Machine Learning in Practice
#3: Neural Networks Basics
"*First Contact with TensorFlow*", Ch. 4

Sang-Hyun Yoon

Summer 2019

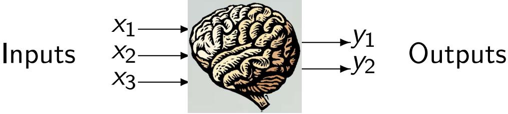
Popular Machine Learning Systems

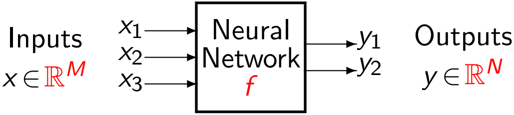
- Support vector machines (SVM)
- Neural networks
  - Convolutional neural network, deep belief network, recurrent NN 등을 통칭하는 deep learning 기법은 모두 이 부류
  - Machine learning의 가장 큰 application인 영상/음성 인식 등의 문제에서는 2010년대 이후로 SVM을 밀어내고 주류
- Cluster analysis
- Bayesian networks
- Decision tree learning
- k-nearest neighbor
- Probabilistic graph model, ...

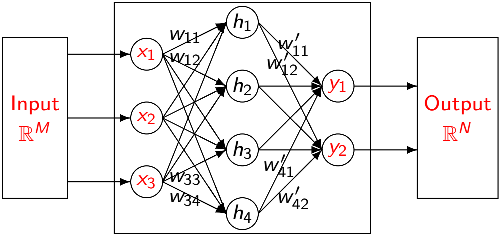
Outline

Neural Network

- An extremely simplified model of the brain and nervous system
- No more than an approximator of functions $f : \mathbb{R}^M \to \mathbb{R}^N$.

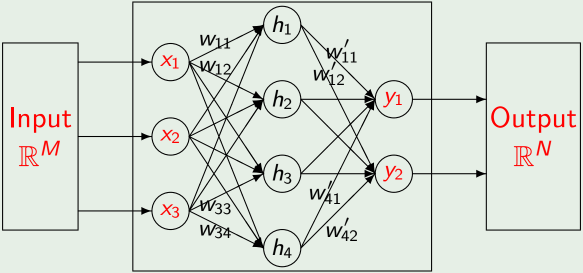






$$h_j = \sigma(\textstyle\sum_i w_{ij} x_i)$$
$$y_k = \sigma(\textstyle\sum_j w'_{jk} h_j)$$
where
$$\sigma(z) = 1/(1 + e^z)$$
$$f : \mathbf{w} \times X \to Y$$
$$f_{\mathbf{w}}(x) \triangleq f(\mathbf{w}, x)$$

Neural Networks



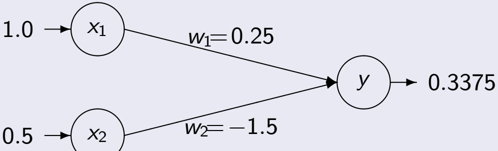
$$h_j = \sigma(\textstyle\sum_i w_{ij} x_i)$$
$$y_k = \sigma(\textstyle\sum_j w'_{jk} h_j)$$
where
$$\sigma(z) = 1/(1+e^z)$$

$$f_{\mathbf{w}} = (y_k)_k$$
$$\mathcal{H} \triangleq \{f_{\mathbf{w}} | \mathbf{w} = (w_{ij}, w'_{jk})\}$$

- $\mathcal{H}$: hypothesis space / representation / model architecture
  $$\mathcal{H} = \{ f_{\mathbf{w}} \,|\, \mathbf{w} \in \mathbb{R}^{\text{very large}} \}$$
- C : cost/evaluation/objective/scoring function
  $$C : \mathcal{H} \to \mathbb{R} ; \quad C(f) = \textstyle\sum_{(x,y)\in T} \|f(x) - y\|^2$$
- how to compute argmin : optimization/learning algorithm
  $$\mathbf{w} \Leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} C \ \left(\text{i.e. } w_{ij} \Leftarrow w_{ij} - \eta \frac{\partial C}{\partial w_{ij}} \text{ and } w'_{jk} \Leftarrow w'_{jk} - \eta \frac{\partial C}{\partial w'_{jk}} \right)$$
- Data acquisition type: supervised/reinforcement

Perceptron

- Unit of neural network $f : \mathbb{R}^N \rightarrow \mathbb{R}$
  - $f(x) = \sigma\left(\sum_i w_i x_i + b\right)$ where σ is any activation function (e.g. sigmoid function $\sigma(z) = 1/(1+e^z)$)
    - w: weight term / b: bias term
  - Majority of NNs use sigmoid function as σ, because it's derivative is very simple (i.e. $\sigma'(z) = \sigma(z)/(1-\sigma(z))$)
  - Deep NNs also use ReLU ($r(z) = \max\{0, z\}$)
    - optimization becomes difficult due to non-differentiability..
- What functions can perceptrons represent? $\{f : \mathbb{R}^N \rightarrow \mathbb{R} \mid f(x) = \sigma(\sum_i w_i x_i),\ N \geq 1, w \in \mathbb{R}^N\}$
  - Essentially, no more than a linear separator..



$$y = \sigma(w_1 x_1 + w_2 x_2)$$
where $\sigma(z) = 1/(1+e^z)$
$$y = \sigma(-0.5) = 0.3375$$

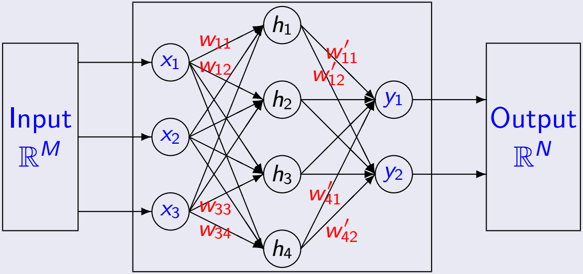
(Feedforward) Neural Network with 1 Hidden Layer

- $f : \mathbb{R}^M \to \mathbb{R}^N$ with adjustable **weights** $\mathbf{w} = (w_{ij}, w'_{jk})_{ijk}$:

$$f_{\mathbf{w}}(x) = (y_k)_k = \Big(\sigma\big(\textstyle\sum_j w'_{jk} h_j\big)\Big)_k = \Big(\sigma\Big(\textstyle\sum_j w'_{jk}\big(\sigma(\sum_i w_{ij} x_i)\big)\Big)\Big)_k$$

- What functions can they **represent**? $\{f_{\mathbf{w}}(\cdot) \mid \mathbf{w}\}$
  - Any continuous function $f : \mathbb{R}^M \to \mathbb{R}^N$
    - with a "sufficiently large" set of weights (i.e. parameters)

[1] For a vector $v = (v_1, v_2, \dots)$, $(v_1, v_2, \dots)$ and $(v_i)_i$ are interchangeably used.



$$h_j = \sigma\big(\textstyle\sum_i w_{ij} x_i\big)$$
$$y_k = \sigma\big(\textstyle\sum_j w'_{jk} h_j\big)$$

where

$$\sigma(z) = 1/(1 + e^z)$$

$$f : \mathbf{w} \times \mathbb{R}^M \to \mathbb{R}^N$$
$$f_{\mathbf{w}}(x) \triangleq f(\mathbf{w}, x)$$

How do neural networks adapt themselves to fit $f_{Simpson}$?

Regression-like numerical optimization

1. Example $T = \{($  , “Homer”$),($  , “Marge”$), \ldots\}$ is given.
2. Find $\mathbf{w}$ s.t. the cost function $C(T, \mathbf{w})$ is minimized.
   - e.g. $C(T, \mathbf{w}) = \sum_{(x,y)\in T} \|f_{\mathbf{w}}(x) - y\|^2$ (squared-error)

Q) How to minimize $C(T, \mathbf{w})$?

A) By search with $\nabla_{\mathbf{w}} C(T, \mathbf{w})$

Gradient-descent heuristic to search for $\text{argmin}_{\mathbf{w}}\, C(T, \mathbf{w})$

- $\mathbf{w} \Leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} C$ $\left(\text{i.e. } w_{ij} \Leftarrow w_{ij} - \eta \frac{\partial C}{\partial w_{ij}} \text{ and } w'_{jk} \Leftarrow w'_{jk} - \eta \frac{\partial C}{\partial w'_{jk}}\right)$
  - $\displaystyle \frac{\partial C}{\partial w'_{jk}} = \sum_{(x,y)\in T} (y_k - y_k^T) y_k (1 - y_k) h_j$
  - $\displaystyle \frac{\partial C}{\partial w_{ij}} = \sum_{(x,y)\in T} \sum_k (y_k - y_k^T) y_k (1 - y_k) w'_{jk} h_j (1 - h_j) x_i$
- Called the back-propagation learning

How do neural networks adapt themselves to fit $f_{Simpson}$?

Regression-like numerical optimization

1. Example $T = \{($  , "Homer"$),($  , "Marge"$), \ldots\}$ is given.
2. Find $\mathbf{w}$ s.t. the cost function $C(T, \mathbf{w})$ is minimized.
   - e.g. $C(T, \mathbf{w}) = \sum_{(x,y)\in T} \|f_{\mathbf{w}}(x) - y\|^2$ (squared-error)

Q) How to minimize $C(T, \mathbf{w})$?

A) By search with $\nabla_{\mathbf{w}} C(T, \mathbf{w})$

Encoding/Decoding problem...

Recall $f_{\mathbf{w}} : \mathbb{R}^M \to \mathbb{R}^N$.

- How to encode Homer's photos into real numbers in $\mathbb{R}^M$?
- How to decode the output of $f_{\mathbf{w}} \in \mathbb{R}^N$ to a name?

In general, no systematic way...

Variants

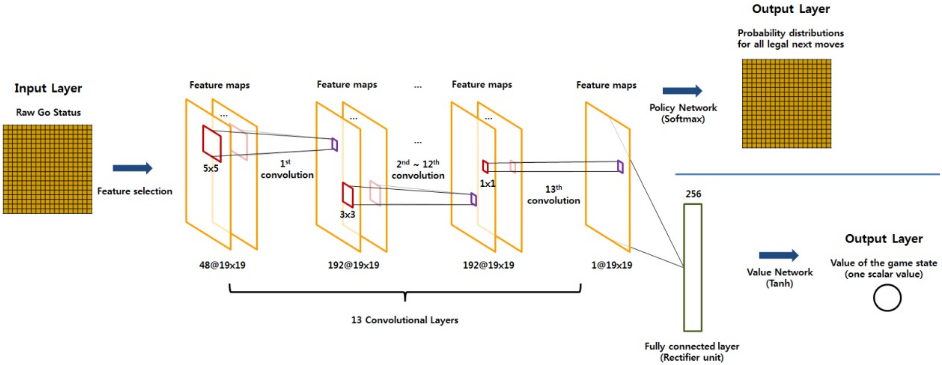
- Hidden layer의 층수가 0일 수도 있고 매우 클 수도 있고
  - 오늘 예제: 0
  - ALPHAGO Ver 18: $\geq 2 \times 13$
- 인접한 층간 노드를 모두 연결 vs. 일부만 연결
- 재귀적 연결 여부 (feedforward vs. recurrent)
- Activation 함수 (e.g. sigmoid, hyperbolic, ReLU, softmax, maxpooling)
- Learning 휴리스틱 (e.g. stochastic, gradient-descent, mini-batch, drop-out, AdaGrad)

TENSORFLOW는 이들 모두를 간단하게 구현할 수 있게 해줌

Variants: Example

CNN(Convolutional NN)의 경우

- hidden layer가 매우 많고
- 층간 연결방식이 convolution/pooling을 번갈아가면서 사용하다 마지막 층은 full connection
- 각 층마다 사용하는 activation 함수가 다름
  - ReLU, maxpool, softmax, hyperbolic, sigmoid 등



Recall: Foundational Issues

Consider a machine learning system for a problem $f : X \to Y$:

- $\mathcal{H}$: hypothesis space / model architecture / representation
- optimization/learning algorithm to find $\operatorname{argmin}_{\mathcal{A} \in \mathcal{H}} C(\mathcal{A})$

Representability of Neural Networks (related to $\mathcal{H}$)

Given any (training data) $T \subseteq X \times Y$, does there exist $\mathcal{A}_T \in \mathcal{H}$ s.t. $\mathcal{A}_T(x) \cong y$ for all $(x, y) \in T$?

- MLF with one hidden layer: any continuous function
- MLF with two hidden layers: any function

Learnability of NNs (related optimization/learning algorithm)

If there exists such $\mathcal{A}_T \in \mathcal{H}$, can $\mathcal{A}_T$ be computed (in poly time)?

- NP-hard in its full generality. Approximation at its best.

NN의 표현력은 뛰어나나 learnability 등의 문제로 사장되었다
convolutional NN 등이 이를 극복하여 부활하게 됨

Limitations

- Essentially black box; No intuitive explanation for causality
  - Because they do not have clear semantics.
  - What can be learned are operational parameters, not general, abstract domain knowledge.
  - c.f. Deductive reasoning in other sub-disciplinaries of AI
  - Will you follow medical prescriptions made by NNs?
- No theoretical results on minimum necessary amount of weights, hidden nodes, training data, ...
  - only by trial-and-error
- No theoretically well-founded way to assess the quality
  - c.f. confidence level in statistical methods
- Neural Networks $\subsetneq$ Turing Machines
  - Church-Turing thesis

Outline

MNIST Dataset (for supervised learning)

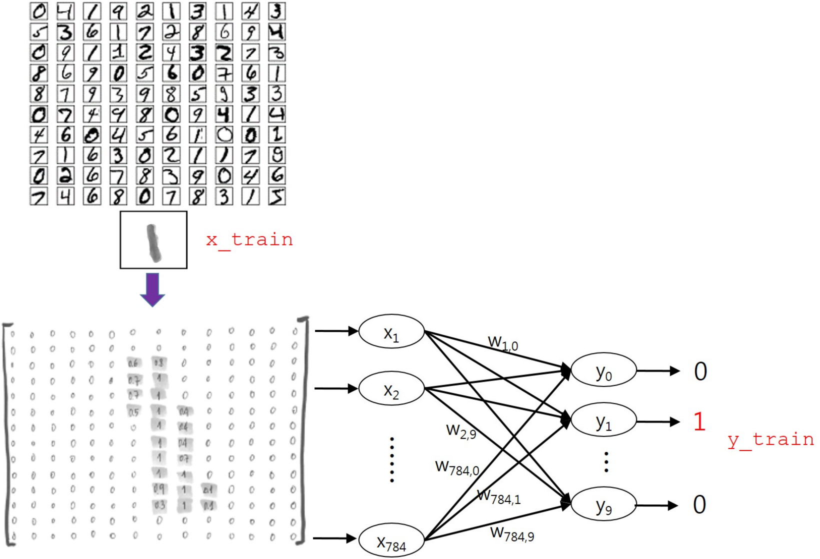
- 손글씨 숫자 흑백 이미지. 각 이미지는 28×28 픽셀
- 훈련용 60,000개 및 테스트용 10,000개

```
mnist = tensorflow.keras.datasets.mnist.load_data()
(x_train, y_train), (x_test, y_test) = mnist
```

- x/y_train은 훈련데이터, x/y_test는 테스트데이터
- x_train/test는 입력 이미지 (입력)
- y_train/test는 출력 숫자 (supervised learning에 필요)
- show_mnist_data.py로 이미지/숫자 확인

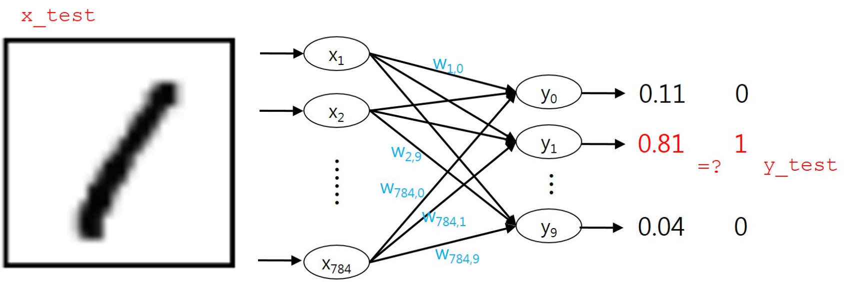
Neural Network (without Hidden Layers) for Digit Classification

학습 단계 (off-line): `Train_NN.py`



Neural Network (without Hidden Layers) for Digit Classification

학습된 NN를 이용 (on-line): `Test_NN.py`



TENSORFLOW Code: Summary

- `Train_NN.py`: $\boxed{\text{NN(x_train)} \approx \text{y_train}}$인 NN을 구성
  - 덤으로 $\boxed{\text{유사도(NN(x_train), y_train)}}$도 측정
  - 그리고 계산된 NN을 파일에 저장

- `Test_NN.py`: `Train_NN.py`에서 구성/저장한 NN을 읽은 후 $\boxed{\text{유사도(NN(x_test), y_test)}}$를 측정
  - 유사도(NN(x_train), y_train)보다 크게 낮으면 overfitting

- `NN_comp_graph.py`: `Train_NN.py`과 `Test_NN.py`에서 공통으로 사용하는 computation graph를 구성
  - graph의 입력 단자: `x_train/test`, `y_train/test` (placeholder 형태)
  - graph의 출력 단자: `train`, `accuracy`
    - `train`: NN(x_train) ≈ y_train인 NN을 구성하라는(즉, cost 함수 최적화) 나타내는 노드로 `Train_NN.py`에서만 사용
    - `accuracy`: 유사도(NN(x), y)

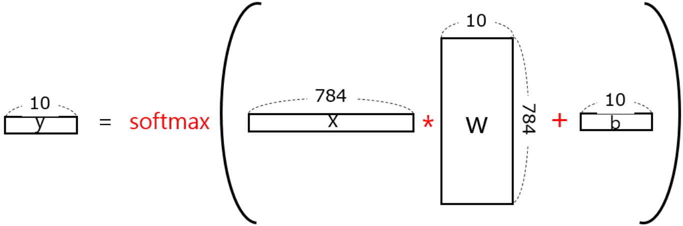
Single Layer Neural Network for Digit Classification

- $f : \mathbb{R}^{784} \to \mathbb{R}^{10}$;

  $$f(x_1, \cdots, x_{784}) = \text{softmax}\left(\sum_{i=1}^{784} w_{i0}x_i + b_0, \cdots, \sum_{i=1}^{784} w_{i9}x_i + b_9\right)$$

  where $\text{softmax}(z_0, \cdots, z_9) = \left(\frac{\exp(z_0)}{\sum_{j=0}^{9}\exp(z_j)}, \cdots, \frac{\exp(z_9)}{\sum_{j=0}^{9}\exp(z_j)}\right)$

  - softmax : $\mathbb{R}^n \to \mathbb{R}^n$의 출력 벡터의 합이 1임에 주목




```
x = tf.placeholder(tf.float32, [1,784])    # only 1 figure
w = tf.Variable(... [784,10] ...)          # weights
b = tf.Variable(... [1,10] ...)            # biases
y = tf.nn.softmax(tf.matmul(x,w) + b)      # why not w*x?
```

TensorFlow Code: Cost Function (pp.106-109)

$$f(x_1,\cdots,x_{784}) = \text{softmax}\left(\sum_{i=1}^{784} w_{i0}x_i + b_0,\cdots,\sum_{i=1}^{784} w_{i9}x_i + b_9\right)$$

```
x = tf.placeholder(tf.float32, [1,784])  # only 1 figure
y = tf.nn.softmax(tf.matmul(x,W) + b)
```

```
x_train = tf.placeholder(tf.float32, [None,784]) # many figures
y_train = tf.placeholder(tf.float32, [None,10])
w = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))  # shape: [10]

m = tf.matmul(x_train,w)         # shape: [?,10]
z = m + b                        # shape: [?,10], broadcast
y = tf.nn.softmax(z)             # shape: [?,10]

cross_entropy = -tf.reduce_sum(y_train*tf.log(y))  # scalar
```

Cross entropy $-\sum_{j=0}^{9} y_j^{\text{train}} \cdot \log y_j$ is minimized when $y^{\text{train}} = y$

TENSORFLOW Code: Training (pp.110-111)

- Computation graph의 입력을 placeholder로 설정함에 유의
  - regression code: 처음부터 list에 값을 담아서 graph 구성
  - 한번에 모든 data를 넣고 optimize하면 메모리/시간 오버헤드 막대해서 batch 크기를 100씩 잘라서 optimize
  - Placeholder를 쓰지 않으면 각 입력마다 graph 만들어줘야 함
- GradientDescentOptimizer외에도 다양한 optimizer들
  - 각 cost 함수마다 최적의 optimizer가 틀림
  - 시행착오를 통해 최적의 optimizer/step을 정할 수 밖에 없음

```
x_train = tf.placeholder(tf.float32, [None,784])
y_train = tf.placeholder(tf.float32, [None,10])
...
train = tf.train.GradientDescentOptimizer(0.01). \
                                  minimize(cross_entropy)
for i in range(1000):
    batch_x, batch_y = get_batch(x_train, y_train, 100, i)
    sess.run(train, feed_dict = {x_train:batch_x, y_train:batch_y
```

TensorFlow Code: Saving/Restoring Variables

- Optimizer가 완료후 variables의 값을 파일로 저장 가능
  - tf.train.Saver().save
- 저장할 변수를 선택하려면 scope 사용 (DQN에서 다룸)
- 다양한 hyperparameter로 실험하면서 다른 파일에 저장

```
w = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))
...
sess.run(train, ...)

tf.train.Saver().save(sess, "./model/mnist_model.ckpt")
```

- 저장된 변수값을 파일에서 복구할 수 있음
  - tf.train.Save().restore

```
tf.train.Saver().restore(sess, "./model/mnist_model.ckpt")
```

TENSORFLOW Code: Testing (pp.110-111)

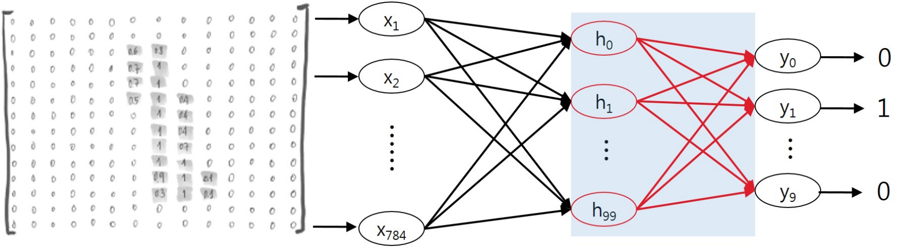
- softmax로 계산한 10개 확률값 중 최대값의 index를 digit로
- accuracy는 앞 페이지의 y부터 edge를 그어서 새로 만든 computation graph의 dest 노드
- cross_entropy를 최소화하는 trained 변수 w,b를 파일에서 읽어서 accuracy를 계산

```
x_train = tf.placeholder(tf.float32, [None,784])
y_train = tf.placeholder(tf.float32, [None,10])
w = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))
...
y = tf.nn.softmax(z)   # shape: [?,10]

correct = tf.equal(tf.argmax(y,1), tf.argmax(y_train,1))
accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

tf.train.Saver().restore(sess, "./model/mnist_model.ckpt")
sess.run(accuracy, feed_dict={x_train:x_test, y_train:y_test})
```

NN with 1 Hidden Layers

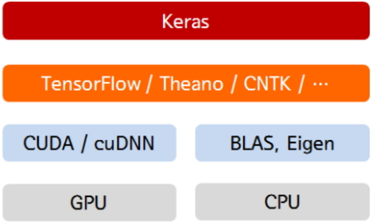


- Hidden layer 없는 앞의 NN의 정확도는 90 ∼ 91%
- Hidden layer를 하나 추가하면 어떨까?
  - `NN_comp_graph.py`의 `NN_comp_graph_with_hidden_layer`와 같이 layer를 하나 더 넣으면 됨
- 실험해보면 정확도가 오히려 크게 떨어짐..
- 위와 같은 연결 방식의 NN은 vanishing gradient 등의 문제로 learnability가 크게 떨어져서 fitting이 잘 안됨
- 이 문제는 다음 시간의 CNN(convolutional NN)으로 극복

Outline

1. Neural Networks
2. Example: Digit Classifier
3. KERAS Library

Keras Library



- Keras는 TensorFlow의 high-level wrapper
  - 즉, Keras 코드는 TensorFlow 코드를 호출
  - Theano 등의 다른 라이브러리도 wrapping
- TensorFlow를 바로 쓰는 것보다 간단하게 NN 구성가능
  - 반대 급부로 표현력이 약간 떨어질 수 있으나, NN을 구성하기 위해 필요한 기능은 대부분 제공
- 초보자들은 TensorFlow에 숙달 후 Keras 사용 추천
  - Keras만 사용할 경우 NN 내부구조를 잘 모르게 될 수도

KERAS Implementation of Digit Classifier (NN with 1 Hidden Layer)

https://keras.io/getting-started/sequential-model-guide

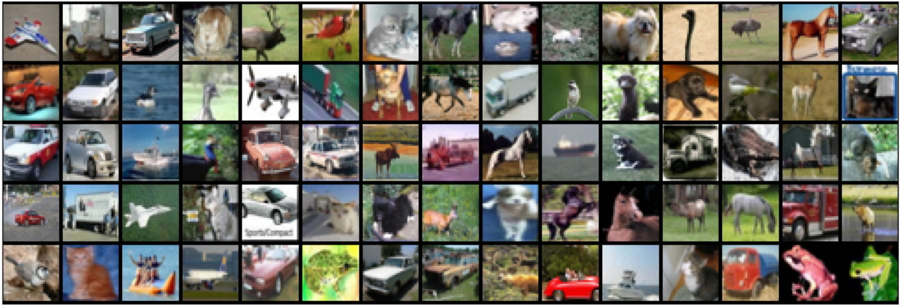
```
model = Sequential()
model.add(Dense(units=size_hidden, input_dim=size_in,
                                     activation='sigmoid'))
model.add(Dense(units=size_out, input_dim=size_hidden,
                                     activation='softmax'))
model.compile(loss="categorical_crossentropy",
                     metrics=["accuracy"], optimizer="sgd")

model.fit(x_train, y_train, batch_size=100, epochs=15)
accuracy = model.evaluate(x_train, y_train, batch_size=100)
y = model.predict_classes(x_test[i])

model.save("./model/mnist_model.h5")
```

- Train_NN_keras.py, Test_NN_keras.py 참조
- Computation graph와 NN의 weight/bias를 모두 저장/복구

Homework: Keras Implementation of Image Classifier (1/2)



- cifar10 dataset: 10 종류의 물체들에 대한 그림 데이터
- 비행기, 자동차, 새, 고양이, 사슴, 개, 개구리, 말, 배, 트럭을 숫자 0∼9로 labeling
- show_cifar_data.py로 이미지/숫자 확인

Homework: Keras Implementation of Image Classifier (2/2)

- `Train_NN_cifar.py`의 `NN_model` 함수에서 neural network model을 만들어서 리턴하도록 구현해야 함
- `mnist_Keras` 폴더의 `Train_NN_Keras.py`의 `NN_mode_with_hidden_layer`함수를 그대로 따라하면 됨
  - Digit classifier의 Keras 버젼으로 hidden layer가 하나 있음
- 다음가 같이 NN 구조를 좀 더 키워서 구현하도록 한다:
  - 데이터 텐서의 shape가 $(28, 28) \rightarrow (32, 32, 3)$로 커졌으므로 입력 neuron의 갯수는 $32 \cdot 32 \cdot 3 = 3072$
  - Hidden layer를 하나 더 추가한다. 첫번째/두번째 hidden layer의 neuron 갯수는 각각 3000/2000
  - Activation 함수는 hidden layer는 모두 sigmoid로, 마지막 layer는 softmax로
- `Test_NN_cifar.py`로 테스트하면 30%의 정확도..
  - 다음 시간에 CNN(convolutional NN)으로 정확도를 올려보자
- model 파일은 생략하고 `.py`만 `cs3.ksa@gmail.com`로 제출