IE 678 Deep Learning

02 – Feedforward Neural Networks

Part 4: Non-Linear Layers

Prof. Dr. Rainer Gemulla

Universität Mannheim

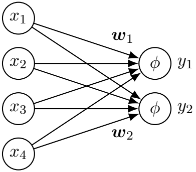
Version: 2024-1

Fully-connected layers (1)

- Recall: Layers in which all layer inputs are connected with all layer outputs are called **dense layers** or **fully-connected layers**
  - n layer inputs ($\boldsymbol{x} \in \mathbb{R}^n$), m layer outputs ($\boldsymbol{y} \in \mathbb{R}^m$)
  - Parameterized by weight vectors $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m \in \mathbb{R}^n$
  - Optionally: biases $b_1, \ldots, b_m \in \mathbb{R}$
  - **Transfer function** $\phi : \mathbb{R} \to \mathbb{R}$
- Outputs given by

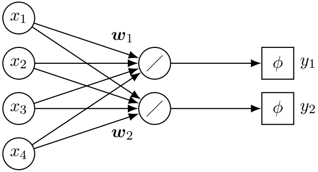
$$y_j = \phi(\langle \boldsymbol{w}_j, \boldsymbol{x} \rangle + b_j)$$

- Example: $n = 4$, $m = 2$, no bias



Fully-connected layers (2)

- We can also interpret a fully-connected layer as a (learned) linear layer followed by a (fixed) non-linearity:



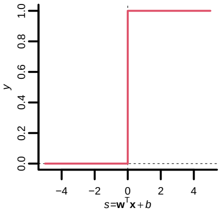
- The action of the layer (without bias) is

$$\boldsymbol{y} = \phi(\boldsymbol{W}^\top \boldsymbol{x}),$$

where we take the convection that ϕ is applied element-wise on vector inputs

Binary threshold neuron

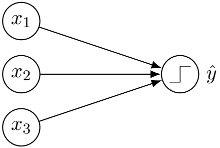
- One of the (seemingly) simplest non-linear neurons is the **binary threshold neuron** (also called *McCulloch-Pitts neuron*)
- Uses the **binary threshold function** as transfer function: outputs fixed "spike" if input s is non-negative, else "nothing"
- I.e,. $\phi(s) = I(s \geq 0) = \begin{cases} 1 & \text{if } s \geq 0 \\ 0 & \text{otherwise} \end{cases}$
- Notation: or with fixed bias ≥ 0, ≥ 1, ...



- One interpretation: each input is the truth value of some proposition, output is truth value of another proposition ($\rightarrow$ exercise)

Perceptron

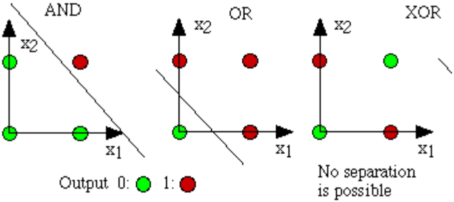
- Invented 1957 by Frank Rosenblatt at the Cornell Aeronautical Laboratory
- Corresponds to an FNN without hidden layers and binary threshold units for outputs (**single-layer perceptron**)



- Already discussed in ML course (which see)
  - Linear decision boundary = $\{\, \boldsymbol{x} : \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b = 0 \,\}$
- Many hopes and much controversy about what it can do at the time (see Olazaran (1996) for history)

Recap: What can perceptrons learn?

- Perceptrons can classify perfectly if there exists an affine hyperplane that separates the classes
  - I.e., when the data is **linearly separable**
- Otherwise, the perceptron must make errors on some inputs
- This is quite limited; e.g., perceptrons cannot learn the XOR function



- We will come back to this later

Complexity of perceptron learning

- Suppose we want to minimize the misclassification rate (0-1 loss)
- If the data is linearly separable $\rightarrow$ "easy"
  - In P; e.g., solve the linear program

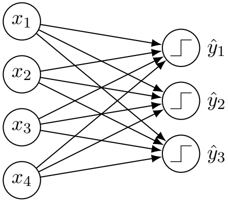
$$\begin{array}{rll} \text{minimize} & 0 & \\ \text{subject to} & \langle \boldsymbol{x}_i, \boldsymbol{w} \rangle \geq 0 & \text{for all } \boldsymbol{x}_i \text{ in pos. class } (y_i = 1) \\ & \langle \boldsymbol{x}_i, \boldsymbol{w} \rangle < 0 & \text{for all } \boldsymbol{x}_i \text{ in neg. class } (y_i = 0) \end{array}$$

- If the data is not linearly separable $\rightarrow$ "difficult"
  - Finding an optimal weight vector is NP-hard (when dimensionality n is part of the input)
  - Remains NP-hard even when weights restricted to $\{-1, 1\}$
  - NP-hard to approximate even when weights restricted to $\{-1, 1\}$
  - Fortunately, we are often able to nevertheless find sufficiently good weights in practice

Amaldi and Kann, 1995

Perceptrons with multiple output units

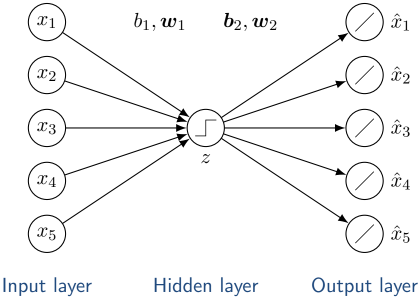
Consider a perceptron with m binary outputs for classification tasks.



1. **Multi-label classification** $\rightarrow$ works
   - Each input is associated with m binary class labels
   - Goal is to predict each of them
   - E.g.: height (small/tall), hair color (light/dark), ...
2. **Multi-class classification** (first option) $\rightarrow$ problematic
   - Each input is associated with one out of 2^m class labels
   - We associate each label with one output vector of the perceptron
   - Problem: Which label with which output vector? (choice matters)
3. **Multi-class classification** (second option) $\rightarrow$ problematic
   - Each input is associated with one out of m class labels
   - We associate each label with its indicator vector (**one-hot encoding**)
   - Problem: What if the network outputs less/more than a single 1?

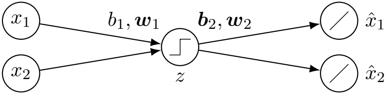
An autoencoder with a binary threshold unit

- Consider the following autoencoder (with biases)

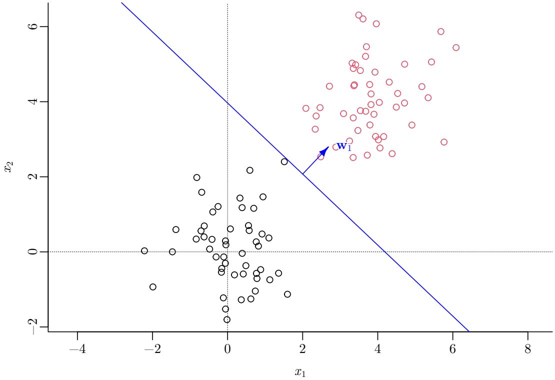


  - Observe: $z \in \{0, 1\}$ is a binary embedding (binary code)
- Assume that we want to minimize squared error over training data
  - What does this autoencoder then compute?

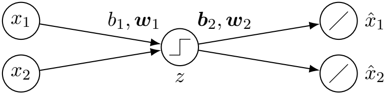
Interpreting the weights (1)



- Suppose that we are given b_1 and $\boldsymbol{w}_1$
- The binary threshold unit then acts as a linear classifier
  - Input $\boldsymbol{x}$ mapped to either $z = 0$ (bottom left) or $z = 1$ (top right)



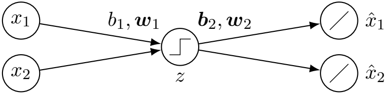
Interpreting the weights (2)



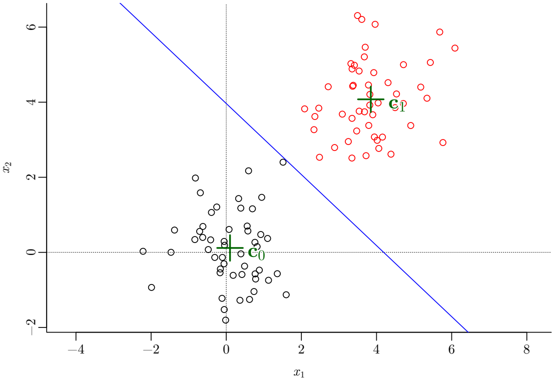
- Let's now look at $\boldsymbol{b}_2$ and $\boldsymbol{w}_2$
  - Given z, output is $\hat{\boldsymbol{x}} = \boldsymbol{w}_2 z + \boldsymbol{b}_2$
  - All points in "class" $z = 0$ are mapped to $\boldsymbol{c}_0 \stackrel{\text{def}}{=} \boldsymbol{b}_2$
  - All points in "class" $z = 1$ are mapped to $\boldsymbol{c}_1 \stackrel{\text{def}}{=} \boldsymbol{b}_2 + \boldsymbol{w}_2$
- Given b and $\boldsymbol{w}_1$, what are the optimal choices of $\boldsymbol{c}_0$ and $\boldsymbol{c}_1$?
  - Denote by z_i the class of input $\boldsymbol{x}_i$
  - Squared error is $\sum_i \sum_j (x_{ij} - \hat{x}_{ij})^2 = \sum_i \|\boldsymbol{x}_i - \boldsymbol{c}_{z_i}\|^2$
  - Alternatively: $\sum_{i:z_i=0} \|\boldsymbol{x}_i - \boldsymbol{c}_0\|^2 + \sum_{i:z_i=1} \|\boldsymbol{x}_i - \boldsymbol{c}_1\|^2$
  - For each class k, our goal is to minimize the squared Euclidean distance between the $\boldsymbol{x}_i$'s of the class and its representative $\boldsymbol{c}_k$
  - Optimum solution is the mean of the examples of the class

$$\boldsymbol{c}_k = \frac{1}{\sum_{i:z_i=k} 1} \sum_{i:z_i=k} \boldsymbol{x}_i$$

Interpreting the weights (3)

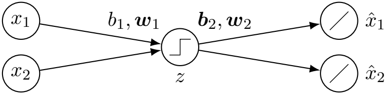


- The overall optimum solution is



- Can you see what the autoencoder does?

Interpreting the weights (4)



- Optimum solution agrees with K-means for $K = 2$
- K-means objective is to minimize the sum of squared distances

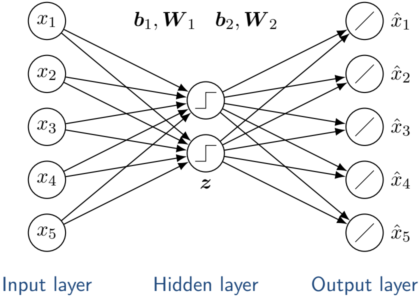
$$\underset{C}{\operatorname{argmin}} \sum_{k=1}^{K} \sum_{\boldsymbol{x} \in C_k} \|\boldsymbol{x} - \boldsymbol{\mu}_k\|^2 \,,$$

  where $\boldsymbol{\mu}_k$ is the mean of the points in cluster C_k

- Given an optimal K-means clustering for $K = 2$
  - Each data point is associated to cluster of closest representative
  - We set $\boldsymbol{c}_k = \boldsymbol{\mu}_k$ (and thus obtain $\boldsymbol{b}_2$ and $\boldsymbol{w}_2$)
  - We set b_1 and $\boldsymbol{w}_1$ such that the decision boundary is the set of points with equal distance to $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ (see previous slide)
  - The binary threshold unit then associates each point $\boldsymbol{x}_i$ with its correct cluster z_i

An autoencoder with multiple binary threshold units

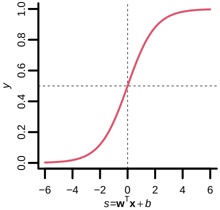
- What happens if we have multiple binary threshold units?



- This autoencoder also "clusters" the data
  - Associates each data point with a "binary code" (00, 01, 10, 11)
  - Each codeword can be seen as a cluster (2^Z in total)
- For $Z > 1$ binary threshold units, the optimum solution does does not correspond to K-means anymore (with $K = 2^Z$)
  - Why? $\rightarrow$ exercise

Recall: Logistic neuron

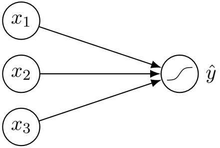
- Use logistic function $\phi(s) = \sigma(s) \overset{\text{def}}{=} \frac{1}{1+\exp(-s)}$
- Notation: 



- Gives a real-valued output that is smooth and bounded in $[0, 1]$
  - Negative activations mapped to value < 0.5
  - 0 activation mapped to 0.5
  - Positive activation mapped to value > 0.5
- Non-linear

An FNN with a single logistic unit

- If the binary threshold unit of a perceptron is replaced by a logistic unit, we obtain an FNN similar to a perceptron



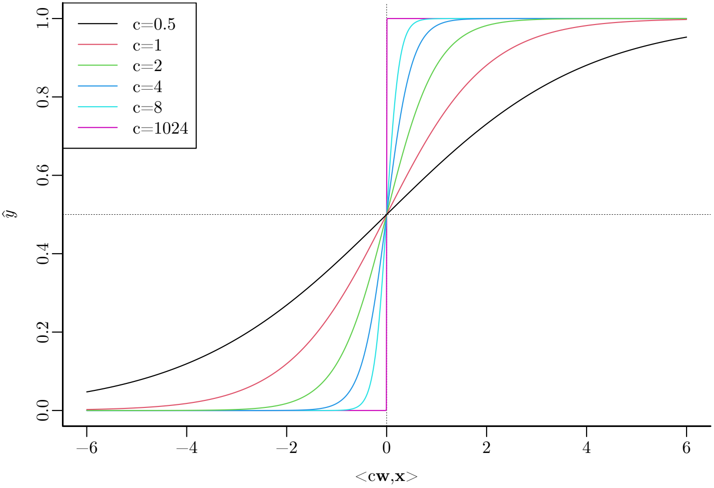
- What's the difference?
  - Fix some weight vector $\boldsymbol{w}$ (and ignore bias)
  - Above neural network outputs $\hat{y} \in [0, 1]$ with

$$\hat{y} = \sigma(\langle \boldsymbol{w}, \boldsymbol{x} \rangle) \begin{cases} < 0.5 & \langle \boldsymbol{w}, \boldsymbol{x} \rangle < 0 \\ \geq 0.5 & \langle \boldsymbol{w}, \boldsymbol{x} \rangle \geq 0 \end{cases}$$

  - If output of the logistic unit is rounded to the closest integer, one obtains output of the corresponding perceptron
  - Logistic unit can be seen as a "smooth" version of a binary threshold unit

Smoothing

If we scale the weights by some constant $c > 0$, we change the degree of smoothing.



Binary classification

- Suppose we use the network for a binary classification task
  - Given a labeled set $\mathcal{D} = \{ (\boldsymbol{x}_i, y_i) \}_{i=1}^{N}$ of input-output pairs
- We can minimize the misclassification error (**0-1 loss**)

$$\sum_i |y_i - \mathrm{round}(\hat{y}_i)|$$

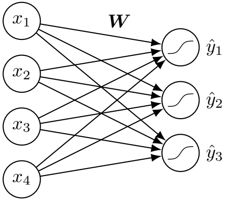
  - Equivalent to perceptron
  - Outputs related to distance from decision boundary, but no probabilistic interpretation possible
- We can maximize the log-likelihood of the provided labels

$$\ln \mathcal{L} = \sum_i [\, y_i \ln \hat{y}_i + (1 - y_i) \ln(1 - \hat{y}_i) \,]$$

  - Equivalent to logistic regression
  - Input $\langle \boldsymbol{w}, \boldsymbol{x} \rangle$ to logistic transfer function interpreted as estimate of log odds of positive class
  - Output $\hat{y}_i$ interpreted as confidence for positive class
- **Output layer of FNNs for binary classification tasks is typically a logistic neuron**

Multi-class classification (bad approach)

- Naive (bad) approach to multi-class classification
  - For C classes, use C logistic neurons
  - Associate each label with its indicator vector (one-hot encoding)
  - We may interpret output $\hat{y}_c$ as confidence in label c and predict the label with the largest confidence



- Problem: Interpretation of $\hat{y}_c$ as confidence not valid
  - Outputs $\hat{y}_c$ may not sum to one $\rightarrow$ $\hat{\boldsymbol{y}}$ is not a probability vector
- Solution: tie the output neurons appropriately
  $\rightarrow$ **softmax layer**

Recap: The softmax function

- The **softmax function** $S(\boldsymbol{\eta})$
  - Takes a real vector $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_C)^\top \in \mathbb{R}^C$
  - And transforms it into an C-dimensional probability vector $S(\boldsymbol{\eta})$

$$S(\boldsymbol{\eta})_c = \frac{\exp(\eta_c)}{\sum_{c'=1}^{C} \exp(\eta_{c'})}$$

  - Called this way because it exaggerates differences and acts somewhat like the max function (approximates indicator function of largest coefficient)

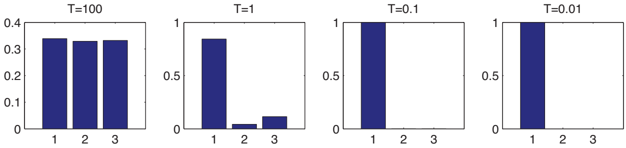
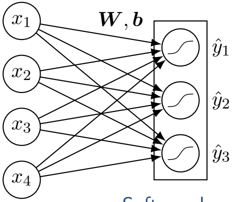


Figure 4.4 Softmax distribution $\mathcal{S}(\boldsymbol{\eta}/T)$, where $\boldsymbol{\eta} = (3, 0, 1)$, at different temperatures T. When the temperature is high (left), the distribution is uniform, whereas when the temperature is low (right), the distribution is "spiky", with all its mass on the largest element. Figure generated by `softmaxDemo2`.

Murphy, 2012

Softmax layer

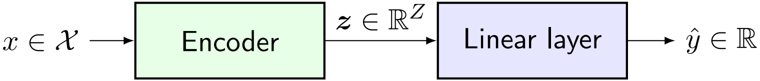


Softmax layer

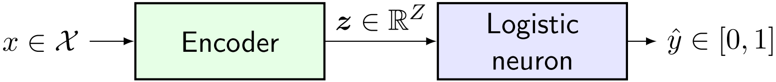
- A **softmax layer** computes $\hat{\boldsymbol{y}} = S\left(\frac{\boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b}}{T}\right)$
  - $\hat{\boldsymbol{y}} \in \mathcal{S}_C$ is a probability vector
  - T is a hyperparameter known as the **temperature**
    $\rightarrow$ Controls smoothness of distribution (assume $T = 1$ for now)
- FNN with single softmax layer trained with MLE / ERM + log loss
  - $\hat{y}_c$ is model confidence in label c
  - Equivalent to multinomial logistic regression (softmax regression)
- **Output layer of FNNs for multi-class classification tasks is typically a softmax layer**

Summary: Typical output layers

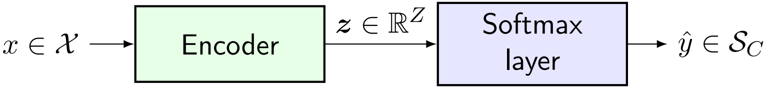
- Regression



- Binary classification



- Multi-class classification (C classes)



- Multi-label classification (C labels)