Hands-On: Deep Learning

Siegfried Kaidisch

Universal approximation theorem

- "… feedforward networks with non-polynomial activation functions are dense in the space of continuous functions between two Euclidean spaces, with respect to the compact convergence topology."
  [https://en.wikipedia.org/wiki/Universal_approximation_theorem]

Universal approximation theorem

- "… feedforward networks with non-polynomial activation functions are dense in the space of continuous functions between two Euclidean spaces, with respect to the compact convergence topology."
[https://en.wikipedia.org/wiki/Universal_approximation_theorem]

- In practice: **When you have a set of data and some quantity can in principle be deduced from that data, a feedforward ANN (artificial neural network) can learn to do so.**

Examples

- **Data → ANN → Quantity**

Examples

- **Data → ANN → Quantity**
- Watch history → ANN → Personal interests

Examples

- **Data → ANN → Quantity**
- Watch history → ANN → Personal interests
- Sensor readings from a factory machine → ANN → Risk of failure

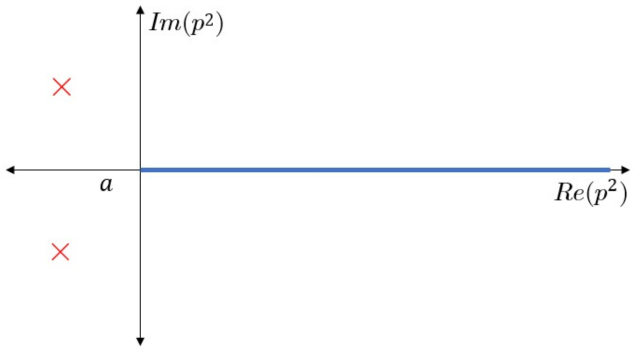
Examples

- **Data → ANN → Quantity**
- Watch history → ANN → Personal interests
- Sensor readings from a factory machine → ANN → Risk of failure
- Brain scan images → ANN → Presence of a tumor

Examples

- **Data → ANN → Quantity**
- Watch history → ANN → Personal interests
- Sensor readings from a factory machine → ANN → Risk of failure
- Brain scan images → ANN → Presence of a tumor
- Function values on R^+ → ANN → Complex poles

$$f(z) = \sum_{n=1}^{N_c} \left\{ \frac{c_n}{z - v_n} + \frac{c_n^*}{z - v_n^*} \right\}$$



ELSEVIER
Computer Physics Communications
Volume 295, February 2024, 108998

Computational Physics

Pole-fitting for complex functions: Enhancing standard techniques by artificial-neural-network classifiers and regressors ☆

Siegfried Kaidisch [a b], Thomas U. Hilger [c], Andreas Krassnigg [a b], Wolfgang Lucha [a]

Show more

Add to Mendeley Share Cite

https://doi.org/10.1016/j.cpc.2023.108998

Get rights and content

Bad descriptors

- **When there is no (or a very intangible) connection between data and desired quantity, the ANN <u>cannot</u> learn to derive the quantity from the data**

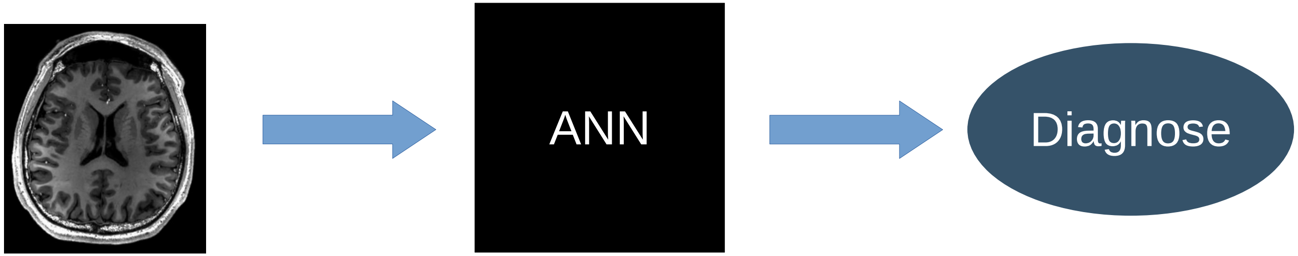
Bad descriptors

- **When there is no (or a very intangible) connection between data and desired quantity, the ANN <u>cannot</u> learn to derive the quantity from the data**
- Last week's lottery numbers → The next draw

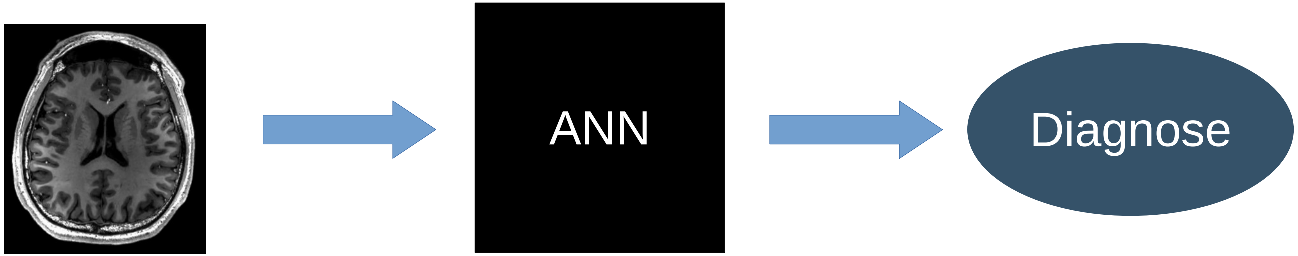
Bad descriptors

- **When there is no (or a very intangible) connection between data and desired quantity, the ANN <u>cannot</u> learn to derive the quantity from the data**
- Last week’s lottery numbers → The next draw
- Name of Titanic passenger → Survival probability
  - Better descriptor: Which deck was the passenger on?

(Feedforward) Artificial neural networks

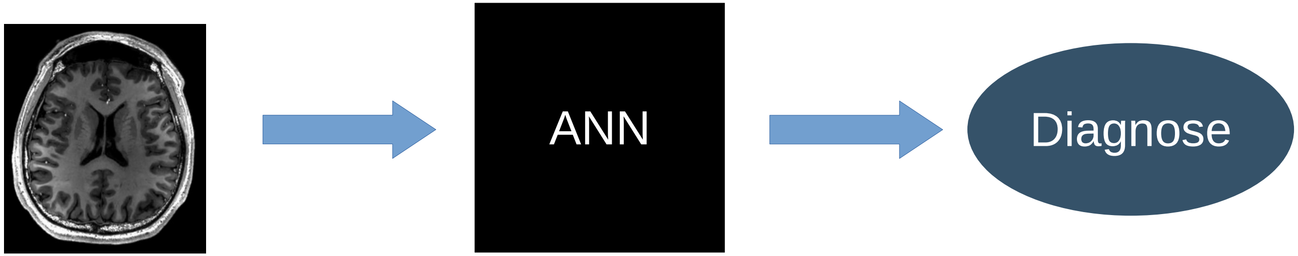


(Feedforward) Artificial neural networks



Q: **What is happening inside the ANN**, that derives the quantity from the data?

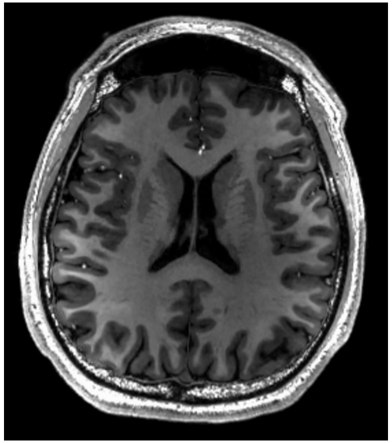
(Feedforward) Artificial neural networks



Q: **What is happening inside the ANN**, that derives the quantity from the data?

A: In short, **a set of matrices and vectors and a non-polynomial function** transform the input into the output.

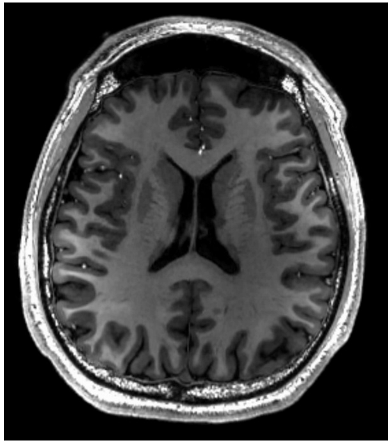
(Feedforward)
Artificial neural networks



[https://
upload.wikimedia.org/
wikipedia/commons/b/b2/
MRI_of_Human_Brain.jpg]

(Feedforward)
Artificial neural networks

[https://upload.wikimedia.org/wikipedia/commons/2/2f/Example_of_a_deep_neural_network.png]

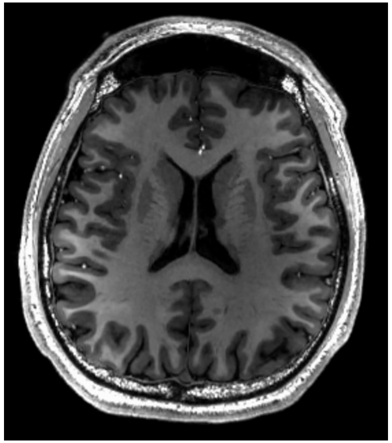


[https://upload.wikimedia.org/wikipedia/commons/b/b2/MRI_of_Human_Brain.jpg]

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_k \end{pmatrix}$$

(Feedforward)
Artificial neural networks

[https://upload.wikimedia.org/wikipedia/commons/2/2f/Example_of_a_deep_neural_network.png]

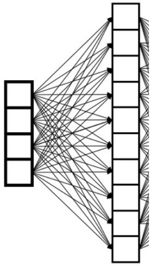


[https://upload.wikimedia.org/wikipedia/commons/b/b2/MRI_of_Human_Brain.jpg]

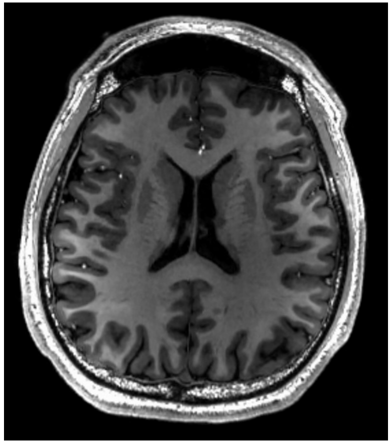
$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_k \end{pmatrix}$$

$$\overleftrightarrow{M}_1 \, \vec{x} + \vec{b}$$

(Feedforward)
Artificial neural networks



[https://upload.wikimedia.org/wikipedia/commons/2/2f/Example_of_a_deep_neural_network.png]

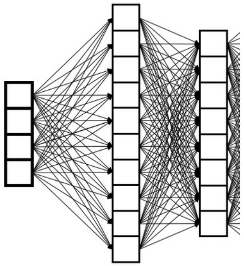


[https://upload.wikimedia.org/wikipedia/commons/b/b2/MRI_of_Human_Brain.jpg]

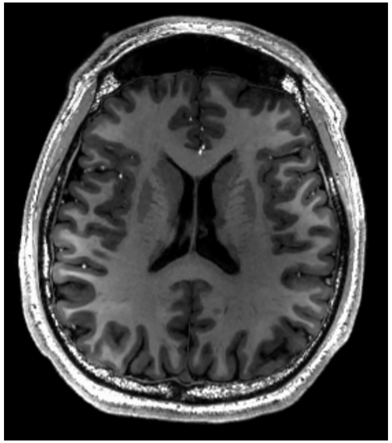
$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_k \end{pmatrix}$$

$$\vec{o}_1 = \varphi\left(\overset{\leftrightarrow}{M}_1 \vec{x} + \vec{b}_1\right)$$

(Feedforward)
Artificial neural networks



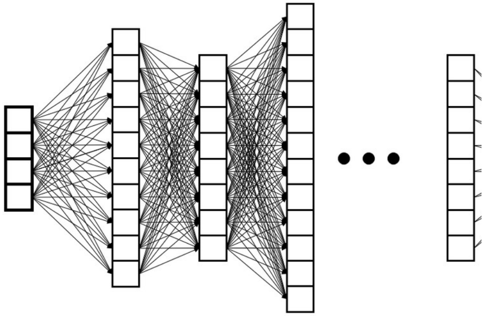
[https://upload.wikimedia.org/wikipedia/commons/2/2f/Example_of_a_deep_neural_network.png]



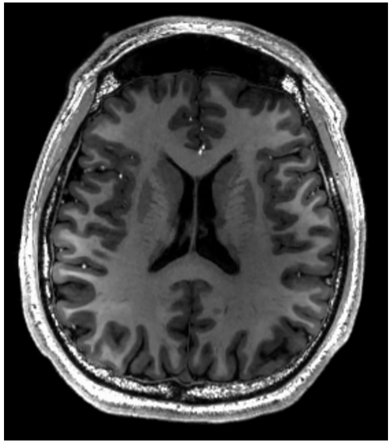
[https://upload.wikimedia.org/wikipedia/commons/b/b2/MRI_of_Human_Brain.jpg]

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_k \end{pmatrix} \Rightarrow \vec{o}_1 = \varphi\left(\overset{\leftrightarrow}{M}_1 \vec{x} + \vec{b}_1\right) \Rightarrow \vec{o}_2 = \varphi\left(\overset{\leftrightarrow}{M}_2 \vec{o}_1 + \vec{b}_2\right)$$

(Feedforward) Artificial neural networks



[https://upload.wikimedia.org/wikipedia/commons/2/2f/Example_of_a_deep_neural_network.png]

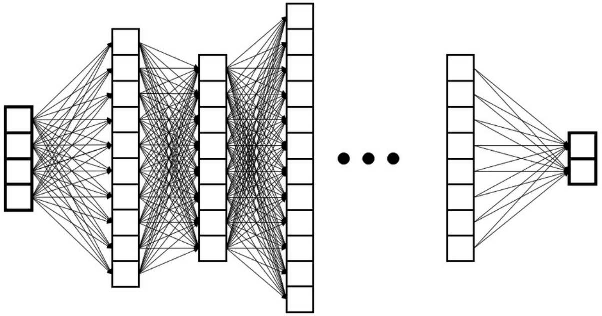


[https://upload.wikimedia.org/wikipedia/commons/b/b2/MRI_of_Human_Brain.jpg]

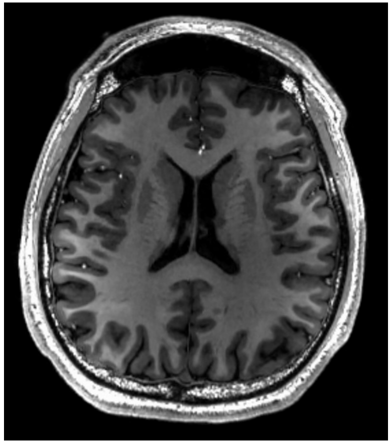
$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_k \end{pmatrix} \qquad \vec{o}_1 = \varphi\left(\overleftrightarrow{M}_1\,\vec{x} + \vec{b}_1\right) \qquad \vec{o}_2 = \varphi\left(\overleftrightarrow{M}_2\,\vec{o}_1 + \vec{b}_2\right)$$

...

(Feedforward)
Artificial neural networks



[https://upload.wikimedia.org/wikipedia/commons/2/2f/Example_of_a_deep_neural_network.png]



[https://upload.wikimedia.org/wikipedia/commons/b/b2/MRI_of_Human_Brain.jpg]

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_k \end{pmatrix} \Rightarrow \vec{o}_1 = \varphi\left(\overset{\leftrightarrow}{M}_1 \vec{x} + \vec{b}_1\right) \Rightarrow \vec{o}_2 = \varphi\left(\overset{\leftrightarrow}{M}_2 \vec{o}_1 + \vec{b}_2\right) \Rightarrow$$

$$\Rightarrow \ldots \Rightarrow \hat{\vec{y}} = \overset{\leftrightarrow}{M}_{\mathrm{out}} \vec{o}_{N_{\mathrm{HL}}} + \vec{b}_{\mathrm{out}}$$ = Diagnose

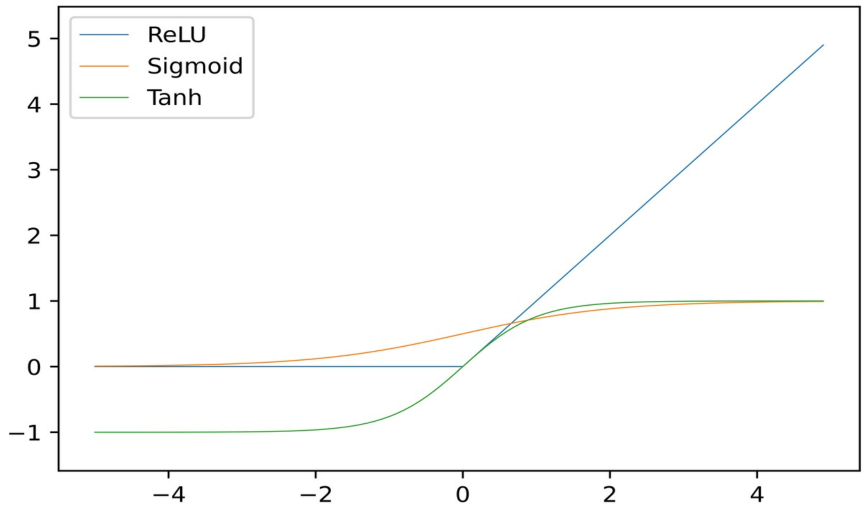
Activation functions

- Non-polynomial function φ

$$ReLU(x) = max\,(0, x)$$

$$Sigmoid(x) = \frac{1}{1 + e^{-x}}$$

$$Tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



Training ANNs

- In short: A set of matrices (weights) and vectors (biases) and a non-polynomial function

Training ANNs

- In short: A set of matrices (weights) and vectors (biases) and a non-polynomial function

- Change values in matrices and vectors → Change performance of the ANN

Training ANNs

- In short: A set of matrices (weights) and vectors (biases) and a non-polynomial function
- Change values in matrices and vectors → Change performance of the ANN
- Training ANN = Changing weights and biases such that performance increases

Loss function

- How can we tell, what the ANN’s performance is?

Loss function

- How can we tell, what the ANN’s performance is?
- We need data, where we know what the output should be!

Loss function

- How can we tell, what the ANN’s performance is?
- We need data, where we know what the output should be!
  - E.g. brainscan with a diagnosis created by a doctor

Loss function

- How can we tell, what the ANN’s performance is?
- We need data, where we know what the output should be!
  - E.g. brainscan with a diagnosis created by a doctor
- Feed the data (brainscan) to the ANN and compare its output/prediction to the correct result (diagnosis by doctor)

Loss function

- How can we tell, what the ANN’s performance is?
- We need data, where we know what the output should be!
  - E.g. brainscan with a diagnosis created by a doctor
- Feed the data (brainscan) to the ANN and compare its output/prediction to the correct result (diagnosis by doctor)
- Loss function: a function of the correct result and the prediction of the ANN, that is a measure of how good the prediction is:

$$loss(\hat{y_i}, y_i)$$

Loss function

- How can we tell, what the ANN's performance is?
- We need data, where we know what the output should be!
  - E.g. brainscan with a diagnosis created by a doctor
- Feed the data (brainscan) to the ANN and compare its output/prediction to the correct result (diagnosis by doctor)
- Loss function: a function of the correct result and the prediction of the ANN, that is a measure of how good the prediction is:

$$loss(\hat{y_i}, y_i)$$

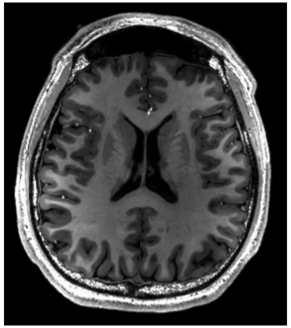
small $\Rightarrow$ prediction is good

Loss function – example

- Q: How effective will a certain treatment be for a patient?

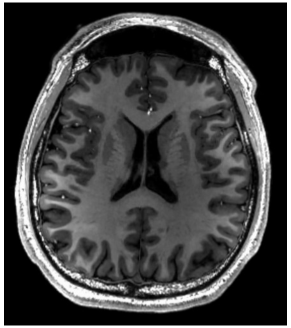
Loss function – example

- Q: How effective will a certain treatment be for a patient?
- Data: brainscan

$x_i =$ 

Loss function – example

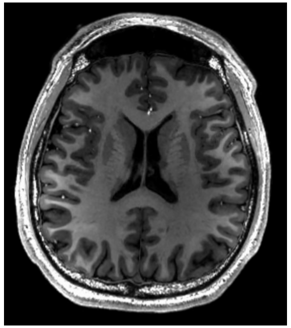
- Q: How effective will a certain treatment be for a patient?
- Data: brainscan
- Quantity to find: Effectiveness of treatment = Number between 0 and 100

$x_i =$ 

y_i = 80

Loss function – example

- Q: How effective will a certain treatment be for a patient?
- Data: brainscan
- Quantity to find: Effectiveness of treatment = Number between 0 and 100
- Loss function: Mean squared error: $loss(\hat{y}_i, y_i) = (y_i - \hat{y}_i)^2$

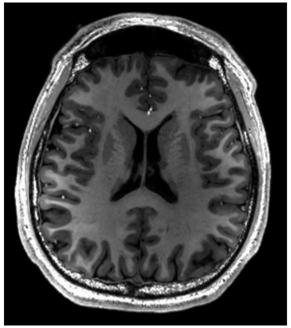
$x_i =$ 

y_i = 80

Loss function – example

- Q: How effective will a certain treatment be for a patient?
- Data: brainscan
- Quantity to find: Effectiveness of treatment = Number between 0 and 100
- Loss function: Mean squared error: $loss(\hat{y}_i, y_i) = (y_i - \hat{y}_i)^2$

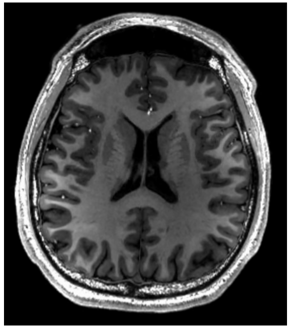
$\hat{y}_i$= 40 $\Rightarrow$ $loss(\hat{y}_i, y_i)$ = 1600

$x_i =$ 

y_i = 80

Loss function – example

- Q: How effective will a certain treatment be for a patient?
- Data: brainscan
- Quantity to find: Effectiveness of treatment = Number between 0 and 100
- Loss function: Mean squared error: $loss(\hat{y}_i, y_i) = (y_i - \hat{y}_i)^2$

$x_i =$ 

y_i = 80

$\hat{y}_i$ = 40 $\Rightarrow$ $loss(\hat{y}_i, y_i)$ = 1600

$\hat{y}_i$ = 75 $\Rightarrow$ $loss(\hat{y}_i, y_i)$ = 25

Cost function

- The loss function tells us how good a single prediction of the ANN is

Cost function

- The loss function tells us how good a single prediction of the ANN is

- Cost function: = Loss function averaged over a batch of data

$$J = \frac{1}{M} \sum_{i=1}^{M} loss(\hat{y}_i, y_i)$$

Cost function

- The loss function tells us how good a single prediction of the ANN is

- Cost function: = Loss function averaged over a batch of data
  - Tells us the average/expected performance of the ANN

$$J = \frac{1}{M} \sum_{i=1}^{M} loss(\hat{y}_i, y_i)$$

Cost function

- The loss function tells us how good a single prediction of the ANN is

- Cost function: = Loss function averaged over a batch of data
  - Tells us the average/expected performance of the ANN

$$J = \frac{1}{M}\sum_{i=1}^{M} loss(\hat{y}_i, y_i)$$

small $\Rightarrow$ ANN is good

Cost function

- The loss function tells us how good a single prediction of the ANN is

- Cost function: = Loss function averaged over a batch of data
  - Tells us the average/expected performance of the ANN

$$J = \frac{1}{M} \sum_{i=1}^{M} loss(\hat{y_i}, y_i)$$

small → ANN is good

→ Training ANN =

Cost function

- The loss function tells us how good a single prediction of the ANN is

- Cost function: = Loss function averaged over a batch of data
  - Tells us the average/expected performance of the ANN

$$J = \frac{1}{M} \sum_{i=1}^{M} loss(\hat{y_i}, y_i)$$

small $\Rightarrow$ ANN is good

$\Rightarrow$ Training ANN = Adapting weights and biases, such that cost function gets smaller

Supervised learning

- Training ANN = Adapting weights and biases, such that cost function gets smaller

Supervised learning

- Training ANN = Adapting weights and biases, such that cost function gets smaller

- Q: How do we actually do this?

Supervised learning

- Training ANN = Adapting weights and biases, such that cost function gets smaller
- Q: How do we actually do this?
- Supervised learning: Use manually created data

Supervised learning

- Training ANN = Adapting weights and biases, such that cost function gets smaller

- Q: How do we actually do this?

- Supervised learning: Use manually created data
  - E.g.: 10.000(=N) brainscans(=x_i) and the corresponding correct diagnosis(=y_i)

Supervised learning

- Training ANN = Adapting weights and biases, such that cost function gets smaller

- Q: How do we actually do this?

- Supervised learning: Use manually created data
  - E.g.: 10.000(=N) brainscans(=x_i) and the corresponding correct diagnosis(=y_i)

$$\{(x_i, y_i), i = 1, ...N\}$$

Supervised learning

- Data: $\{(x_i, y_i), i = 1, ...N\}$

Supervised learning

- Data: $\{(x_i, y_i), i = 1, ...N\}$
- Goal: minimize cost function

$$J = \frac{1}{M} \sum_{i=1}^{M} loss(\hat{y}_i, y_i)$$

Supervised learning

- Data: $\{(x_i, y_i), i = 1, ...N\}$
- Goal: minimize cost function

$$J = \frac{1}{M}\sum_{i=1}^{M} loss(\hat{y}_i, y_i)$$

- Procedure (backpropagation and stochastic gradient descent):

Supervised learning

- Data: $\{(x_i, y_i), i = 1, ...N\}$
- Goal: minimize cost function

$$J = \frac{1}{M} \sum_{i=1}^{M} loss(\hat{y}_i, y_i)$$

- Procedure (backpropagation and stochastic gradient descent):
  - Calculate ANN prediction for a batch of data (e.g. 100 brainscans)

Supervised learning

- Data: $\{(x_i, y_i), i = 1, ...N\}$
- Goal: minimize cost function

$$J = \frac{1}{M}\sum_{i=1}^{M} loss(\hat{y}_i, y_i)$$

- Procedure (backpropagation and stochastic gradient descent):
  - Calculate ANN prediction for a batch of data (e.g. 100 brainscans)
  - Calculate cost function

Supervised learning

- Data: $\{(x_i, y_i), i = 1, ...N\}$

$$J = \frac{1}{M} \sum_{i=1}^{M} loss(\hat{y}_i, y_i)$$

- Goal: minimize cost function
- Procedure (backpropagation and stochastic gradient descent):
  - Calculate ANN prediction for a batch of data (e.g. 100 brainscans)
  - Calculate cost function
  - Calculate derivative of the cost function w.r.t. ANN parameters (weights and biases)

Supervised learning

- Data: $\{(x_i, y_i), i = 1, ...N\}$
- Goal: minimize cost function

$$J = \frac{1}{M} \sum_{i=1}^{M} loss(\hat{y}_i, y_i)$$

- Procedure (backpropagation and stochastic gradient descent):
  - Calculate ANN prediction for a batch of data (e.g. 100 brainscans)
  - Calculate cost function
  - Calculate derivative of the cost function w.r.t. ANN parameters (weights and biases)
  - Update ANN parameters

$$p \rightarrow p - \alpha \frac{\partial J}{\partial p}$$

Supervised learning

- Data: $\{(x_i, y_i), i = 1, ...N\}$
- Goal: minimize cost function

$$J = \frac{1}{M} \sum_{i=1}^{M} loss(\hat{y}_i, y_i)$$

- Procedure (backpropagation and stochastic gradient descent):
  - Calculate ANN prediction for a batch of data (e.g. 100 brainscans)
  - Calculate cost function
  - Calculate derivative of the cost function w.r.t. ANN parameters (weights and biases)
  - Update ANN parameters

$$p \to p - \alpha \frac{\partial J}{\partial p}$$

  - Repeat with next batch ...

What have we achieved? - Applying the ANN

- Supervised Learning: use manually created data to train ANN

$$\{(x_i, y_i), i = 1, ...N\}$$

What have we achieved? - Applying the ANN

- Supervised Learning: use manually created data to train ANN

$$\{(x_i, y_i), i = 1, ...N\}$$

- → ANN learns to derive quantity from data (ANN: $x_i \rightarrow y_i$)

What have we achieved? - Applying the ANN

- Supervised Learning: use manually created data to train ANN
$$\{(x_i, y_i), i = 1, ...N\}$$
- → ANN learns to derive quantity from data (ANN: $x_i \rightarrow y_i$)
- → Can apply ANN to new data x_i, where quantity, y_i, is unknown (undiagnosed brain scans)

What have we achieved? - Applying the ANN

- Supervised Learning: use manually created data to train ANN
$$\{(x_i, y_i), i = 1, \ldots N\}$$
- → ANN learns to derive quantity from data (ANN: $x_i \rightarrow y_i$)
- → Can apply ANN to new data x_i, where quantity, y_i, is unknown (undiagnosed brain scans)
- Cheap, fast and accurate

Training data vs. unseen data

- Supervised Learning: Use manually created data to train ANN

Training data vs. unseen data

- Supervised Learning: Use manually created data to train ANN
- **Important: ANN will usually perform better on the data that it was trained on, than on data, that it has never seen!**

Training data vs. unseen data

- Supervised Learning: Use manually created data to train ANN
- **Important: ANN will usually perform better on the data that it was trained on, than on data, that it has never seen!**

  → Performance may seem better than it actually is!

Training data vs. unseen data

- Supervised Learning: Use manually created data to train ANN
- **Important: ANN will usually perform better on the data that it was trained on, than on data, that it has never seen!**

  → Performance may seem better than it actually is!
- Reason: Overfitting

Training data vs. unseen data

- Supervised Learning: Use manually created data to train ANN
- **Important: ANN will usually perform better on the data that it was trained on, than on data, that it has never seen!**

  → Performance may seem better than it actually is!
- Reason: Overfitting
  - E.g., may have 10.000 brainscans but ANN has 100.000 parameters

Training data vs. unseen data

- Supervised Learning: Use manually created data to train ANN
- **Important: ANN will usually perform better on the data that it was trained on, than on data, that it has never seen!**

  → Performance may seem better than it actually is!
- Reason: Overfitting
  - E.g., may have 10.000 brainscans but ANN has 100.000 parameters
  - ANN may not actually learn to detect patterns, but just memorizes the training data

Training data vs. unseen data

- Supervised Learning: Use manually created data to train ANN
- **Important: ANN will usually perform better on the data that it was trained on, than on data, that it has never seen!**

  → Performance may seem better than it actually is!
- Reason: Overfitting
  - E.g., may have 10.000 brainscans but ANN has 100.000 parameters
  - ANN may not actually learn to detect patterns, but just memorizes the training data
- Solution: Split manually prepared data up

Splitting up data: training and validation

- Split data into three separate sets:

Splitting up data: training and validation

- Split data into three separate sets:
  - Training set: - Update parameters to enhance performance

Splitting up data: training and validation

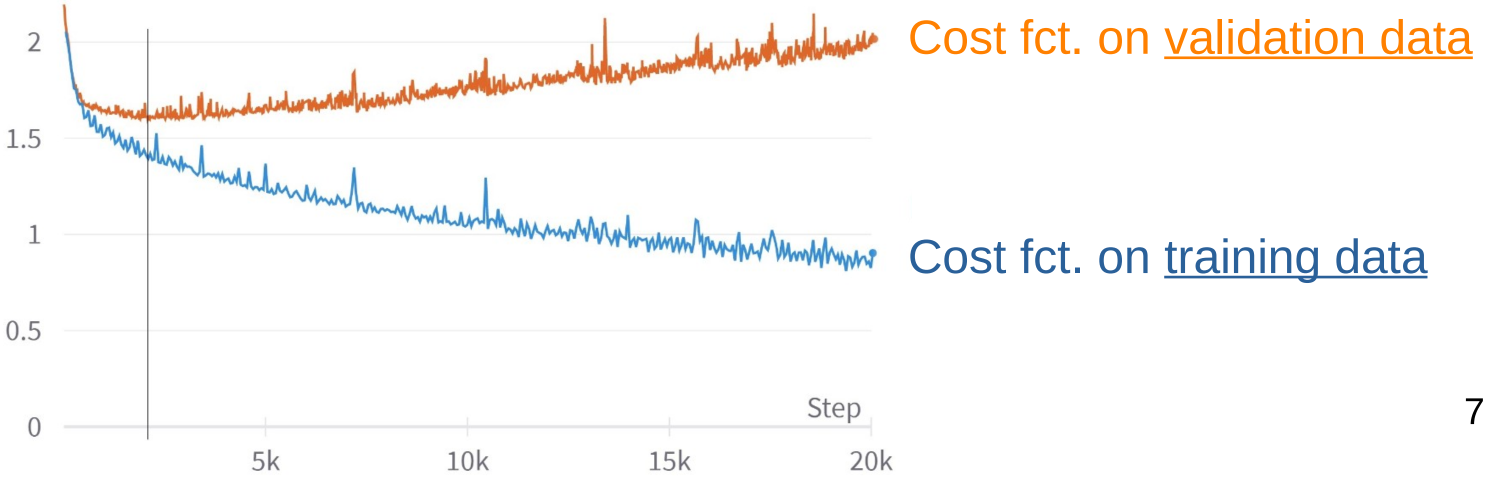
- Split data into three separate sets:
  - Training set: - Update parameters to enhance performance
  - Validation set: - Check performance during training → prevent overfitting

    - Never used to update ANN parameters!

Splitting up data: training and validation

- Split data into three separate sets:
  - Training set: - Update parameters to enhance performance
  - Validation set: - Check performance during training → prevent overfitting

    - Never used to update ANN parameters!



Splitting up data: testing

- Split data into three separate sets:
  - Training set: - Update parameters to enhance performance
  - Validation set: - Check performance during training, prevent overfitting
  - **Test set**: - Unbiased performance evaluation, when hyperparameters have been chosen and training is finished

Splitting up data: testing

- Split data into three separate sets:
  - Training set: - Update parameters to enhance performance
  - Validation set: - Check performance during training, prevent overfitting
  - **Test set**: - Unbiased performance evaluation, when hyperparameters have been chosen and training is finished

Hyperparameters:
- Number and size of hidden layers
- Batch size
- Choice of loss function
- Optimization algorithm and learning rate
- ...

Regression

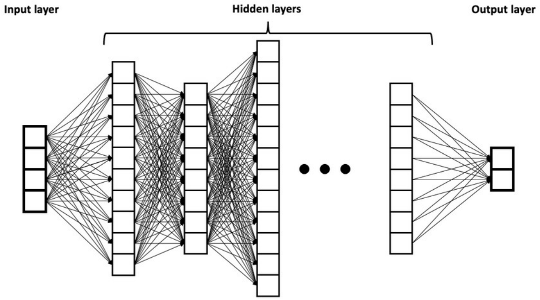
Regression

- Goal: Predict a vector of real numbers

Regression

- Goal: Predict a vector of real numbers

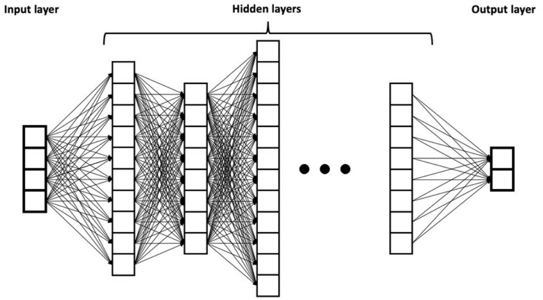
$$\{(x_i, y_i), i = 1, ...N\}$$



Regression

- Goal: Predict a vector of real numbers

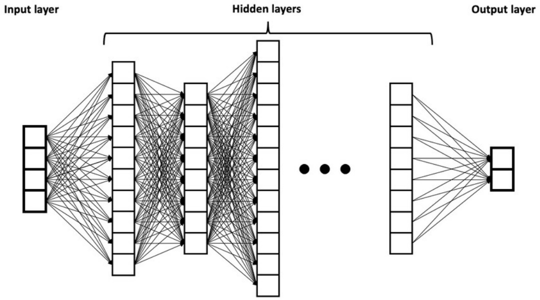
$$\{(x_i, y_i), i = 1, ...N\}$$



Regression

- Goal: Predict a vector of real numbers

$$\{(x_i, y_i), i = 1, ...N\}$$



House i: x_i = (200, 10, 1, ...)

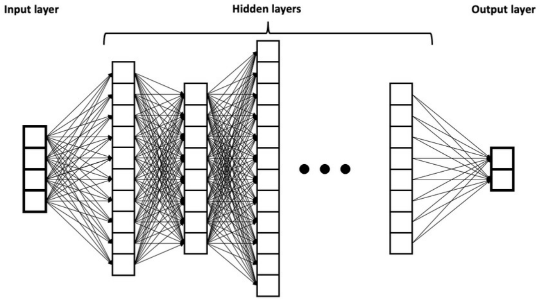
$$y_i = \begin{pmatrix} 1,000,000 \\ 5,000 \end{pmatrix}$$

Regression

- Goal: Predict a vector of real numbers
- Loss function: Mean-Squared Error (MSE)

$$loss\left(\hat{y}_i, y_i\right) = \frac{1}{k}\sum_{j=1}^{k}\left(y_{i,j} - \hat{y}_{i,j}\right)^2$$

$$\{(x_i, y_i), i = 1, ...N\}$$



House i: x_i = (200, 10, 1, ...)

$$y_i = \begin{pmatrix} 1,000,000 \\ 5,000 \end{pmatrix}$$

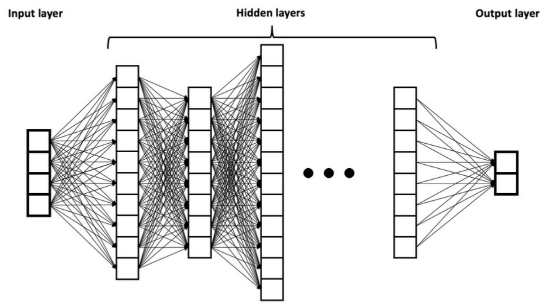
Regression

- Goal: Predict a vector of real numbers
- Loss function: Mean-Squared Error (MSE)

$$loss\left(\hat{y}_i, y_i\right) = \frac{1}{k}\sum_{j=1}^{k}\left(y_{i,j} - \hat{y}_{i,j}\right)^2$$

- ANN output = ANN prediction
- Example: Find house price and rental income from size, location score, ag

$$\{(x_i, y_i), i = 1, \ldots N\}$$



House i: x_i = (200, 10, 1, ...)

$$y_i = \begin{pmatrix} 1,000,000 \\ 5,000 \end{pmatrix}, \quad \hat{y}_i = \begin{pmatrix} 800,000 \\ 4,500 \end{pmatrix}$$

Classification

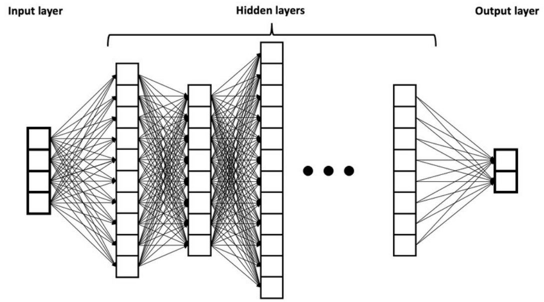
Classification

- Goal: Assign input to class

Classification

- Goal: Assign input to class

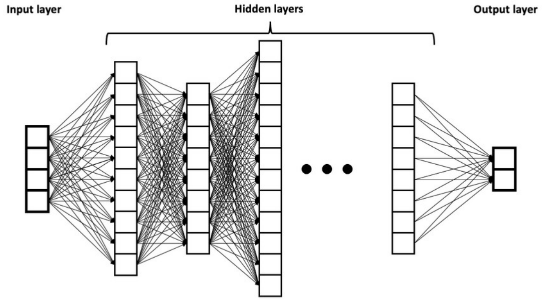
$$\{(x_i, y_i), i = 1, \ldots N\}$$



Classification

- Goal: Assign input to class

$$\{(x_i, y_i), i = 1, ...N\}$$

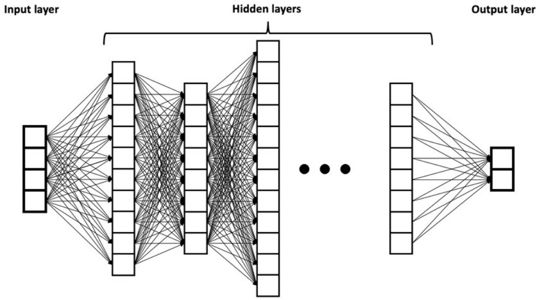


[Karen Zack (@TeenyBiscuit)]

Classification

- Goal: Assign input to class

$$\{(x_i, y_i), i = 1, ...N\}$$

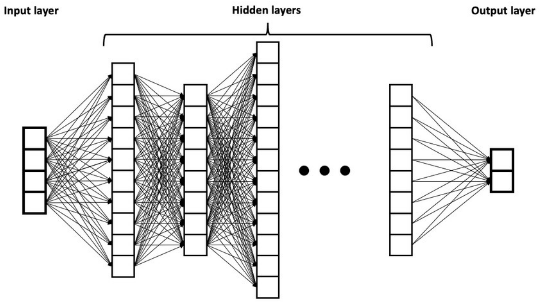


[Karen Zack (@TeenyBiscuit)]

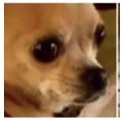
Classification

- Goal: Assign input to class

$$\{(x_i, y_i), i = 1, \dots N\}$$



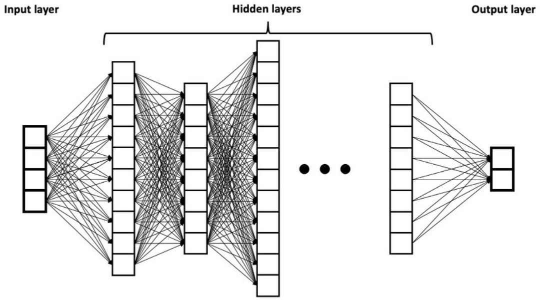
[Karen Zack (@TeenyBiscuit)]

$x_i =$  $\qquad y_i = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

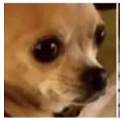
Classification

- Goal: Assign input to class
- Loss function: Cross-Entropy Loss

$$\{(x_i, y_i), i = 1, \ldots N\}$$



[Karen Zack (@TeenyBiscuit)]

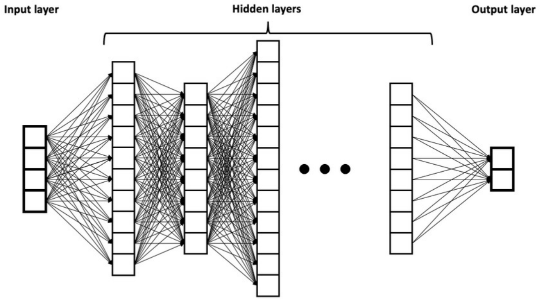
$x_i =$  $\qquad y_i = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

Classification

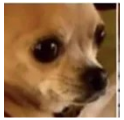
- Goal: Assign input to class
- Loss function: Cross-Entropy Loss

$$p_{i,c} = \frac{exp(\hat{y}_{i,c})}{\sum_{c'} exp(\hat{y}_{i,c'})}$$

$$\{(x_i, y_i), i = 1, ...N\}$$



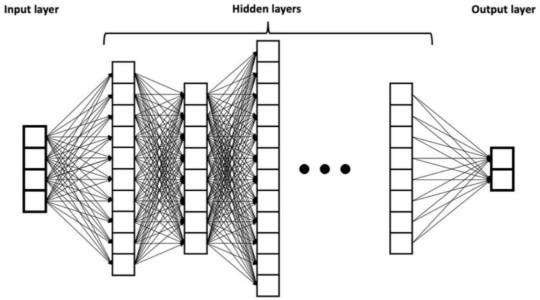
[Karen Zack (@TeenyBiscuit)]

$x_i =$  $\quad y_i = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad p_i = \begin{pmatrix} 0.8 \\ 0.2 \end{pmatrix}$

Classification

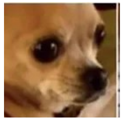
$$\{(x_i, y_i), i = 1, ...N\}$$

- Goal: Assign input to class
- Loss function: Cross-Entropy Loss



$$p_{i,c} = \frac{exp(\hat{y}_{i,c})}{\sum_{c'} exp(\hat{y}_{i,c'})} \qquad loss(\hat{y}_i, y_i) = -\sum_c y_{i,c} * log(p_{i,c})$$

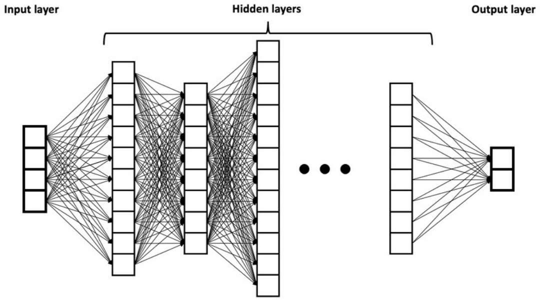
[Karen Zack (@TeenyBiscuit)]

$x_i =$ 

$$y_i = \begin{pmatrix}1\\0\end{pmatrix}, \quad p_i = \begin{pmatrix}0.8\\0.2\end{pmatrix}$$

Classification

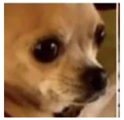
$$\{(x_i, y_i), i = 1, ...N\}$$



- Goal: Assign input to class
- Loss function: Cross-Entropy Loss

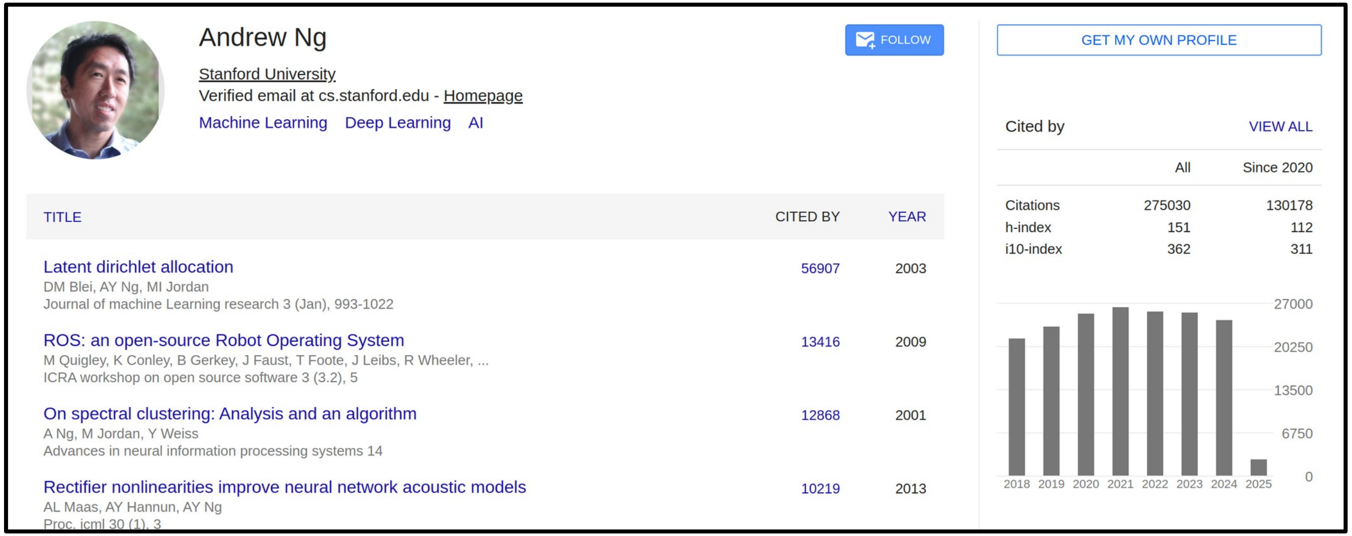
$$p_{i,c} = \frac{exp(\hat{y}_{i,c})}{\sum_{c'} exp(\hat{y}_{i,c'})} \qquad loss(\hat{y}_i, y_i) = -\sum_c y_{i,c} * log(p_{i,c})$$

- ANN prediction: $argmax_c(p_{i,c})$
- Example: Chihuahua or muffin?
  - 2 classes

$x_i =$  $\quad y_i = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad p_i = \begin{pmatrix} 0.8 \\ 0.2 \end{pmatrix}$ → Chihuahua

[Karen Zack (@TeenyBiscuit)]

“DeepLearningAI” YouTube channel

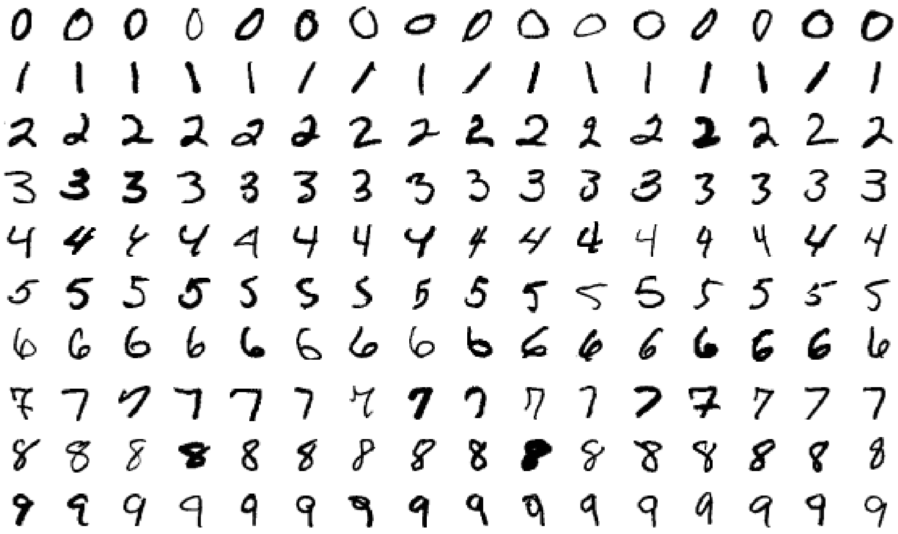


- Start with “Course 1 of the Deep Learning Specialization”

Hands-on part

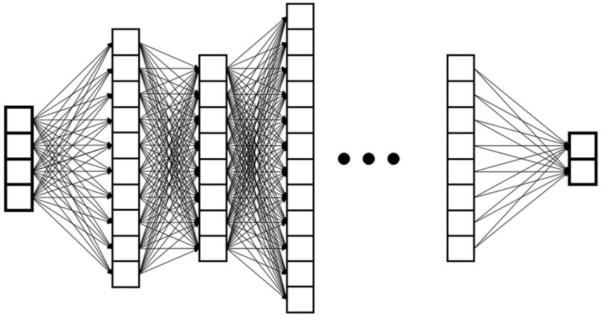
Hands-on example

- MNIST
  - https://www.kaggle.com/datasets/scolianni/mnistasjpg

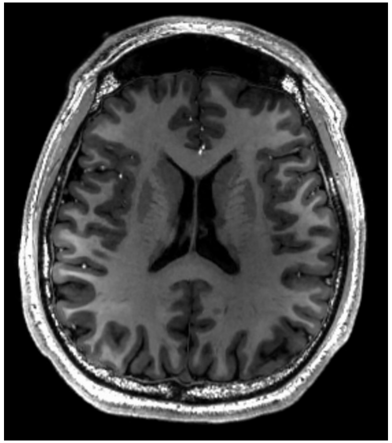


https://upload.wikimedia.org/wikipedia/commons/2/27/MnistExamples.png

(Feedforward) Artificial neural networks



[https://upload.wikimedia.org/wikipedia/commons/2/2f/Example_of_a_deep_neural_network.png]



[https://upload.wikimedia.org/wikipedia/commons/b/b2/MRI_of_Human_Brain.jpg]

$$\vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_k \end{pmatrix} \quad \vec{o}_1 = \varphi\left(\overleftrightarrow{M}_1\,\vec{x} + \vec{b}_1\right) \quad \vec{o}_2 = \varphi\left(\overleftrightarrow{M}_2\,\vec{o}_1 + \vec{b}_2\right)$$

...

$$\hat{\vec{y}} = \overleftrightarrow{M}_{\text{out}}\,\vec{o}_{N_{\text{HL}}} + \vec{b}_{\text{out}}$$ = Diagnose