Natural Language Processing

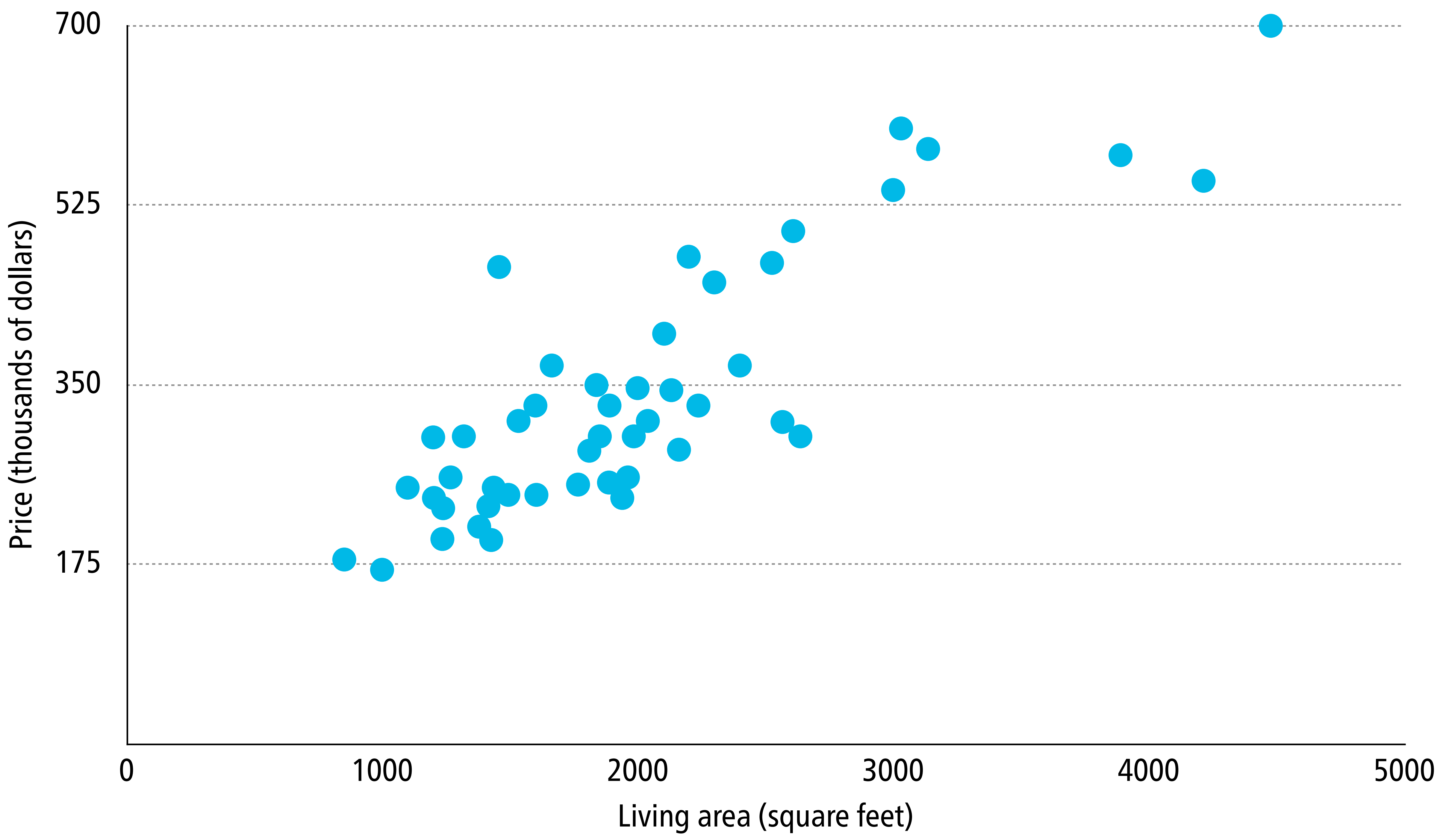
Linear neural networks

Marco Kuhlmann
Department of Computer and Information Science

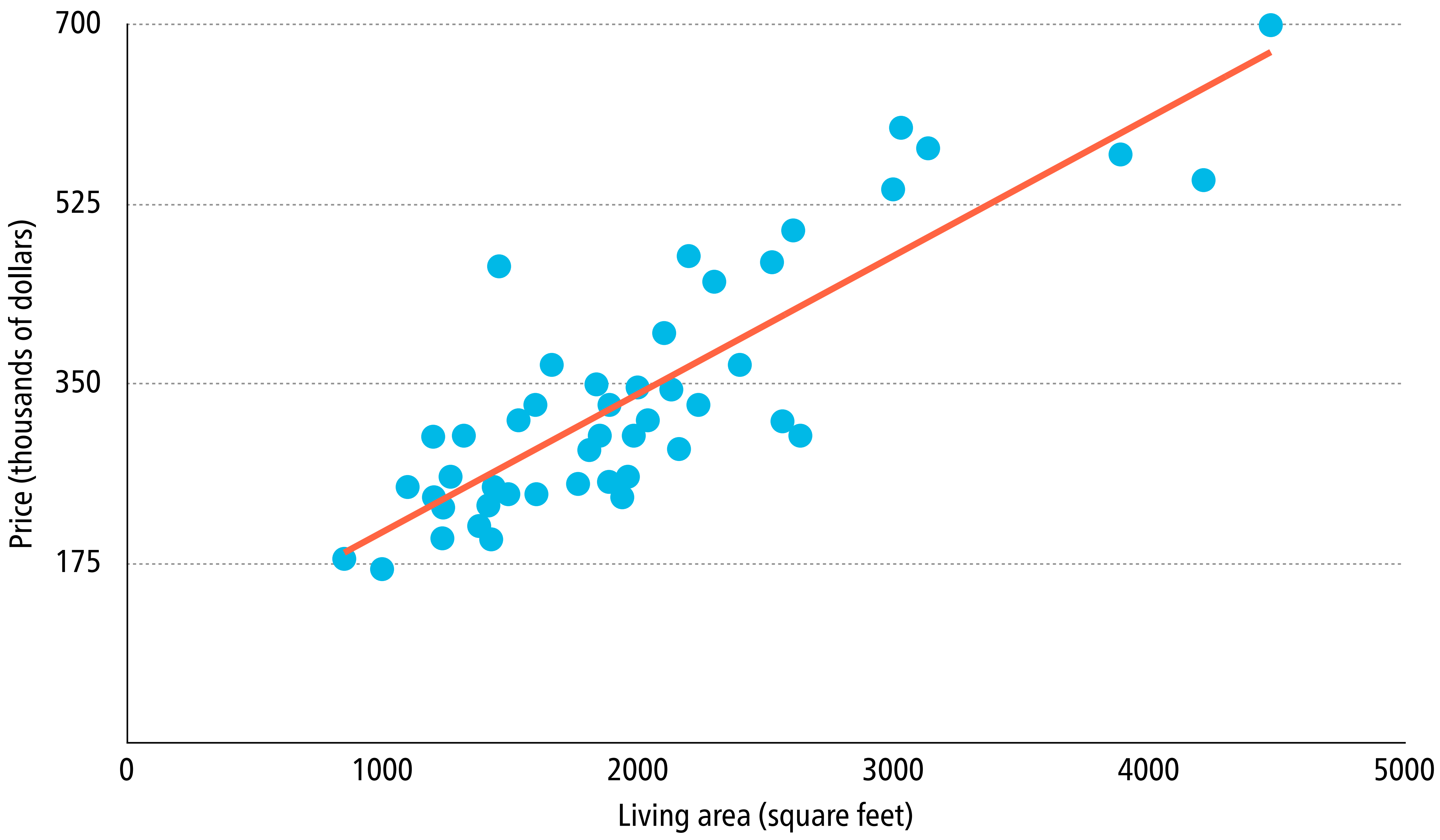
LINKÖPING UNIVERSITY

This work is licensed under a
Creative Commons Attribution 4.0 International License.

Linear regression with one variable



Linear regression with one variable



The linear model

input vector (1-by-m)

Convention:
Input vectors
are row vectors

output
value

$$z = \boldsymbol{x}\boldsymbol{w} + b$$

bias

weight vector (m-by-1)

m = number of features (independent variables)

The linear model (multivariate version)

input vector (1-by-m)

output vector (1-by-n)

$$\boldsymbol{z} = \boldsymbol{x}\boldsymbol{W} + \boldsymbol{b}$$

bias vector (1-by-n)

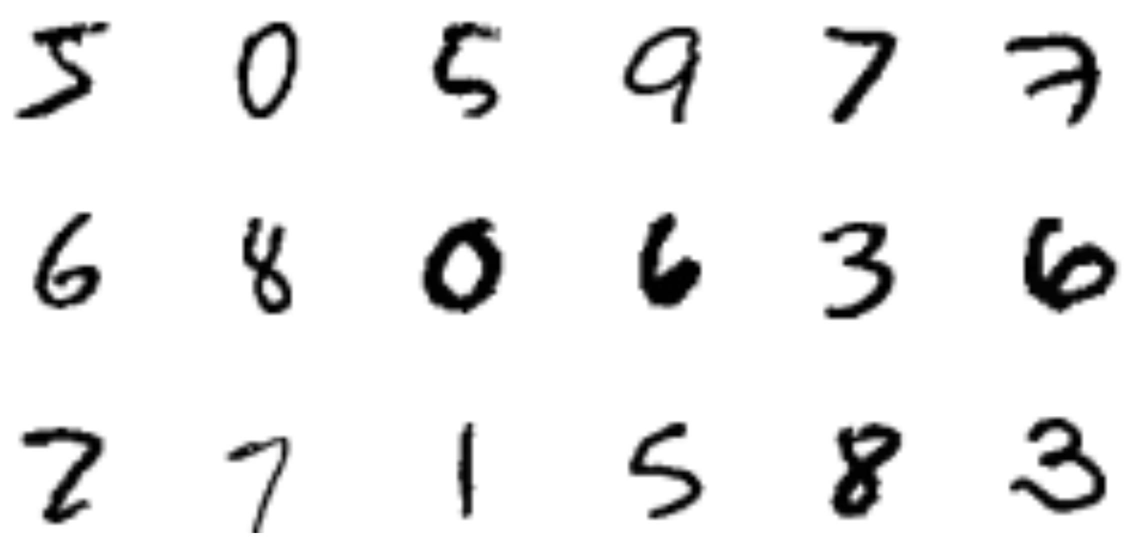
weight matrix (m-by-n)

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

Linear classification

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

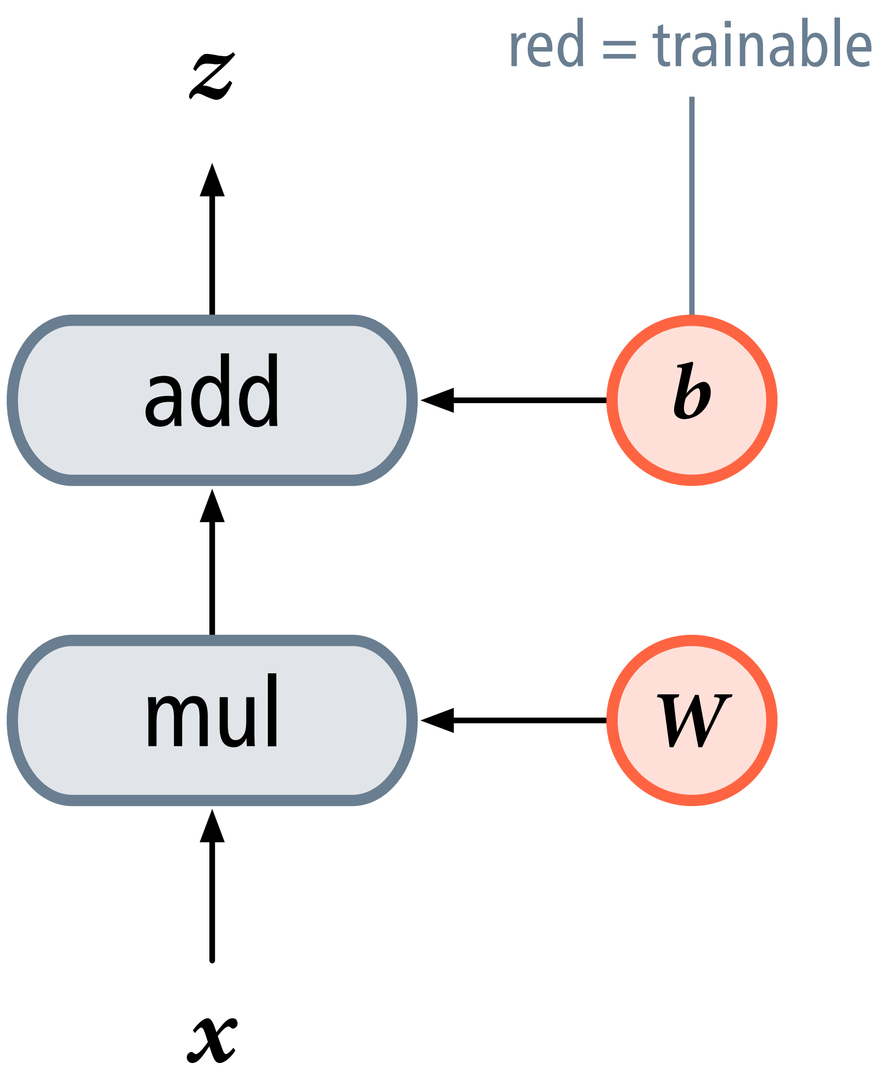
Handwritten digit recognition



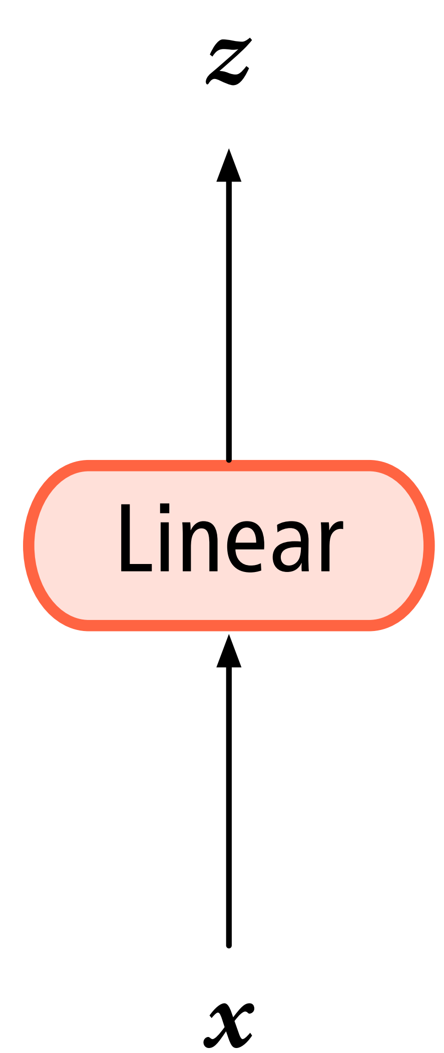
Input: an image of a digit, represented as a 784-dimensional vector of greyscale values.

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 $p(k \mid \boldsymbol{x})$ over the classes by sending them through the **softmax function**:

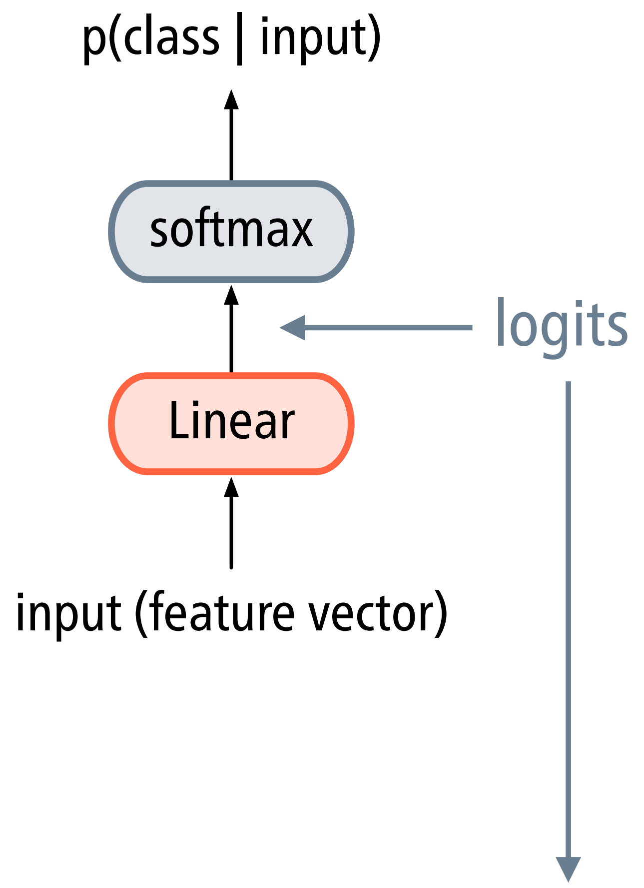
class index

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

- 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



$$p(k \mid \boldsymbol{x}) = \text{softmax}(\boldsymbol{x}\boldsymbol{W} + \boldsymbol{b})$$

Training a linear model

- We present the model with training samples of the form $(\boldsymbol{x}, y)$ where $\boldsymbol{x}$ is a feature vector and y is the gold-standard class.
- The output of the model is a vector of conditional probabilities $p(k \mid \boldsymbol{x})$ 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

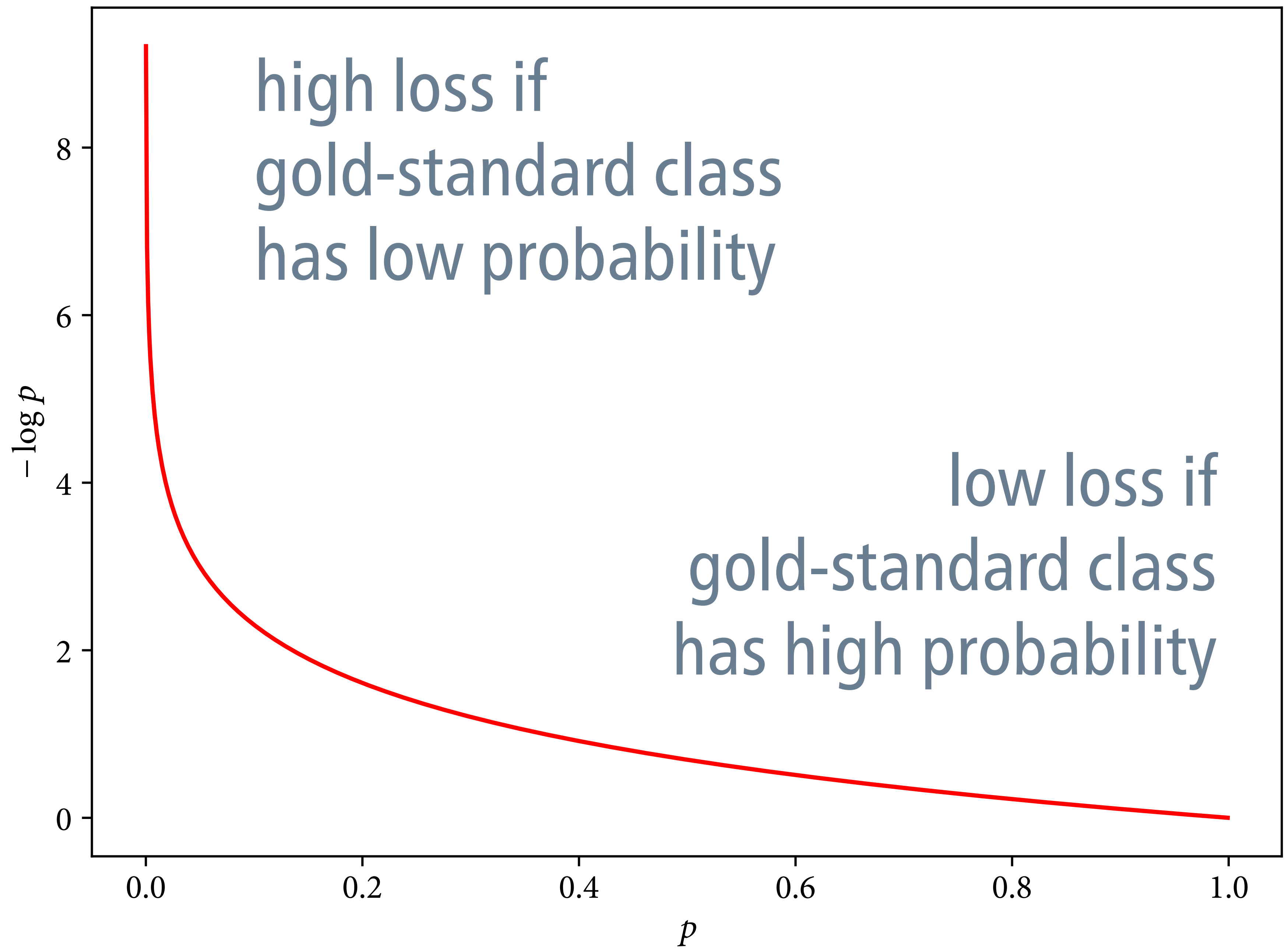
- Instead of maximising the likelihood of the training data, we minimise the model's **cross-entropy loss**.

- The cross-entropy loss for a specific sample $(\boldsymbol{x}, y)$ is the negative log probability of the gold-standard class y, in our case:

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

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 $\boldsymbol{\theta}$.
- **Step 1:** Compute the gradient of the loss function for the current parameter settings, $\nabla L(\boldsymbol{\theta})$.
- **Step 2:** Update the parameters $\boldsymbol{\theta}$ as follows: $\boldsymbol{\theta} \coloneqq \boldsymbol{\theta} - \alpha \nabla L(\boldsymbol{\theta})$

  The hyperparameter α is the learning rate.
- Repeat step 1–2 until the loss is sufficiently low.