# INF721

2024/2

UFV

# Deep Learning

## L12: Normalization

# Logistics

**Announcements**

- PA3 is due on Oct 30th, Wednesday, 11:59pm

**Last Lecture**

- Pooling Layers
  - Max Pooling and Average Pooling
- Classic CNNs
  - LeNet-5, AlexNet and VGG-16
- Residual Neural Networks

# Lecture Outline

- Normalization
  - Input Normalization
  - Batch Normalization
  - Layer Normalization
- Recurrent Neural Networks

# Image Normalization

You might have noticed that in the first two programming assigments we've normalized the inputs images by dividing all pixel values by 255:

| | | | | |
|---|---|---|---|---|
| 206 | 205 | 247 | 245 | 244 |
| 244 | 161 | 137 | 244 | 254 |
| 192 | 154 | 75 | 200 | 249 |
| 90 | 109 | 96 | 143 | 223 |
| 67 | 69 | 107 | 196 | 236 |

Original image

image / 255 →

| | | | | |
|---|---|---|---|---|
| 0.80 | 0.80 | 0.96 | 245 | 0.96 |
| 0.95 | 0.63 | 0.53 | 0.95 | 0.99 |
| 0.75 | 0.60 | 0.29 | 0.78 | 0.97 |
| 0.35 | 0.42 | 0.37 | 0.56 | 0.87 |
| 0.26 | 0.27 | 0.41 | 0.76 | 0.92 |

Normalized image

**This type of normalization makes the learning process faster, because we are bringing the input values close to zero!**

UFV

# Input Normalization

Often we encounter datasets in which different input variables span very different ranges:

**House Price Prediction Dataset**

| Size (m2) | Number of Beds. | Nearest Subway Station (m) | Price (1000's of USD) |
|---|---|---|---|
| 152 | 4 | 7200 | 1550 |
| 229 | 3 | 3000 | 2286 |
| 84 | 1 | 1500 | 2930 |
| 95 | 3 | 12000 | 196 |
| ... | ... | | ... |

Such variations can make gradient descent training much more challenging!

- Assume Linear Regression with SGD :

$$\mathbf{w} = \mathbf{w} - \alpha \frac{\partial L}{\partial \mathbf{w}}$$

$$w = [0,0,0]$$
$$\alpha = 0.1$$

$$\mathbf{w} = [0,0,0] - 0.1(\hat{y}^{(i)} - y^{(i)})\mathbf{x}^{(i)}$$

$$\mathbf{w} = [0,0,0] - 0.1(0 - 1550)\mathbf{x}^{(i)}$$

$$\mathbf{w} = [0,0,0] + 155 \cdot [152,4,\underline{7200}]$$

**Changes in $w_3$ afect much more the output than $w_1$ and $w_2$**

# Input Normalization

When the input data $X$ is **not** normalized, the error surface will have very different curvatures along different axis:

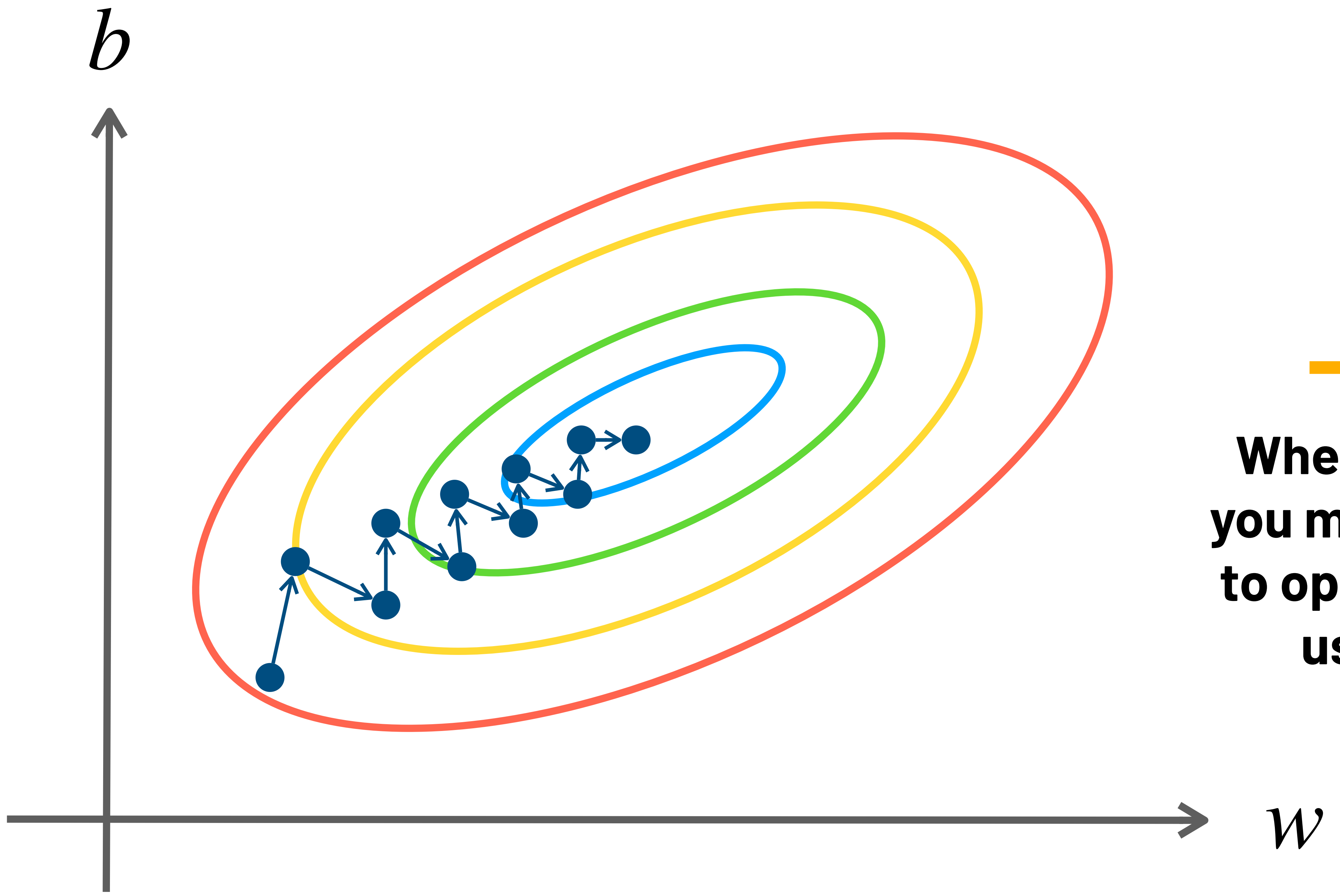



**Figure 1:** The curvature of the $w$ axis is much larger than the curvature of the $b$ axis.

Normalize $X$

**When you normalize your data, you make the loss surface easier to optimize, so you can typically use larger learning rates.**

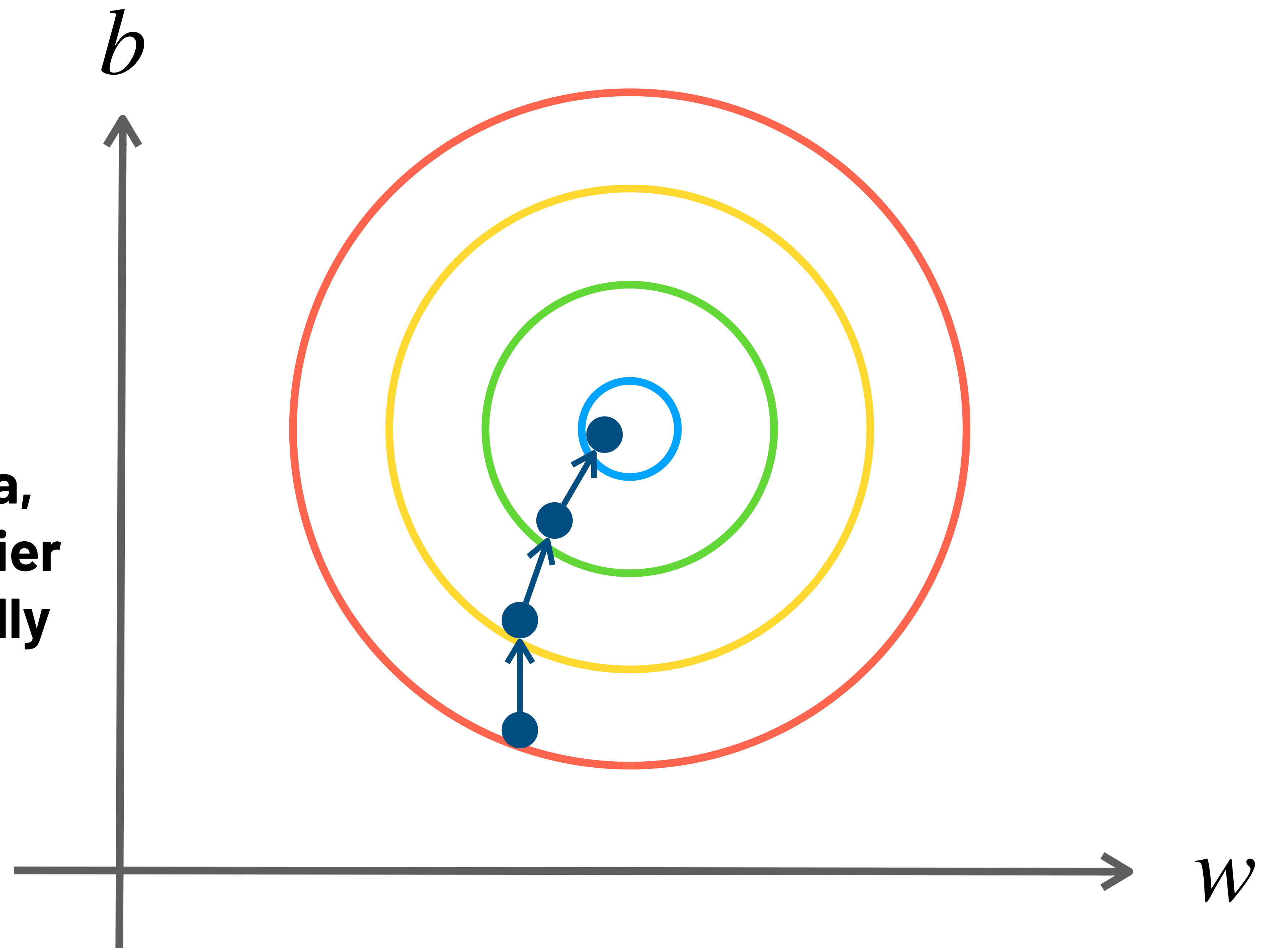
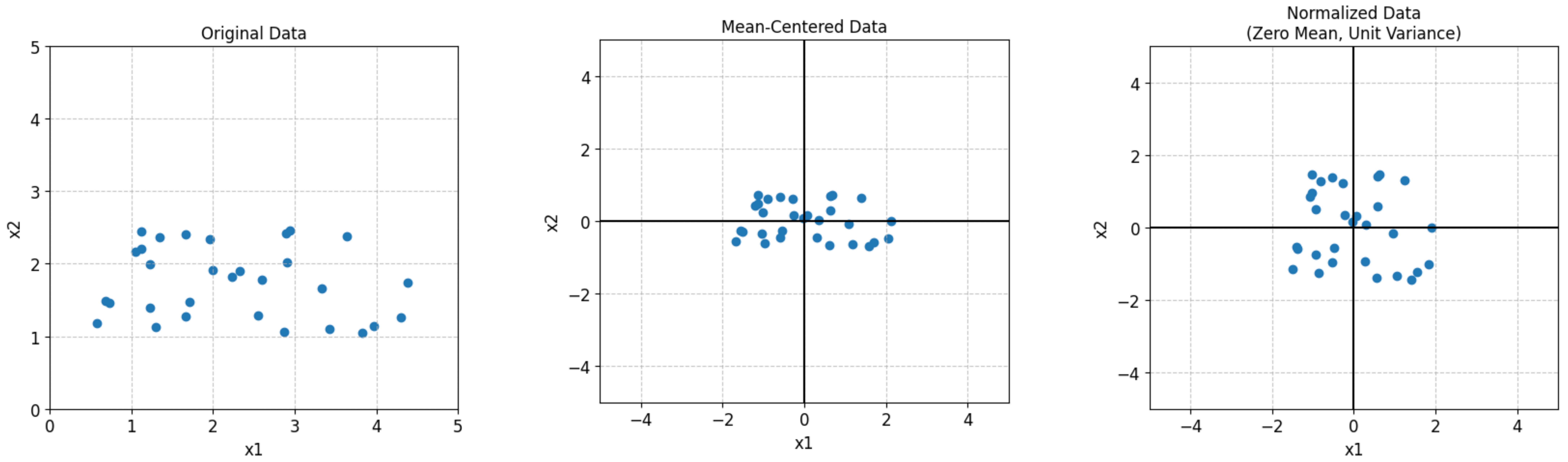



**Figure 2:** The curvature of the $w$ axis is equal to the curvature of the $b$ axis.

# How to Normalize the Input Data

To normalize your data, you need to make your examples have mean $\mu = 0$ and std dev. $\sigma = 1$:




Note that the same values of $\mu$ and $\sigma$ must be used to normalize the training, validation and test sets!

**1. Subtract the mean:**

$$\mathbf{x}^{(i)} = \mathbf{x}^{(i)} - \mu$$

$$\mathbf{x}^{(i)} = \mathbf{x}^{(i)} - \left(\frac{1}{m}\sum_{i=1}^{m} \mathbf{x}^{(i)}\right)$$

**2. Divide std. deviation:**

$$\mathbf{x}^{(i)} = \mathbf{x}^{(i)} / \sigma$$

$$\mathbf{x}^{(i)} = \frac{\mathbf{x}^{(i)}}{\sqrt{\left(\frac{1}{m}\sum_{i=1}^{m}\left((\mathbf{x}^{(i)} - \mu)^2\right)\right)}}$$

# Example 1: Normalizing Structured Datasets

We can also apply this idea to normalize images, which can be done across channels or not:

**House Price Prediction Dataset**

| Size (m2) | Number of Beds. | Nearest Subway Station (m) |
|---|---|---|
| -0.0733 | -0.1898 | 5.4734 |
| 0.2292 | -0.1980 | 5.4688 |
| 0.1178 | -0.1960 | 5.4724 |
| -0.1454 | -0.1888 | 5.4720 |
| ... | ... | |

**1. Subtract the mean:**

$$\mathbf{x}^{(i)} = \mathbf{x}^{(i)} - \mu$$

$$\mathbf{x}^{(i)} = \mathbf{x}^{(i)} - \left(\frac{1}{m}\sum_{i=1}^{m}\mathbf{x}^{(i)}\right)$$

**2. Divide std. deviation:**

$$\mathbf{x}^{(i)} = \mathbf{x}^{(i)}/\sigma$$

$$\mathbf{x}^{(i)} = \frac{\mathbf{x}^{(i)}}{\sqrt{\left(\frac{1}{m}\sum_{i=1}^{m}\left((\mathbf{x}^{(i)} - \mu)^2\right)\right.}}$$

```
Parameter:
-----------
- X: dataset of size (d, m)

mean = np.mean(X, axis=1, keepdims=True)
std = np.std(X, axis=1, keepdims=True)
normalized = (X - mean) / (std + 1e-8)
```

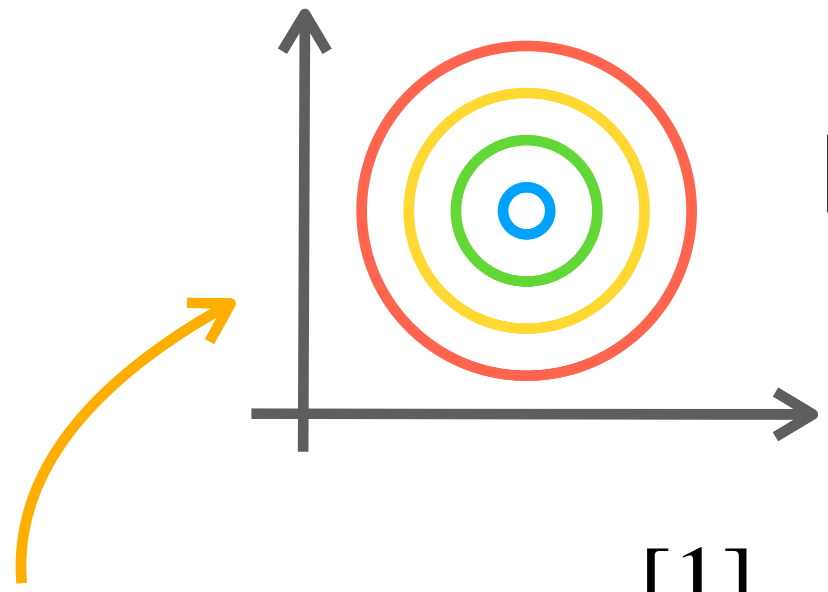
UFV

# Example 2: Normalizing Images

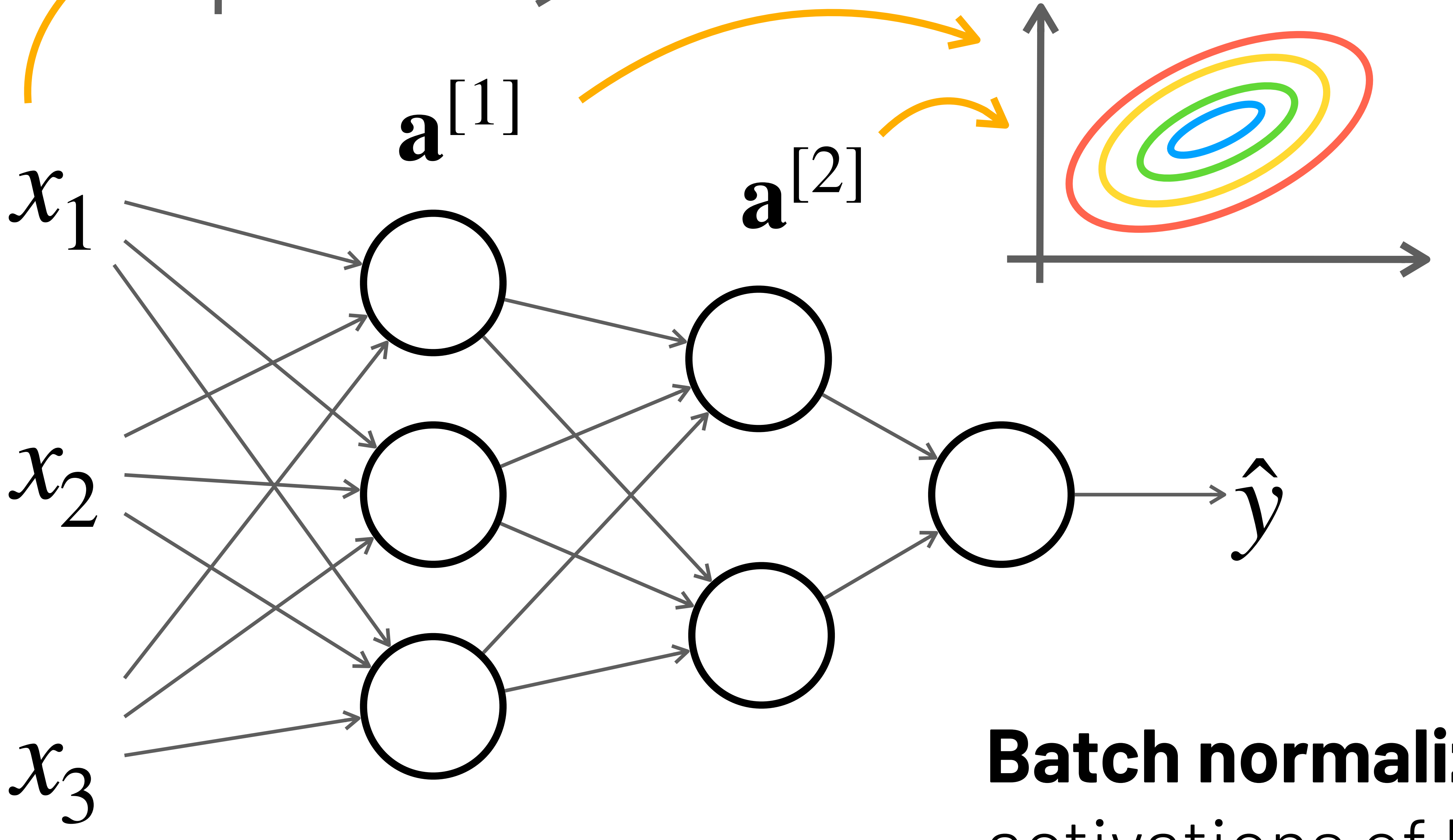
We can also apply this idea to normalize images, which can be done across channels or not:

```
Parameter:
-----------
- images : numpy.ndarray of shape (n_images, height, width, 3)

if normalization_type == 'zero_mean':
    # Zero mean and unit variance across all pixels and channels
    mean = np.mean(images)
    std = np.std(images)
    normalized = (images - mean) / (std + 1e-8)

elif normalization_type == 'zero_mean_per_channel':
    # Zero mean and unit variance per RGB channel
    mean = np.mean(images, axis=(0, 1, 2), keepdims=True)
    std = np.std(images, axis=(0, 1, 2), keepdims=True)
    normalized = (images - mean) / (std + 1e-8)
```

# Batch Normalization



Input normalization helps only the first hidden layer..




...but it does not guarantee that the next activations will be normalized

**Batch normalization** is a technique to normalize the activations of hidden layers in neural networks

Ioffe, S. ; Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Proceedings of the 32nd International Conference on Machine Learning.

# Batch Normalization

Given the linear outputs $\mathbf{z}^{[l](1)}, \mathbf{z}^{[l](2)}, \ldots, \mathbf{z}^{[l](m)}$ of a layer $l$ for a minibatch with $m$ examples, batch normalization normalizes $\mathbf{z}^{[l](i)}$ these values as follows:

Batch mean

$$\mu = \frac{1}{m}\sum_{i=1}^{m} \mathbf{z}^{[l](i)}$$

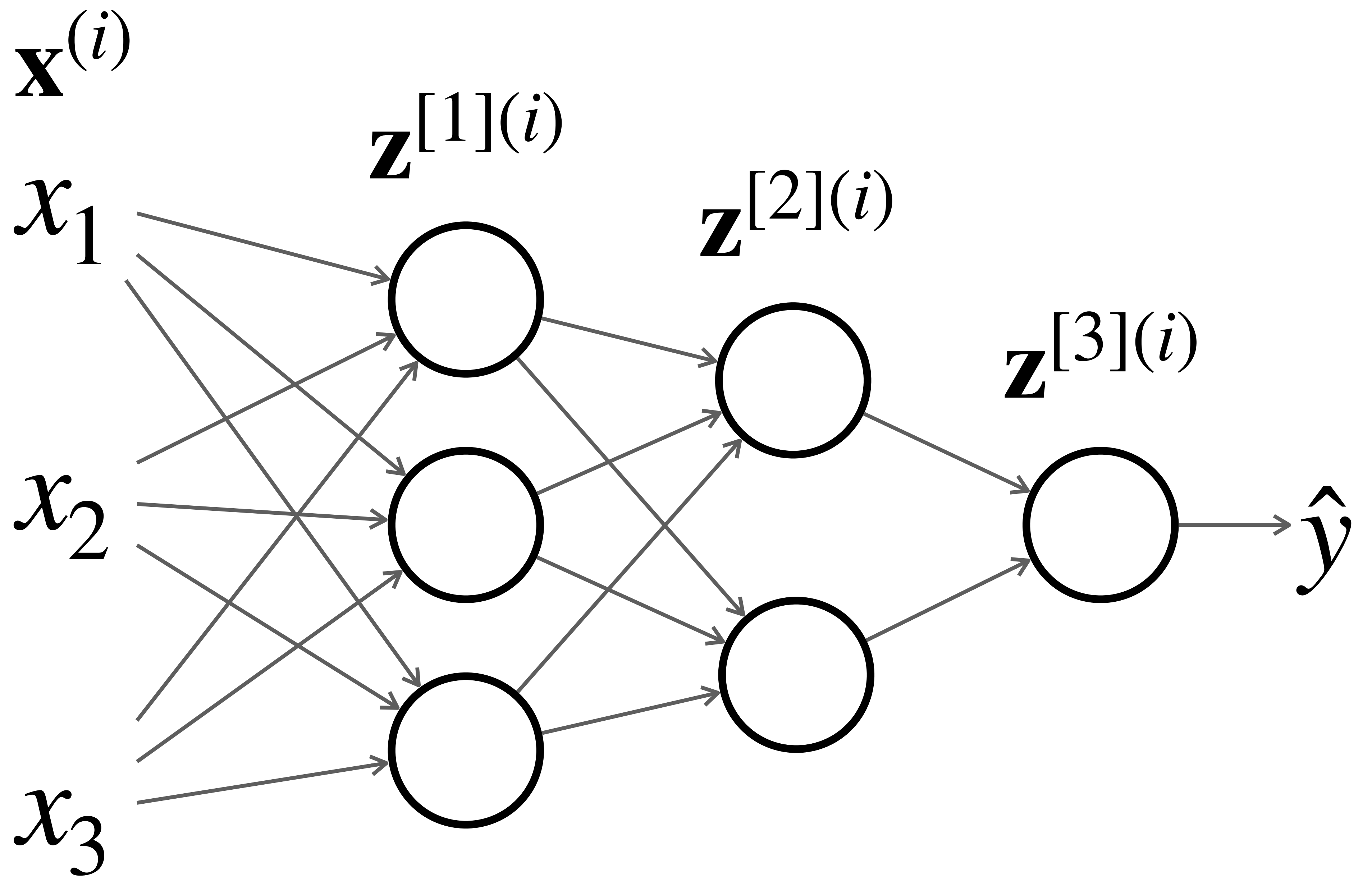
Batch variance

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} (\mathbf{z}^{[l](i)} - \mu)^2$$

$$\mathbf{z}^{[l](i)} = \frac{\mathbf{z}^{[l](i)} - \mu}{\sqrt{(\sigma^2) + \epsilon}}$$

**Learnable parameters!**

$$\tilde{\mathbf{z}}^{[l](i)} = \gamma \odot \mathbf{z}^{[l](i)} + \beta$$




Batch norm learn the mean $\beta$ and variance $\gamma$ of the activations!

UFV

Ioffe, S. ; Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Proceedings of the 32nd International Conference on Machine Learning.

# Example: Batch Normalization

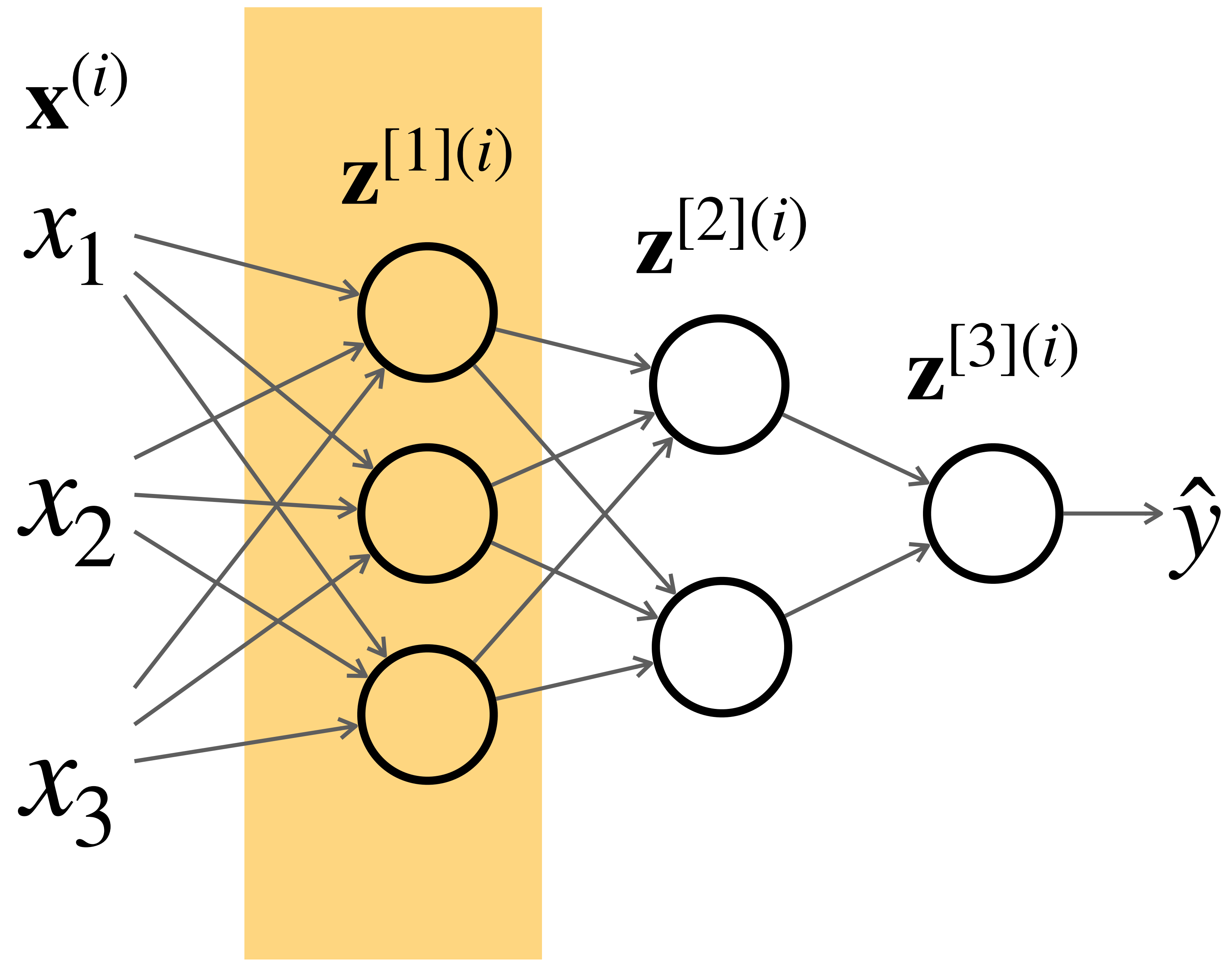
$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$X = \begin{array}{cccc} \begin{bmatrix} 1.0 & 2.0 & -1.0 & 0.0 \\ 0.5 & 1.0 & 0.5 & -1.0 \\ 0.0 & 0.0 & 1.0 & -0.5 \end{bmatrix} \\ \mathbf{x}^{(1)} \quad \mathbf{x}^{(2)} \quad \mathbf{x}^{(3)} \quad \mathbf{x}^{(4)} \end{array} \qquad W^{[1]} = \begin{bmatrix} 0.1 & 0.2 & -0.1 \\ -0.2 & 0.1 & 0.2 \\ 0.1 & -0.1 & 0.1 \end{bmatrix} \qquad b^{[1]} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$Z^{[1]} = \begin{array}{c} \begin{bmatrix} 0.2 & 0.4 & -0.1 & -0.15 \\ -0.15 & -0.3 & 0.45 & -0.2 \\ 0.05 & 0.1 & -0.05 & 0.05 \end{bmatrix} \\ \mathbf{z}^{[1](1)} \quad \mathbf{z}^{[1](2)} \quad \mathbf{z}^{[1](3)} \quad \mathbf{z}^{[1](4)} \end{array}$$

$$\mu = \frac{1}{m}\sum_{i=1}^{m} \mathbf{z}^{[l](i)}$$ **Batch mean**

$$= \frac{1}{4} \cdot \left( \begin{bmatrix} 0.2 \\ -0.15 \\ 0.05 \end{bmatrix} + \begin{bmatrix} 0.4 \\ -0.3 \\ 0.1 \end{bmatrix} + \begin{bmatrix} -0.1 \\ 0.45 \\ -0.5 \end{bmatrix} + \begin{bmatrix} -0.15 \\ -0.2 \\ 0.05 \end{bmatrix} \right) = \begin{bmatrix} 0.08 \\ -0.05 \\ 0.03 \end{bmatrix}$$



# Example: Batch Normalization

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$X$

| | | | |
|---|---|---|---|
| 1.0 | 2.0 | -1.0 | 0.0 |
| 0.5 | 1.0 | 0.5 | -1.0 |
| 0.0 | 0.0 | 1.0 | -0.5 |
| $\mathbf{x}^{(1)}$ | $\mathbf{x}^{(2)}$ | $\mathbf{x}^{(3)}$ | $\mathbf{x}^{(4)}$ |

$W^{[1]}$

| | | |
|---|---|---|
| 0.1 | 0.2 | -0.1 |
| -0.2 | 0.1 | 0.2 |
| 0.1 | -0.1 | 0.1 |

$b^{[1]}$

| |
|---|
| 0 |
| 0 |
| 0 |

$\mu$

| |
|---|
| 0.08 |
| -0.05 |
| 0.03 |

$Z^{[1]}$

| | | | |
|---|---|---|---|
| 0.2 | 0.4 | -0.1 | -0.15 |
| -0.15 | -0.3 | 0.45 | -0.2 |
| 0.05 | 0.1 | -0.05 | 0.05 |
| $\mathbf{z}^{[1](1)}$ | $\mathbf{z}^{[1](2)}$ | $\mathbf{z}^{[1](3)}$ | $\mathbf{z}^{[1](4)}$ |

$$\sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{z}^{[l](i)} - \mu)^2$$ **Batch variance**

$$= \frac{1}{4} \cdot \left( \left(\begin{bmatrix} 0.2 \\ -0.15 \\ 0.05 \end{bmatrix} - \begin{bmatrix} 0.08 \\ -0.05 \\ 0.03 \end{bmatrix}\right)^2 + \left(\begin{bmatrix} 0.4 \\ -0.3 \\ 0.1 \end{bmatrix} - \begin{bmatrix} 0.08 \\ -0.05 \\ 0.03 \end{bmatrix}\right)^2 + \left(\begin{bmatrix} -0.1 \\ 0.45 \\ -0.5 \end{bmatrix} - \begin{bmatrix} 0.08 \\ -0.05 \\ 0.03 \end{bmatrix}\right)^2 + \left(\begin{bmatrix} -0.15 \\ -0.2 \\ 0.05 \end{bmatrix} - \begin{bmatrix} 0.08 \\ -0.05 \\ 0.03 \end{bmatrix}\right)^2 \right) = \begin{bmatrix} 0.05 \\ 0.08 \\ 0.02 \end{bmatrix}$$

# Example: Batch Normalization

**Batch normalization**

$$\mathbf{z}^{[l](i)} = \frac{\mathbf{z}^{[l](i)} - \mu}{\sqrt{(\sigma^2)} + \epsilon}$$

$$\mu = \begin{bmatrix} 0.08 \\ -0.05 \\ 0.03 \end{bmatrix} \qquad \sigma^2 = \begin{bmatrix} 0.05 \\ 0.08 \\ 0.02 \end{bmatrix}$$

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$\begin{array}{cccc} \mathbf{z}^{[1](1)} & \mathbf{z}^{[1](2)} & \mathbf{z}^{[1](3)} & \mathbf{z}^{[1](4)} \\ \end{array}$$

$$\begin{bmatrix} 0.2 & 0.4 & -0.1 & -0.15 \\ -0.15 & -0.3 & 0.45 & -0.2 \\ 0.05 & 0.1 & -0.05 & 0.05 \end{bmatrix}$$

$$\frac{\begin{bmatrix} 0.2 \\ -0.15 \\ 0.05 \end{bmatrix} - \begin{bmatrix} 0.08 \\ -0.05 \\ 0.03 \end{bmatrix}}{\sqrt{\begin{bmatrix} 0.05 \\ 0.08 \\ 0.02 \end{bmatrix}}} \quad \frac{\begin{bmatrix} 0.4 \\ -0.3 \\ 0.1 \end{bmatrix} - \begin{bmatrix} 0.08 \\ -0.05 \\ 0.03 \end{bmatrix}}{\sqrt{\begin{bmatrix} 0.05 \\ 0.08 \\ 0.02 \end{bmatrix}}} \quad \frac{\begin{bmatrix} -0.1 \\ 0.45 \\ -0.05 \end{bmatrix} - \begin{bmatrix} 0.08 \\ -0.05 \\ 0.03 \end{bmatrix}}{\sqrt{\begin{bmatrix} 0.05 \\ 0.08 \\ 0.02 \end{bmatrix}}} \quad \frac{\begin{bmatrix} -0.15 \\ -0.2 \\ 0.05 \end{bmatrix} - \begin{bmatrix} 0.08 \\ -0.05 \\ 0.03 \end{bmatrix}}{\sqrt{\begin{bmatrix} 0.05 \\ 0.08 \\ 0.02 \end{bmatrix}}} =$$

$$\underset{\gamma}{\begin{bmatrix} a1 \\ a2 \\ a3 \end{bmatrix}} \odot \underset{Z^{[1]}\text{ normalized}}{\begin{bmatrix} 0.50 & 1.39 & -0.83 & -1.05 \\ -0.34 & -0.85 & 1.70 & -0.51 \\ 0.22 & 1.14 & -1.60 & 0.22 \end{bmatrix}} + \underset{\beta}{\begin{bmatrix} b1 \\ b2 \\ b3 \end{bmatrix}}$$

$$\begin{array}{cccc} \mathbf{z}^{[1](1)} & \mathbf{z}^{[1](2)} & \mathbf{z}^{[1](3)} & \mathbf{z}^{[1](4)} \\ \end{array}$$

# Batch Normalization in Numpy

Batch normalization takes the mean and averages across the examples (axis = 1):

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

| $\mathbf{z}^{[1](1)}$ | $\mathbf{z}^{[1](2)}$ | $\mathbf{z}^{[1](3)}$ | $\mathbf{z}^{[1](4)}$ |
|---|---|---|---|
| 0.2 | 0.4 | -0.1 | -0.15 |
| -0.15 | -0.3 | 0.45 | -0.2 |
| 0.05 | 0.1 | -0.5 | 0.05 |

$Z^{[1]}$ normalized

| $\mathbf{z}^{[1](1)}$ | $\mathbf{z}^{[1](2)}$ | $\mathbf{z}^{[1](3)}$ | $\mathbf{z}^{[1](4)}$ |
|---|---|---|---|
| 0.50 | 1.39 | -0.83 | -1.05 |
| -0.34 | -0.85 | 1.70 | -0.51 |
| 0.22 | 1.14 | -1.60 | 0.22 |

```python
def batch_norm(Z, gamma, beta, epsilon=1e-8):
  m = Z.shape[1]

  # Calculate the mean
  mean = 1/m * np.sum(Z, axis=1, keepdims=True)

  # Calculate the variance
  variance = 1/m * np.sum((Z - mean)**2, axis=1, keepdims=True)

  # Normalize Z
  Z_norm = (Z - mean)/(np.sqrt(variance) + epsilon)

  # Rescale distribution to mean beta and variance gamma
  return gamma * Z_norm + beta
```

# Batch Normalization in PyTorch

Defining a fully connected network in PyTorch with Batch Normalization:

```
# Define your neural network architecture with batch normalization
class MLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.layers = nn.Sequential(
            nn.Flatten(),              # Flatten the input image tensor
            nn.Linear(28 * 28, 64),    # Fully connected layer from 28*28 to 64 neurons
            nn.BatchNorm1d(64),        # Batch normalization
            nn.ReLU(),                 # ReLU activation function
            nn.Linear(64, 32),         # Fully connected layer from 64 to 32 neurons
            nn.BatchNorm1d(32),        # Batch normalization
            nn.ReLU(),                 # ReLU activation function
            nn.Linear(32, 10)          # Fully connected layer from 32 to 10 neurons
        )

    def forward(self, x):
        return self.layers(x)
```

UFV

# Layer Normalization

Instead of normalizing across examples withing a mini-batch, layer normalization normalizes the activations across features, for each example separately:

Layer mean

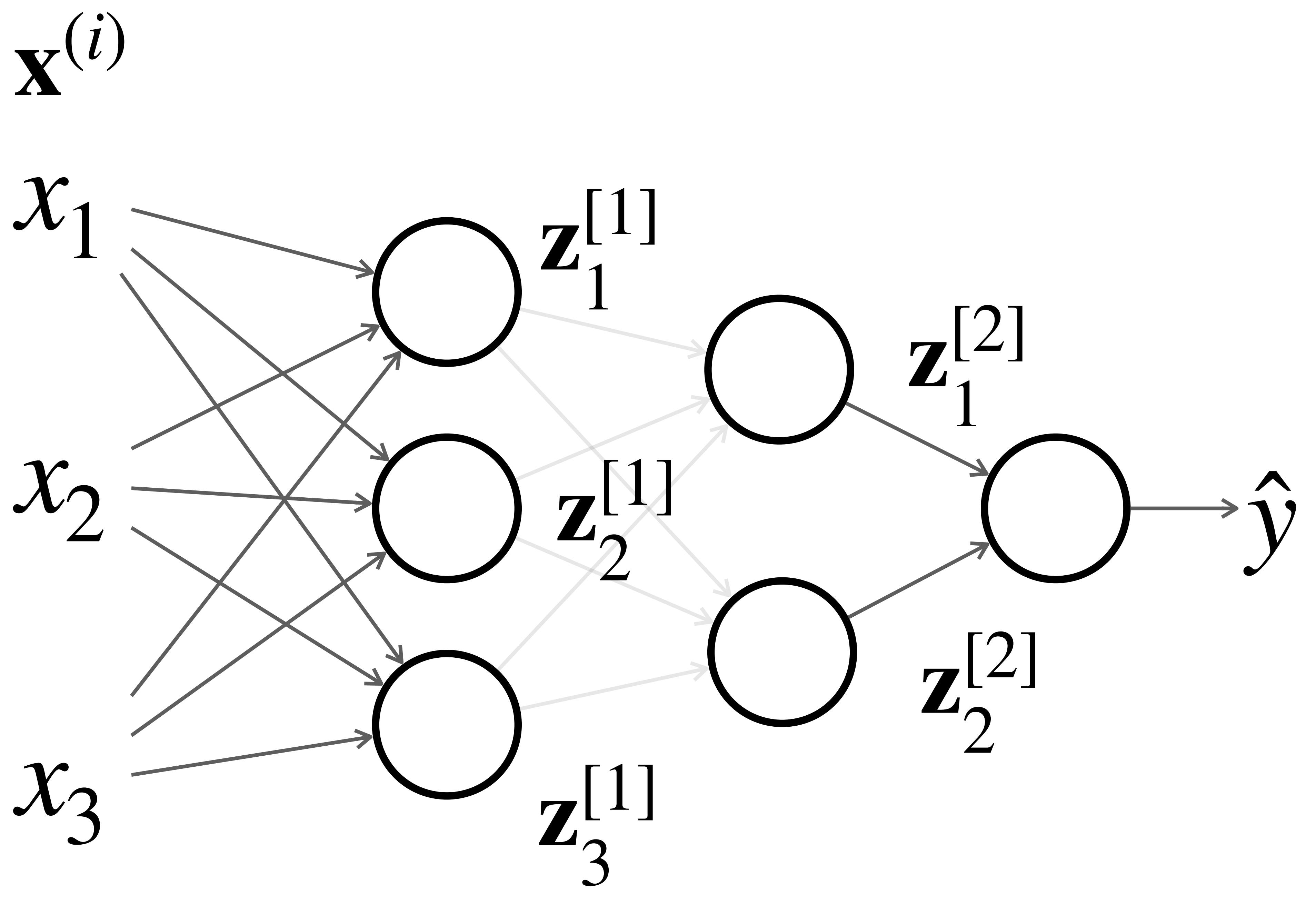
$$\mu = \frac{1}{n^{[l]}} \sum_{i=1}^{n^{[l]}} \mathbf{z}_i^{[l]}$$

Layer variance

$$\sigma^2 = \frac{1}{n^{[l]}} \sum_{i=1}^{n^{[l]}} (\mathbf{z}_i^{[l]} - \mu)^2$$

$$\mathbf{z}_i^{[l]} = \frac{\mathbf{z}_i^{[l]} - \mu}{\sqrt{(\sigma^2) + \epsilon}}$$

$$\tilde{\mathbf{z}}_i^{[l]} = \gamma \odot \mathbf{z}_i^{[l]\{i\}} + \beta$$




**Why not batch norm?** If the batch size is too small, then the estimates of mean and variance become too noisy

UFV

# Example: Layer Normalization

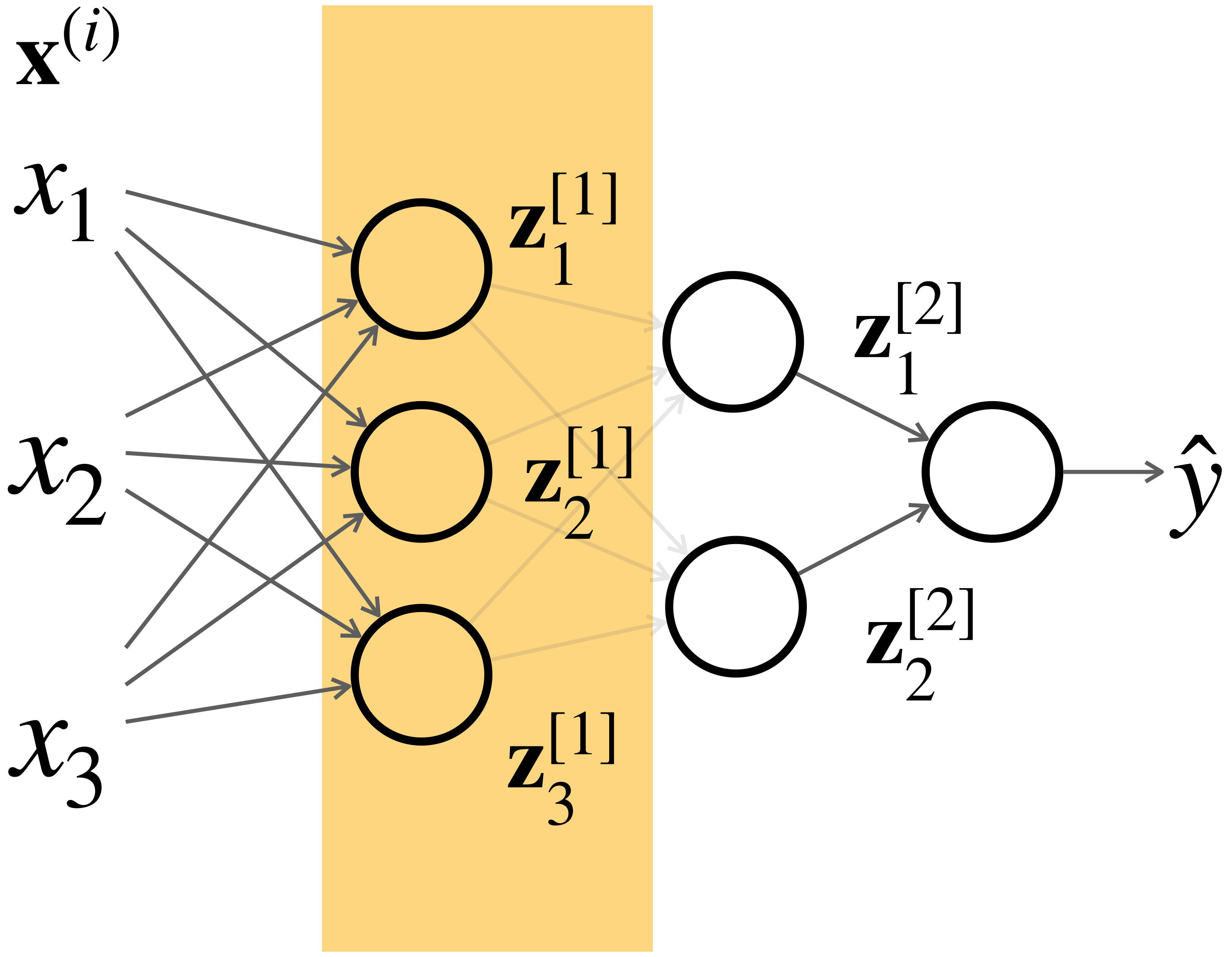
$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$$X = \begin{array}{cccc} \begin{bmatrix} 1.0 & 2.0 & -1.0 & 0.0 \\ 0.5 & 1.0 & 0.5 & -1.0 \\ 0.0 & 0.0 & 1.0 & -0.5 \end{bmatrix} \\ \mathbf{x}^{(1)} \quad \mathbf{x}^{(2)} \quad \mathbf{x}^{(3)} \quad \mathbf{x}^{(4)} \end{array} \qquad W^{[1]} = \begin{bmatrix} 0.1 & 0.2 & -0.1 \\ -0.2 & 0.1 & 0.2 \\ 0.1 & -0.1 & 0.1 \end{bmatrix} \qquad b^{[1]} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{array}{c} \mathbf{z}_1^{[1]} \\ \mathbf{z}_2^{[1]} \\ \mathbf{z}_3^{[1]} \end{array} \begin{bmatrix} 0.2 & 0.4 & -0.1 & -0.15 \\ -0.15 & -0.3 & 0.45 & -0.2 \\ 0.05 & 0.1 & -0.05 & 0.05 \end{bmatrix}$$

$$\mu = \frac{1}{n^{[l]}} \sum_{i=1}^{n^{[l]}} \mathbf{z}_i^{[l]} = \frac{1}{3} \cdot \left( \begin{bmatrix} 0.2 & 0.4 & -0.1 & -0.15 \end{bmatrix} + \begin{bmatrix} -0.15 & -0.3 & 0.45 & -0.2 \end{bmatrix} + \begin{bmatrix} 0.05 & 0.1 & -0.5 & 0.05 \end{bmatrix} \right) = \begin{bmatrix} 0.03 & 0.06 & 0.1 & -0.1 \end{bmatrix}$$

**Layer mean**




UFV

# Example: Layer Normalization

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

$X$

| | | | |
|---|---|---|---|
| 1.0 | 2.0 | -1.0 | 0.0 |
| 0.5 | 1.0 | 0.5 | -1.0 |
| 0.0 | 0.0 | 1.0 | -0.5 |
| $\mathbf{x}^{(1)}$ | $\mathbf{x}^{(2)}$ | $\mathbf{x}^{(3)}$ | $\mathbf{x}^{(4)}$ |

$W^{[1]}$

| | | |
|---|---|---|
| 0.1 | 0.2 | -0.1 |
| -0.2 | 0.1 | 0.2 |
| 0.1 | -0.1 | 0.1 |

$b^{[1]}$

| |
|---|
| 0 |
| 0 |
| 0 |

| | | | | |
|---|---|---|---|---|
| $\mathbf{z}_1^{[1]}$ | 0.2 | 0.4 | -0.1 | -0.15 |
| $\mathbf{z}_2^{[1]}$ | -0.15 | -0.3 | 0.45 | -0.2 |
| $\mathbf{z}_3^{[1]}$ | 0.05 | 0.1 | -0.05 | 0.05 |

$$\sigma^2 = \frac{1}{n^{[l]}} \sum_{i=1}^{n^{[l]}} (\mathbf{z}_i^{[l]} - \mu)^2 = \frac{1}{3} \cdot \left( \underbrace{\begin{bmatrix} 0.2 & 0.4 & -0.1 & -0.15 \end{bmatrix}}_{\mathbf{z}_1^{[1]}} - \underbrace{\begin{bmatrix} 0.03 & 0.06 & 0.1 & -0.1 \end{bmatrix}}_{\mu} \right)^2 + \left( \underbrace{\begin{bmatrix} -0.15 & -0.3 & 0.45 & -0.2 \end{bmatrix}}_{\mathbf{z}_2^{[1]}} - \underbrace{\begin{bmatrix} 0.03 & 0.06 & 0.1 & -0.1 \end{bmatrix}}_{\mu} \right)^2 + \left( \underbrace{\begin{bmatrix} 0.05 & 0.1 & -0.5 & 0.05 \end{bmatrix}}_{\mathbf{z}_3^{[1]}} - \underbrace{\begin{bmatrix} 0.03 & 0.06 & 0.1 & -0.1 \end{bmatrix}}_{\mu} \right)^2$$

$$= \begin{bmatrix} 0.02 & 0.08 & 0.06 & 0.01 \end{bmatrix}$$

**Layer variance**

# Example: Layer Normalization

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

| | | | | |
|---|---|---|---|---|
| $\mathbf{z}_1^{[1]}$ | 0.2 | 0.4 | -0.1 | -0.15 |
| $\mathbf{z}_2^{[1]}$ | -0.15 | -0.3 | 0.45 | -0.2 |
| $\mathbf{z}_3^{[1]}$ | 0.05 | 0.1 | -0.05 | 0.05 |

**Layer normalization**

$$\mathbf{z}_i^{[l]} = \frac{\mathbf{z}_i^{[l]} - \mu}{\sqrt{(\sigma^2) + \epsilon}}$$

| | | | | |
|---|---|---|---|---|
| $\mu$ | 0.03 | 0.06 | 0.1 | -0.1 |
| $\sigma^2$ | 0.02 | 0.08 | 0.06 | 0.01 |

$$\frac{\begin{bmatrix} 0.2 & 0.4 & -0.1 & -0.15 \end{bmatrix} - \begin{bmatrix} 0.03 & 0.06 & 0.1 & -0.1 \end{bmatrix}}{\sqrt{\begin{bmatrix} 0.02 & 0.08 & 0.06 & 0.01 \end{bmatrix}}}$$

$$\frac{\begin{bmatrix} -0.15 & -0.3 & 0.45 & -0.2 \end{bmatrix} - \begin{bmatrix} 0.03 & 0.06 & 0.1 & -0.1 \end{bmatrix}}{\sqrt{\begin{bmatrix} 0.02 & 0.08 & 0.06 & 0.01 \end{bmatrix}}}$$

$$\frac{\begin{bmatrix} 0.05 & 0.1 & -0.5 & 0.05 \end{bmatrix} - \begin{bmatrix} 0.03 & 0.06 & 0.1 & -0.1 \end{bmatrix}}{\sqrt{\begin{bmatrix} 0.02 & 0.08 & 0.06 & 0.01 \end{bmatrix}}}$$

$$= \overset{\gamma}{\begin{bmatrix} a1 \\ a2 \\ a3 \end{bmatrix}} \odot \begin{array}{c} \\ \mathbf{z}_1^{[1]} \\ \mathbf{z}_2^{[1]} \\ \mathbf{z}_3^{[1]} \end{array} \begin{array}{c} Z^{[1]} \text{ normalized} \\ \begin{bmatrix} 1.16 & 1.16 & -0.80 & -0.46 \\ -1.27 & -1.27 & 1.40 & -0.92 \\ 0.11 & 0.11 & -0.60 & 1.38 \end{bmatrix} \end{array} + \overset{\beta}{\begin{bmatrix} b1 \\ b2 \\ b3 \end{bmatrix}}$$

UFV

# Layer Normalization in Numpy

Layer normalization takes the mean and averages across the features (axis = 0):

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$

| | | | | |
|---|---|---|---|---|
| $\mathbf{z}_1^{[1]}$ | 0.2 | 0.4 | -0.1 | -0.15 |
| $\mathbf{z}_2^{[1]}$ | -0.15 | -0.3 | 0.45 | -0.2 |
| $\mathbf{z}_3^{[1]}$ | 0.05 | 0.1 | -0.05 | 0.05 |

$Z^{[1]}$ normalized

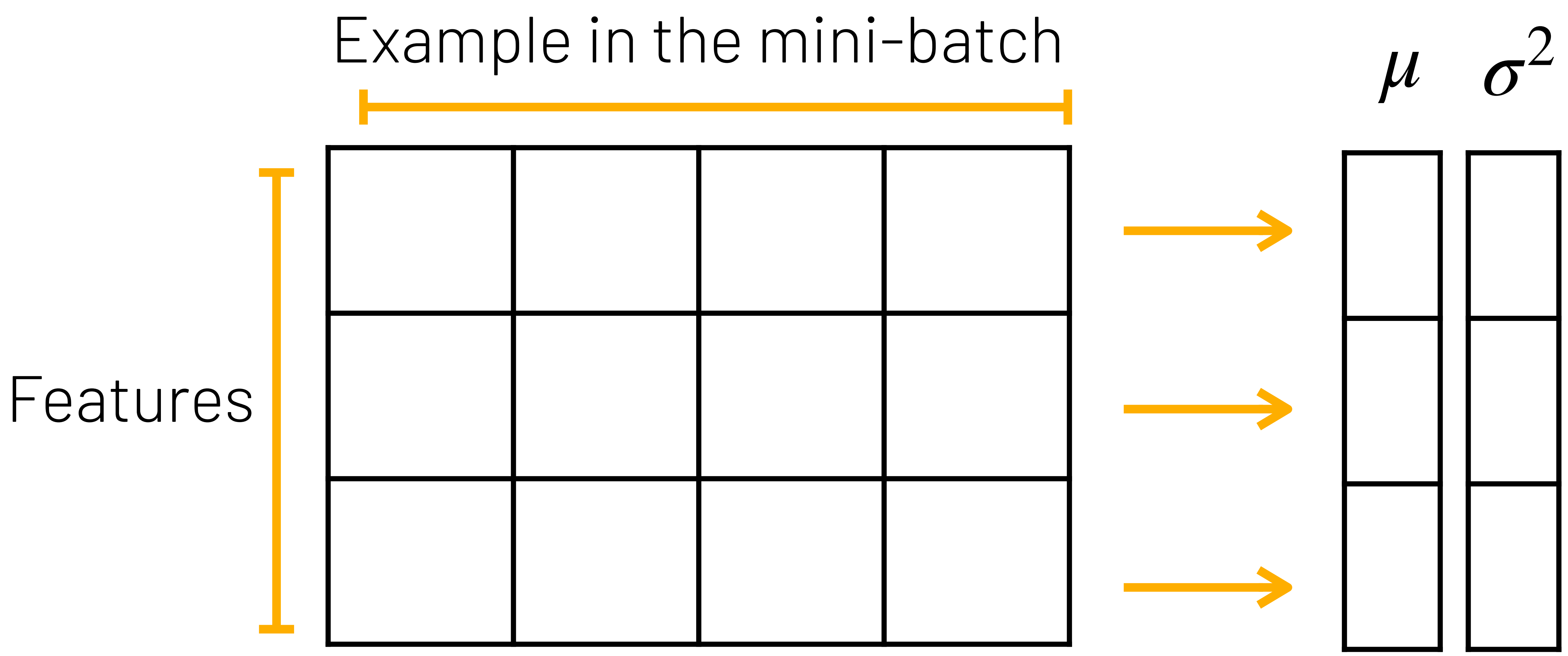
| | | | | |
|---|---|---|---|---|
| $\mathbf{z}_1^{[1]}$ | 1.16 | 1.16 | -0.80 | -0.46 |
| $\mathbf{z}_2^{[1]}$ | -1.27 | -1.27 | 1.40 | -0.92 |
| $\mathbf{z}_3^{[1]}$ | 0.11 | 0.11 | -0.60 | 1.38 |

```python
def layer_norm(Z, gamma, beta, epsilon=1e-8):
  n = Z.shape[0]

  # Calculate the mean
  mean = 1/n * np.sum(Z, axis=0, keepdims=True)

  # Calculate the variance
  variance = 1/n * np.sum((Z - mean)**2, axis=0, keepdims=True)

  # Normalize Z
  Z_norm = (Z - mean)/(np.sqrt(variance) + epsilon)

  # Rescale distribution to mean beta and variance gamma
  return gamma * Z_norm + beta
```

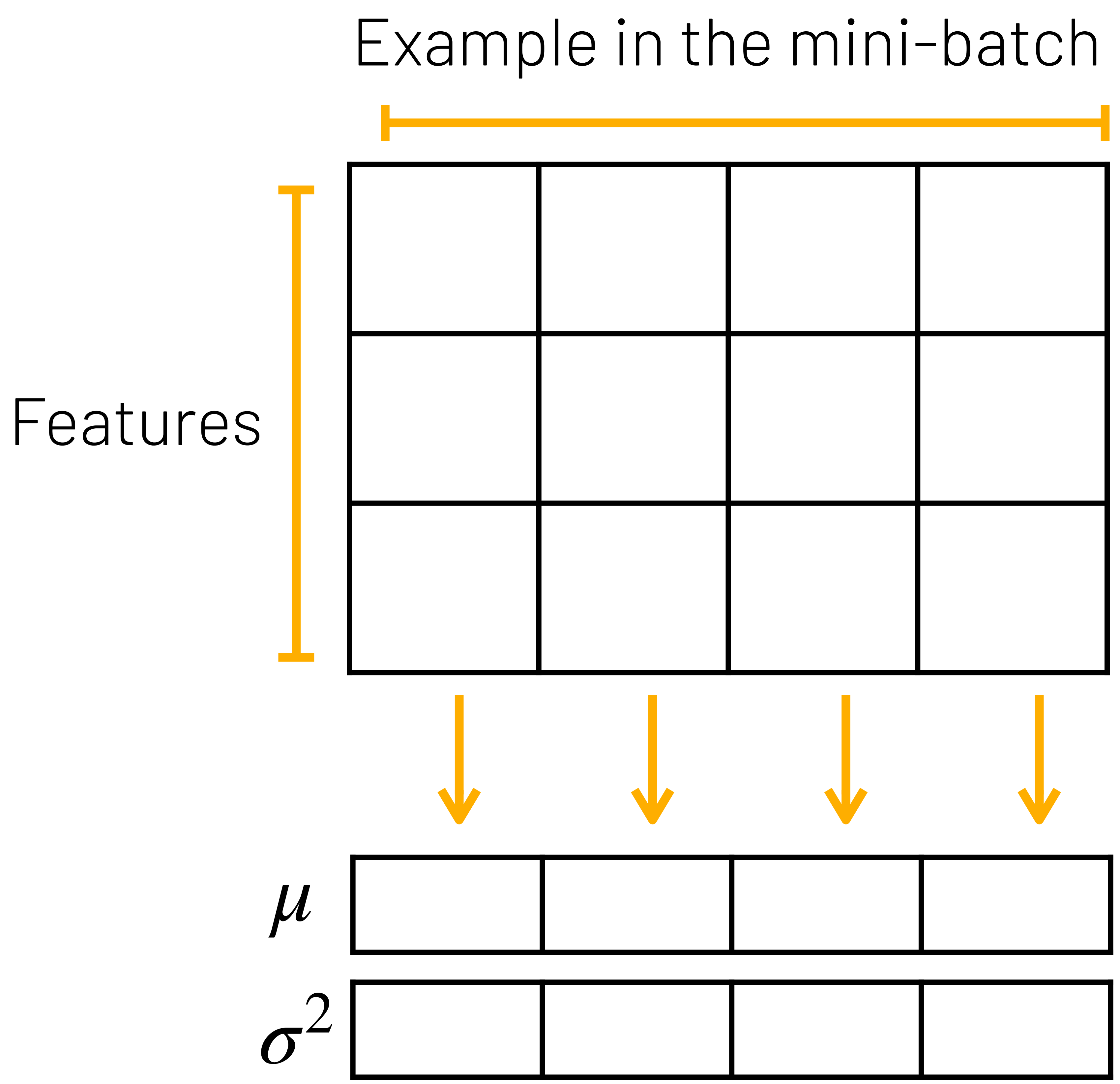
UFV

# Batch Norm vs. Layer Norm

**Batch norm:** normalizes across the examples (axis = 1)




**Layer norm:** normalizes across the input features (axis = 0)




UFV

# Next Lecture

**L12**: Recurrent Neural Networks

Sequential problems, basic recurrent neural networks, backpropagation through time, one-hot encodging, language models