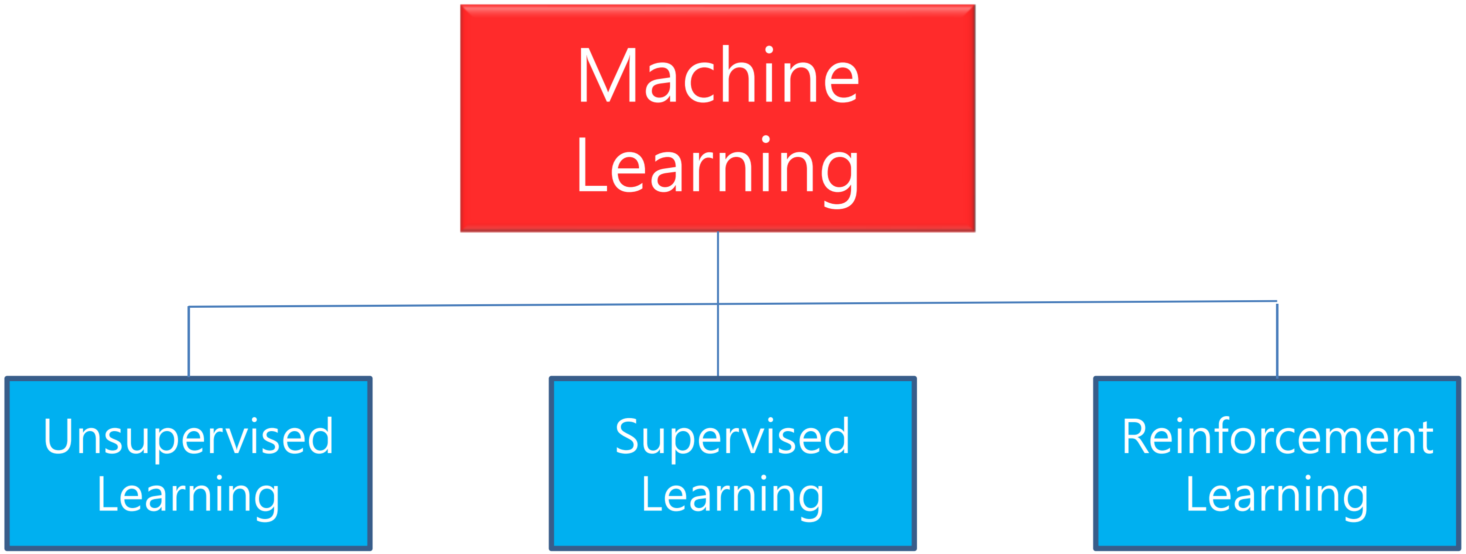
Machine Learning



Unsupervised Learning	Supervised Learning	Reinforcement Learning
(Clustering)	(Regression / Classification)	Algorithms learns to react to an environment
K-means	Neural Network Linear Regression Logistic Regression	Q-learning Deep Q-learning

Supervised Learning

1. classification problem (binary classification)
2. classification problem (categorical classification)
3. regression problem

1. Neural Network (Deep Neural Network)
2. Decision Tree
3. Random Forest
4. AdaBoost (Adaptive Boosting)
5. Gradient Boosting

Neural Network Examples 1 - 3

- (Ex 1) Banknote authentication
  - Binary classification problem
  - 1372 instances
  - non-authentic(위조) 762 + authentic(진품) 610
- (Ex 2) MNIST database
  - Modified National Institute of Standards and Technology
  - Handwritten digits from 0 to 9
  - Categorical classification problem
  - 60000 training images
  - 10000 testing images
- (Ex 3) house price
  - Boston house price
  - Regression problem

(Ex 1) Banknote authentication

```
import numpy as np
import pandas as pd
df = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/00267/data_banknote_authentication.txt')
```

"http://archive.ics.uci.edu/ml/datasets/banknote+authentication"

instance (sample)

attribute (feature)

class (level)

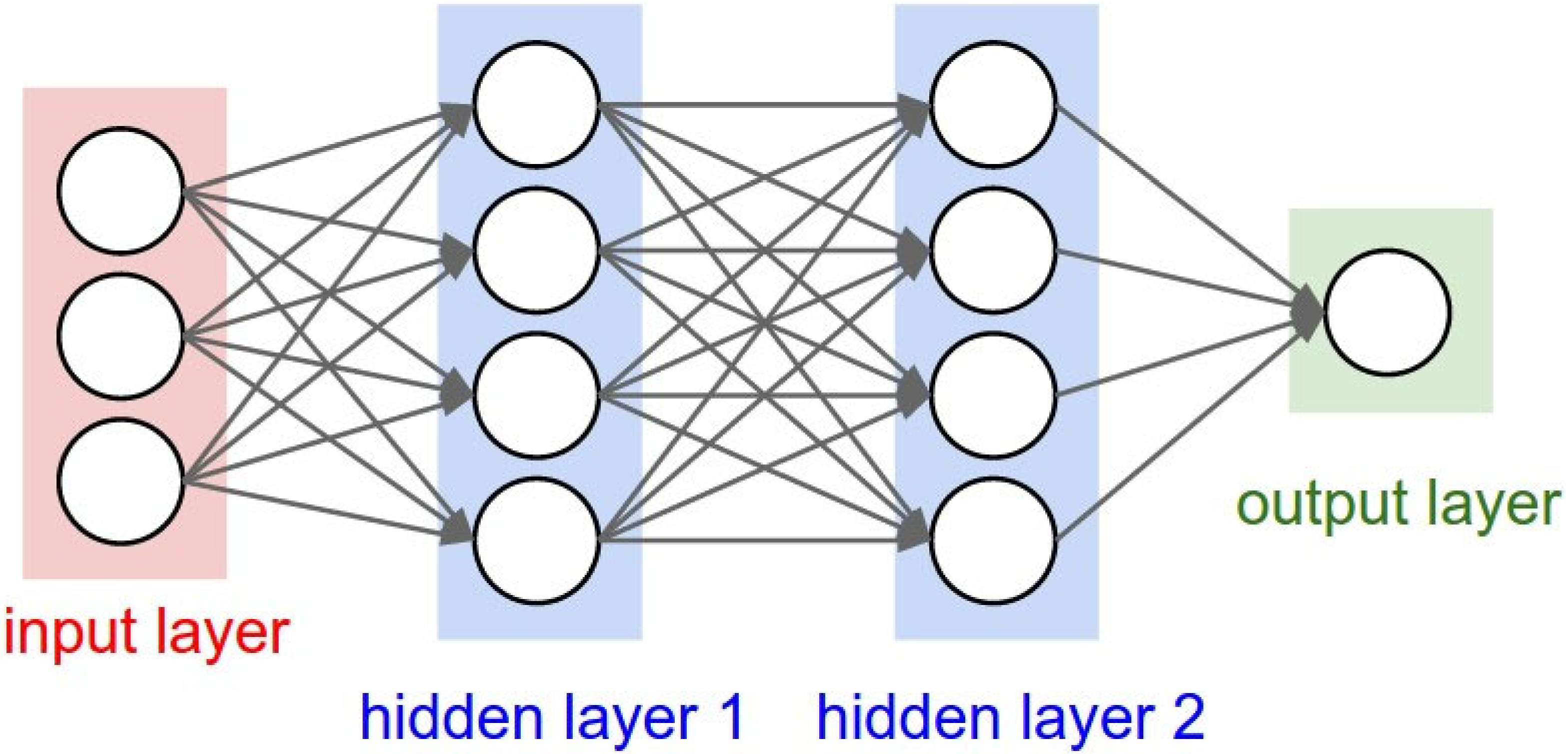
instance, attribute, class

- 1372 instances
- Attributes
  - Variance of Wavelet transformed image
  - Skewness
  - Kurtosis
  - Entropy
  - Class (0-non authentic, 1-authentic) 762+610

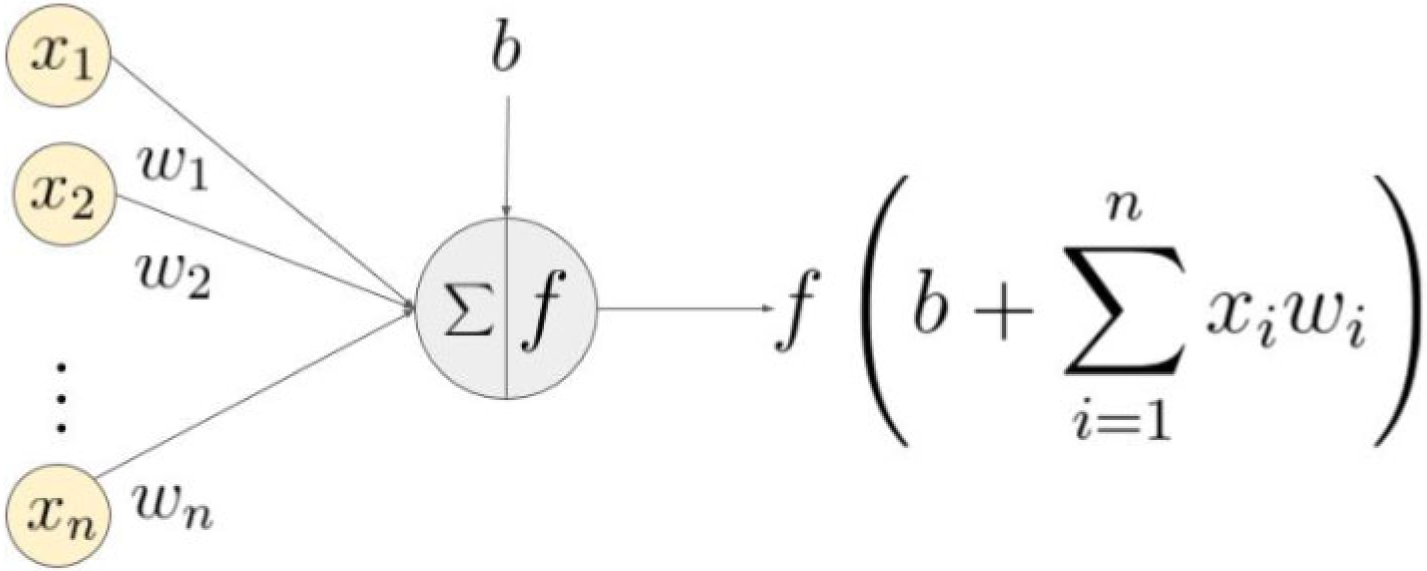
Neural Network for banknote

```
import numpy as np
import pandas as pd
columns = ['var','skewness','kurtosis','entropy','class']
df = pd.read_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/00267/data_banknote_authentication.txt',names=columns)
data = np.array(df)
x = data[:, :4]
y = data[:, 4]
from keras.models import Sequential
from keras.layers import Dense
model=Sequential()
model.add(Dense(6,input_dim=4,activation='relu'))
model.add(Dense(1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
model.fit(x,y,epochs=10)
y_ = model.predict(x).flatten()
y_pred = (y_ >= 0.5)
acc = np.mean(y_pred == y)
loss = np.sum( y*(-np.log(y_)) + (1-y)*(-np.log(1-y_)) ) / len(y)
loss, acc
```

Neural Network

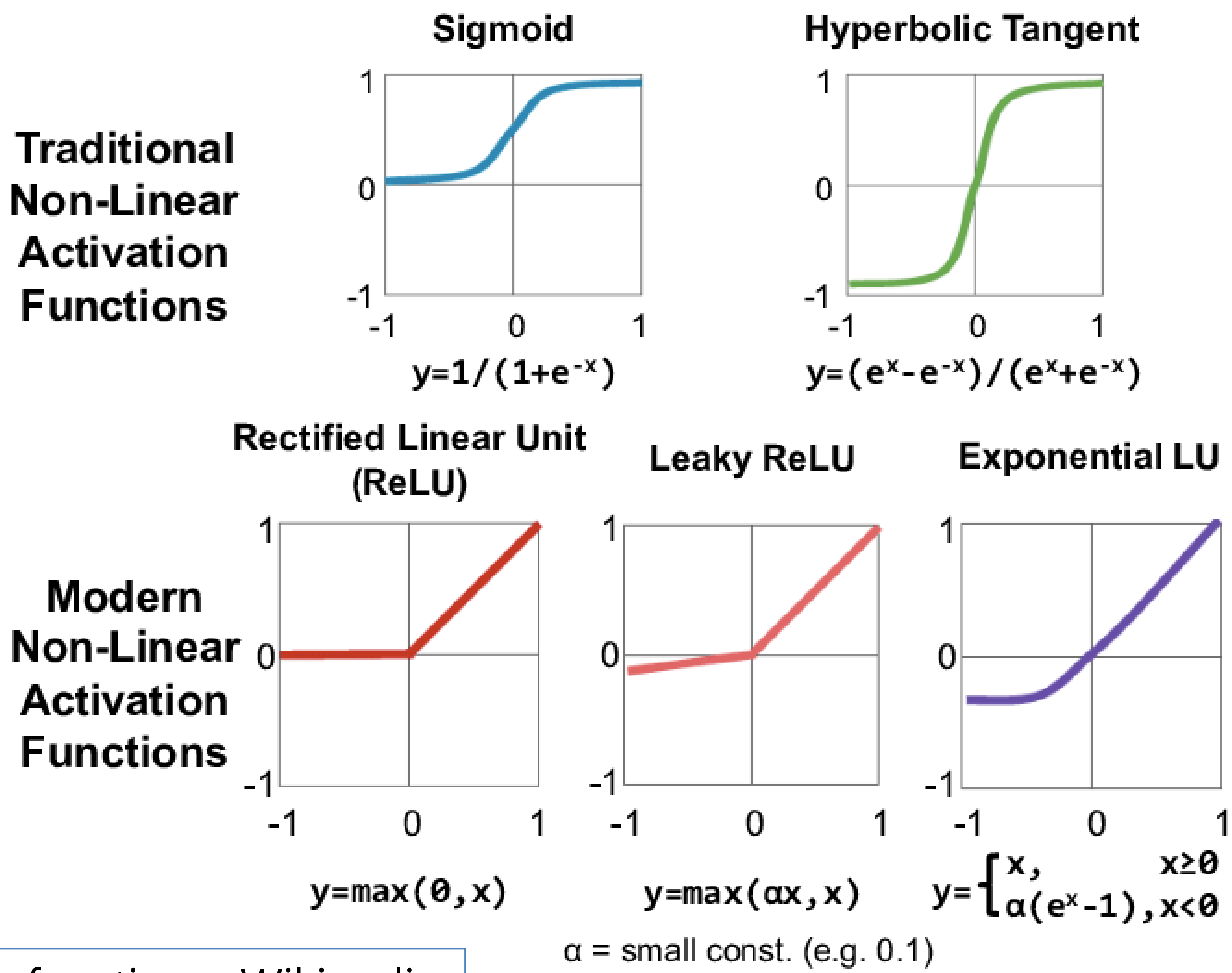


Inputs and outputs



An example of a neuron showing the input (x_1 - x_n), their corresponding weights (w_1 - w_n), a bias (b) and the activation function f applied to the weighted sum of the inputs.

Activation functions



Activation function - Wikipedia

Gradient Descent method

$$x^{(k+1)} = x^{(k)} + t \times \Delta x \qquad (k = 0,1,2,\dots)$$

Δx is the search direction

t is the step size or learning rate

Given a starting point $x^{(0)}$

Repeat

1. Determine a search direction, $\Delta x = -\nabla f(x)$
2. Choose a step size, t
3. Update $x^{(k+1)} = x^{(k)} + t \times (-\nabla f(x))$

Until stopping criteria is satisfied

1. $x^{(k+1)} - x^{(k)} < 0.0001$
2. $f(x^{(k+1)}) - f(x^{(k)}) < 0.0001$
3. $\text{iter} > 100$

Stochastic Gradient Descent

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x)$$

$$x^{(k+1)} = x^{(k)} - t\nabla f\left(x^{(k)}\right)$$

$$\nabla f(x) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x) \approx \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(x) \qquad (m \ll n)$$

Momentum

Local minimum and saddle point

$$v^{(k+1)} = \rho v^{(k)} + \nabla f(x) \qquad \rho = 0.9 \; or \; 0.99$$

$$x^{(k+1)} = x^{(k)} - t v^{(k+1)}$$

(tf.keras.optimizers)

v(t+1) = momentum * v(t) - learning_rate * gradient

theta(t+1) = theta(t) + v(t+1)

Nesterov Momentum

(momentum)

Gradient is evaluated at theta(t)

(nesterov momentum)

Gradient is evaluated at theta(t) + momentum * v(t)

Adagrad

Adagrad is an optimizer with parameter-specific learning rates, which are adapted relative to how frequently a parameter gets updated during training. The more updates a parameter receives, the smaller the updates.

$$accum_{g_t} = accum_{g_{t-1}} + g^2$$

$$\theta_t = \theta_{t-1} - lr * g / \left(\sqrt{accum_{g_t}} + \epsilon \right)$$

(default value)

learning_rate = 0.001

initial_accumulator_value = 0.1

epsilon = 1e-07

RMSprop

RMSprop uses a moving(discounted) average of the square of the gradients

$$square_t = \rho * square_{t-1} + (1-\rho)g^2$$

$$m_t = momentum * m_{t-1} + lr * g/(\sqrt{square_t} + \epsilon)$$

$$\theta_t = \theta_{t-1} - m_t$$

(default value)
learning_rate = 0.001
rho = 0.9
momentum = 0.0
epsilon = 1e-07

Adam

Adam combines momentum and AdaGrad/RMSprop

$$lr_t = lr * \sqrt{1 - {\beta_2}^t}/(1 - {\beta_1}^t)$$

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1)g$$

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2)g^2$$

$$\theta_t = \theta_{t-1} - lr_t m_t/(\sqrt{v_t} + \epsilon)$$

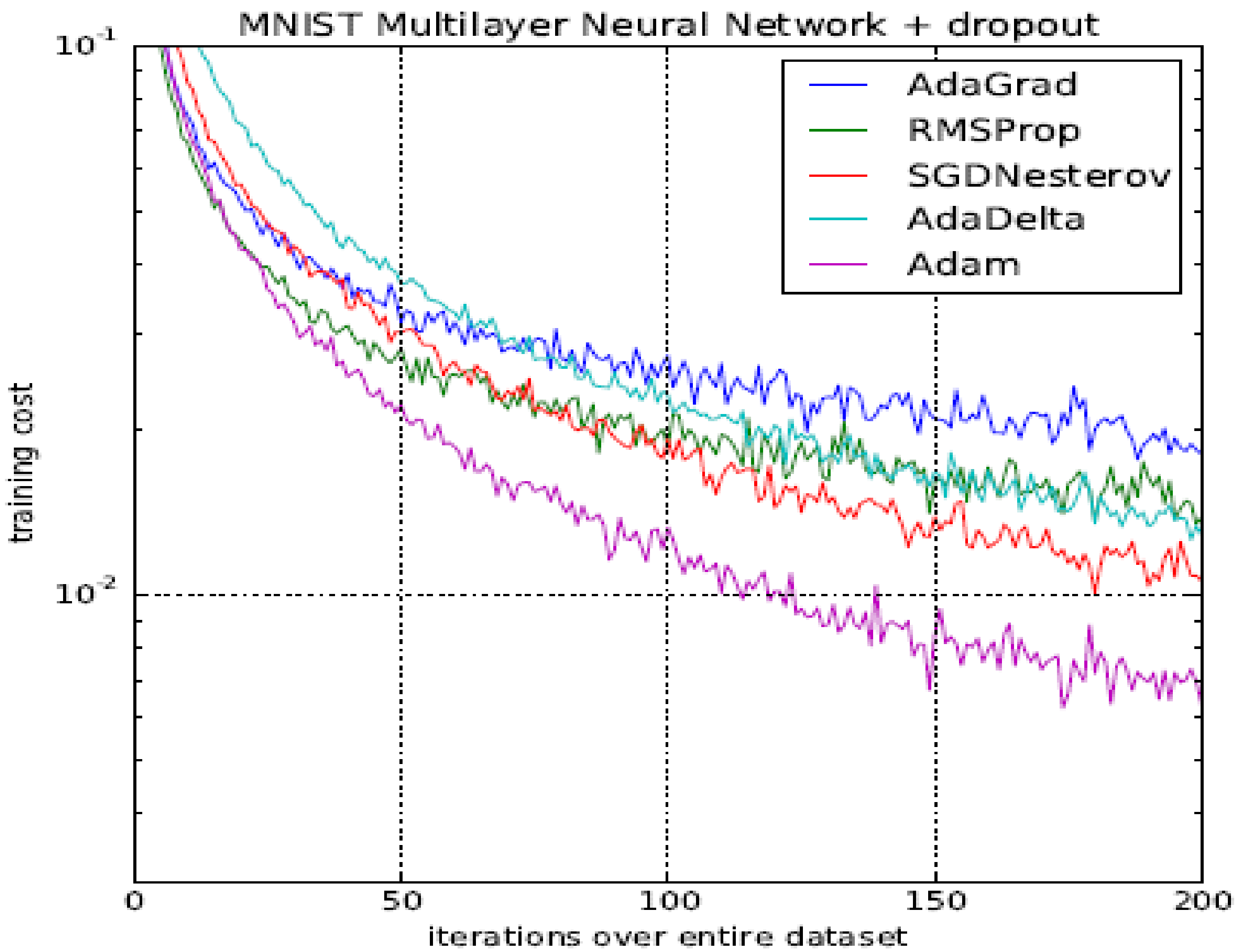
(default value)
learning_rate = 0.001
beta_1 = 0.9
beta_2 = 0.999
epsilon = 1e-07

Adam (2014,v1)(2017,v9)

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1$^{\text{st}}$ moment vector)
  $v_0 \leftarrow 0$ (Initialize 2$^{\text{nd}}$ moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t/(1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t/(1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t/(\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

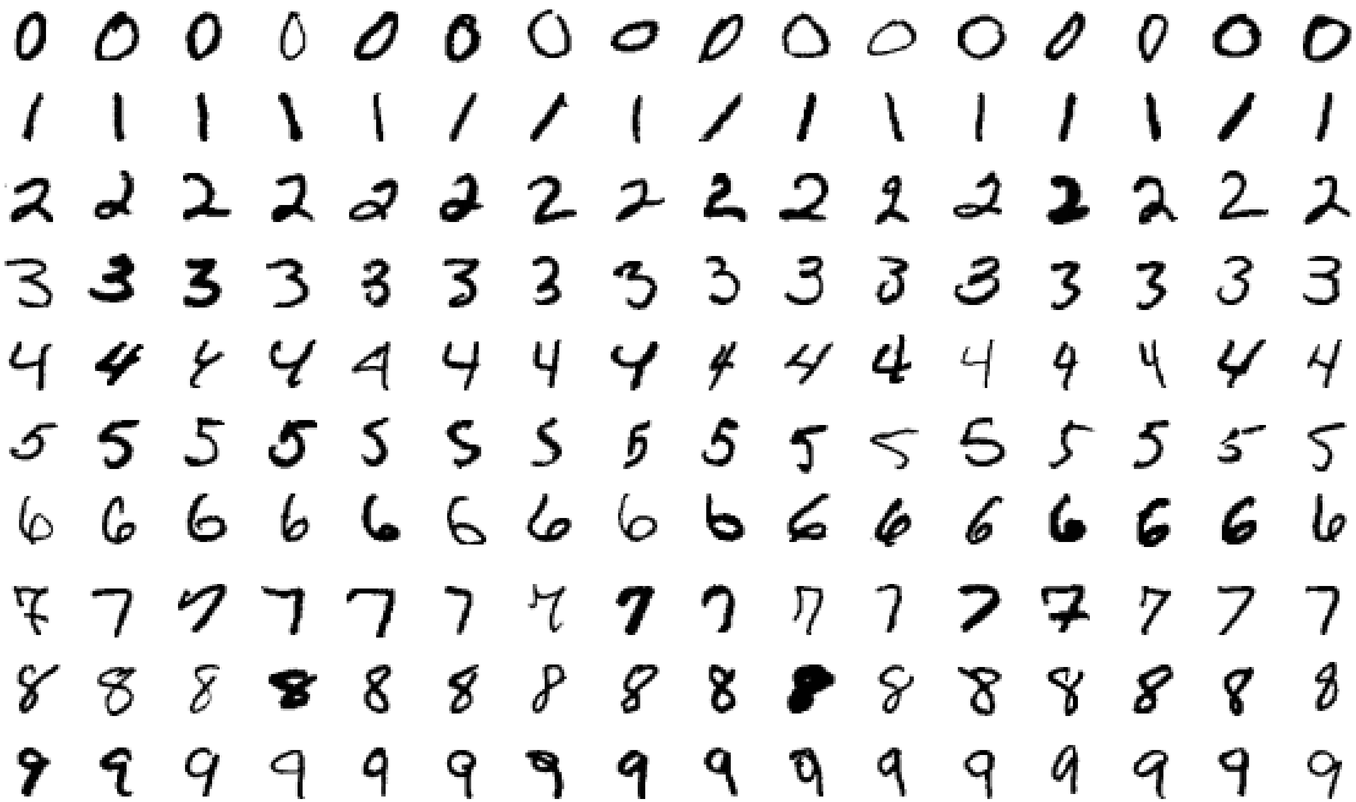
Comparison


MNIST Multilayer Neural Network + dropout
training cost
10^{-1}
10^{-2}
0
50
100
150
200
iterations over entire dataset
AdaGrad
RMSProp
SGDNesterov
AdaDelta
Adam

tf.keras.optimizers

- Adadelta
- Adagrad
- **Adam (2014)**
- Adamax (2014, section 7)
- Ftrl
- Nadam (nesterov + Adam)
- RMSprop
- SGD

(Ex 2) MNIST database



Homework

(9, 9) size

(7, 7) 49

(0, 16) green

(0, 24) red

plt.xlabel()

Tensorflow (softmax)

```
import tensorflow as tf
x=[-1,0,1.]
tf.nn.softmax(x)

import numpy as np
np.exp(x)/sum(np.exp(x))

tf.exp(x)/sum(tf.exp(x))

tf.exp(x)/tf.reduce_sum(tf.exp(x))
```

Tensorflow (sum,argmax,one_hot)

```
x=[[1,2,3],[4,5,6]]
x=np.array([[1,2,3],[4,5,6]])
np.sum(x)
np.sum(x,0)
np.sum(x,1)

tf.reduce_sum(x)
tf.reduce_sum(x,0)
tf.reduce_sum(x,1)
tf.reduce_sum(x,1).numpy()
```

```
tf.argmax(x)
tf.argmax(x,0)
tf.argmax(x,1)

a=[0,1,2]
depth=3
tf.one_hot(a,depth)

y=[5,1,7,9,0]
depth=10
tf.one_hot(y,depth)
```

Tensorflow (categorical crossentropy)

```
cce=tf.keras.losses.CategoricalCrossentropy()
loss=cce([1.,0.,0.],[.9,.05,.05])
loss=tf.keras.losses.categorical_crossentropy([1.,0.,0.],[.9,.05,.05])
```

<< binary crossentropy >>

(y=0) loss = -log(1-y_)

(y=1) loss = -log(y_)

loss = y(-log(y_))+(1-y)(-log(1-y_))

<< categorical crossentropy >>

(y=0) loss= 0

(y=1) loss= -log(y_)

loss = y(-log(y_))

sparse_categorical_crossentropy

```
y=[1,0,0]
y_=[0.9,0.05,0.05]
sum(y*(-np.log(y_)))

tf.reduce_sum(y * (-tf.math.log(y_)))

loss = tf.keras.losses.categorical_crossentropy(y, y_)

y = 0
loss = tf.keras.losses.sparse_categorical_crossentropy(y, y_)
```

(Ex 3) Boston Housing Price

BostonHousing 데이터 설명

[01]	CRIM	자치시(town) 별 1인당 범죄율
[02]	ZN	25,000 평방피트를 초과하는 거주지역의 비율
[03]	INDUS	비소매상업지역이 점유하고 있는 토지의 비율
[04]	CHAS	찰스강에 대한 더미변수(강의 경계에 위치한 경우는 1, 아니면 0)
[05]	NOX	10ppm 당 농축 일산화질소
[06]	RM	주택 1가구당 평균 방의 개수
[07]	AGE	1940년 이전에 건축된 소유주택의 비율
[08]	DIS	5개의 보스턴 직업센터까지의 접근성 지수
[09]	RAD	방사형 도로까지의 접근성 지수
[10]	TAX	10,000 달러 당 재산세율
[11]	PTRATIO	자치시(town)별 학생/교사 비율
[12]	B	1000(Bk-0.63)^2, 여기서 Bk는 자치시별 흑인의 비율을 말함.
[13]	LSTAT	모집단의 하위계층의 비율(%)
[14]	MEDV	본인 소유의 주택가격(중앙값) (단위: $1,000)

Neural Network for boston.csv

```
import pandas as pd
df = pd.read_csv('boston.csv')
import numpy as np
data = np.array(df)
x = data[:,:13]
y = data[:,13]
from keras.models import Sequential
from keras.layers import Dense
model = Sequential()
model.add(Dense(30, input_dim = 13, activation = 'relu'))
model.add(Dense(6, activation = 'relu'))
model.add(Dense(1))
model.compile(loss = 'mean_squared_error', optimizer = 'adam')
model.fit(x,y,epochs=500)
y_ = model.predict(x).flatten()
y_[:5], y[:5]
```