Deep Learning

Università Roma Tre
Dipartimento di Ingegneria

Anno Accademico 2022 - 2023

Multilayer Perceptrons, One-hot encoding e Softmax

Sommario

- Monotonicità
- MLP e Hidden layers
- Non linearità
- Funzioni di attivazione

- Datasets
- MLP e Tensorflow
- Da regressione lineare a classificazione
- Funziona softmax
- One-hot encoding e misure di distanza

Motivazioni

- Tale dispensa richiama in modo sommario molti concetti trattati nel corso di ML con particolare attenzione ai concetti che interessano maggiormente lo sviluppo di architetture DL (architetture MLP).
- Si rimanda al materiale del corso di ML per i dettagli

Monotonicità

- Le architetture lineare impongono l'assunzione di **monotonicità**: l'incremento di una feature generare un incremento/decremento nel valore in output del modello, a seconda del valore dei pesi (o parametri).
- Per certi task è verosimile, sebbene non sempre vero, ad esempio:
  - Task: "*un individuo sarà regolare con le rate del mutuo?*". Se il salario passa da 0K a 50K la probabilità che ripaghi il mutuo sarà molto diversa; mentre se il salario passa da 1M a 1,05M la probabilità non cambierà molto.
  - Task: "*predire se un individuo è malato in base alla temperatura*". T << 37 o T >> 37 indica una possibile patologia.
- Come pensi si può risolvere il problema impiegando un algoritmo di regressione lineare?

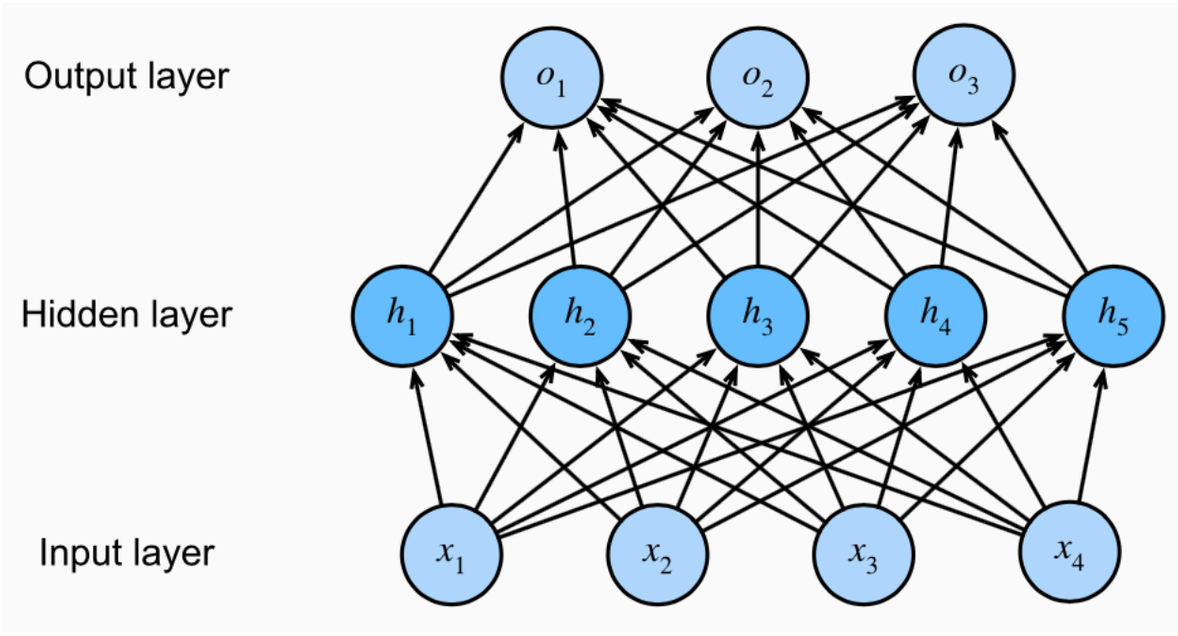
Monotonicità

- Per un task "*l'immagine contiene un cane*?", come possiamo creare una relazione tra un certo pixel è una classe in output?
- L'assunzione di linearità ci impone un vincolo tra:
  - luminosità del pixel <-> classe di appartenenza;
- ignorando però il contesto (altri pixel) e la complesse relazioni tra essi che portano a rappresentare visivamente un oggetto.

- Invece di definire una rappresentazione adeguata, impieghiamo reti neurali *multistrato*, dove gli *hidden layer* si occupano di riconoscere una rappresentazione adeguata dei dati in input, che viene impiegata da un predittore lineare per generare l'output.

Hidden layers e MLP

- L'approccio più semplice per aggiungere strati nascosti è *impilarli* (stack) uno dopo l'altro, ottenendo L layers.
- Interpretiamo gli L layer, tranne l'ultimo, come l'insieme di nodi impiegati per la rappresentazione, e l'ultimo come predittore lineare.
- Otteniamo una architettura **Multilayer perceptron (MLP)** fully connected.



Da lineare a non lineare

- Indichiamo con $\mathbf{X} \in \mathbb{R}^{n \times d}$ un *minibatch* (una sottoinsieme del dataset di training) di *n* istanze dove ogni istanza ha *d* features.
- Per un hidden layer con *h* unità, indichiamo con $\mathbf{H} \in \mathbb{R}^{n \times h}$ il relativo output. Avendo layer fully connected, abbiamo come parametri:
  - i pesi $\mathbf{W}^{(1)} \in \mathbb{R}^{d \times h}$ e i bias $\mathbf{b}^{(1)} \in \mathbb{R}^{1 \times h}$.
- Il layer di output avrà parametri: $\mathbf{W}^{(2)} \in \mathbb{R}^{h \times q}$ e $\mathbf{b}^{(2)} \in \mathbb{R}^{1 \times q}$
- L'output è ricavato nel seguente modo:
  - $\mathbf{H} = \mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}$
  - $\mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}$
- Secondo te, combinando più funzioni affini, siamo riusciti a introdurre non linearità nel modello?

Da lineare a non lineare

- Combinando le equazioni viste in precedenza otteniamo un modello equivalente ad un singolo layer:
- $\mathbf{O} = (\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})\mathbf{W}^{(2)} + \mathbf{b}^{(2)} = \mathbf{X}\mathbf{W}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)} = \mathbf{X}\mathbf{W} + \mathbf{b}$
- La non linearità viene espressa mediante funzioni di attivazione σ non lineari (es. ReLU) impiegate all'interno delle unità nascoste, a valle della trasformazione affine.

- Facendo *stacking* di più hidden layer con funzioni non lineari, es:
  - $\mathbf{H}^{(1)} = \sigma_1(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})$
  - $\mathbf{H}^{(2)} = \sigma_2(\mathbf{H}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)})$
- si ottengono architetture **deep**, che approssimano funzioni più complesse.

Universal approximators

- Ci si può chiedere quanta capacità rappresentativa (i.e. quanto è *potente*) è espressa da una rete neurale.
- Alcuni risultati suggeriscono che *perfino con un solo hidden layer* è possibile approssimare qualsiasi funzione con un numero adeguato di unità.
  - Una rete neurale deep può essere pensata come un programma in C, cioè puoi risolvere qualsiasi problema software, ma i programmi possono raggiungere complessità molto elevate.
- Vedremo architetture di reti deep possono risolvere gli stessi task in modo molto più efficiente.

Funzioni di attivazione

- Ne esistono molte, ad esempio:
  - Funzione Sigmoide
  - Tangente iperbolica (tanh)
  - Relu
  - Leaky Relu
  - Swish
  - Relu parametrizzato
  - ELU
  - Softplus e Softsign
  - Selu
  - Gelu
- Durante il corso discuteremo pro e contro delle principali.

- Colab 03-funzioni_di_attivazione_5.1.2

Alcuni toy datasets

- Elenchiamo alcuni dataset che vengono spesso impiegati negli approcci di ML e DL:
  - MNIST
  - notMNIST
  - fashion-MNIST

- Dataset più complessi saranno introdotti più avanti.

Dataset MNIST

- Composto da cifre numeriche, usato per addestrare sistemi OCR.
  - "If it doesn't work on MNIST, it won't work at all"; "Well, if it does work on MNIST, it may still fail on others."
- Contiene 60K immagini di addestramento e 10K di training.
  - 1998: un linear classifier ha ottenuto 7.6% di errore rate.
  - 2012: per mezzo di una architettura DL (convolutional neural networks) si è arrivati al 0.23%.
- Ogni immagine è rappresentata in scala di grigi (256 livelli). Le cifre sono centrate in un box 28x28 pixel: abbiamo 784 valori in [0-255] per rappresentare una cifra.

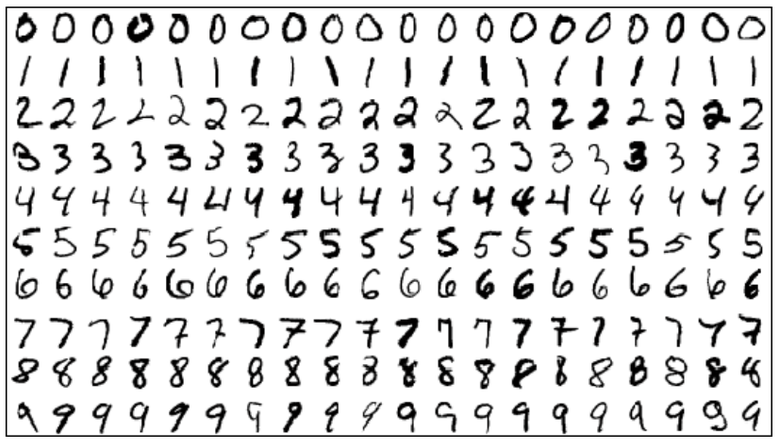
- http://yann.lecun.com/exdb/mnist/
- https://www.kaggle.com/c/digit-recognizer/data
- Implementazione online JS (ott'17) http://myselph.de/neuralNet.html

Dataset MNIST: train.csv e test.csv

- Il file train.csv contiene una matrice con 785 colonne. La prima colonna è il *label* della cifra (es. 3) e le restanti colonne sono la rappresentazione sequenziale dell'immagine:

```
000 001 002 003 ... 026 027
028 029 030 031 ... 054 055
056 057 058 059 ... 082 083
 |   |   |   |  ...  |   |
728 729 730 731 ... 754 755
756 757 758 759 ... 782 783
```

- Il file test.csv ha la stessa rappresentazione senza la prima colonna.
- Esempio di immagini:



Dataset MNIST - svantaggi

- Troppo semplice: algoritmi classici di ML raggiungono i 97% di precisione, architetture DL il 99.7%
- Si rischia di ideare nuove architetture adatte solo per questo dataset e difficilmente adattabili in altri contesti.
- Molto diverso dai task studiati attualmente nell'ambito del DL.

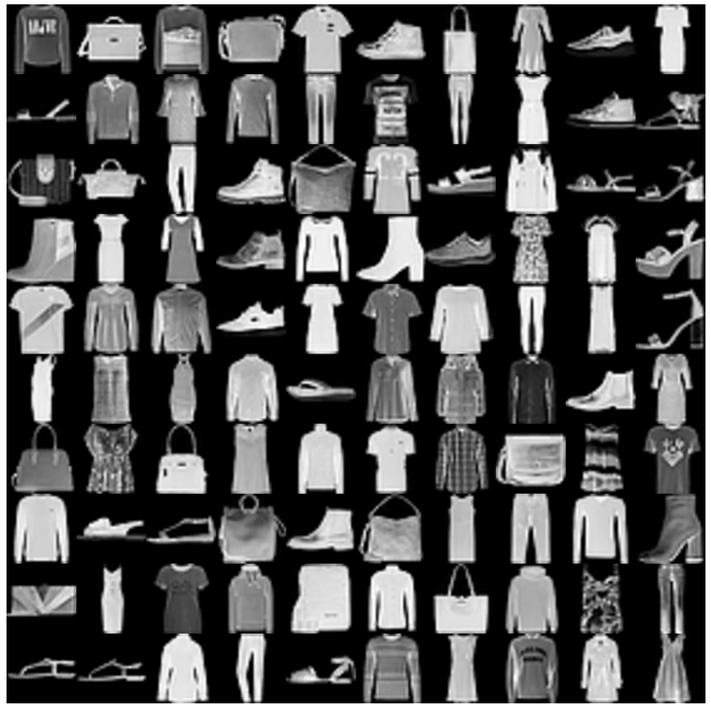
Dataset notMNIST

- Simile a MNIST: contiene 10 labels (lettere da A a J), ma ogni lettera nel dataset ha un font molto diverso dalle altre, es.:

- http://yaroslavvb.blogspot.fi/2011/09/notmnist-dataset.html
- Download http://yaroslavvb.com/upload/notMNIST/
  - notMNIST_large.tar.gz -> training e validazione
  - notMNIST_small.tar.gz -> test

Dataset fashion-MNIST

- Fornito da Zalando: 10 classi che fanno riferimento a generi di vestiario (es. sandali, t-shirt, borse, etc).
- Contiene 60K immagini di addestramento e 10K di training.
- Ogni immagine è rappresentata in scala di grigi di 28x28 pixel



- https://github.com/zalandoresearch/fashion-mnist
- Side-by-side accuracy MNIST vs fashion MNIST:
  - http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/#

Altri dataset popolari sulle immagini

- **CIFAR-10 (e 100)**: 60K 32x32 colour images in 10 classes.
- **ImageNet**: 1,5 milioni di immagini organizzate etichettate su WordNet. In media 1K immagini per concetto.
- **ILSVRC2012 task 1**: 10 milioni di immagini e +1K classi.
- **Open Image**: 9 milioni di URLs di immagini annotate con bounding boxes e migliaia di classi.
- **VisualQA**: open-ended questions su 265K immagini. In media 5.4 questions per immagini con 10 ground truth answers per question.
- **The Street View House Numbers**: 600K immagini di numeri civici.

- Risultati sperimentali ottenuti per varie architetture
  - http://rodrigob.github.io/are_we_there_yet/build/#datasets

MLP e Tensorflow

- Proviamo a costruire una MLP con Tensorflow (Keras).

- Coalb 04-mlp_5.2.1.ipynb

Da regressione lineare a classificazione

- Nei problemi di regressione rispondiamo a domande del tipo "*Quale quantità o valore?*". Ma molti problemi mirano a trovare una classe di appartenenza,
  - es. è una email di spam? è più probabile che un utente si iscriva ad un abbonamento oppure no?
- Ci può interessare la classe più verosimile (*hard assignements*), oppure la distribuzione di probabilità sulle classi possibili (*soft assignements*), o siamo in presenza di più classi di appartenenza (*multi-label classification*).
- In caso di più valori in output (es. un layer di output con più nodi), ogni valore può essere interpretato come *il grado di appartenenza dell'istanza in ingresso ad una certa classe*. La loss misura il discostamento tra classe attesa e valori prodotti dal modello.

Classificazione binaria

- Si ha interesse ad associare una istanza in input ad un valore in y∈{0,1}

$$\hat{y} = \text{argmax}_y P(y|x)$$

- Se usiamo un modello di regressione, estraiamo dall'istanza x features numeriche e le combinano linearmente. Il risultato dipende dalle somme dei valori di input e dei parametri del modello.
- Al risultato del modello applichiamo la funzione *logistic*, che restituisce un valore in [0,1]. La funzione è facilmente differenziabile.
- Interpretiamo tale valore come la probabilità di appartenenza ad una delle due classi.
- Si ottiene una **logistic regression**.

- Vogliamo generallizzare la logistic regression al caso K classi, con K>2

Esempio

- Supponiamo di avere 3 classi e l'output della combinazione lineare sia:

$$y = \begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix}$$

- Sebbene la classe più probabile sia associata all'indice 1, i valori non sono direttamente interpretabili come distribuzioni di probabilità, infatti:
  - I valori non sono in in [0,1]
  - La somma non è pari 1

La funzione Softmax

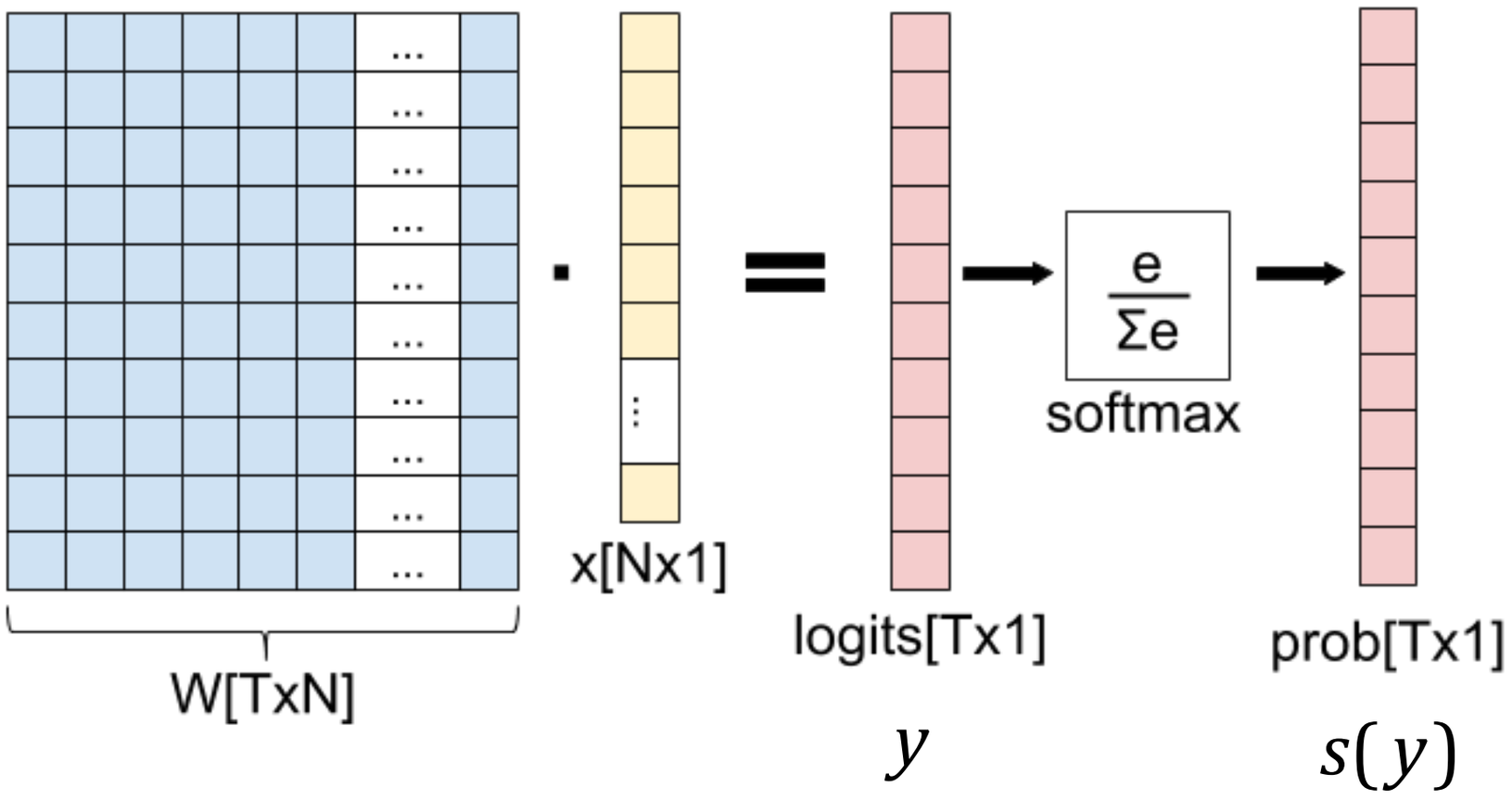
- La funzione **softmax** prende in input un vettore in $\mathbb{R}^T$ e dà in output un vettore $\mathbb{R}^T$ nell'intervallo (0,1] la cui somma è pari a 1. È definita:

$$S(y_i) = \frac{e^{y^i}}{\sum_j^K e^{y^j}}$$

- L'output può essere interpretato come distribuzione di probabilità su K classi, a differenza di altri modelli (es. classificatore SVM).

Layer softmax nelle reti neurali

- La funzione softmax è tipicamente applicata all'output di un layer fully-connected, creando un nuovo layer chiamato **softmax**.
- Il seguente esempio rappresenta un singolo layer, con funzione di attivazione softmax su T classi.



- Nota: la funzione softmax introduce non linearità.

Softmax in Keras

- In Keras è semplice implementare il modello precedente con il parametro *activation* di layer Dense:

```
# this is a logistic regression in Keras
x = Input(shape=(32,))
y = Dense(16, activation='softmax')(x)
model = Model(x, y)
```

One-hot encoding

- Nel ML le rappresentazioni dell'input e output sono sottoinsiemi dei domini $\mathbb{N}$ e $\mathbb{R}$. Tali insieme introducono implicitamente ordinamenti.
  - Es. se abbiamo 3 categorie (es. rosso=1, bianco=2 e nero=3) e gli assegniamo 3 numeri, introduciamo una relazione di ordinamento che non esiste nei dati.
- Durante l'addestramento tali relazioni possono essere considerate potenziali features, e di conseguenza apprese dall'algoritmo
  - Es. Le due istanze Rosso-Nero possono considerarsi più distanti rispetto a Rosso-Bianco
- La rappresentazione ***one-hot*** caratterizza ogni istanza con una configurazione univoca, costituita da una sequenza binaria di zero, tranne un solo elemento pari a 1.

One-hot encoding in Python

- Colab 05_onehot.ipynb

Loss e one-hot encoding

- Se la *softmax* genera una distribuzione di probabilità su K possibili, la codifica one-hot genera una distribuzione che "concentra" tutta la densità di probabilità sulle classi corrette, es.:

$$[0, \ldots, 0, 1, 0 \ldots, 0].$$

- Per addestrare il modello occorre definire una misura di loss che tenga conto della distanza tra le due distribuzioni.

Misura di Distanza: cross entropy

- Per confrontare due generici vettori *p* e *q* che rappresentano distribuzioni di probabilità si impiega la misura **cross entropy**:

$$H(p,q) = -\sum_{x} p(x) \cdot \log \mathrm{q(x)}$$

- Dove *x* si estende su tutte i valori potenziali della variabile causale su cui sono definite le probabilità, cioè le classi in output.

- Attenzione: la funzione H non è simmetrica:

$$H(p,q) \neq H(q,p)$$

- Se uno dei parametri (*p* o *q*) è codificato one-hot, in che posizione conviene averlo?

Misura di Distanza: cross entropy

- Nella fase di addestramento un parametro della cross entropy è l'output della funzione softmax s(y), mentre il secondo è la codifica one-hot che indica una o più classi di appartenenza.

- Supponiamo di usare la codifica one-hot per il calcolo dei logaritmi:

$$\hat{y} = \begin{bmatrix} 1.0 \\ 0.0 \\ 0.0 \end{bmatrix} \qquad D(s(y), \hat{y}) = -\left(s(y_1) \cdot \log 1.0 + s(y_2) \cdot \log 0 + s(y_3) \cdot \log 0\right)$$

- Anche il layer softmax può generare valori 0, ma è un problema raro e facilmente risolvibile (es. aggiungendo un ε).

Multinomial logistic classification

Input

$$x = \begin{bmatrix} 2.0 \\ 0.7 \\ 1.5 \\ \dots \\ 8.0 \end{bmatrix}$$

Linear model

$$y = \begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix}$$

Softmax

$$S(y) = \begin{bmatrix} 0.659 \\ 0.242 \\ 0.099 \end{bmatrix}$$

Labels
Onehot rep.

$$\hat{y} = \begin{bmatrix} 1.0 \\ 0.0 \\ 0.0 \end{bmatrix}$$

$D(\hat{y}, S(y))$

Cross entropy distance

Esercizio

- Supponiamo di avere 3 istanze di addestramento che consistono in varie features (es. sex, age, etc) e vogliamo predire se un elettore voterà democratico o repubblicano con una rete neurale.
- Avendo due reti che producono in output i seguenti valori:

#1

computed	targets	correct?
0.3 0.3 0.4	0 0 1 (democrat)	yes
0.3 0.4 0.3	0 1 0 (republican)	yes
0.1 0.2 0.7	1 0 0 (other)	no

#2

computed	targets	correct?
0.1 0.2 0.7	0 0 1 (democrat)	yes
0.1 0.7 0.2	0 1 0 (republican)	yes
0.3 0.4 0.3	1 0 0 (other)	no

- Calcola l'errore impiegando: (1) cross entropy, (2) mean squared error, (3) accuratezza (binaria).

Confronto tra misure di loss

- Cross entropy
  - #1: -(ln(0.4) + ln(0.4) + ln(0.1)) / 3 = 1.38
  - #2: -(ln(0.7) + ln(0.7) + ln(0.3)) / 3 = 0.64 (smaller)
- Mean squared error
  - #1: [(0.3 - 0)^2 + (0.3 - 0)^2 + (0.4 - 1)^2 + …] / 3
    - (0.54 + 0.54 + 1.34) / 3 = 0.81
  - #2: (0.14 + 0.14 + 0.74) / 3 = 0.34 (smaller)
- Accuratezza (binaria)
  - Entrambi: classification error 1/3 = 0.33, accuracy 2/3 = 0.67
- Nota: le implementazione delle misure discusse sono in sklearn.metrics

Confronto tra misure di loss

- Cross entropy
  - #1: 1.38
  - #2: 0.64 (migliore)
- Mean squared error
  - #1: 0.81
  - #2: 0.34 (migliore)
- Accuratezza (binaria)
  - Entrambi: 0.67
- Rispetto alla cross entropy, MSE da molta importanza agli output sbagliati, ma allo stesso tempo, se la rete si avvicina ai risultati corretti, i gradienti diventano assai bassi, rallentando notevolmente la convergenza.

#1

computed	targets	correct?
0.3 0.3 0.4	0 0 1 (democrat)	yes
0.3 0.4 0.3	0 1 0 (republican)	yes
0.1 0.2 0.7	1 0 0 (other)	no

#2

computed	targets	correct?
0.1 0.2 0.7	0 0 1 (democrat)	yes
0.1 0.7 0.2	0 1 0 (republican)	yes
0.3 0.4 0.3	1 0 0 (other)	no