-1} \left(\sum_{i=1}^{n} \mathbf{x}^{(i)} y^{(i)} \right)$$

- Using matrix notation, we can write the solution as:

$$\mathbf{w} = \left(\mathbf{X}\mathbf{X}^{\top} \right)^{-1} \mathbf{X}\mathbf{y}$$

Linear regression: matrix notation

- To compute **w**, do *not* use np.linalg.inv.
- Instead, use np.linalg.solve, which avoids explicitly computing the matrix inverse.

Linear regression: matrix notation

- Once we've "trained" the weights **w**, we can estimate the *y*-value (label) for any **x**.
- We can compute the { $\hat{y}^{(i)}$ } for a set of images { $\mathbf{x}^{(i)}$ } in one-fell-swoop using matrix operations.
- Let's define our design matrix **X** as before:

$$\mathbf{X} = \left[\begin{array}{ccc} | & & | \\ \mathbf{x}^{(1)} & \dots & \mathbf{x}^{(n)} \\ | & & | \end{array} \right]$$

- Then our estimates of the labels is given by:

$$\hat{\mathbf{y}} = \mathbf{X}^\top \mathbf{w}$$

Linear regression: matrix notation

- Suppose we have *n* images, each with just 2 pixels.

$$\hat{\mathbf{y}} = \mathbf{X}^\top \mathbf{w}$$

Linear regression: matrix notation

- Suppose we have *n* images, each with just 2 pixels.

$$
\begin{aligned}
\hat{\mathbf{y}} &= \mathbf{X}^\top \mathbf{w} \\
&= \left[\begin{array}{ccc} \mathbf{x}_1^{(1)} & \dots & \mathbf{x}_1^{(n)} \\ \mathbf{x}_2^{(1)} & \dots & \mathbf{x}_2^{(n)} \end{array} \right]^\top \left[\begin{array}{c} w_1 \\ w_2 \end{array} \right]
\end{aligned}
$$

This is the index of the *image*.

Linear regression: matrix notation

- Suppose we have *n* images, each with just 2 pixels.

$$\begin{aligned}\hat{\mathbf{y}} &= \mathbf{X}^\top \mathbf{w} \\ &= \begin{bmatrix} \mathbf{x}_1^{(1)} & \dots & \mathbf{x}_1^{(n)} \\ \mathbf{x}_2^{(1)} & \dots & \mathbf{x}_2^{(n)} \end{bmatrix}^\top \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}\end{aligned}$$

This is the index of the *pixel*.

Linear regression: matrix notation

- Suppose we have *n* images, each with just 2 pixels.

$$
\begin{aligned}
\hat{\mathbf{y}} &= \mathbf{X}^\top \mathbf{w} \\
&= \begin{bmatrix} \mathbf{x}_1^{(1)} & \dots & \mathbf{x}_1^{(n)} \\ \mathbf{x}_2^{(1)} & \dots & \mathbf{x}_2^{(n)} \end{bmatrix}^\top \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{x}_1^{(1)} & \mathbf{x}_2^{(1)} \\ \vdots & \vdots \\ \mathbf{x}_1^{(n)} & \mathbf{x}_2^{(n)} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}
\end{aligned}
$$

Linear regression: matrix notation

- Suppose we have *n* images, each with just 2 pixels.

$$
\begin{aligned}
\hat{\mathbf{y}} &= \mathbf{X}^\top \mathbf{w} \\
&= \left[\begin{array}{ccc} \mathbf{x}_1^{(1)} & \dots & \mathbf{x}_1^{(n)} \\ \mathbf{x}_2^{(1)} & \dots & \mathbf{x}_2^{(n)} \end{array} \right]^\top \left[\begin{array}{c} w_1 \\ w_2 \end{array} \right] \\
&= \left[\begin{array}{cc} \mathbf{x}_1^{(1)} & \mathbf{x}_2^{(1)} \\ \vdots & \vdots \\ \mathbf{x}_1^{(n)} & \mathbf{x}_2^{(n)} \end{array} \right] \left[\begin{array}{c} w_1 \\ w_2 \end{array} \right] \\
&= \left[\begin{array}{c} \mathbf{x}_1^{(1)} w_1 + \mathbf{x}_2^{(1)} w_2 \\ \vdots \\ \mathbf{x}_1^{(n)} w_1 + \mathbf{x}_2^{(n)} w_2 \end{array} \right]
\end{aligned}
$$

Linear regression: matrix notation

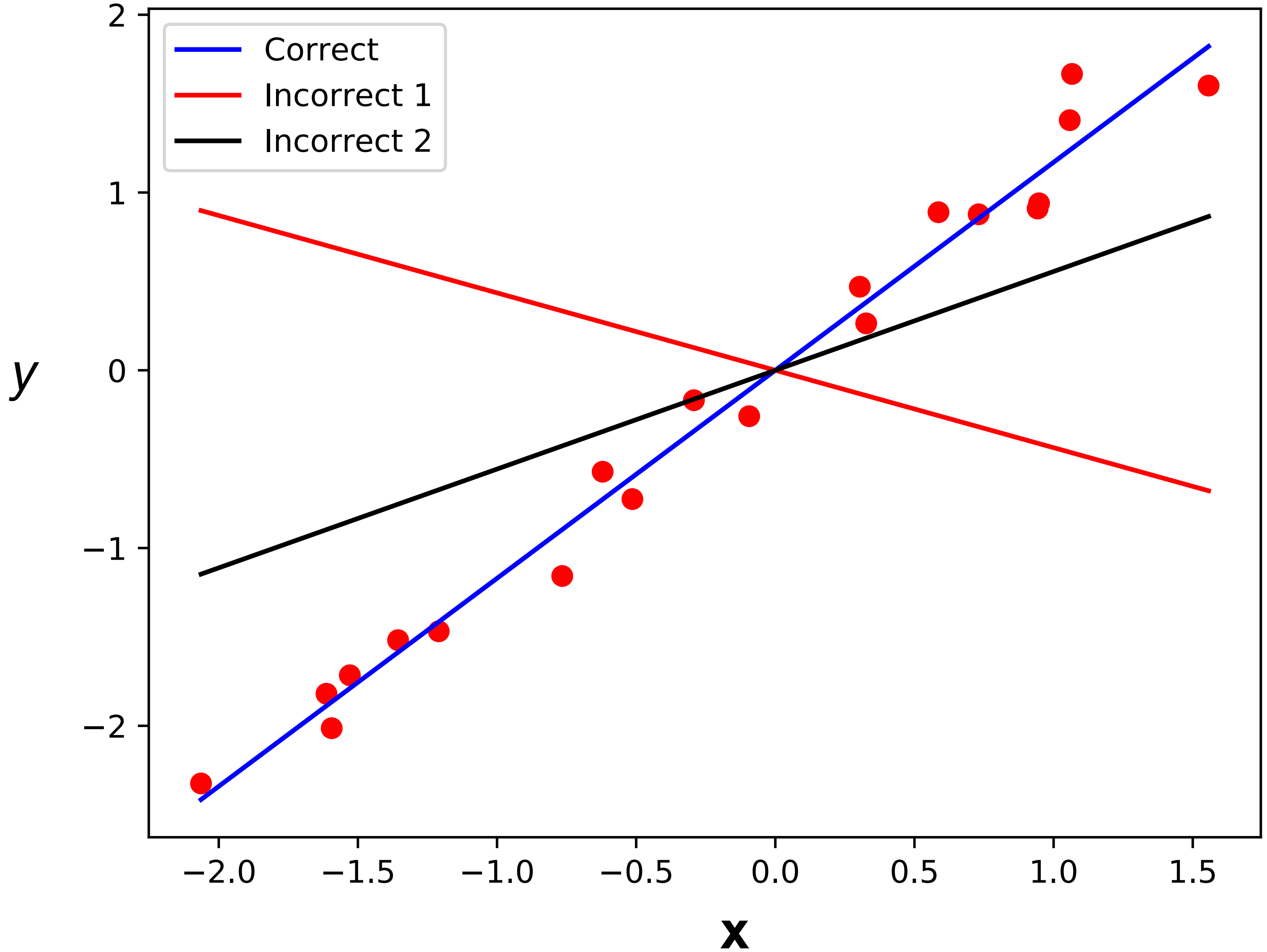
- Suppose we have *n* images, each with just 2 pixels.

$$
\begin{aligned}
\hat{\mathbf{y}} &= \mathbf{X}^\top \mathbf{w} \\
&= \begin{bmatrix} \mathbf{x}_1^{(1)} & \dots & \mathbf{x}_1^{(n)} \\ \mathbf{x}_2^{(1)} & \dots & \mathbf{x}_2^{(n)} \end{bmatrix}^\top \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{x}_1^{(1)} & \mathbf{x}_2^{(1)} \\ \vdots & \vdots \\ \mathbf{x}_1^{(n)} & \mathbf{x}_2^{(n)} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \\
&= \begin{bmatrix} \mathbf{x}_1^{(1)} w_1 + \mathbf{x}_2^{(1)} w_2 \\ \vdots \\ \mathbf{x}_1^{(n)} w_1 + \mathbf{x}_2^{(n)} w_2 \end{bmatrix} \\
&= \begin{bmatrix} \hat{y}^{(1)} \\ \vdots \\ \hat{y}^{(n)} \end{bmatrix}
\end{aligned}
$$

With just a single matrix-vector multiplication, we have computed our predictions for *all* of the *n* images simultaneously.

1-d example

- Linear regression finds the weight vector **w** that minimizes the f_{MSE}. Here's an example where each **x** is just 1-d…

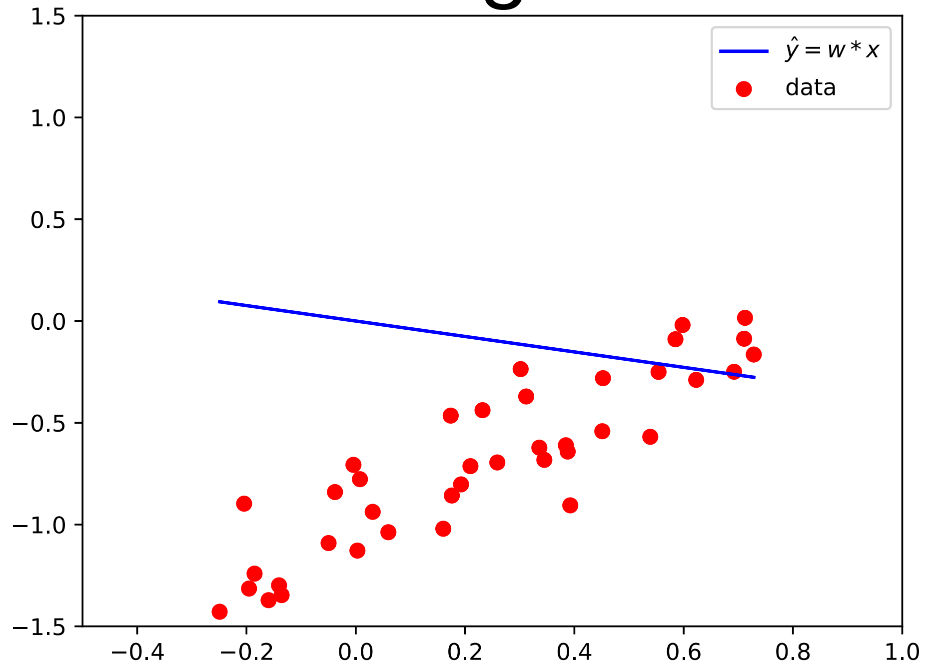


The best **w** is the one such that $f_{\mathrm{MSE}}(\mathbf{y}, \hat{\mathbf{y}})$ is as small as possible, where each $\hat{y} = \mathbf{x}^\top \mathbf{w}$.

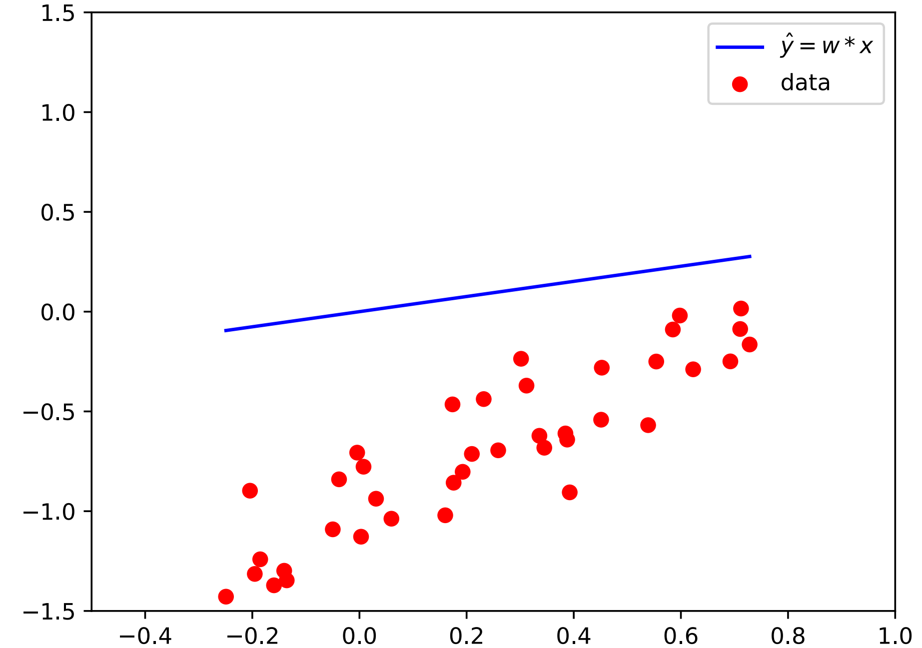
Exercise

- Which of the following regression lines would be predicted using the model described above?

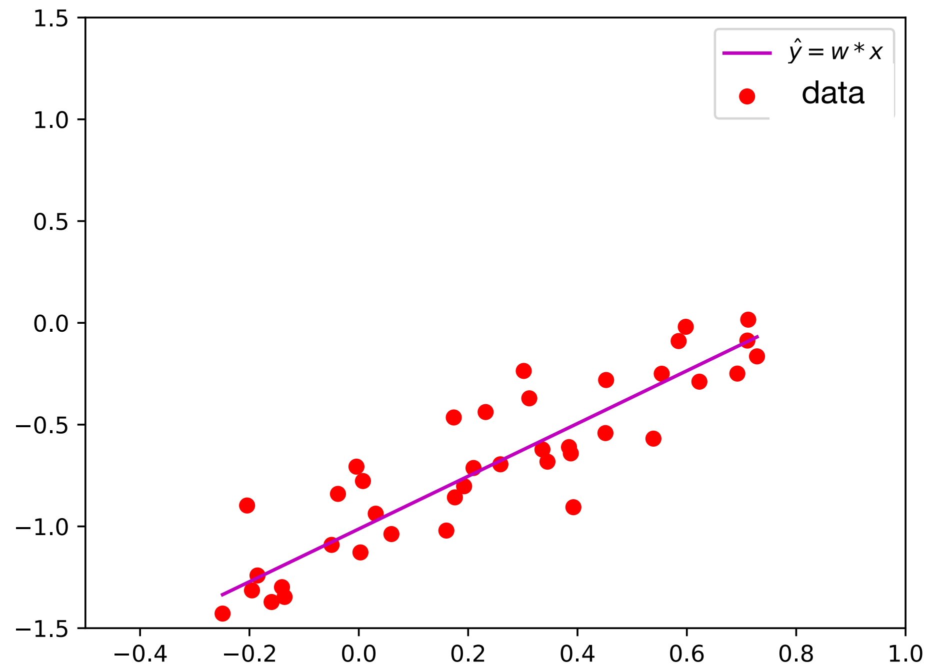
1.



2.

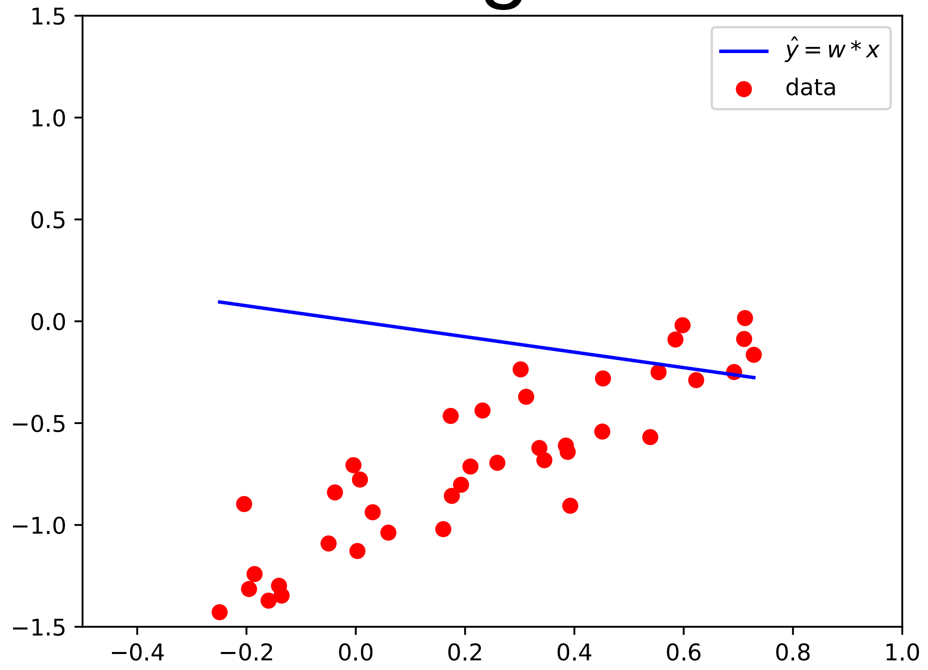


3.



Exercise

- Which of the following regression lines would be predicted using the model described above?

1. 

Notice that the model enforces that (x,y)=(0,0) lie in the graph. Because of this constraint, the model learns the wrong slope to minimize the MSE.

Bias term

- In order to account for target values *y* with non-zero mean, we could add a **bias term** *b* to our model:

$$\hat{y} = \mathbf{x}^\top \mathbf{w} + b$$

- We could then compute the gradient w.r.t. both **w** and *b* and solve.

$$\begin{aligned}
\nabla_{\mathbf{w}} f_{\mathrm{MSE}}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w}, b) &= \nabla_{\mathbf{w}} \left[\frac{1}{2n} \sum_{i=1}^{n} \left(\mathbf{x}^{(i)^\top} \mathbf{w} + b - y^{(i)} \right)^2 \right] \\
\nabla_{b} f_{\mathrm{MSE}}(\mathbf{y}, \hat{\mathbf{y}}; \mathbf{w}, b) &= \nabla_{b} \left[\frac{1}{2n} \sum_{i=1}^{n} \left(\mathbf{x}^{(i)^\top} \mathbf{w} + b - y^{(i)} \right)^2 \right]
\end{aligned}$$

Bias term

- Alternatively, we can implicitly include a bias term by augmenting each input vector **x** with a 1 at the end:

$$\tilde{\mathbf{x}} = \left[\begin{array}{c} \mathbf{x} \\ 1 \end{array} \right]$$

- Correspondingly, our weight vector **w** will have an extra component (bias term) at the end.

$$\tilde{\mathbf{w}} = \left[\begin{array}{c} \mathbf{w} \\ b \end{array} \right]$$

Bias term

- To see why, notice that:

$$\begin{aligned}
\hat{y} &= \tilde{\mathbf{x}}^\top \tilde{\mathbf{w}} \\
&= \left[\begin{array}{c} \mathbf{x} \\ 1 \end{array}\right]^\top \left[\begin{array}{c} \mathbf{w} \\ b \end{array}\right] \\
&= \left[\begin{array}{cc} \mathbf{x}^\top & 1 \end{array}\right] \left[\begin{array}{c} \mathbf{w} \\ b \end{array}\right] \\
&= \mathbf{x}^\top \mathbf{w} + b
\end{aligned}$$

Bias term

- We can find the optimal **w** and *b* based on all the training data using matrix notation.

- First define an augmented design matrix:

$$\tilde{\mathbf{X}} = \begin{bmatrix} \mathbf{x}^{(1)} & \dots & \mathbf{x}^{(n)} \\ 1 & \dots & 1 \end{bmatrix}$$

- Then compute:

$$\tilde{\mathbf{w}} = \left(\tilde{\mathbf{X}}\tilde{\mathbf{X}}^\top\right)^{-1}\tilde{\mathbf{X}}\mathbf{y}$$

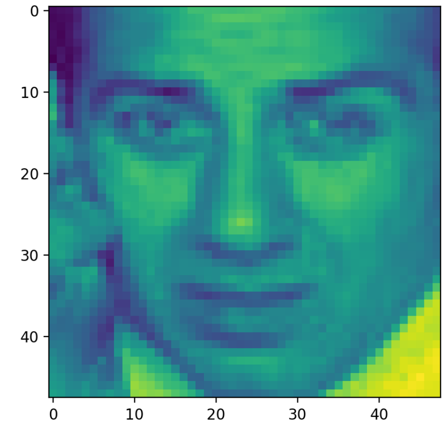
Example: age estimation

- Regress the age from 48x48 face images.
- Show demo…

Data augmentation

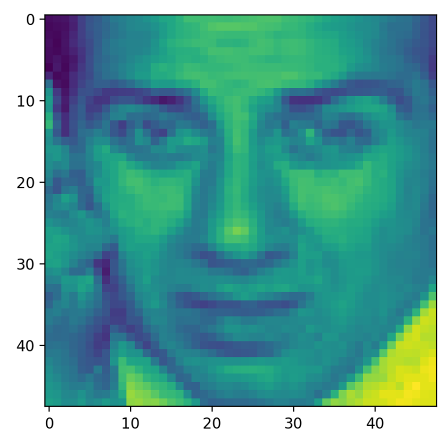
Data Augmentation

- How can we easily increase the number of training images?

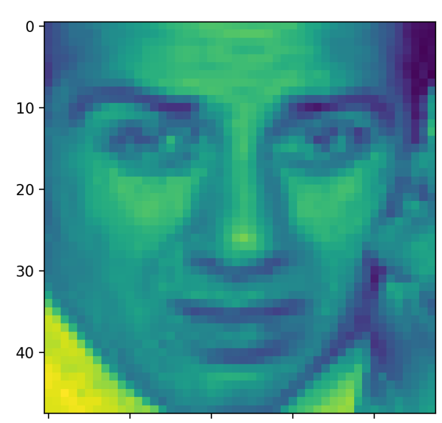


Data Augmentation

- How can we easily increase the number of training images?



- Flip them left-right:



Data Augmentation

- In **`numpy`**:
  - Let **`faces`** be a (*n* x 48 x 48) matrix containing *n* images.
  - Then **`facesFlipped = faces[:, :, ::-1]`** contains the left-right flipped images.

All *n* images.

Data Augmentation

- In **`numpy`**:
  - Let **`faces`** be a (n x 48 x 48) matrix containing n images.
  - Then **`facesFlipped = faces[:, :, ::-1]`** contains the left-right flipped images.

All 48 rows.

Data Augmentation

- In **`numpy`**:
  - Let **`faces`** be a (n x 48 x 48) matrix containing n images.
  - Then **`facesFlipped = faces[:, :, ::-1]`** contains the left-right flipped images.

All 48 columns but in reverse order.

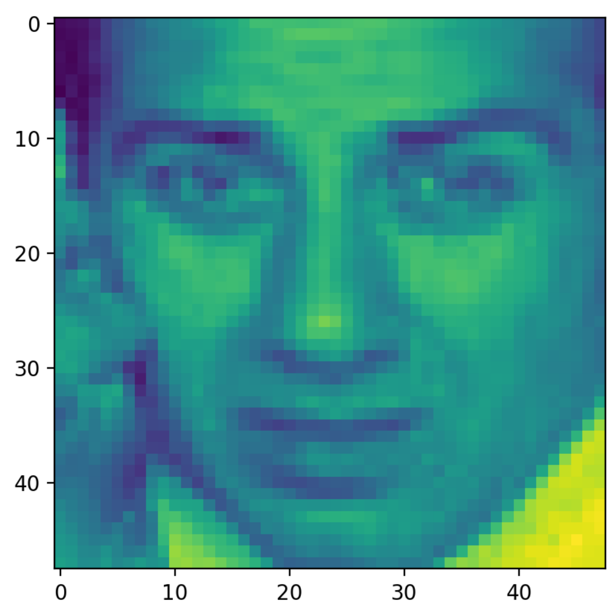
Data Augmentation

- Avoid leakage of facial identity information:
  - Make sure that no “flipped” image in the testing set has an “unflipped” image in the training set (and vice-versa).
- **Data leakage**: information in the training set which divulges information about the test set and which can bias the accuracy estimates.

Learned weights

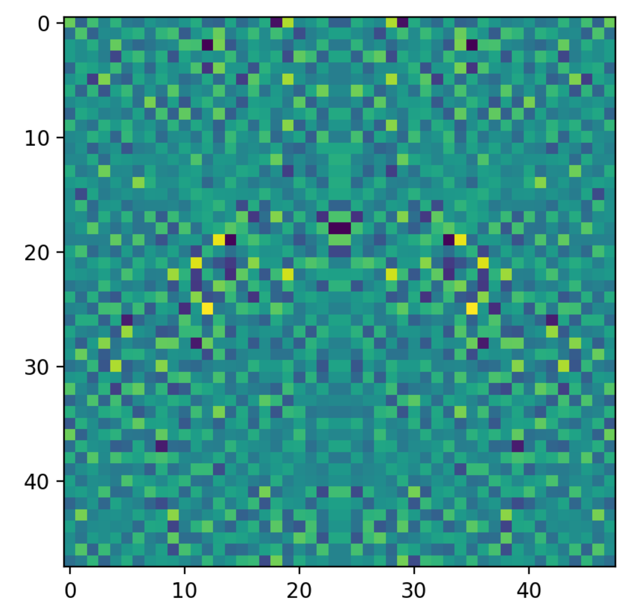
- Inspecting what the machine learned can be useful for debugging (and kinda fun to look at).

- For age estimation:



x

Furrows around nose?
Wrinkles in forehead?



w

Higher temperatures associated with larger age values.

Data Augmentation

- To augment a training set with more examples automatically, we need to use a **label-preserving transformation**:
    - Flipping left/right is a **transformation.**
    - It is **label-preserving** because faces look equally old whether you look at the original or a mirror-image.
- What are other examples of label-preserving transformation on images?

Regression for categorical data

Categorical data

- While computer vision and image analysis have motivated a lot of ML research, they are by no means the only application domain.
- Other big areas:
  - Speech
  - Text
  - Event logs

Case study: housing price prediction

- Suppose we want to predict housing prices, i.e., how much a house will sell for given a set of attributes (area, access to street, # fireplaces, etc.) about it.

- We could define a linear regression model:

  SalePrice = w_1 * Area + w_2 * NumFireplaces + … + b

- We can then use linear regression to train the optimal weights **w** to minimize the MSE.

Kaggle

- A handy resource to practice your ML skills is Kaggle, which is a website that hosts may machine learning competitions (with fabulous prizes).
- Example for House Prices: https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data