

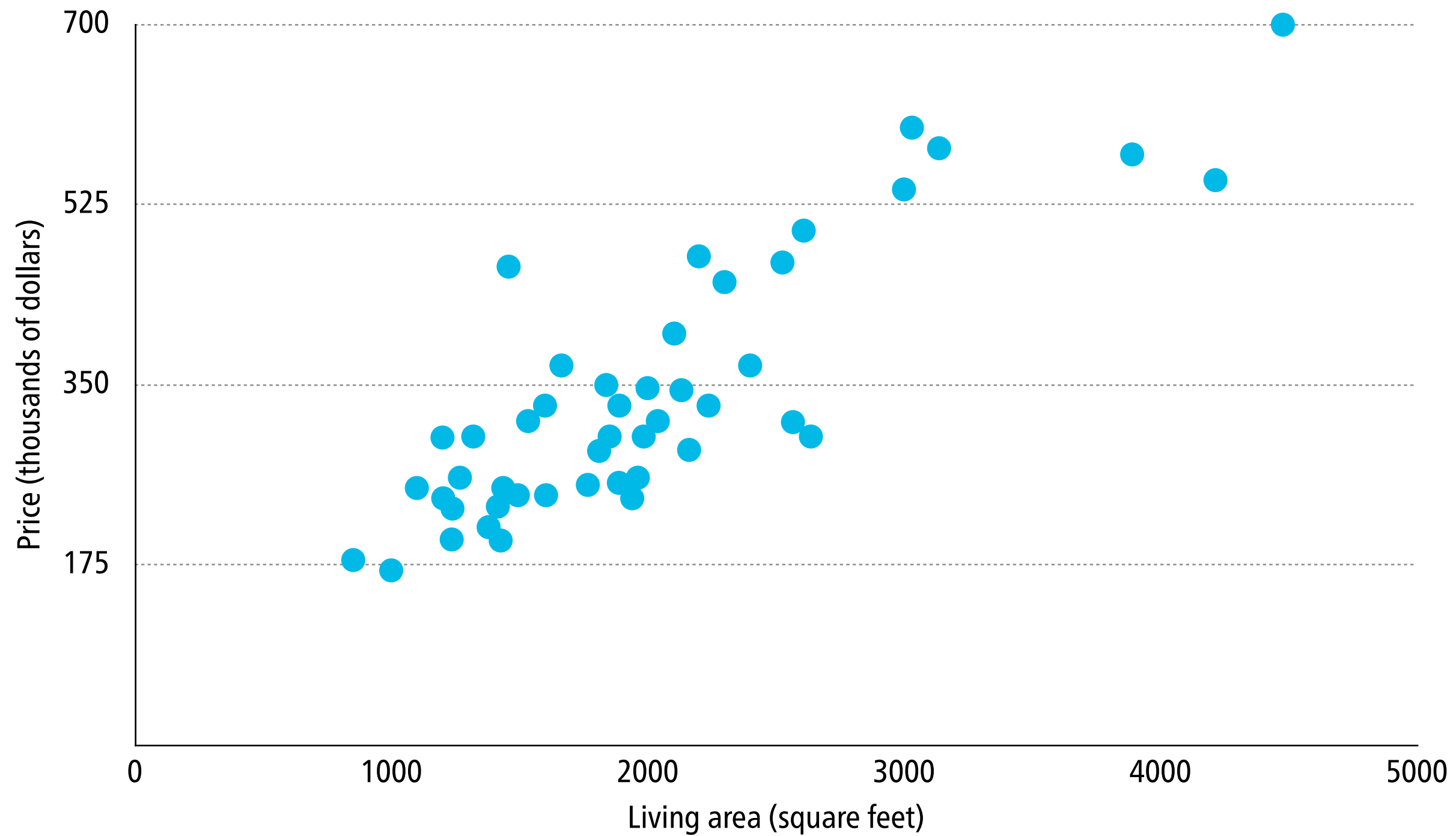
Natural Language Processing

# Linear layers

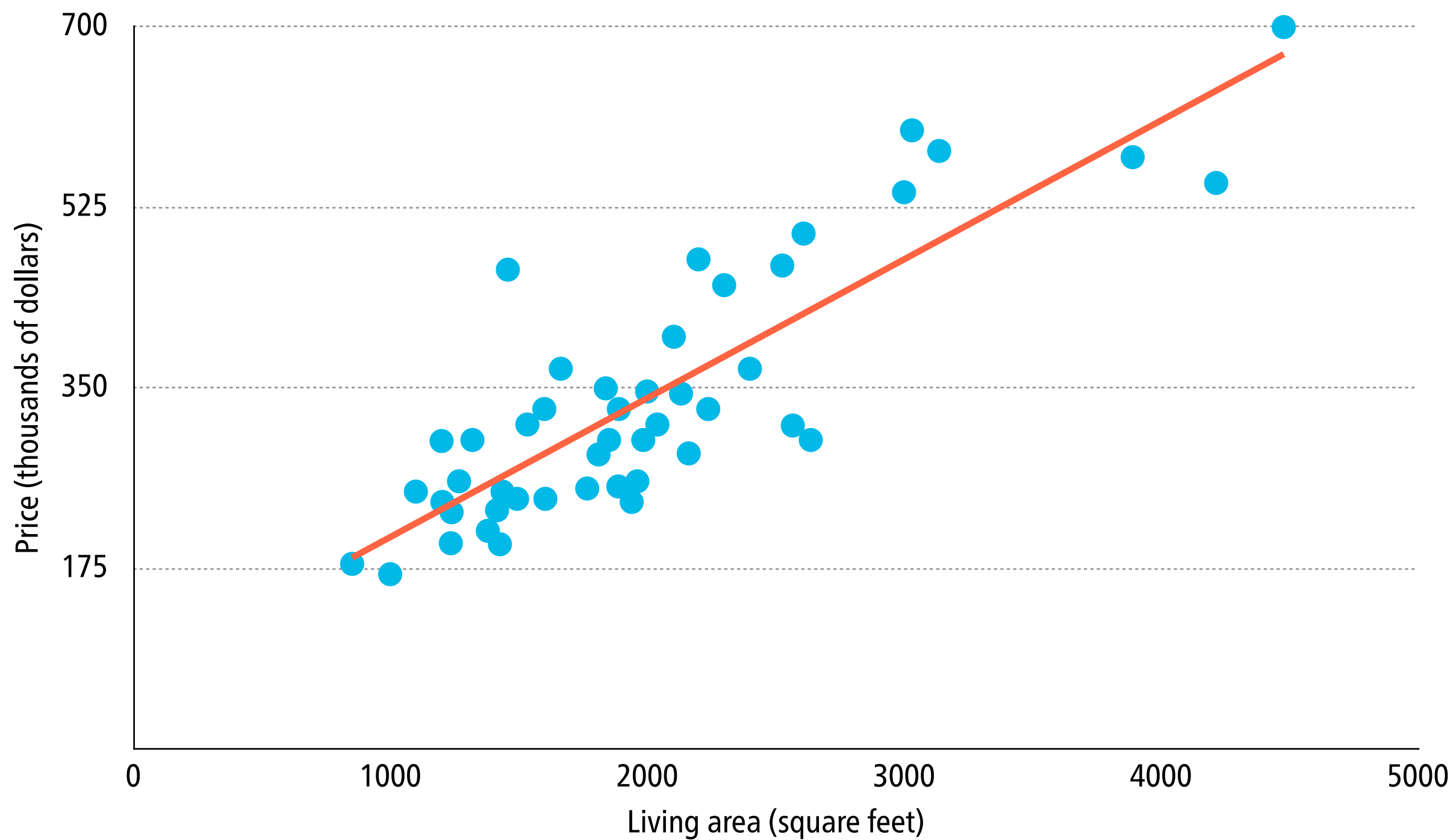
Marco Kuhlmann

Department of Computer and Information Science

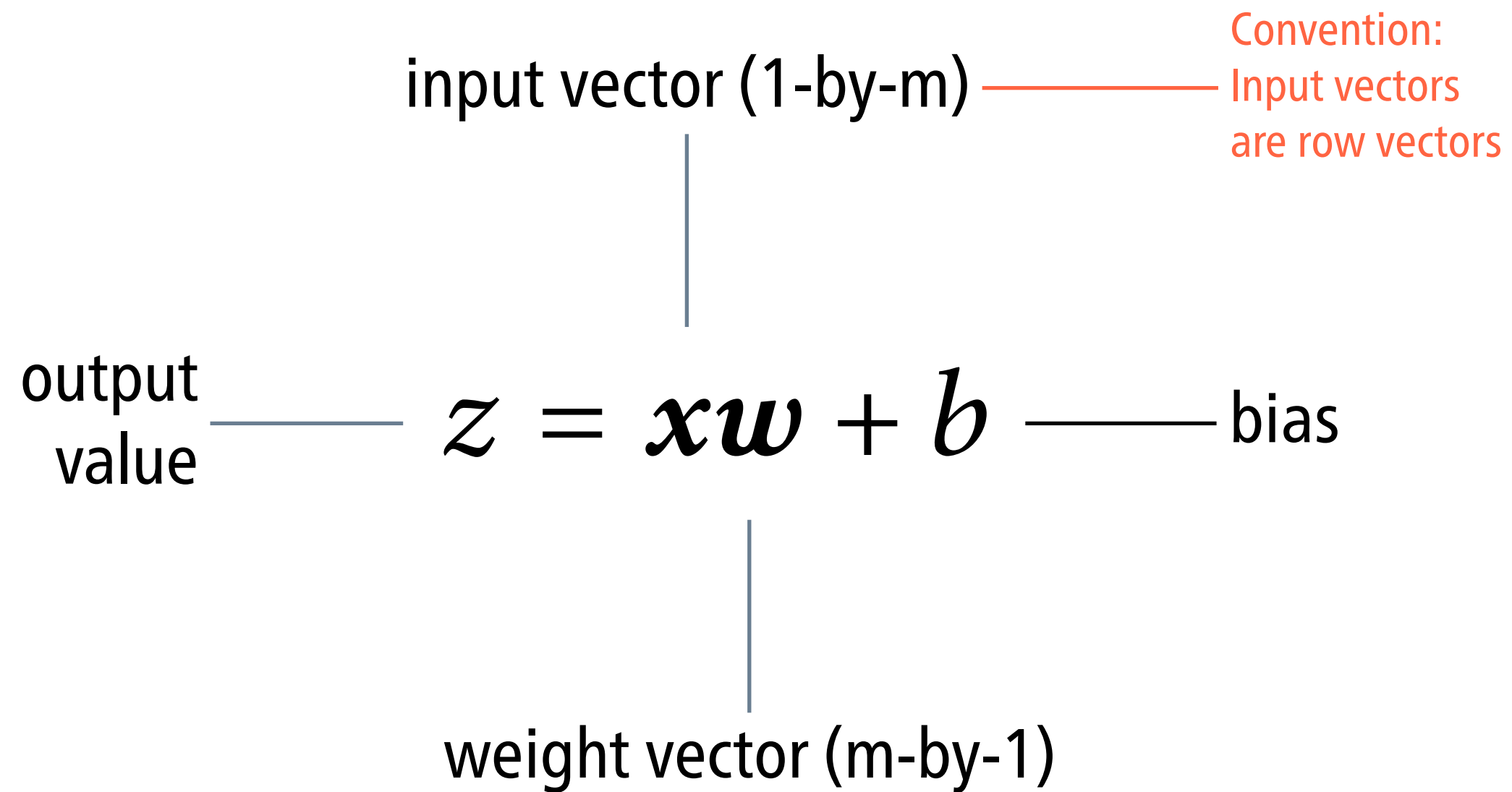
# Linear regression with one variable



# Linear regression with one variable

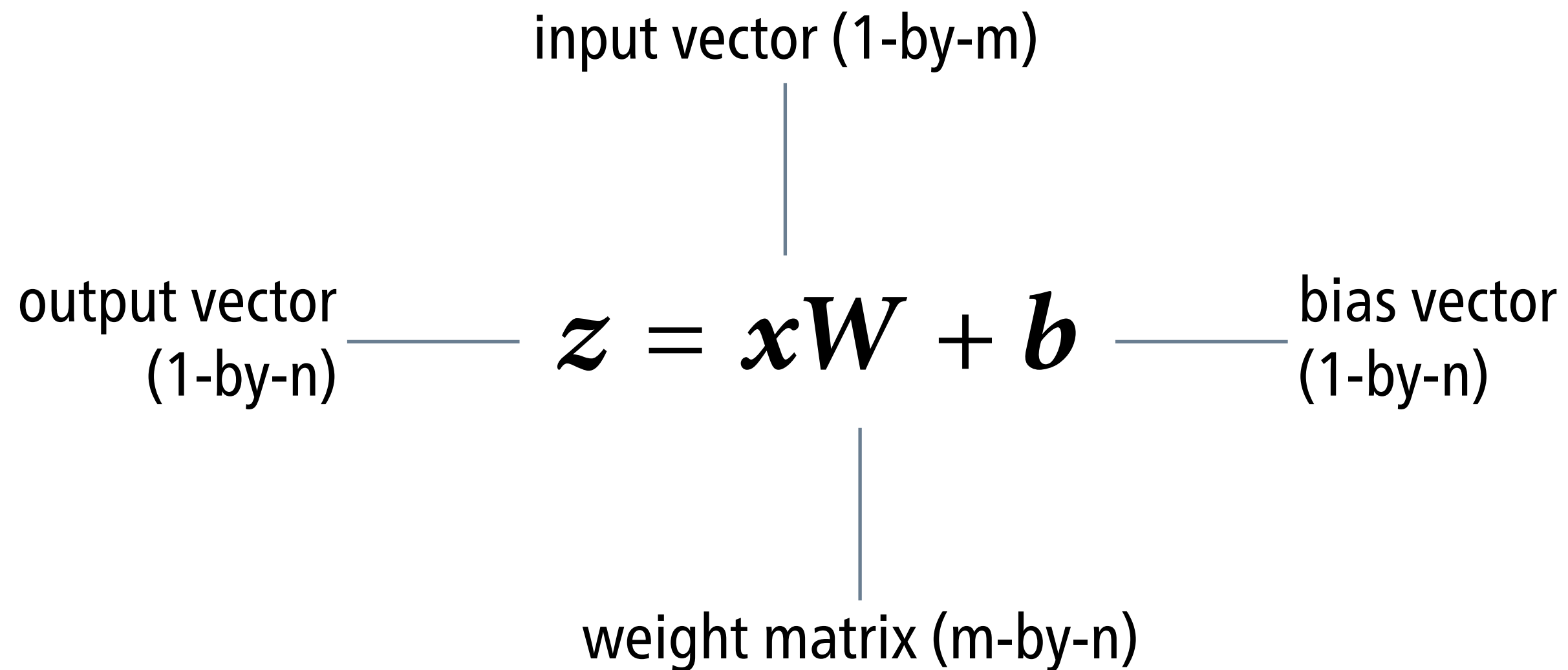


# The linear model



$m$  = number of features (independent variables)

# The linear model (multivariate version)



$m$  = number of input features,  $n$  = number of output features

# Linear classification

- We can think of  $\mathbf{z} = \mathbf{x}\mathbf{W} + \mathbf{b}$  as a vector of class-specific scores. The higher the score  $z[k]$ , the more likely  $\mathbf{x}$  belongs to class  $k$ .
- We can use these scores for classification: We predict the input  $\mathbf{x}$  to belong to the highest-scoring class  $k$ .
- With linear models, we can only solve a rather restricted class of classification problems (linearly separable).

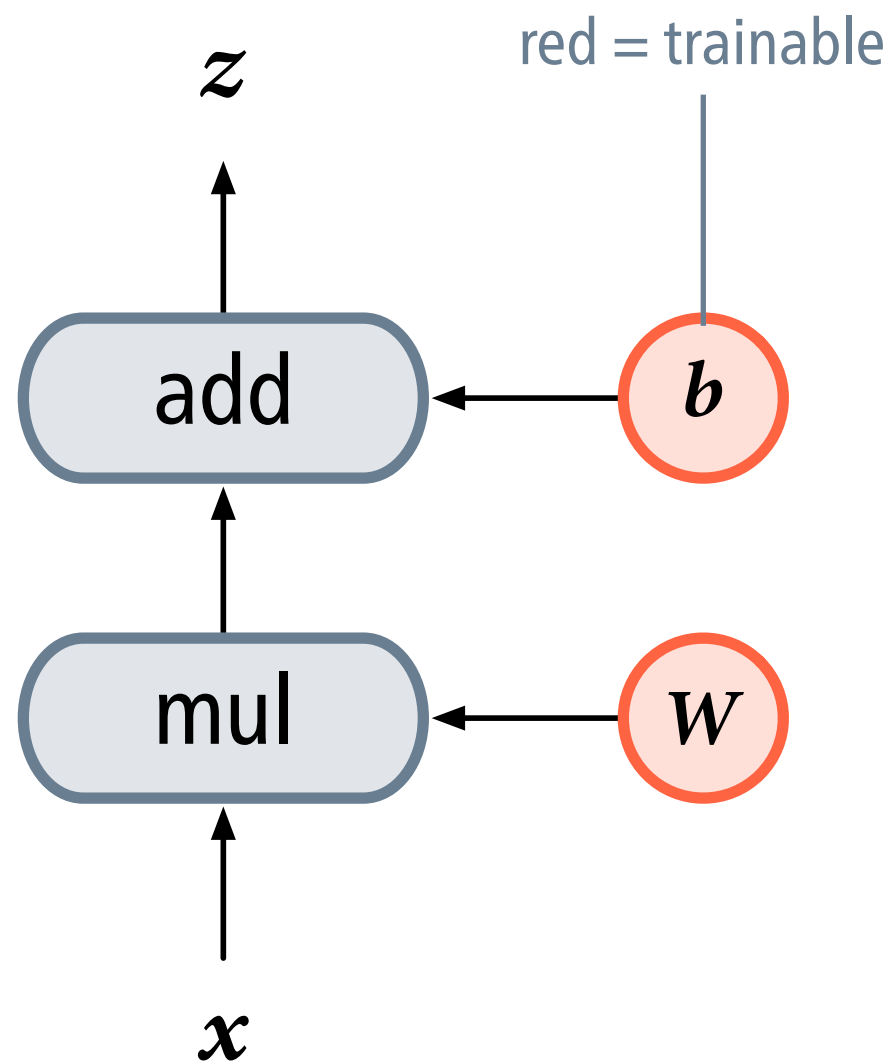
# Handwritten digit recognition

5 0 5 9 7 7  
6 8 0 6 3 6  
2 7 1 5 8 3

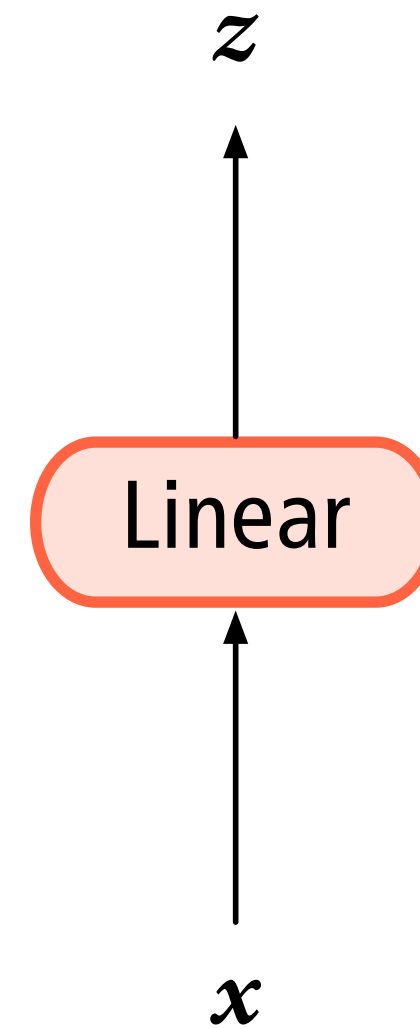
**Input:** an image of a digit, represented as a 784-dimensional vector of greyscale values.

**Predict:** the digit depicted in the image

# Graphical notation



computation graph



shorthand notation



# Linear models in PyTorch

```
>>> import torch
```

```
>>> # Create a linear model
```

```
>>> model = torch.nn.Linear(784, 10)
```

```
>>> # Inspect the shapes of the model parameters
```

```
>>> [p.shape for p in model.parameters()]  
[torch.Size([10, 784]), torch.Size([10])]
```

```
>>> # Feed random data and inspect the shape of the output
```

```
>>> model.forward(torch.rand(784)).shape  
torch.Size([10])
```

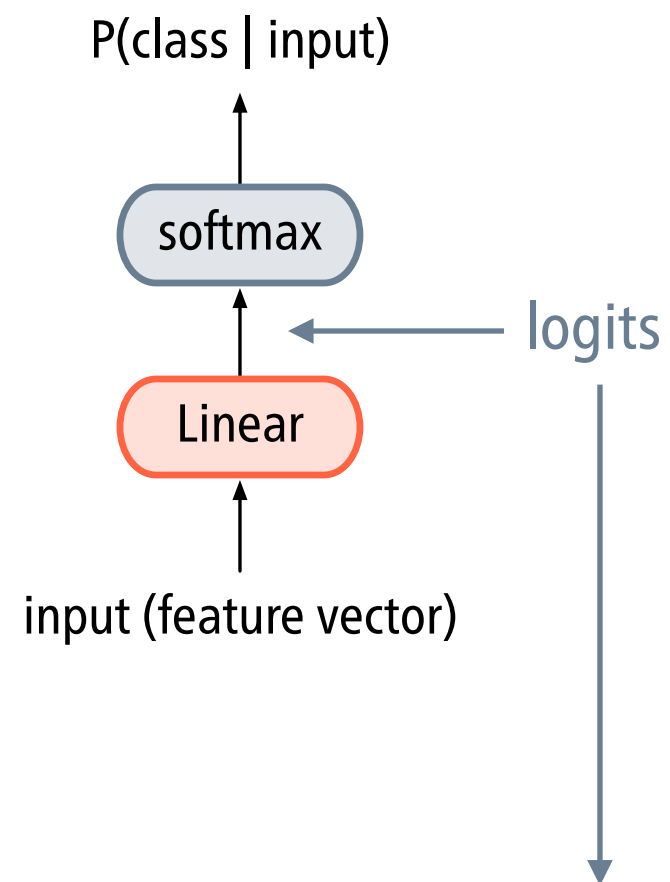
# The softmax function

- We can convert the scores into a probability distribution  $P(k | \mathbf{x})$  over the classes by sending them through the **softmax function**:

$$\text{softmax}(\mathbf{z})[k] = \frac{\exp(\mathbf{z}[k])}{\sum_i \exp(\mathbf{z}[i])}$$

- This normalises the scores to the interval  $[0, 1]$  but does not affect the relative ordering of the scores.
- In this context, the unnormalised (raw) scores are called **logits**.

# Linear layer + softmax function



$$P(k | \mathbf{x}) = \text{softmax}(\mathbf{x}\mathbf{W} + \mathbf{b})$$

# Training a linear model

- We present the model with training samples of the form  $(\mathbf{x}, y)$  where  $\mathbf{x}$  is a feature vector and  $y$  is the gold-standard class.
- The output of the model is a vector of conditional probabilities  $P(k | \mathbf{x})$  where  $k$  ranges over the possible classes.
- We want to train the model so as to maximise the likelihood of the training data under this probability distribution.

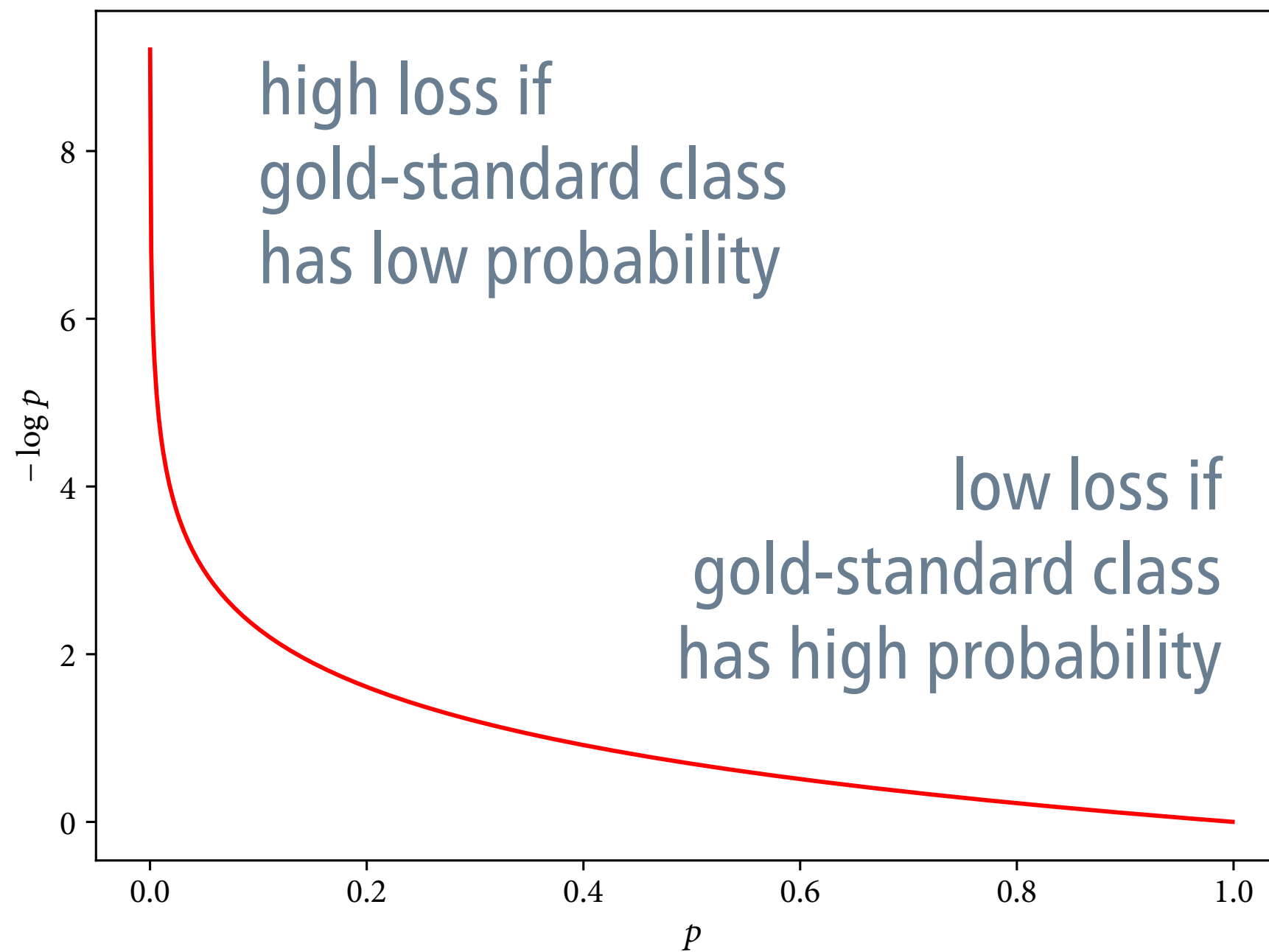
# Cross-entropy loss

- Instead of maximising the likelihood of the training data, we minimise the model's **cross-entropy loss**.
- The cross-entropy loss for a specific sample  $(\mathbf{x}, y)$  is the negative log probability of the gold-standard class  $y$ , in our case:

$$L(\boldsymbol{\theta}) = -\log \text{softmax}(\mathbf{x}\mathbf{W} + \mathbf{b})[y]$$

all trainable  
parameters

# Cross-entropy loss



# Gradient descent

“Follow the gradient into valleys of low error.”

- **Step 0:** Start with random values for the parameters  $\theta$ .
- **Step 1:** Compute the gradient of the loss function for the current parameter settings,  $\nabla L(\theta)$ .
- **Step 2:** Update the parameters  $\theta$  as follows:  $\theta := \theta - \alpha \nabla L(\theta)$   
The hyperparameter  $\alpha$  is the learning rate.
- Repeat step 1–2 until the loss is sufficiently low.