Language Technology

Chapters 7 and 8: Linear Regression, Logistic Regression, and Neural Networks

https://link.springer.com/chapter/10.1007/978-3-031-57549-5_7
https://link.springer.com/chapter/10.1007/978-3-031-57549-5_8

Pierre Nugues

Pierre.Nugues@cs.lth.se

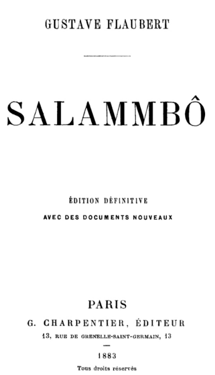
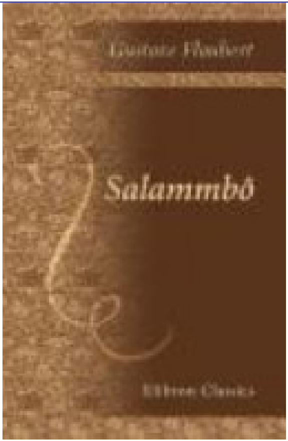
September 18, 2025

Some Definitions

1. Machine learning always starts with **data sets**: a collection of objects or observations.
2. Machine-learning algorithms can be classified along two main lines: **supervised** and **unsupervised** classification.
3. Supervised algorithms need a **training set**, where the objects are described in terms of attributes and belong to a known class or have a known output.
4. The performance of the resulting classifier is measured against a **test set**.
5. We can also use N-fold cross validation, where the test set is selected randomly from the training set N times, usually 10.
6. Unsupervised algorithms consider objects, where no class is provided.
7. Unsupervised algorithms learn regularities in data sets.

A Dataset: *Salammbô*

A corpus is a collection – a body – of texts.

French original	English translation
	

GUSTAVE FLAUBERT

SALAMMBÔ

ÉDITION DÉFINITIVE
AVEC DES DOCUMENTS NOUVEAUX

PARIS
G. CHARPENTIER, ÉDITEUR
13, RUE DE GRENELLE-SAINT-GERMAIN, 13

1883
Tous droits réservés

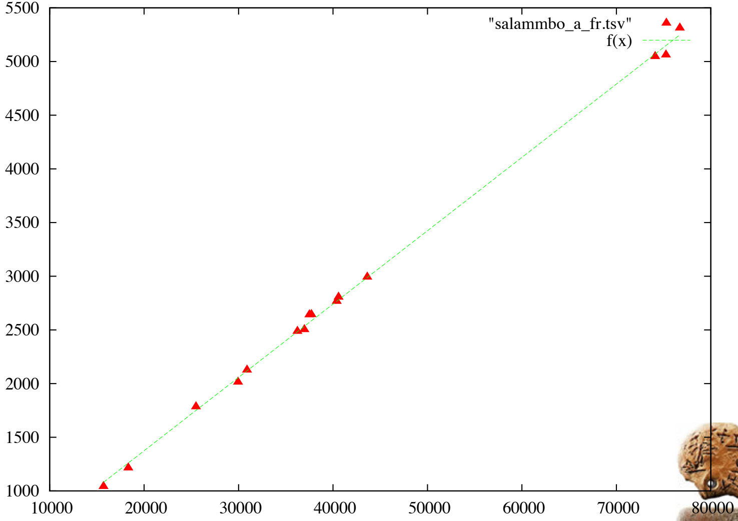
Supervised Learning

Letter counts from *Salammbô*

Chapter	French		English	
	# characters	# *A*	# characters	# *A*
Chapter 1	36,961	2,503	35,680	2,217
Chapter 2	43,621	2,992	42,514	2,761
Chapter 3	15,694	1,042	15,162	990
Chapter 4	36,231	2,487	35,298	2,274
Chapter 5	29,945	2,014	29,800	1,865
Chapter 6	40,588	2,805	40,255	2,606
Chapter 7	75,255	5,062	74,532	4,805
Chapter 8	37,709	2,643	37,464	2,396
Chapter 9	30,899	2,126	31,030	1,993
Chapter 10	25,486	1,784	24,843	1,627
Chapter 11	37,497	2,641	36,172	2,375
Chapter 12	40,398	2,766	39,552	2,560
Chapter 13	74,105	5,047	72,545	4,597
Chapter 14	76,725	5,312	75,352	4,871
Chapter 15	18,317	1,215	18,031	1,119

Supervised Learning: Regression

Letter count from *Salammbô* in French



Models

We will assume that data sets are governed by functions or models.
For instance given the set:

$$\{(\mathbf{x}_i, y_i)|0 < i \leqslant N\},$$

there exists a function such that:

$$f(\mathbf{x}_i) = y_i.$$

Supervised machine learning algorithms will produce hypothesized functions or models fitting the data.

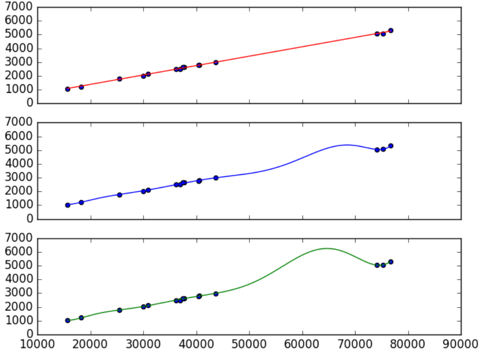
Notations

We will follow these notations:

- $\mathbf{x}$, the vector representing an observation (or sample, or example, or input);
  in *Salammbô*, an observation is the number of letters in a chapter. We have 15 observations;
- y, the observed response (or target, or output); in programs, the variable names are `y` or `y_true`;
  in *Salammbô*, the number of *A*s in a chapter. We have 15 responses;
- $\hat{y}$, the value predicted by the model; in programs, the variable names are `y_pred` or `y_hat`;
- $\mathbf{w}$, the weights or parameters of the model, so that $\mathbf{w} \cdot \mathbf{x} = \hat{y}$;
  another possible notation for $\mathbf{w}$ is $\boldsymbol{\beta}$
- X, the matrix of all the observations
- $\mathbf{y}$, the vector of all the responses and $\hat{\mathbf{y}}$, for all the predictions

Selecting a Model

Often, multiple models can fit a data set:
Three polynomials of degree: 1, a straight line, 8, and 9 to fit the *Salammbô* dataset.



A general rule in machine learning is to prefer the simplest hypotheses, here the lower polynomial degrees.
Otherwise, the model can **overfit** the data.
In our case, the optimal model $\mathbf{w}$ has two parameters: (w_0, w_1).

Loss or Objective Function

What are the optimal values of $\mathbf{w}$?
The model should minimize the difference between:

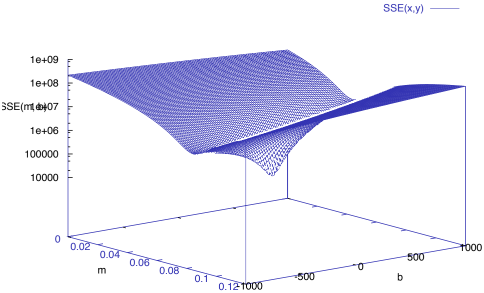
- the predicted values $\hat{\mathbf{y}}$ and
- the observed values $\mathbf{y}$.

This is called the **loss**
For *Salammbô*, the loss is the *mean of the squared errors* (MSE):

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Visualizing the Loss

$$\hat{y} = mx + b$$



We will use the notation

$$\hat{y} = w_1 x + w_0$$

to generalize to any dimension

The Matrices

$$X = \begin{bmatrix} 1 & 36961 \\ 1 & 43621 \\ 1 & 15694 \\ 1 & 36231 \\ 1 & 29945 \\ 1 & 40588 \\ 1 & 75255 \\ 1 & 37709 \\ 1 & 30899 \\ 1 & 25486 \\ 1 & 37497 \\ 1 & 40398 \\ 1 & 74105 \\ 1 & 76725 \\ 1 & 18317 \end{bmatrix}; \mathbf{w} = \begin{bmatrix} 8.7253 \\ 0.0683 \end{bmatrix}; \hat{\mathbf{y}} = \begin{bmatrix} 2533.22 \\ 2988.11 \\ 1080.65 \\ 2483.36 \\ 2054.02 \\ 2780.95 \\ 5148.76 \\ 2584.31 \\ 2119.18 \\ 1749.46 \\ 2569.83 \\ 2767.97 \\ 5070.21 \\ 5249.16 \\ 1259.81 \end{bmatrix}; \mathbf{y} = \begin{bmatrix} 2503 \\ 2992 \\ 1042 \\ 2487 \\ 2014 \\ 2805 \\ 5062 \\ 2643 \\ 2126 \\ 1784 \\ 2641 \\ 2766 \\ 5047 \\ 5312 \\ 1215 \end{bmatrix}; \mathbf{se} = \begin{bmatrix} 913.26 \\ 15.14 \\ 1493.86 \\ 13.25 \\ 1601.31 \\ 578.40 \\ 7527.51 \\ 3444.53 \\ 46.57 \\ 1193.04 \\ 5065.18 \\ 38920 \\ 538.909 \\ 3948.29 \\ 2007.53 \end{bmatrix}.$$

Minimizing the Loss

The loss function is convex and has a unique minimum.
The loss reaches a minimum when the partial derivatives are zero:

$$
\begin{aligned}
\frac{\partial Loss}{\partial m} &= \sum_{i=1}^{q} \frac{\partial}{\partial m}(y_i - (mx_i + b))^2 &= -2\sum_{i=1}^{q} x_i(y_i - (mx_i + b)) &= 0 \\
\frac{\partial Loss}{\partial b} &= \sum_{i=1}^{q} \frac{\partial}{\partial b}(y_i - (mx_i + b))^2 &= -2\sum_{i=1}^{q} (y_i - (mx_i + b)) &= 0
\end{aligned}
$$

The Gradient Descent

The gradient descent is a numerical method to find the minimum of $f(w_0, w_1, w_2, ..., w_n) = y$, when there is no analytical solution.
Let us denote $\mathbf{w} = (w_0, w_1, w_2, ..., w_n)$
We derive successive approximations to find the minimum of f:

$$f(\mathbf{w}_1) > f(\mathbf{w}_2) > ... > f(\mathbf{w}_k) > f(\mathbf{w}_{k+1}) > .. > min$$

Points in the neighborhood of $\mathbf{w}$ are defined by $\mathbf{w} + \mathbf{v}$ with $||\mathbf{v}||$ small
Given $\mathbf{w}$, find $\mathbf{v}$ subject to $f(\mathbf{w}) > f(\mathbf{w} + \mathbf{v})$

The Gradient Descent (Cauchy, 1847)

Using a Taylor expansion: $f(\mathbf{w}+\mathbf{v}) = f(\mathbf{w}) + \mathbf{v}\cdot\nabla f(\mathbf{w}) + \ldots$

The gradient is a direction vector corresponding to the steepest slope:

$$\nabla f(w_0, w_1, w_2, \ldots, w_n) = (\frac{\partial f}{\partial w_0}, \frac{\partial f}{\partial w_1}, \frac{\partial f}{\partial w_2}, \ldots, \frac{\partial f}{\partial w_n}).$$

$f(\mathbf{w}+\mathbf{v})$ reaches a minimum or a maximum when $\mathbf{v}$ and $\nabla f(\mathbf{w})$ are colinear:

- Steepest ascent: $\mathbf{v} = \alpha \nabla f(\mathbf{w})$,
- Steepest descent: $\mathbf{v} = -\alpha \nabla f(\mathbf{w})$,

where $\alpha > 0$.

We have then: $f(\mathbf{w} - \alpha\nabla f(\mathbf{w})) \approx f(\mathbf{w}) - \alpha||\nabla f(\mathbf{w})||^2$.

The inequality:

$$f(\mathbf{w}) > f(\mathbf{w} - \alpha\nabla f(\mathbf{w}))$$

enables us to move one step down to the minimum.

We then use the iteration:

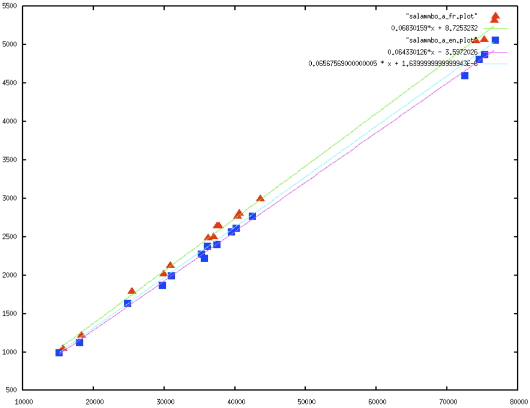
$$\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha_k \nabla f(\mathbf{w}_k).$$

Classification Dataset

A binary classification.

	# char.	# A	class (y)	# char.	# A	class (y)
Chapter 1	36,961	2,503	1	35,680	2,217	0
Chapter 2	43,621	2,992	1	42,514	2,761	0
Chapter 3	15,694	1,042	1	15,162	990	0
Chapter 4	36,231	2,487	1	35,298	2,274	0
Chapter 5	29,945	2,014	1	29,800	1,865	0
Chapter 6	40,588	2,805	1	40,255	2,606	0
Chapter 7	75,255	5,062	1	74,532	4,805	0
Chapter 8	37,709	2,643	1	37,464	2,396	0
Chapter 9	30,899	2,126	1	31,030	1,993	0
Chapter 10	25,486	1,784	1	24,843	1,627	0
Chapter 11	37,497	2,641	1	36,172	2,375	0
Chapter 12	40,398	2,766	1	39,552	2,560	0
Chapter 13	74,105	5,047	1	72,545	4,597	0
Chapter 14	76,725	5,312	1	75,352	4,871	0
Chapter 15	18,317	1,215	1	18,031	1,119	0

Separating Classes



Given the data set, $\{(\mathbf{x}_i, y_i) | 0 < i \leqslant N\}$ and a model f:

- Classification: $f(\mathbf{x}) = y$ is discrete,
- Regression: $f(\mathbf{x}) = y$ is continuous.

Supervised Machine-Learning Algorithms

Linear classifiers:

1. Perceptron
2. Logistic regression
3. Neural networks (with many flavors)

Classification

We represent classification using a threshold function (a variant of the signum function):

$$H(\mathbf{w} \cdot \mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

The classification function associates P with 1 and N with 0.
We want to find the separating hyperplane:

$$\begin{aligned} \hat{y}(\mathbf{x}) &= H(\mathbf{w} \cdot \mathbf{x}) \\ &= H(w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n), \end{aligned}$$

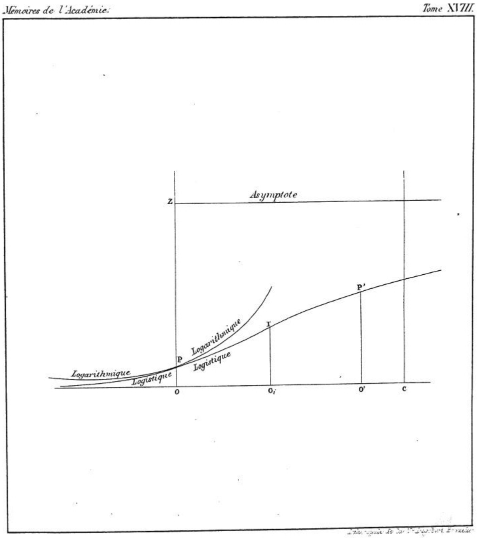
given a data set of q examples: $DS = \{(1, x_1^j, x_2^j, \ldots, x_n^j, y^j) | j : 1..q\}$.
We use $x_0 = 1$ to simplify the equations.
For a binary classifier, y has then two possible values $\{0, 1\}$ corresponding in our example to $\{\text{French}, \text{English}\}$.

Berkson's Data Set (1944)

Drug concentration	Number exposed	Survive Class 0	Die Class 1	Mortality rate	Expected mortality
40	462	352	110	.2359	.2206
60	500	301	199	.3980	.4339
80	467	169	298	.6380	.6085
100	515	145	370	.7184	.7291
120	561	102	459	.8182	.8081
140	469	69	400	.8529	.8601
160	550	55	495	.9000	.8952
180	542	43	499	.9207	.9195
200	479	29	450	.9395	.9366
250	497	21	476	.9577	.9624
300	453	11	442	.9757	.9756

Table: A data set. Adapted and simplified from the original article that described how to apply logistic regression to classification by Joseph Berkson, Application of the Logistic Function to Bio-Assay. *Journal of the American Statistical Association* (1944).

Another Model, the Logistic Curve (Verhulst)



Mémoire sur la population par M. P. Verhulst.

$$Logistic(x) = \frac{1}{1+e^{-x}}$$

$$\hat{y}(\mathbf{x}) = Logistic(\mathbf{w}\cdot\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}\cdot\mathbf{x}}}$$

Loss: Binary Cross Entropy

In practice, we use the mean and the natural logarithm:

$$H(P, M) = -\frac{1}{|X|} \sum_{x \in X} P(x) \log M(x),$$

where P is the truth, and M is the prediction of the model, a probability in the case of logistic regression.

In binary classification:

- $P(x) = 1$
- $M(x)$ is the predicted probability of being class 1.
- If the observation belongs to class 0, its predicted probability is $1 - M(x)$.

Example of Cross Entropy

Computing the cross-entropy of six observations:

Observations	1	2	3	4	5	6
Dose	140	300	140	160	140	250
Observed class (Truth)	0	1	1	1	1	1
Model prediction of being class 1	0.3487	0.9964	0.8557	0.9056	0.8557	0.9882
Model prediction of being class 0	0.6513					
$-P(x) \log M(x)$:	0.4287	0.0036	0.1559	0.0992	0.1559	0.0119

Mean = 0.14252826

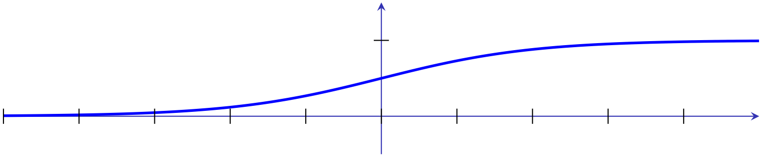
Code Example: Logistic Regression with sklearn

Experiment: Jupyter Notebook:
`https://github.com/pnugues/pnlp/tree/main/notebooks`

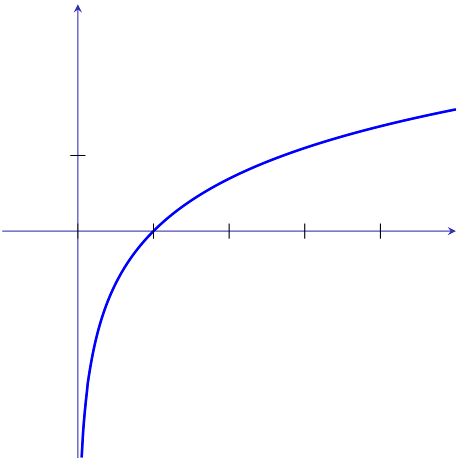
Logistic Curve

The logistic function:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Logarithm Function

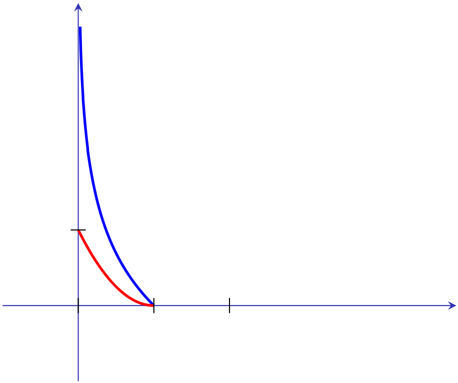


Loss

The logistic loss, in blue, is defined as: $L(\hat{y}) = -\log \hat{y}$, where

$$\hat{y} = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}.$$

here compared with the squared error loss in red.



PyTorch Loop

```
model.train()
for epoch in range(250):
    y_pred = model(X) # We compute Xw = y_hat
    loss = loss_fn(y_pred, y) # (h_hat - y)^2
    optimizer.zero_grad()
    loss.backward() # we compute the gradients
    optimizer.step() # we update the weights
```

Updates

To carry out an update, the optimizer uses:

- The whole dataset: batch gradient descent
- One observation: stochastic gradient descent
- A few observations: mini-batch gradient descent

Computing the Binary Crossentropy

For binary classification, we use binary cross entropy.
One strategy is to apply the logistic function to the dot product:

$$
\begin{aligned}
\hat{y} &= \sigma(\mathbf{w} \cdot \mathbf{x}), \\
&= \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}.
\end{aligned}
$$

And then compute the loss: `https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html`

Binary Crossentropy from Logits

Another option is to compute the dot product:

$$\mathbf{w} \cdot \mathbf{x}$$

and then incorporate the logistic function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

in the loss.

To have a stable binary cross entropy, we use this option:

1. `https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html`

   This loss combines a Sigmoid layer and the BCELoss in one single class. This version is more numerically stable than using a plain Sigmoid followed by a BCELoss as, by combining the operations into one layer, we take advantage of the log-sum-exp trick for numerical stability.

Softmax Dangers

Softmax can be unstable even with relatively small numbers.
For instance, it is easy to compute:

$$\begin{aligned}\sigma(1000, 1000) &= \left(\frac{e^{1000}}{e^{1000}+e^{1000}}, \frac{e^{1000}}{e^{1000}+e^{1000}}\right)\\ &= \left(\frac{1}{2}, \frac{1}{2}\right)\end{aligned}$$

but in Python:

```
>>> math.exp(1000)
...
OverflowError: math range error
```

The log-sum-exp trick will factor terms and make the computation possible
For a description, see:
https://gregorygundersen.com/blog/2020/02/09/log-sum-exp/

Code Example: Binary Classification with PyTorch

For binary classification, we use binary cross entropy

Experiment 1, PyTorch: Jupyter Notebook:

`https://github.com/pnugues/pnlp/tree/main/notebooks`

More than two Classes: Types of Iris

Iris virginica

Iris setosa

Iris versicolor

Courtesy Wikipedia

Supervised Learning: Fisher's Iris data set (1936)

180 MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS

Table I

Iris setosa				*Iris versicolor*				*Iris virginica*			
Sepal length	Sepal width	Petal length	Petal width	Sepal length	Sepal width	Petal length	Petal width	Sepal length	Sepal width	Petal length	Petal width
5·1	3·5	1·4	0·2	7·0	3·2	4·7	1·4	6·3	3·3	6·0	2·5
4·9	3·0	1·4	0·2	6·4	3·2	4·5	1·5	5·8	2·7	5·1	1·9
4·7	3·2	1·3	0·2	6·9	3·1	4·9	1·5	7·1	3·0	5·9	2·1
4·6	3·1	1·5	0·2	5·5	2·3	4·0	1·3	6·3	2·9	5·6	1·8
5·0	3·6	1·4	0·2	6·5	2·8	4·6	1·5	6·5	3·0	5·8	2·2
5·4	3·9	1·7	0·4	5·7	2·8	4·5	1·3	7·6	3·0	6·6	2·1
4·6	3·4	1·4	0·3	6·3	3·3	4·7	1·6	4·9	2·5	4·5	1·7
5·0	3·4	1·5	0·2	4·9	2·4	3·3	1·0	7·3	2·9	6·3	1·8
4·4	2·9	1·4	0·2	6·6	2·9	4·6	1·3	6·7	2·5	5·8	1·8
4·9	3·1	1·5	0·1	5·2	2·7	3·9	1·4	7·2	3·6	6·1	2·5
5·4	3·7	1·5	0·2	5·0	2·0	3·5	1·0	6·5	3·2	5·1	2·0
4·8	3·4	1·6	0·2	5·9	3·0	4·2	1·5	6·4	2·7	5·3	1·9
4·8	3·0	1·4	0·1	6·0	2·2	4·0	1·0	6·8	3·0	5·5	2·1
4·3	3·0	1·1	0·1	6·1	2·9	4·7	1·4	5·7	2·5	5·0	2·0
5·8	4·0	1·2	0·2	5·6	2·9	3·6	1·3	5·8	2·8	5·1	2·4
5·7	4·4	1·5	0·4	6·7	3·1	4·4	1·4	6·4	3·2	5·3	2·3
5·4	3·9	1·3	0·4	5·6	3·0	4·5	1·5	6·5	3·0	5·5	1·8
5·1	3·5	1·4	0·3	5·8	2·7	4·1	1·0	7·7	3·8	6·7	2·2
5·7	3·8	1·7	0·3	6·2	2·2	4·5	1·5	7·7	2·6	6·9	2·3
5·1	3·8	1·5	0·3	5·6	2·5	3·9	1·1	6·0	2·2	5·0	1·5
5·4	3·4	1·7	0·2	5·9	3·2	4·8	1·8	6·9	3·2	5·7	2·3
5·1	3·7	1·5	0·4	6·1	2·8	4·0	1·3	5·6	2·8	4·9	2·0

Multiple Categories

We can generalize logistic regression to multiple categories.
We use then the *softmax* function:

$$P(y = i|\mathbf{x}) = \frac{e^{\mathbf{w}_i \cdot \mathbf{x}}}{\sum_{j=1}^{C} e^{\mathbf{w}_j \cdot \mathbf{x}}},$$

that defines the probability of an observation represented by $\mathbf{x}$ to belong to class i.
Again, we use stochastic gradient descent to compute the weights: $\mathbf{w}$.
Note: In physics, *softmax* is defined as:

$$P(y = i|\mathbf{x}) = \frac{e^{-\mathbf{w}_i \cdot \mathbf{x}}}{\sum_{j=1}^{C} e^{-\mathbf{w}_j \cdot \mathbf{x}}},$$

See the Boltzmann distribution
`https://en.wikipedia.org/wiki/Boltzmann_distribution`

Representing the True and Predicted Distributions

As notation, we use y for the true value and $\hat{y}$ for the prediction.
Example with three classes:

Truth

```
> [[0, 0, 1],
   [0, 1, 0],
   [0, 0, 1],
   [1, 0, 0]]
```

Prediction

```
> [[2.7991e-03, 1.2680e-10, 9.9720e-01],
   [9.4966e-02, 8.9969e-01, 5.3394e-03],
   [7.4423e-01, 1.7318e-03, 2.5404e-01],
   [7.9428e-01, 7.0174e-03, 1.9870e-01]]
```

In PyTorch, the default representation of y and $\hat{y}$ are vectors (as opposed to sklearn)
y is a tensor of indices

```
y[:4]
> tensor([2, 1, 2, 0])
```

and $\hat{y}$, is a probability distribution

Logits

- In the context of neural networks, logits are the last matrix output.
- With just one layer, this would be:

$$W\mathbf{x}$$

- The logit function is in fact the inverse of the logistic function (`https://en.wikipedia.org/wiki/Logit`).
- Once we have applied a softmax or logistic function, we obtain $\hat{y}$, a probability distribution
- Note that this a bit weird: The activation function would be:

$$\sin(x)$$

We would call the equivalent

$$\arcsin(x)$$

Logistic Loss

1. For one observation, logistic regression yields the predicted probabilities of the classes
2. The logistic loss is defined as the opposite of the logarithm of predicted probability of the true class.
3. For a dataset, it can be reformulated as a cross entropy:

$$-\frac{1}{N}\sum_{i=1}^{N} \mathbf{y}_i \cdot \log \hat{\mathbf{y}}_i,$$

where $\mathbf{y}_i$ is a one-hot vector giving the position of the true value:

$$\mathbf{y}_i = (0, 0, ..., 0, \mathbf{1}, 0, ...0)$$

and $\hat{\mathbf{y}}_i$ the vector of estimated probabilities of the observations for all the classes

$$\hat{\mathbf{y}}_i = (0.01, 0.005, ..., \mathbf{0.70}, 0.10, ...0.001)$$

4. For this term: $-1 \times \log 0.7 = 0.36$

A complete example

The original categories:

```
y[:4]
[2, 1, 2, 0]
```

The logits:

```
model(X[:4])
[[  3.9234, -12.9865,   9.7991],
[  0.1311,   2.3797,  -2.7473],
[  2.3955,  -3.6677,   1.3206],
[  2.0336,  -2.6955,   0.6480]]
```

The predicted probabilities:

```
[[2.7991e-03, 1.2680e-10, 9.9720e-01],
[9.4966e-02, 8.9969e-01, 5.3394e-03],
[7.4423e-01, 1.7318e-03, 2.5404e-01],
[7.9428e-01, 7.0174e-03, 1.9870e-01]]
```

The predicted classes:

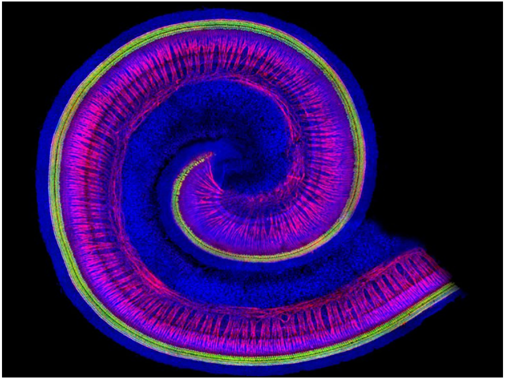
```
torch.argmax(model(X[:4]), dim=-1)
 [2, 1, 0, 0]
```

The loss:

Probability of the truth.

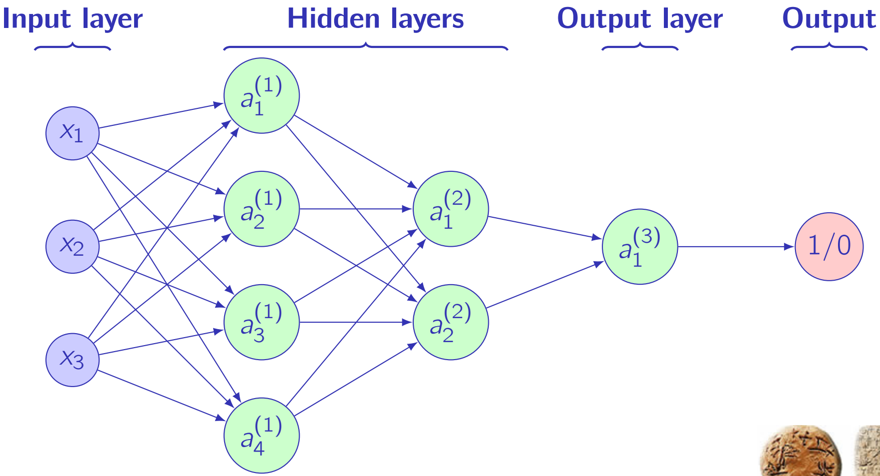
$$-\frac{1}{N}(\log(9.9720 \cdot 10^{-1}) + \log(8.9969 \cdot 10^{-1}) + \log(2.5404 \cdot 10^{-1}) + \log(7.9428 \cdot 10^{-1}))$$

Neural Networks



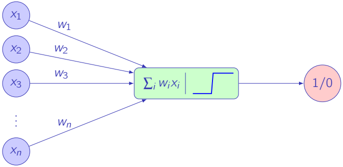
A photomicrograph showing the classic view of the snail-shaped cochlea with hair cells stained green and neurons showing reddish-purple. [Decibel Therapeutics (`https://www.decibeltx.com`)]. Source: `https://www.genengnews.com/insights/targeting-the-inner-ear/`

Neural Networks: Computer Model

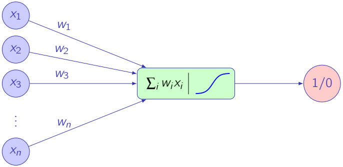


Activation Functions

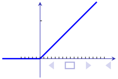
Heaviside (perceptron)



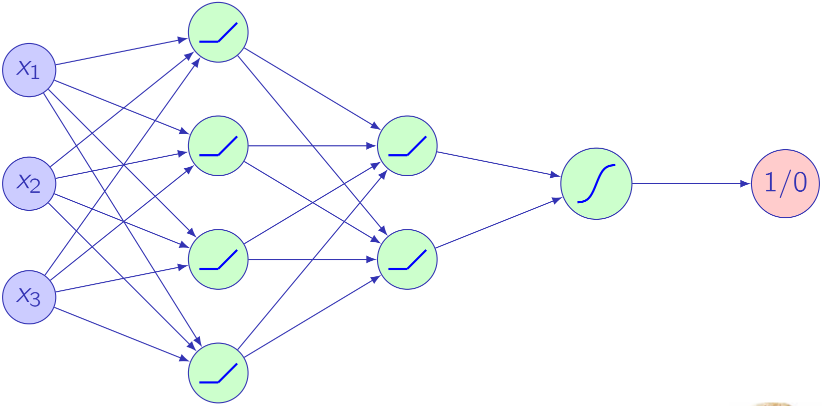
Logistic function (logistic regression)



Rectified linear unit (ReLU) (hidden layers)



Neural Networks with Hidden Layers



Code Example: Multinomial Classification with PyTorch

For multinomial (multiclass) classification, we use (categorical) cross entropy

Experiment 2, PyTorch: Jupyter Notebook:
`https://github.com/pnugues/pnlp/tree/main/notebooks`

Note that the target is represented by a list of integers, i.e. the list of classes. This can be confusing as it is different from BCE. See here:
`https://github.com/pytorch/pytorch/issues/56542`

Last Activation Layer in PyTorch

There is a discrepancy in the activation of the last layer in PyTorch between the binary and multiclass outputs:

1. `https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html`
2. `https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html`

`BCELoss` uses the mathematical definition, while `CrossEntropyLoss` is a composition of a softmax function and a cross-entropy.

Model Selection

- Validation can apply to one classification method
- We can use it to select a classification method and its parametrization.
- Needs three sets: training set, development set, and test set.

Evaluation

There are different kinds of measures to evaluate the performance of machine learning techniques, for instance:

- Precision and recall in information retrieval and natural language processing;
- The *receiver operating characteristic* (ROC) in medicine.

	Positive examples: P	**Negative examples:** N
Classified as P	True positives: A	False positives: B
Classified as N	False negatives: C	True negatives: D

More on the receiver operating characteristic here: `http://en.wikipedia.org/wiki/Receiver_operating_characteristic`

Recall, Precision, and the F-Measure

The **accuracy** is $\dfrac{|A \cup D|}{|P \cup N|}$.

Recall measures how much relevant examples the system has classified correctly, for P:

$$\text{Recall} = \frac{|A|}{|A \cup C|}.$$

Precision is the accuracy of what has been returned, for P:

$$\text{Precision} = \frac{|A|}{|A \cup B|}.$$

Recall and precision are combined into the **F-measure**, which is defined as the harmonic mean of both numbers:

$$F = \frac{2 \cdot \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$

Measuring Quality: The Confusion Matrix

A task in natural language processing: Identify the parts of speech (POS) of words.

Example: *The can rusted*

- The human: *The*/art (DT) *can*/noun (NN) *rusted*/verb (VBD)
- The POS tagger: *The*/art (DT) *can*/modal (MD) *rusted*/verb (VBD)

↓**Correct**	**Tagger** →									
	DT	IN	JJ	NN	RB	RP	VB	VBD	VBG	VBN
DT	99.4	0.3	–	–	0.3	–	–	–	–	–
IN	0.4	97.5	–	–	1.5	0.5	–	–	–	–
JJ	–	0.1	93.9	1.8	0.9	–	0.1	0.1	0.4	1.5
NN	–	–	2.2	95.5	–	–	0.2	–	0.4	–
RB	0.2	2.4	2.2	0.6	93.2	1.2	–	–	–	–
RP	–	24.7	–	1.1	12.6	61.5	–	–	–	–
VB	–	–	0.3	1.4	–	–	96.0	–	–	0.2
VBD	–	–	0.3	–	–	–	–	94.6	–	4.8
VBG	–	–	2.5	4.4	–	–	–	–	93.0	–
VBN	–	–	4.6	–	–	–	–	4.3	–	90.6

After Franz (1996, p. 124)