T3. NEURAL NETWORKS & DEEP LEARNING

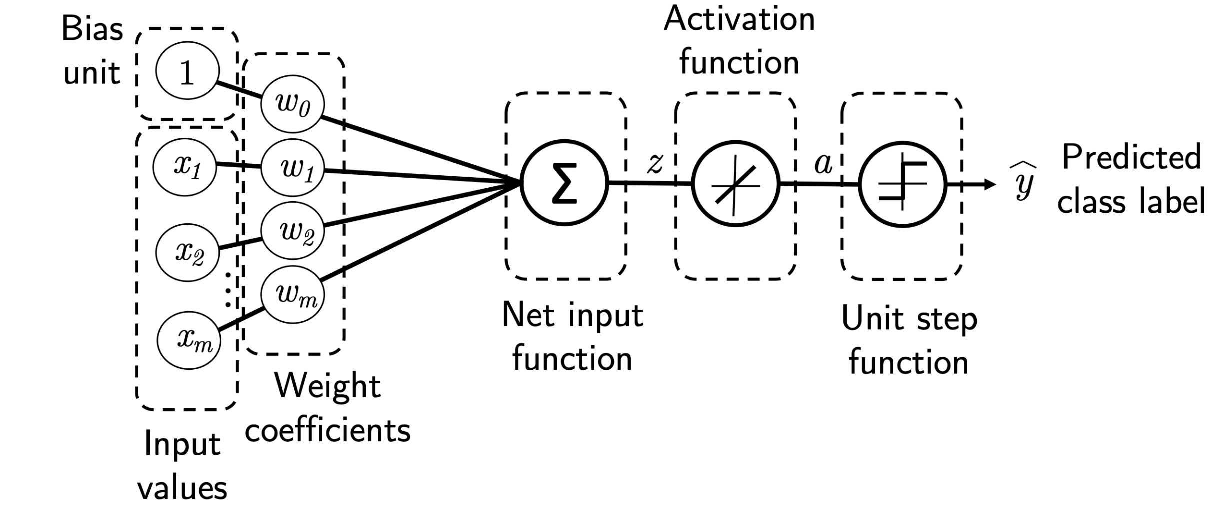
Multilayer Artificial Neural Network

Introduction:

The topics that we will cover in this class are as follows:

- Gaining a conceptual understanding of multilayer NNs
- Implementing the fundamental backpropagation algorithm for NN training from scratch
- Training a basic multilayer NN for image classification

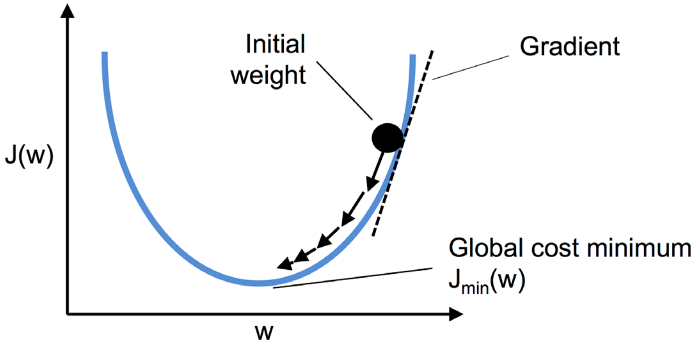
Single-layer neural network recap:



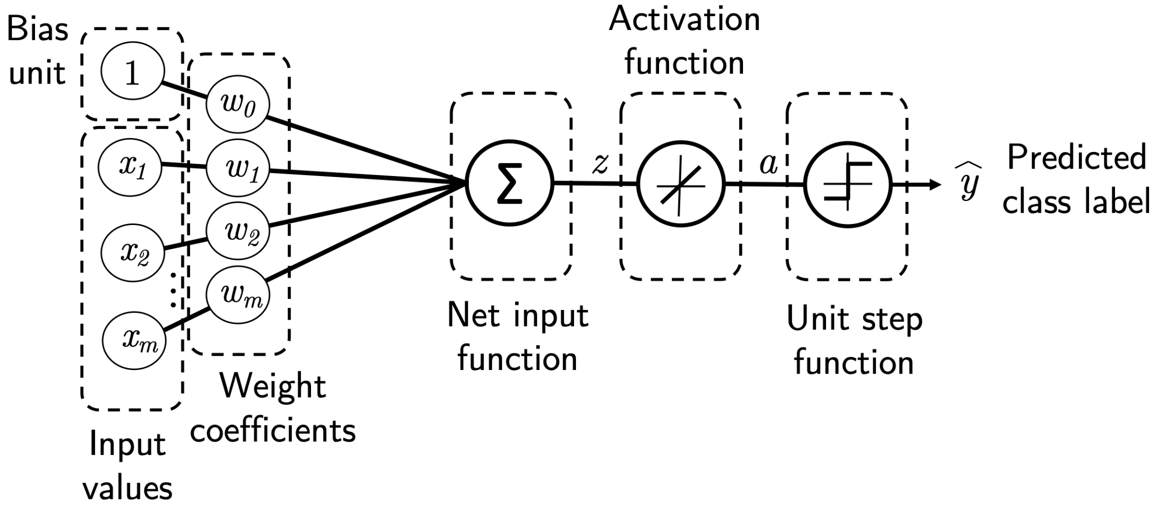
- ADAptive LInear NEuron (Adaline) algorithm for binary classification.
- Gradient descent optimization for model coefficients

$$\Delta \boldsymbol{w} = -\eta \boxed{\nabla J(\boldsymbol{w})} \rightarrow \text{Gradient of } \boxed{J(\boldsymbol{w})}$$

Cost function



Single-layer neural network recap:

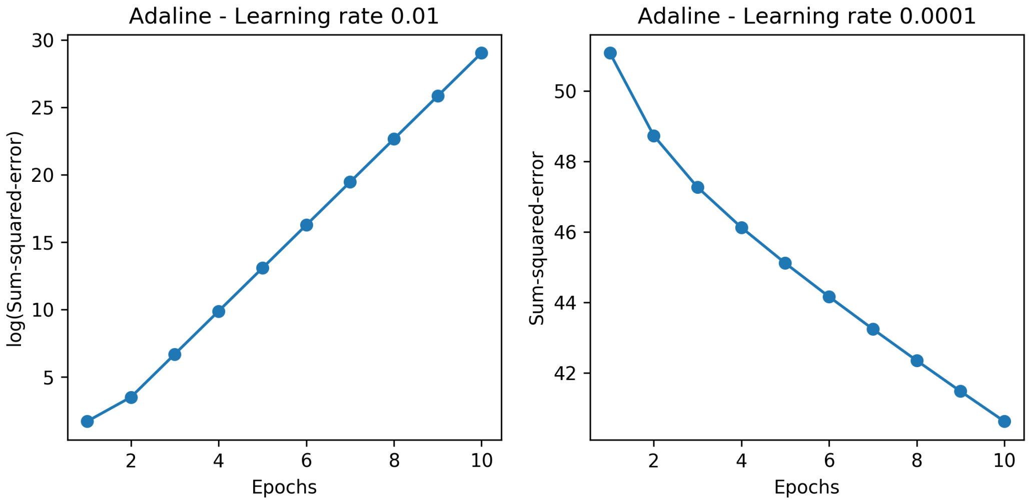


- ADAptive LInear NEuron (Adaline) algorithm for binary classification.
- Gradient descent optimization for model coefficients

$$\Delta \boldsymbol{w} = -\eta \boxed{\nabla J(\boldsymbol{w})}$$

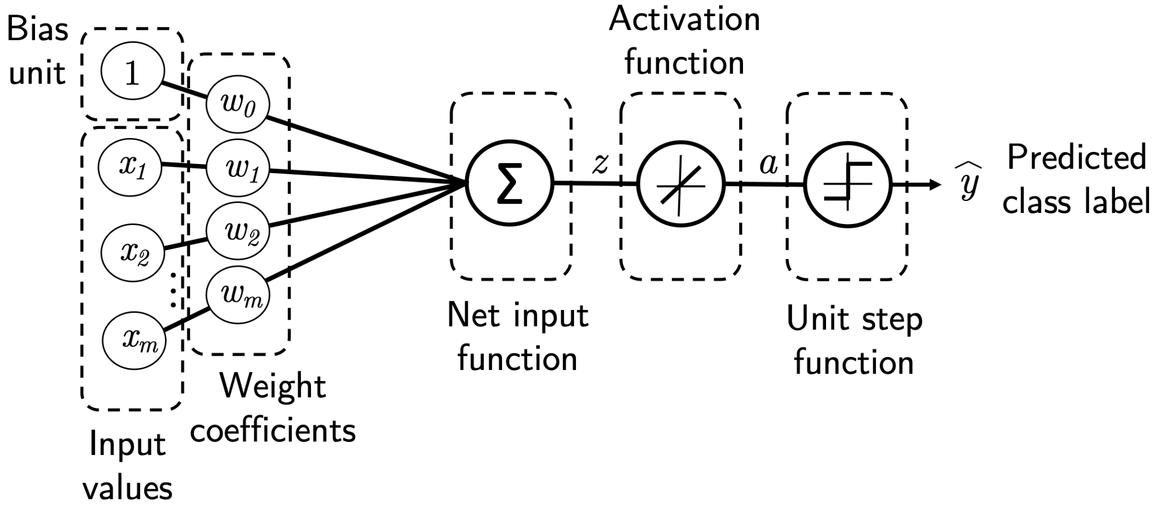
Gradient of $\boxed{J(\boldsymbol{w})}$

Cost function



- Hyperparamters: learning rate $\boldsymbol{\eta}$ and the number of iteration or **epochs.**

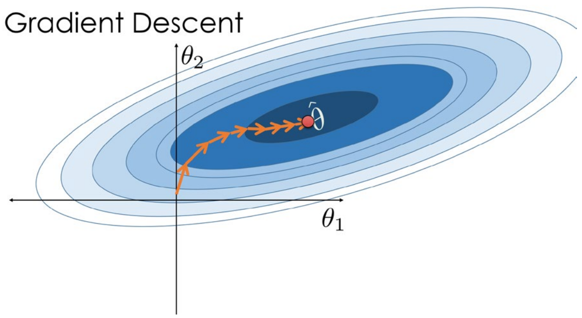
Single-layer neural network recap:

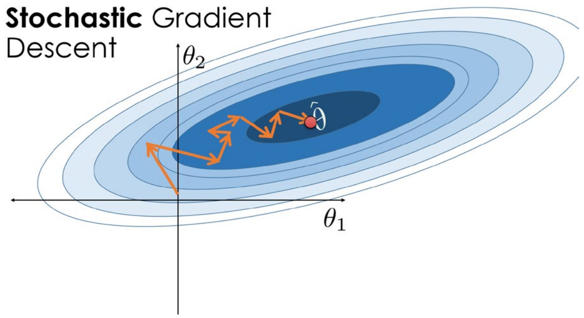


- ADAptive LInear NEuron (Adaline) algorithm for binary classification.
- Gradient descent optimization for model coefficients

$$\Delta \boldsymbol{w} = -\eta \boxed{\nabla J(\boldsymbol{w})} \longrightarrow \text{Gradient of } \boxed{J(\boldsymbol{w})}$$

Cost function





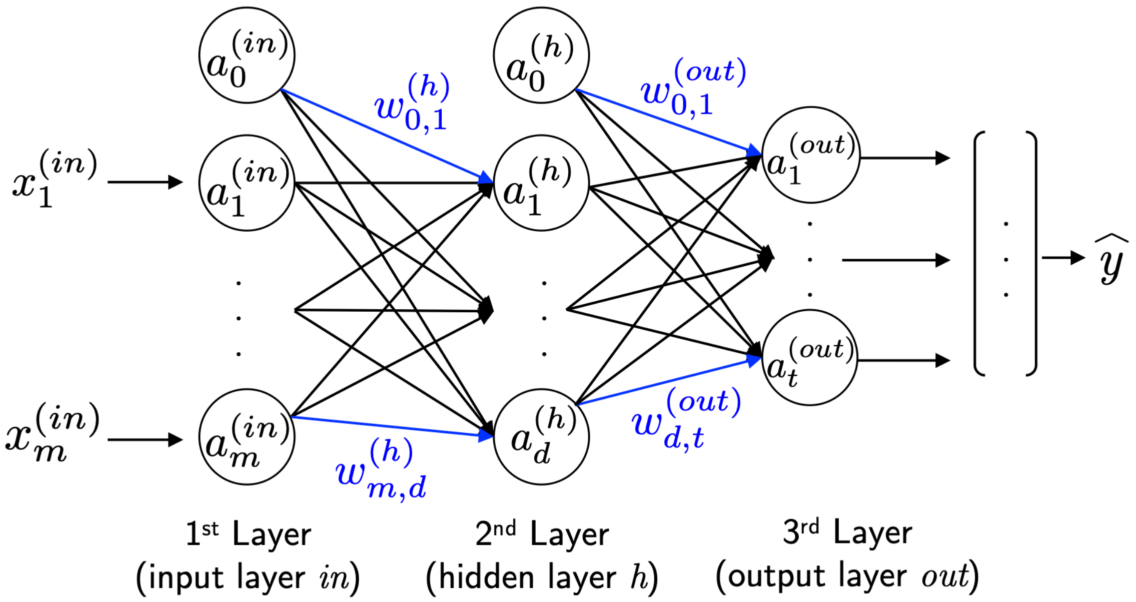
- Hyperparamters: learning rate $\boldsymbol{\eta}$ and the number of iteration or **epochs.**
- Stochastic gradient descent (SGD)

Online learning

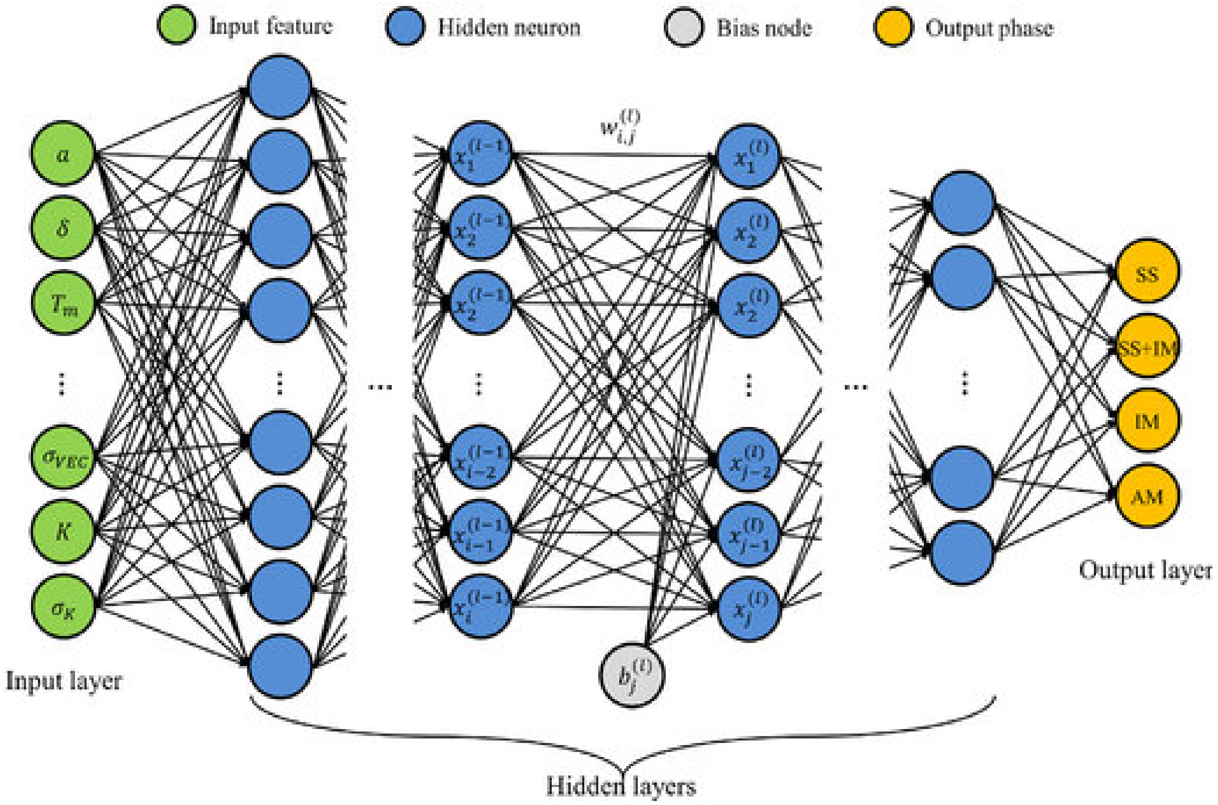
Mini-batch learning

Multilayer feedforward neural network architecture

Multilayer NN

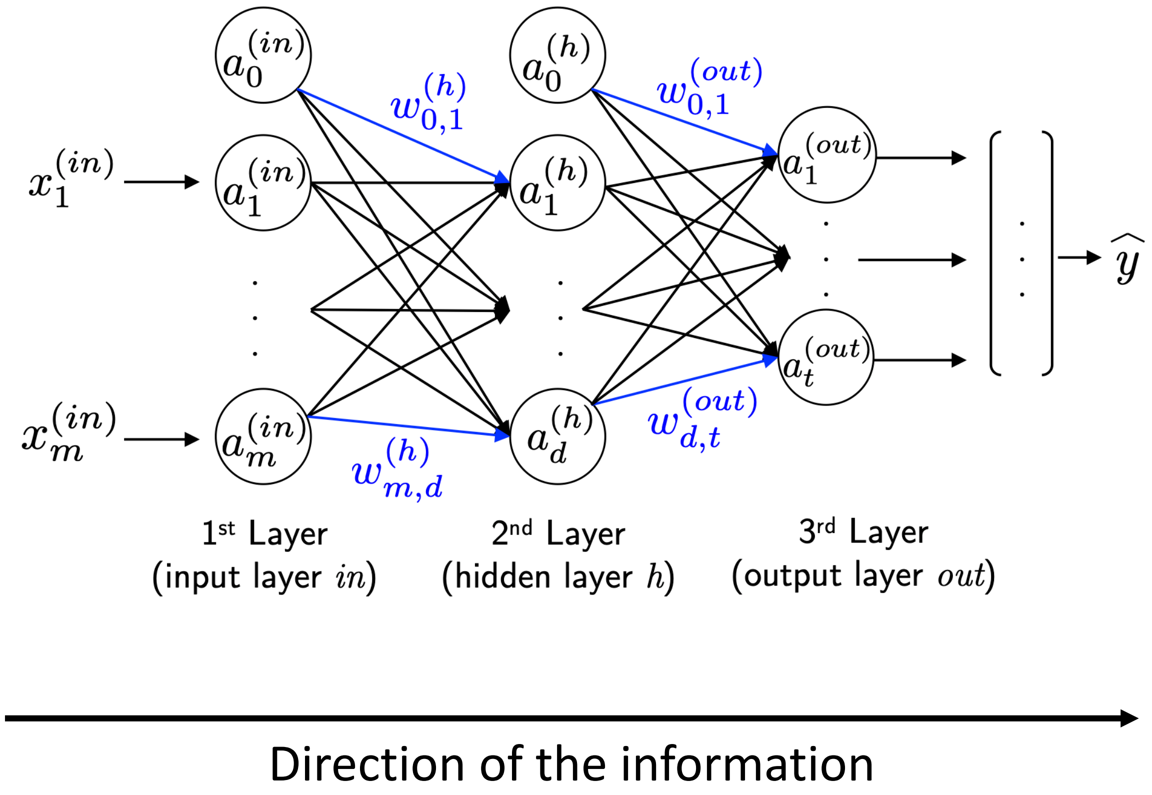


Deep NN



Deep NN makes use of deep learning for training

Multilayer feedforward neural network architecture



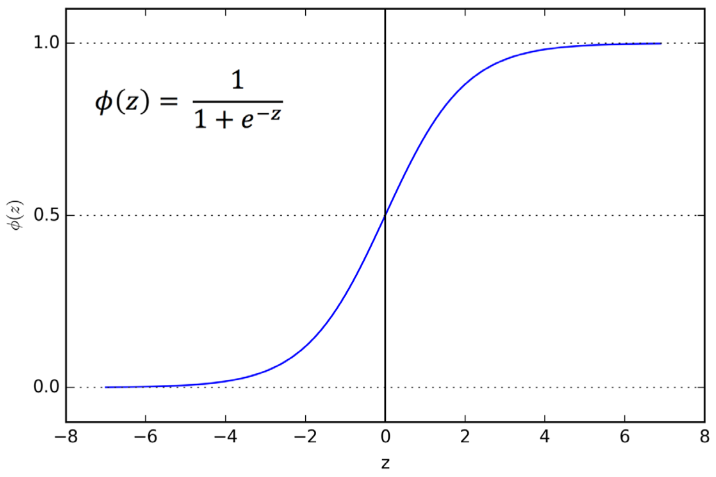
$$\boldsymbol{a}^{(in)} = \begin{bmatrix} a_0^{(in)} \\ a_1^{(in)} \\ \vdots \\ a_m^{(in)} \end{bmatrix} = \begin{bmatrix} 1 \\ x_1^{(in)} \\ \vdots \\ x_m^{(in)} \end{bmatrix}$$

in input layer
h hidden layer
out output layer

Weight matrix $\boldsymbol{W}^{(h)} \in \mathbb{R}^{m \times d}$
d number of hidden units
m number of input units

Depth = number of layers of a NN (excluding the input layer by convention)
Width = number of neurons (units) in a given layer

Computing the logistic cost function



$$J(\boldsymbol{w}) = -\sum_{i=1}^{n} y^{[i]} \log\left(\boxed{a^{[i]}}\right) + \left(1 - y^{[i]}\right) \log\left(1 - \boxed{a^{[i]}}\right)$$

Sigmoid activation of the *i*th sample in the dataset

$$a^{[i]} = \phi\left(z^{[i]}\right)$$

NOTE *[i]* index for training example

L2 regulation term

$$L2 = \lambda \|\boldsymbol{w}\|_2^2 = \lambda \sum_{j=1}^{m} w_j^2$$

Final logistic cost function

$$J(\boldsymbol{w}) = -\left[\sum_{i=1}^{n} y^{[i]} \log\left(a^{[i]}\right) + \left(1 - y^{[i]}\right) \log\left(1 - a^{[i]}\right)\right] + \frac{\lambda}{2} \|\boldsymbol{w}\|_2^2$$

$$a^{(out)} = \begin{bmatrix} 0.1 \\ 0.9 \\ \vdots \\ 0.3 \end{bmatrix}, \qquad y = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

Computing the logistic cost function

$$\boxed{J(\boldsymbol{W})} = -\left[\sum_{i=1}^{n}\sum_{j=1}^{t} y_j^{[i]} \log\left(a_j^{[i]}\right) + \left(1 - y_j^{[i]}\right)\log\left(1 - a_j^{[i]}\right)\right] + \boxed{\frac{\lambda}{2}\sum_{l=1}^{L-1}\sum_{i=1}^{u_l}\sum_{j=1}^{u_{l+1}}\left(w_{j,i}^{(l)}\right)^2}$$

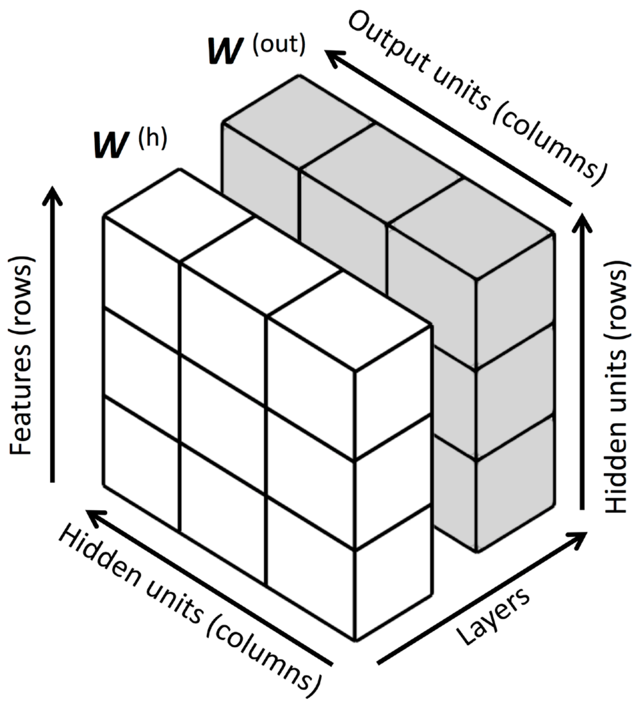
Penalty term

Final Aim: To minimize $J(\boldsymbol{W})$

$$\boxed{\frac{\partial}{\partial w_{j,i}^{(l)}} J(\boldsymbol{W})}$$

Compute by backpropagation

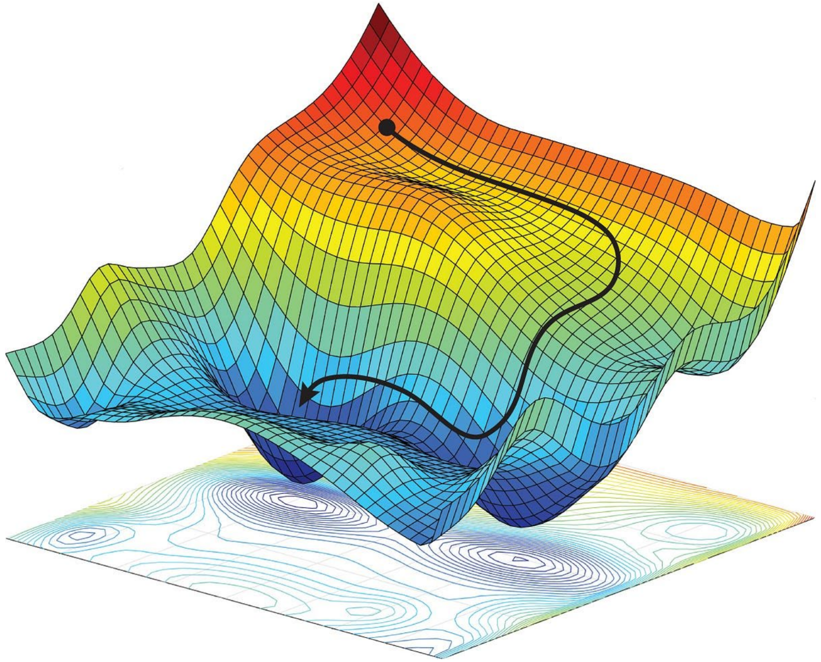
$\boldsymbol{W}^{(h)}$ = weight matrix connecting the input to the hidden layer
$\boldsymbol{W}^{(out)}$ = weight matrix connecting the hidden layer to the output layer



Backpropagation

Efficient way to compute partial derivatives

AIM: To use those derivative to learn weight coefficients in a ML NN



The chain rule

$$\frac{d}{dx}[\underline{f(g(x))}] = \frac{df}{dg} \cdot \frac{dg}{dx}$$

Nested function

Backpropagation

$$F(x) = f(g(h(u(v(x)))$$

The chain rule

$$\frac{dF}{dx} = \frac{d}{dx}F(x) = \frac{d}{dx}f(g(h(u(v(x))))) = \frac{df}{dg}\cdot\frac{dg}{dh}\cdot\frac{dh}{du}\cdot\frac{du}{dv}\cdot\frac{dv}{dx}$$

Automatic differentiation

Forward mode

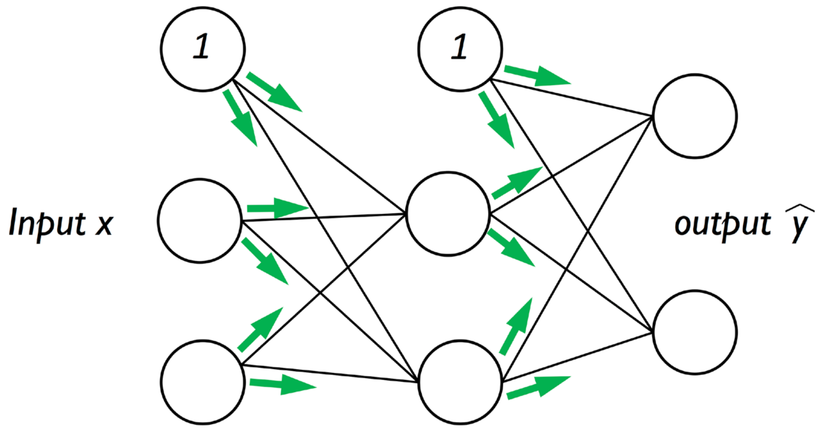
Computationally expensive

Backward mode

We start from right to left: we multiply a matrix by a vector, which yields another vector that is multiplied by the next matrix and so on

Training neural networks via backpropagation

1. Forward propagation



$$
\begin{aligned}
\boldsymbol{Z}^{(h)} &= \boldsymbol{A}^{(in)}\boldsymbol{W}^{(h)} && \text{(net input of the hidden layer)}\\
\boldsymbol{A}^{(h)} &= \phi\left(\boldsymbol{Z}^{(h)}\right) && \text{(activation of the hidden layer)}\\
\boldsymbol{Z}^{(out)} &= \boldsymbol{A}^{(h)}\boldsymbol{W}^{(out)} && \text{(net input of the output layer)}\\
\boldsymbol{A}^{(out)} &= \phi\left(\boldsymbol{Z}^{(out)}\right) && \text{(activation of the output layer)}
\end{aligned}
$$

2. Backpropagation

Error vector → Error term of the hidden layer → Hidden layer error matrix

$$\boldsymbol{\delta}^{(out)} = \boldsymbol{a}^{(out)} - \boxed{\boldsymbol{y}}$$

True class labels

$$\boldsymbol{\delta}^{(h)} = \boldsymbol{\delta}^{(out)}\left(\boldsymbol{W}^{(out)}\right)^T \odot \boxed{\frac{\partial\phi\left(z^{(h)}\right)}{\partial z^{(h)}}}$$

Derivative of the sigmoid activation function

$$\delta^{(h)} = \delta^{(out)}\left(\boldsymbol{W}^{(out)}\right)^T \odot \left(a^{(h)} \odot \left(1 - a^{(h)}\right)\right)$$

$$\rightarrow \frac{\partial\phi(z)}{\partial z} = \left(a^{(h)} \boxed{\odot} \left(1 - a^{(h)}\right)\right)$$

Element-wise multiplication

Training neural networks via backpropagation

$\boldsymbol{W}^{(out)} = h \times t$ dimensionality matrix
t = number of output class
h=number of hidden units

$\delta^{(out)} = n \times t$ Error vector

$$\delta^{(h)} = \delta^{(out)} \left(\boldsymbol{W}^{(out)}\right)^{T} \odot \left(a^{(h)} \odot \left(1 - a^{(h)}\right)\right)$$

$n \times h$ matrix

$n \times h$ matrix

$n \times h$ Derivative of the sigmoid activation function

Training neural networks via backpropagation

Cost function → Vector form → Regulatory terms → Update weights

$$\frac{\partial}{\partial w_{i,j}^{(out)}} J(\boldsymbol{W}) = a_j^{(h)} \delta_i^{(out)}$$

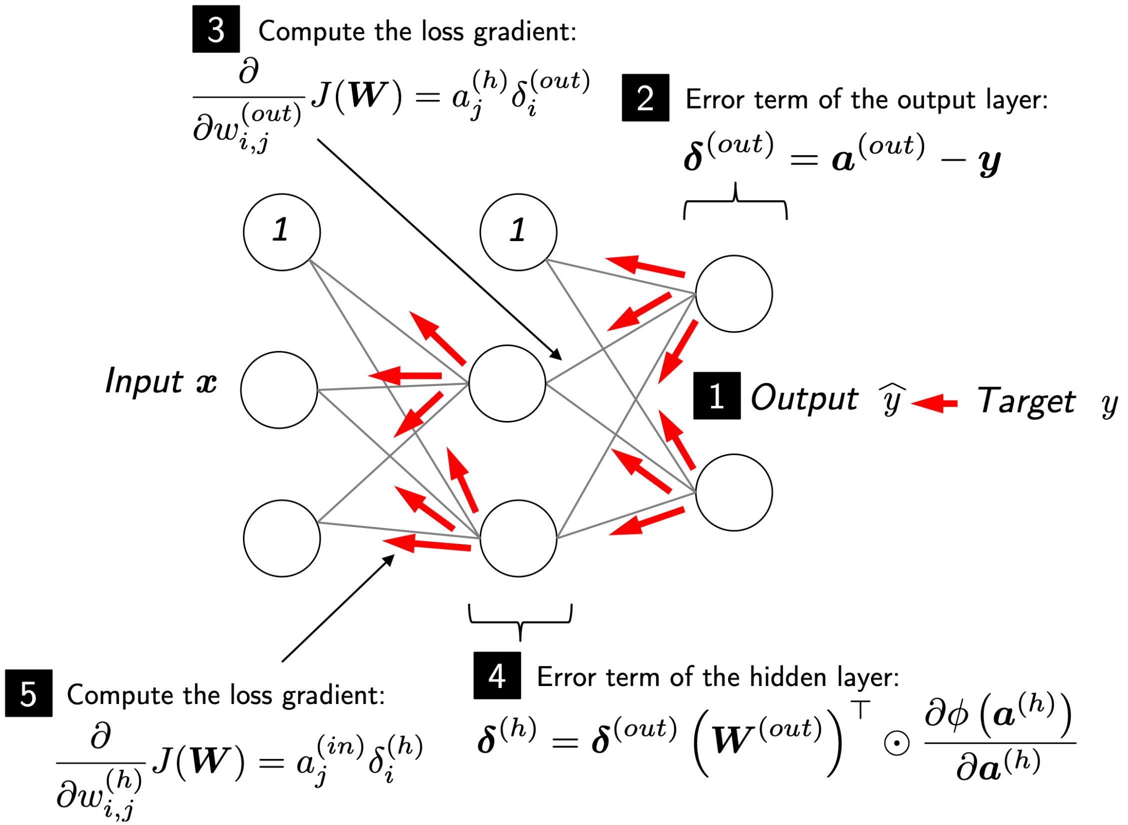
$$\frac{\partial}{\partial w_{i,j}^{(h)}} J(\boldsymbol{W}) = a_j^{(in)} \delta_i^{(h)}$$

$$\Delta^{(h)} = \left(\boldsymbol{A}^{(in)}\right)^T \delta^{(h)}$$

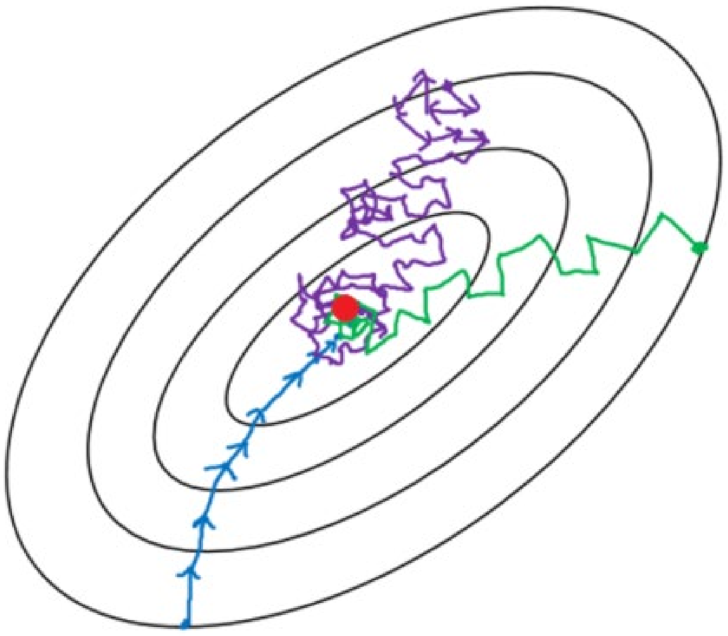
$$\Delta^{(out)} = \left(\boldsymbol{A}^{(h)}\right)^T \delta^{(out)}$$

$$\Delta^{(l)} := \Delta^{(l)} + \lambda^{(l)} \boldsymbol{W}^{(l)}$$

$$\boldsymbol{W}^{(l)} := \boldsymbol{W}^{(l)} - \eta \Delta^{(l)}$$



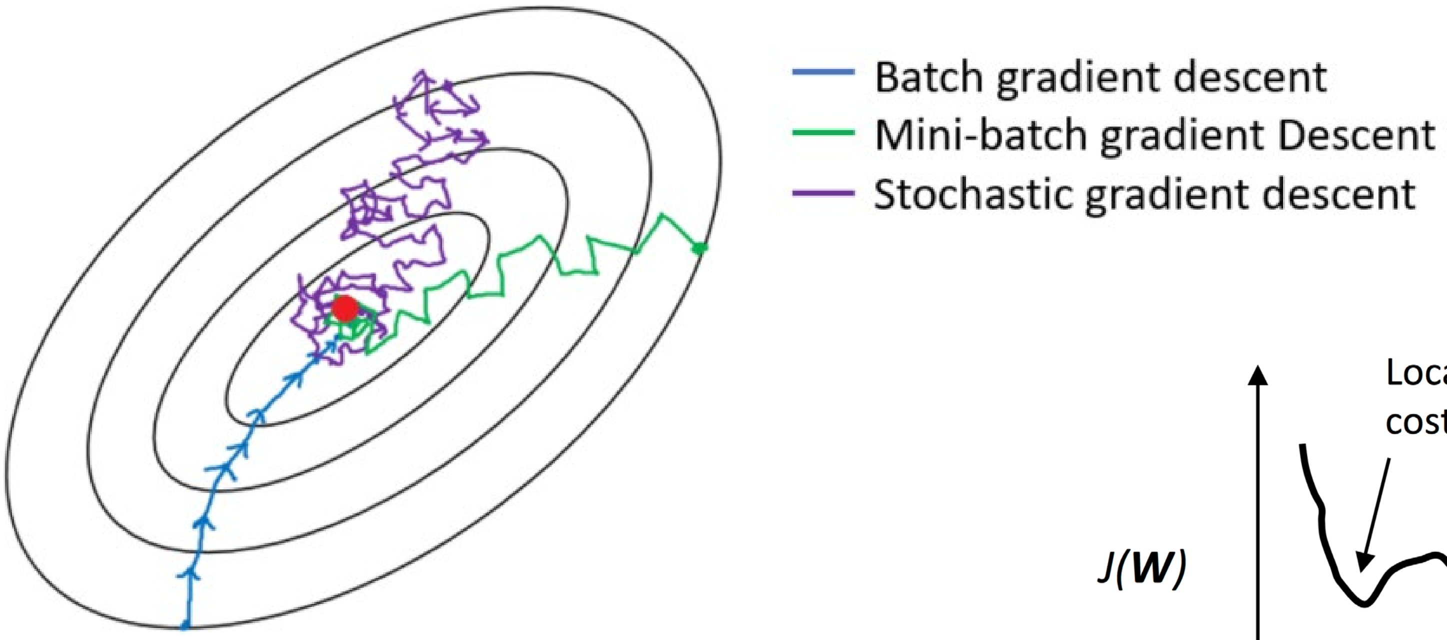
The convergence in neural networks

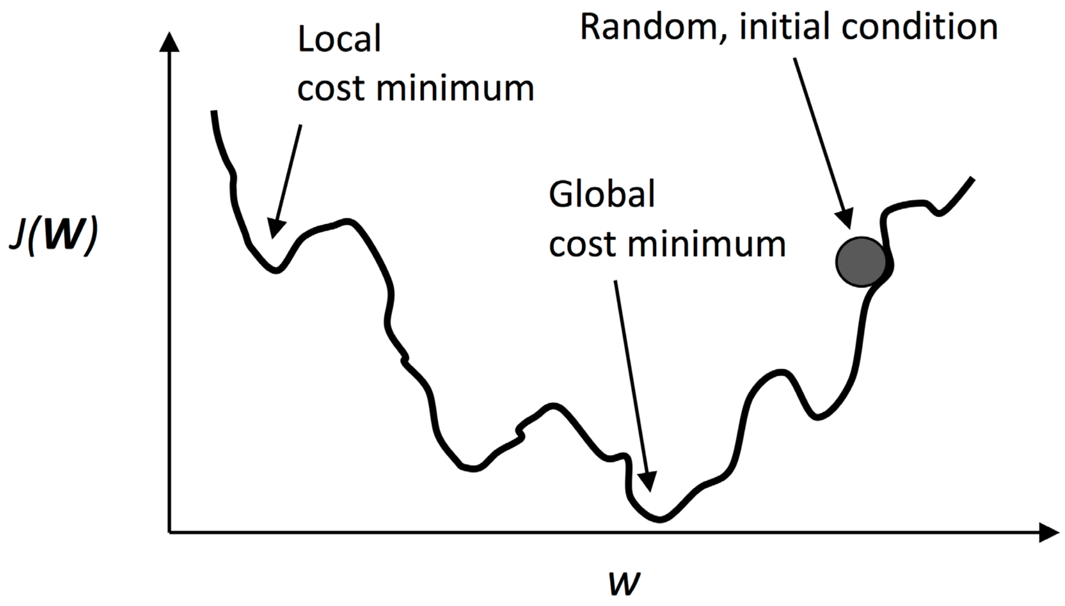


Advantages of mini-batch:
1. Easily fits in the memory
2. It is computationally efficient
3. If stuck in local minimums, some noisy steps can lead the way out of them
4. Average of the training samples produces stable error gradients and convergence

Cons:
- Mini-batch requires the configuration of an additional “mini-batch size” hyperparameter for the learning algorithm.

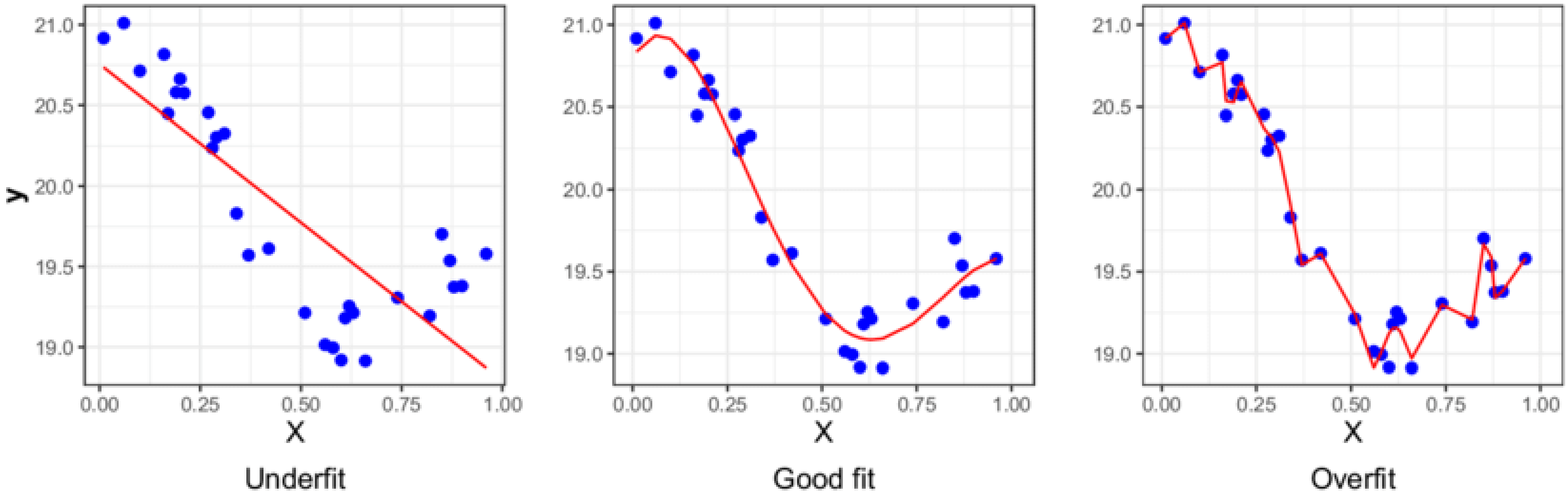
The convergence in neural networks





Learning rate could help

Overfitting



A large number of different methods have been developed.

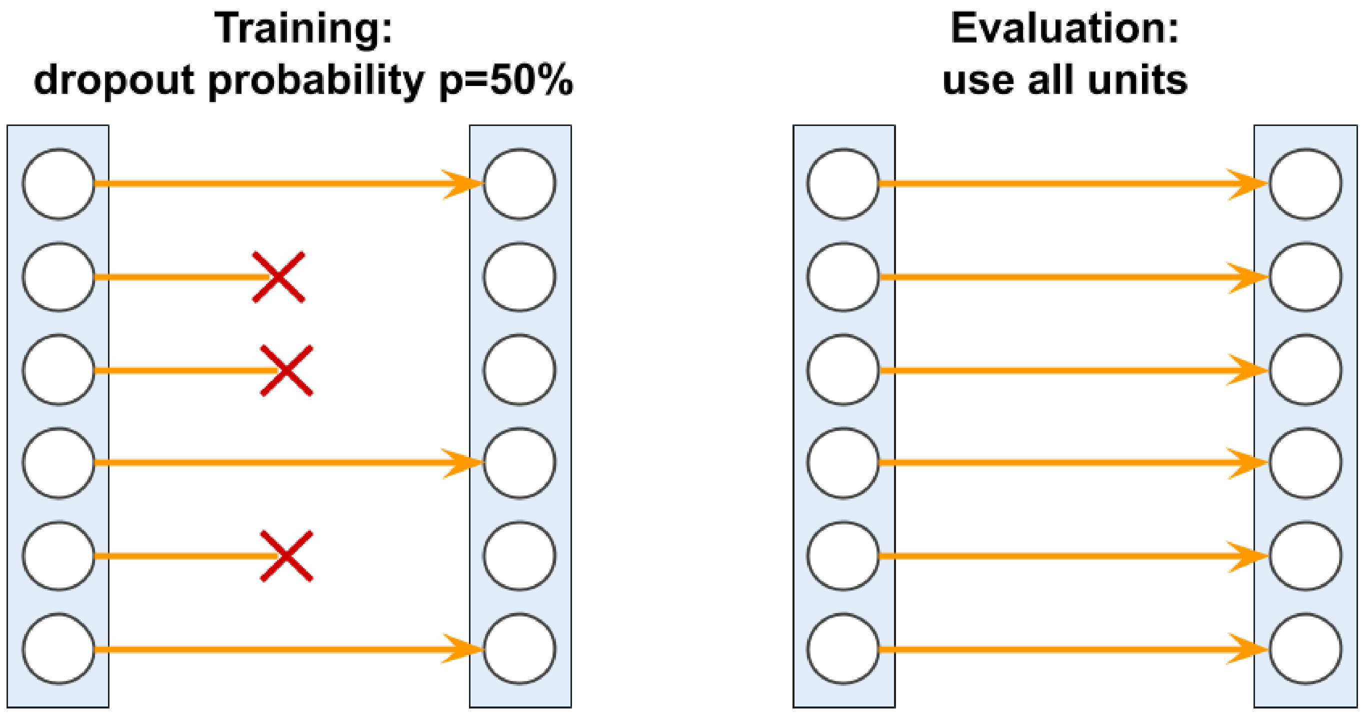
– Weight-decay
– Weight-sharing
– Early stopping
– Model averaging
– Bayesian fitting of neural nets
– Dropout
– Generative pre-training

Others common problems

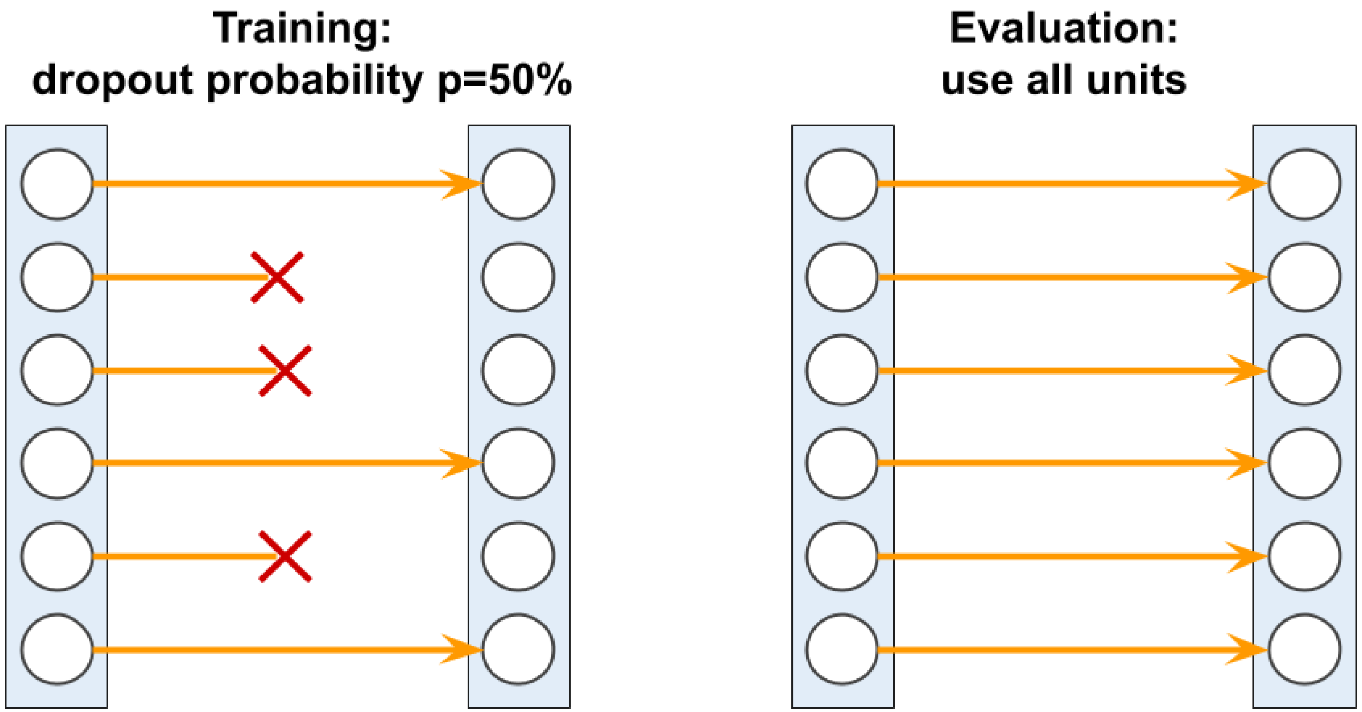
Start with a big learning rate, the weights of each hidden unit will all become very big and positive or very big and negative. → Plateau mistaken for a local minimum

Classification NNs which use squared error or cross-entropy, the best guessing strategy is to make each output unit always produce an output equal to the proportion of time it should be a 1. → Plateau mistaken for a local minimum

Regularizing an NN with dropout:



Regularizing an NN with dropout:



Advantage: random dropout forces the network to learn redundant representation of the data.

During prediction, all neurons will contribute to computing the pre-activations of the next layer.

Loss functions for classification:

Loss function	Usage	Examples: Using probabilities *from_logits=False*	Examples: Using logits *from_logits=True*
`BinaryCrossentropy`	Binary classification	y_true: 1 y_pred: 0.69	y_true: 1 y_pred: 0.8
`CategoricalCrossentropy`	Multiclass classification	y_true: 0 0 1 y_pred: 0.30 0.15 0.55	y_true: 0 0 1 y_pred: 1.5 0.8 2.1
`Sparse CategoricalCrossentropy`	Multiclass classification	y_true: 2 y_pred: 0.30 0.15 0.55	y_true: 2 y_pred: 1.5 0.8 2.1

Three loss functions available in Keras for dealing with all three cases: binary classification, multiclass and multiclass with integer (sparse) labels.

TensorFlow

What is TensorFlow?

TensorFlow is an open source machine learning framework for carrying out high-performance numerical computations.

It provides excellent architecture support which allows easy deployment of computations across a variety of platforms ranging from desktops to clusters of servers, mobiles, and edge devices.

Works better with GPUs

What is TensorFlow?

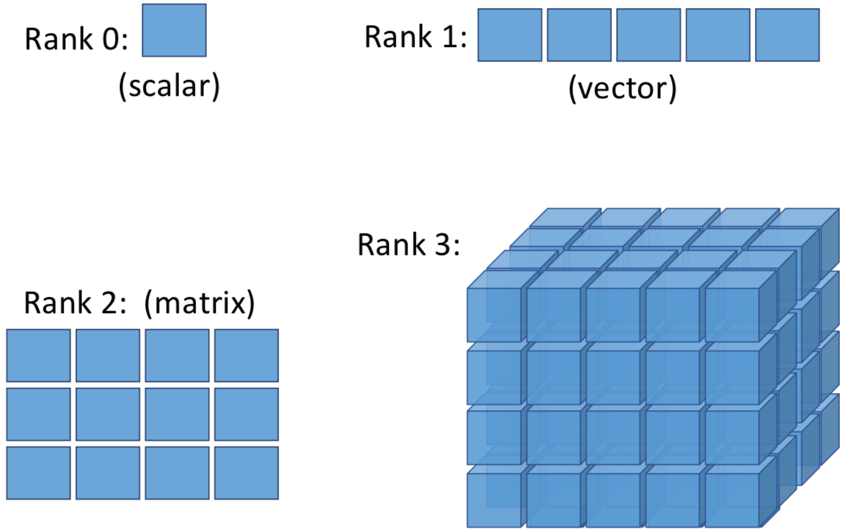
TensorFlow is an open source machine learning framework for carrying out high-performance numerical computations.

It provides excellent architecture support which allows easy deployment of computations across a variety of platforms ranging from desktops to clusters of servers, mobiles, and edge devices.

TensorFlow

Google Brain

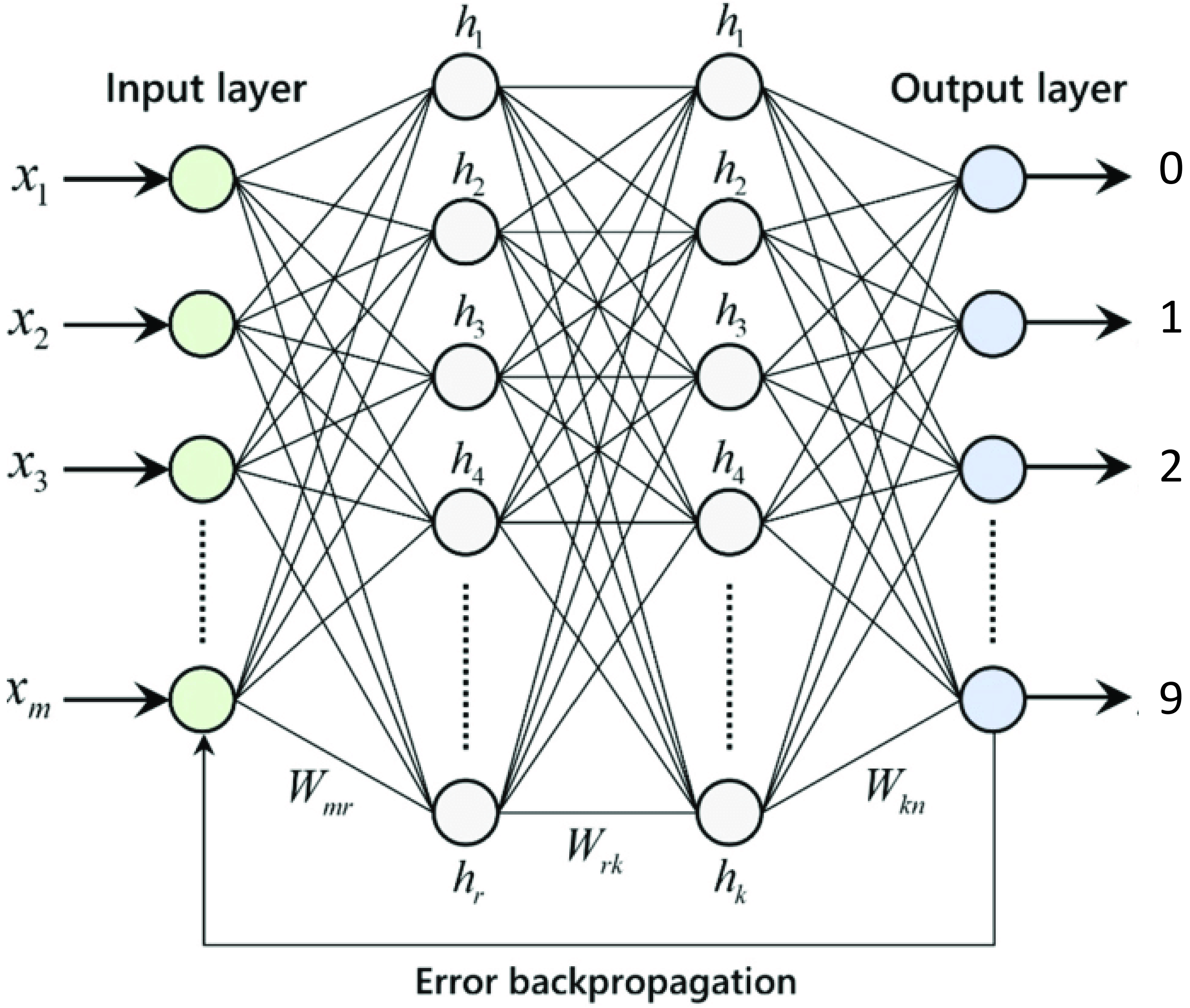
Works better with GPUs



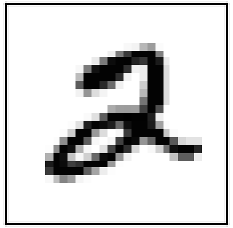
How will we learn TensorFlow?

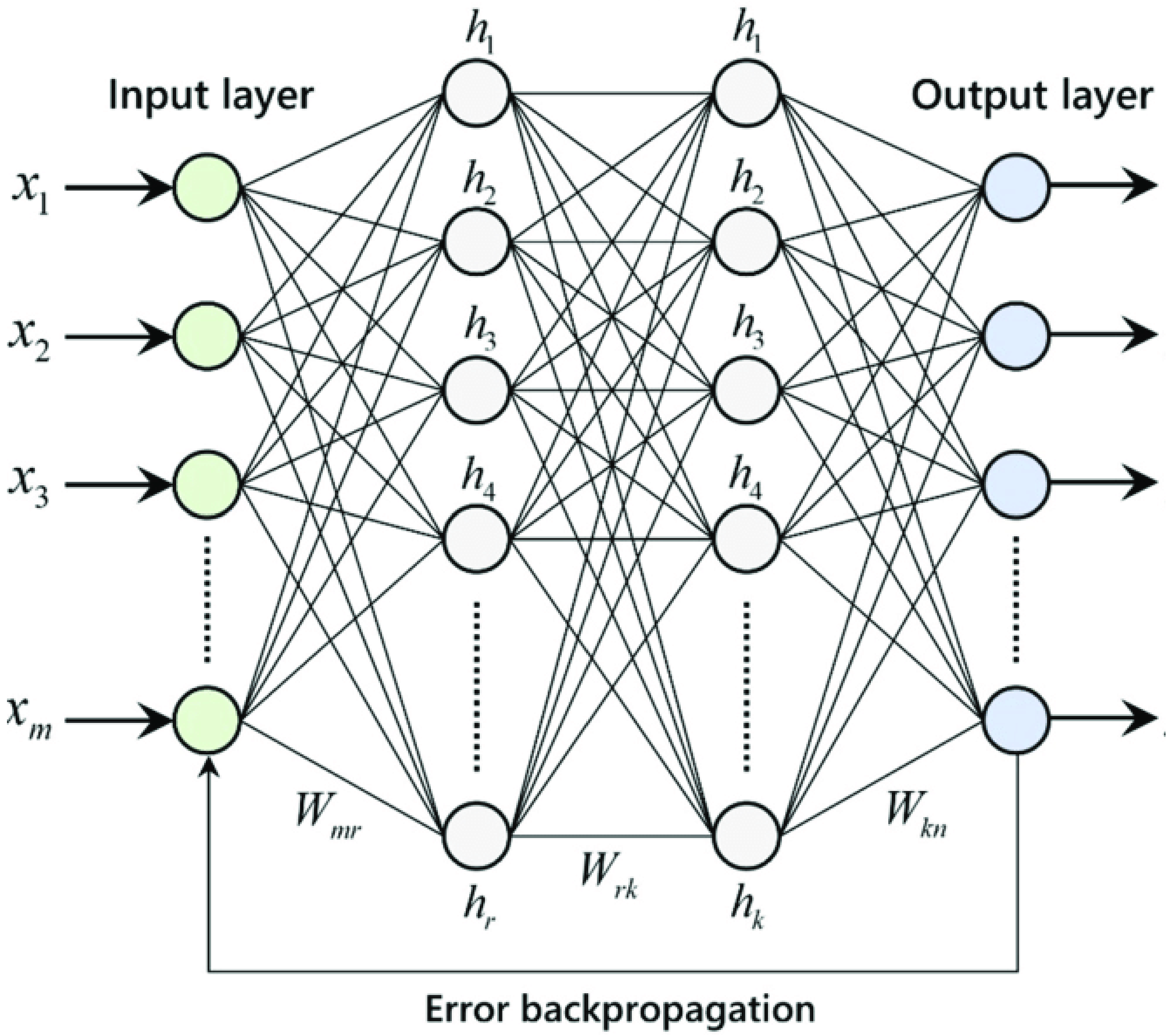
Neural Network For Handwritten Digits Classification

Classifying handwritten digits



Classifying handwritten digits



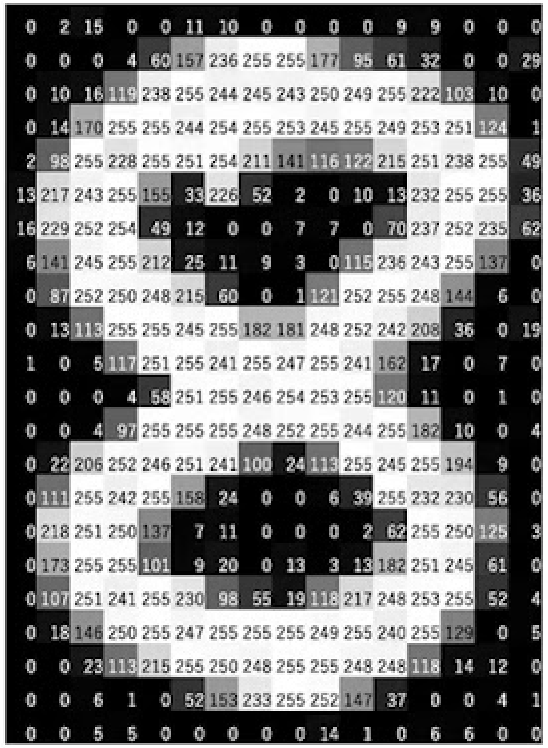


Output of a sigmoid function

0	0,01
1	0,10
2	0,82
9	0,32

0 – Black, 255 - white

Original figure



0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

2D array
25 x 16

Classifying handwritten digits