Neural networks

Regression, logistic regression, neural networks

(Attila Bagoly)

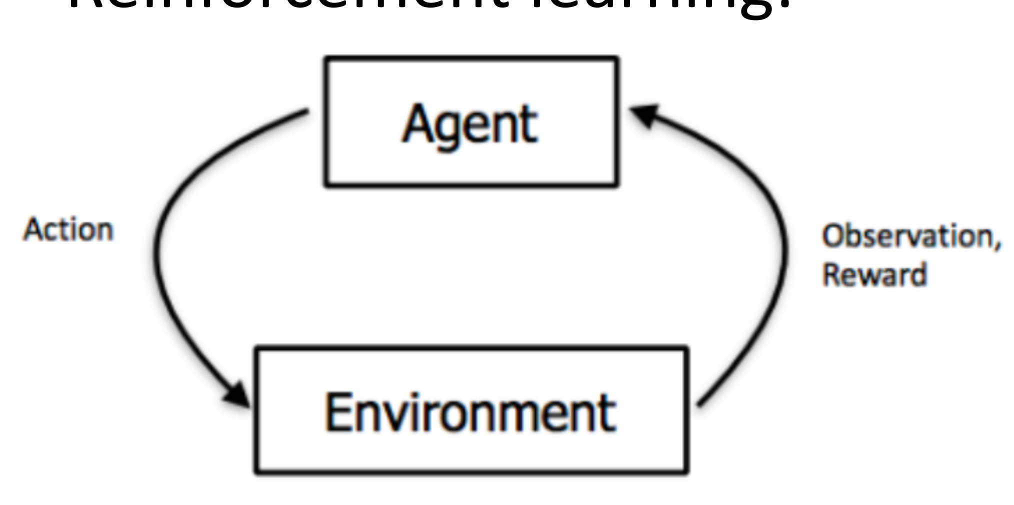
- Google Form:
  - Accidentally numeric filter on Kaggle username
  - Who entered profile ID, please go back, and change it to your username
  - Also GitHub username field: just the username
- Homework:
  - Rename the file: hw01_numpy.ipynb => hw01_numpy_solved.ipynb
  - Don’t write your code on other files! All of your solutions should be in the notebook!
- Course mail: deeplearninginsciences@gmail.com
- Discussion: GitHub issues, Kaggle forums
- Homework points: soon

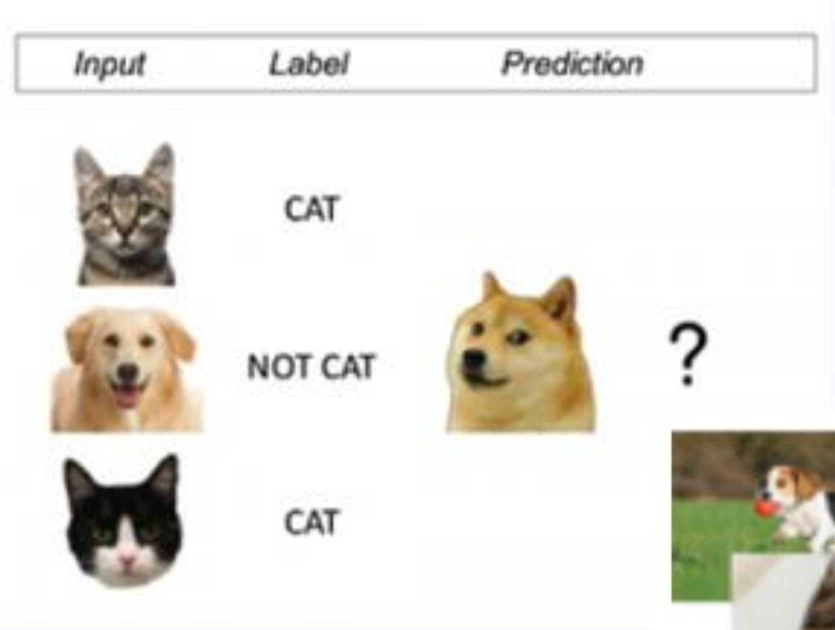
Technical info

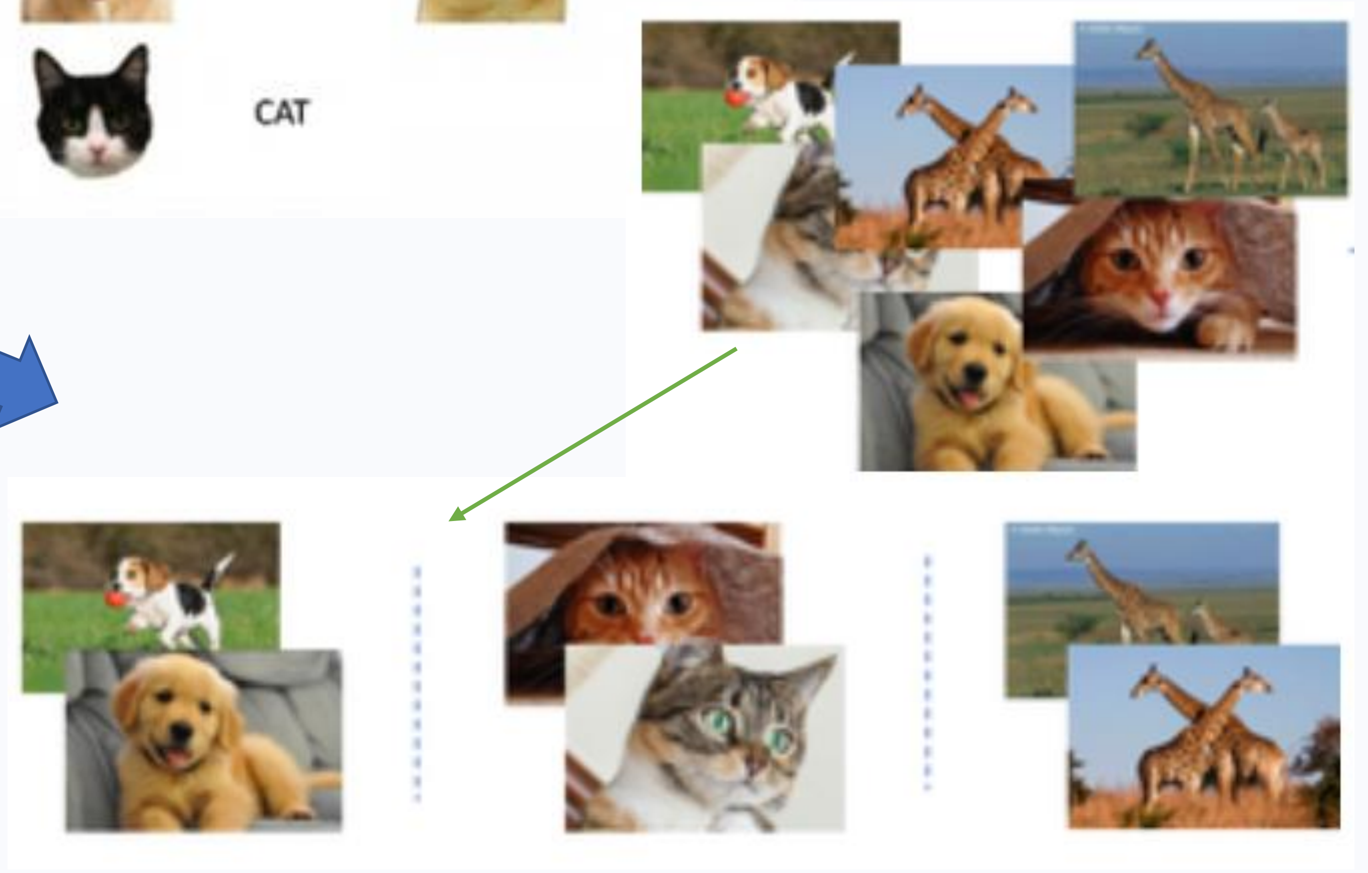
- GitHub forks: 39/43 (problems?)
- Filled form: 38/43 (problems?)
- Homework: questions?
- Nvidia GPU access: 14/38
- Who doesn’t have GPU, will receive Google Cloud access
- If you did not requested GCP in the form, but you need send an email to Attila
- Credits are limited: 50$/student
- Costs:
  - 4 core, 15GB RAM: 0.43$/10 hour
  - 4 core, 15GB RAM, K80 GPU: 0.262$/hour
- Google Cloud tutorial: later
- Who doesn’t know what is conditional probability?

Problem formulation

- Supervised learning
  - Given: (x, y) datapoints
  - We want: $x \to y$ mapping
- Unsupervised learning:
  - Given: x data points
  - No labels!
  - We want: hidden structure
- Reinforcement learning:







Traditional modelling and machine learning

Traditional modelling

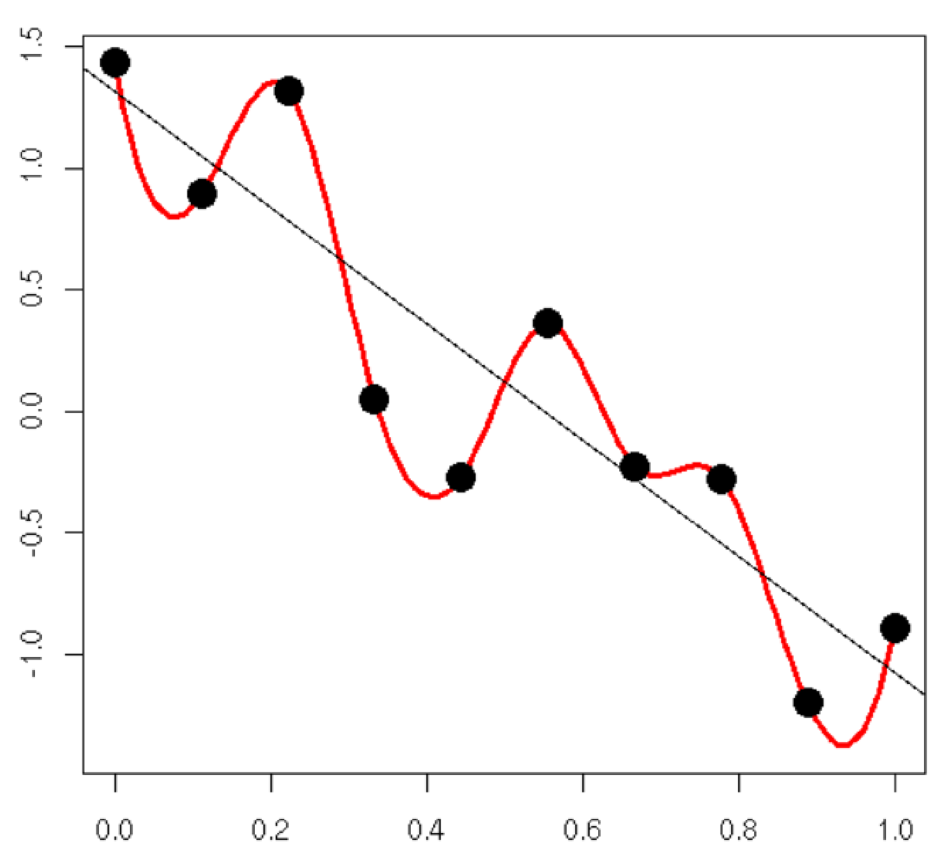
- We have data: $(x^{(i)}, y^{(i)})$
- We want to model the data: $x \to y$
- Goal: understanding, transfer model to new phenomenon, accurately predict new unmeasured data
- Often doesn't work for complex data
- Model:
  - From theoretical considerations
  - Model value: subjective ("understanding" and simplicity matters: is it nice?)
  - Simple (we understand it) with a few parameters (1-20)

Machine learning

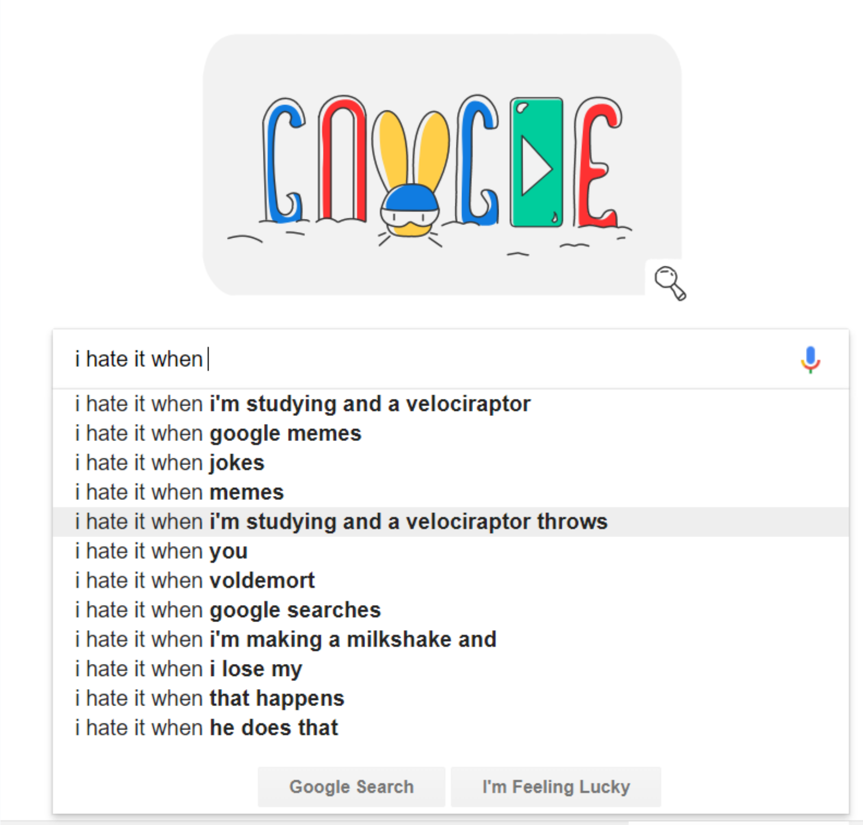
- We have data: $(x^{(i)}, y^{(i)})$
- We want to model the data: $x \to y$
- Goal: accurately predict new (unseen) experiments/data
- Works for complex data
- Model:
  - Universal function approximators
  - Model value: objective (only performance matters)
  - Complex (we do not understand it) with a lot of parameters (100k-150M)
  - Lot of parameters not a problem

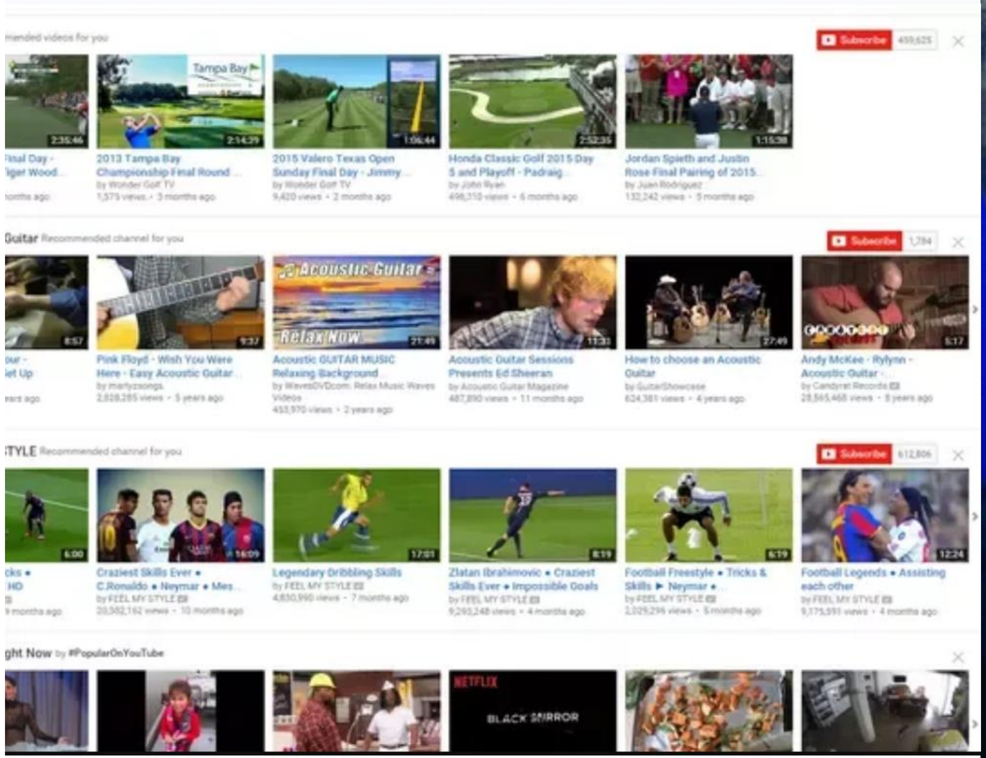
Performance

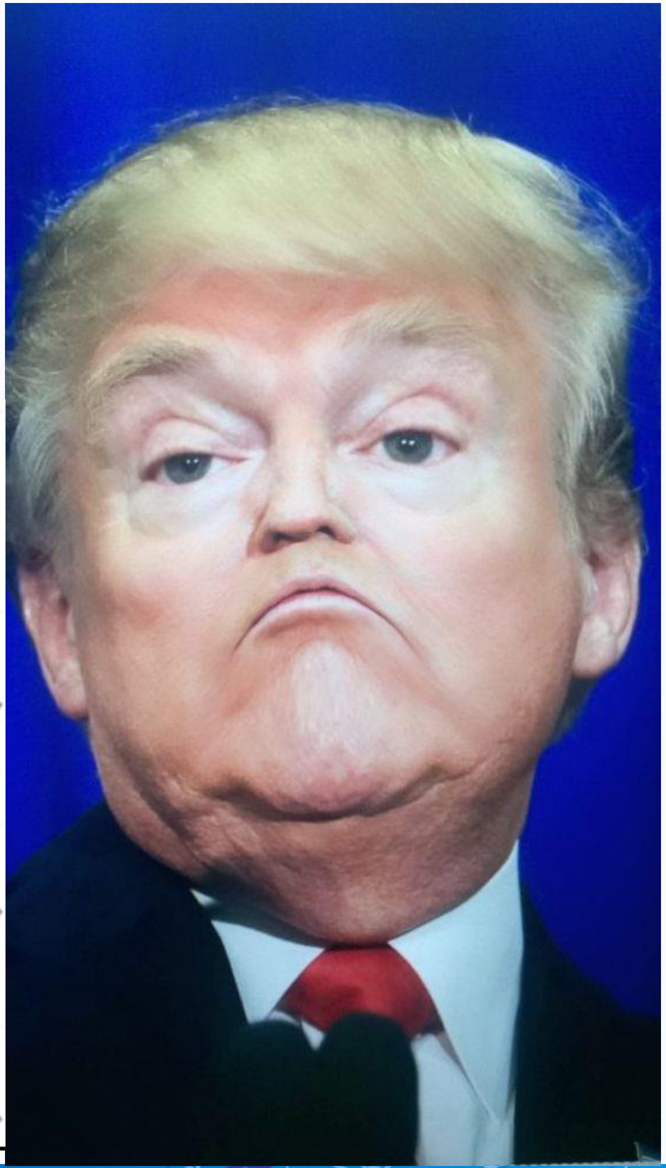
- Machine learning models, especially deep neural nets: lot of parameters
- Easy to “memorize” the data: train set accuracy can be high easily
- But what we really want: how well it work’s for unseen data
- Splitting the data: train/test sets
- Accuracy: on test set
- Train set accuracy doesn’t matter much



- Search (e.g. Google)
- Recommender systems (YouTube, Facebook, etc.)
- Picture labeling (Facebook, etc.), filters (e.g. Snapchat)
- Advertising (Google, Facebook, etc.)





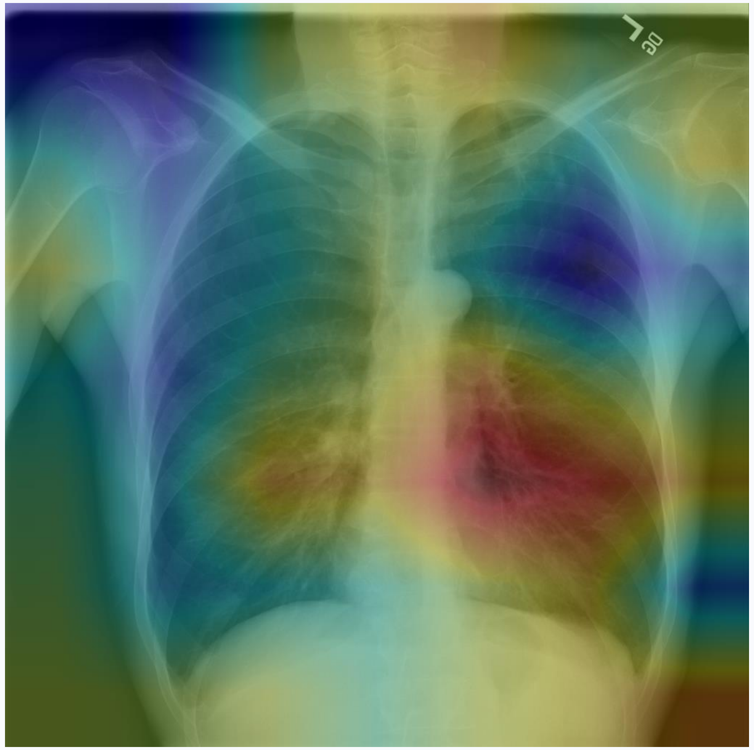


- Earthquake detection, localization
- Surveillance
- Self-driving cars

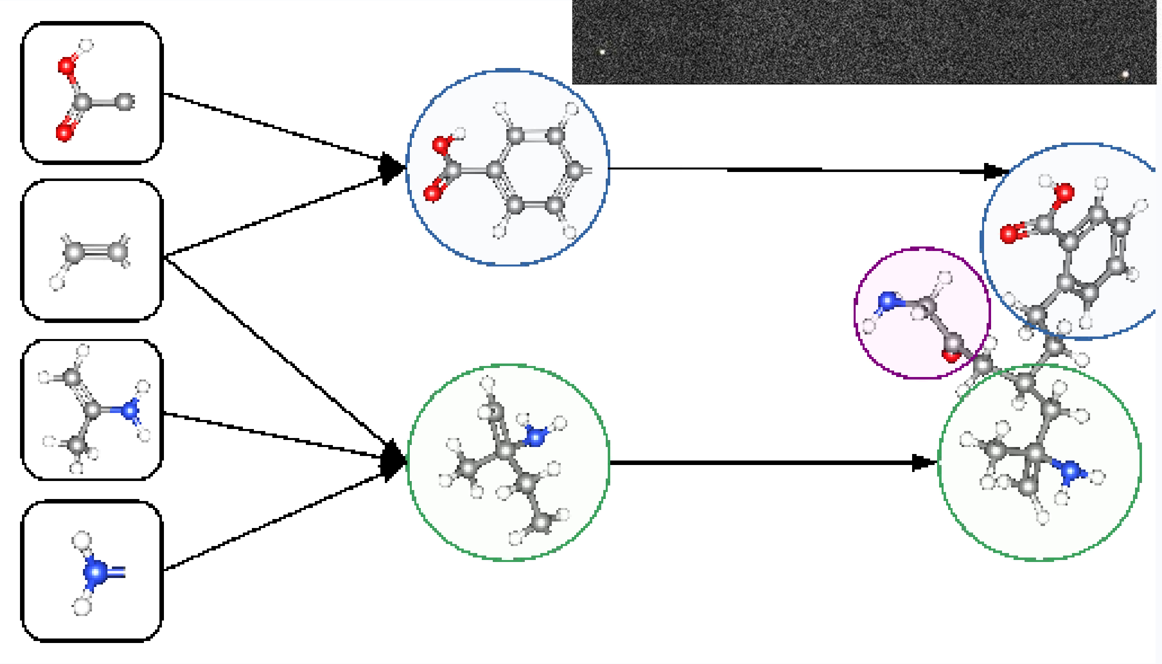
Motivations

- Medical: x-rays, MRI (tumor detection, etc.)
- Drug discovery
- Astronomy: image reconstruction, galaxy evolution, grav. wave detection etc.
- Particle physics: e.g. search for new physics

KIYOSHI TAKAHASE SEGUNDO/ALAMY STOCK PHOTO



CheXNet



DeepTox

Motivations

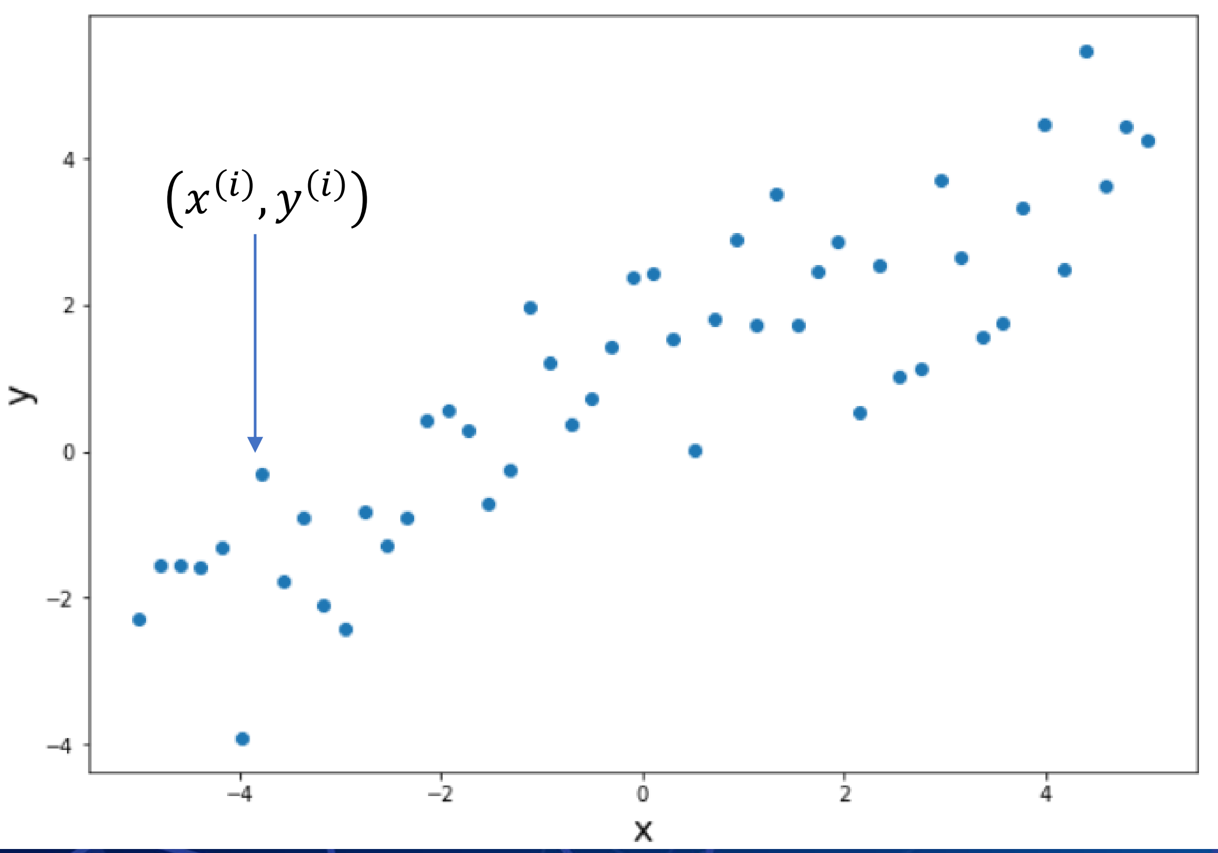
- Machine translation
- Playing games (eg. Go)
- Style transfer
- Expression transfer

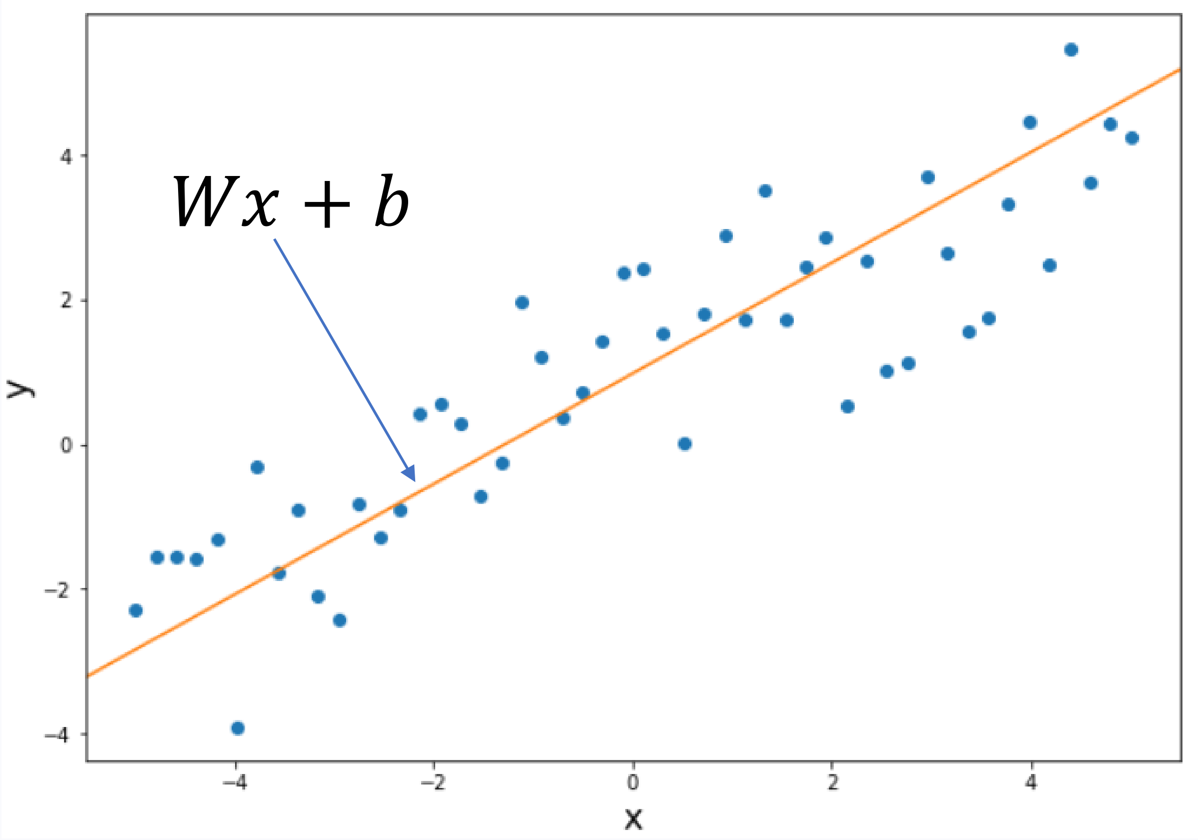
AlphaGo

Face2Face

credit to @DmitryUlyanovML

- Problem:
  - Given: $\left\{\left(x^{(1)}, y^{(1)}\right), \left(x^{(2)}, y^{(2)}\right), \ldots, \left(x^{(m)}, y^{(m)}\right)\right\}, x \in \mathbb{R}^N, y \in \mathbb{R}^K$
  - We want a model: $f: \mathbb{R}^N \to \mathbb{R}^K, f\left(x^{(i)}\right)$ *is close to* $y^{(i)}, \forall i$
- Linear regression: f is linear: $f(x) = Wx + b, where\ W \in \mathbb{R}^{K \times N}, b \in \mathbb{R}^K$





- How can we determine W, b?
- Notation: $X^{(i)} = (x^{(i)}, y^{(i)})$
- Given W, b the likelihood of $f_{W,b}\left(x^{(1)}\right)\, is\ y^{(1)},\ f_{W,b}\left(x^{(2)}\right)\, is\ y^{(2)}, \ldots$

$$P_f\left(X^{(1)}, X^{(2)}, \ldots, X^{(m)} \middle| W, b\right) = l(W, b)$$

- One approach: find W, b which maximizes the likelihood
- This is the maximum likelihood method
- Finding W, b: $\underset{W,b}{\operatorname{argmax}}\, l(W, b)$

- We assume, X random variables are IID (independent and identically distributed):
  $$l(W,b) = P_f\left(X^{(1)}, X^{(2)}, \ldots, X^{(m)} \middle| W,b\right) = \prod_{i=1}^{m} P_f(\, X^{(i)} |\, W, b\,)$$
- We know: $\underset{W,b}{\operatorname{argmax}}\, l(W,b) = \underset{W,b}{\operatorname{argmax}} \log l(W,b)$ (log is monotonic)
  $$\log l(W,b) = \sum_{i=1}^{m} \log P_f(X^{(i)} |\, W, b\,)$$
- We assume, normal distribution for P_f:
  - Lot of random errors contribute to the distribution
  - Without systematical errors, we can assume the errors are IID
  - The summed errors distribution approaches normal distribution (Central limit theorem)
- For simplicity in 1D:

$$P_f(X^{(i)} |\, W, b\,) = P_f((x^{(i)}, y^{(i)}) |\, W, b\,) \sim e^{-\frac{\left(f\left(x^{(i)}\right) - y^{(i)}\right)^2}{2\sigma_i^2}}$$

- With normal distribution (for simplicity 1D):

$$\log P_f(X^{(i)} | W, b) \sim -0.5\left(f\left(x^{(i)}\right) - y^{(i)}\right)^2$$

- The log-likelihood function (for simplicity 1D):

$$\log l(W, b) = C_1 - C_2 \frac{1}{2} \sum_{i=1}^{m} \left[f\left(x^{(i)}\right) - y^{(i)}\right]^2$$

- Finding the parameters:

$$\underset{W,b}{\operatorname{argmax}} \log l(W, b) = \underset{W,b}{\operatorname{argmin}} \frac{1}{2m} \sum_{i=1}^{m} \left\| f\left(x^{(i)}\right) - y^{(i)} \right\|^2$$

- The argument: “loss” function (mean squared error)

- Optimization algorithms to solve $\underset{W,b}{\operatorname{argmin}} \mathrm{L}(W, b)$

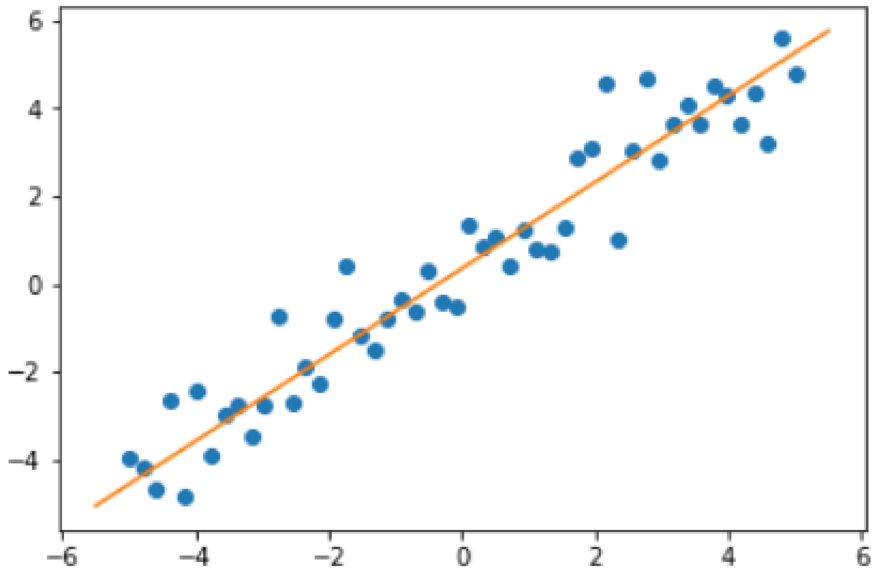
- Problem:
  - Given: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}, x \in \mathbb{R}^N, y \in \mathbb{R}^K$
  - We want a model: $f: \mathbb{R}^N \to \mathbb{R}^K, f(x^{(i)})\ is\ close\ to\ y^{(i)}, \forall i$
- Linear regression: $f(x) = Wx + b, where\ W \in \mathbb{R}^{K\times N}, b \in \mathbb{R}^K$
- To model the dataset with f, we have to solve:

$$\underset{W,b}{\operatorname{argmin}} L(W,b) = \underset{W,b}{\operatorname{argmin}} \left[\frac{1}{2m}\sum_{i=1}^{m}\left\|f(x^{(i)}) - y^{(i)}\right\|^2\right]$$

Linear regression example

```
In [3]: x.shape
Out[3]: (50, 1)

In [4]: y.shape
Out[4]: (50, 1)

In [5]: def linear_regression(W, b, X):
            return W*X+b

        def loss(model, X, Y):
            m = float(len(X))
            def get_value(W, b):
                difference = np.linalg.norm(model(W, b, X)-Y)
                return np.sum(difference**2)/2./m
            return get_value

In [6]: W = np.random.randn(1,1)
        b = np.random.randn(1,1)

        lin_reg_loss = loss(linear_regression, x, y)  # function of W, b

        W, b = optimize(lin_reg_loss, W, b)
```

```
In [7]: x_new = np.linspace(-5.5,5.5, 100)
        y_pred = lin_reg(W,b, x_new)

In [9]: plt.plot(x,y, 'o')
        plt.plot(x_new, y_pred)

Out[9]: [<matplotlib.lines.Line2D at 0x20065fae438>]
```



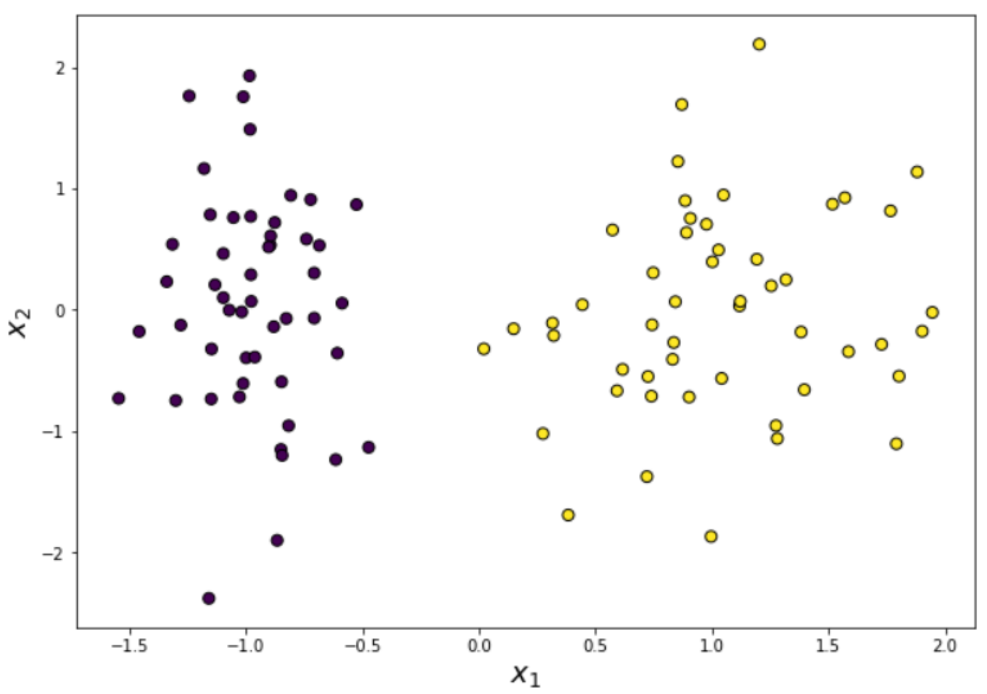
- Problem:

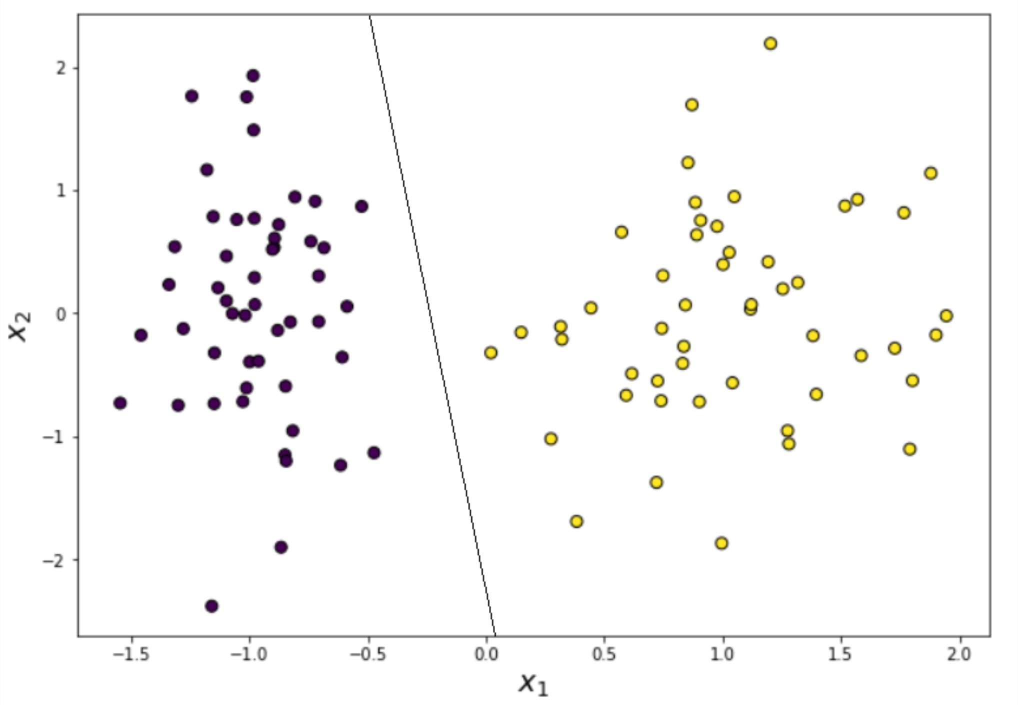
  Given: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}, x \in \mathbb{R}^N, y \in \{0,1\}$

  We want a model: $f\colon \mathbb{R}^N \to \{0,1\}, f(x^{(i)})\ is\ close\ to\ y^{(i)}, \forall i$

- Linear: $g\colon \mathbb{R}^N \to \mathbb{R}$ is linear: $\mathrm{g}(x) = Wx + b, where\ W \in \mathbb{R}^{1\times N}, b \in \mathbb{R},$

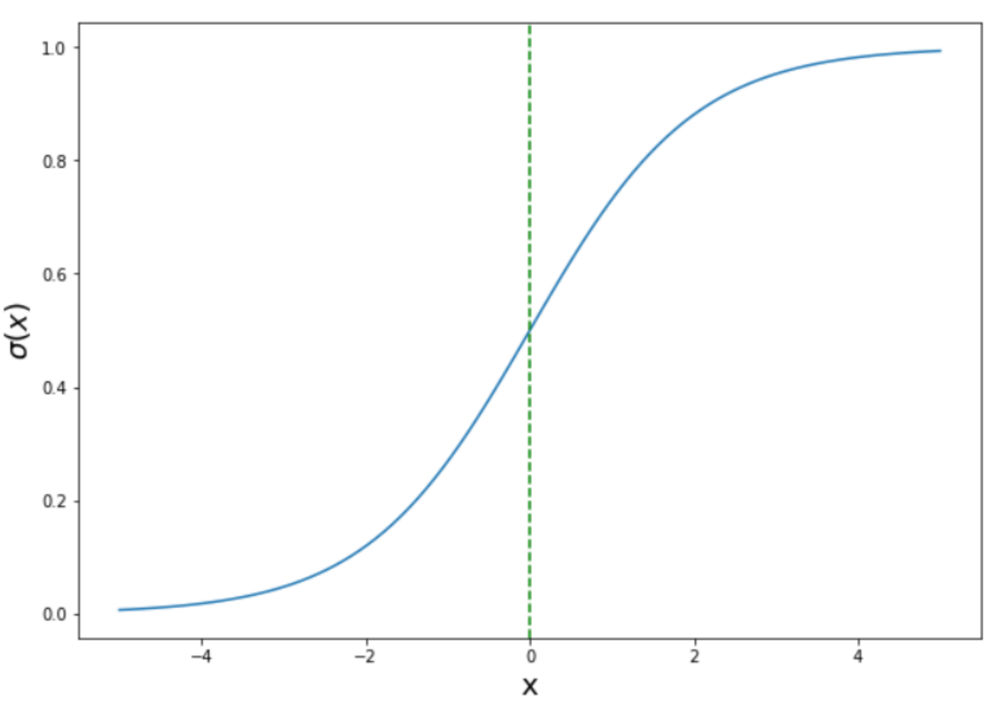
- But we want $\{0,1\}$ prediction (not $\mathbb{R}$ values). Solution: predict 1, when $g(x) > 0$; predict 0, when $g(x) < 0$





Logistic regression

- We want $\{0,1\}$ prediction (not $\mathbb{R}$ values). Solution: predict 1, when $g(x) > 0$; predict 0, when $g(x) < 0$
- Instead of thresholding g at 0, we map to $[0,1]$
- We can interpret it as the probability of predicting class 1: $P_g(y = 1|x)$
- Mapping: sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$



- Logistic regression: $f: \mathbb{R}^n \to [0,1],\ f(x) = \sigma(Wx + b)$
- Interpretation: $f(x) = P(y = 1|x)$
- Question: what is the Loss function?

- Linear regression: $P\big((x,y)|\,W,b\big)\sim \mathrm{Normal}(Wx+b,\mu=y,\sigma)$
- Logistic regression: binary classification problem

$$P\big(y|x^{(i)}\big)=\sigma\big(Wx^{(i)}+b\big)$$

- Probability of data point: Bernoulli distribution

$$P\left(\big(x^{(i)},y^{(i)}\big)|\,W,b\right)=P\big(y|x^{(i)}\big)^{y^{(i)}}\left(1-P\big(y|x^{(i)}\big)\right)^{1-y^{(i)}}$$

- Log-probability (to maximize):

$$\log \mathrm{P}\left(\big(x^{(i)},y^{(i)}\big)|\,W,b\right)=y^{(i)}\log P\big(y|x^{(i)}\big)+\big(1-y^{(i)}\big)\log\left(1-P\big(y|x^{(i)}\big)\right)$$

- Loss function: binary cross-entropy (to minimize)

$$L(W,b)=-\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log P\big(y|x^{(i)}\big)+\big(1-y^{(i)}\big)\log\left(1-P\big(y|x^{(i)}\big)\right)\right]$$

- Problem:
  - Given: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}, x \in \mathbb{R}^N, y \in \{0,1\}$
  - We want a model: $f: \mathbb{R}^N \to \{0,1\}, f(x^{(i)})\ is\ close\ to\ y^{(i)}, \forall i$

- Logistic regression: $P(y|x) = \sigma(Wx + b),\ x \in \mathbb{R}^N, W \in \mathbb{R}^{1\times N}, b \in \mathbb{R}$

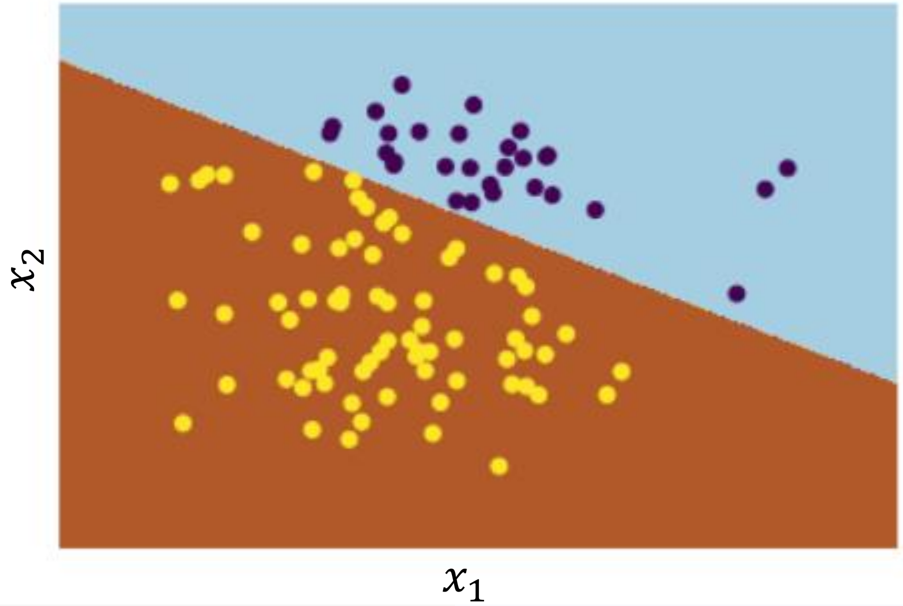
- To model the dataset with $P(y|x)$, we have to solve:

$$\underset{W,b}{\operatorname{argmin}} \left[-\frac{1}{2m} \sum_{i=1}^{m} \left[y^{(i)} \log P(y|x^{(i)}) + (1 - y^{(i)}) \log\left(1 - P(y|x^{(i)})\right) \right] \right]$$

Logistic regression example

```
In [64]: x.shape
Out[64]: (100, 2)

In [65]: labels.shape
Out[65]: (100, 1)

In [66]: labels[0:10].T
Out[66]: array([[0, 1, 1, 1, 1, 1, 1, 1, 1, 0]])

In [67]: def sigmoid(X):
             return 1./(1.+np.exp(-X))

         def logistic_regression(W, b, X):
             return sigmoid(np.matmul(X,W)+b)

         def loss(model, X, Y):
             m = float(len(X))
             def get_value(W,b):
                 prediction = model(W,b,X)
                 return -np.dot(Y.T,np.log(prediction))+np.dot((1-Y).T,np.log(1-prediction))/2./m
             return get_value

In [68]: lr_loss = loss(logistic_regression, x, labels)

         W0 = np.random.randn(2,1)
         b0 = np.random.randn(1,1)

         W,b = optimize(lr_loss, W0, b0)
```

```
In [69]: plot_decision_boundary(logistic_regression, W,b, x, labels)
```



- Problem:
  - Given: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}, x \in \mathbb{R}^N, y \in \{0,1,\ldots,K\}$
  - For example:
    - $y^{(1)} = 0$ means $x^{(1)}$ is a cat picture
    - $y^{(2)} = 1$ means $x^{(2)}$ is a dog picture, etc.
- Instead of predicting discrete values, it's more meaningful to predict class probability
- How can we transform $y \in \{0,1,\ldots,K\}$ to probabilities?

- We want:

$$\text{oh}\colon \{0,1,\dots,K\} \to [0,1]^K$$
$$\sum_{i=0}^{K} \text{oh}(\text{y}_i) = 1$$

- One-hot encoding:

$$y = l \xrightarrow{one-hot} oh(y)_l = 1, oh(y)_i = 0, i = 0,\dots,l-1, l+1,\dots K$$

- Example: $K = 2$

$$y = 0 \;\to \begin{bmatrix}1\\0\\0\end{bmatrix} \qquad y = 1 \;\to \begin{bmatrix}0\\1\\0\end{bmatrix} \qquad y = 2 \;\to \begin{bmatrix}0\\0\\1\end{bmatrix}$$

- Notation: $y_k = oh(y)_k$

K-class logistic regression

- Model (linear): $z = Wx + b \in \mathbb{R}^K$, $x \in \mathbb{R}^N, W \in \mathbb{R}^{K\times N}$, $b \in \mathbb{R}^K$

- Probability: softmax of z:

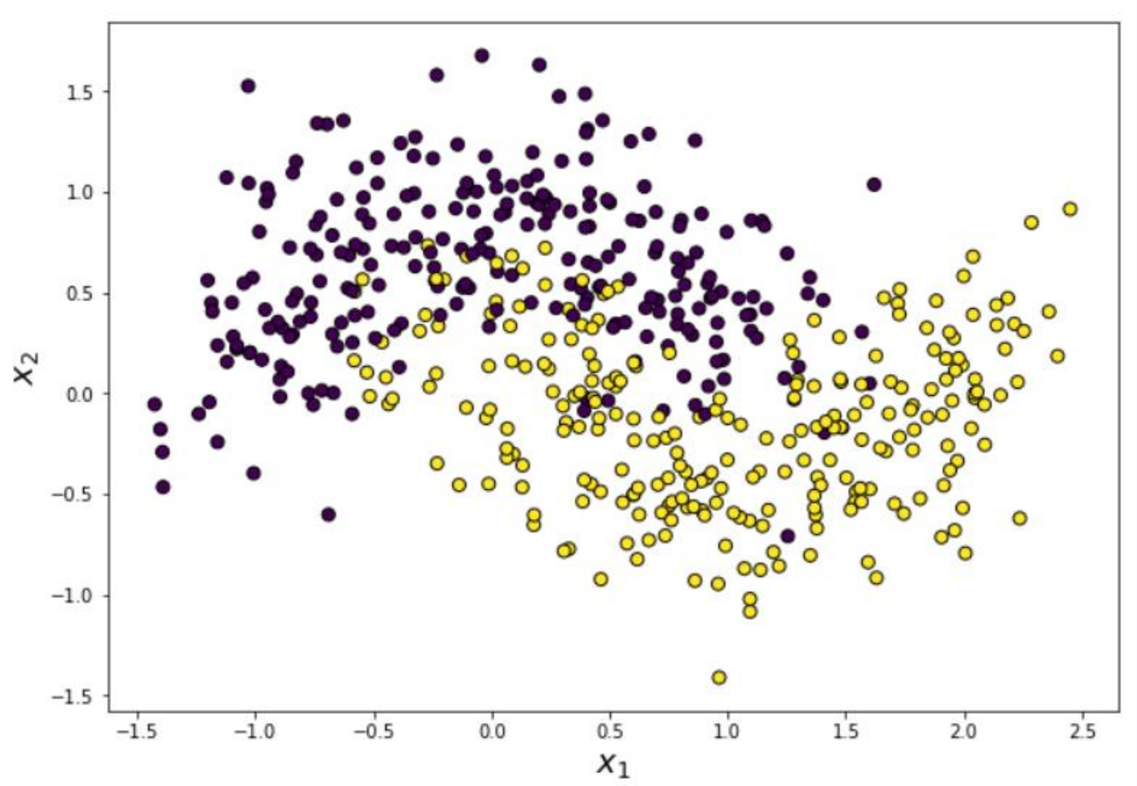
$$P(y|x) = \frac{1}{\sum_{j=0}^{K} e^{z_j}} \begin{bmatrix} e^{z_0} \\ \vdots \\ e^{z_K} \end{bmatrix}$$

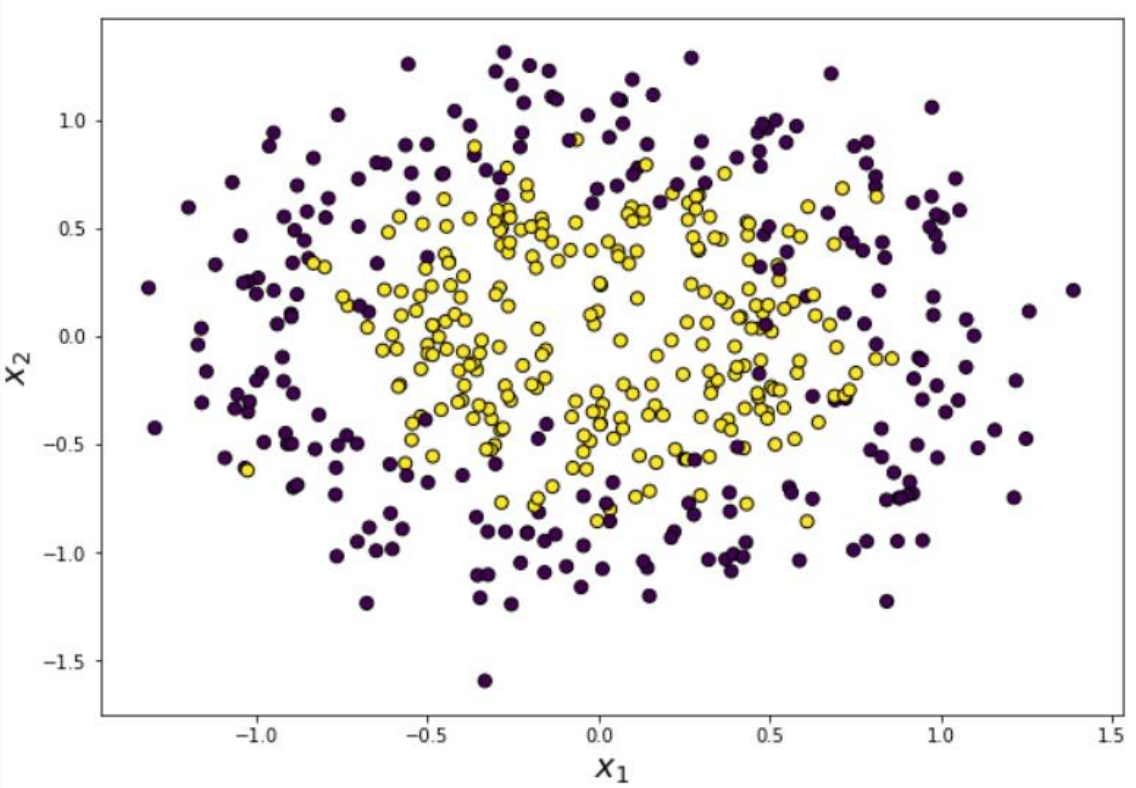
- Interpretation: $P(y|x)_k = probability\ of\ x\ being\ in\ class\ number\ k$

- To model the dataset with $P(y|x)$, we have to solve (cross-entropy between target and predicted $P(y|x)$ distributions):

$$\underset{W,b}{\operatorname{argmin}} \left[-\frac{1}{2m} \sum_{i=1}^{m} \sum_{k=0}^{K} {y^{(i)}}_k \log P\left(y|x^{(i)}\right)_k \right]$$

- Regression, classification: function approximation (“scientific” regression: we know the model function)
- Most of the times: linear approximation doesn’t work
- We need complex function approximators



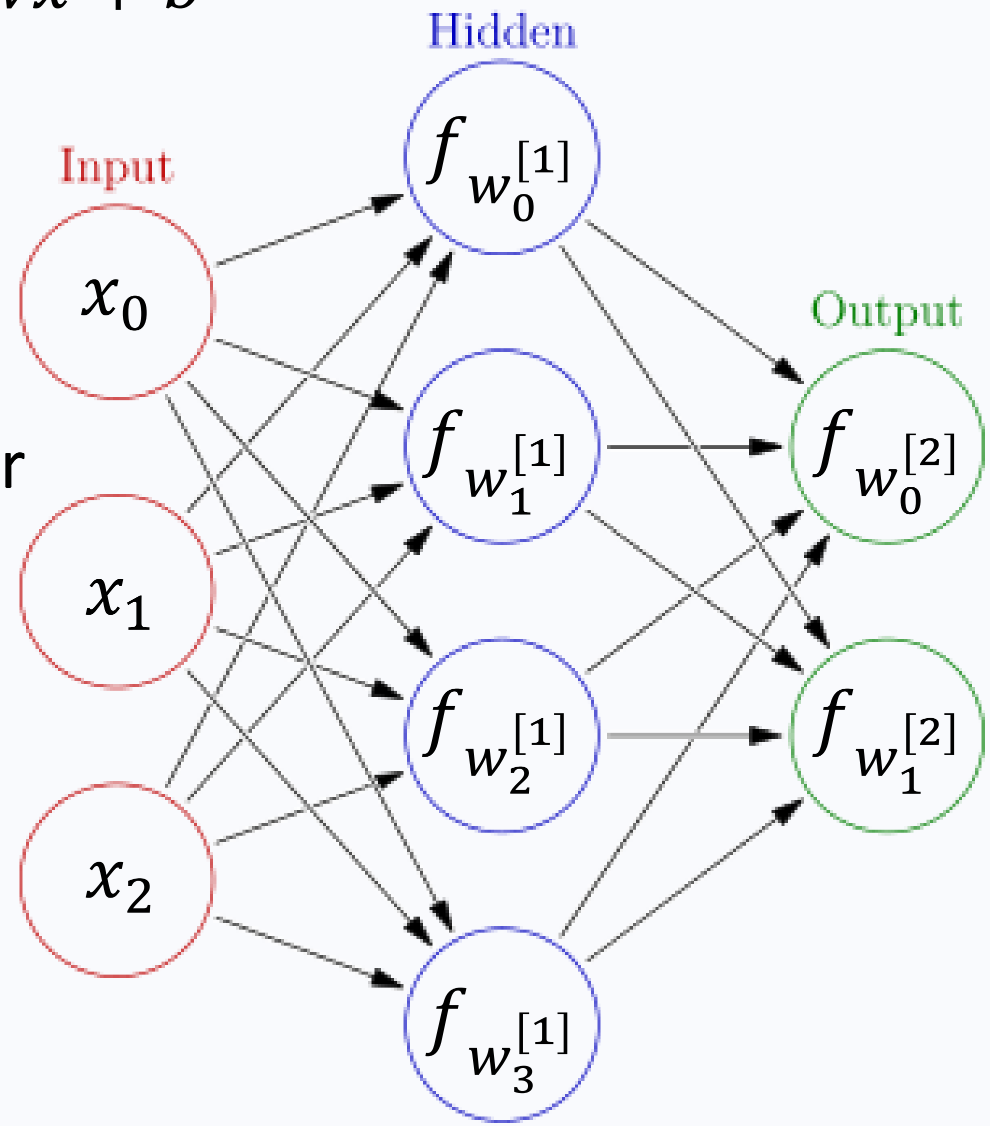


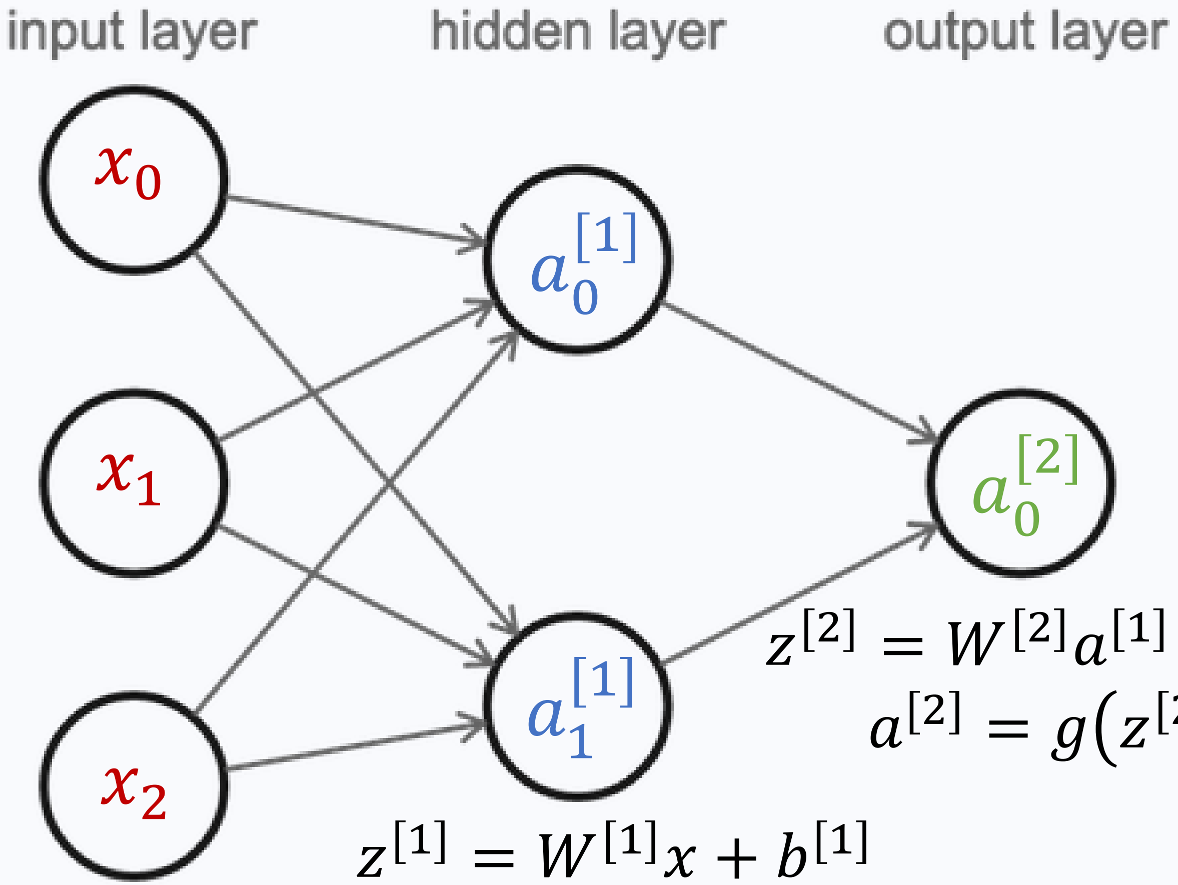
- Regression:

$$f: \mathbb{R}^N \to \mathbb{R}, \qquad f(x) = wx + b$$

- Non-linear function: g (tanh, sigmoid)
- Neuron: $f_w: \mathbb{R}^N \to \mathbb{R},\ f_w(x) = g(wx + b)$
- Layer: we stack n neurons together
- Neural network: we stack layers after each other
- Notation:
  - *i neuron in l layer*: $w_i^{[l]}$
  - Number of neurons in layer l: $n^{[l]}$
  - Activations:

$$z_i^{[l]} = w_i^{[l]} a^{[l-1]} + b_i^l, \qquad a_i^{[l]} = g(z_i^{[l]})$$





$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$
$$a^{[2]} = g(z^{[2]})$$

$$z^{[1]} = W^{[1]} x + b^{[1]}$$
$$a^{[1]} = g(z^{[1]})$$

$$z_0^{[1]} = w_{0,0}^{[1]} x_0 + w_{0,1}^{[1]} x_1 + w_{0,2}^{[1]} x_2 + b_0^{[1]}$$
$$a_0^{[1]} = g\left(z_0^{[1]}\right)$$

$$z_1^{[1]} = w_{1,0}^{[1]} x_0 + w_{1,1}^{[1]} x_1 + w_{1,2}^{[1]} x_2 + b_0^{[1]}$$
$$a_1^{[1]} = g\left(z_1^{[1]}\right)$$

$$z_0^{[2]} = w_{0,0}^{[2]} a_0 + w_{0,1}^{[2]} a_1 + b_0^{[2]}$$
$$a_0^{[2]} = g\left(z_0^{[2]}\right)$$

<u>Vectorization</u>: $x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$, $a^{[1]} = \begin{bmatrix} a_0^{[1]} \\ a_1^{[1]} \end{bmatrix}$, $W^{[1]} = \begin{bmatrix} w_{0,0}^{[1]} & w_{0,1}^{[1]} & w_{0,2}^{[1]} \\ w_{1,0}^{[1]} & w_{1,1}^{[1]} & w_{1,2}^{[1]} \end{bmatrix}$, $b^{[1]} = \begin{bmatrix} b_0^{[1]} \\ b_1^{[1]} \end{bmatrix}$

L-layer neural network

$\mathrm{x} \in \mathbb{R}^N, \mathrm{y} \in \mathbb{R}^K$, neural network: $\mathbb{R}^N \rightarrow \mathbb{R}^K$

$$z^{[1]} = W^{[1]}x + b^{[1]}, \qquad W\colon n^{[1]} \times N, \qquad b\colon n^{[1]} \times 1$$
$$a^{[1]} = g\left(z^{[1]}\right)$$

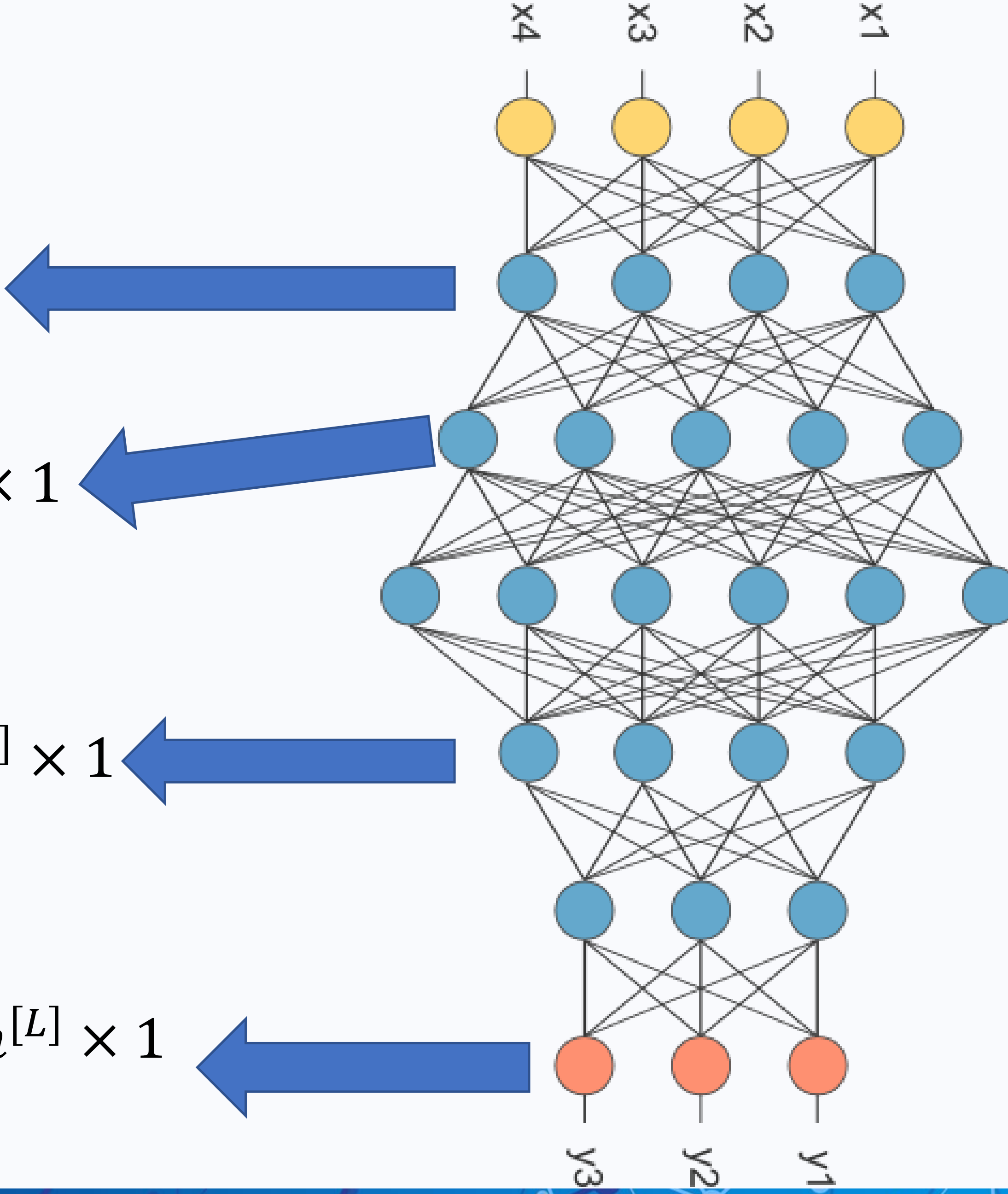
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}, \qquad W\colon n^{[2]} \times n^{[1]}, \qquad b\colon n^{[2]} \times 1$$
$$a^{[2]} = g\left(z^{[2]}\right)$$

$\vdots$

$$z^{[i]} = W^{[i]}a^{[i-1]} + b^{[i]}, \qquad W\colon n^{[i]} \times n^{[i-1]}, \qquad b\colon n^{[i]} \times 1$$
$$a^{[i]} = g\left(z^{[i]}\right)$$

$\vdots$

$$z^{[L]} = W^{[L]}a^{[L-1]} + b^{[L]}, \qquad W\colon n^{[L]} \times n^{[L-1]}, \qquad b\colon n^{[L]} \times 1$$
$$y = a^{[L]} = g\left(z^{[L]}\right)$$



- How to find weights?
- Given: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}, x \in \mathbb{R}^N, y \in \mathbb{R}^K/[0,1]^K$
- We define a cost function: for example:

$$L = \frac{1}{2m} \sum_{i=1}^{m} \left\| a^{[L](i)} - y^{(i)} \right\|^2 \text{ (regression)}$$

$$L = -\frac{1}{2m} \sum_{i=1}^{m} \sum_{k=0}^{K} y^{(i)}{}_k \log a^{[L](i)}{}_k \text{ (classification)}$$

- Weights:

$$\underset{W^{[1]}, b^{[1]}, \dots, W^{[L]}, b^{[L]}}{\operatorname{argmin}} L\left(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, \dots, W^{[L]}, b^{[L]}\right)$$

Gradient descent

- Problem:

$$\underset{W,b}{\operatorname{argmin}}\, L(W,b)$$

- One solution:

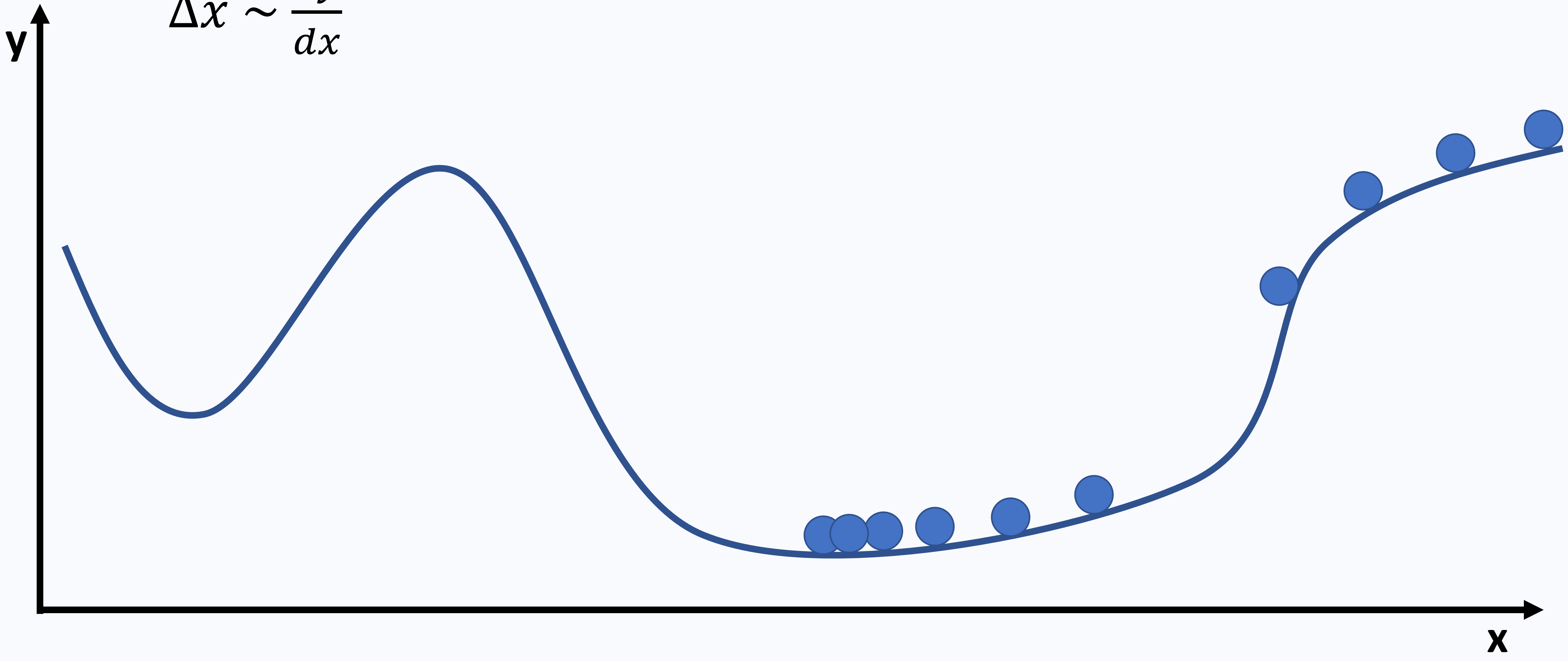
$$\frac{\partial L}{\partial W} = 0, \qquad \frac{\partial L}{\partial b} = 0$$

- Problem: too complicated for neural networks
- Solution: gradient descent

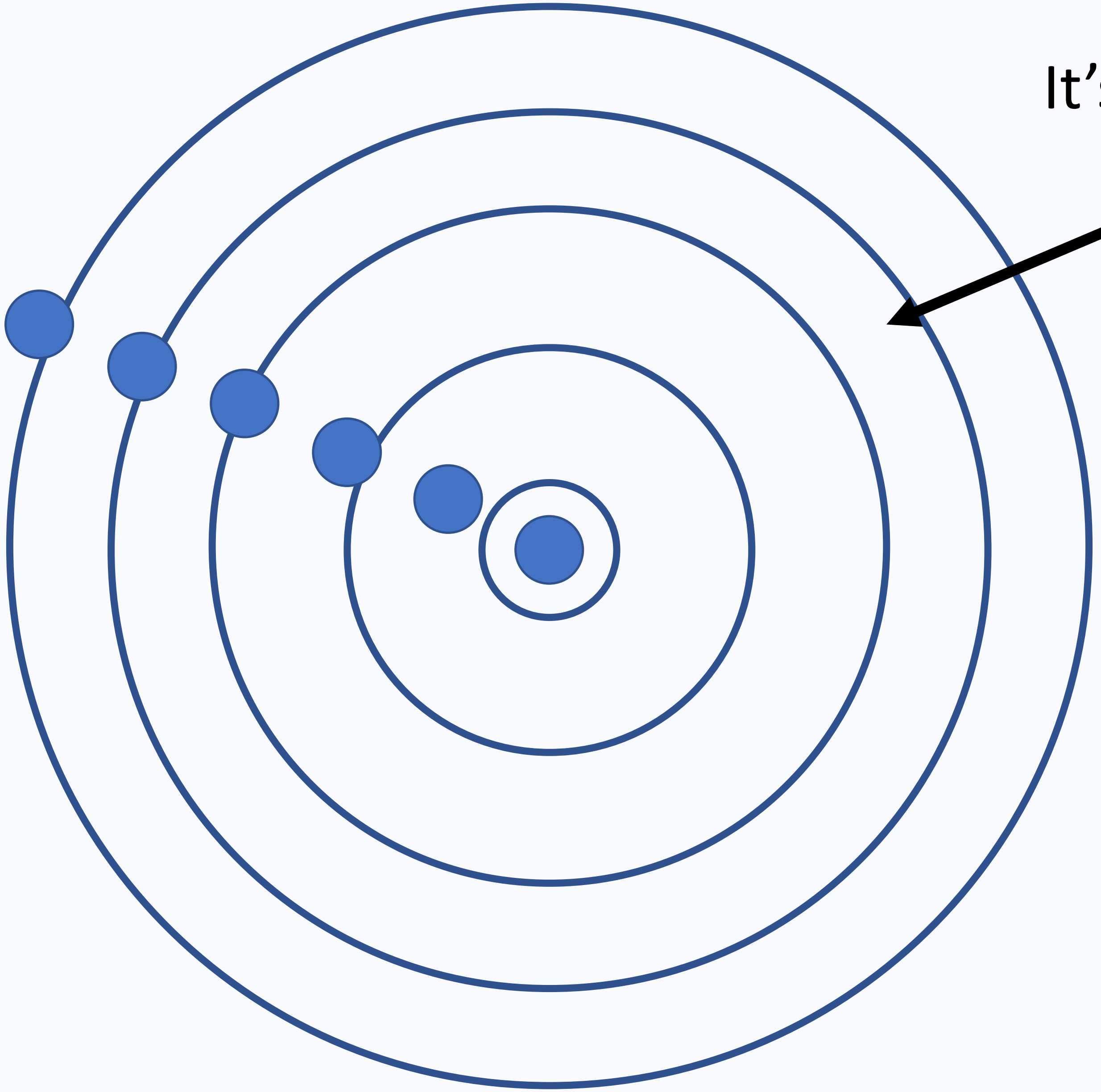
$$\text{repeat} \quad \left\{ \begin{aligned} W &= W - \alpha \frac{\partial L}{\partial W} \\ b &= b - \alpha \frac{\partial L}{\partial b} \end{aligned} \right\}$$

Step size in x is proportional to the derivate.

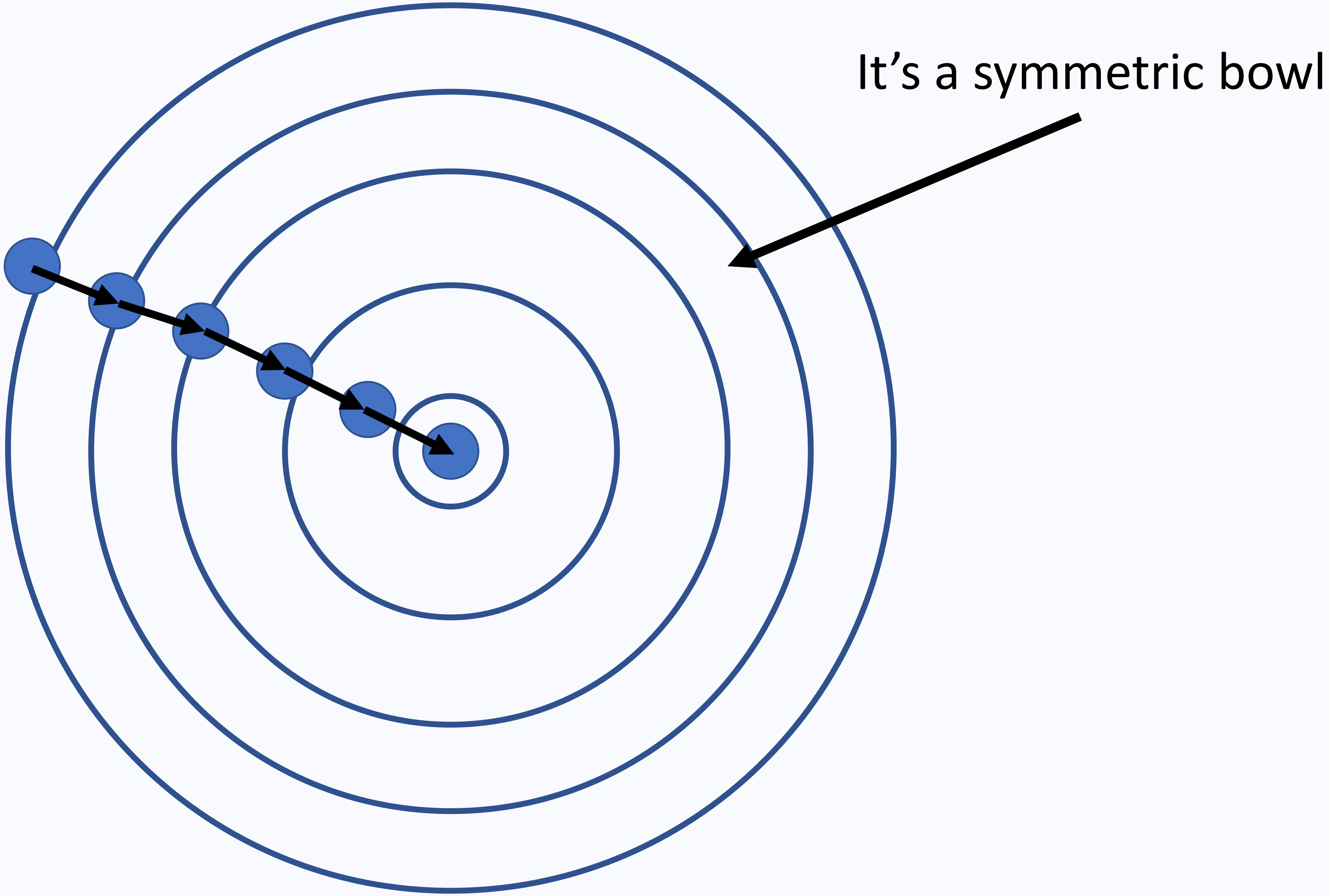
$$\Delta x \sim \frac{dy}{dx}$$



Gradient descent, contour line view.



It’s a symmetric bowl


It’s a symmetric bowl

Imagine sources

- https://hu.pinterest.com/pin/701013498218377093/
- https://www.quora.com/Why-does-YouTube-struggle-to-recommend-quality-videos-to-me
- https://en.wikipedia.org/wiki/Artificial_neural_network
- http://www.opennn.net
- https://stats.stackexchange.com/questions/104738/is-this-the-definition-of-over-fitting