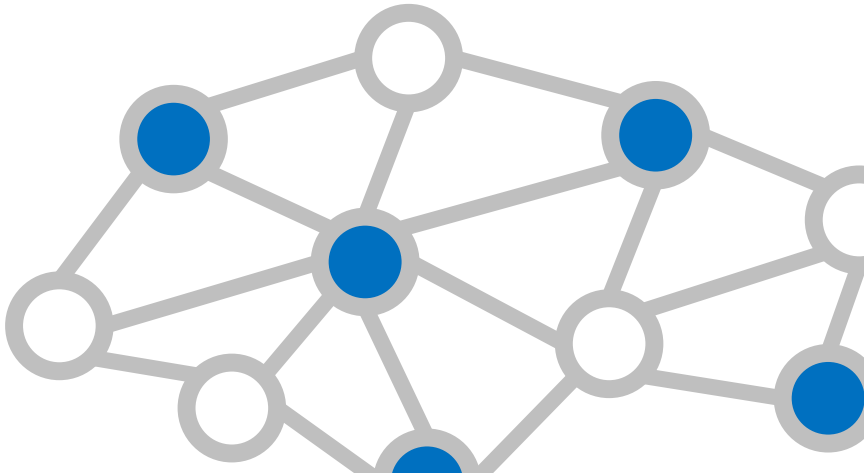
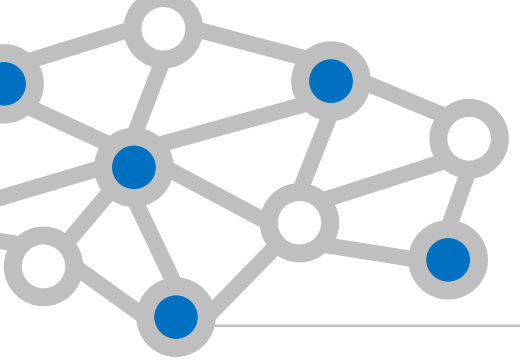
Tensorflow 실습

Kim Dae Sik

MIPAL aboratory
machine intelligence
&pattern analysis



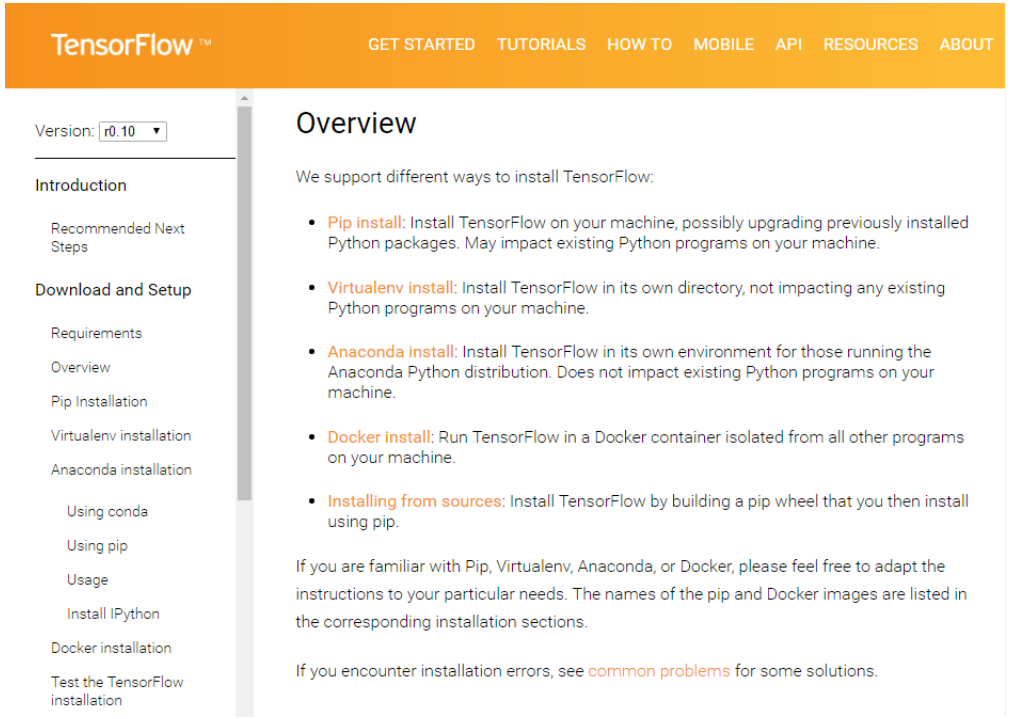


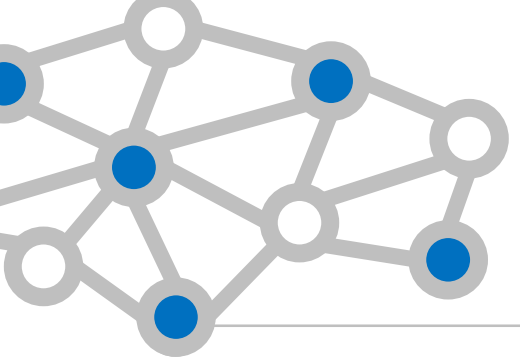
목차

- Tensorflow 설치
- Tensorflow의 구성 및 특징
- MNIST 예제

Tensorflow 설치

- **설치방식**
  - Pip install - 시스템에 가상화 없이 설치
  - Virtualenv Install - 정해진 디렉토리내에 설치
  - Anaconda install
  - Docker- 가상화 설치 (윈도우 지원)
  - Installing from sources
- **설치 요구 사항**
  - Python 2.7 혹은 Python 3.3+
  - GPU 버전의 Tensorflow는 CUDA 7.5 와 cuDNN v4 버전만 지원
  - CUDA toolkit >= 7.0 과 cuDNN 6.5(v2), 7.0(v3), v5는 오직 소스를 컴파일 하는 방식으로 설치 가능





Tensorflow의 특징

DistBelief - 1세대 기계 학습 시스템

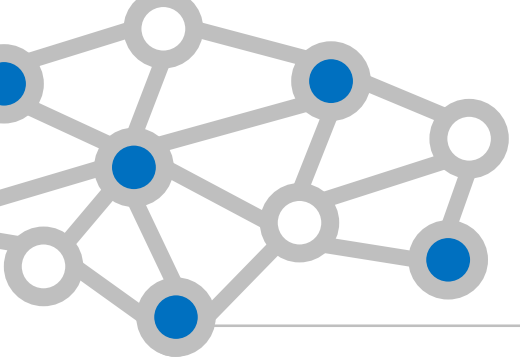
- Google Brain 팀이 2011년부터 시작한 프로젝트
- Deep Learning 라이브러리
- Google의 50개의 팀에서 서비스에 활용
- Scalability, 기본적인 모델의 학습이 용이
- Not flexible, 유연한 구조 설계의 한계

Tensorflow 특징

Tensorflow - 2세대 기계 학습 시스템

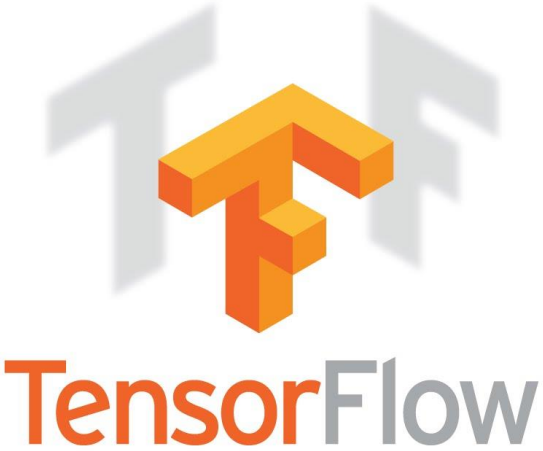
- 2015년 11월 9일, Tensorflow 발표
- Scalability, flexibility, fast, robust
- Application level의 오픈소스 기계학습 라이브러리 (Apache 2.0 License)
- Jeff Dean, Jeffrey Hinton 그리고 여러 오픈 소스 기계학습 라이브러리의 참여자들!

MIPAL aboratory
machine intelligence &pattern analysis



Tensorflow 특징

- **표현의 용이성**: 기계 학습과 관련된 여러 아이디어와 알고리즘을 쉽게 표현
- **확장성** : 빠른 실험이 가능
- **이식성**: 우분투, 맥OS와 안드로이드 등 여러 플랫폼에 이식 가능
- **재생산성**: 공유하기 쉽고 연구의 재생산 가능
- **상용화 용이성**: 연구에서 서비스로 상용화가 가능
- **Auto-Differentiation :** 자동 유도-미분
- **Language Options :** Python 기본, C++

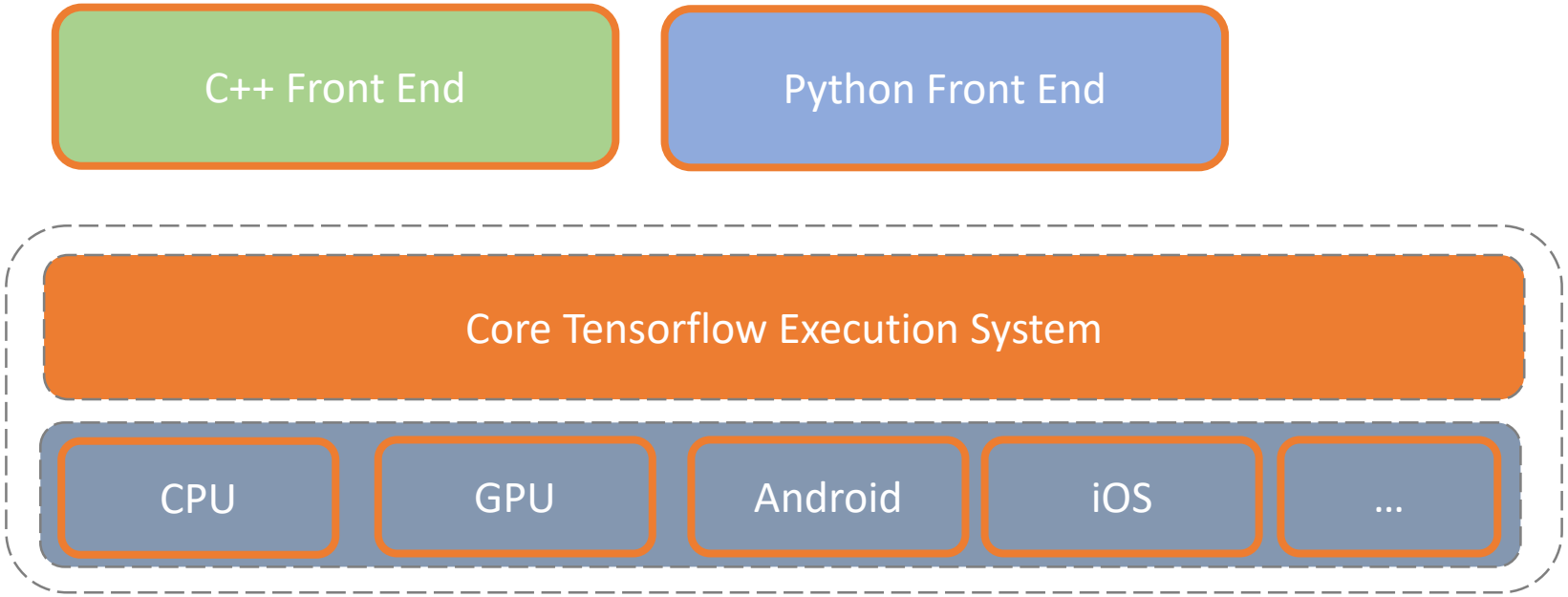


Tensorflow 특징

- **표현의 용이성**: 기계 학습과 관련된 여러 아이디어와 알고리즘을 쉽게 표현

  수준 라이브러리 ~ 저수준 함수까지 Python 또는 C/C++를 이용하여 구성 가능

  - Core in C++
  - 여러가지 Front Ends를 지원함으로서 high-level에서 알고리즘을 설계할 수 있다.

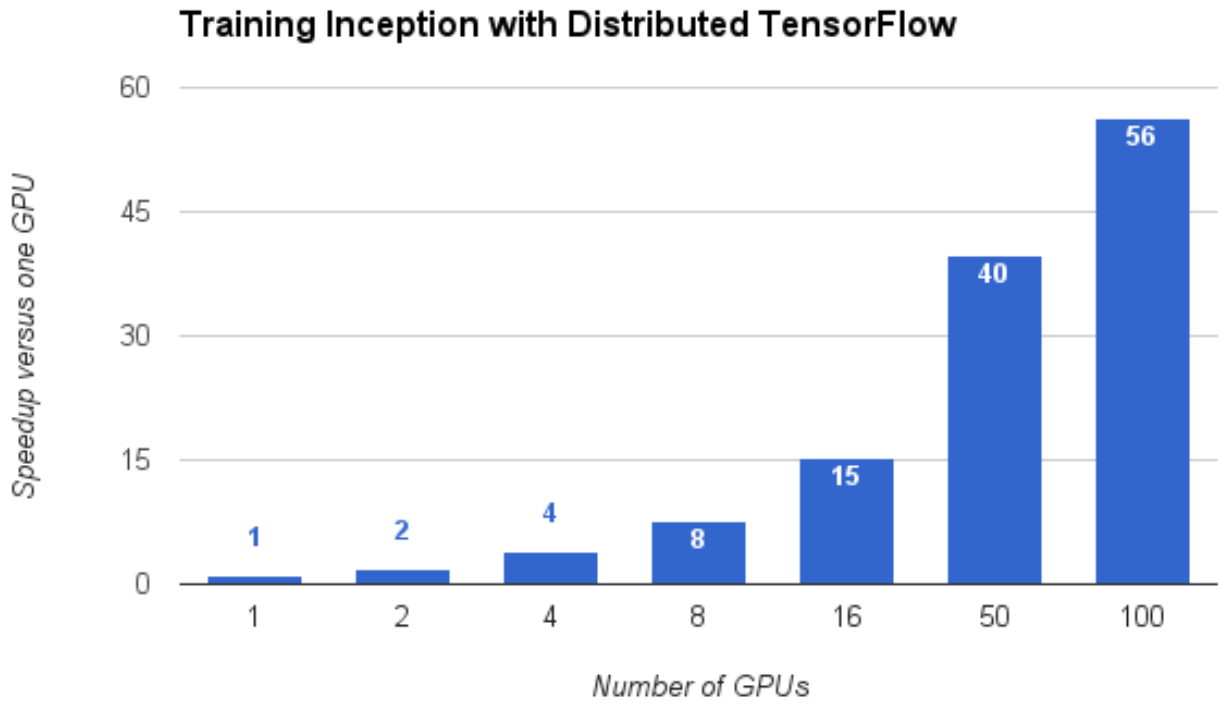


Tensorflow 특징

- **확장성** : 빠른 실험이 가능

  최대한 CPU core, GPU 사용

  - Model Parallelism
  - Data Parallelism
  - Distributed Training



Tensorflow 특징

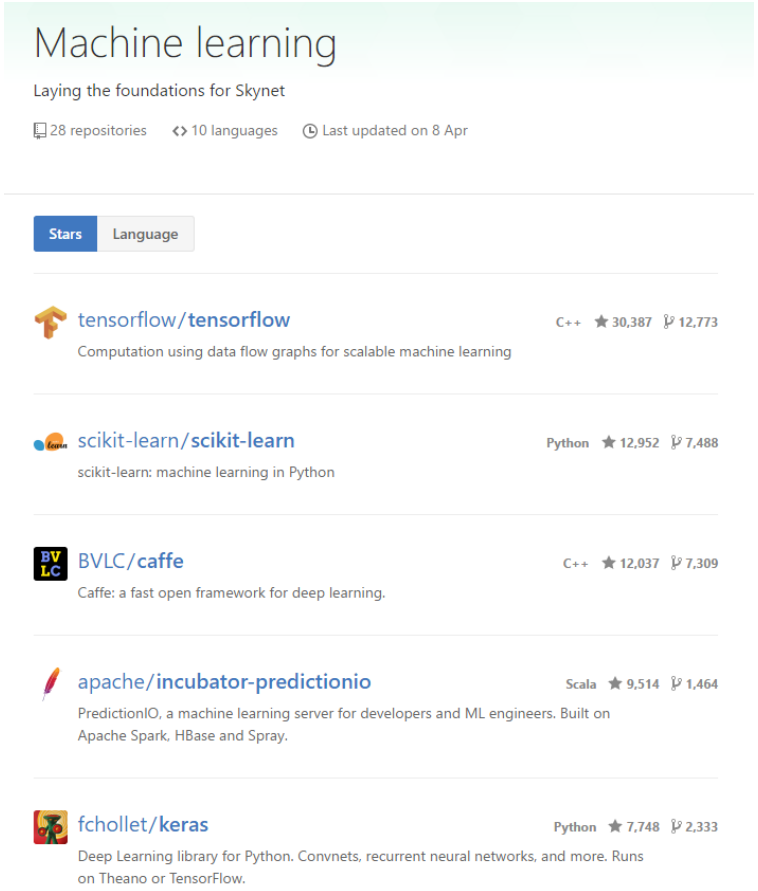
- **이식성**: 작성한 코드의 수정없이, 우분투, 맥OS와 안드로이드 등 여러 플랫폼으로 이식 가능
  - Ubuntu/Linux
  - Mac OS X
  - Android/iOS
  - Raspberry Pi/Custom Hardware
  - Server/cluster

Tensorflow 특징

- **재생산성**: 공유하기 쉽고 연구의 재생산 가능
  - Open Source – 비교적 유연한 Apache 2.0

    오픈소스 라이선스
  - 연구 중인 논문들을 비교적 쉽게 구현할 수 있다.
  - 점점 많은 사람들이 현재 연구중인 코드들을

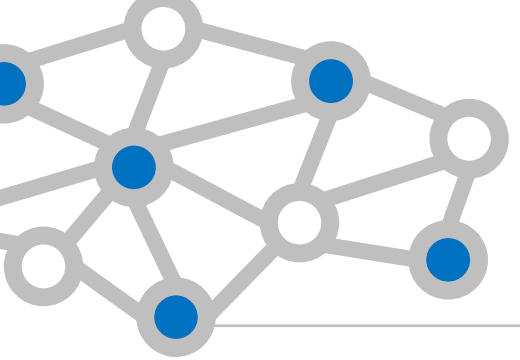
    온라인 저장소에 올리는 추세
  - 다큐멘테이션 잘 되어있고 튜토리얼이 많다!



MIPAL aboratory
machine intelligence &pattern analysis

Tensorflow 특징

- **상용화 용이성**: 연구에서 서비스로 상용화가 가능
  - 현재 50여개 이상의 구글 서비스에서 Tensorflow로 전환 하였다.



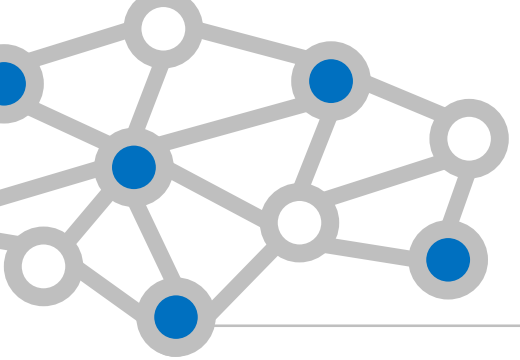
Tensorflow의 기본 구조

Tensor는 무엇인가?

- 데이터를 효과적으로 다루기 위한 자료구조 형태이다.
- 다차원의 array 혹은 list라고 생각하면 된다.
- Numpy의 확장이라고 생각하면 쉽다.

Flow는 무엇인가?

- Flow는 데이터의 흐름을 이야기 한다.
- 하나의 Graph라고 생각할 수 있다.
- 모든 계산을 각각의 연산을 잘게 쪼개어 하나의 Graph로 연결한 것이라고 생각하면 된다.



Tensorflow의 기본 구조

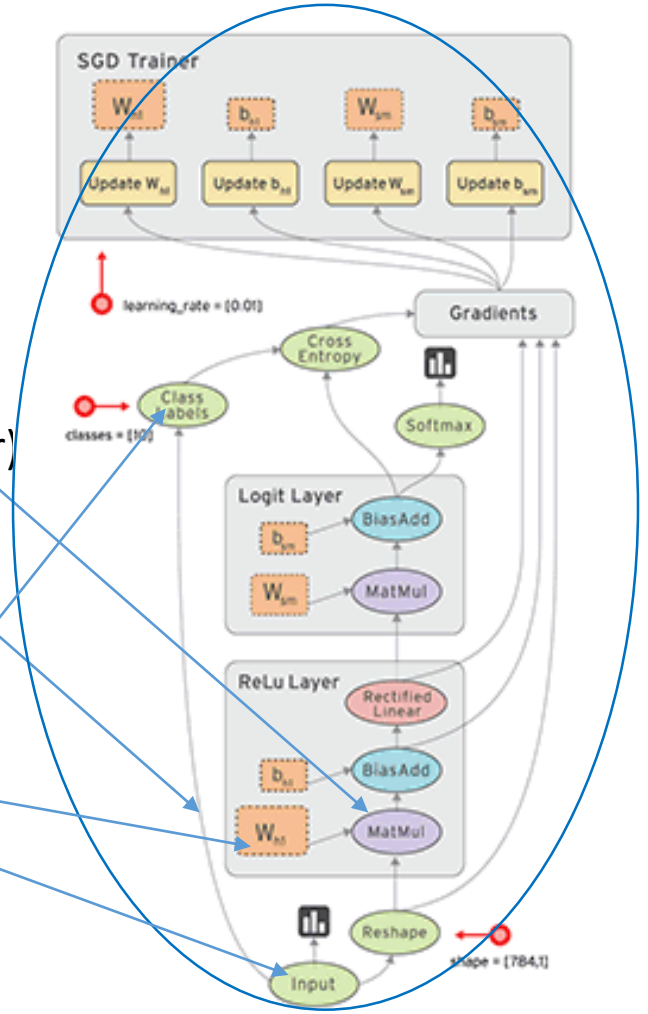
Tensorflow의 개요

- Tensorflow는 computation을 그래프로 나타내는 시스템이다.
- 노드는 op (operation) 라고 불린다. op는 1개 이상의 텐서를 받고, 1개 이상의 텐서를 생산한다.
- 텐서는 다차원의 어레이 (multi-dimensional array) 다. 예를 들어, 이미지의 mini-batch를 [batch, height, width, channels]의 4D floating point array로 표현할 수 있다.
- TensorFlow에서 그래프는 계산을 기술한 것입니다. 어떤 것이든 계산하기 위해서는 반드시 그래프를 Session에 올려야 합니다.
- Session은 CPU나 GPU 같은 Devices에 그래프 연산을 올린 뒤 연산을 실행할 수 있는 메소드를 제공합니다. 이 메소드는 연산에 의해 생성된 텐서를 반환하는데, 이 텐서의 형태는 파이썬에서는 numpy의 ndarray 객체, C와 C++에서는 tensorflow::Tensor 개체입니다.

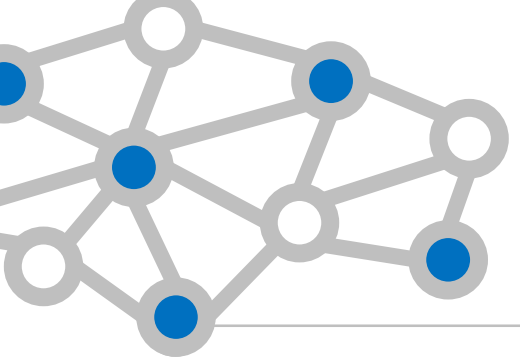
Tensorflow의 기본 구조

Tensorflow의 개요

- tensors로 data를 표현
- 동작을 정의한 것은 operation 이다.
- operation 정의를 포함한 것이 node이다.
- node와 node를 연결하는 것이 edge이다.(Tensor)
- graph로 computation을 표현한다. (construction)
- Sessions 안의 graph를 실행. (Execution)
- Variables로 상태 관리.
- feeds와 fetches로 데이터를 넣거나 빼기



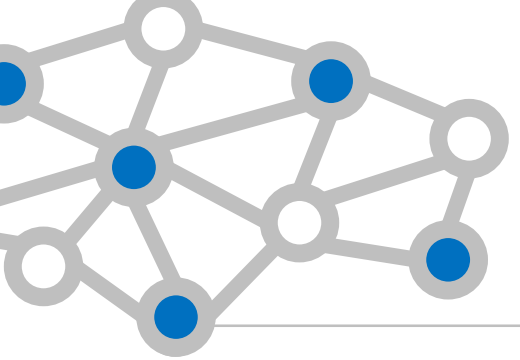
MIPAL aboratory
machine intelligence &pattern analysis



Tensorflow의 기본 구조

Computation graph

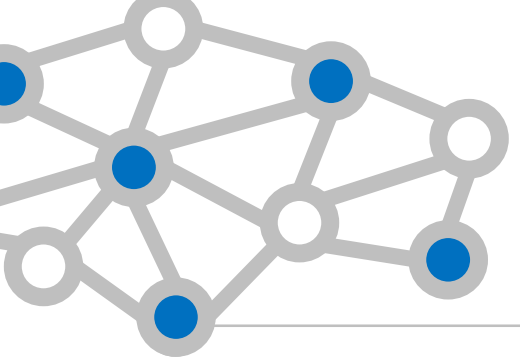
- TensorFlow 프로그램은 크게 두 가지 단계로 구성되어 있다. 구성(construction) 단계에서는 그래프를 조립(assemble)하고, 실행(execution) 단계에서는 세션을 이용해 그래프의 연산을 실행한다.
- 예를 들어, 구성 단계에서 신경망을 나타내고 훈련할 수 있는 그래프를 생성하고, 실행 단계에서 반복적으로 그래프의 훈련 연산을 실행하는 식의 진행은 흔하다.
- TensorFlow는 C, C++, 파이썬 프로그램에서 사용할 수 있다. 아직까지는 파이썬 라이브러리를 사용하는 편이 그래프를 조립하기에 훨씬 용이하다. C나 C++ 라이브러리가 제공하지 않는 방대한 도움 함수를 사용할 수 있기 때문이다.
- 세션 라이브러리는 세 언어에서 같은 수준의 기능을 제공한다.



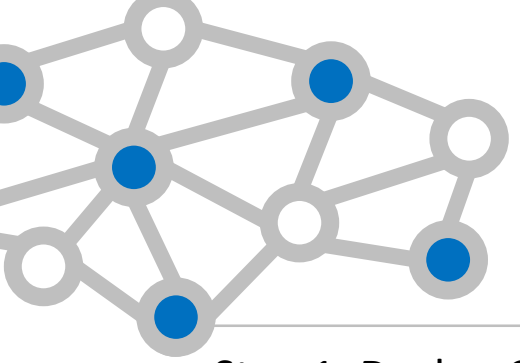
Tensorflow의 기본 구조

Building Graph

- graph를 만들기 위해서, “Constant”와 같이 어떤 입력도 필요로 하지 않는 ops로 시작하여, 그 출력을 계산을 위한 다른 ops로 넘긴다. Python라이브러리의 ops 생성자는, 생성된 ops의 출력을 의미하는 객체를 리턴한다. 그래서 리턴된 것을 다른 ops 생성자의 입력으로 넘길 수 있다.
- TensorFlow Python 라이브러리는 ops생성자가 node를 추가할 수 있는 default graph를 가지고 있으며, 많은 응용에서 충분히 사용할 수 있다.
- 다음 파이선 코드를 보자. default graph는 3개의 노드를 갖게 된다. 2개는 constant() op이고 1개는 matmul() op이다. 실제로 행렬 곱하기를 수행하여 결과를 얻기 위해서는 graph를 session 안에 띄워야(launch) 한다.

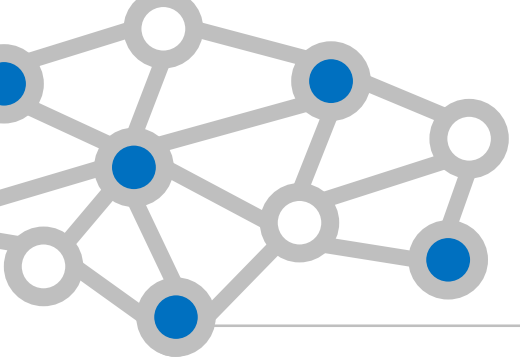


Tensorflow 기본 예제



Tensorflow 예제

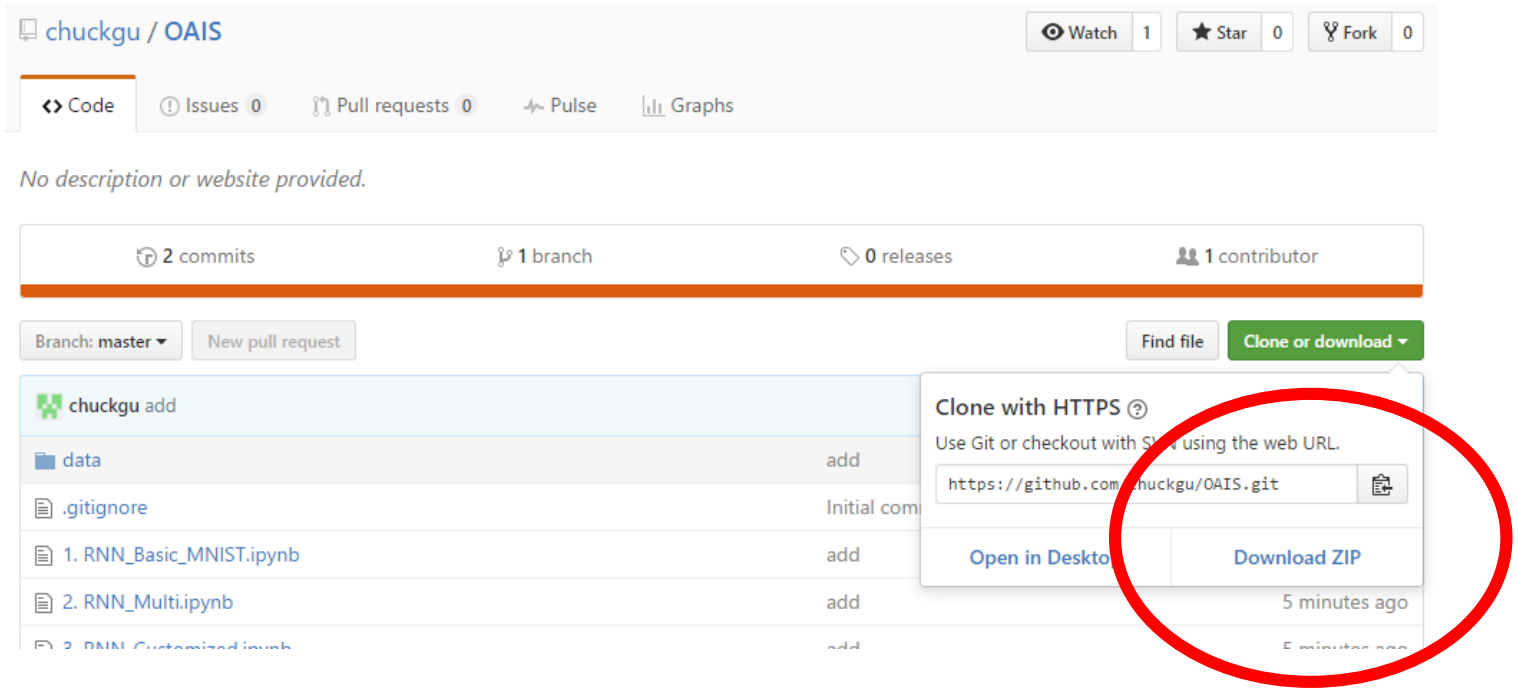
Step 1: Docker Quickstart Terminal을 연다
Step 2: docker-machine ls로 상태를 확인한다.
Step 3: docker run –it –p 8888:8888 b.gcr.io/tensorflow/tensorflow:latest-devel 를 실행한다
Step 4: 웹 브라우저를 열고 자신의 상태란에서 확인한 ip:8888을 넣고 실행한다. (예: 192.169.99.101:8888) , Ubuntu/Mac OS X의 경우 cmd창에서 jupyter notebook을 실행한다.



MNIST 학습 예제

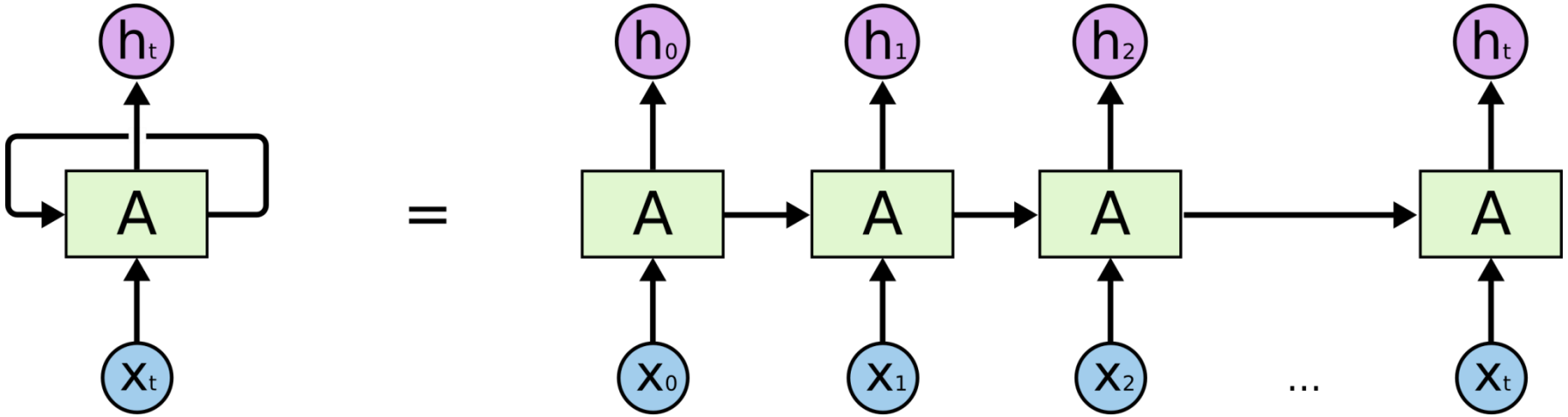
Code Download

- https://github.com/chuckgu/OAIS
- Git clone https://github.com/chuckgu/OAIS.git

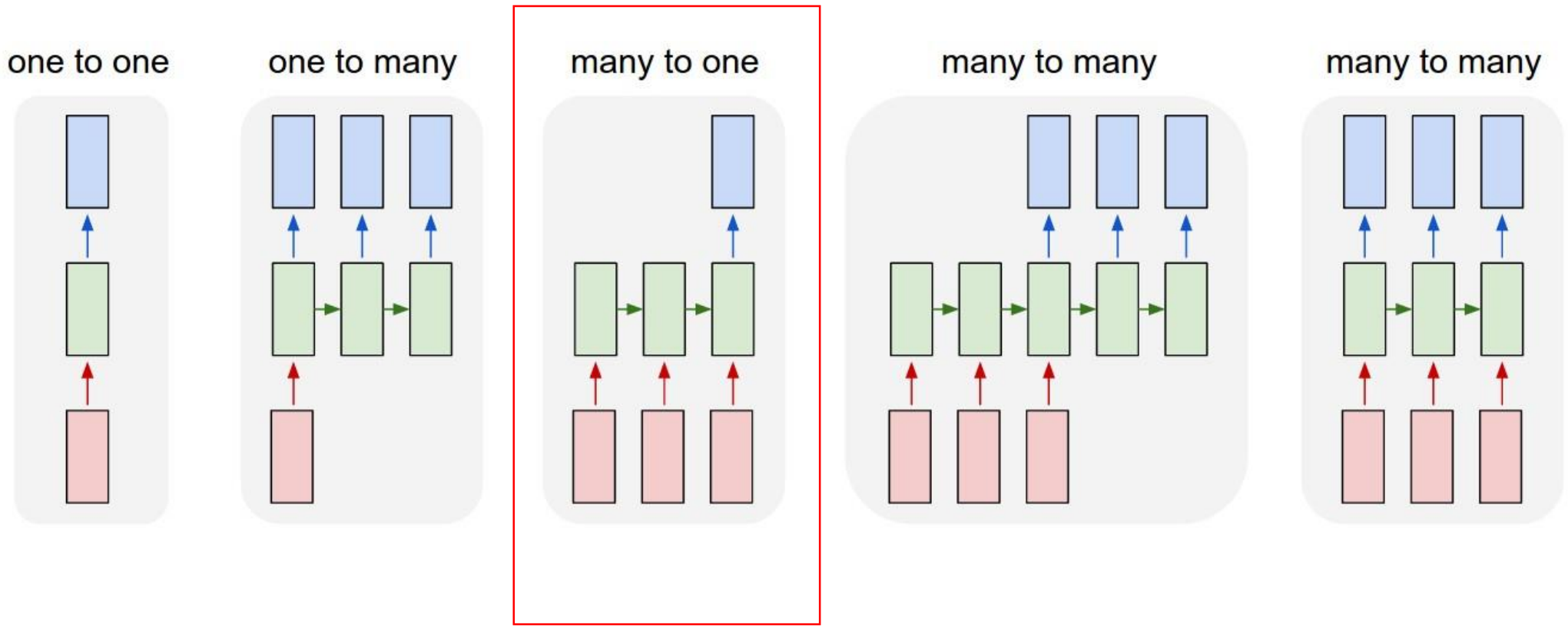


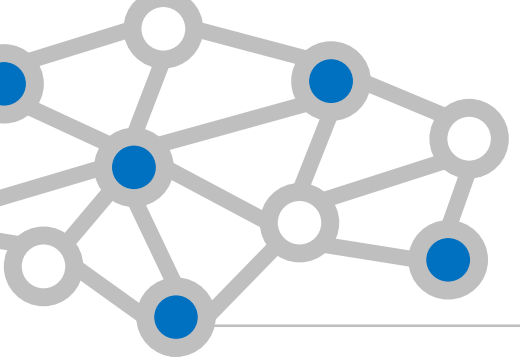
MIPAL aboratory
machine intelligence &pattern analysis

Recurrent Neural Network



Recurrent Neural Network





Default RNN Cells in Tensorflow

- RNNCELL, GRUCELL, LSTMCELL

```
from tensorflow.python.ops import rnn, rnn_cell
from tensorflow.python.ops.rnn_cell import BasicRNNCell,BasicLSTMCell,GRUCell
```

- https://www.tensorflow.org/versions/r0.9/api_docs/python/rnn_cell.html

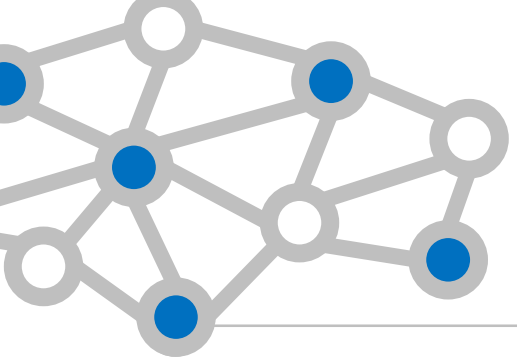
Neural Network RNN Cells

Contents

- Neural Network RNN Cells
  - Base interface for all RNN Cells
    - class tf.nn.rnn_cell.RNNCell
  - RNN Cells for use with TensorFlow's core RNN methods
    - class tf.nn.rnn_cell.BasicRNNCell
    - class tf.nn.rnn_cell.BasicLSTMCell
    - class tf.nn.rnn_cell.GRUCell
    - class tf.nn.rnn_cell.LSTMCell
  - Classes storing split RNNCell state
    - class tf.nn.rnn_cell.LSTMStateTuple
  - RNN Cell wrappers (RNNCells that wrap other RNNCells)
    - class tf.nn.rnn_cell.MultiRNNCell
    - class tf.nn.rnn_cell.DropoutWrapper
    - class tf.nn.rnn_cell.EmbeddingWrapper
    - class tf.nn.rnn_cell.InputProjectionWrapper
    - class tf.nn.rnn_cell.OutputProjectionWrapper

빠른 업데이트 중이므로 설명과 다를수 있습니다

MIPAL aboratory
machine intelligence &pattern analysis



RNN Cell

- Tensorflow.python.ops.rnn_cell.RNNCell
  - Rnn cell, Lstm cell, GRU cell들의 부모 클래스
  - `tf.nn.rnn_cell.RNNCell.output_size`

    Integer: size of outputs produced by this cell.

    `tf.nn.rnn_cell.RNNCell.state_size` ← RNN cell의 hidden node의 수

    Integer or tuple of integers: size(s) of state(s) used by this cell.

    `tf.nn.rnn_cell.RNNCell.zero_state(batch_size, dtype)` ← RNN cell의 hidden node의 초기값(0으로 초기화)

    Return zero-filled state tensor(s).

MIPAL aboratory
machine intelligence &pattern analysis

BasicRNNCell

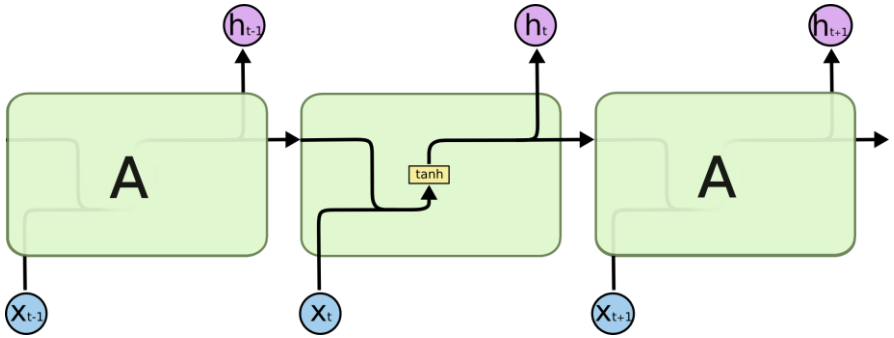
- Tensorflow.python.ops.rnn_cell.BasicRNNCell
  - 기본적인 RNN Cell : hidden node로만 이루어져 있음

- `tf.nn.rnn_cell.BasicRNNCell.__init__(num_units, input_size=None, activation=tanh)`

Deprecated and unused

RNN cell의 hidden node의 수

Activation function 설정

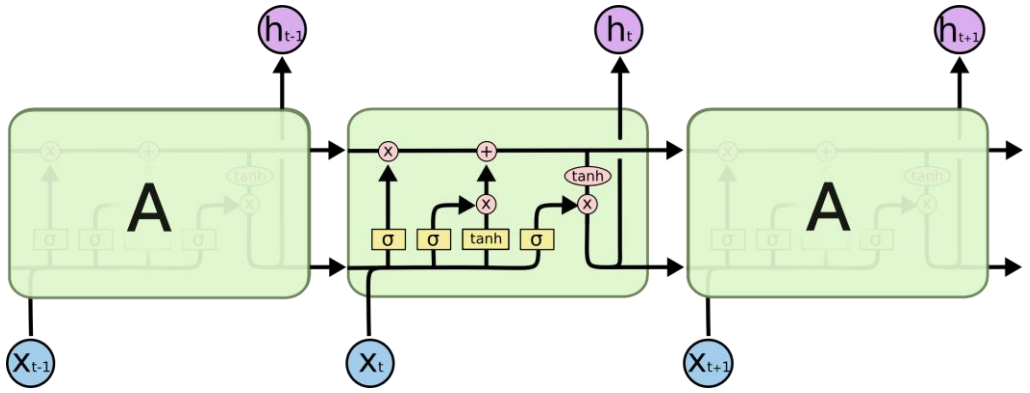


BasicLSTMCell

- Tensorflow.python.ops.rnn_cell.BasicLSTMCell
  - LSTM 셀로 4개의 게이트로 이루어짐

Forget gate의 bias 초기값

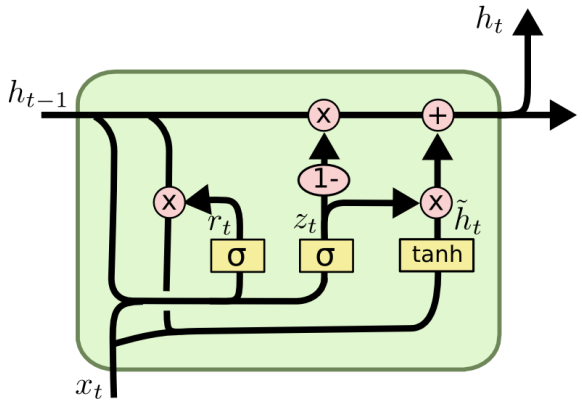
- ```
  tf.nn.rnn_cell.BasicLSTMCell.__init__(num_units, forget_bias=1.0,
  input_size=None, state_is_tuple=False, activation=tanh)
  ```


  ```

GRUCell

- Tensorflow.python.ops.rnn_cell.GRUCell
  - GRU 셀로 LSTM보다 간단한 구조

- `tf.nn.rnn_cell.GRUCell.__init__(num_units, input_size=None, activation=tanh)`

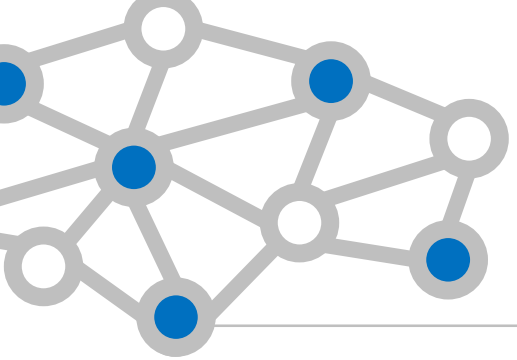


$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$



RNN Cell wrappers

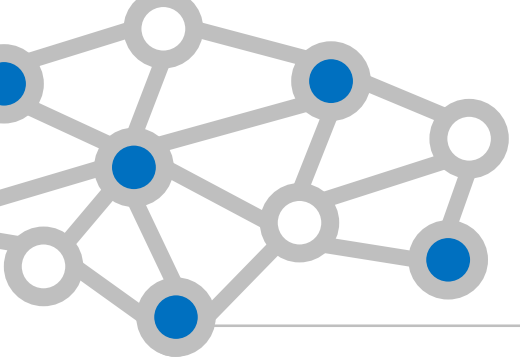
- Tensorflow.python.ops.rnn_cell.MultiRNNCell

```
tf.nn.rnn_cell.MultiRNNCell.__init__(cells, state_is_tuple=False)
```

- Tensorflow.python.ops.rnn_cell.DropoutWrapper

```
tf.nn.rnn_cell.DropoutWrapper.__init__(cell, input_keep_prob=1.0,
output_keep_prob=1.0, seed=None)
```

Dropout 비율



RNN Constructing Module

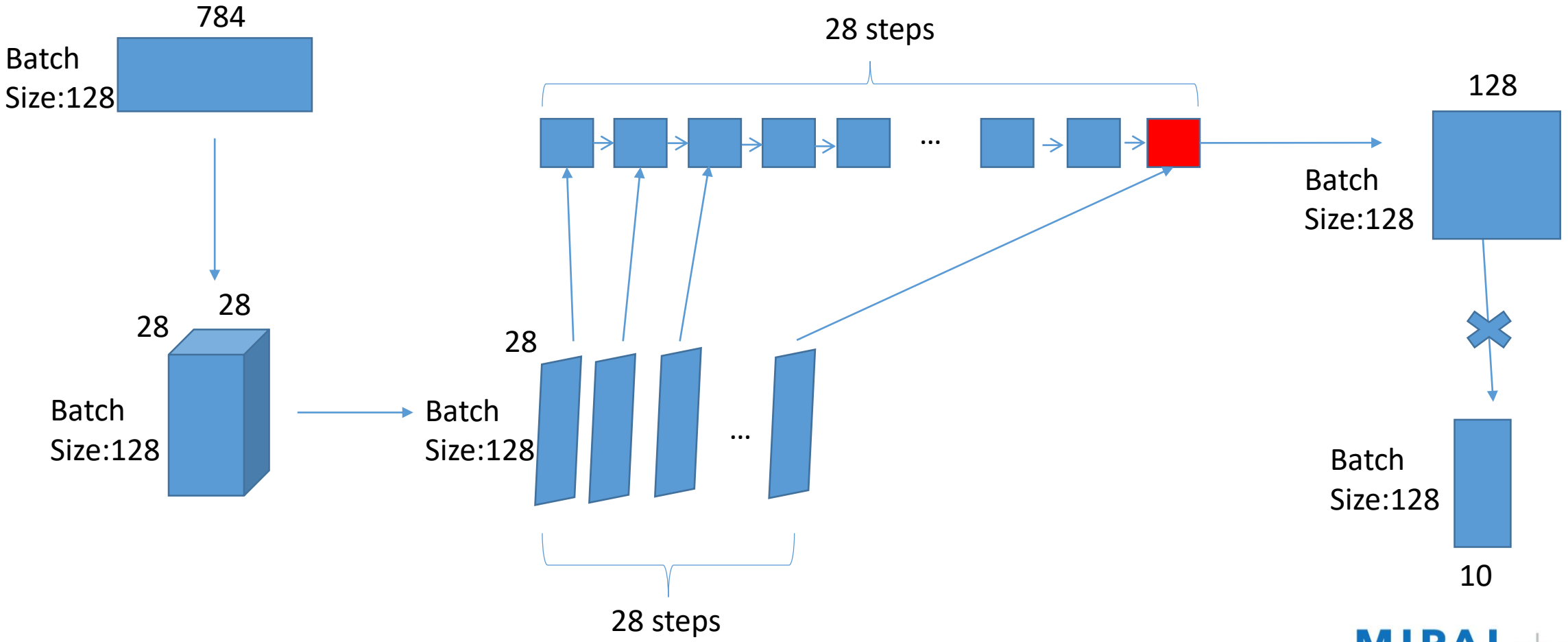
- Tensorflow.python.ops.rnn.rnn

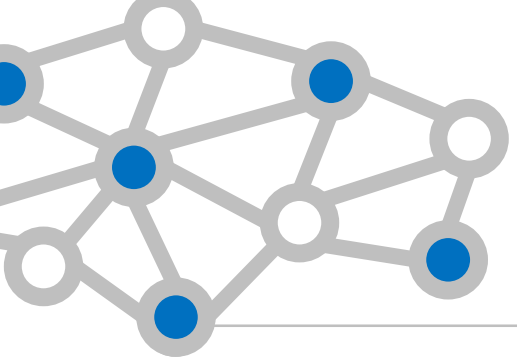
```
tf.nn.rnn(cell, inputs, initial_state=None,
dtype=None, sequence_length=None, scope=None)
```

Args:

- `cell`: An instance of RNNCell. ← RNN cell
- `inputs`: A length T list of inputs, each a `Tensor` of shape `[batch_size, input_size]`, or a nested tuple of such elements. ← Input data
- `initial_state`: (optional) An initial state for the RNN. If `cell.state_size` is an integer, this must be a `Tensor` of appropriate type and shape `[batch_size, cell.state_size]`. If `cell.state_size` is a tuple, this should be a tuple of tensors having shapes `[batch_size, s] for s in cell.state_size`. ← State의 초기값
- `dtype`: (optional) The data type for the initial state and expected output. Required if initial_state is not provided or RNN state has a heterogeneous dtype.
- `sequence_length`: Specifies the length of each sequence in inputs. An int32 or int64 vector (tensor) size `[batch_size]`, values in `[0, T)`.
- `scope`: VariableScope for the created subgraph; defaults to "RNN".

실습 : MNIST 예제

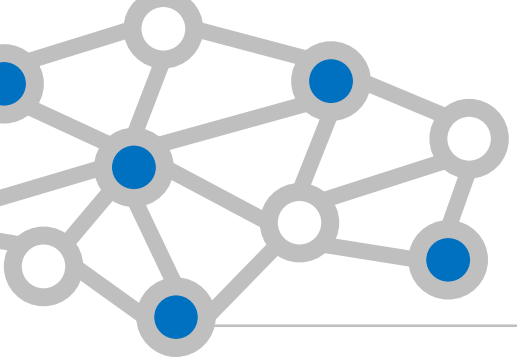




Multi RNN

```
cell = rnn_cell.BasicRNNCell(n_hidden)
cell = DropoutWrapper(cell, output_keep_prob=0.5)
cell = MultiRNNCell([cell] * num_layers)
outputs, states = rnn.rnn(cell, x_t, dtype=tf.float32)
```

- RNN Cell의 List를 MultiRNNCell의 initial argument로 입력
- DropoutWrapper를 이용하여 layer간에 dropout적용

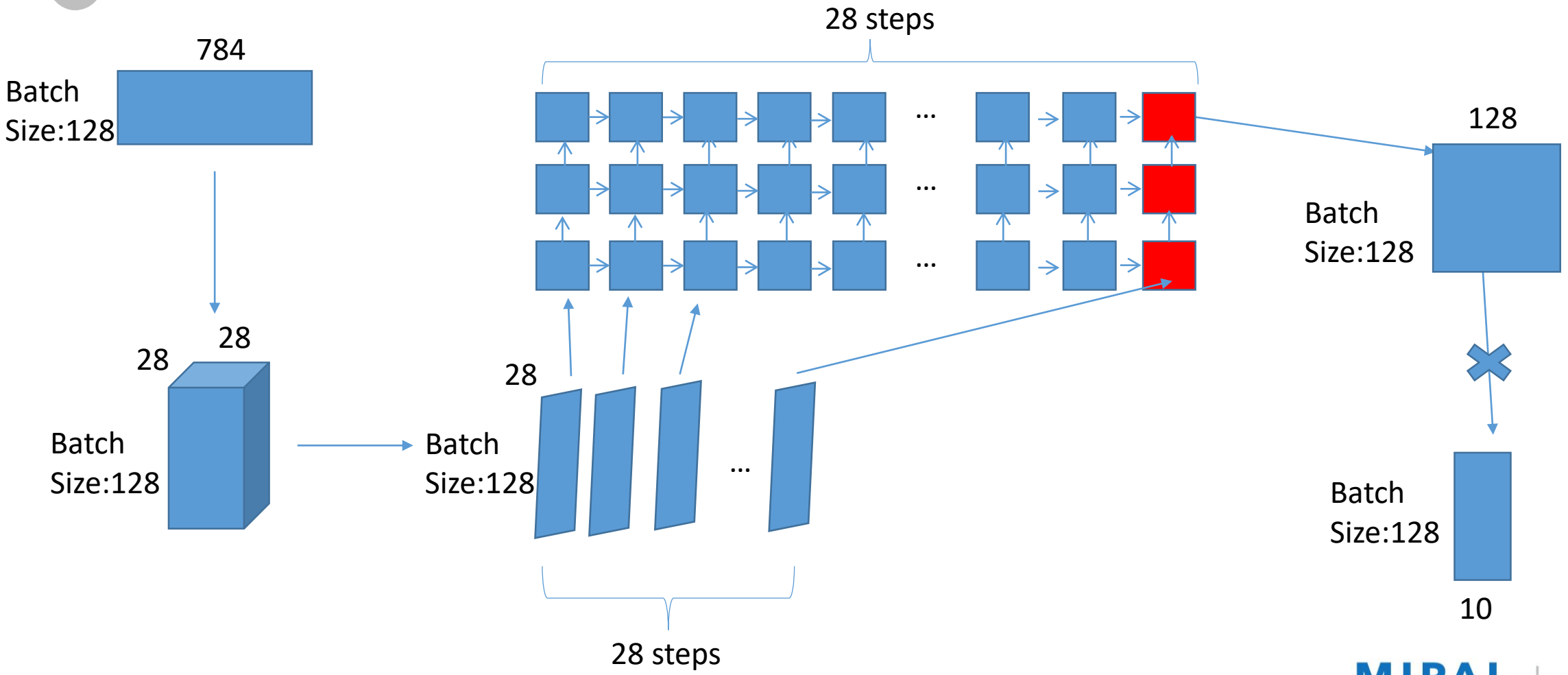


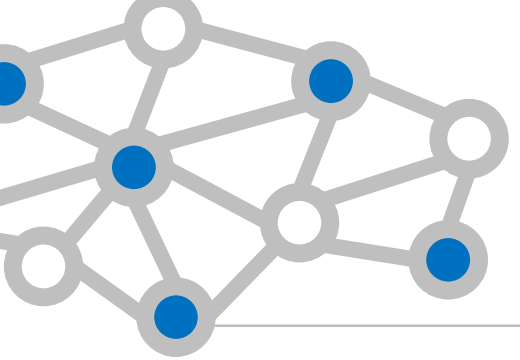
RNN Construct with For loop

```
cell = rnn_cell.BasicRNNCell(n_hidden)
state = cell.zero_state(batch_size, tf.float32)
outputs=[]
for i, x_ in enumerate(x_t):
        if i > 0 :
            tf.get_variable_scope().reuse_variables()

(hidden, state) = cell(x_,state)
outputs.append(hidden)
```

실습 : MNIST 예제



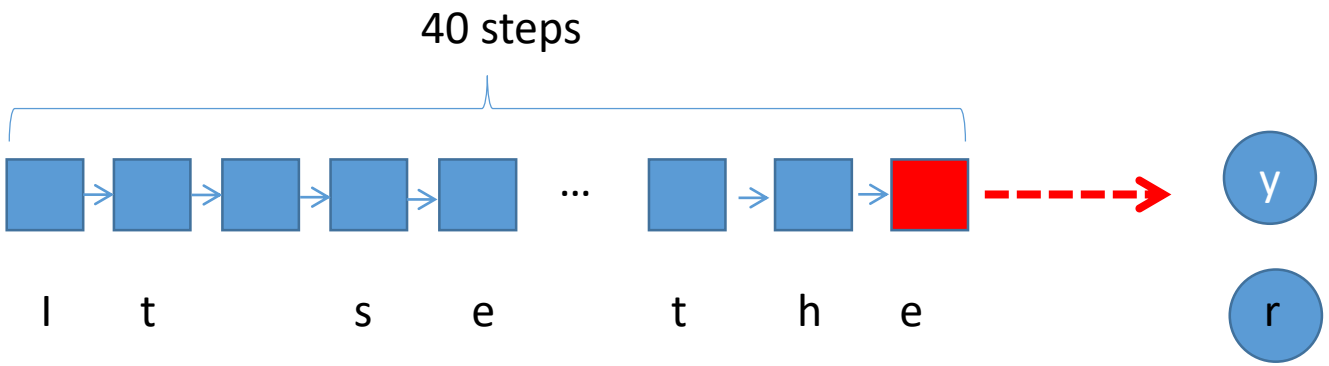


실습: Text Generation(1/5)

- RNN이 가장 많이 이용되는 분야인 NLP 예제
- 그 중 word단위가 아닌 character단위로 텍스트 분석 및 예측
- 알파벳 character 전후 관계와 문장 전체의 information을 이용

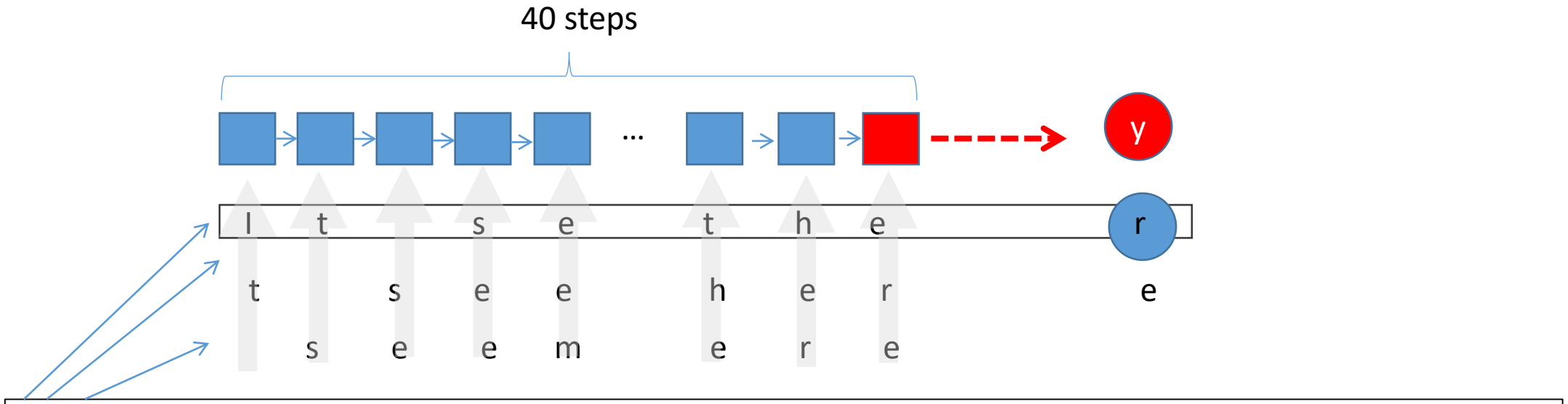
Text Generation (2/5)

- Dataset : nietzsche의 선악의 저편 text (OSIA/data)
- 40 steps의 LSTM 1 layer를 이용하여 다음 character 예측

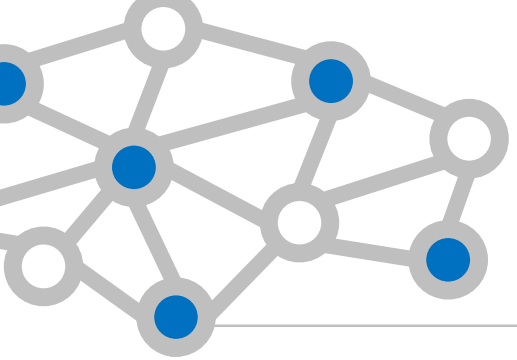


Text Generation (3/5)

- 데이터 전처리
  - Character 59개를 모두 index화
  - 1 character씩 움직이면서 sentence와 예측할 다음 character를 target으로 선정

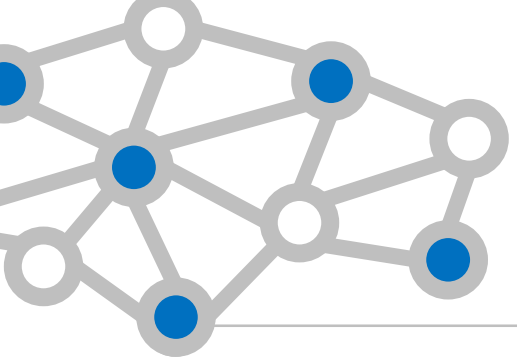


It seems to me that there is everywhere an attempt at present to divert attention



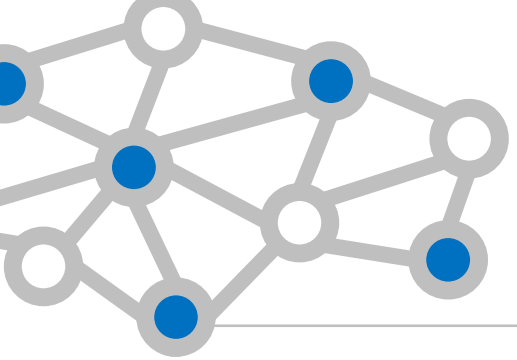
Text Generation (4/5)

- Training Detail
  - Adam optimizer 사용
  - Learning rate = 0.01
  - Batch size = 128
  - LSTM Hidden cell의 수 = 128



Text Generation (5/5)

- Character Sampling
  - 1000 iteration마다 200 characters 연속 생성
  - 생성되는 character를 다시 input으로 사용
  - 트레이닝이 진행될수록 문장을 이루는 character 생성



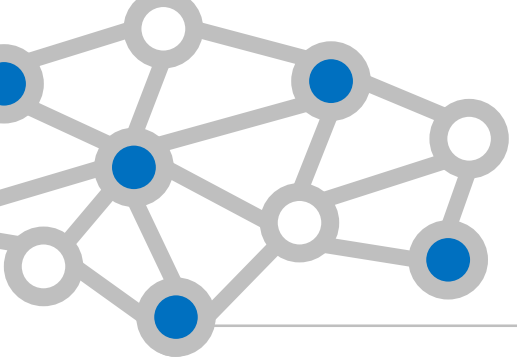
추가 모델

- Many-to-many sequence RNN

  : http://karpathy.github.io/2015/05/21/rnn-effectiveness/

- Sequence to sequence Model

  : https://www.tensorflow.org/versions/r0.10/tutorials/seq2seq/index.html

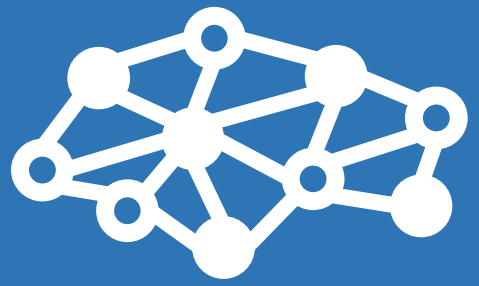


Reference

- https://github.com/aymericdamien/TensorFlow-Examples
- https://github.com/fchollet/keras
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Thanks for Watching

Q&A



MIPAL aboratory

machine intelligence &pattern analysis