Hochschule Fulda
University of Applied Sciences

Machine Learning

Foundations of machine learning
The linear classifier

Alexander Gepperth, November 2021

Recap: foundations of machine learning

Mathematical formalization

- Machine learning is about:
  - data: matrix $X \in \mathbb{R}^{N,n}$ with rows $\vec{x}_i \in \mathbb{R}^n$
  - targets (“labels”): matrix $T \in \mathbb{R}^{N,k}$ with row vectors $\vec{t}_i \in \mathbb{R}^k$ (why vectors? later!)
  - we wish to find a “good” model function ... $\vec{f} : X \in \mathbb{R}^{N,n}, \vec{w} \in \mathbb{R}^m \mapsto Y \in \mathbb{R}^{N,k}$
  - ... such that correct target values are obtained

How to represent model function?

- Choose a function family with parameter vector $\vec{w}$
  $\vec{f}(X, \vec{w})$
- parameters controls behavior of model function
- better parameters $\rightarrow$ better model!
- after functional form is fixed, improving the model is done by finding a better $\vec{w}$ $\rightarrow$ **LEARNING**

Loss functions

- Learning: adapt parameters in $\vec{f}(X,\vec{w}) \in \mathbb{R}^{N,k}$ in order to find best (or better) model
- “better'': measured by **loss function** $\mathcal{L}\left(\vec{f},X,T,\vec{w}\right) \equiv \mathcal{L}(\vec{w})$
- so, effectively: learning means adapting the parameters $\vec{w}$ so that $\mathcal{L}(\vec{w})$ is minimized!
- Loss function must be chosen according to the problem

Summary

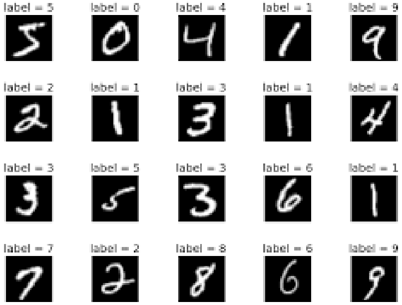
- For supervised machine learning:
  – we need data samples
  – we need targets
  – we must specify a model function
  – we must specify a loss function

CUT: Q&A

Classification problems

What are classification problems?

- Model should group samples into one out of a finite number of distinct **classes** or **categories**



- Examples:
  - cars or non-cars
  - cats, dogs or horses
  - hand-written digits (0 – 9)
  - ...

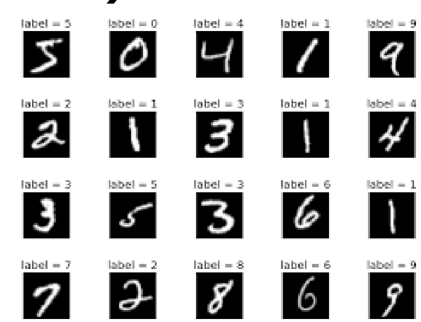
What are classification problems?

- Terminology:
  - binary classification problems: 2 classes
  - multi-class classification problems: >2 classes
  - classifier: model that (learns to) perform classification

Classification problems: target encoding

Classification problems: Scalar target encoding

- Encode class as a single scalar
- Car classification example:
  assign to each image one of two possible values $t_i \in \{1, 2\}$ (binary classification)
- Also possible: multi-class classification with K classes:
  $t_i \in \{1, \ldots, K\}$



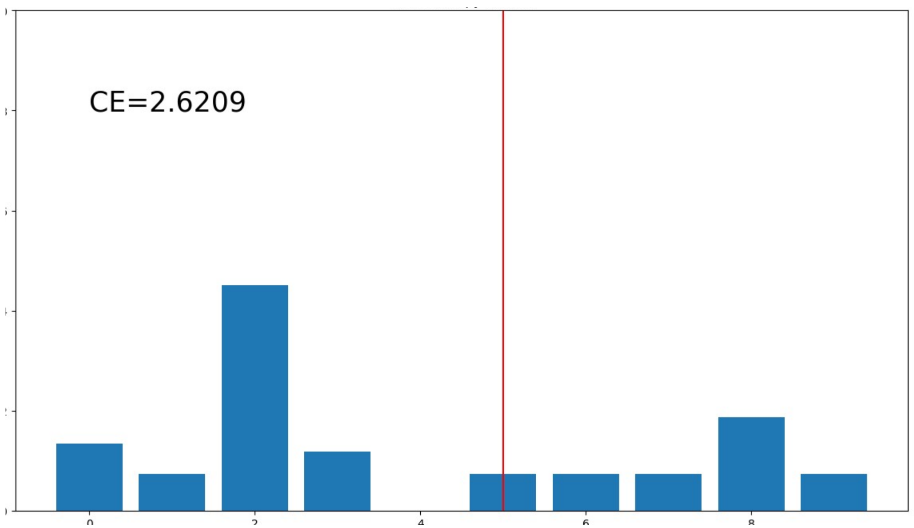
Classification problems: One-hot target encoding

- Encode individual scalar targets $t_i \in \{1, \dots, K\}$ as "one- hot"-vectors: $\rightarrow \ \vec{t_i} \in \mathbb{R}^k$

- Target matrix form: $T_{ij} = \begin{cases} 1 & \text{if } j = t_i \\ 0 & \text{else} \end{cases}$

- for K=*3* classes: $t_i = 3 \rightarrow \vec{t_i} = (0, 0, 1,)$
  $t_i = 1 \rightarrow \vec{t_i} = (1, 0, 0)$

Classification problems: model properties



Classification problems: model properties

- If targets are vectors, model outputs should also be vectors: $Y = \vec{f}(X) \in \mathbb{R}^{N,k}$
- Interpretation: largest vector component determines class: $c_i = \text{argmax}_k Y_{ik}$
- advantage: separate confidence for each class! $t_i = 3 \rightarrow \vec{t_i}^T = (0, 0, 1,)$ $\vec{y_i}^T = (0.45, 0.05, 0.5)$

Correct classification?

Classification problems: model properties

- If targets are vectors, model outputs should also be vectors: $Y = \vec{f}(X) \in \mathbb{R}^{N,k}$
- Interpretation: largest vector component determines class: $c_i = \mathrm{argmax}_k Y_{ik}$
- advantage: separate confidence for each class! $t_i = 3 \rightarrow \vec{t_i}^T = (0,0,1,)$ $\vec{y_i}^T = (0.45, 0.05, 0.5)$

YES!

Classification problems: model properties

- If targets are vectors, model outputs should also be vectors: $Y = \vec{f}(X) \in \mathbb{R}^{N,k}$
- Interpretation: largest vector component determines class: $c_i = \mathrm{argmax}_k Y_{ik}$
- advantage: separate confidence for each class! $t_i = 3 \rightarrow \vec{t_i}^T = (0, 0, 1,)$ $\vec{y_i}^T = (0.5, 0.05, 0.45)$

NO!

Classification problems: loss functions

Classification error

- For binary/multi-class problems
- Assumes that model output matrix $Y = f(X)$ tries to approximate one-hot-coded target vectors

$$\mathcal{L} = \frac{1}{N} \sum_{i=1}^{N} \begin{cases} 1 & \operatorname{argmax}_k Y_{ik} \neq \operatorname{argmax}_k T_{ik} \\ 0 & \text{else} \end{cases}$$

- **Example!**
- not differentiable, constant almost everywhere

Cross-entropy

- For binary/multi-class classification problems
- Assumes: $Y_{ik} \in]0,1]$ and $\sum_k Y_{ik} = 1 \, \forall i$
- Cross-entropy for single sample $\vec{x}$ and target $\vec{t}$

$$\tilde{\mathcal{L}}^{CE} = -\sum_k t_k \log y_k$$

matrices

- Cross-entropy for multiple samples X and targets T

$$\mathcal{L}^{CE} = -\frac{1}{N} \sum_{i=1}^{N} \sum_k T_{ik} \log Y_{ik}$$

What does cross-entropy express?

- Cross-entropy is determined by log of model output at the place k^* where label = 1:

$$\tilde{\mathcal{L}}^{CE} = -\sum_k t_k \log y_k = -\log y_{k^*}$$

- log is always negative (since model ouputs y_k <= 1)
- lowest value is 0, unbounded from above
- if model has nonzero values aside from k^*
  → lower value at k^* → lower cross-entropy

Demo: cross-entropy

The linear classifier model

The linear classifier

- preliminaries: softmax function $S_l(\vec{x}) = \frac{\exp(x_l)}{\sum_k \exp(x_k)}$
  - ensures normalization: $\sum_l S_l(\vec{x}) = 1$
  - positive and bounded: $S_l(\vec{x}) \in [0, 1]$
  - enhances biggest values, suppresses smallest values
- Simple partial derivatives: $\frac{\partial S_i}{\partial x_j} = \delta_{ij} S_i - S_i S_j$

The linear classifier

Vector-to-vector!

- preliminaries: softmax function $S_l(\vec{x}) = \frac{\exp(x_l)}{\sum_k \exp(x_k)}$
  - ensures normalization: $\sum_l S_l(\vec{x}) = 1$
  - positive and bounded: $S_l(\vec{x}) \in [0, 1]$
  - enhances biggest values, suppresses smallest values
- Simple partial derivatives: $\frac{\partial S_i}{\partial x_j} = \delta_{ij} - S_i S_j$

The linear classifier

- Model takes a matrix $X \in \mathbb{R}^{N,n}$ and produces an output matrix $Y = \vec{f}(X) \in \mathbb{R}^{N,k}$ (*k*: # classes)
- normalization and boundedness ensured by softmax
- Model formula: $\vec{f}(\vec{X}, W, \vec{b}) = \vec{S}\Big(X\underbrace{W}_{\text{weights}} + \underbrace{\vec{b}^T}_{\text{biases}}\Big)$

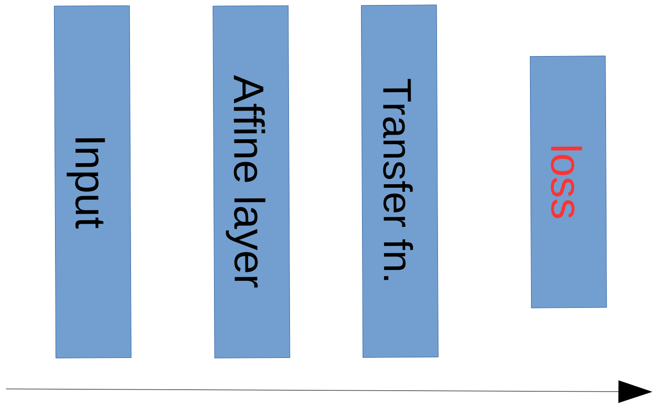
- Cross-entropy loss

The linear classifier

- Model takes a matrix $X \in \mathbb{R}^{N,n}$ and produces an output matrix $Y = \vec{f}(X) \in \mathbb{R}^{N,k}$ (**k**: # classes)
- normalization and boundedness ensured by softmax **perfect for cross-entropy!!**
- Model formula: $\vec{f}(\vec{X}, W, \vec{b}) = \vec{S}\Big(X\underbrace{W}_{\text{weights}} + \underbrace{\vec{b}^T}_{\text{biases}}\Big)$
- Cross-entropy loss

The linear softmax MC classifier as a simple DNN

- Layered model:
  Affine layer: $A = XW + \vec{b}^T$
  Softmax layer: $Y = \vec{S}(A)$
- Loss computed from model outputs Y



Demo: The linear softmax MC classifier on MNIST

- K=10, loss function: cross-entropy
- model outputs look like probabilities, are they?
- why such a complex model? Why one-hot coding of targets?
  - more information in classifier outputs
  - more free parameters in model, one set per class

How to optimize linear softmax MC?

- Obviously: need to adapt all the weights W and biases $\vec{b}$
- How?

How to optimize linear softmax MC?

- Obviously: need to adapt all the weights W and biases $\vec{b}$
- How?
- Gradient descent!!