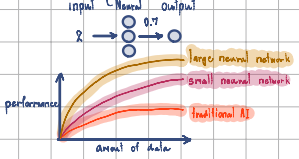
Neural networks

Origins: Algorithms that try to mimic the brain

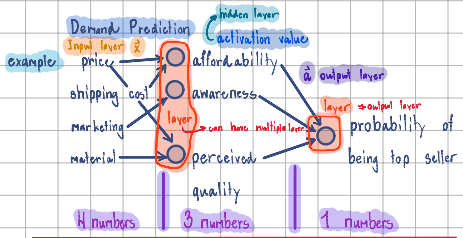
speech → images → text (Neural Language Program) → anything

Simplified mathematical model of a neuron

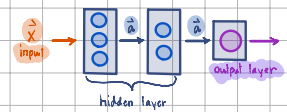
input → Neural (just one neuron) → output



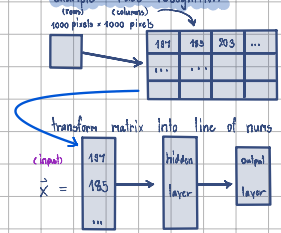
Demand Prediction



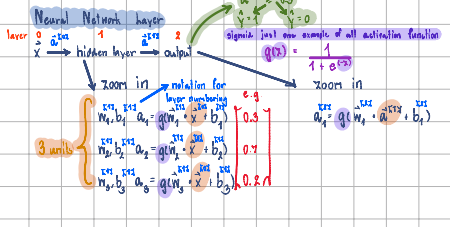
summary: It just logis regression, they can learn its own features like "feature engineering but Neural Networks it doing by yourself that make it easier to accurate predictions



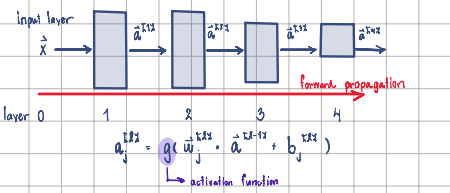
example Face recognition



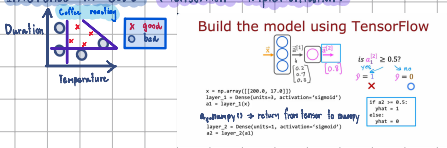
Neural Network layer



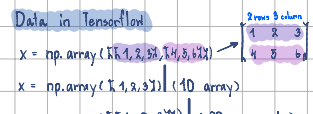
More complex neural networks



Inference in Code (Tensorflow implementation)



Data in Tensorflow

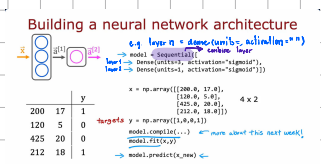


x = np.array([[1,2,3,4,5,6]]) (2D array)

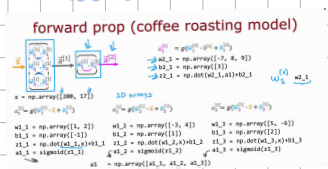
x = np.array([1,2,3]) (1D array)

x = np.array([[1,2,3]]) (2D row vector)

Building a neural network

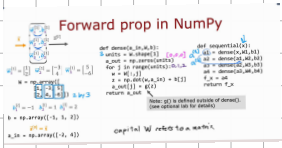


Forward prop in a single layer



General implementation of forward propagation

Forward prop in Numpy

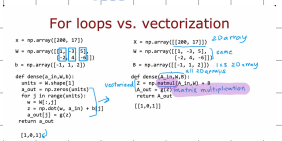


Speculations on artificial general intelligence (Is there a path to AGI?)

AI
- ANI (artificial narrow intelligence) (Specific task)
- AGI (artificial general intelligence) do anything a human can do

Neural networks + Vectorization

For loops vs. vectorization



Matrix multiplication

np.matmul(,) or A.T @ W

Dot product $\begin{bmatrix}1\\2\end{bmatrix}\cdot\begin{bmatrix}3\\4\end{bmatrix} \Rightarrow (1\cdot3)+(2\cdot4) \Rightarrow 11$

$\vec{a}\cdot\vec{w} \Rightarrow \vec{a}^T\vec{w}$

transpose $\vec{a} = \begin{bmatrix}1\\2\end{bmatrix}$ $\vec{a}^T = \begin{bmatrix}1&2\end{bmatrix}$

$z = (\vec{a}^T)(\vec{w}) = (\vec{a})(\vec{w})$

$1\times2 \quad 2\times3 \quad \begin{bmatrix}1&4&2&5&3&6\end{bmatrix}$

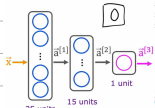
$\vec{a}^T = \begin{bmatrix}1&2\end{bmatrix}$, $\vec{w} = \begin{bmatrix}3&4&5\\6&1&2\end{bmatrix}$

$z = \begin{bmatrix}1\cdot3+2\cdot10 & 3+12 & 4+15\end{bmatrix} = \begin{bmatrix}23 & 15 & 19\end{bmatrix}$

$(-7)+(-16)$

Tensorflow implementation

Train a Neural Network in TensorFlow



Given set of (x,y) examples
How to build and train this in code?

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
model = Sequential([                                          ①
    Dense(units=25, activation='sigmoid'),
    Dense(units=15, activation='sigmoid'),
    Dense(units=1, activation='sigmoid'),
    ])
from tensorflow.keras.losses import BinaryCrossentropy
model.compile(loss=BinaryCrossentropy())                      ②
model.fit(X,Y,epochs=100)                                     ③
```

epochs: number of steps in gradient descent

Training Details

define the architecture

1 specify how to compute output given input x and parameters w,b

$f_{\vec{w},b}(\vec{x}) = ?$ | logistic regression `z = np.dot(w,x) + b`, `f_x = 1 / (1 + np.exp(-z))`

neural network `model = Sequential([Dense(...), Dense(...)])`

2 specify loss and cost

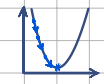
$L(f_{\vec{w},b}(\vec{x}), y)$ | $J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m} L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})$

logistic loss `loss = -y * np.log(f_x) - (1-y) * np.log(1-f_x)`

binary cross entropy `model.compile(loss = BinaryCrossentropy())` (can change loss function to another)

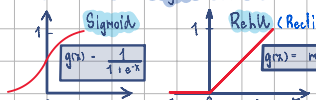
3 train on data to minimize $J(\vec{w},b)$

logistic loss $w = w - \alpha \frac{dJ}{dw}$, $b = b - \alpha \frac{dJ}{db}$



binary cross entropy `model.fit(X, Y, epochs=100)`

Alternatives to the sigmoid activation



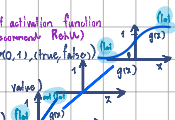
Choosing activation functions

(output can be any of activation function but hidden layer recommend ReLU)

sigmoid ⇒ binary classification (e.g. output 0,1) (true, false)
(Don't use in hidden layer)

Linear activation function ⇒ regression (e.g. +/- value)

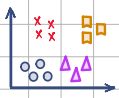
ReLU ⇒ only non negative value (e.g. price)
(most common choice) (ReLU faster than sigmoid)



Why do we need activation function

⇒ if it not have activation function | learn more complex is just a series of linear regression

Multiclass ⇒ more than two possible values



Softmax ⇒ generalization of logistic regression to the multiclass

example Softmax regression (N possible outputs)

$z_1 = \vec{w}_1 \cdot \vec{x} + b_1$ $\quad$ $z_2 = \vec{w}_2 \cdot \vec{x} + b_2$

$$a_1 = \frac{e^{z_1}}{e^{z_1}+e^{z_2}+\dots+e^{z_N}} = P(y=1|\vec{x}) \qquad a_2 = \frac{e^{z_2}}{e^{z_1}+e^{z_2}+\dots+e^{z_N}} = P(y=2|\vec{x})$$

$z_3 = \vec{w}_3 \cdot \vec{x} + b_3$ $\quad$ $z_N = \vec{w}_N \cdot \vec{x} + b_N$

$$a_3 = \frac{e^{z_3}}{e^{z_1}+e^{z_2}+\dots+e^{z_N}} = P(y=3|\vec{x}) \qquad a_N = \frac{e^{z_N}}{e^{z_1}+e^{z_2}+\dots+e^{z_N}} = P(y=N|\vec{x})$$

(2 possible output) Logistic regression	(N possible output) Softmax regression
$z = \vec{w} \cdot \vec{x} + b$	$z_j = \vec{w}_j \cdot \vec{x} + b_j \; ; \; j = 1,2,\dots,N$
$a_1 = g(z) = \frac{1}{1+e^{-z}} = P(y=1\|\vec{x})$	$a_j = \frac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}} = P(y=j\|\vec{x})$
$a_2 = 1 - a_1 = P(y=0\|\vec{x})$	

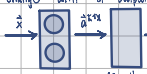
Cost Function for Softmax regression

$$a_N = \frac{e^{z_N}}{e^{z_1}+e^{z_2}+\dots+e^{z_N}}$$

$$loss(a_1,\dots,y) = \begin{cases} -\log a_1 & \text{if } y=1 \\ \vdots & \\ -\log a_n & \text{if } y=n \end{cases}$$

Neural Network with Softmax output

change unit of output layer equal to classes



$$\vec{a}^{[2]} = (a_1^{[2]},\dots,a_{10}^{[2]}) = g(z_1^{[2]},\dots,z_{10}^{[2]})$$

Improved implementation of softmax

Numerical Roundoff Errors e.g. $\frac{2}{10000} \neq \left(1+\frac{1}{10000}\right) - \left(1-\frac{1}{10000}\right)$

↳ more numerically accurate implementation of logistic loss

Numerical Roundoff Errors

More numerically accurate implementation of logistic loss: $1+\frac{1}{10,000}$ $\quad$ $1-\frac{1}{10,000}$

Logistic regression:

$$a = g(z) = \frac{1}{1+e^{-z}}$$

```
model = Sequential([
    Dense(units=25, activation='relu'),
    Dense(units=15, activation='relu'),
    Dense(units=1, activation='sigmoid')
])
```

Original loss

$$loss = -y\log a - (1-y)\log(1-a)$$

```
model.compile(loss=BinaryCrossEntropy() )
```

More accurate loss (in code)

$$loss = -y\log\left(\frac{1}{1+e^{-z}}\right) - (1-y)\log\left(1-\frac{1}{1+e^{-z}}\right)$$

```
model.compile(loss=BinaryCrossEntropy(from_logits=True) )
```

logit: z

More numerically accurate implementation of softmax

Softmax regression

$$(a_1,\dots,a_{10}) = g(z_1,\dots,z_{10})$$

$$Loss = L(\vec{a},y) = \begin{cases} -\log a_1 & \text{if } y=1 \\ \vdots & \\ -\log a_{10} & \text{if } y=10 \end{cases}$$

```
model = Sequential([
    Dense(units=25, activation='relu'),
    Dense(units=15, activation='relu'),
    Dense(units=10, activation='softmax')
])
```

'linear'

```
model.compile(loss=SparseCategoricalCrossEntropy() )
```

More Accurate

$$L(\vec{a},y) = \begin{cases} -\log\frac{e^{z_1}}{e^{z_1}+\dots+e^{z_{10}}} & \text{if } y=1 \\ \vdots & \\ -\log\frac{e^{z_{10}}}{e^{z_1}+\dots+e^{z_{10}}} & \text{if } y=10 \end{cases}$$

```
model.compile(loss=SparseCategoricalCrossEntropy(from_logits=True))
```

logistic regression (more numerically accurate)

```
model   model = Sequential([
          Dense(units=25, activation='sigmoid'),
          Dense(units=15, activation='sigmoid'),
          Dense(units=1, activation='linear')
        ])
loss    from tensorflow.keras.losses import
        BinaryCrossentropy
        model.compile(..., BinaryCrossentropy(from_logits=True))
fit     model.fit(X,Y,epochs=100)
predict logit = model(X)        z
        f_x = tf.nn.sigmoid(logit)
```

Classification with multiples outputs

⇒ create specific Neural Network for single output and combine them or create it in one neural network with multiples outputs

Advanced Optimization ⇒ another algorithm not just gradient descent

- Adam algorithm (Adaptive Moment estimation) ⇒ not just one α (learning rate)

If w_j and b keeps moving in same direction ⇒ increase α (local minimum)

If w_j and b keeps oscillating ⇒ reduce α (local minimum)

MNIST Adam

model

```
model = Sequential([
        tf.keras.layers.Dense(units=25, activation='sigmoid'),
        tf.keras.layers.Dense(units=15, activation='sigmoid'),
        tf.keras.layers.Dense(units=10, activation='linear')
])
```

compile

$\alpha = 10^{-3} = 0.001$

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
```

fit

```
model.fit(X,Y,epochs=100)
```

Additional layer Types

- Dense layer ⇒ each neurons output is a function of all the activation outputs of the previous layer

- Convolution layer ⇒ faster computation, need less training data (different part of the previous layer's output) ⇒ only looks at part of the input

Tips

Deciding what to try next

- Get more training examples
- Try smaller sets of feature
- Try getting additional features
- Try adding polynomial features (x_1^2, x_2^2, x_1, etc)
- Try decrease / increase learning rates

Evaluating a model

⇒ seperate dataset into training set and test set

compute test error → test / train

$$J_{test}(\vec{w},b) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} \left(f_{\vec{w},b}(\vec{x}^{(i)}_{test}) - \vec{y}^{(i)}_{test} \right)^2$$

Model selection and training / cross validation / test sets

Model selection

$f_{\vec{w},b}(\vec{x}) = w_1 x + b$

$f_{\vec{w},b}(\vec{x}) = w_2 x + w_2 x^2 + b$

$\vdots$

validation set / dev set / development set

training / cross validation / test sets

training set ⇒ 60 %

cross validation ⇒ 20 %

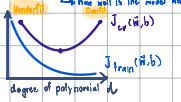
test set ⇒ 20 %

Diagnosing bias and variance

underfit ⇒ high bias (cost function) J_{train} ⇒ high, J_{cv} ⇒ high (curve satisfaction)

overfit ⇒ high variance J_{train} ⇒ low, J_{cv} ⇒ high

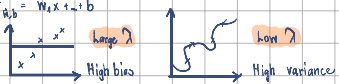
⇒ how well is the model work with new data



Regularization and bias / variance

$$J(\vec{w},b) = \frac{1}{2m} \sum_{i=1}^{n} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right)^2 + \frac{\lambda}{2m} \sum_{j=1}^{n} w_j^2$$

$f_{\vec{w},b} = w_1 x + \dots + b$



How to choosing the regularization parameter λ

Try λ from low to high

Establishing a baseline level of performance

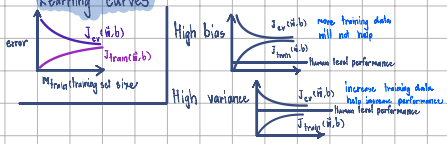
⇒ compare Training error (J_{train}), cross Validation (J_{cv}) to Human level performance or competing algorithms performance

example

Baseline performance	⇒	10.6 %	10.6 %
Training error (J_{train})	⇒	10.8 %	15.0 %
Cross Validation error (J_{cv})	⇒	14.8 %	15.5 %
		high variance	high bias

(0.2% / 4.4% / 4.0% / 0.5%)

learning curves



Deciding what to try next revisited

- Get more training examples
- Try smaller sets of features
- Try Increase λ

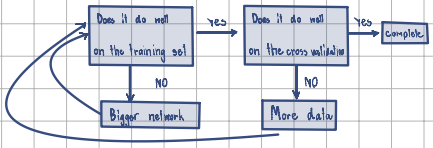
} fix high variance

- Try getting additional features
- Try adding polynomial features
- Try decrease λ

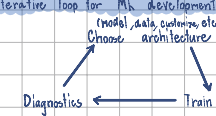
} fix high bias

Bias / Variance and neural networks

Large neural networks are low bias machines



Iterative loop for ML development



Adding data → add more data of the types where error analysis has indicated

Data augmentation → modifying an existing example to create new training example

e.g. original sound + noise

Data synthesis (use next in computer vision) → using artificial data inputs to create a new training example

e.g. change type of font

Engineering the data used by your system

1) Conventional model-centric approach (Improve code)

2) Data-centric approach (Improve data)

AI = Code + Data

Transfer learning: using data from a different task

(using pretrained model)

supervised pretraining → train network with large data

fine tuning → train the network on your own data

2 option

→ 1) only train output layers parameters (Small training set)

2) train all parameters (large training set)

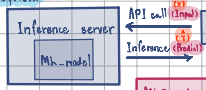
Full cycle of a machine learning project

Scope project → collect data → train model (error analysis) → deploy in production

Iterative improvement

monitor and maintain system

Deployment



MLOps (Machine learning operations)

Error metrics for skewed datasets

99.1% or 99.2% or 99.5% (What is the best?) → how to find best algorithm

(good prediction or useful prediction)

Precision / recall

$y = 1$ in presence of rare class

		actual class 1	actual class 0
predicted class	1	15 (True positive)	5 (False positive)
	0	10 (False negative)	70 (True negative)
		25	75

Precision → check how much that model predicted

$\frac{\text{True positives}}{\text{predicted positive}} = \frac{\text{True positive}}{\text{True pos + False pos}} = \frac{15}{20} = 0.75$

Recall → check how much model predicting correctly

$\frac{\text{True positives}}{\text{actual positive}} = \frac{\text{True positive}}{\text{True pos + False neg}} = \frac{15}{25} = 0.6$

Trading off precision and recall

(selecting threshold)

Logistic regression → $0 \leq f_{\vec{w},b}(\vec{x}) \leq 1$ (values of threshold)

Predict 1 ; If $f_{\vec{w},b}(\vec{x}) \geq$ 0.5 0.7 0.3

Predict 0 ; If $f_{\vec{w},b}(\vec{x}) <$ 0.5 0.7 0.3

more generally predict 1 if: $f_{\vec{w},b}(\vec{x}) \geq$ threshold

F1 score → compare precision/recall

(Harmonic mean)

$F_1 \text{ score} = \frac{1}{\frac{1}{2}\left(\frac{1}{P} + \frac{1}{R}\right)}$ (Precision or Recall)

	Precision	Recall	Average (not recommend)	F_1 score
Algorithm 1	0.5	0.4	0.45	0.444 (bad)
Algorithm 2	0.7	0.1	0.8	0.175
Algorithm 3	0.02	1.0	0.501	0.0392

(always be lower value than the higher value)