Tutorial 8

Neural Networks

Dan Qiao
danqiao@link.cuhk.edu.cn
OH: 16:00~17:00, Friday, Seat 81, SDS Lab in ZX Building
Nov. 8, 2022

Outlines

Recap (10 min)

- Multi Layer Perceptron
- Computation Graph
- Backpropagation & Chain rules

Demo (40 min)

- Data Preprocessing
- Example 1: Iris Classification
- Example 2: LDAPS Regression
- Extension: MNIST Classification by Pytorch

MLP

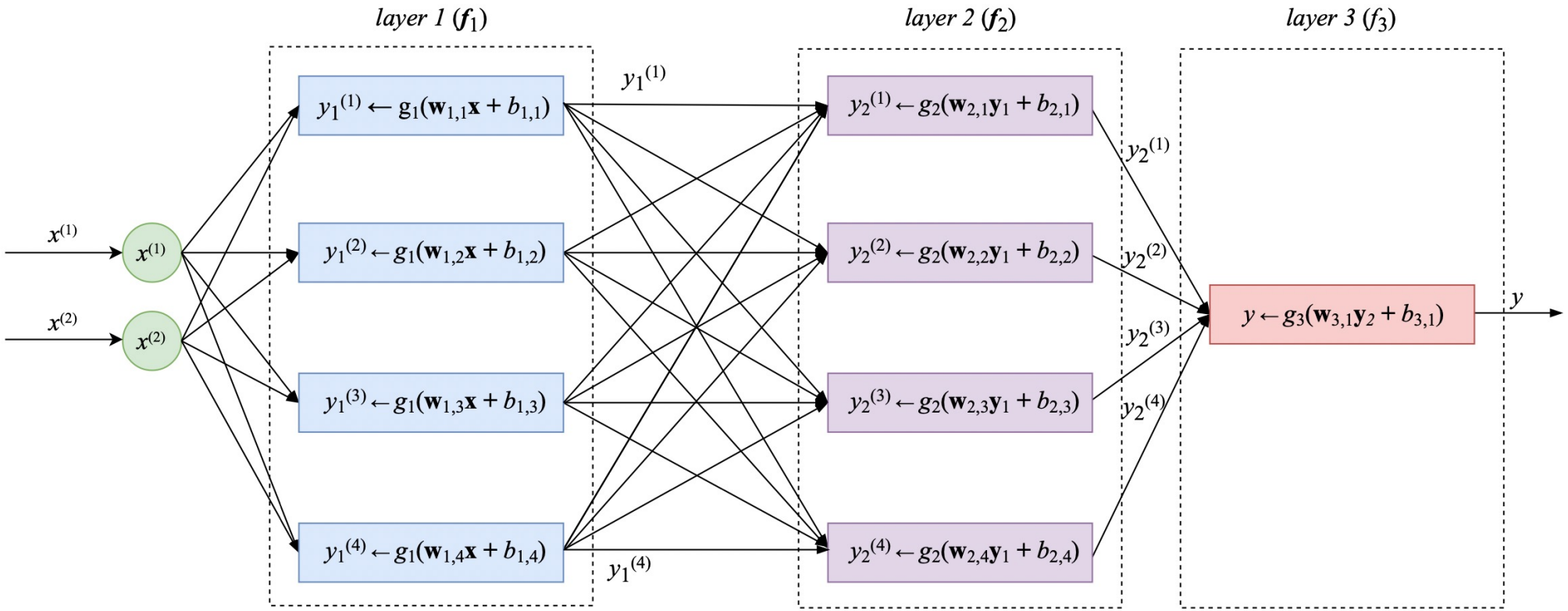


Figure 1: A multilayer perceptron with two-dimensional input, two layers with four units and one output layer with one unit.

MLP

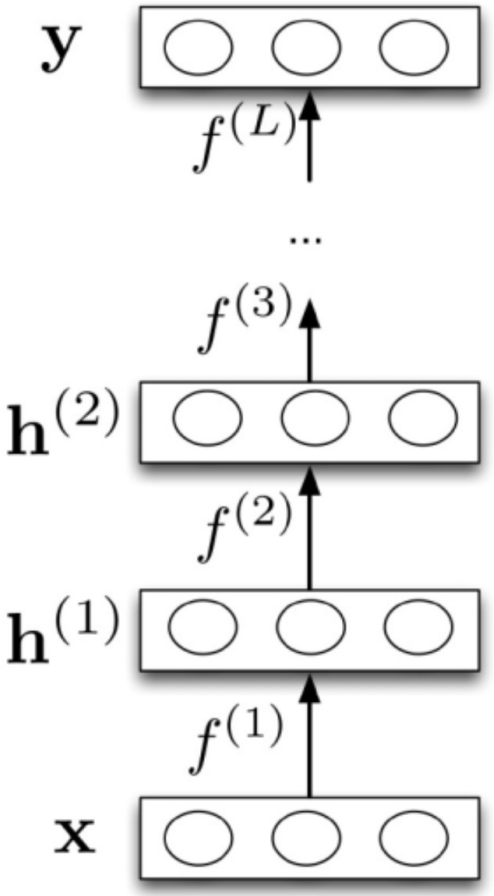
- Each layer computes a function, so the network computes a composition of functions:

$$\mathbf{h}^{(1)} = f^{(1)}(\mathbf{x})$$
$$\mathbf{h}^{(2)} = f^{(2)}(\mathbf{h}^{(1)})$$
$$\vdots$$
$$\mathbf{y} = f^{(L)}(\mathbf{h}^{(L-1)})$$

- Or more simply:

$$\mathbf{y} = f^{(L)} \circ \ldots \circ f^{(1)}(\mathbf{x})$$

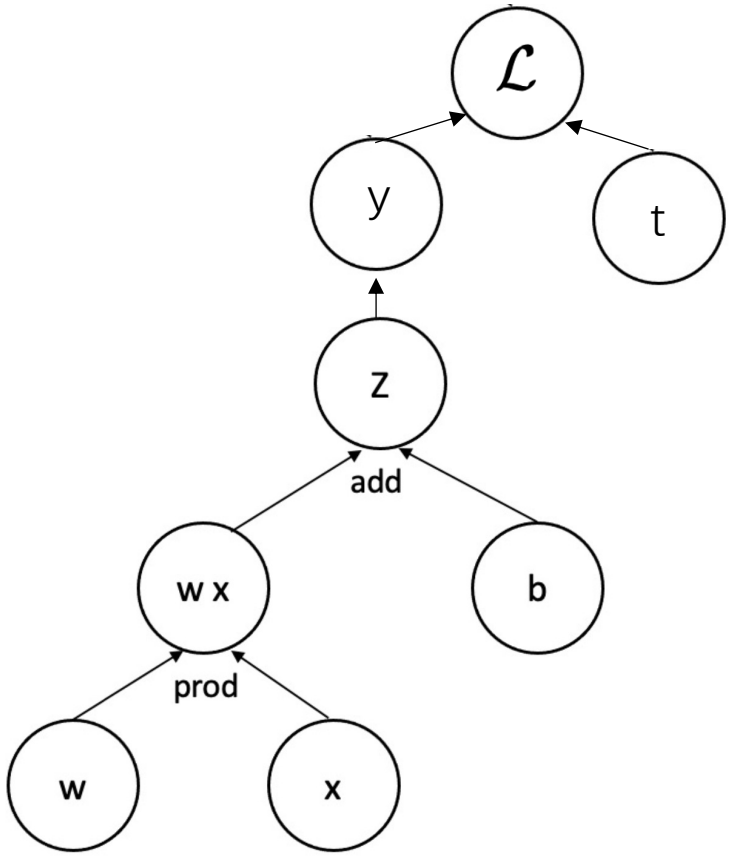
- Neural nets provide modularity: we can implement each layer's computations as a black box.



Computation Graph

Consider a loss function

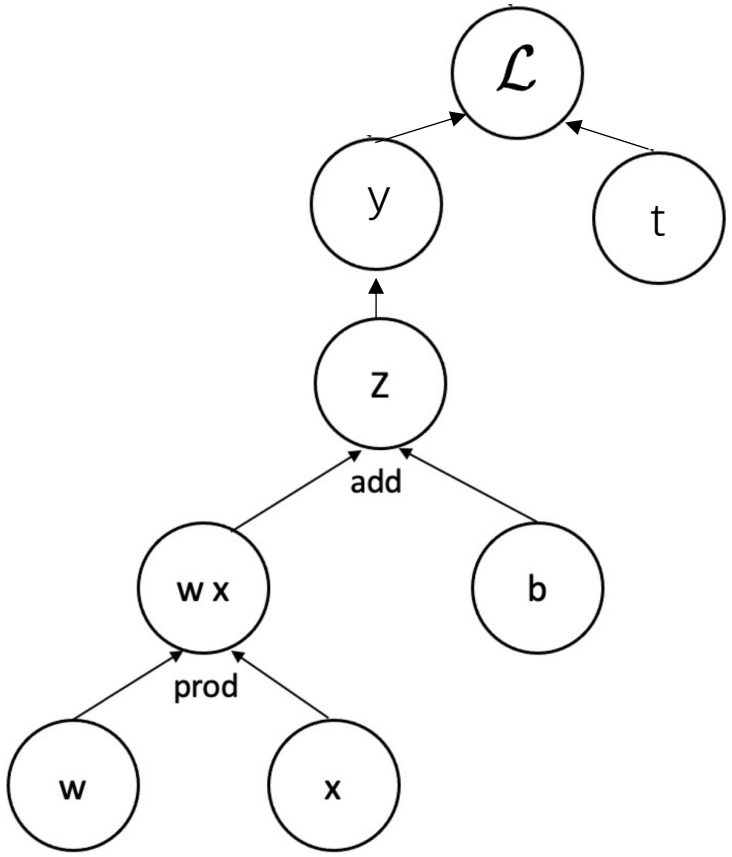
$$\mathcal{L} = \frac{1}{2}\big(\sigma(wx + b) - t\big)^2.$$



Computation Graph

Consider a loss function

$$\mathcal{L} = \frac{1}{2}\big(\sigma(wx + b) - t\big)^2.$$



Computing the loss:

$$z = wx + b$$

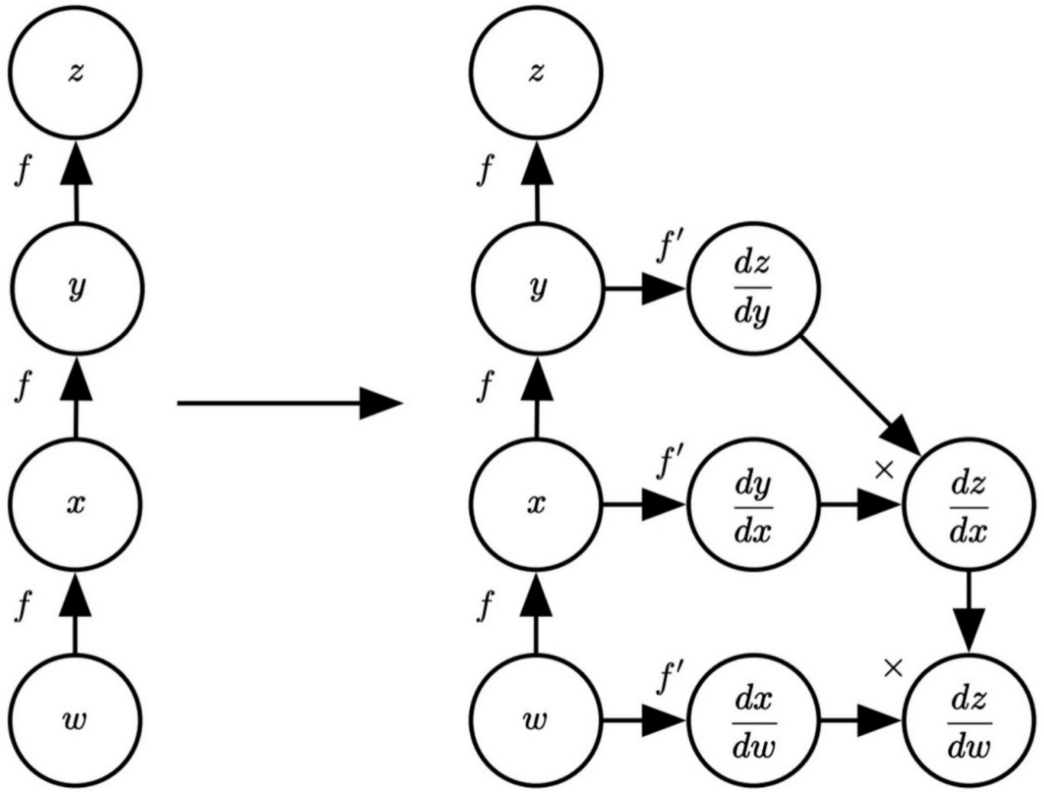
$$y = \sigma(z)$$

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

Chain Rule

Chain rule: if $f(h)$ and $h(x)$ are univariate functions, then

$$\frac{\mathrm{d}f(h(x))}{\mathrm{d}x} = \frac{\mathrm{d}f}{\mathrm{d}h} \cdot \frac{\mathrm{d}h}{\mathrm{d}x}.$$



Back Propagation

Simple Example: The goal is to compute $J = \cos(\sin(x^2) + 3x^2)$ on the forward pass and the derivative $\frac{dJ}{dx}$ on the backward pass.

Forward

$J = cos(u)$

$u = u_1 + u_2$

$u_1 = sin(t)$

$u_2 = 3t$

$t = x^2$

Back Propagation

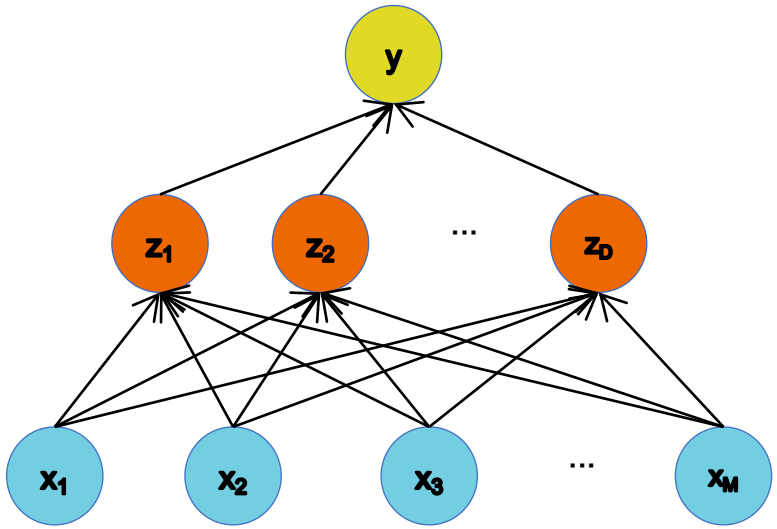
Simple Example: The goal is to compute $J = \cos(\sin(x^2) + 3x^2)$ on the forward pass and the derivative $\frac{dJ}{dx}$ on the backward pass.

Forward	Backward
$J = cos(u)$	$\frac{dJ}{du}$ += $-sin(u)$
$u = u_1 + u_2$	$\frac{dJ}{du_1}$ += $\frac{dJ}{du}\frac{du}{du_1}$, $\frac{du}{du_1} = 1$ $\quad$ $\frac{dJ}{du_2}$ += $\frac{dJ}{du}\frac{du}{du_2}$, $\frac{du}{du_2} = 1$
$u_1 = sin(t)$	$\frac{dJ}{dt}$ += $\frac{dJ}{du_1}\frac{du_1}{dt}$, $\frac{du_1}{dt} = \cos(t)$
$u_2 = 3t$	$\frac{dJ}{dt}$ += $\frac{dJ}{du_2}\frac{du_2}{dt}$, $\frac{du_2}{dt} = 3$
$t = x^2$	$\frac{dJ}{dx}$ += $\frac{dJ}{dt}\frac{dt}{dx}$, $\frac{dt}{dx} = 2x$

Back Propagation

Loss $J = y^* \log y + (1 - y^*) \log(1 - y)$

Neural Network



Forward

$$J = y^* \log y + (1 - y^*) \log(1 - y)$$

$$y = \frac{1}{1 + \exp(-b)}$$

$$b = \sum_{j=0}^{D} \beta_j z_j$$

$$z_j = \frac{1}{1 + \exp(-a_j)}$$

$$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$$

Backward

$$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$$

$$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$$

$$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \frac{db}{d\beta_j} = z_j$$

$$\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \frac{db}{dz_j} = \beta_j$$

$$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$$

$$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \frac{da_j}{d\alpha_{ji}} = x_i$$

$$\frac{dJ}{dx_i} = \frac{dJ}{da_j}\frac{da_j}{dx_i}, \frac{da_j}{dx_i} = \alpha_{ji}$$

Outlines

Recap (10 min)

- Multi Layer Perceptron
- Computation Graph
- Backpropagation & Chain rules

Demo (40 min)

- Data Preprocessing
- Example 1: Iris Classification
- Example 2: LDAPS Regression
- Extension: MNIST Classification by Pytorch

Data Preprocessing

Preprocessing Steps:

- Load Datasets
- Observe Data
- Data Clean
- Split Train-Test Set
- Choose Your ML Model

In [2]:
```
import math
import numpy as np
import pandas as pd


import sklearn

# import sklearn
sklearn.__version__
```

Out[2]: '1.1.1'

In [49]:
```
# import datasets in sklearn
import sklearn.datasets as sk_datasets
iris = sk_datasets.load_iris()
iris_X = iris.data #data
iris_y = iris.target #label

iris_X.shape, len(iris_y)
```

Out[49]: ((150, 4), 150)

In [29]:
```
# read LDAPS.csv with pandas
data_csv = pd.read_csv("Regression.csv")
data_csv.shape
```

Out[29]: (7752, 25)

Data Preprocessing

Preprocessing Steps:

- Load Datasets
- Observe Data
- Data Clean
- Split Train-Test Set
- Choose Your ML Model

```
In [44]: iris.items(), iris_X.shape, len(iris_y), type(iris_X), type(iris_y)

Out[44]: (dict_items([('data', array([[5.1, 3.5, 1.4, 0.2],
                [4.9, 3. , 1.4, 0.2],
                [4.7, 3.2, 1.3, 0.2],
                [4.6, 3.1, 1.5, 0.2],
                [5. , 3.6, 1.4, 0.2],
                [5.4, 3.9, 1.7, 0.4],
                [4.6, 3.4, 1.4, 0.3],
                [5. , 3.4, 1.5, 0.2],
                [4.4, 2.9, 1.4, 0.2],
                [4.9, 3.1, 1.5, 0.1],
                [5.4, 3.7, 1.5, 0.2],
                [4.8, 3.4, 1.6, 0.2],
                [4.8, 3. , 1.4, 0.1],
                [4.3, 3. , 1.1, 0.1],
                [5.8, 4. , 1.2, 0.2],
                [5.7, 4.4, 1.5, 0.4],
                [5.4, 3.9, 1.3, 0.4],
                [5.1, 3.5, 1.4, 0.3],
                [5.7, 3.8, 1.7, 0.3],
```

```
                [6.2, 3.4, 5.4, 2.3],
                [5.9, 3. , 5.1, 1.8]])), ('target', array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
,
                0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])), ('frame', None), ('target_names', array(['setosa',
'versicolor', 'virginica'], dtype='<U10')), ('DESCR', '.. _iris_dataset:\n\nIris plants dataset\n------------------
--\n\n**Data Set Characteristics:**\n\n    :Number of Instances: 150 (50 in each of three classes)\n    :Number of
Attributes: 4 numeric, predictive attributes and the class\n    :Attribute Information:\n        - sepal length in
cm\n        - sepal width in cm\n        - petal length in cm\n        - petal width in cm\n        - class:\n
- Iris-Setosa\n                - Iris-Versicolour\n                - Iris-Virginica\n                \n    :Summary
```

```
more ...'), ('feature_names', ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']),
('filename', 'iris.csv'), ('data_module', 'sklearn.datasets.data')]),
 (150, 4),
 150,
 numpy.ndarray,
 numpy.ndarray)
```

Data Preprocessing

Preprocessing Steps:

- Load Datasets
- Observe Data
- Data Clean
- Split Train-Test Set
- Choose Your ML Model

The Iris Dataset

```
f_label = iris.feature_names

iris_X_pd = pd.DataFrame(iris_X, columns = f_label)
iris_y_pd = pd.DataFrame(iris_y, columns = ['Class'])

iris_X_pd
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

```
In [6]: iris_y_pd
```

Out[6]:

	Class
0	0
1	0
2	0
3	0
4	0
...	...
145	2
146	2
147	2
148	2
149	2

150 rows × 1 columns

Data Preprocessing

Preprocessing Steps:

- Load Datasets
- Observe Data
- Data Clean
- Split Train-Test Set
- Choose Your ML Model

The Iris Dataset

```
iris_X_pd.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
iris_y_pd.head()
```

	Class
0	0
1	0
2	0
3	0
4	0

Data Preprocessing

Preprocessing Steps:

- Load Datasets
- Observe Data
- Data Clean
- Split Train-Test Set
- Choose Your ML Model

The Iris Dataset

```
iris_total = pd.DataFrame(iris.data, columns = iris.feature_names)
iris_total['iris_type'] = iris.target
iris_total.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	iris_type
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
iris_total['iris_type'] = iris.target_names[iris.target]
iris_total.head()
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	iris_type
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

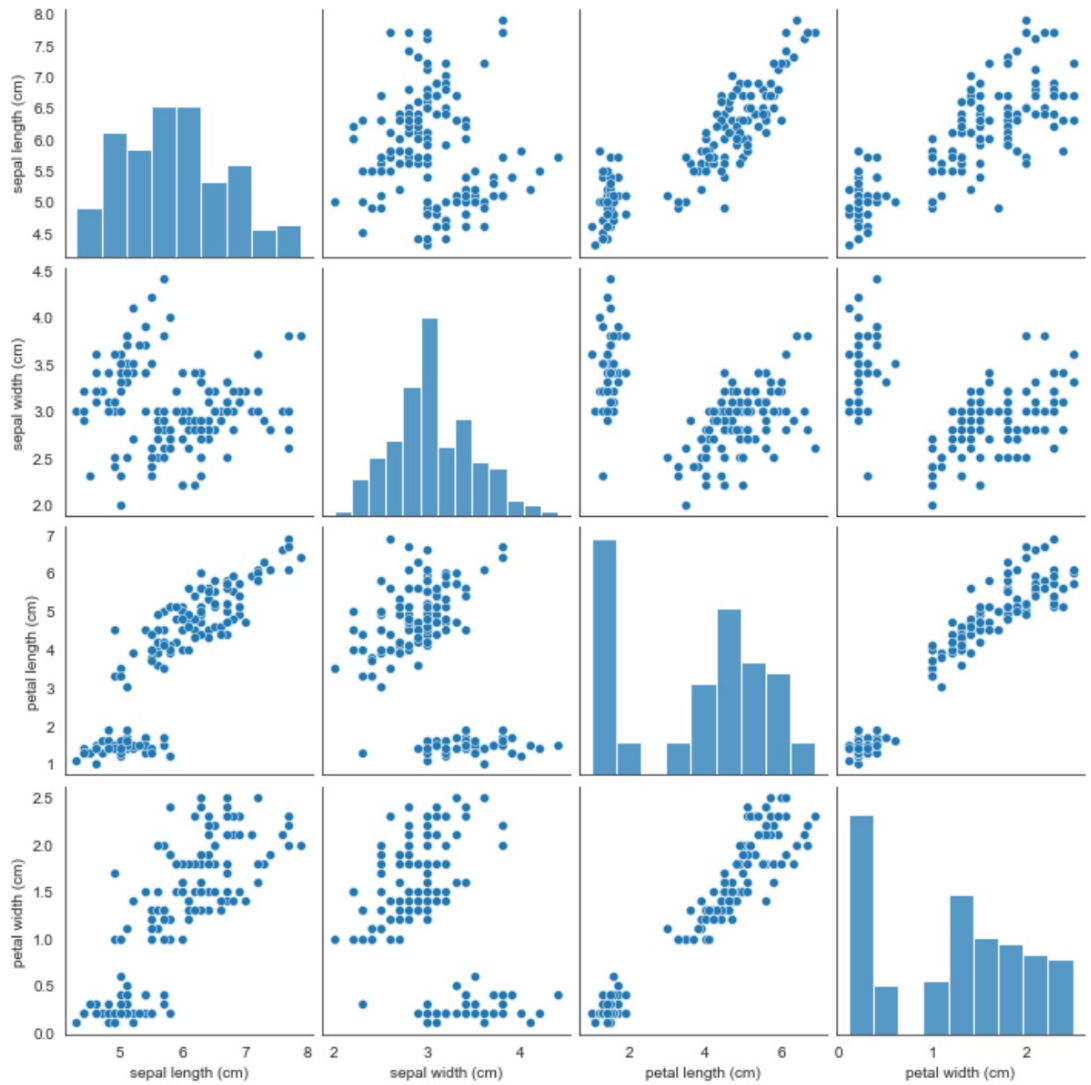
Data Preprocessing

Preprocessing Steps:

- Load Datasets
- Observe Data
- Data Clean
- Split Train-Test Set
- Choose Your ML Model

```
sns.set_style('white',{'font.sans-serif':['simhei', 'Arial']})
sns.pairplot(iris_total)
```

```
<seaborn.axisgrid.PairGrid at 0x7fd852437c70>
```

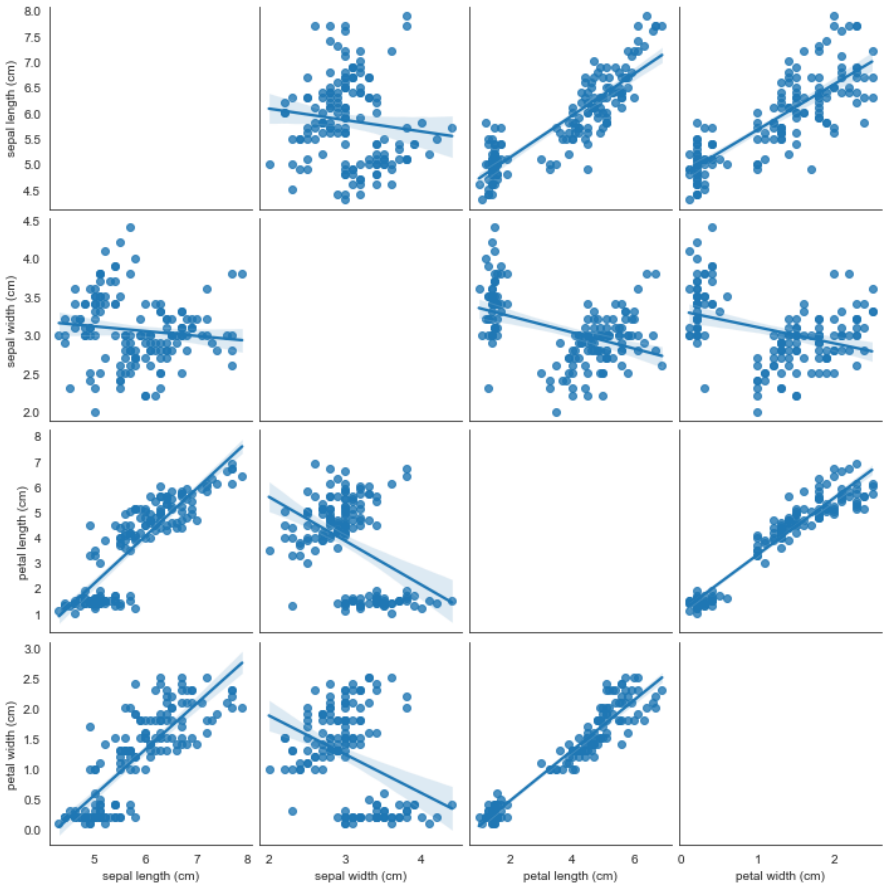


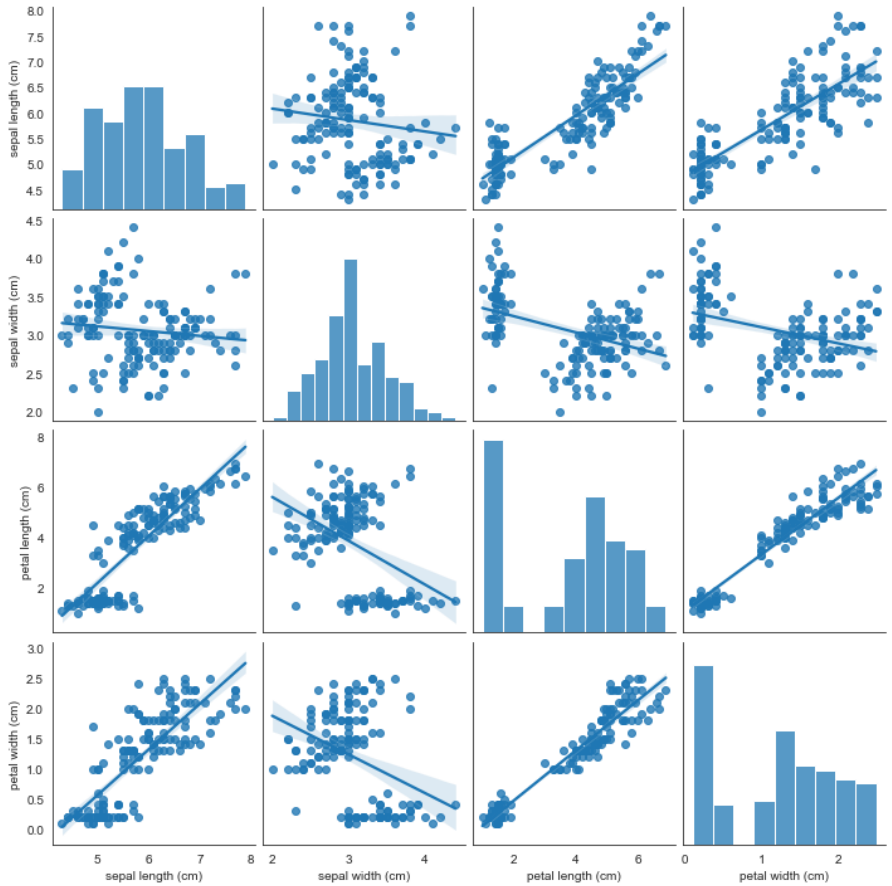
Data Preprocessing

Preprocessing Steps:

- Load Datasets
- Observe Data
- Data Clean
- Split Train-Test Set
- Choose Your ML Model

```
sns.pairplot(iris_total, kind='reg', diag_kind = 'ked')
sns.pairplot(iris_total, kind='reg', diag_kind = 'hist')
# kind:{'scatter', 'reg'}, diag_kind:{'hist', 'kde'}
```





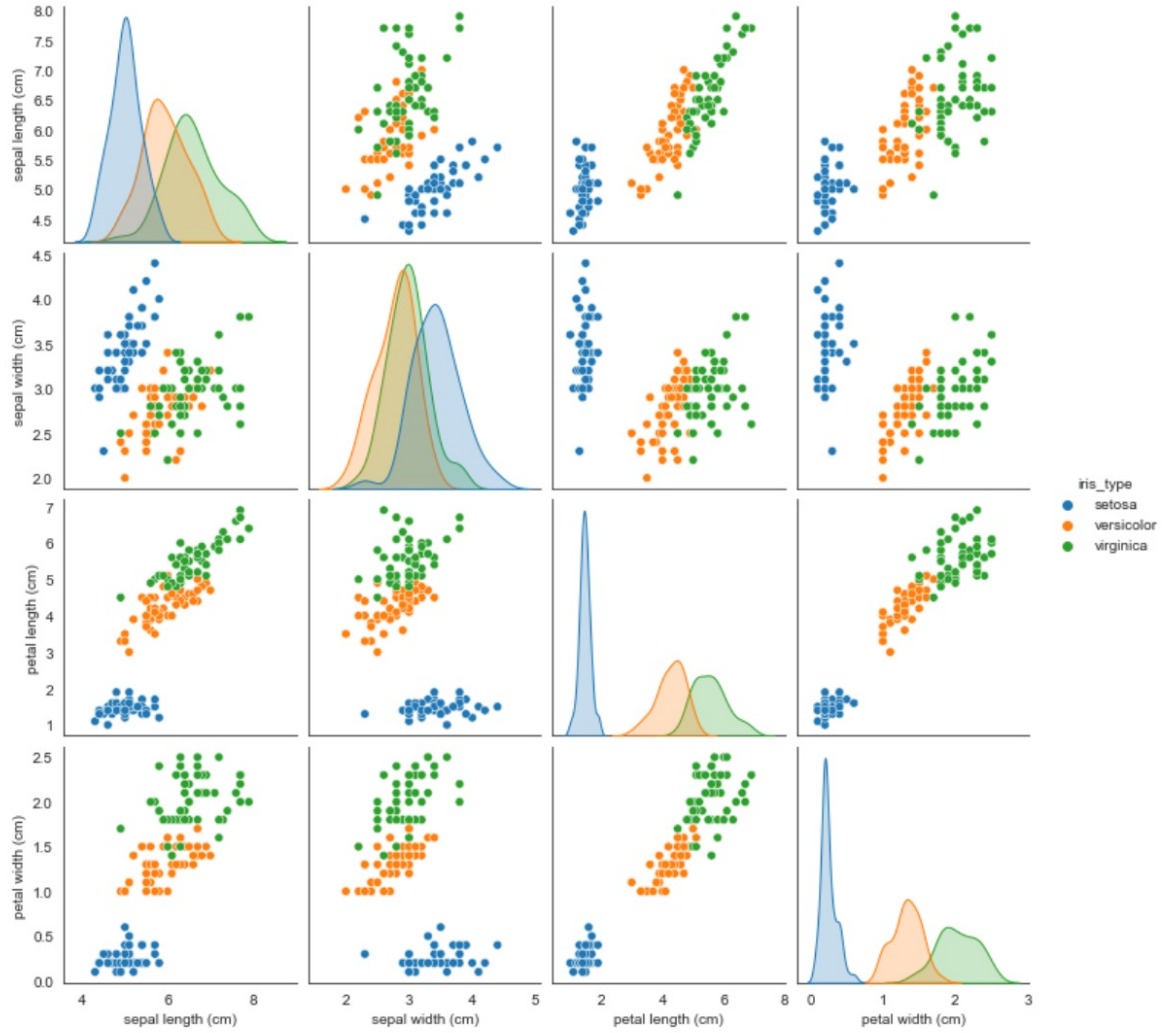
Data Preprocessing

Preprocessing Steps:

- Load Datasets
- Observe Data
- Data Clean
- Split Train-Test Set
- Choose Your ML Model

```
sns.pairplot(iris_total, hue='iris_type')
```

```
<seaborn.axisgrid.PairGrid at 0x7fd8724d7fd0>
```



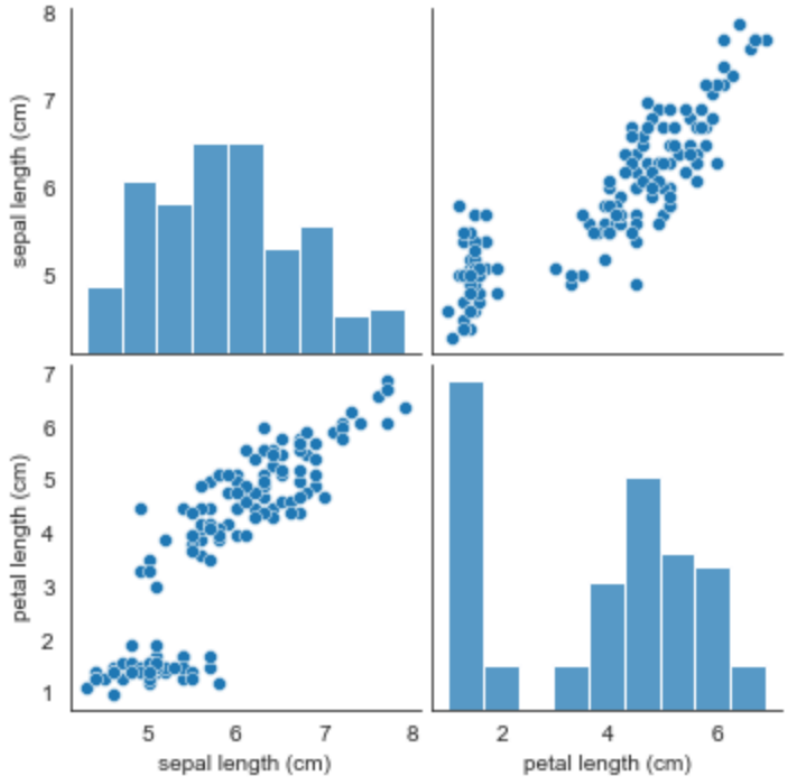
Data Preprocessing

Preprocessing Steps:

- Load Datasets
- Observe Data
- Data Clean
- Split Train-Test Set
- Choose Your ML Model

```
sns.pairplot(iris_total, vars=['sepal length (cm)','petal length (cm)'])
# plot the variance of two columns
```

```
<seaborn.axisgrid.PairGrid at 0x7fd829559550>
```



Data Preprocessing

Preprocessing Steps:

- Load Datasets
- Observe Data
- Data Clean
- Split Train-Test Set
- Choose Your ML Model

The LADPS Dataset

```
In [31]: data_csv
```

Out[31]:

	station	Date	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	LDAPS_Tmax_lapse	LDAPS_Tmin_lapse	LDAPS_WS	LDAPS_LH	...	LDAP
0	1.0	2013-06-30	28.7	21.4	58.255688	91.116364	28.074101	23.006936	6.818887	69.451805	...	0
1	2.0	2013-06-30	31.9	21.6	52.263397	90.604721	29.850689	24.035009	5.691890	51.937448	...	0
2	3.0	2013-06-30	31.6	23.3	48.690479	83.973587	30.091292	24.565633	6.138224	20.573050	...	0
3	4.0	2013-06-30	32.0	23.4	58.239788	96.483688	29.704629	23.326177	5.650050	65.727144	...	0
4	5.0	2013-06-30	31.4	21.9	56.174095	90.155128	29.113934	23.486480	5.735004	107.965535	...	0
...	...	...	...	...	...	...	...	...	...	...	...	
7747	23.0	2017-08-30	23.3	17.1	26.741310	78.869858	26.352081	18.775678	6.148918	72.058294	...	0
7748	24.0	2017-08-30	23.3	17.7	24.040634	77.294975	27.010193	18.733519	6.542819	47.241457	...	0
7749	25.0	2017-08-30	23.2	17.4	22.933014	77.243744	27.939516	18.522965	7.289264	9.090034	...	0
7750	NaN	NaN	20.0	11.3	19.794666	58.936283	17.624954	14.272646	2.882580	-13.603212	...	0
7751	NaN	NaN	37.6	29.9	98.524734	100.000153	38.542255	29.619342	21.857621	213.414006	...	21

7752 rows × 25 columns

Data Preprocessing

Preprocessing Steps:

- Load Datasets
- Observe Data
- Data Clean
- Split Train-Test Set
- Choose Your ML Model

There are some unusual data point

```
In [44]: data_csv[270:310]
```

Out[44]:

	station	Date	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	LDAPS_Tmax_lapse	LDAPS_Tmin_lapse	LDAPS_WS	LDAPS_LH	...	LDAPS
270	21.0	2013-07-10	26.2	24.7	69.812981	93.854797	27.970640	24.910094	9.329822	13.594018	...	0.0
271	22.0	2013-07-10	NaN	NaN	72.196007	95.168205	28.097980	24.510159	8.374849	38.782242	...	0.0
272	23.0	2013-07-10	26.4	24.9	77.231949	96.423988	26.700805	24.083975	10.198014	41.800662	...	0.0
273	24.0	2013-07-10	26.6	24.9	73.525047	95.597847	27.198894	24.318422	10.418321	26.420496	...	0.0
274	25.0	2013-07-10	26.8	25.2	69.310020	94.079872	27.941781	24.660475	9.427851	15.670115	...	0.0
275	1.0	2013-07-11	22.4	21.7	95.818939	98.020599	23.356801	21.447190	11.558632	25.803418	...	4.3
276	2.0	2013-07-11	25.2	23.8	89.471741	96.026222	24.905254	22.832034	9.144509	43.099107	...	4.2
277	3.0	2013-07-11	25.5	23.9	84.330078	93.854469	25.542354	23.089602	8.686600	48.461266	...	3.6

Data Preprocessing

Preprocessing Steps:

- Load Datasets
- Observe Data
- Data Clean
- Split Train-Test Set
- Choose Your ML Model

Drop the full data row, or fill the NaN with the mean value

```
# directly drop the NaN row
LDAPS_drop = data_csv.drop(["station", "Date"], axis = 1).dropna()
LDAPS_drop
```

	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	LDAPS_Tmax_lapse	LDAPS_Tmin_lapse	LDAPS_WS	LDAPS_LH	LDAPS_CC1	LDAPS_CC2
0	28.7	21.4	58.255688	91.116364	28.074101	23.006936	6.818887	69.451805	0.233947	0.20389
1	31.9	21.6	52.263397	90.604721	29.850689	24.035009	5.691890	51.937448	0.225508	0.25177
2	31.6	23.3	48.690479	83.973587	30.091292	24.565633	6.138224	20.573050	0.209344	0.25746
3	32.0	23.4	58.239788	96.483688	29.704629	23.326177	5.650050	65.727144	0.216372	0.22600
4	31.4	21.9	56.174095	90.155128	29.113934	23.486480	5.735004	107.965535	0.151407	0.24999
...	...	...	...	...	...	...	...	...	...	..
7747	23.3	17.1	26.741310	78.869858	26.352081	18.775678	6.148918	72.058294	0.030034	0.08103
7748	23.3	17.7	24.040634	77.294975	27.010193	18.733519	6.542819	47.241457	0.035874	0.07496
7749	23.2	17.4	22.933014	77.243744	27.939516	18.522965	7.289264	9.090034	0.048954	0.05986
7750	20.0	11.3	19.794666	58.936283	17.624954	14.272646	2.882580	-13.603212	0.000000	0.00000
7751	37.6	29.9	98.524734	100.000153	38.542255	29.619342	21.857621	213.414006	0.967277	0.96835

7590 rows × 23 columns

Data Preprocessing

Preprocessing Steps:

- Load Datasets
- Observe Data
- Data Clean
- Split Train-Test Set
- Choose Your ML Model

Drop the full data row, or fill the NaN with the mean value

```
## pandas iloc, mean NaN

LDAPS_X = data_csv.iloc[:,2:-2]
LDAPS_Y = data_csv.iloc[:,-2::]

LDAPS_X_clean = LDAPS_X.fillna(LDAPS_X.mean())
LDAPS_Y_clean = LDAPS_Y.fillna(LDAPS_Y.mean())

LDAPS_X_clean.shape, LDAPS_Y_clean.shape
```

```
((7752, 21), (7752, 2))
```

Out[44]:

	station	Date	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax
270	21.0	2013-07-10	26.2	24.7	69.812981	93.854797
271	22.0	2013-07-10	NaN	NaN	72.196007	95.168205
272	23.0	2013-07-10	26.4	24.9	77.231949	96.423988
273	24.0	2013-07-10	26.6	24.9	73.525047	95.597847
274	25.0	2013-07-10	26.8	25.2	69.310020	94.079872
275	1.0	2013-07-11	22.4	21.7	95.818939	98.020599
276	2.0	2013-07-11	25.2	23.8	89.471741	96.026222
277	3.0	2013-07-11	25.5	23.9	84.330078	93.854469

Out[45]:

	station	Date	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax
270	21.0	2013-07-10	26.200000	24.700000	69.812981	93.854797
271	22.0	2013-07-10	29.768211	23.225059	72.196007	95.168205
272	23.0	2013-07-10	26.400000	24.900000	77.231949	96.423988
273	24.0	2013-07-10	26.600000	24.900000	73.525047	95.597847
274	25.0	2013-07-10	26.800000	25.200000	69.310020	94.079872
275	1.0	2013-07-11	22.400000	21.700000	95.818939	98.020599
276	2.0	2013-07-11	25.200000	23.800000	89.471741	96.026222
277	3.0	2013-07-11	25.500000	23.900000	84.330078	93.854469
278	4.0	2013-07-11	25.600000	24.400000	93.700851	98.600471
279	5.0	2013-07-11	25.600000	24.300000	88.987877	96.696747
280	6.0	2013-07-11	26.100000	24.600000	83.727264	95.016861

Data Preprocessing

Preprocessing Steps:

- Load Datasets
- Observe Data
- Data Clean
- Split Train-Test Set
- Choose Your ML Model

```
In [72]: # X
         LDAPS_X_clean
```

Out[72]:

	Present_Tmax	Present_Tmin	LDAPS_RHmin	LDAPS_RHmax	LDAPS_Tmax_lapse	LDAPS_Tmin_lapse	LDAPS_WS	LDAPS_LH	LDAPS_CC1	LDAPS_CC2
0	28.7	21.4	58.255688	91.116364	28.074101	23.006936	6.818887	69.451805	0.233947	0.20389
1	31.9	21.6	52.263397	90.604721	29.850689	24.035009	5.691890	51.937448	0.225508	0.25177
2	31.6	23.3	48.690479	83.973587	30.091292	24.565633	6.138224	20.573050	0.209344	0.25746
3	32.0	23.4	58.239788	96.483688	29.704629	23.326177	5.650050	65.727144	0.216372	0.22600
4	31.4	21.9	56.174095	90.155128	29.113934	23.486480	5.735004	107.965535	0.151407	0.24999
...	...	...	...	...	...	...	...	...	...	..
7747	23.3	17.1	26.741310	78.869858	26.352081	18.775678	6.148918	72.058294	0.030034	0.08103
7748	23.3	17.7	24.040634	77.294975	27.010193	18.733519	6.542819	47.241457	0.035874	0.07496
7749	23.2	17.4	22.933014	77.243744	27.939516	18.522965	7.289264	9.090034	0.048954	0.05986
7750	20.0	11.3	19.794666	58.936283	17.624954	14.272646	2.882580	-13.603212	0.000000	0.00000
7751	37.6	29.9	98.524734	100.000153	38.542255	29.619342	21.857621	213.414006	0.967277	0.96835

7752 rows × 21 columns

```
# Y
LDAPS_Y_clean
```

	Next_Tmax	Next_Tmin
0	29.1	21.2
1	30.5	22.5
2	31.1	23.9
3	31.7	24.3
4	31.2	22.5
...	...	...
7747	28.3	18.1
7748	28.6	18.8
7749	27.8	17.4
7750	17.4	11.3
7751	38.9	29.8

7752 rows × 2 columns

Data Preprocessing

Sometimes we need to normalize the datasets

Preprocessing Steps:

- Load Datasets
- Observe Data
- Data Clean
- Split Train-Test Set
- Choose Your ML Model

```
scaler = sk_preprocessing.StandardScaler().fit(LDAPS_X_clean)
LDAPS_std_X = scaler.transform(LDAPS_X_clean)
print('Normalization with std:\n', LDAPS_std_X)
```

```
Normalization with std:
 [[-0.36132577 -0.75952931  0.10251523 ...  2.77224286  1.11500407
   1.51793488]
 [ 0.72108401 -0.6762959  -0.30802721 ... -0.31515742 -0.54215762
   1.22994952]
 [ 0.61960809  0.03118815 -0.55281415 ... -0.52621832 -0.7231326
   1.21653443]
 ...
 [-2.22171758 -2.42419767 -2.31750374 ... -0.77904331 -0.71933797
  -2.0743251 ]
 [-3.30412736 -4.9628169  -2.5325175  ... -0.91196325 -0.8454552
  -2.35821196]
 [ 2.64912642  2.77789093  2.86141898 ...  2.77224286  2.86143459
   1.51793488]]
```

```
# std norm Y
scaler_y = sk_preprocessing.StandardScaler().fit(LDAPS_Y_clean)
LDAPS_std_Y = scaler_y.transform(LDAPS_Y_clean)
print('Normalization with std:\n', LDAPS_std_Y)
```

```
Normalization with std:
 [[-0.37628214 -0.6975993 ]
 [ 0.07209725 -0.17406357]
 [ 0.26425985  0.38974413]
 ...
 [-0.79263444 -2.2279345 ]
 [-4.12345278 -4.68452522]
 [ 2.76237361  2.76579089]]
```

```
scaler = sk_preprocessing.MinMaxScaler(feature_range=(0,1)).fit(LDAPS_X_clean)
LDAPS_mm_norm = scaler.transform(LDAPS_X_clean)
print('normalize into the interval [0, 1]: \n', LDAPS_mm_norm)
```

```
normalize into the interval [0, 1]:
 [[0.49431818 0.54301075 0.48851757 ... 1.         0.52886905 1.        ]
 [0.67613636 0.55376344 0.41240573 ... 0.16199035 0.08181996 0.92570319]
 [0.65909091 0.64516129 0.36702385 ... 0.10470232 0.03299871 0.92224225]
 ...
 [0.18181818 0.32795699 0.03986212 ... 0.03607831 0.03402239 0.07323945]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [1.         1.         1.         ... 1.         1.         1.        ]]
```

Data Preprocessing

Preprocessing Steps:

- Load Datasets
- Observe Data
- Data Clean
- Split Train-Test Set
- Choose Your ML Model

```
LDAPS_std_X.shape, LDAPS_std_Y.shape
```

```
((7752, 21), (7752, 2))
```

Split into two 80%-20% Train-Test Sets

```
# LDAPS split
import sklearn.model_selection as sk_model_selection
X_train, X_test, y_train, y_test = sk_model_selection.train_test_split(LDAPS_std_X,
                                                                       LDAPS_std_Y,
                                                                       train_size=0.8,
                                                                       random_state=20)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((6201, 21), (1551, 21), (6201, 2), (1551, 2))
```

Data Preprocessing

Preprocessing Steps:

- Load Datasets
- Observe Data
- Data Clean
- Split Train-Test Set
- Choose Your ML Model

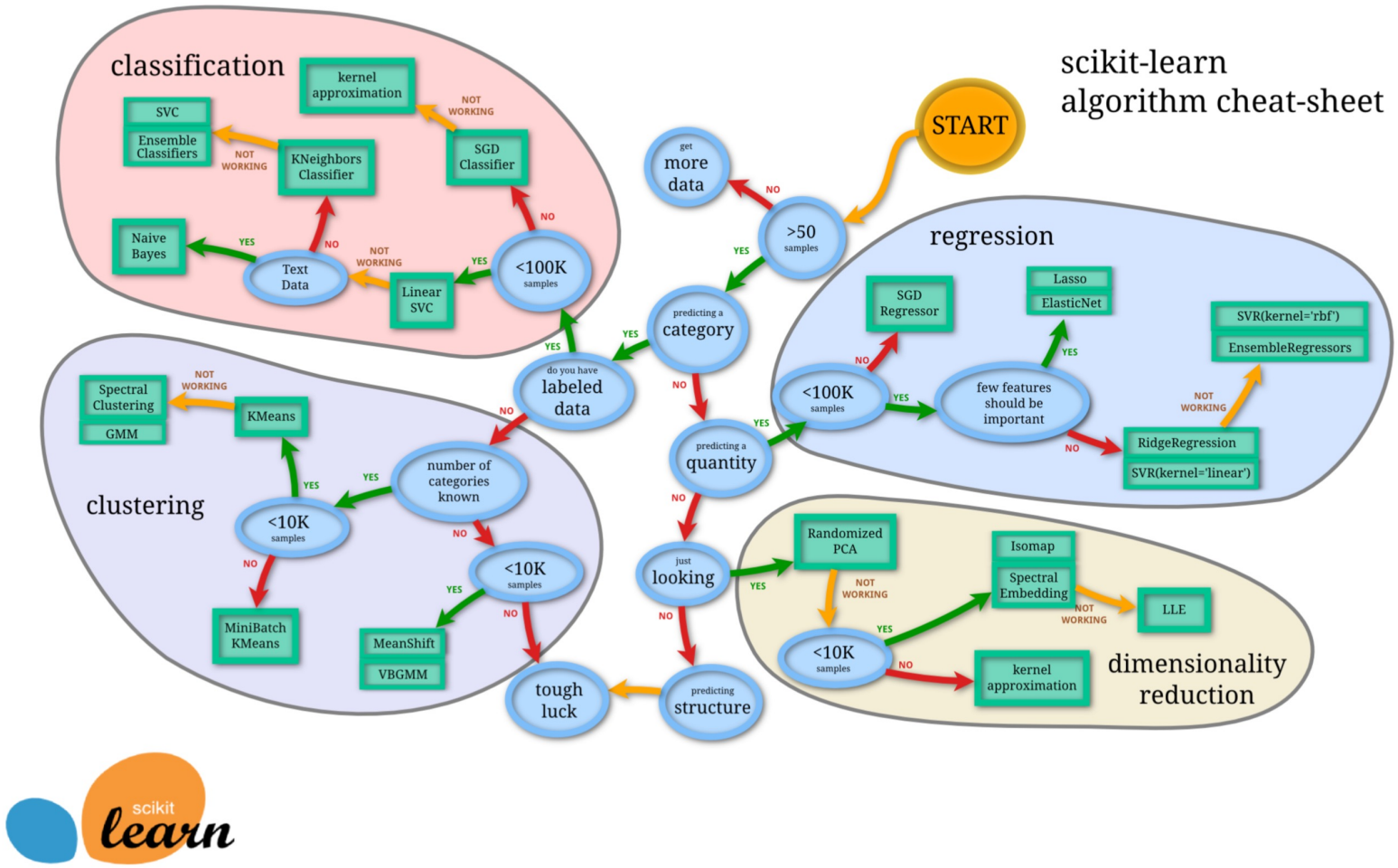


Figure 1: Machine learning algorithm selection diagram for scikit-learn.

The hundred-page machine learning book-Andriy Burkov (2019)

Example 1: Iris Classification

```
# Iris split
import sklearn.model_selection as sk_model_selection
iris_X_train, iris_X_test, iris_y_train, iris_y_test = sk_model_selection.train_test_split(
    iris_std_X,iris_y,train_size=0.8,random_state=20)

iris_X_train.shape, iris_X_test.shape, iris_y_train.shape, iris_y_test.shape
```

```
((120, 4), (30, 4), (120,), (30,))
```

Adaptive Moment Estimation

```
import sklearn.neural_network as sk_nn
model = sk_nn.MLPClassifier(activation='tanh',solver='adam',alpha=0.0001,learning_rate='adaptive',
                            learning_rate_init=0.001,max_iter=2000)
model.fit(iris_X_train, iris_y_train)
predi = model.predict(iris_X_test)
acc = model.score(iris_X_test, iris_y_test) # prediction correctness rate

print('Correctness Rate:\n', acc), predi, iris_y_test
```

```
Correctness Rate:
 0.9666666666666667

(None,
 array([0, 1, 1, 2, 1, 1, 2, 0, 2, 0, 2, 1, 2, 0, 0, 2, 0, 1, 2, 1, 1, 2,
        2, 0, 1, 1, 1, 0, 2, 1]),
 array([0, 1, 1, 2, 1, 1, 2, 0, 2, 0, 2, 1, 2, 0, 0, 2, 0, 1, 2, 1, 1, 2,
        2, 0, 1, 1, 1, 0, 2, 2]))
```

```
RMSE = math.sqrt(np.square(np.subtract(predi, iris_y_test)).mean())

RMSE
```

```
0.18257418583505536
```

Example 1: Iris Classification

Stochastic Gradient Descent

```
# SGD iris
model_SGD = sk_nn.MLPClassifier(activation='tanh', solver='sgd', alpha=0.0001, learning_rate='adaptive',
                                learning_rate_init=0.001, max_iter=2000)
model_SGD.fit(iris_X_train, iris_y_train)
predi_SGD = model_SGD.predict(iris_X_test)
acc_SGD = model_SGD.score(iris_X_test, iris_y_test)
RMSE_sgd = math.sqrt(np.square(np.subtract(predi_SGD, iris_y_test)).mean())


print('Rate:\n', acc_SGD), predi_SGD, iris_y_test, RMSE_sgd
```

```
Rate:
 0.9333333333333333

(None,
 array([0, 1, 1, 2, 1, 1, 2, 0, 2, 0, 2, 1, 1, 0, 0, 2, 0, 1, 2, 1, 1, 2,
        2, 0, 1, 1, 1, 0, 2, 1]),
 array([0, 1, 1, 2, 1, 1, 2, 0, 2, 0, 2, 1, 2, 0, 0, 2, 0, 1, 2, 1, 1, 2,
        2, 0, 1, 1, 1, 0, 2, 2]),
 0.2581988897471611)
```

Limited-memory Broyden-Fletcher-Goldfarb-Shanno

```
# LBFGS iris
model_lb = sk_nn.MLPClassifier(activation='relu', solver='lbfgs', alpha=0.0001, learning_rate='adaptive',
                               learning_rate_init=0.001, max_iter=2000)
model_lb.fit(iris_X_train, iris_y_train)
predi_lb = model_lb.predict(iris_X_test)
acc_lb = model_lb.score(iris_X_test, iris_y_test)
RMSE_lbfgs = math.sqrt(np.square(np.subtract(predi_lb, iris_y_test)).mean())

print('Rate:\n', acc_lb), predi_lb, iris_y_test
```

```
Rate:
 0.9333333333333333

(None,
 array([0, 1, 1, 2, 1, 1, 2, 0, 2, 0, 2, 1, 1, 0, 0, 2, 0, 1, 2, 1, 1, 2,
        2, 0, 1, 1, 1, 0, 2, 1]),
 array([0, 1, 1, 2, 1, 1, 2, 0, 2, 0, 2, 1, 2, 0, 0, 2, 0, 1, 2, 1, 1, 2,
        2, 0, 1, 1, 1, 0, 2, 2]))
```

Example 2: LDAPS Regression

```
# build a 2 hidden layers NN
import sklearn.neural_network as sk_nn
model = sk_nn.MLPRegressor(activation='tanh', hidden_layer_sizes = (50, 50), solver='adam', max_iter=5000)
model.fit(X_train,y_train)
Y_predict = model.predict(X_test)
print('Prediction of f(test_X):\n', Y_predict)
```

```
Prediction of f(test_X):
 [[ 1.43682978  1.16296132]
 [-1.46865143 -0.99652923]
 [-0.01273257  0.60038319]
 ...
 [-0.5165651   0.15398494]
 [ 0.67950669  0.48070747]
 [ 1.65730677  1.50085333]]
```

```
Y_predict, y_test
```

```
(array([[ 1.43682978,  1.16296132],
        [-1.46865143, -0.99652923],
        [-0.01273257,  0.60038319],
        ...,
        [-0.5165651 ,  0.15398494],
        [ 0.67950669,  0.48070747],
        [ 1.65730677,  1.50085333]]),
 array([[ 1.35318123,  1.23545569],
        [-1.56128482, -0.09351962],
        [ 0.00804305,  0.59110403],
        ...,
        [-0.05601115, -0.41569545],
        [ 0.52047664,  0.59110403],
        [ 1.28912703,  1.71871944]]))
```

Example 2: LDAPS Regression

```
# build a 2 hidden layers NN
import sklearn.neural_network as sk_nn
model = sk_nn.MLPRegressor(activation='tanh', hidden_layer_sizes = (50, 50), solver='adam', max_iter=5000)
model.fit(X_train,y_train)
Y_predict = model.predict(X_test)
print('Prediction of f(test_X):\n', Y_predict)
```

```
Prediction of f(test_X):
 [[ 1.43682978  1.16296132]
 [-1.46865143 -0.99652923]
 [-0.01273257  0.60038319]
 ...
 [-0.5165651   0.15398494]
 [ 0.67950669  0.48070747]
 [ 1.65730677  1.50085333]]
```

```
RMSE_LA = math.sqrt(np.square(np.subtract(Y_predict, y_test)).mean())

RMSE_LA
```

```
0.33024128923885576
```

```
LDAPS_metric = model.score(X_test, y_test)
LDAPS_metric
```

```
0.8929162304305842
```

Remark

MLPClassifier use Cross entropy(log-loss) as loss function
MLPRegressor use MSE as loss function
More details refer to the sklearn tutorial website

3

According to the docs:

> This model optimizes the log-loss function using LBFGS or stochastic gradient descent.

Log-loss is basically the same as cross-entropy.

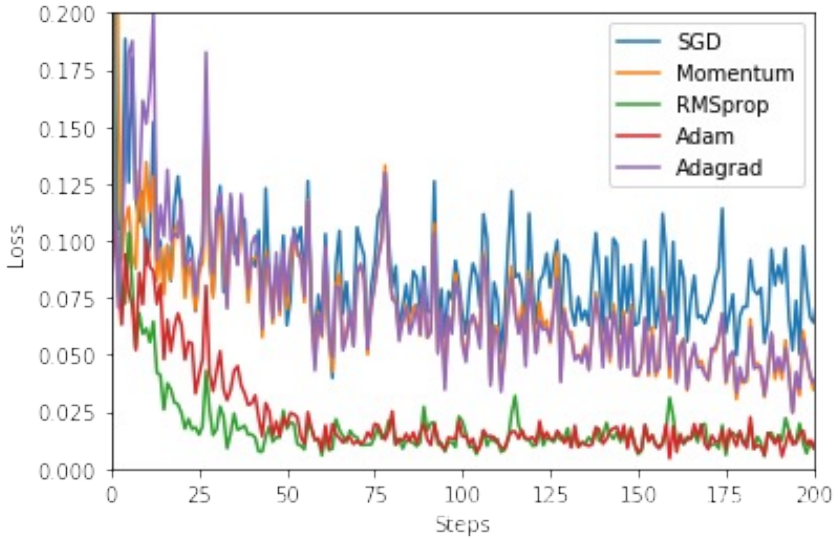
There is no way to pass another loss function to `MLPClassifier`, so you cannot use MSE. But `MLPRegressor` uses MSE, if you really want that.

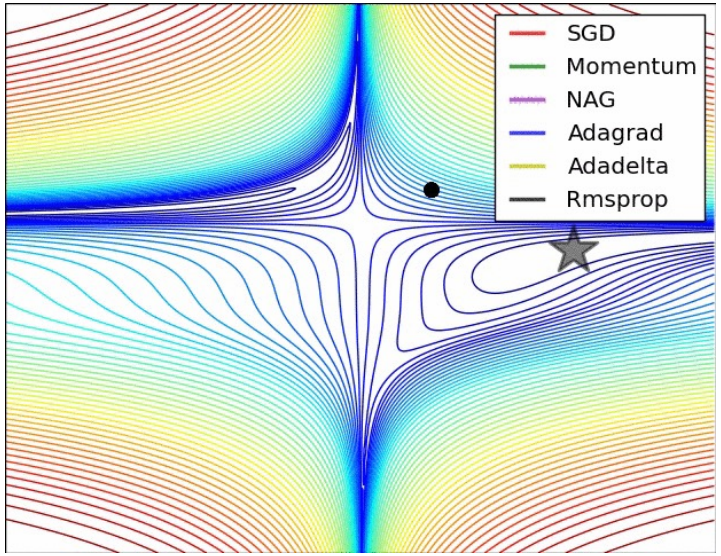
However, the general advice is to stick to cross-entropy loss for classification, it is said to have some advantages over MSE. So you may just want to use `MLPClassifier` as is for your classification problem.

Optimizer

Common Optimizers:

- SGD
- Nesterov Momentum
- ASGD
- Rprop
- Adagrad
- Adadelta
- RMSprop
- Adam(AMSGrad)
- Adamax
- SparseAdam
- LBFGS

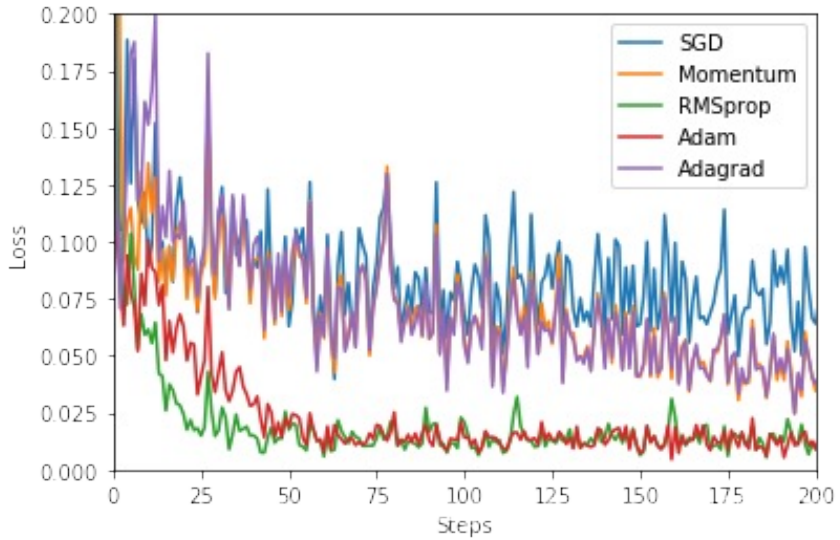


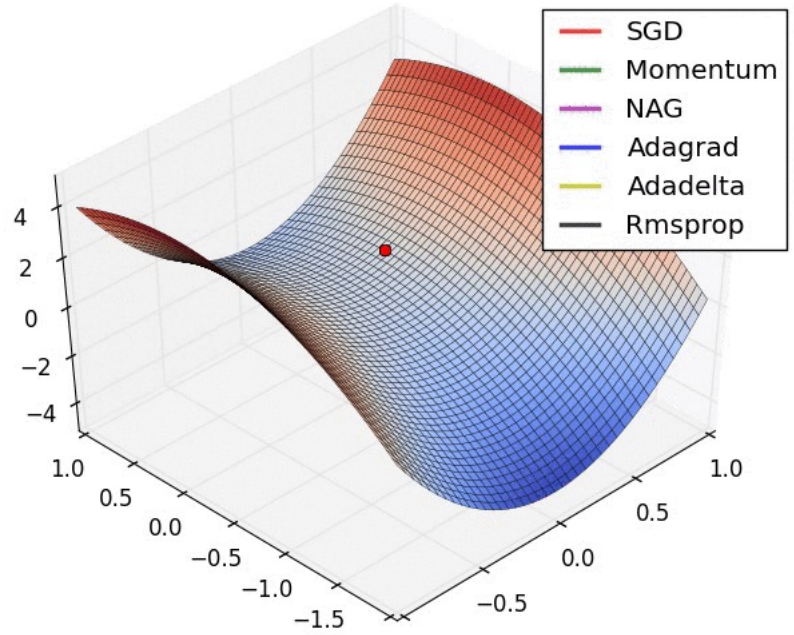


Optimizer

Common Optimizers:

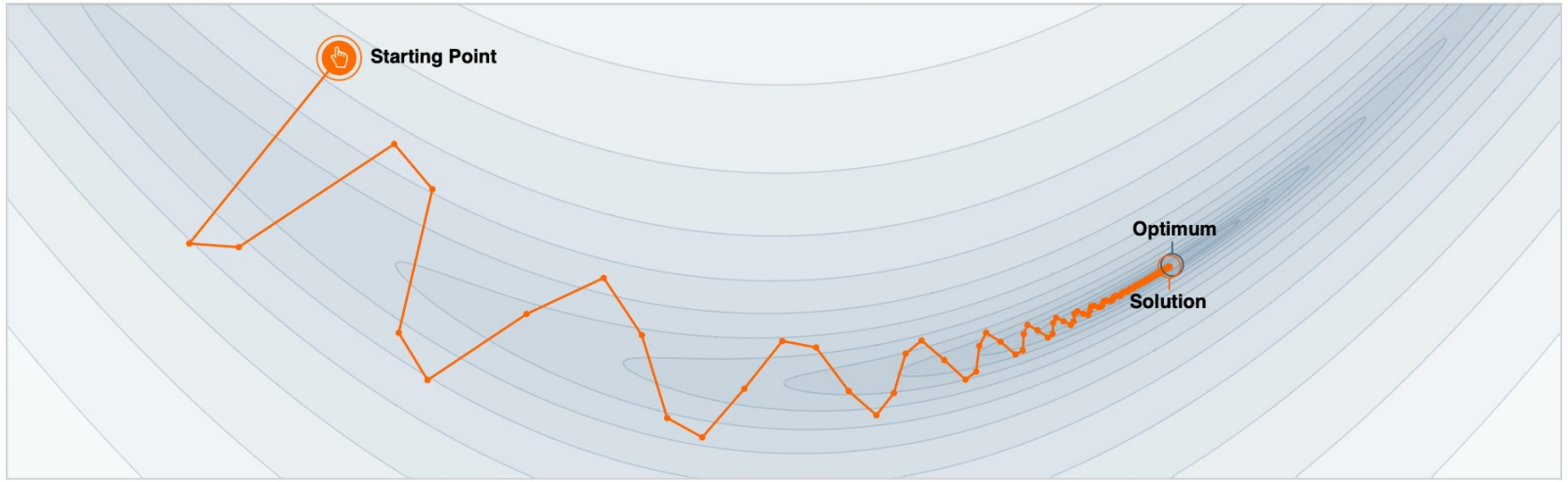
- SGD
- Nesterov Momentum
- ASGD
- Rprop
- Adagrad
- Adadelta
- RMSprop
- Adam(AMSGrad)
- Adamax
- SparseAdam
- LBFGS





Optimizer

Why Momentum Really Works



Step-size α = 0.0030

0 0.003 0.006

Momentum β = 0.83

0.00 0.500 0.990

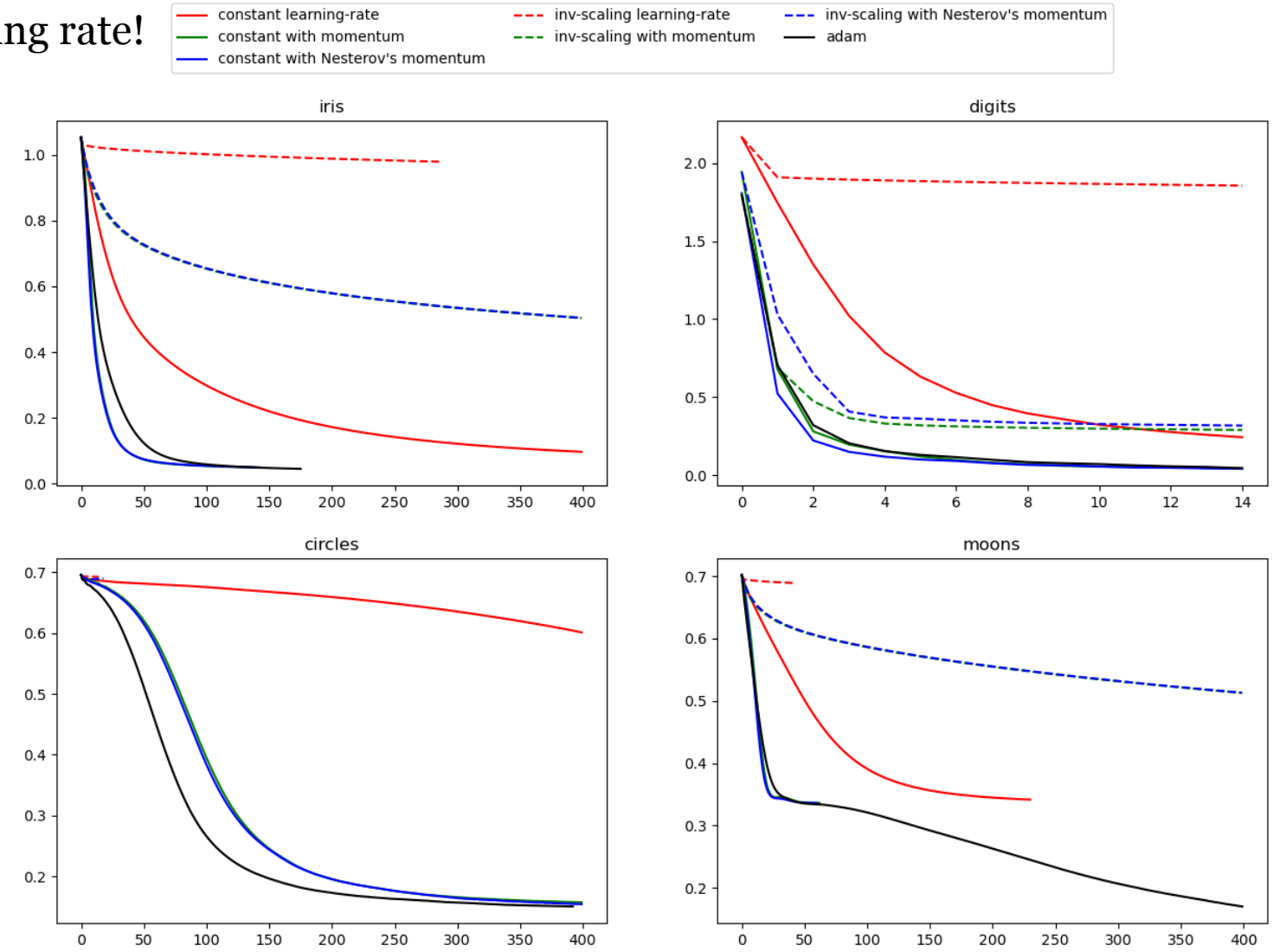
We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

GABRIEL GOH UC Davis
April. 4 2017
Citation: Goh, 2017

https://distill.pub/2017/momentum/

Optimizer Parameter Selection

Take care your learning rate!



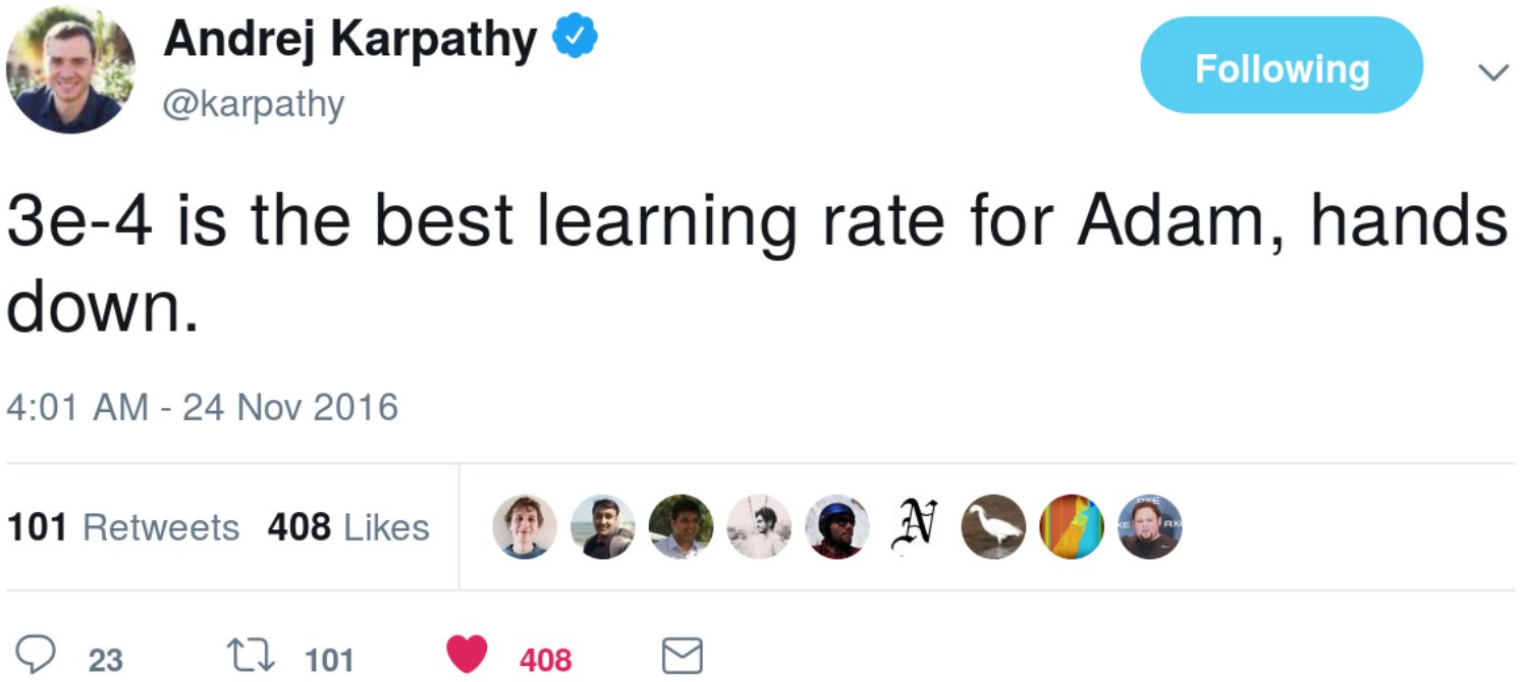
SGD learning rate

$$x_{t+1} = x_t - \eta \nabla f_i(x_t)$$

- Very sensitive:
  - Too high → early plateau or even divergence
  - Too low → slow convergence
  - Try a large value first: $\eta = 0.1$ or even $\eta = 1$
  - Divide by 10 and retry in case of divergence
- SGD (with Nesterov momentum)
  - Simple to implement
  - Very sensitive to initial value of η
  - Need learning rate scheduling
- Adam: adaptive learning rate scale for each param
  - Global η set to 3e-4 often works well enough
  - Good default choice of optimizer (often)

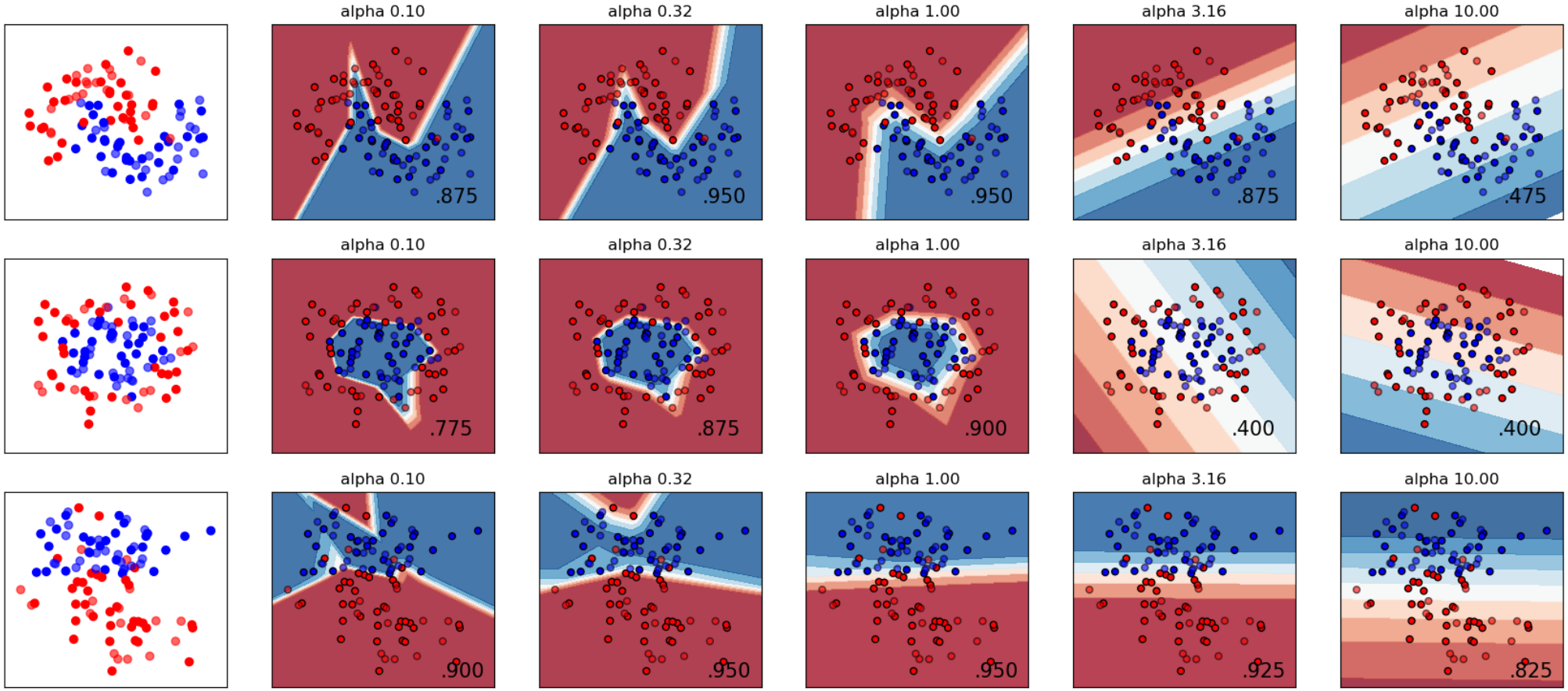
Optimizer Parameter Selection

If you don't know how to select learning rate, maybe you can follow this *Karpathy Constant*

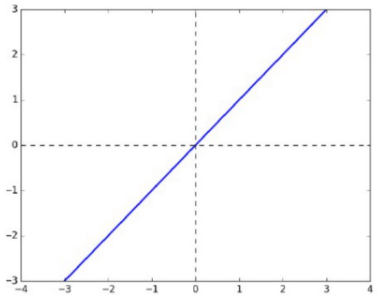


Regularization Parameter Selection

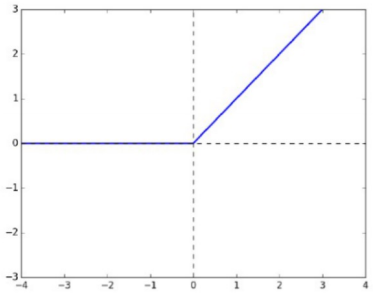
Train SVM model with different regularization parameters \alpha from 0.10 to 10.00



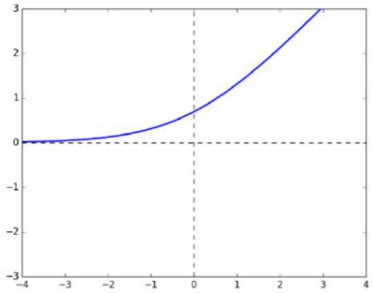
Activation Functions Selection



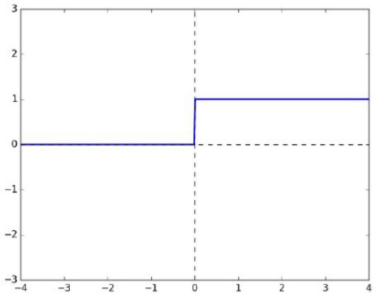
(a) Linear
$y = z$



(b) Rectified Linear Unit(ReLU) $y = \max(0, z)$

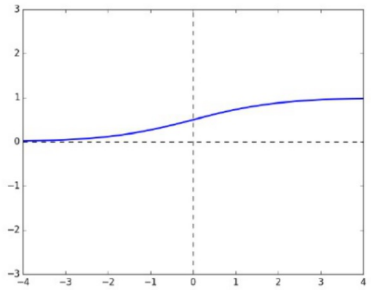


(c) Soft ReLU $y = \log 1 + e^z$

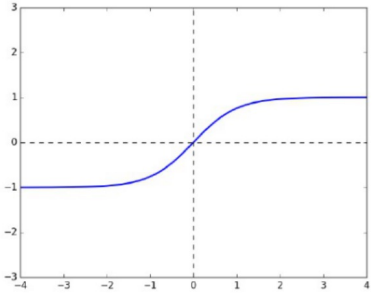


(d) Hard Threshold

$$y = \begin{cases} 1 & \text{if } z > 0 \\ 0 & \text{if } z \leq 0 \end{cases}$$



(e) Logistic
$y = \frac{1}{1+e^{-z}}$

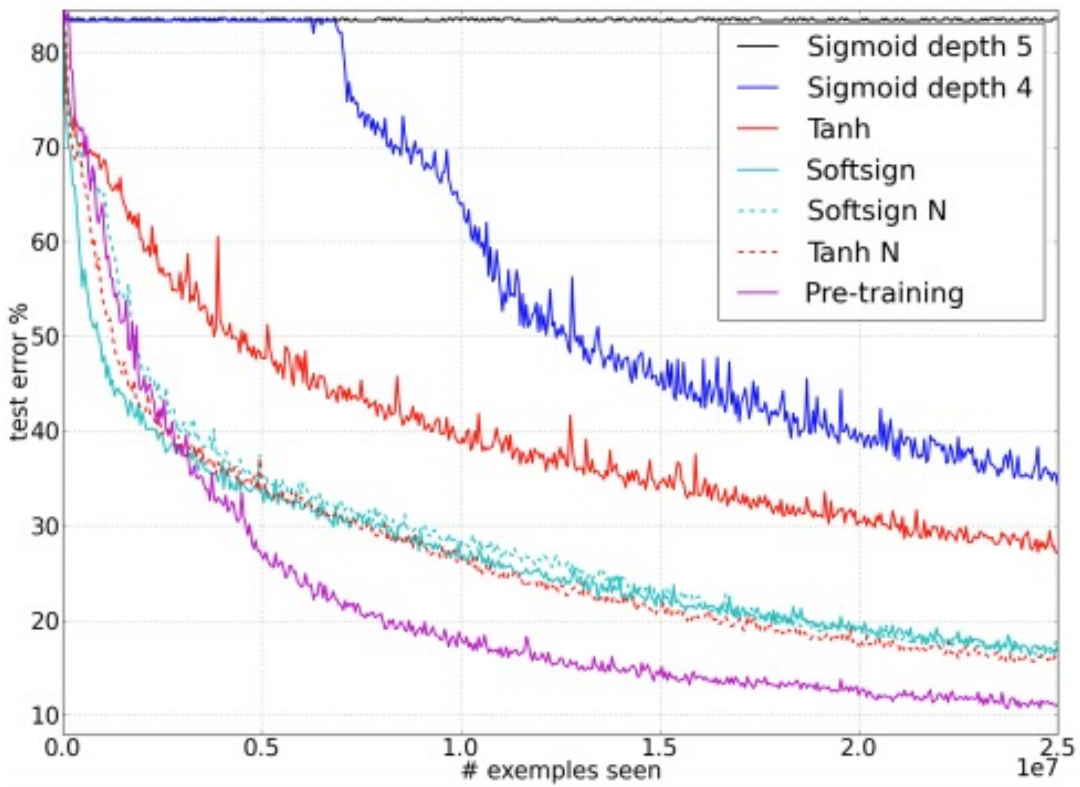


(f) Hyperbolic Tangent (tanh)
$y = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

Activation Functions Selection

Understanding the difficulty of training deep feedforward neural networks

Xavier Glorot **Yoshua Bengio**
DIRO, Université de Montréal, Montréal, Québec, Canada

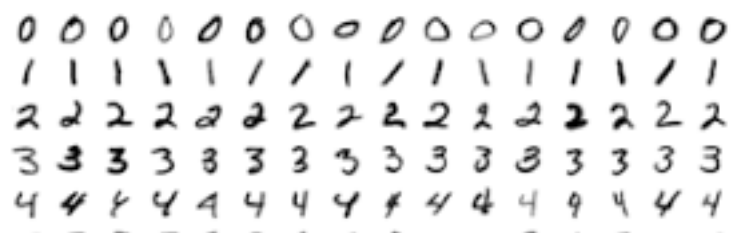


sigmoid vs. tanh

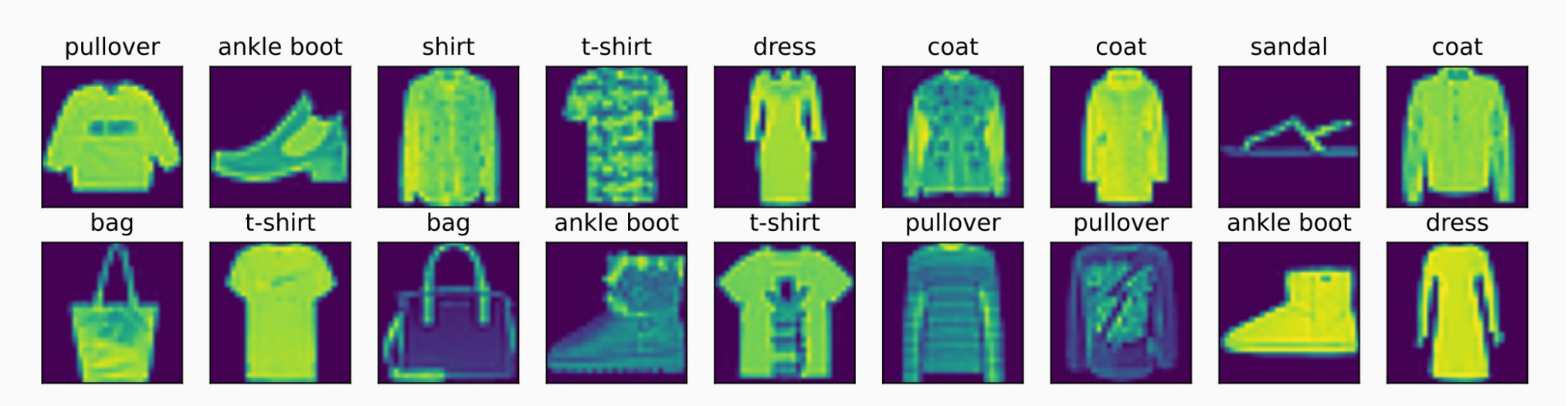
Figure from Glorot & Bentio (2010)

Extension: MNIST Classification by Pytorch

Pytorch Deep NN for MNIST
PyTorch Deep Explainer MNIST example: https://www.kaggle.com/code/ceshine/pytorch- deep-explainer-mnist-example/notebook



Dive into Deep Learning_Li Mu, Fashion-MNIST datasets
http://zh.d2l.ai/chapter_linear-networks/image-classification-dataset.html



Xiao, H., Rasul, K., & Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*.

Thanks!

Dan Qiao
danqiao@link.cuhk.edu.cn
OH: 16:00~17:00, Friday, Seat 81, SDS Lab in ZX Building
Nov. 8, 2022