# https://www.jiqizhixin.com/articles/2018-02-27-6

# 简单介绍

## 1、导入关键组件

**from** **deeppavlov.skills.pattern\_matching\_skill** **import** PatternMatchingSkill

**from** **deeppavlov.agents.default\_agent.default\_agent** **import** DefaultAgent

**from** **deeppavlov.agents.processors.highest\_confidence\_selector** **import** HighestConfidenceSelector

## 2、为用户的输入创建预设响应

hello = PatternMatchingSkill(responses=['Hello world!'], patterns=["hi", "hello", "good day"])

bye = PatternMatchingSkill(['Goodbye world!', 'See you around'],

patterns=["bye", "chao", "see you"])

fallback = PatternMatchingSkill(["I don't understand, sorry", 'I can say "Hello world!"'])

## 3、高可用代理执行预设响应

agent = DefaultAgent([hello, bye, fallback], skills\_selector=HighestConfidenceSelector())

## 4、代理发言

agent(['Hello', 'Bye', 'Or not'])

# 安装

支持window、Linux ； 依赖Python3.6

## 基于3.6创建虚拟环境

virtualenv env

## 激活环境

Linux

source ./env/bin/activate

Windows

.**\e**nv**\S**cripts**\a**ctivate.bat

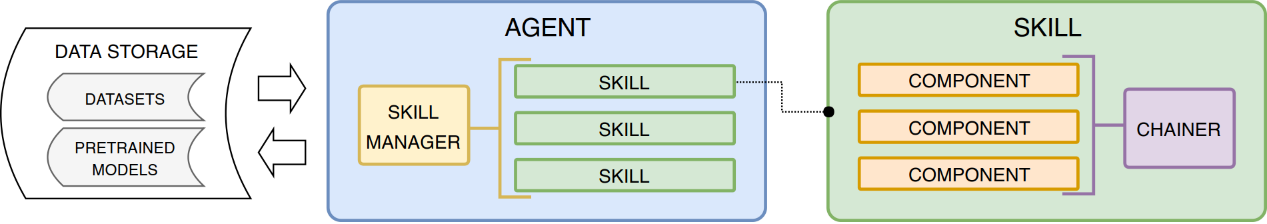
## 安装到虚拟环境

pip install deeppavlov

# 概念概述

## 框架提供

* 一组预先训练的NLP模型，预定义的对话系统组件（基于ML / DL /规则）和管道模板;
* 用于实现和测试自己的对话模型的框架;
* 应用程序与相邻基础设施集成的工具（信使，服务台软件等）;
* 会话模型的基准环境和相关数据集的统一访问。



## 关键概念

Agent 是一种会话代理，以自然语言（文本）与用户进行通信。

Skill在某些领域实现用户的目标。通常，这是通过提供信息或完成交易（例如，通过常见问题解答回答问题，预订门票等）来实现的。然而，对于某些任务，成功的交互被定义为持续参与。

Component是一个可重复使用的功能部分Skill。

Rule-based Models 无法接受培训。

Machine Learning Models 只能独自训练。

Deep Learning Models 可以独立培训，并以端到端的方式连接在一起。

Skill Manager执行选择Skill以生成响应。

Chainer从异构组件构建代理/组件管道（基于规则/ ML / DL）。它允许在整个管道中训练和推断模型。

 Component代表NLP管道中的任何类型的功能。它可以实现为神经网络，非神经ML模型或基于规则的系统。除此之外，Component可以有嵌套结构，即Component可以包含其他Components。

Components可以加入到Skill。Skill与之相比，解决了更大的NLP任务Component。但是，在实现方面，Skills与Components 没有区别。Skills 的唯一限制是它们的输入和输出都应该是字符串。因此，Skills通常与对话任务相关联。

Agent应该是一个包含多个Skills 的多用途对话系统，可以在它们之间切换。它可以是一个包含目标导向和聊天机器人技能的对话系统，并根据用户输入选择用于生成答案的对话系统。

# 框架特性

## Component

### [NER component](http://docs.deeppavlov.ai/en/master/components/ner.html)

基于神经命名实体识别网络。NER组件从[混合Bi-LSTM-CRF模型的应用](https://arxiv.org/pdf/1709.09686.pdf)再现[了俄罗斯命名实体识别任务的](https://arxiv.org/pdf/1709.09686.pdf) 架构，其受到来自[https://arxiv.org/pdf/1603.01360.pdf的](https://arxiv.org/pdf/1603.01360.pdf) Bi-LSTM + CRF架构的启发

### [Slot filling components](http://docs.deeppavlov.ai/en/master/components/slot_filling.html)

基于模糊Levenshtein搜索从文本中提取归一化槽值。组件要么依赖于NER结果，要么在大海捞针搜索中执行针。

### [Classification component](http://docs.deeppavlov.ai/en/master/components/classifiers.html)

用于字级别的分类任务（意图，情感等）的组件。提出了具有自我关注的浅宽CNN，深CNN，BiLSTM，BiLSTM等模型。该模型还允许对文本进行多标签分类。有几种预先训练的模型可供选择，如下表所示。

### [Goal-oriented bot](http://docs.deeppavlov.ai/en/master/skills/go_bot.html)

基于混合代码网络（HCN）架构，来自[Jason D. Williams，Kavosh Asadi，Geoffrey Zweig，混合代码网络：实用且高效的端到端对话控制，具有监督和强化学习 - 2017年](https://arxiv.org/abs/1702.03274)。它允许在面向目标的对话框中预测响应。该模型是可定制的：嵌入，槽填充和意图分类器可以根据需要打开和关闭。

### [Seq2seq goal-oriented bot](http://docs.deeppavlov.ai/en/master/skills/seq2seq_go_bot.html)

对话代理在面向目标的对话框中预测响应，并且能够处理多个域（预训练的机器人允许日历安排，天气信息检索和兴趣点导航）。该模型是端到端的可区分的，不需要明确地模拟对话状态或信念跟踪器。

### [Automatic spelling correction component](http://docs.deeppavlov.ai/en/master/components/spelling_correction.html)

使用候选的管道在静态字典和ARPA语言模型中搜索以纠正拼写错误。

### [Ranking component](http://docs.deeppavlov.ai/en/master/components/neural_ranking.html)

基于[LSTM的深层学习模型，用于非仿真答案选择](https://arxiv.org/abs/1511.04108)。该模型通过它们与给定上下文的相关性来执行来自某个数据库的响应或上下文的排名。

### [TF-IDF Ranker component](http://docs.deeppavlov.ai/en/master/components/tfidf_ranking.html)

基于[阅读维基百科来回答开放域问题](https://github.com/facebookresearch/DrQA/)。该模型解决了给定查询的文档检索任务。

### [Question Answering component](http://docs.deeppavlov.ai/en/master/components/squad.html)

基于[R-NET：具有自匹配网络的机器阅读理解](https://www.microsoft.com/en-us/research/publication/mcr/)。该模型解决了在给定上下文中查找问题答案的任务（[SQuAD](https://rajpurkar.github.io/SQuAD-explorer/)任务格式）

### [Morphological tagging component](http://docs.deeppavlov.ai/en/master/components/morphotagger.html)

基于基于特征的形态标记方法[Heigold等，2017年。对14种语言的基于字符的形态标记进行了广泛的实证评估](http://www.aclweb.org/anthology/E17-1048)。俄罗斯和其他几种语言的最先进模型。模型作为输入标记化句子并以[UD格式](http://universaldependencies.org/format.html)输出相应的形态标签序列。下表包含UD2.0数据集的单词和句子准确性。有关更多分数，请参阅[全表](http://docs.deeppavlov.ai/en/master/components/morphotagger.html)。

### [Frequently Asked Questions (FAQ) component](http://docs.deeppavlov.ai/en/master/skills/faq.html)

常见问题任务的管道集：将传入的问题分类为一组已知问题并返回准备好的答案。您可以基于以下内容构建不同的管道：tf-idf，加权fasttext，余弦相似度，逻辑回归。

## Skills

### [eCommerce bot](http://docs.deeppavlov.ai/en/master/skills/ecommerce.html)

电子商务机器人打算按排序顺序从目录中检索产品项目。此外，它要求用户提供额外信息以指定搜索。

### [ODQA](http://docs.deeppavlov.ai/en/master/skills/odqa.html)

一个开放的域名问题回答技巧。该技能接受关于世界的自由形式问题，并根据其维基百科知识输出答案。

## AutoML

### [Hyperparameters optimization](http://docs.deeppavlov.ai/en/master/intro/hypersearch.html)

DeepPavlov模型的超参数优化（通过交叉验证或神经演化），只需要在配置文件中进行一些小的更改。

## Embeddings

### [Pre-trained embeddings for the Russian language](http://docs.deeppavlov.ai/en/master/intro/pretrained_vectors.html)

在联合[俄语维基百科](https://ru.wikipedia.org/)和[Lenta.ru](https://lenta.ru/)语料库上训练的俄语语言向量。

## 一些组件的例子

使用Telegram接口运行面向目标的bot：

python -m deeppavlov interactbot deeppavlov/configs/go\_bot/gobot\_dstc2.json -d -t <TELEGRAM\_TOKEN>

使用控制台界面运行面向目标的bot：

python -m deeppavlov interact deeppavlov/configs/go\_bot/gobot\_dstc2.json -d

使用REST API运行面向目标的bot：

python -m deeppavlov riseapi deeppavlov/configs/go\_bot/gobot\_dstc2.json -d

使用Telegram接口运行插槽填充模型：

python -m deeppavlov interactbot deeppavlov/configs/ner/slotfill\_dstc2.json -d -t <TELEGRAM\_TOKEN>

使用控制台界面运行插槽填充模型：

python -m deeppavlov interact deeppavlov/configs/ner/slotfill\_dstc2.json -d

使用REST API运行插槽填充模型：

python -m deeppavlov riseapi deeppavlov/configs/ner/slotfill\_dstc2.json -d

预测文件中每一行的意图：

python -m deeppavlov predict deeppavlov/configs/classifiers/intents\_snips.json -d --batch-size 15 < /data/in.txt > /data/out.txt

# 配置文件

## NLP管道配置文件（json）-只有一个节点

{

"chainer": {

"in": ["x"],

"in\_y": ["y"],

"pipe": [

...

],

"out": ["y\_predicted"]

}

}

## 变量配置文件

DP\_VARIABLE\_NAME will override VARIABLE\_NAME inside a configuration file

{

"chainer": {

"in": ["x"],

"pipe": [

{

"class\_name": "my\_component",

"in": ["x"],

"out": ["x"],

"load\_path": "{MY\_PATH}/file.obj"

},

{

"in": ["x"],

"out": ["y\_predicted"],

"config\_path": "{CONFIGS\_PATH}/classifiers/intents\_snips.json"

}

],

"out": ["y\_predicted"]

},

"metadata": {

"variables": {

"MY\_PATH": "/some/path",

"CONFIGS\_PATH": "{DEEPPAVLOV\_PATH}/configs"

}

}

}

## Training

 Two abstract classes for trainable components: [**Estimator**](http://docs.deeppavlov.ai/en/master/apiref/core/models.html#deeppavlov.core.models.estimator.Estimator) and **[NNModel](http://docs.deeppavlov.ai/en/master/apiref/core/models.html" \l "deeppavlov.core.models.nn_model.NNModel" \o "deeppavlov.core.models.nn_model.NNModel)**.

### [Estimator](http://docs.deeppavlov.ai/en/master/apiref/core/models.html#deeppavlov.core.models.estimator.Estimator)

are fit once on any data with no batching or early stopping, so it can be safely done at the time of pipeline initialization. **fit()** method has to be implemented for each [**Estimator**](http://docs.deeppavlov.ai/en/master/apiref/core/models.html#deeppavlov.core.models.estimator.Estimator). One example is **Vocab**.

### [NNModel](http://docs.deeppavlov.ai/en/master/apiref/core/models.html#deeppavlov.core.models.nn_model.NNModel)

requires more complex training. It can only be trained in a supervised mode (as opposed to [**Estimator**](http://docs.deeppavlov.ai/en/master/apiref/core/models.html#deeppavlov.core.models.estimator.Estimator) which can be trained in both supervised and unsupervised settings). This process takes multiple epochs with periodic validation and logging.**train\_on\_batch()** method has to be implemented for each **[NNModel](http://docs.deeppavlov.ai/en/master/apiref/core/models.html" \l "deeppavlov.core.models.nn_model.NNModel" \o "deeppavlov.core.models.nn_model.NNModel)**.

Training is triggered by **[train\_evaluate\_model\_from\_config()](http://docs.deeppavlov.ai/en/master/apiref/core/commands.html" \l "deeppavlov.core.commands.train.train_evaluate_model_from_config" \o "deeppavlov.core.commands.train.train_evaluate_model_from_config)** function.

### Train config

[**Estimator**](http://docs.deeppavlov.ai/en/master/apiref/core/models.html#deeppavlov.core.models.estimator.Estimator) s that are trained should also have fit\_on parameter which contains a list of input parameter names. An **[NNModel](http://docs.deeppavlov.ai/en/master/apiref/core/models.html" \l "deeppavlov.core.models.nn_model.NNModel" \o "deeppavlov.core.models.nn_model.NNModel)**should have the in\_y parameter which contains a list of ground truth answer names. For example:

[

{

"id": "classes\_vocab",

"class\_name": "default\_vocab",

"fit\_on": ["y"],

"level": "token",

"save\_path": "vocabs/classes.dict",

"load\_path": "vocabs/classes.dict"

},

{

"in": ["x"],

"in\_y": ["y"],

"out": ["y\_predicted"],

"class\_name": "intent\_model",

"save\_path": "classifiers/intent\_cnn",

"load\_path": "classifiers/intent\_cnn",

"classes\_vocab": {

"ref": "classes\_vocab"

}

}

]

The config for training the pipeline should have three additional elements: dataset\_reader, dataset\_iterator and train:

{

"dataset\_reader": {

"class\_name": ...,

...

},

"dataset\_iterator": {

"class\_name": ...,

...

},

"chainer": {

...

},

"train": {

...

}

}

Simplified version of training pipeline contains two elements: dataset and train. The dataset element currently can be used for train from classification data in csv and json formats. You can find complete examples of how to use simplified training pipeline in [intents\_sample\_csv.json](https://github.com/deepmipt/DeepPavlov/blob/0.1.5.1/deeppavlov/configs/classifiers/intents_sample_csv.json) and [intents\_sample\_json.json](https://github.com/deepmipt/DeepPavlov/blob/0.1.5.1/deeppavlov/configs/classifiers/intents_sample_json.json) config files.

### Train Parameters

* epochs — maximum number of epochs to train NNModel, defaults to -1 (infinite)
* batch\_size,
* metric\_optimization — maximize or minimize a metric, defaults to maximize
* validation\_patience — how many times in a row the validation metric has to not improve for early stopping, defaults to 5
* val\_every\_n\_epochs — how often to validate the pipe, defaults to -1 (never)
* log\_every\_n\_batches, log\_every\_n\_epochs — how often to calculate metrics for train data, defaults to -1 (never)
* validate\_best, test\_best flags to infer the best saved model on valid and test data, defaults to true
* tensorboard\_log\_dir — path to write logged metrics during training. Use tensorboard to visualize metrics plots.
* metrics — list of [metrics](http://docs.deeppavlov.ai/en/master/apiref/metrics.html#module-deeppavlov.metrics) to evaluate the model.

### Metrics

"train": {

"metrics": [

"f1",

{

"name": "accuracy",

"inputs": ["y", "y\_labels"]

},

{

"name": "roc\_auc",

"inputs": ["y", "y\_probabilities"]

}

],

...

}

The first metric in the list is used for early stopping.

Each metric can be described as a JSON object with name and inputs properties, where name is a registered name of a metric function and inputs is a list of parameter names from chainer’s inner memory that will be passed to the metric function.

If a metric is described as a single string, this string is interpreted as a registered name.

Default value for inputs parameter is a concatenation of chainer’s in\_y and out parameters.

### DatasetReader

DatasetReader class reads data and returns it in a specified format. A concrete DatasetReader class should be inherited from this base class and registered with a codename:

from deeppavlov.core.common.registry import register

from deeppavlov.core.data.dataset\_reader import DatasetReader

@register('dstc2\_datasetreader')

class DSTC2DatasetReader(DatasetReader):

### DataLearningIterator and DataFittingIterator

[**DataLearningIterator**](http://docs.deeppavlov.ai/en/master/apiref/core/data.html#deeppavlov.core.data.data_learning_iterator.DataLearningIterator) forms the sets of data (‘train’, ‘valid’, ‘test’) needed for training/inference and divides them into batches. A concrete **DataLearningIterator** class should be registered and can be inherited from **deeppavlov.data.data\_learning\_iterator.DataLearningIterator** class. This is a base class and can be used as a **DataLearningIterator** as well.

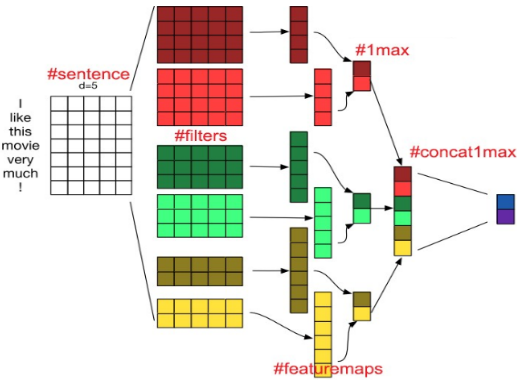
[**DataFittingIterator**](http://docs.deeppavlov.ai/en/master/apiref/core/data.html#deeppavlov.core.data.data_fitting_iterator.DataFittingIterator) iterates over provided dataset without train/valid/test splitting and is useful for [**Estimator**](http://docs.deeppavlov.ai/en/master/apiref/core/models.html#deeppavlov.core.models.estimator.Estimator) s that do not require training.

### Inference(推断)

All components inherited from [**Component**](http://docs.deeppavlov.ai/en/master/apiref/core/models.html#deeppavlov.core.models.component.Component) abstract class can be used for inference. The **\_\_call\_\_()** method should return standard output of a component. For example, a tokenizer should return tokens, a NER recognizer should return recognized entities, a bot should return an utterance. A particular format of returned data should be defined in **\_\_call\_\_()**.

Inference is triggered by **[interact\_model()](http://docs.deeppavlov.ai/en/master/apiref/core/commands.html" \l "deeppavlov.core.commands.infer.interact_model" \o "deeppavlov.core.commands.infer.interact_model)** function. There is no need in a separate JSON for inference.

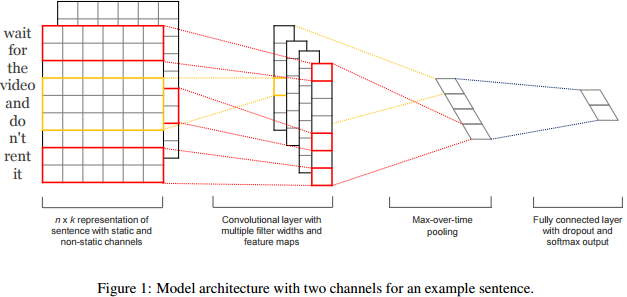
# 其他论述



一维卷积在语义理解中的应用，莫斯科物

理技术学院（MIPT）开源聊天机器人DeepPavlov解析及代码

莫斯科物理技术学院（MIPT）前不久[开源了聊天机器人DeepPavlov](https://github.com/deepmipt/DeepPavlov)（基于TensorFlow和Keras），对于有志学习NLP的同学是个不错的借鉴。其中的语义理解分类模块，不是采用的[马尔科夫随机场（CRF）](https://baike.baidu.com/item/%E9%A9%AC%E5%B0%94%E5%8F%AF%E5%A4%AB%E9%9A%8F%E6%9C%BA%E5%9C%BA)或RNN， 而是使用**一维卷积**去做语义分类：

[](http://nooverfit.com/wp/wp-content/uploads/2018/04/Screenshot-from-2018-04-16-105554.png)来自：https://arxiv.org/pdf/1408.5882.pdf

这一模型是来自纽约大学的论文：[Convolutional Neural Networks for Sentence Classification](https://arxiv.org/pdf/1408.5882.pdf)，用**一维卷积**为句子的语义分类。

值得注意的是，上图模型中，**第一层**句子的**词嵌入输入**（embedding）并不是单个通道，而是**两个通道**的。每个通道用相同的word2vec的embedding方法初始化（表征同一个句子），有意思的是，文章指出，把其中一个通道的词表征静态化（static），而另一个通道的词表征设为**可训练的**（non-static），这样两个通道组合在一起可以有更好效果。

**第二层中**，与图像卷积把输出特征通道叠加的做法不同，这里的1D卷积是用**concatenate**拼接操作而不是加和**add**操作。

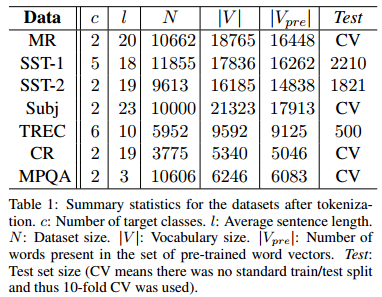
当然，[DeepPavlov源码中](https://github.com/deepmipt/DeepPavlov)似乎没有考虑到原版论文中的双通道方法，只是用了单通道的embedding输入：

1. **def** cnn\_model(self, params):
2. """
3. Build un-compiled model of shallow-and-wide CNN
4. Args:
5. params: dictionary of parameters for NN
6. Returns:
7. Un-compiled model
8. """
9. inp = Input(shape=(params['text\_size'], params['embedding\_size']))

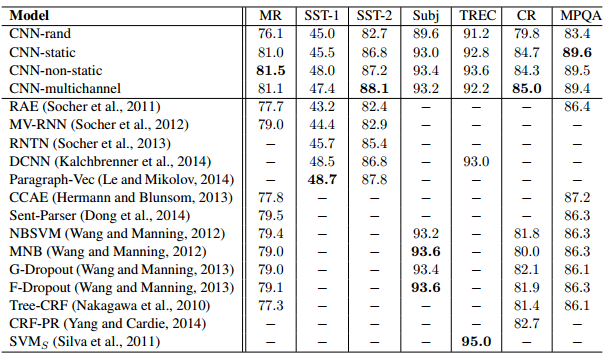
参见源码：**deeppavlov/models/classifiers/intents/intent\_model.py**

最后了解一下论文实验方法：

首先数据集作者用的是影评感受MR（movie review）的相关数据集和扩展，给出一句话的影片评价，判别是褒扬/批评/中性等等的评价等级，SST-1，SST-2等都是MR数据集的扩展数据集，这些数据集的一些信息如下：

[](http://nooverfit.com/wp/wp-content/uploads/2018/04/Screenshot-from-2018-04-16-202546.png)来自：https://arxiv.org/pdf/1408.5882.pdf

模型性能方面，1D的CNN方法较传统机器学习方法有一定竞争力，当然在一些数据集上表现并不算好：

[](http://nooverfit.com/wp/wp-content/uploads/2018/04/Screenshot-from-2018-04-16-202739.png)来自：https://arxiv.org/pdf/1408.5882.pdf

参考文献：

1. <https://github.com/deepmipt/DeepPavlov>
2. [Convolutional Neural Networks for Sentence Classification](https://arxiv.org/pdf/1408.5882.pdf)
3. [Understanding how Convolutional Neural Network (CNN) perform text classification with word embeddings](https://towardsdatascience.com/understanding-how-convolutional-neural-network-cnn-perform-text-classification-with-word-d2ee64b9dd0b)