Reti neurali

Le reti neurali (NN) o “multilayer perceptron” sono divise in layers ciascun layer è compostto da “neuroni".

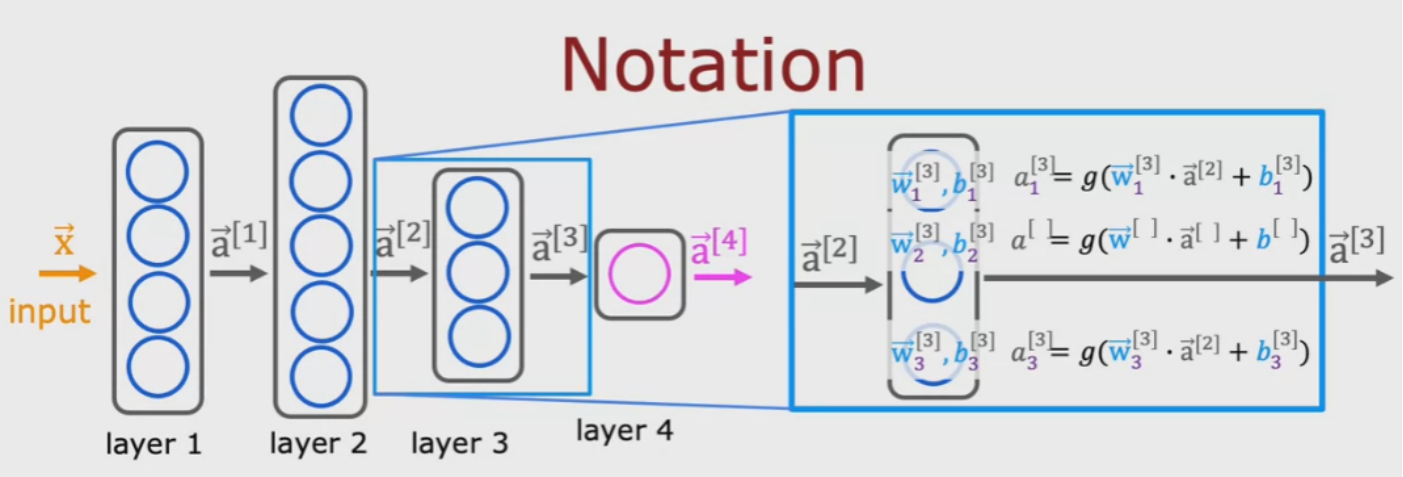
I layers possono essere: 1) input, 2) hidden e 3) output. L'input layer può essere considerato come “hidden”.

Ciascun neurone del layer possiene gli input (o features) e un un output.

L'inferenza di una rete neurale consiste nel ricavare i valori del modello e applicarli alla propria rete

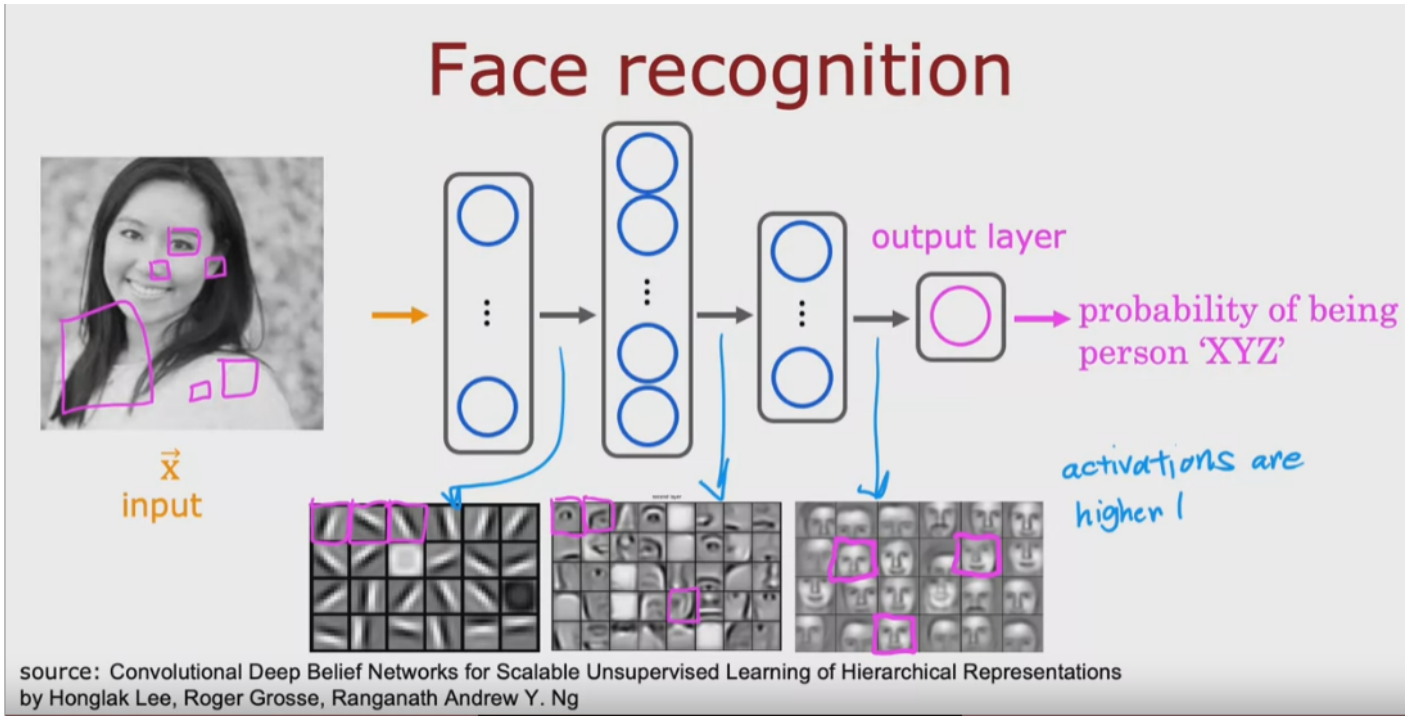
per riprodurre un comportamento. (per valori intendo l'insieme dei w e dell b)

La somma di questi output è anche detta “activations”.

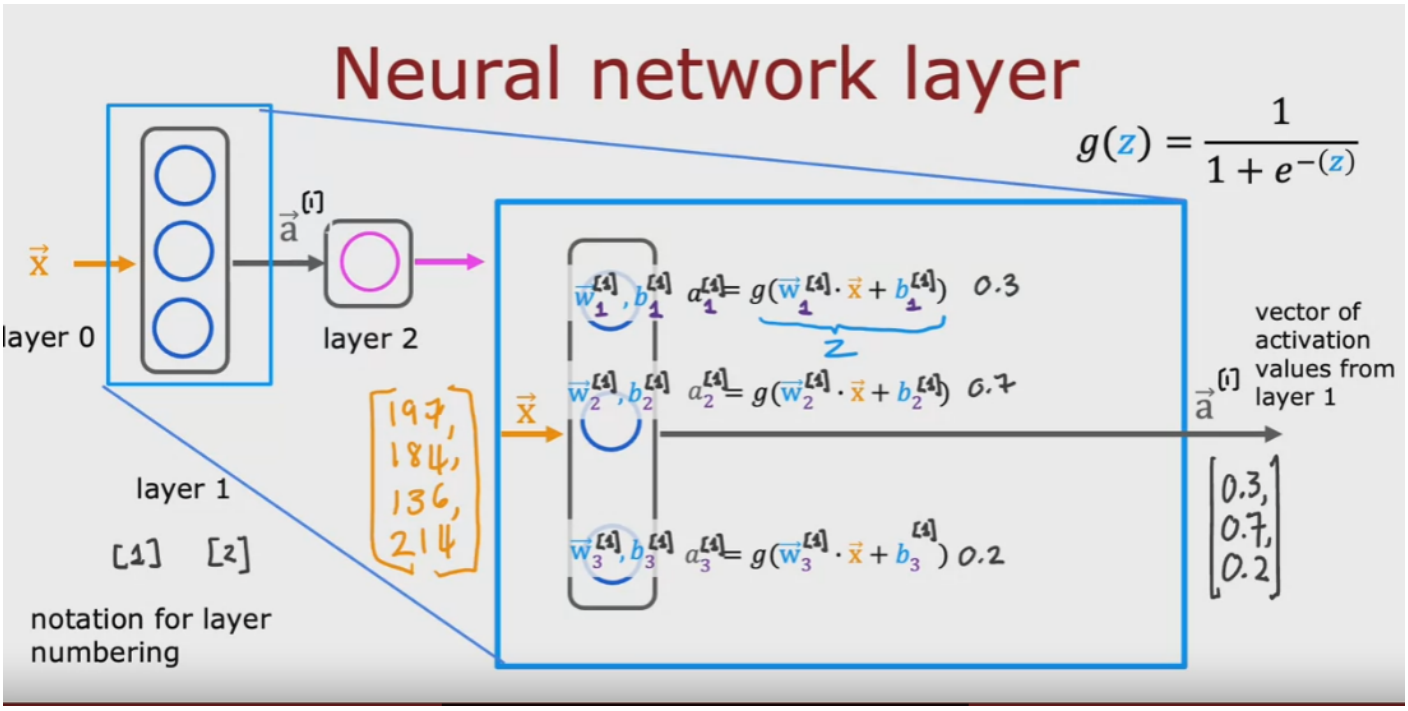


In generale ciascun layer si occupa di “riconoscere” o estrarra specifiche caratteristiche del dataset, nel
caso di immagini di volti, per esempio, il primo layer riconoscere solo righe orizzonati/verticali, il secondo
parti del viso e il terzo un viso completo. (è solo un esempio ovviamente)

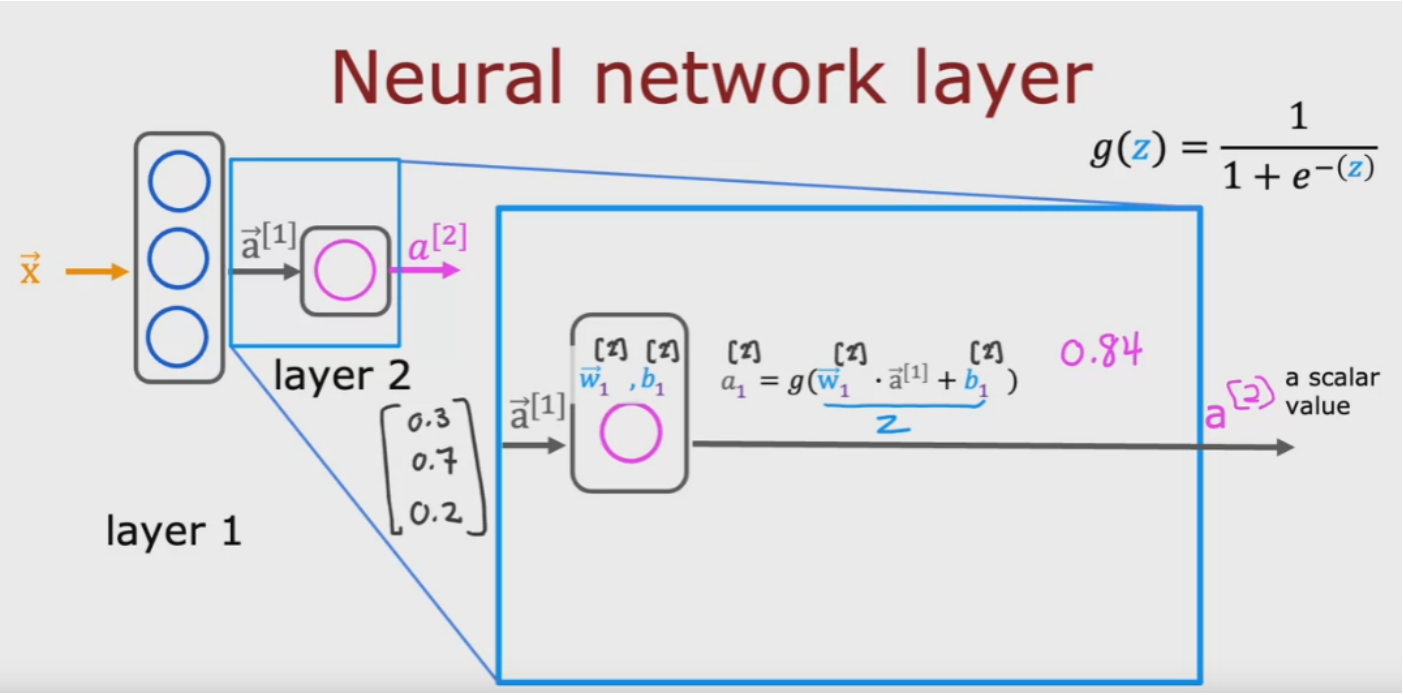
per es:



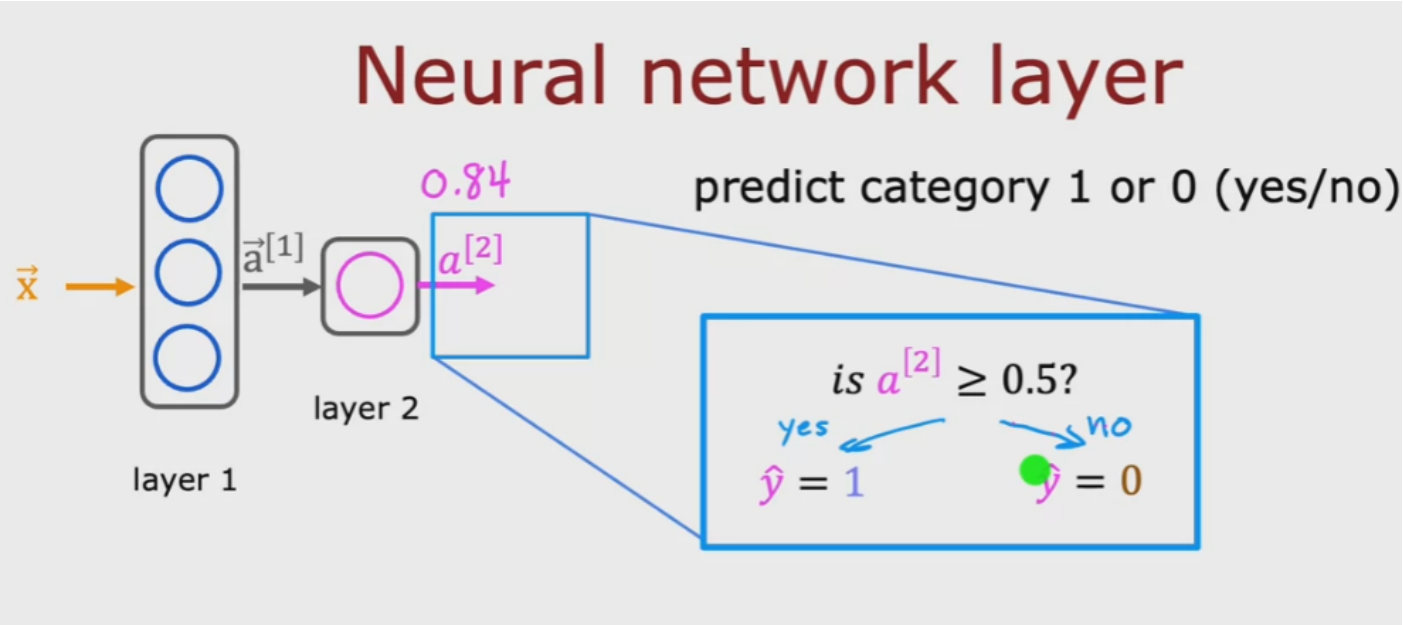
ciascun neurone del layer utilizzata una funzione di attivazione, come per es. la regressione logistica, il cui
output, è il valore di attivazione che diventa l'input del layer successivo:



nel secondo layer infatti il vettore di parametri in input vengono dati in pasto alla funzione di regressione logistica del layer:

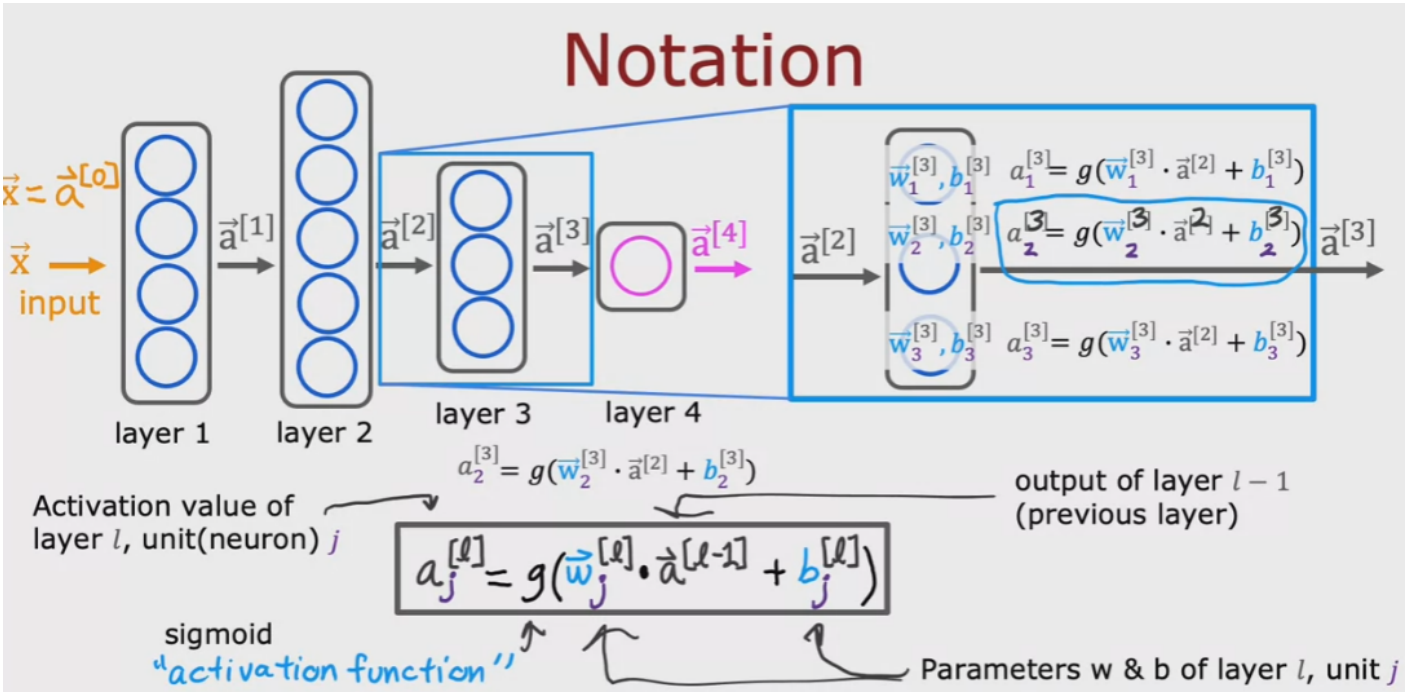


l'output del l'ultimo layer non è più un vettore ma uno scalare (quindi un numero semplice) almeno in questo esempio
al quale viene applicato una soglio per determinare l'output true o false. (è giusto è un esempio, vedi immagine sotto)



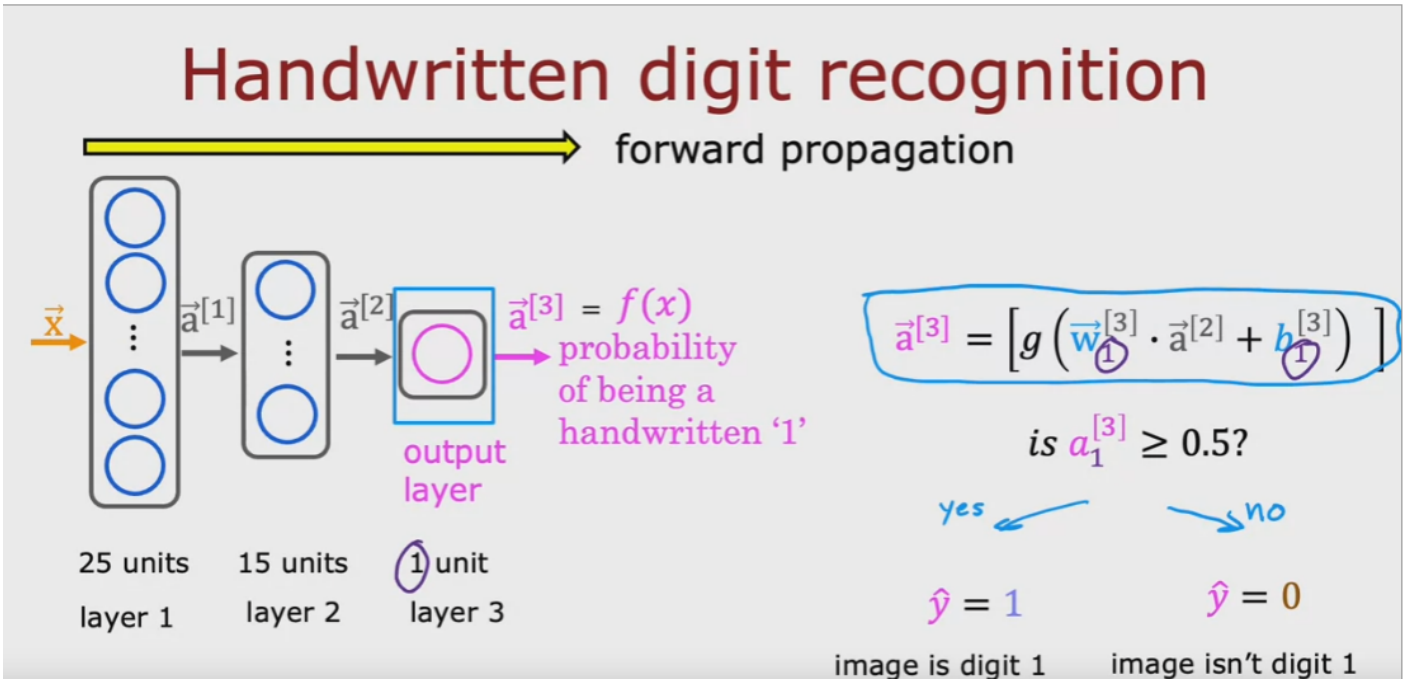
'attivazione di un layer si basa applicando la funzione di attivazione (activation function) che nel caso
specifico è la sigmoid (ma in realtà che ne sono altre migliori) L'output della funzione di attivazione è un vettore di valori che diventa l'input del layer successivo.

Di seguita vengono rappresentati i passaggi per l'elaborazione delle funzioni di attivazioni declinate sui vari layer:



Foward propagation

Quando i valori di attivazioni vanno dall'input layer, passando per l'hidden e terminando nell'output layer seguendo esattamente questo schema, allora parliamo di “forward propagation”. (da sinistra verso destra)



TensorFlow

I framework maggiormente utilizzati sono TF e PyTorch.

Per ora verrà utilizzato TF con layer “densi".

Differenze di rappresentazione dei dati tra Numpy e TF

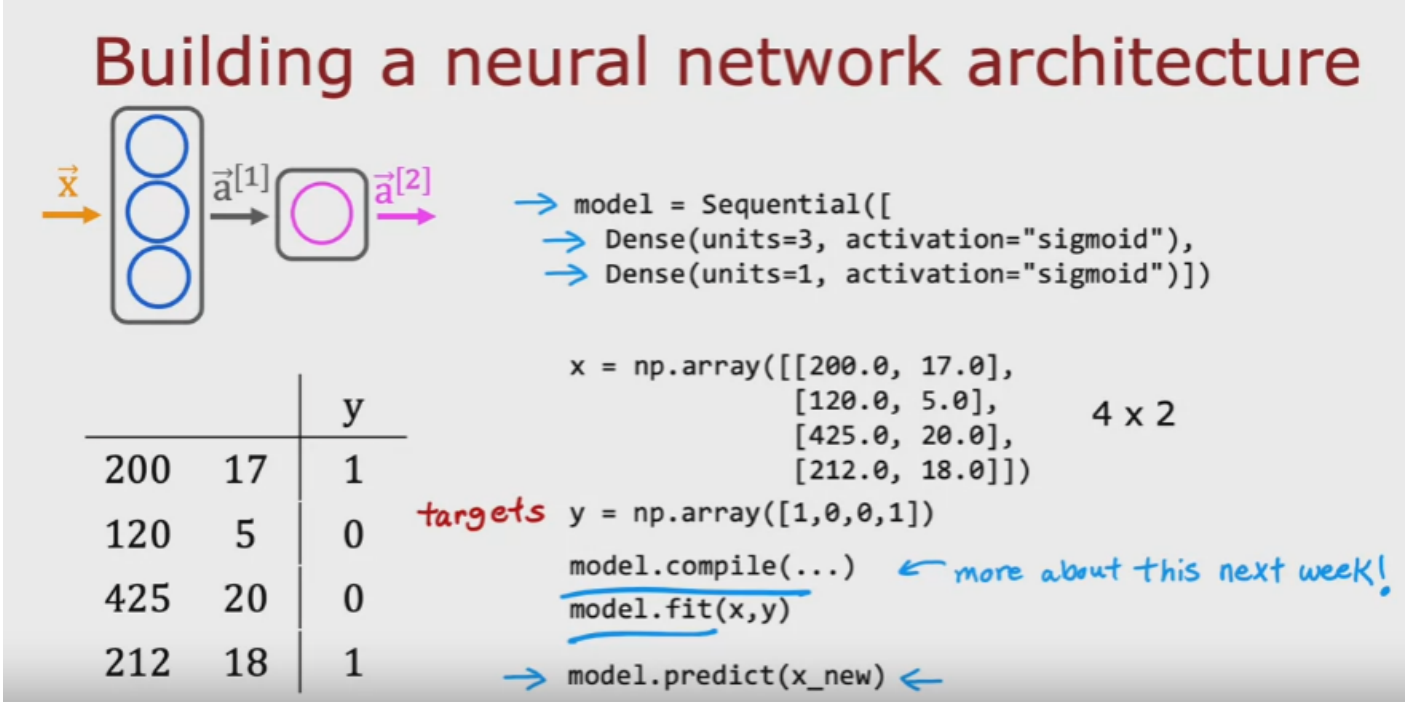
quando passo i dati a TF, es come input ad un Dense layer, TF converte la matrice

di valori np in un “tensore” che nella pratica è un np “wrappato” per esigenze computazionali e di architettura.

E' possibile convertire un tensore un np applicando il medodo .numpy del tensore che rappresenta il dato.

Di seguito viene rappresentata una tipica rete neurale di tipo “forward propagation” con

due layers di 3 e 1 neurone.



COMPILE:

Nella definizione di compile tra i vari paramtri viene impostato il totale delle epoche: es: 40

questo significa che ad ogni epica viene elaborato l'intero dataset. Ogni epoca è suddivisa

in batch dove in TF il numero massimo di batch per epoca è 32.

Quindi per es. se il dataset è fatto di 5000 records (dove ogni record ha N feautues, ma questo non conta adess)

il numero di batch per epoca sarà 5000/32 ovvero 157. es.

.

.

Epoch 39/40

157/157 [==============================] - 0s 2ms/step - loss: 0.0312

Epoch 40/40

157/157 [==============================] - 0s 2ms/step - loss: 0.0294

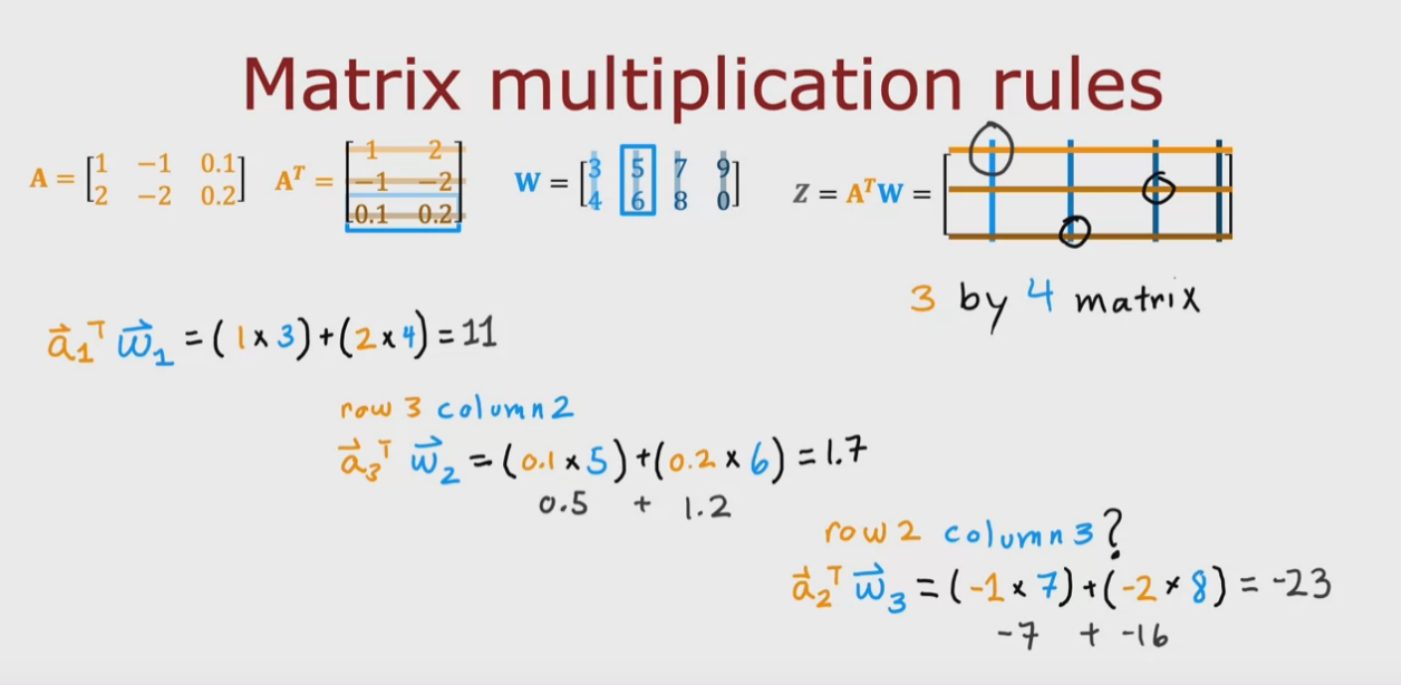
Vettorizzazione

La vettorizzazione corrispoonde alla moltiplacazione delle matrici
Supponendo di avere M1 e M2 per moltiplicare bisogna:
1) fare trasposta di M1
2) moltiplicare la riga 1 per la colonna 1 e via così...

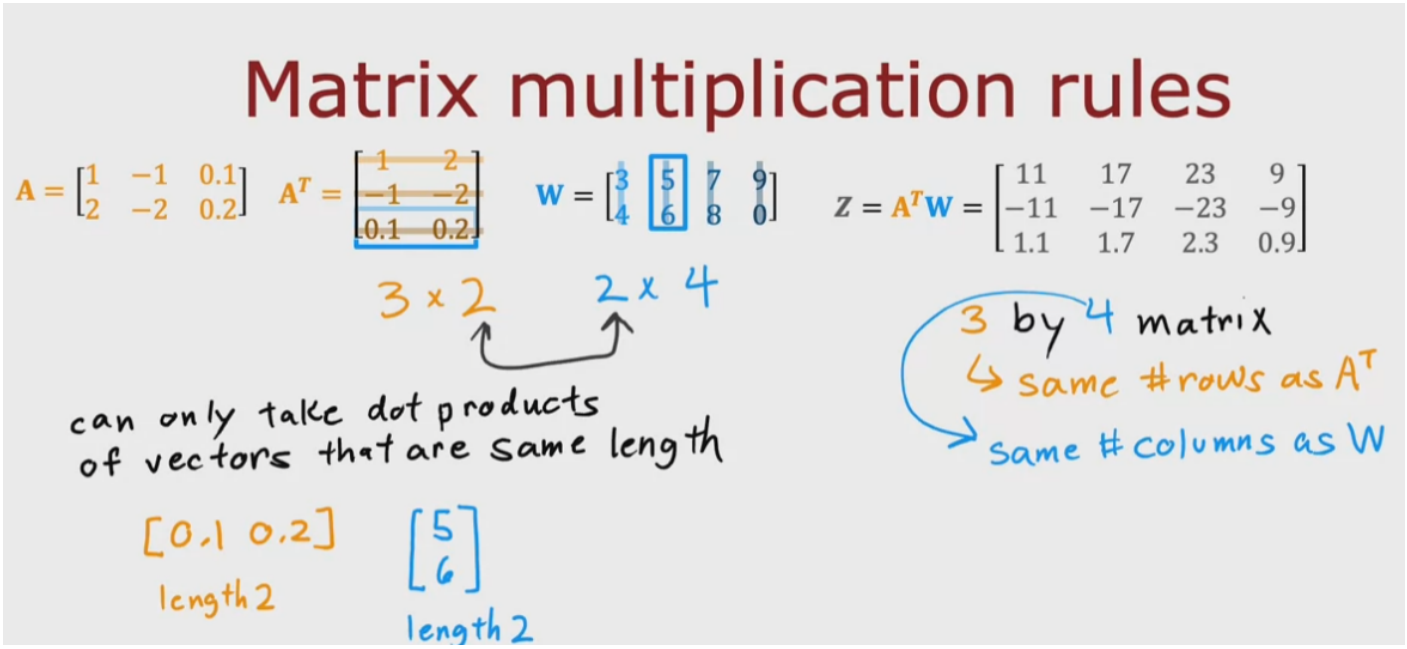
NB:
1) il requisito è che la matrice trasposta M1 per la matrice M2 abbiamo il numero di colonne di M1 uguale al numero di righe di M2
2) inoltre la matrice risultante avrà il numero di righe di M1 trasposta e il numero di colonne di M2

NB2: per fare la trasposta bisogna convertire la collanna (es. 1) nella riga 1.

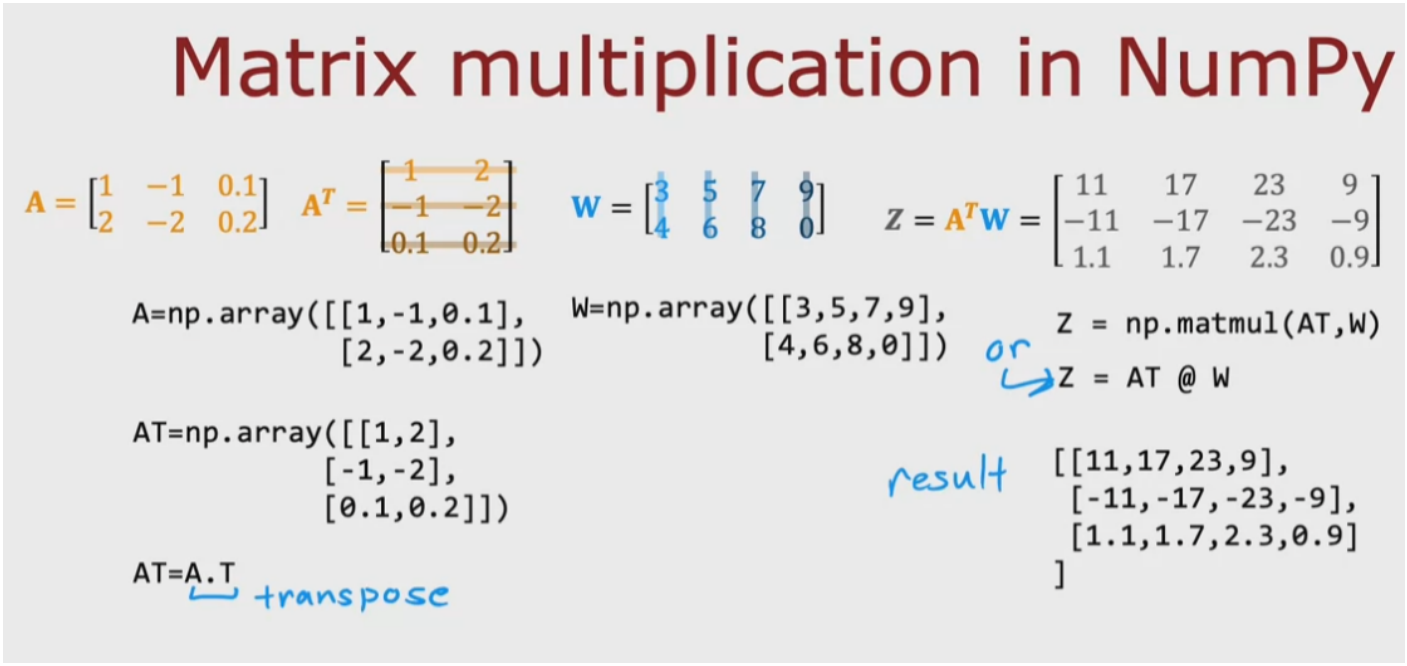
vedi esempio sotto riportato.



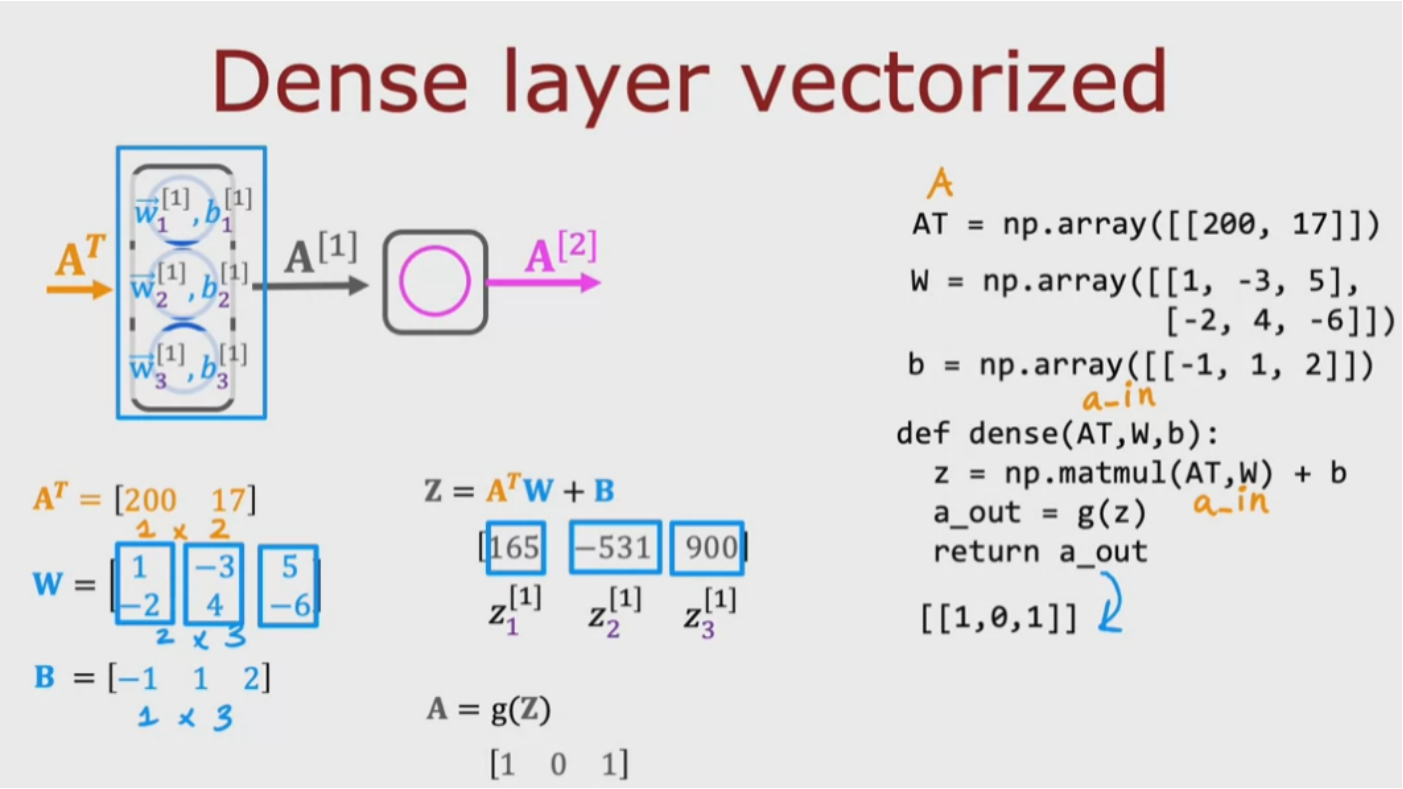
che si converte in:



per effettuare la trasposta di in numpy basta chiare il metodo .T dell'oggetto matrice di np.



che nel codice si traduce:

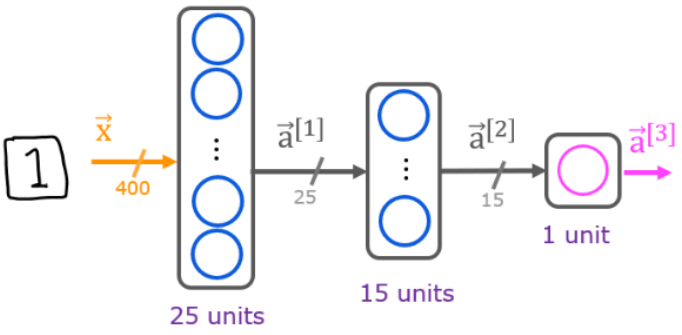


di seguito un esempio di rete nurale sequenziale di 25 neuroni L1, 15 neuroni L2 e 1 neurone L3

2.3 Model representation

The neural network you will use in this assignment is shown in the figure below.

- This has three dense layers with sigmoid activations.
  - Recall that our inputs are pixel values of digit images.
  - Since the images are of size 20×20, this gives us 400 inputs



- The parameters have dimensions that are sized for a neural network with 25 units in layer 1, 15 units in layer 2 and 1 output unit in layer 3.
  - Recall that the dimensions of these parameters are determined as follows:
    - If network has s_{in} units in a layer and s_{out} units in the next layer, then
      - W will be of dimension $s_{in} \times s_{out}$.
      - b will a vector with s_{out} elements
  - Therefore, the shapes of `W` , and `b` , are
    - layer1: The shape of `W1` is (400, 25) and the shape of `b1` is (25,)
    - layer2: The shape of `W2` is (25, 15) and the shape of `b2` is: (15,)
    - layer3: The shape of `W3` is (15, 1) and the shape of `b3` is: (1,)

> **Note:** The bias vector `b` could be represented as a 1-D (n,) or 2-D (n,1) array. Tensorflow utilizes a 1-D representation and this lab will maintain that convention.

con TF è possibile visualizzare il dettaglio del modello, da notare il numero di parametri che corrisponde - per ciascun layer - alle w + le b. Per esempio in primo layer che è composto da 15 neuroni il cui input è un array di 400 valori (sono i 20x20 pixel dell'immagine) avrà un totale di 10025 parametri dato da: 400x25 -> w + 25 -> b

In [10]:
```
# UNQ_C1
# GRADED CELL: Sequential model

model = Sequential(
    [
        tf.keras.Input(shape=(400,)),    #specify input size
        ### START CODE HERE ###
        tf.keras.layers.Dense (units=25, activation='sigmoid'),
        tf.keras.layers.Dense (units=15, activation='sigmoid'),
        tf.keras.layers.Dense (units=1, activation='sigmoid')
        ### END CODE HERE ###
    ], name = "my_model"
)
```

In [38]:
```
model.summary()
```

```
Model: "my_model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_32 (Dense)            (None, 25)                10025

 dense_33 (Dense)            (None, 15)                390

 dense_34 (Dense)            (None, 1)                 16

=================================================================
Total params: 10,431
Trainable params: 10,431
Non-trainable params: 0
_________________________________________________________________
```

nel caso specifico il modello predice la probabilità che il numero si un uno (1)

prediction = model.predict(X[0].reshape(1,400)) # a zero

print(f" predicting a zero: {prediction}")

prediction = model.predict(X[500].reshape(1,400)) # a one

print(f" predicting a one: {prediction}")

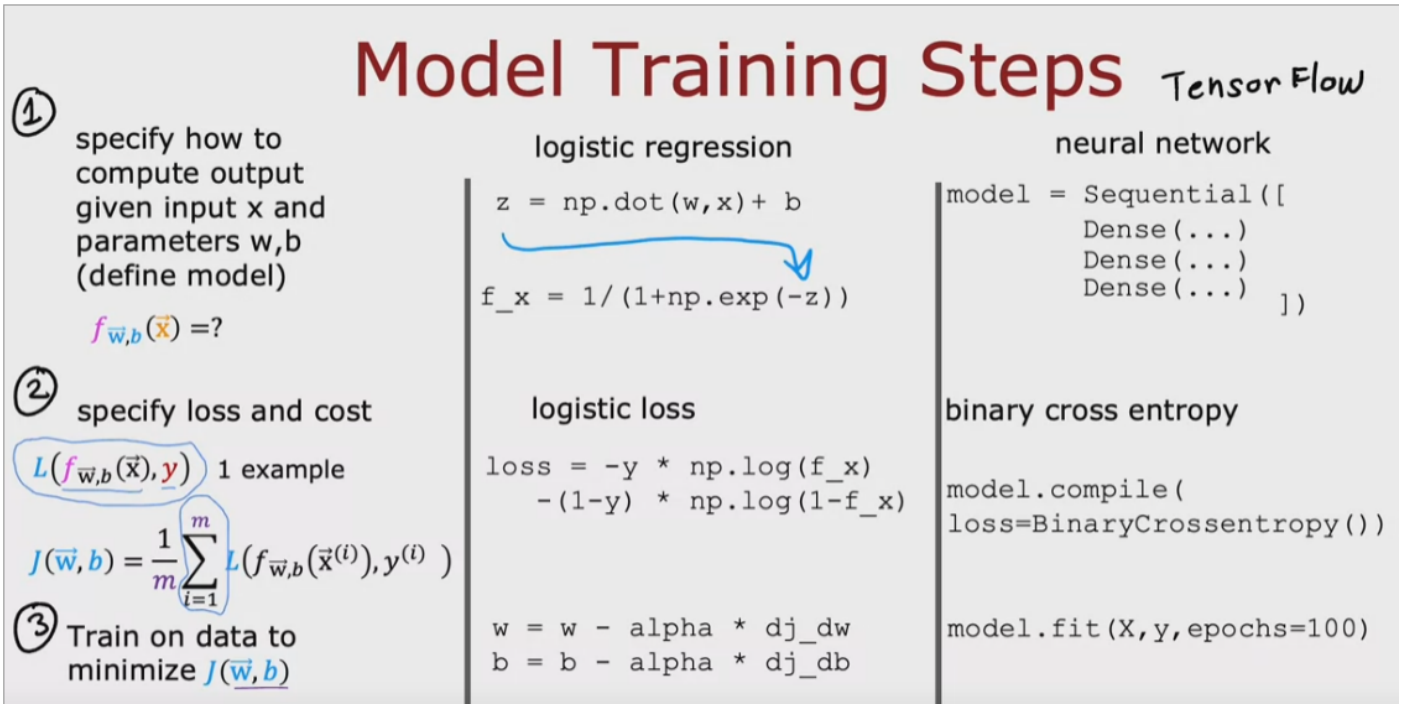
predicting a zero: [[0.0191125]]

predicting a one: [[0.9788295]]

The output of the model is interpreted as a probability. In the first example above, the input is a zero. The model predicts the probability that the input is a one is nearly zero. In the second example, the input is a one. The model predicts the probability that the input is a one is nearly one. As in the case of logistic regression, the probability is compared to a threshold to make a final prediction.

Training

Paragonando i concentti appresi nella prima settimana, costo della funzione, discesa del gradiente con quelli appresi nella seconda settimana, rete neuale , modello, fit possiamo riassumere che:



nel modello la funzione di costo è esplicatata dal parametro “loss” e corrisponde alla funzione alla funziona di costo visto
come media dei valori “loss” (Cost)
mentre il metodo fot non è altro che il calcolo della discesa del gradiente nella regressione logistica.

Funzioni di attivazione

PERCHE' UTILIZZARE LE FUNZIONI DI ATTIVAZIONE?
L'uso delle regressione lineare negli hidden layer non serve in quanto basta
calcolare la regressione lineare a livello di funzione matematica. (vedi settimana 1)
E in questo caso il modello può essere utulizzato solo per modelli molto semplici.

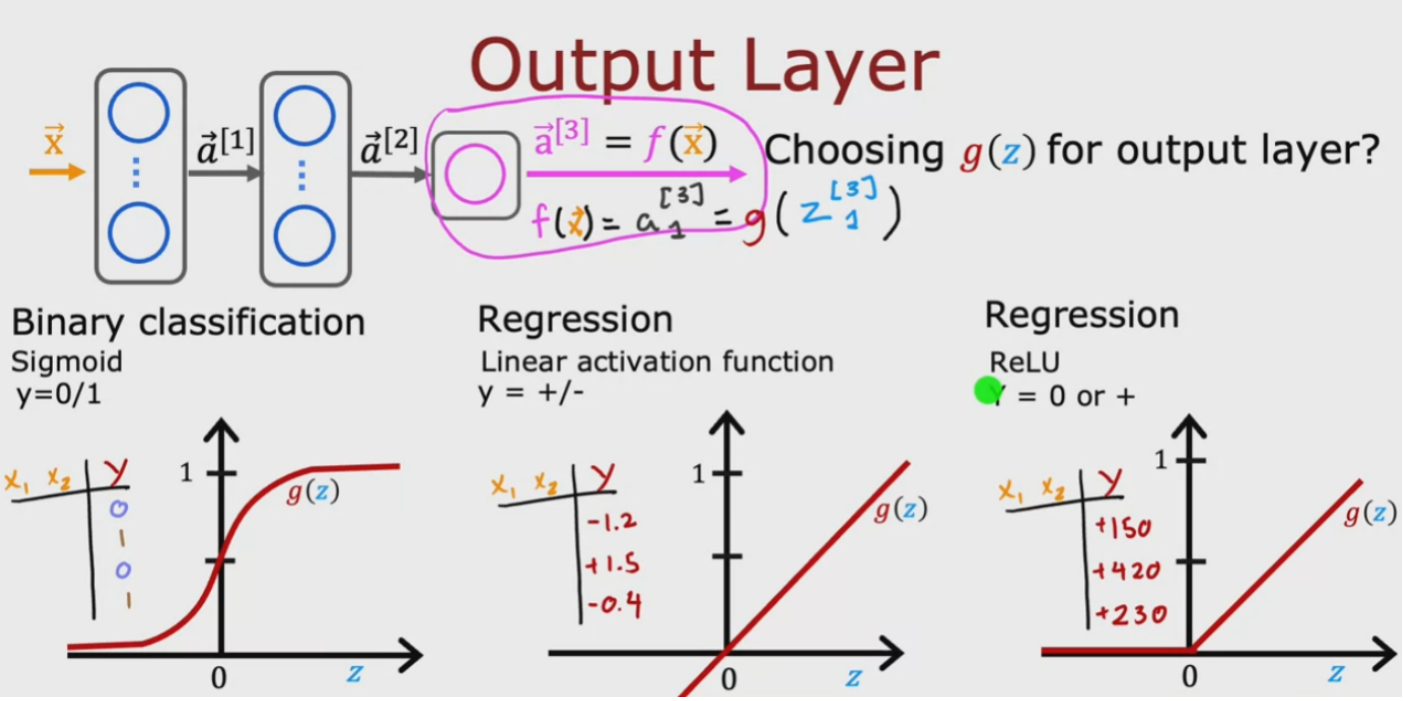
FUNZIONI DI ATTIVAZIONE PER L'OUTPUT LAYER

Ci sono vari tipi di “funziona di attivazione” ciascuna delle quali
una sua peculiarità a seconda del fenomeno che si vuole modellare.

Per problemi legati alla classificazione binaria dove l'output è 0/1 or Tryue/False
la funzione Sigmoid può andar bene. Perchè il modello calcola la probabilità
di ottenere un output uguale a 1.

Nel caso in cui invece si deve predirre un output che può assumere più valori
è meglio utilizzare il tipo “linear regression function”. (per es. nel mercato dei titoli)
In questo caso l'output può quindi assumere sia valori positivi che negativi.

Nel caso in cui invece i valori possono essere varibili ma solo positivi, come per
esempio il prezzo delle case, allora è meglio utilizzare la funzione “ReLU”



FUNZIONE DI ATTIVAZIONE PER GLI HIDDEN LAYER:
in generale per gli hidden è meglio utilzzare “ReLU”

Classificazione multiclasse

Come evidenziato negli esempi precedenti, nel caso in cui si voglio classificare un
evento binario, es. true/false utilizzo la funziona di attivazione Sigmoid.

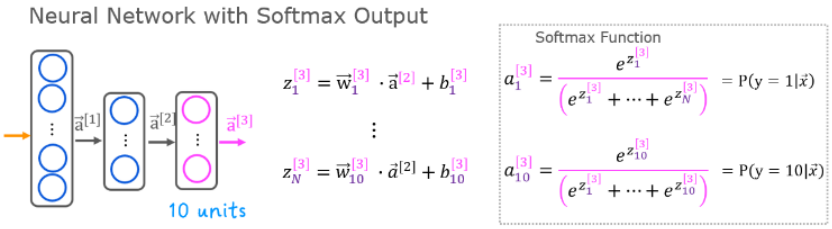
2)
Nel caso di valori lineari, come l'andamento dei prezzi delle case, utilizzo la regressione
lineare seplice. (permane comunque un unico output che può assumere diversi valori)

3)
Nel caso invece in cui ho più valori in output da prevedere, utilizzo la classificazione multiclasse.
In questo caso si utilizza la funzione “SoftMax” che nella pratica rappresente la generalizzazione

della regressione logistica.
In questo caso ogni attivare del layer di output assume la probabilità che lo isesimo valore si verifichi.
La probabilità vare tra zero e uno, ovvero da dallo zero al cento per cento.
La formula per il caloclo della funzione Softmax è:

3 - Softmax Function

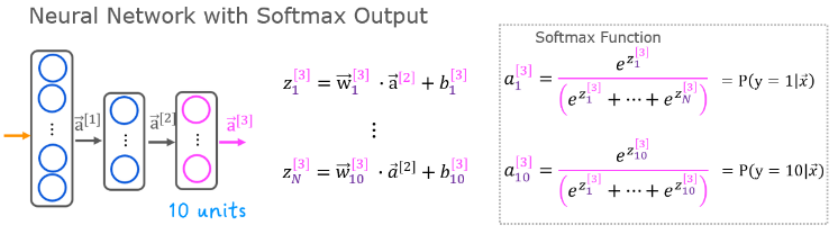
A multiclass neural network generates N outputs. One output is selected as the predicted answer. In the output layer, a vector **z** is generated by a linear function which is fed into a softmax function. The softmax function converts **z** into a probability distribution as described below. After applying softmax, each output will be between 0 and 1 and the outputs will sum to 1. They can be interpreted as probabilities. The larger inputs to the softmax will correspond to larger output probabilities.



che generalizzando si può scrivere come:

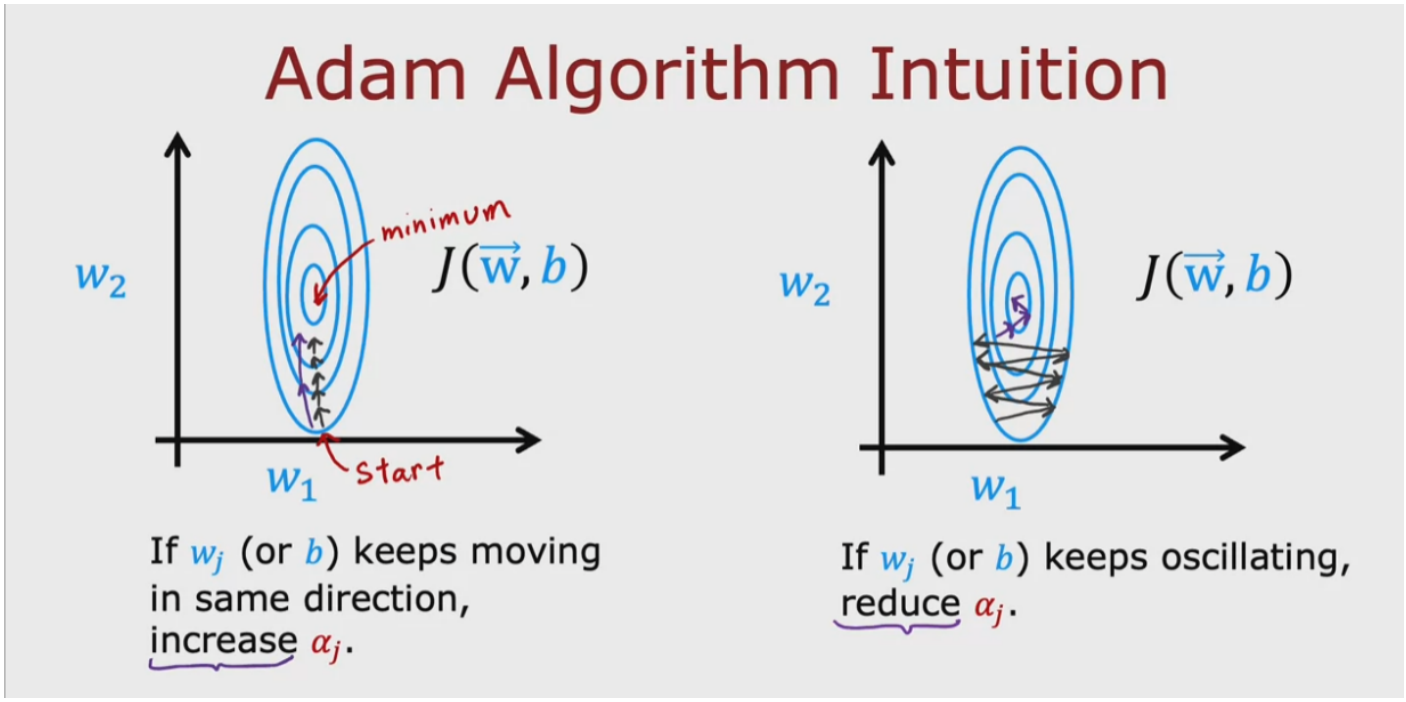
3 - Softmax Function

A multiclass neural network generates N outputs. One output is selected as the predicted answer. In the output layer, a vector **z** is generated by a linear function which is fed into a softmax function. The softmax function converts **z** into a probability distribution as described below. After applying softmax, each output will be between 0 and 1 and the outputs will sum to 1. They can be interpreted as probabilities. The larger inputs to the softmax will correspond to larger output probabilities.



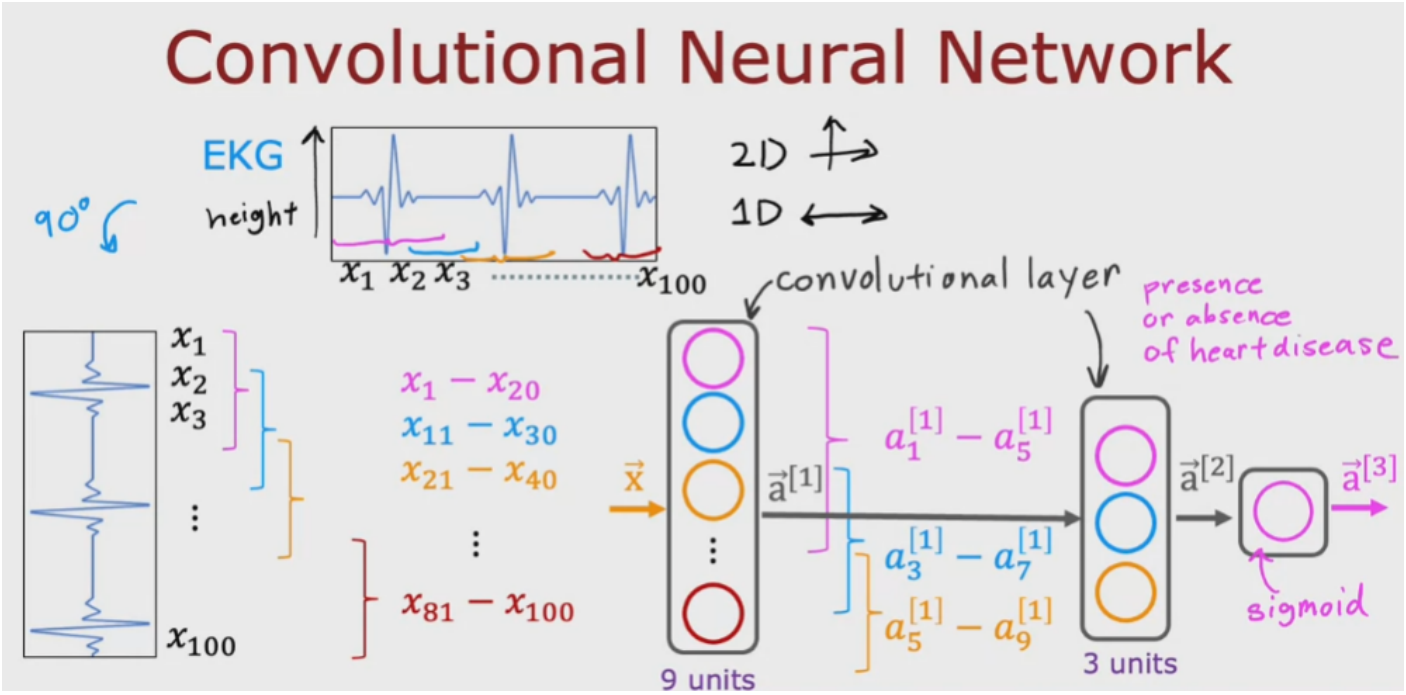
Ottimizzazione avanzata

La discesa del gradiente è un ottimo algoritmo che utilizza
piccoli passi per arrivare al minimo del costo della funzione.
Esiste però un algorirmo più veloce che utilizza degli step “maggiorati”
che fanno in modo di arrivare al minimo più velocemente. Il nome
di questo algoritmo è “Adam”.
Adam sta per “Adaptive Moment Estimation” che nella pratica fa si
che il passo “learning rate” sia variabile ovvero possa variare da piccolo a grande
in maniera ottimale.



Layer "convoluzionale"

Oltre al layer fato di neuroni “densi” esite il layer fatto di neuroni “convoluzionari” che a differenze dei densi, ricevano solo alcune attivazioni del layer precedente.
es.



Bias validation

Nell'ambuito della fase di training del modello, per valutare l'efficacia utilizziamo la midura dell'errore relativamente a:
1) il trainset, in genere il 60% dei dati da dare in pasto al modell
2) il cross validation set, il 20% dei dati

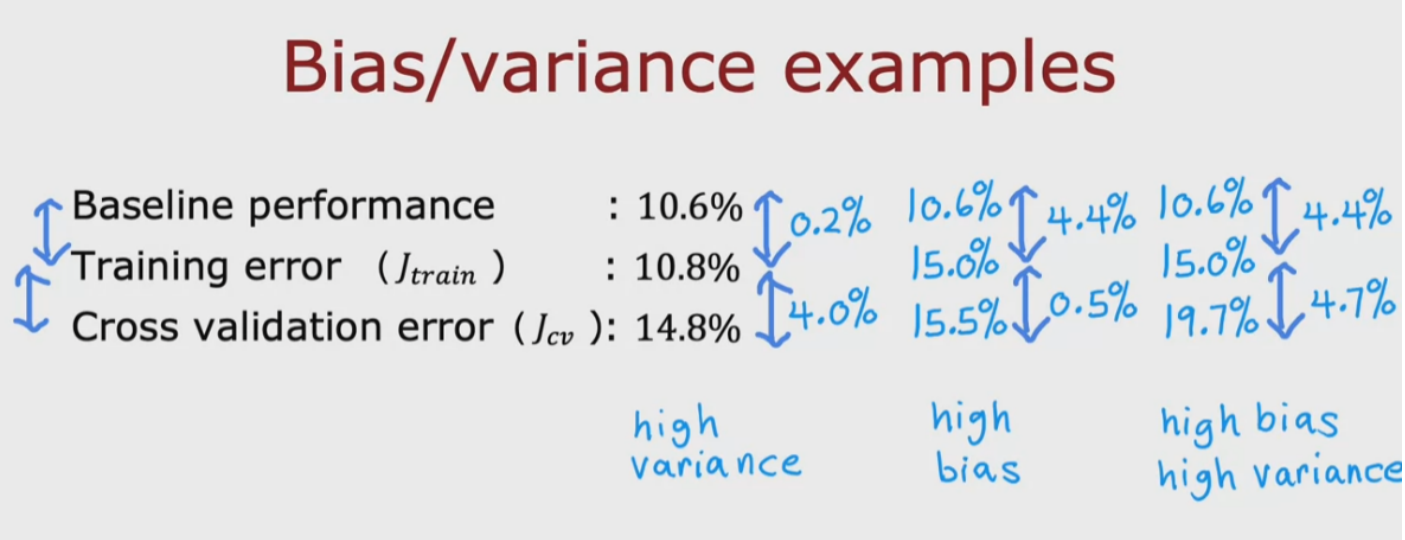
3) il test set, restante 20 %

L'errore relativo al trainset è detto “bias error" che se è alto significa che il modello è stato trainito male e quindi bisogna cambiarlo, se è basso puà essere che sia in overfitting oppure che vada effettivamente bene

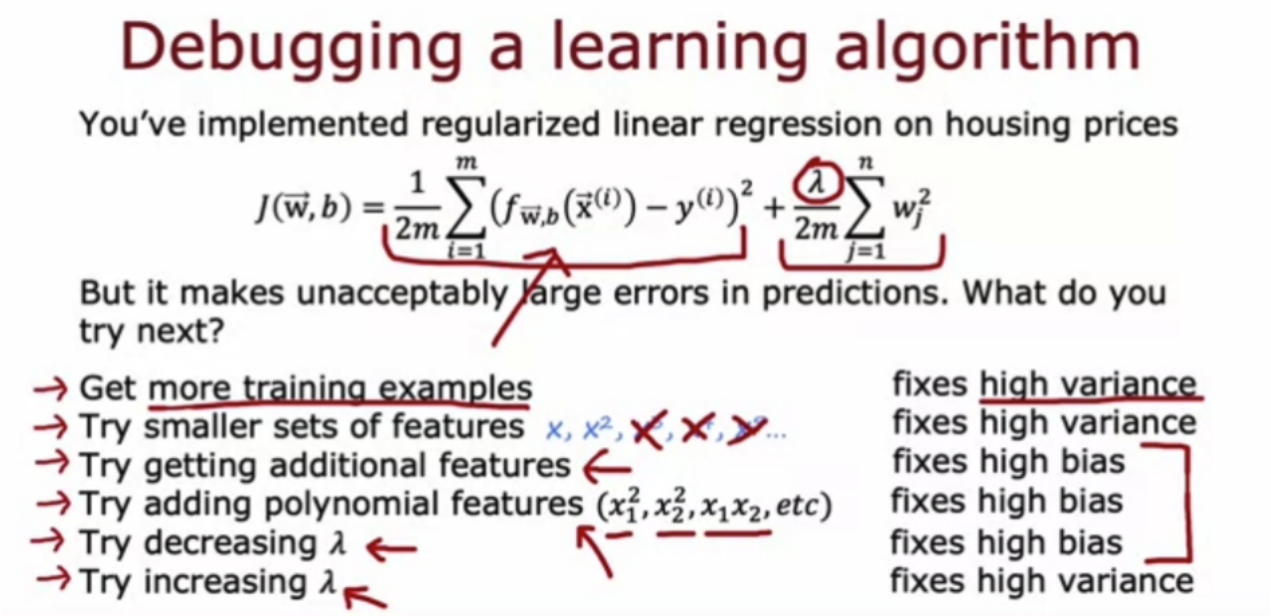
L'errore relativo al cross validation indica quanto si discosta il modello utilizzando dei dati che non sono stati utilizzati nella fase di fitting.

La baseling è il riferimento preso da altri modelli diversi da quello che si sta elaborando.

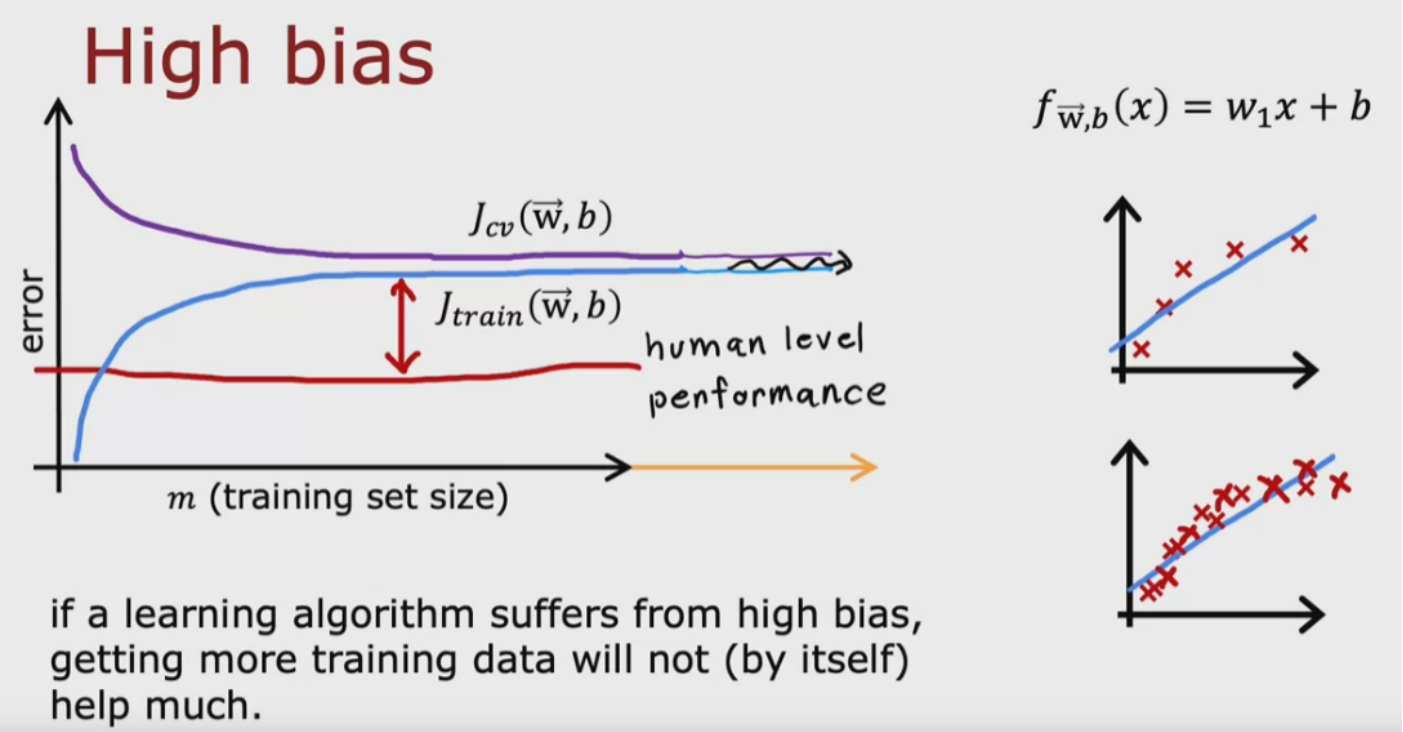
Si possono quindi verificare i seguenti casi:



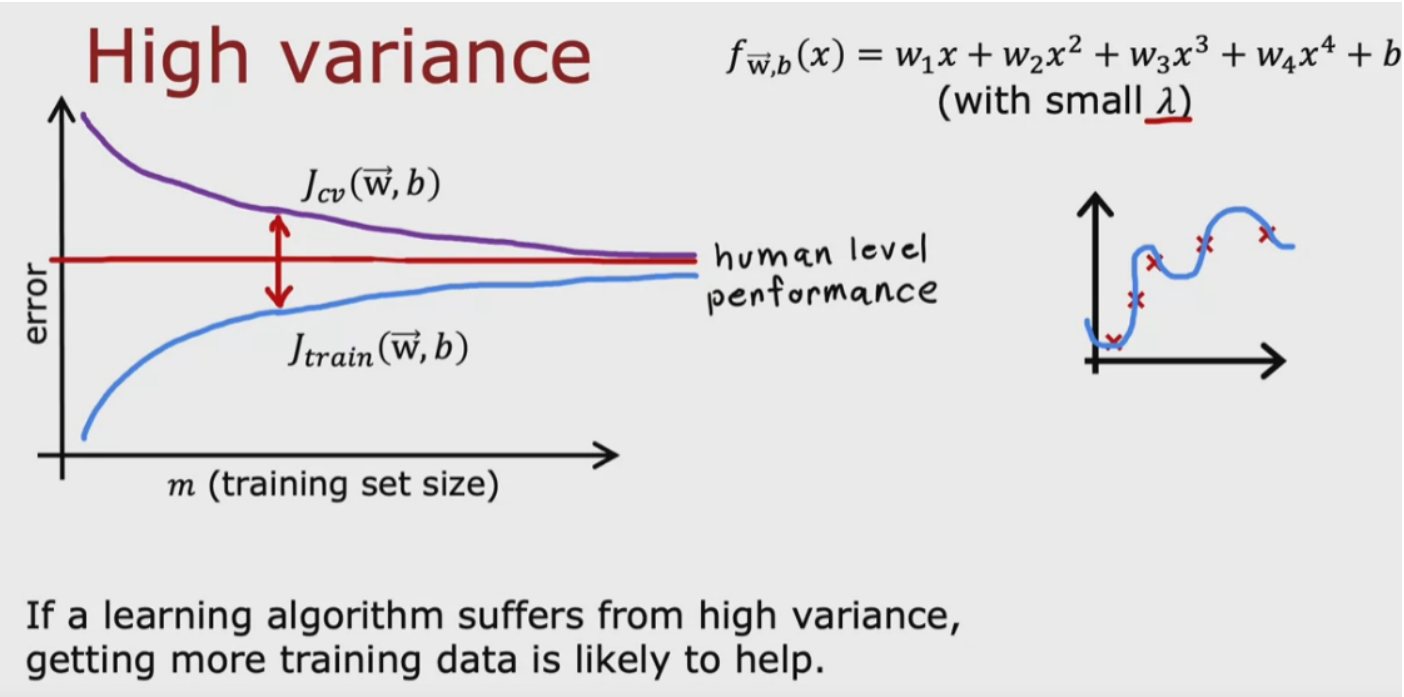
Per migliorare il training del modello ci sono alcuni accorgimenti da seguire, di seguito:



Nel grafico sottoriportato si può vedere il caso in cui la differenza tra il modello e il riferimento si discosta di molto in peggio, in questo
caso all'aumentare del numero del trainset la curva tende ad appiattirsi e non miglioare. Il modello quindi ma ba bene

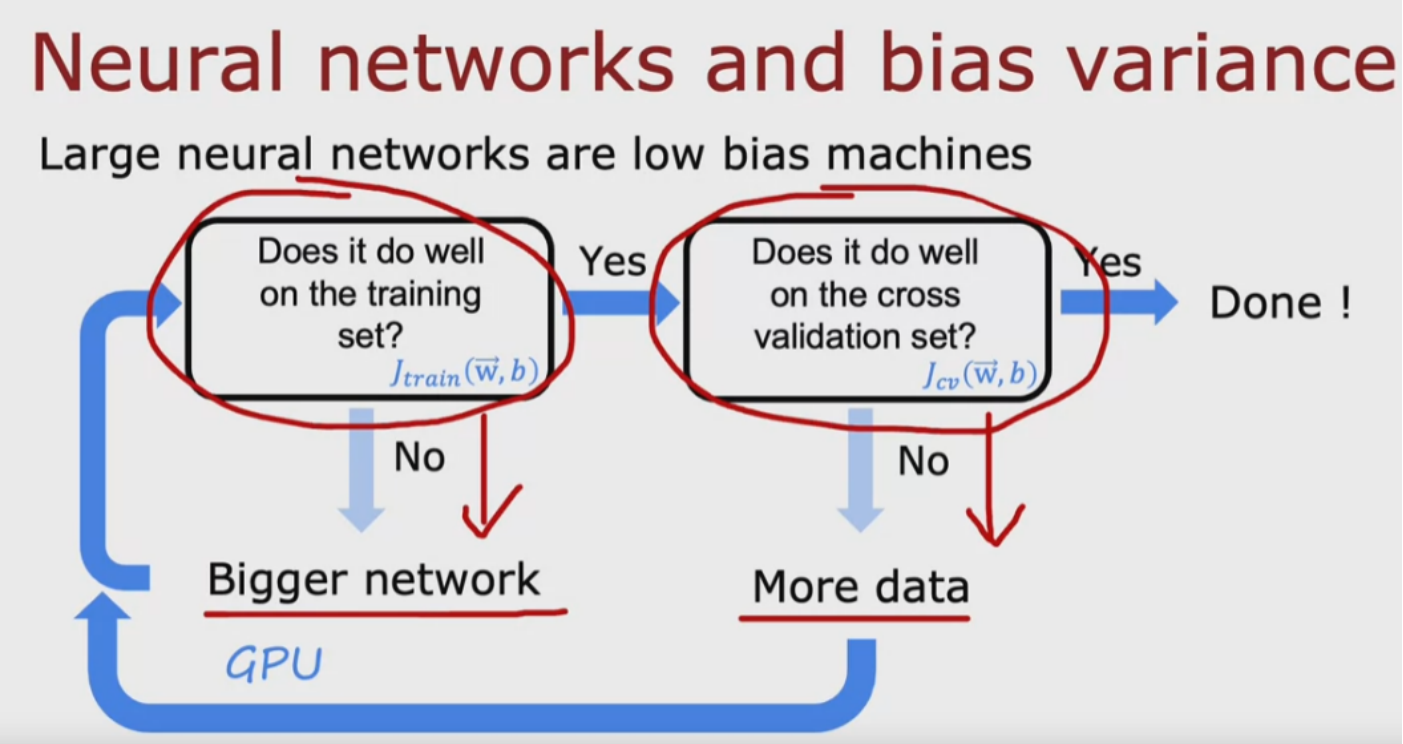


Nel caso invece in qui il trainset performi meglio del riferimento, può essere che all'aumentare dei samples l'errore
di cross validation diminuisca e arriva al livello del riferimento.



In generale trovare bisogna cercare di diminuire i valori di bias e variance attraverso un processo di affinamento

come sotto rappresentato:



lynTrymo92ZvbyNy-image.png

Revision #5
Created 2023-03-11 16:43:24 UTC by marco
Updated 2024-10-13 15:09:54 UTC by marco