Introduction to Neural Nets

Steven Elsworth

School of Mathematics, The University of Manchester

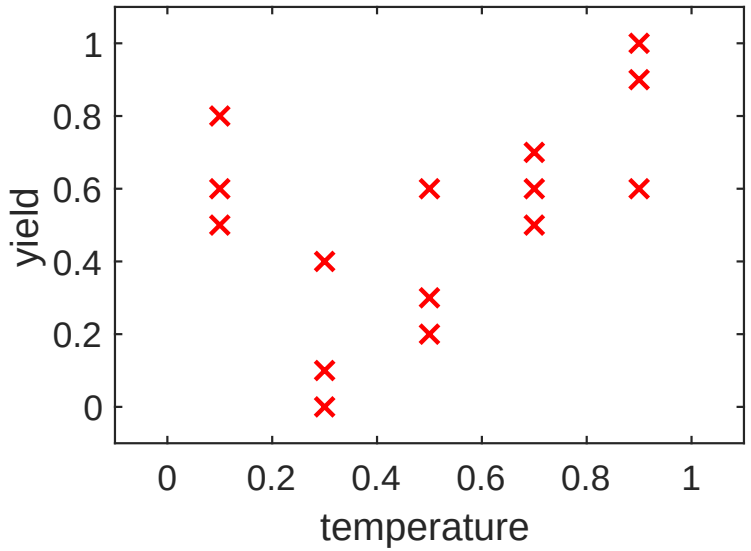
Regression Example

Consider the set of pairs

$$\begin{aligned}\Omega =&\{(0.1,0.8),(0.1,0.5),(0.1,0.6),(0.3,0.1),(0.3,0.4),\\ &(0.3,0),(0.5,0.3),(0.5,0.6),(0.5,0.2),(0.7,0.6),\\ &(0.7,0.7),(0.7,0.5),(0.9,0.9),(0.9,1),(0.9,0.6)\}\subset\mathbb{R}\times\mathbb{R}.\end{aligned}$$

We want to find a function $f:\mathbb{R}\to\mathbb{R}$ such that

$$f(x)=y \text{ for } (x,y)\in\Omega.$$



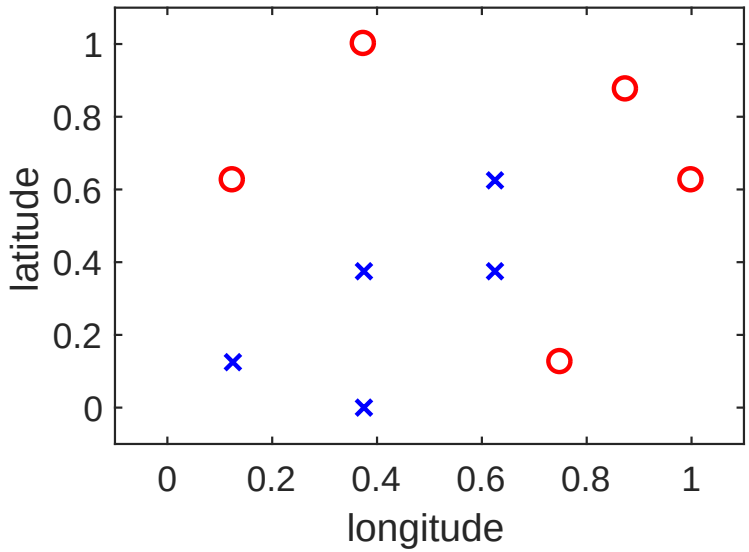
Classification Example

Consider the set of pairs

$$\begin{aligned}\Omega = \{&((0.125, 0.625), 0), ((0.375, 1), 0), ((0.875, 0.875), 0),\\ &((1, 0.625), 0), ((0.75, 0.125), 0), ((0.625, 0.625), 1),\\ &((0.625, 0.375), 1), ((0.375, 0.375), 1), ((0.375, 0), 1),\\ &((0.125, 0.125), 1)\} \subset \mathbb{R}^2 \times \mathbb{R}.\end{aligned}$$

We want to find a function $f : \mathbb{R}^2 \to \mathbb{R}$ such that

$$f(\mathbf{x}) = y \text{ for } (\mathbf{x}, y) \in \Omega.$$

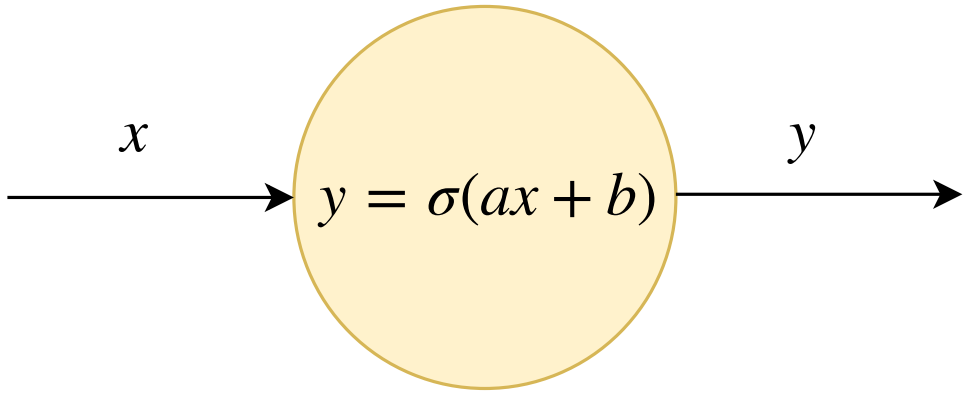


What types of functions can we use?

- Polynomial: $f(x) = a_0 + a_1 x + \cdots + a_n x^n$
- Rational: $f(x) = \frac{a_0 + a_1 x + \cdots + a_p x^p}{b_0 + b_1 x + \cdots + b_q x^q}$
- Trigonometric: $f(x) = sin(ax + b) + cosh(cx + d) + \cdots$
- Multi-input: $f(x, y) = x^2 + y^2 + 5xy + 7$
- Multi-output: $f(x) = (6x^2 + 7, x + 4)$
- Neural network?

A single neuron approach

What if we set $f(x) = \sigma(ax + b)$ `for` $a, b \in \mathbb{R}$, where $\sigma(z)$ is a nonlinear function.



$a, b, \sigma(z)$

Given $\sigma(z) = 1/(1 + \exp^{-z})$ (the sigmoid function), how do we find a and b?

General Framework

$$f(a, b, x) = \sigma(ax + b)$$

- Initialise parameters: Lets choose $a = 0.5$, $b = 0.5$.

General Framework

$$f(a, b, x) = \sigma(ax + b)$$

- Initialise parameters: Lets choose $a = 0.5$, $b = 0.5$.
- Evaluate the function: Define $u(x) = ax + b$ and so $f(x) = \sigma(u(x))$. (Note: a and b are fixed.)

General Framework

$$f(a, b, x) = \sigma(ax + b)$$

- Initialise parameters: Lets choose $a = 0.5$, $b = 0.5$.
- Evaluate the function: Define $u(x) = ax + b$ and so $f(x) = \sigma(u(x))$. (Note: a and b are fixed.)
- Evaluate loss function: $J(a, b) = \frac{1}{2}(y - f(a, b))^2$. (Note: x and y are fixed.) The cost function is always greater than or equal to zero.

What if we permute the weights?

Multi variable Taylor series:

$$f(\mathbf{x}+\Delta\mathbf{x}) = f(\mathbf{x}) + Df(\mathbf{x})f(\Delta\mathbf{x}) + \frac{1}{2}\Delta\mathbf{x}Hf(\mathbf{x})\Delta\mathbf{x} + \cdots,$$

where $Df(\mathbf{x}) = \begin{bmatrix} \frac{\delta f}{\delta x_1} & \cdots & \frac{\delta f}{\delta x_n} \end{bmatrix}$ and $Hf(\mathbf{x}) = \begin{bmatrix} \frac{\delta^2 f}{\delta^2 x_1} & \cdots & \frac{\delta^2 f}{\delta x_1 \delta x_n} \\ \vdots & \ddots & \vdots \\ \frac{\delta^2 f}{\delta x_n \delta x_1} & \cdots & \frac{\delta^2 f}{\delta^2 x_n} \end{bmatrix}$.

Let $\mathbf{p} = [a, b]$, then

$$J(\mathbf{p}+\Delta\mathbf{p}) \approx J(\mathbf{p}) + \begin{bmatrix} \frac{\delta J}{\delta a} & \frac{\delta J}{\delta b} \end{bmatrix} \begin{bmatrix} \Delta a \\ \Delta b \end{bmatrix}$$

How should we permute them?

We want $J(\mathbf{p} + \Delta\mathbf{p}) < J(\mathbf{p})$

So we want $\begin{bmatrix} \frac{\delta J}{\delta a} & \frac{\delta J}{\delta b} \end{bmatrix} \begin{bmatrix} \Delta a \\ \Delta b \end{bmatrix}$ to be as negative as possible.

Cauchy Schwarz Inequality

Given $\mathbf{u}, \mathbf{v} \in \mathbf{R}^n$, $|\mathbf{u}^T\mathbf{v}| \leq \|\mathbf{u}\|_2\|\mathbf{v}\|_2$.

Most negative when $\mathbf{u} = -\mathbf{v}$.

So we set $\Delta a = -\frac{\delta J}{\delta a}$ and $\Delta b = -\frac{\delta J}{\delta b}$

Chain rule

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

Derivative of sigmoid

$$\begin{aligned}\frac{d}{dz}(1+\exp^{-z})^{-1} &= -1(1+\exp^{-z})^{-2}(-\exp^{-z}) \\ &= \frac{1}{1+\exp^{-z}}\frac{(1+\exp^{-z})-1}{1+\exp^{-z}} \\ &= \sigma(z)(1-\sigma(z))\end{aligned}$$

Now

$$\frac{\delta J}{\delta a} = \frac{\delta J}{\delta f}\frac{\delta f}{\delta u}\frac{\delta u}{\delta a} = (f(x)-y)(f(x)(1-f(x)))(x)$$

$$\frac{\delta J}{\delta b} = \frac{\delta J}{\delta f}\frac{\delta f}{\delta u}\frac{\delta u}{\delta b} = (f(x)-y)(f(x)(1-f(x)))$$

General Framework

- Update weights:

$$a = a - \alpha \frac{\delta J}{\delta a}$$

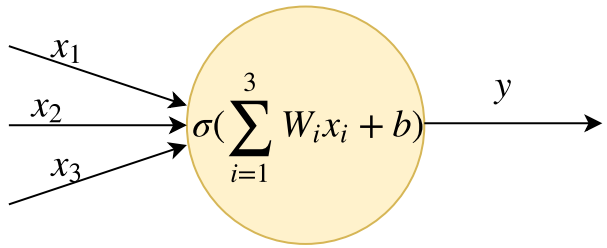
$$b = b - \alpha \frac{\delta J}{\delta b}$$

- Repeat over all data, lots of times!!!

The hyperparameter θ is the learning rate. This should stay be small as our Taylor expansion was a rough approximation.

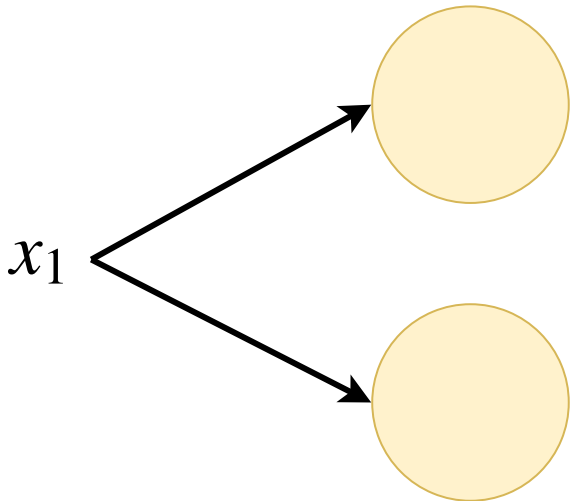
Make function more complex?

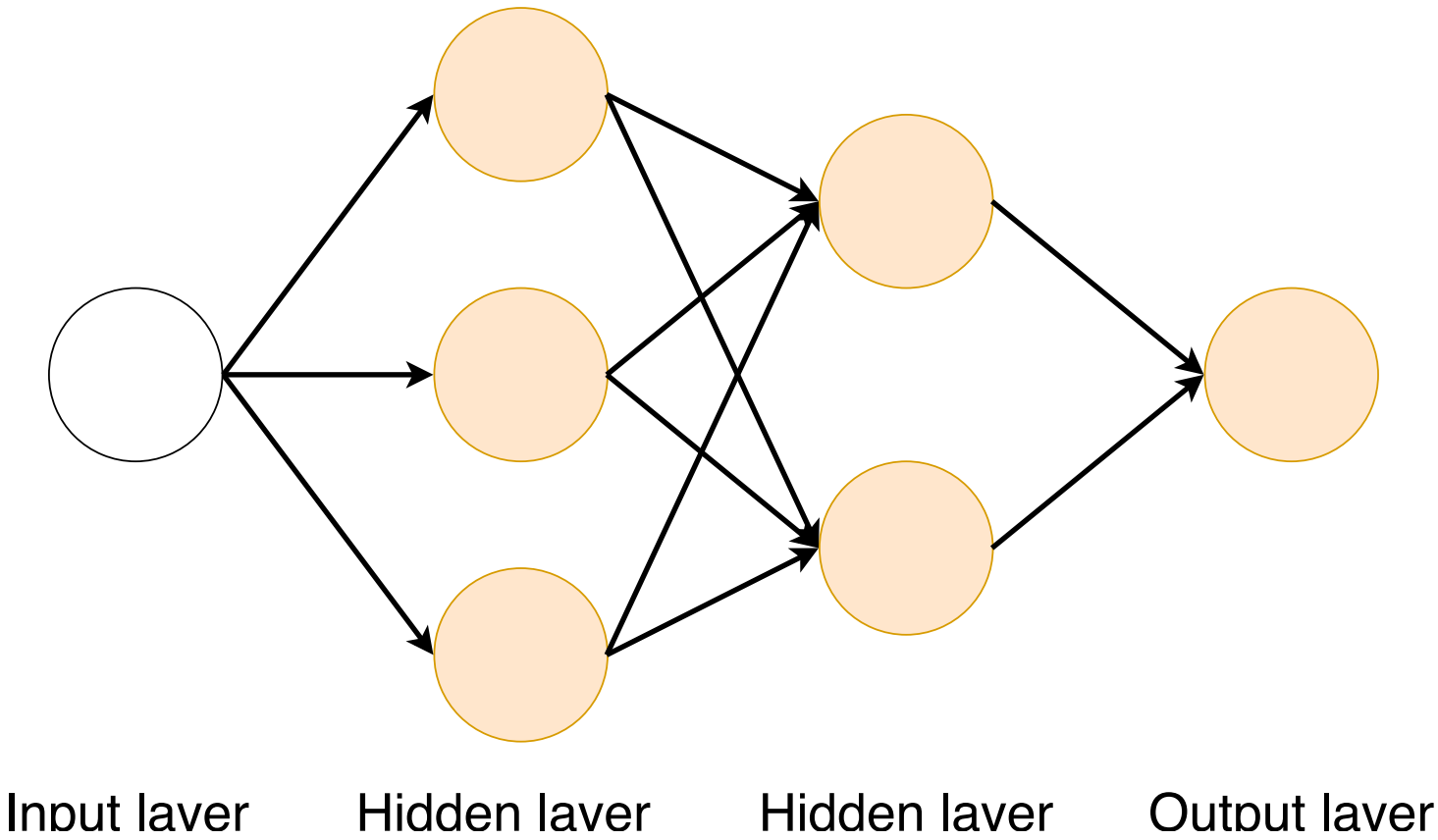
- Each neuron can have more than one input (Note it is still a scalar output).



$W, b, \sigma(z)$

- We can feed our value into more than one neuron.



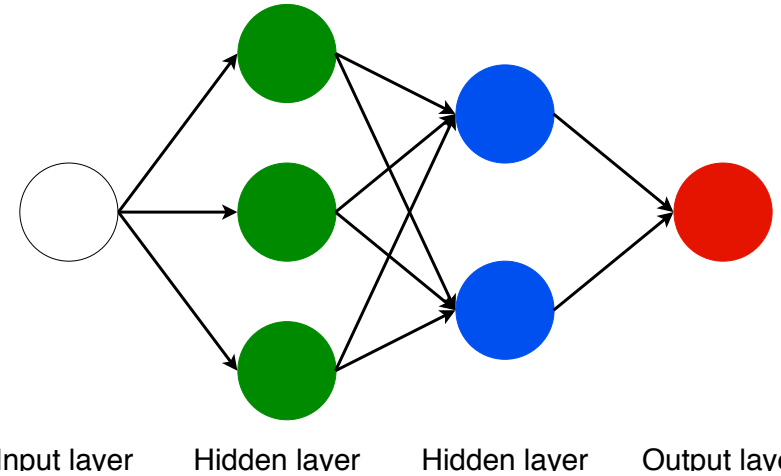

Input layer
Hidden layer
Hidden layer
Output layer

Visualising a net, using MATLAB

```
run net_layer_111.m

run net_layer_121.m

run net_layer_211.m

run net_layer_221.m
```

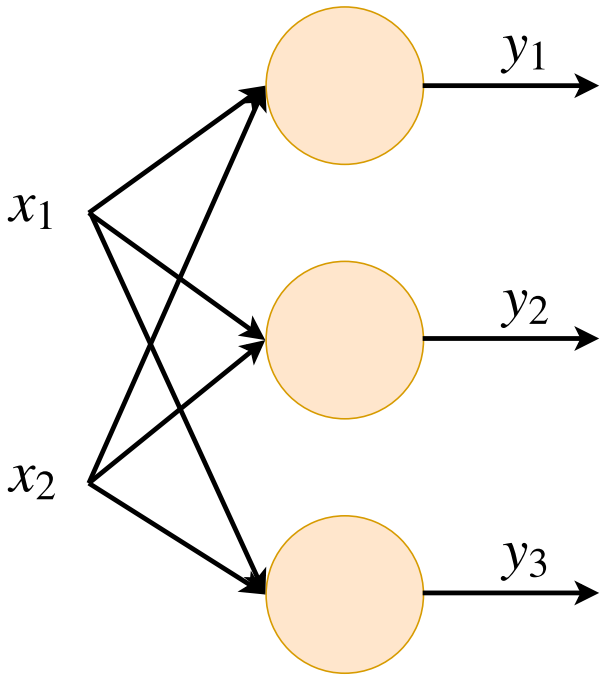
What are neural nets?

A graphical representation of a nonlinear function $f : \mathbb{R}^n \to \mathbb{R}^m$.



$$f(x,p) = \sigma\Bigg(w_{10}\Big[\sigma\Big(w_4[\sigma(w_1 x + b_1)] + w_5[\sigma(w_2 x + b_2)] + w_6[\sigma(w_3 x + b_3)] + b_4\Big)\Big] + w_{11}\Big[\sigma\Big(w_7[\sigma(w_1 x + b_1)] + w_8[\sigma(w_2 x + b_2)] + w_9[\sigma(w_3 x + b_3)] + b_5\Big)\Big] + b_6\Bigg)$$

Layer by layer



$$y_1 = \sigma(w_{11}x_1 + w_{12}x_2 + b_1)$$

$$y_2 = \sigma(w_{21}x_1 + w_{22}x_2 + b_2)$$

$$y_2 = \sigma(w_{31}x_1 + w_{32}x_2 + b_3)$$

$$\mathbf{y} = \sigma(W\mathbf{x} + \mathbf{b})$$

$$W = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix}$$

$$\mathbf{b} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

How are they coded?

- Initialise:

```
W2 = 0.5*randn(3,1); W3 = 0.5*randn(2,3); W4 = 0.5*randn(1,2);
b2 = 0.5*randn(3,1); b3 = 0.5*randn(2,1); b4 = 0.5*randn(1,1);
```

- Feedforward:

```
a2 = sigmoid(x(i),W2,b2);
a3 = sigmoid(a2,W3,b3);
a4 = sigmoid(a3,W4,b4);
```

- Compute error:

```
J = 0.5*(a4 - y(i))^2;
```

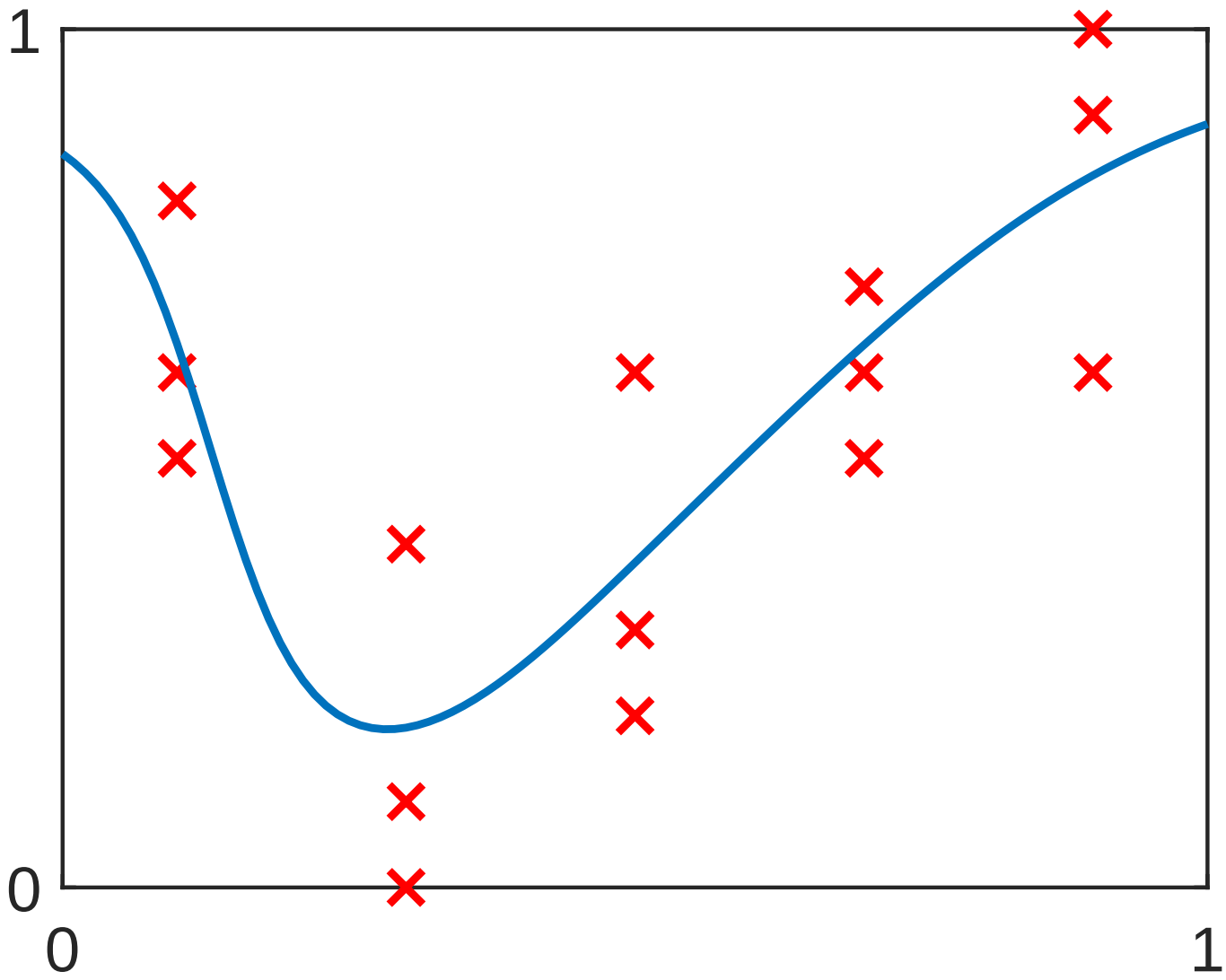
- Backpropogate:

```
delta4 = a4.*(1-a4).*(a4-y(i));
delta3 = a3.*(1-a3).*(W4'*delta4);
delta2 = a2.*(1-a2).*(W3'*delta3);
```

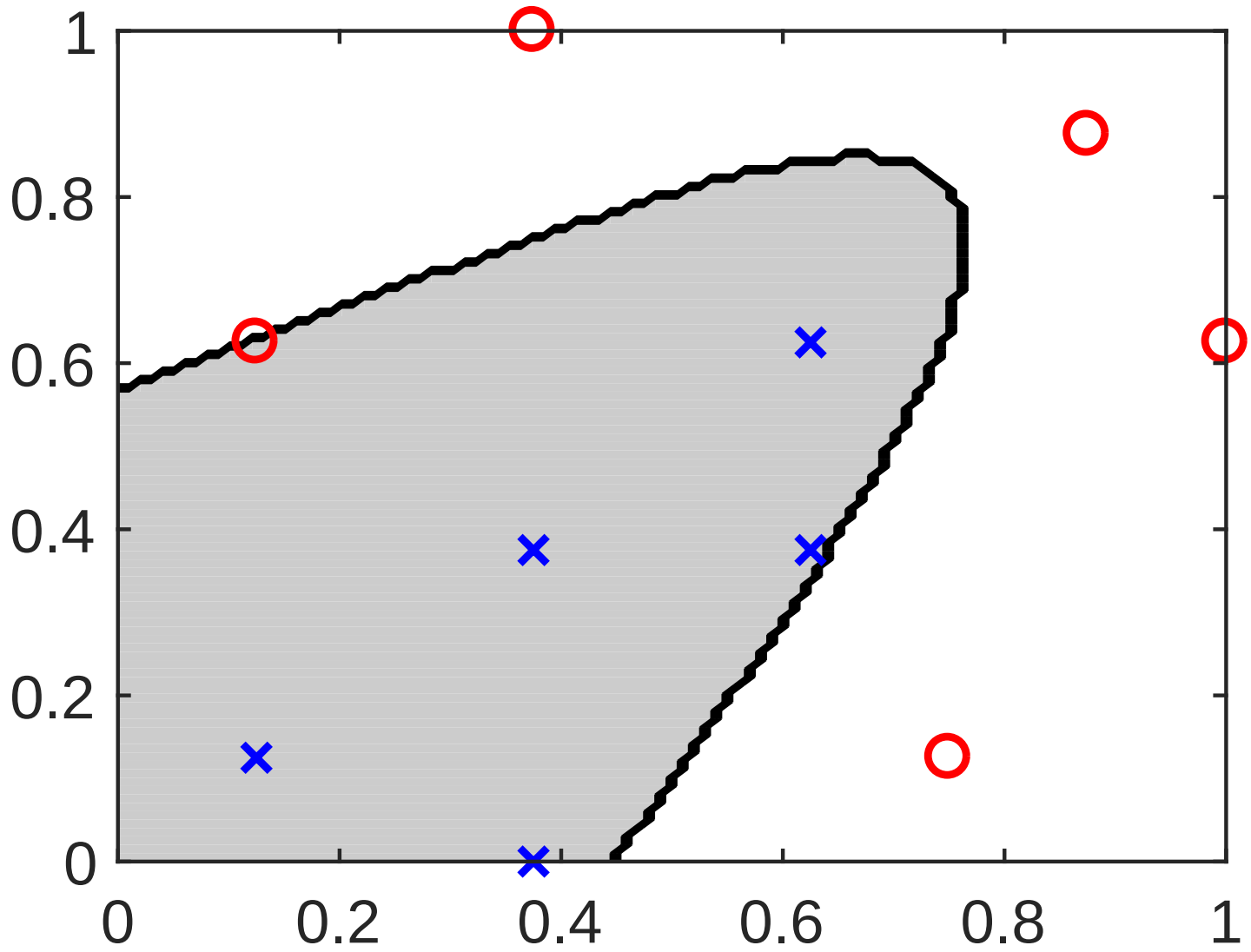
- Update:

```
W2 = W2 - 0.01*delta2*x(i)';
W3 = W3 - 0.01*delta3*a2';
W4 = W4 - 0.01*delta4*a3';
b2 = b2 - 0.01*delta2;
b3 = b3 - 0.01*delta3;
b4 = b4 - 0.01*delta4;
```

Regression example



Classification example



Over 650 new papers a week on arXiv in Machine Learning!

`https://etymoio.github.io/newsletter/`

newsletter.etymo

21st September - 4th October 2018

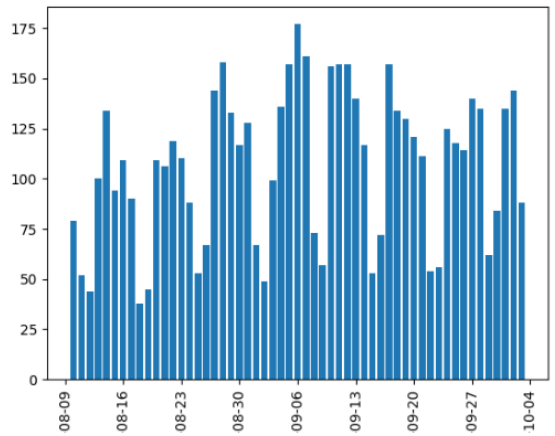
In this newsletter from Etymo, you can find out the latest development in machine learning research, including the most popular datasets used, the most frequently appearing keywords and the important research papers associated with the keywords, and the most trending papers in the past two weeks.

If you and your friends like this newsletter, you can subscribe to our fortnightly newsletters here.

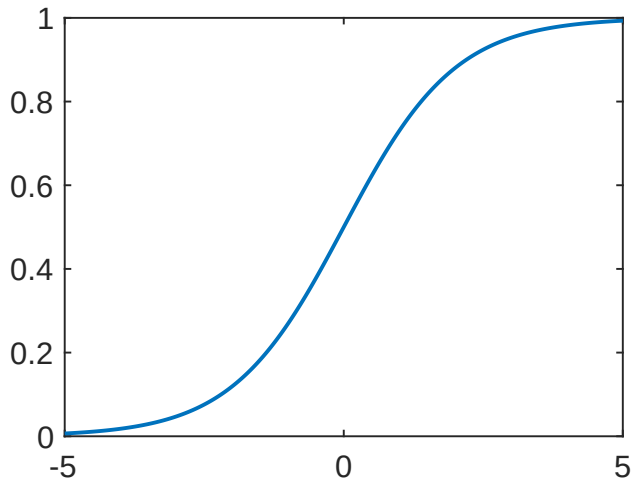
1366 new papers

Etymo added 1366 new papers published in the past two weeks. These newly published papers on average have 3.9 authors for each paper.

The bar diagram below indicates the number of papers published each day from some major sources, including arXiv, DeepMind, Facebook and etc. This diagram also indicates the pattern of publishing machine learning research papers.

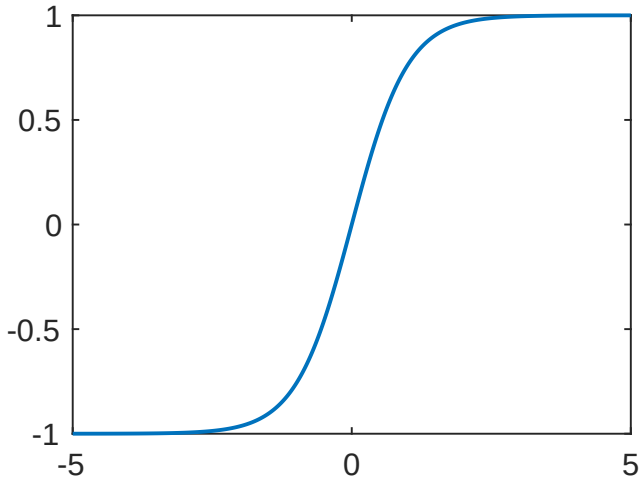


Activation functions



sigmoid: $1/(1+e^{-x})$

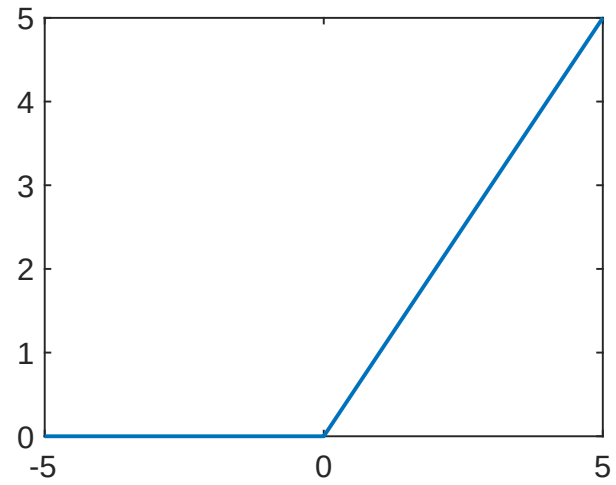
- Smooth step like function.
- Encourages value to end of curve (classification).
- Vanishing gradient problem.
- Output in range $(0, 1)$.

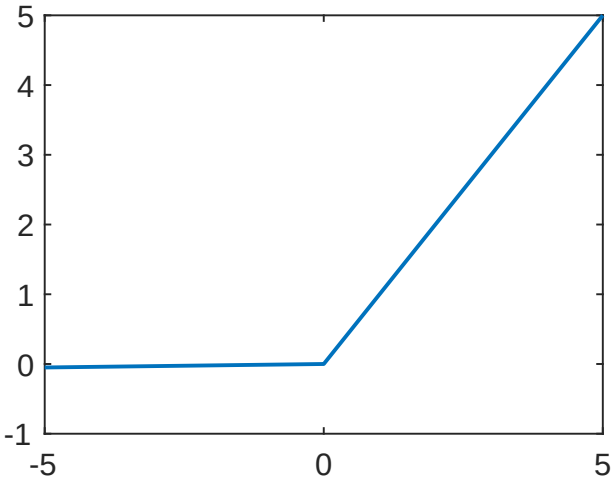


tanh: $\tanh(x)$

- Scaled and shifted sigmoid.
- Vanishing gradient problem.
- Output in range $(-1, 1)$.

Activation functions





relu: $\max(0, x)$

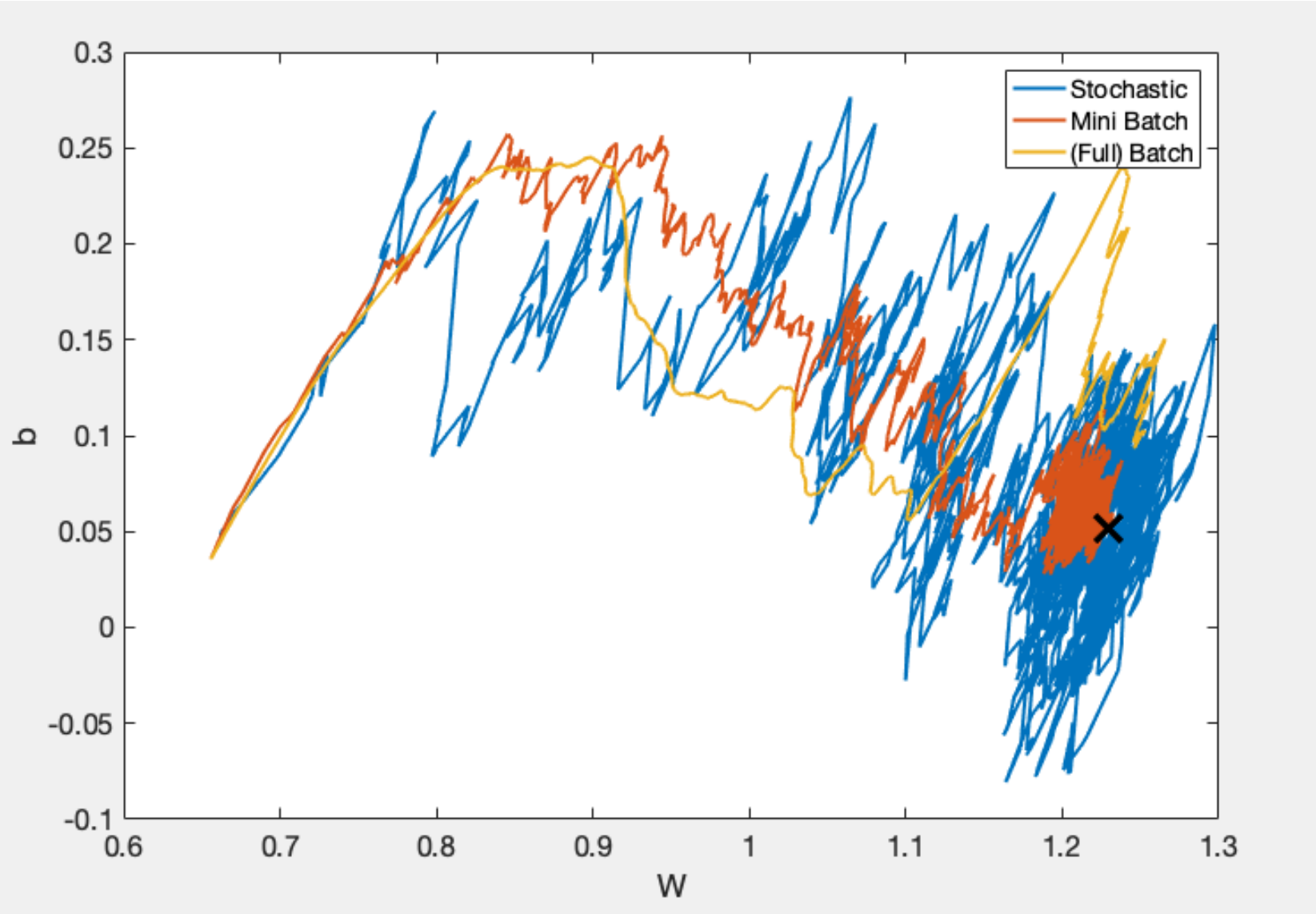
- Encourages sparcity.
- Faster training speeds.
- Can blow up activation.
- Dying relu.
- Cheapest to compute.

leaky relu:

- Prevents dying relu.

Batch/ Mini-batch

`run batch.m`



Gradient Descent Optimisation Algorithms

- Momentum: Add fraction of previous delta to current delta.

$$\Delta p_t = \gamma \Delta p_{t-1} + \theta D(J(p)).$$

- Nesterov accelerated gradient (NAG): Calculate loss when momentum term included.

$$\Delta p_t = \gamma \Delta p_{t-1} + \theta D(J(p - \gamma \Delta p_{t-1}).$$

- Adagrad: Learning rate depends on frequency of occurring feature.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \Delta p_{t-1}$$

- Adadelta RMSProp: Reduce Adagrads aggressive, monotonically decreasing learning rate.
- Adam: Uses both first and second order moments of gradient
- $\vdots$

An overview of gradient descent optimization algorithms*

Sebastian Ruder
Insight Centre for Data Analytics, NUI Galway
Aylien Ltd., Dublin
ruder.sebastian@gmail.com

http://ruder.io/optimizing-gradient-descent/

Parameter initialisation

- **random normal:**
  - Breaks symmetry in weights.
  - Initialised close to zero.
- **Xavier:**
  `http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf`
  - Random weights, range depends on the size of the previous layer of neurons.
  - Allows faster convergence with gradient descent.
- **He et al.:** `https://arxiv.org/abs/1502.01852`
  - Very similar concept to Xavier initialisation.
- **Transfer Learning:**
  - Train net on similar application with large ready available dataset.

REVIEW

doi:10.1038/n

Deep learning

Yann LeCun[1,2], Yoshua Bengio[3] & Geoffrey Hinton[4,5]

"Recent theoretical and empirical results strongly sugguest that local minima are not a series issue in general. Instead the landscape is packed with combinatorially large number of saddle points where the gradient is zero, and the surface curves up in most dimensions and down in the remainder."

Parameters vs Hyperparameters

Parameters:

- Weights
- Bias

Hyper-parameters:

- Learning rate
- Number of neurons per layer
- Number of layers
- Number of iterations
- Activations on layers
- Momentum, dropout, regularisation, . . .

Common Tips

- Larger the net, more data and iterations needed.
- Data augmentation.
- One hot encode, with sigmoid or tanh on final layer.
- Regularisation prevents overfitting.
- Use a pre-trained model? (Transfer learning).
- Batch normalisation layers.
- Apply dropout.
- Select right learning algorithm.

- Higham and Higham: Deep Learning: An Introduction for Applied Mathematicians, https://arxiv.org/pdf/1801.05894.pdf
- Gilbert Strangs: Linear Algebra and Learning from Data, http://math.mit.edu/~gs/learningfromdata/
- Yann LeCun and Yoshua Bengio and Geoffrey Hinton: Deep learning, https://www.nature.com/articles/nature14539
- Ian Goodfellow and Yoshua Bengio and Aaron Courville: Deep learning, https://www.deeplearningbook.org
- Andrew NG: Deep Learning Specialization, https://www.coursera.org/specializations/deep-learning