自编码与预训练模型-C21

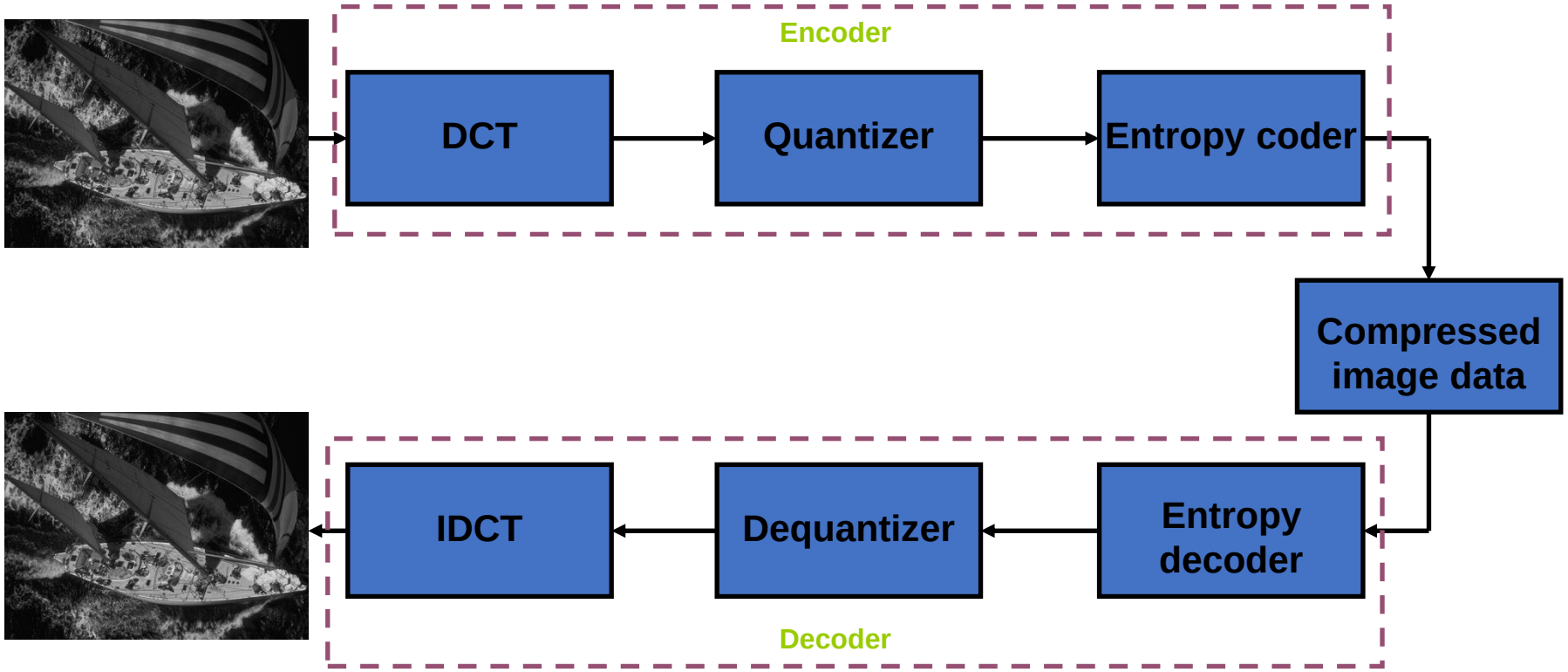
信息科学与技术学院

胡俊峰

主要内容

- 自编码概念
- 预训练网络模型

自编码概念的提出


Encoder
DCT
Quantizer
Entropy coder
Compressed image data
Entropy decoder
Dequantizer
IDCT
Decoder

有损与无损编码

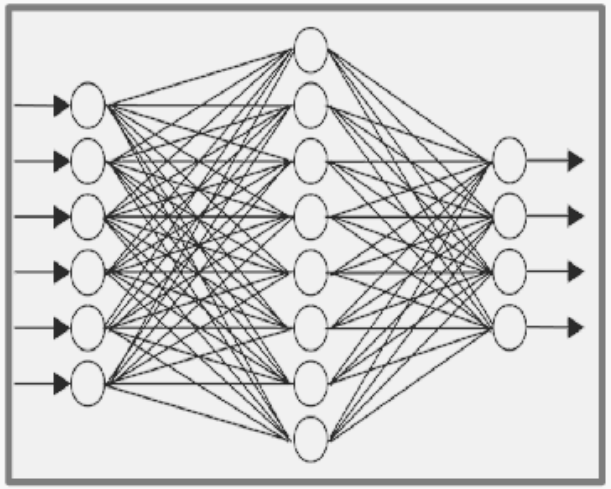
- Huffman编码
- JPEG编码

https://jpeg.org/jpeg2000/index.html

基于目标任务的有监督网络模型

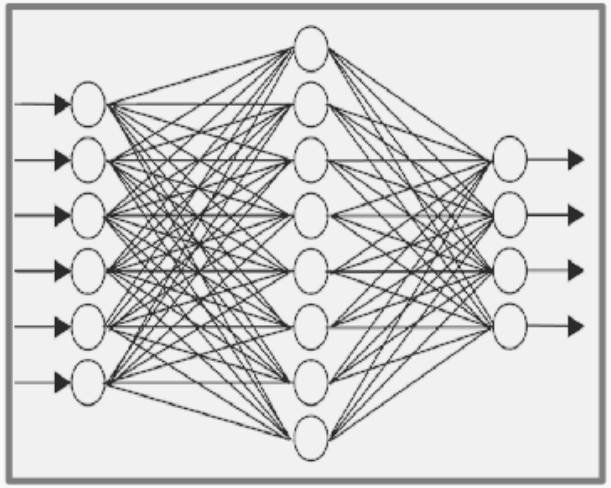
We train the two networks by minimizing the loss function (**cross entropy loss**)

X



Feature Discovery Network

Features
$\phi(x)$



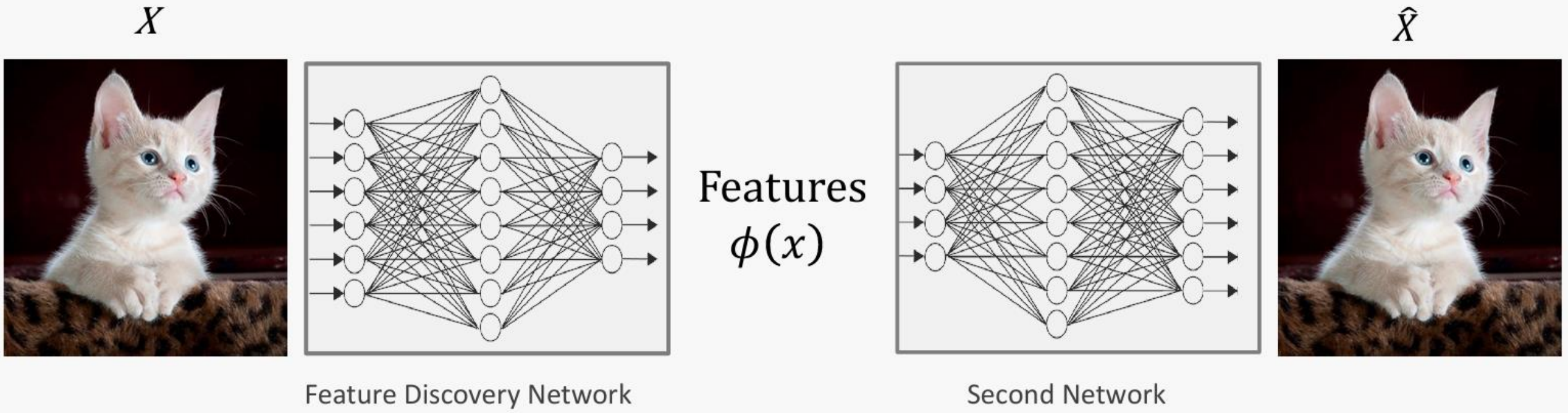
Classification Network

Y

{Cat,Dog}

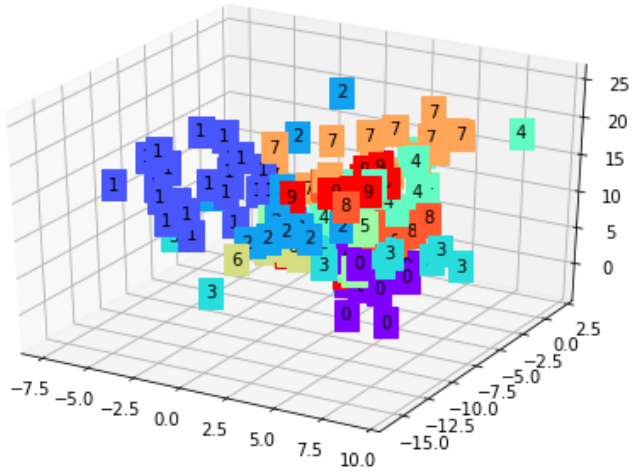
基于自编码损失的自编码网络

We train the two networks by minimizing the **reconstruction** loss function: $\mathcal{L} = \sum(x_i - \hat{x}_i)^2$



多层全连接自编码网络

```python
# 开始训练自动编码器
for e in range(100):
    for im, _ in train_data:
        im = im.view(im.shape[0], -1)
        im = Variable(im)
        # 前向传播
        _, output = net(im)
        loss = criterion(output, im) / im.shape[0] # 平均
        # 反向传播
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```




```python
class autoencoder(nn.Module):
    def __init__(self):
        super(autoencoder, self).__init__()

        self.encoder = nn.Sequential(
            nn.Linear(28*28, 128),
            nn.ReLU(True),
            nn.Linear(128, 64),
            nn.ReLU(True),
            nn.Linear(64, 12),
            nn.ReLU(True),
            nn.Linear(12, 3) # 输出的 code 是 3 维，便于可视化
        )

        self.decoder = nn.Sequential(
            nn.Linear(3, 12),
            nn.ReLU(True),
            nn.Linear(12, 64),
            nn.ReLU(True),
            nn.Linear(64, 128),
            nn.ReLU(True),
            nn.Linear(128, 28*28),
            nn.Tanh()
        )
```

https://wizardforcel.gitbooks.io/learn-dl-with-pytorch-liaoxingyu/content/6.1.html

卷积-反卷积自编码网络

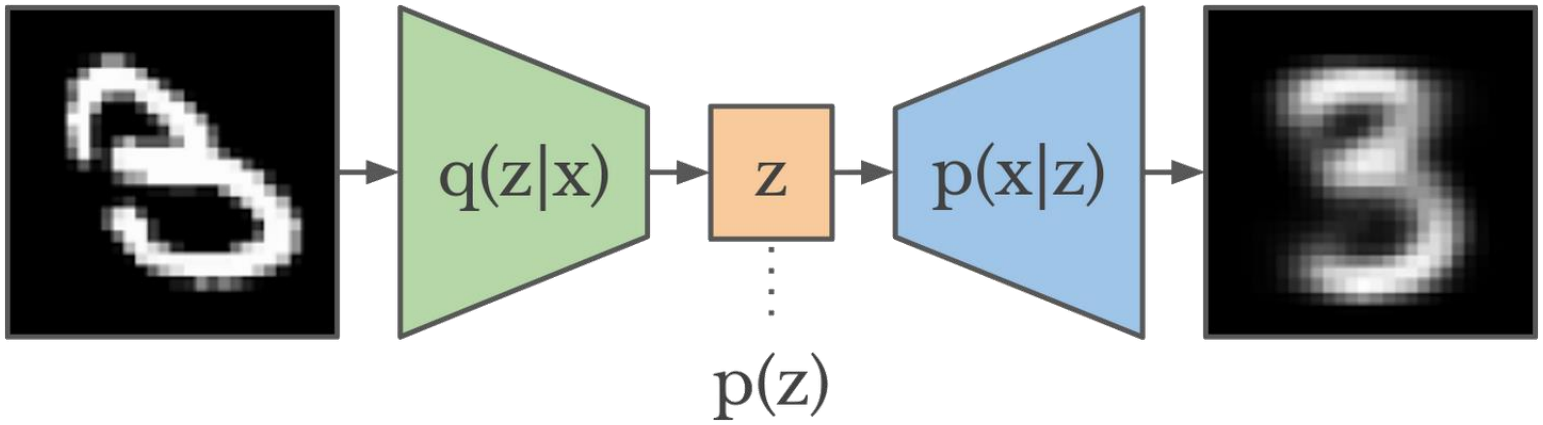
```
# 开始训练自动编码器
for e in range(40):
    for im, _ in train_data:
        if torch.cuda.is_available():
            im = im.cuda()
        im = Variable(im)
        # 前向传播
        _, output = conv_net(im)
        loss = criterion(output, im) / im.shape[0]
        # 反向传播
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

```
class conv_autoencoder(nn.Module):
    def __init__(self):
        super(conv_autoencoder, self).__init__()

        self.encoder = nn.Sequential(
            nn.Conv2d(1, 16, 3, stride=3, padding=1),  # (b, 16, 10, 10)
            nn.ReLU(True),
            nn.MaxPool2d(2, stride=2),  # (b, 16, 5, 5)
            nn.Conv2d(16, 8, 3, stride=2, padding=1),  # (b, 8, 3, 3)
            nn.ReLU(True),
            nn.MaxPool2d(2, stride=1)  # (b, 8, 2, 2)
        )

        self.decoder = nn.Sequential(
            nn.ConvTranspose2d(8, 16, 3, stride=2),  # (b, 16, 5, 5)
            nn.ReLU(True),
            nn.ConvTranspose2d(16, 8, 5, stride=3, padding=1),  # (b, 8, 15, 15)
            nn.ReLU(True),
            nn.ConvTranspose2d(8, 1, 2, stride=2, padding=1),  # (b, 1, 28, 28)
            nn.Tanh()
        )
```

https://d2l.ai/chapter_computer-vision/transposed-conv.html

基于隐空间的自编码模型（VAE）

- 变分自编码器：



https://blog.csdn.net/ChronoPrison/article/details/104685318

文本序列的自编码模型

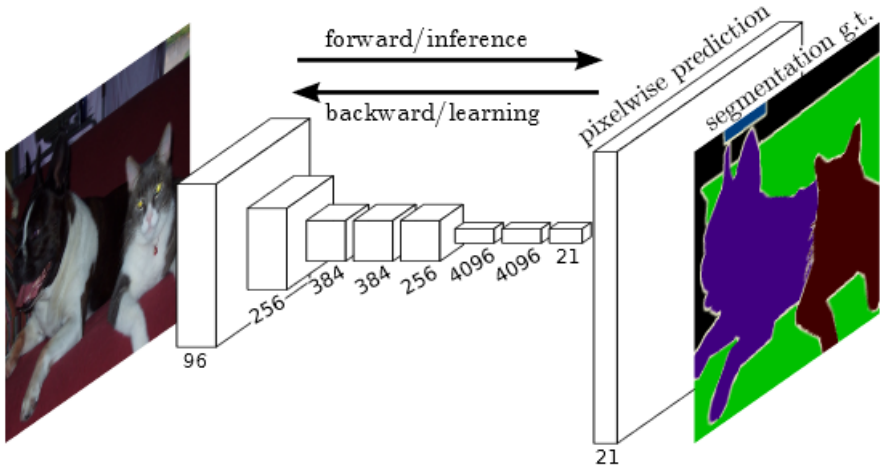
- Word embedding
- Revers translation
- BERT-预训练模型

字面义上的预训练模型（Pre-train model）

- 利用数据资源训练好的模型，可以直接拿来应用（forward）
- 例如：
  - 用已经训练好的词向量模型（word-embedding）进行词汇建模
    - 在词向量的基础上做词义检索
    - 用词向量做下一步句子建模（LSTM）的输入

早期的预训练模型（Pre-train + fine tune）

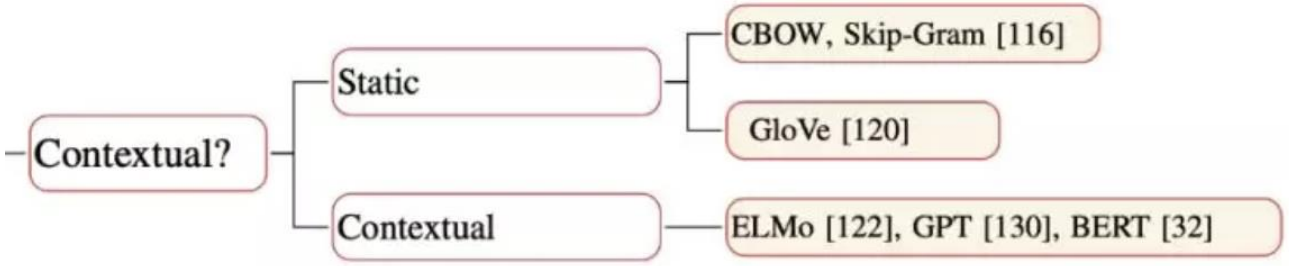
- 针对特定任务用大规模数据进行训练，取得好的模型效果
- 然后fine tune
  - 在某个特定领域加入领域数据进行模型微调
    - 例：大规模wiki的词向量 + 医学语料fine tune → 医学领域词向量
  - 在某个特定任务上进行大规模训练，得到多层模型，然后再根据另一个相关任务对模型的后两层或中间某些层进行重训练
  - 例： ResNet-101 目标任务是semantic-segmentation
    - 可以用较少的任务数据训练模型的最后两层



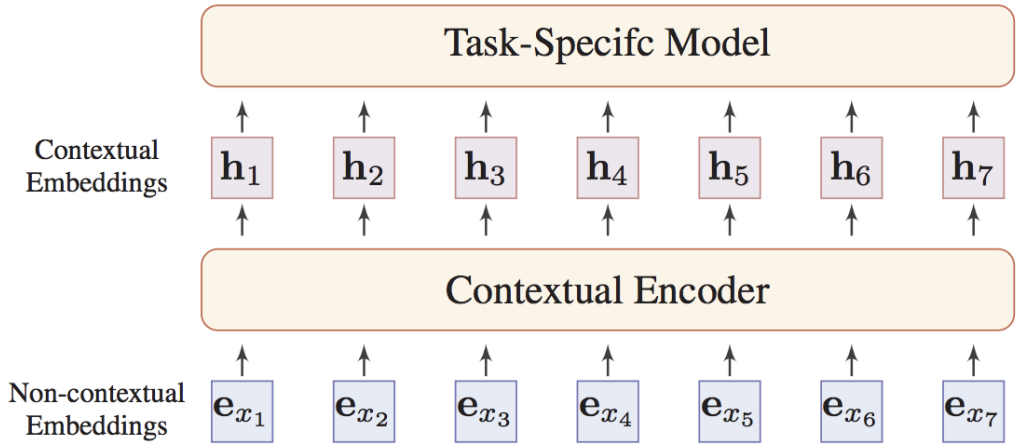
基于大型语料库的PTM

- 大量的研究工作表明，大型语料库上的预训练模型（PTM）已经可以学习通用的语言表征，这对于下游的 NLP 相关任务是非常有帮助的，可以避免大量从零开始训练新模型。而随着算力的发展、深层模型（Transformer）出现以及训练技能的不断提高，PTM 体系结构已然从浅层发展到了深层。现有的PTM大致从以下四个方面进行探究：
  - 预训练方法（PTM）使用的词表征类型
  - 预训练方法使用的主干网络
  - PTM 使用的 预训练任务类型
  - 为特定场景与输入类型所设计的 PTM

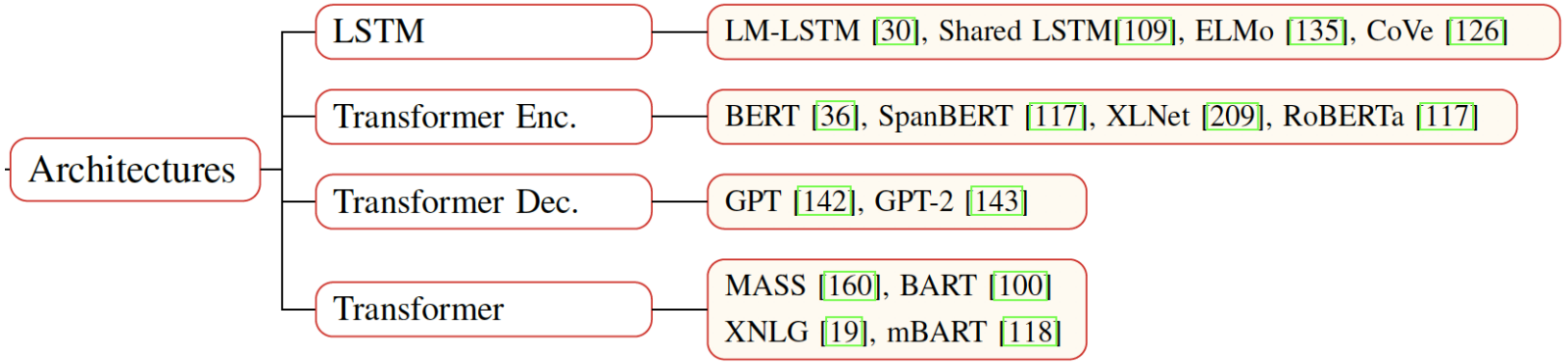
Contextual?

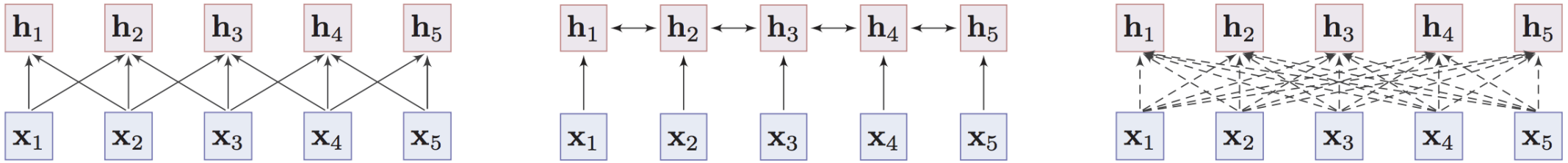


- 第一代 PTM 旨在学习词嵌入。由于下游的任务不再需要这些模型的帮助，因此为了计算效率，它们通常采用浅层模型，如 Skip-Gram 和 GloVe。尽管这些经过预训练的嵌入向量也可以捕捉单词的语义，但它们却不受上下文限制，只是简单地学习「共现词频」。这样的方法明显无法理解更高层次的文本概念，如句法结构、语义角色、指代等等。
- 而第二代 PTM 专注于学习上下文的词嵌入，如 CoVe、ELMo、OpenAI GPT 以及 BERT。它们会学习更合理的词表征，这些表征囊括了词的上下文信息，可以用于问答系统、机器翻译等后续任务。另一层面，这些模型还提出了各种语言任务来训练 PTM，以便支持更广泛的应用，因此它们也可以称为预训练语言模型。



主干网络



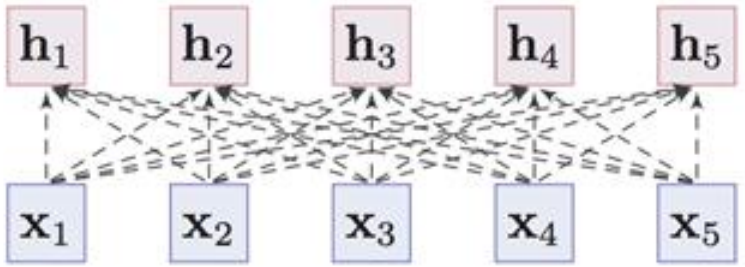


(a) Convolutional Model (b) Recurrent Model (c) Fully-Connected Self-Attention Model

- 对于预训练，Transformer 已经被证实是一个高效的架构。然而 Transformer 最大的局限在于其计算复杂度（输入序列长度的平方倍）。受限于 GPU 显存大小，目前大多数 PTM 无法处理超过 512 个 token 的序列长度。打破这一限制需要改进 Transformer 的结构设计，例如 Transformer-XL。因此，寻找更高效的 PTM 架构设计对于捕捉长距上下文信息十分重要。设计深层神经网络结构很有挑战，或许使用如神经结构搜索 (NAS) 这类自动结构搜索方法不失为一种好的选择。

自然语言处理（NLP）中的预训练模型

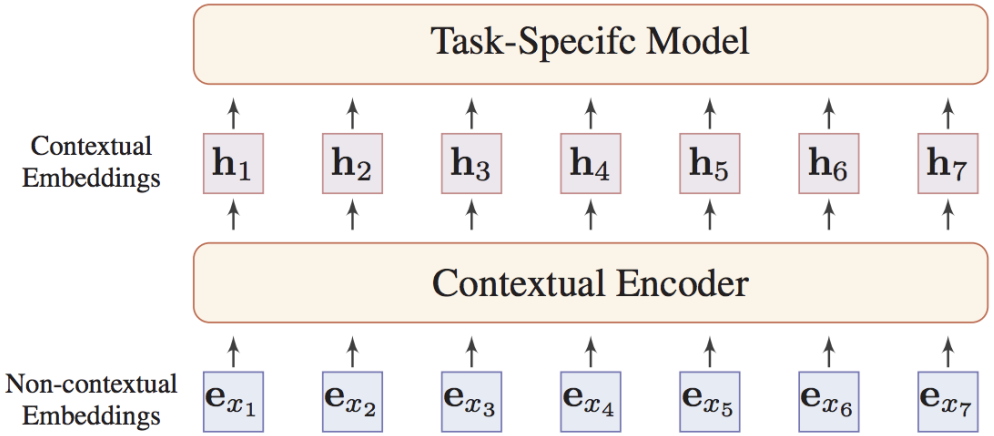
- 通过自编码方案可以有效的生成大规模的训练集
  - Mask-predict任务，noise-recover任务(auto-encoder)…
- 自注意力机制提供了句子内词汇相关关系特征的提取方式



(c) Fully-Connected Self-Attention Model

- 足够深的网络层能有效的分解合成复杂的语言特征
  - 短语、句法结构、语义结构等
- 通过后接任务迁移训练，实现特定任务的建模。

  实践表明，预训练模型在NLP领域有着很好的表现



BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

Key Words : Pre-training、Deep、Bidirectional、Transformer、Language Understanding

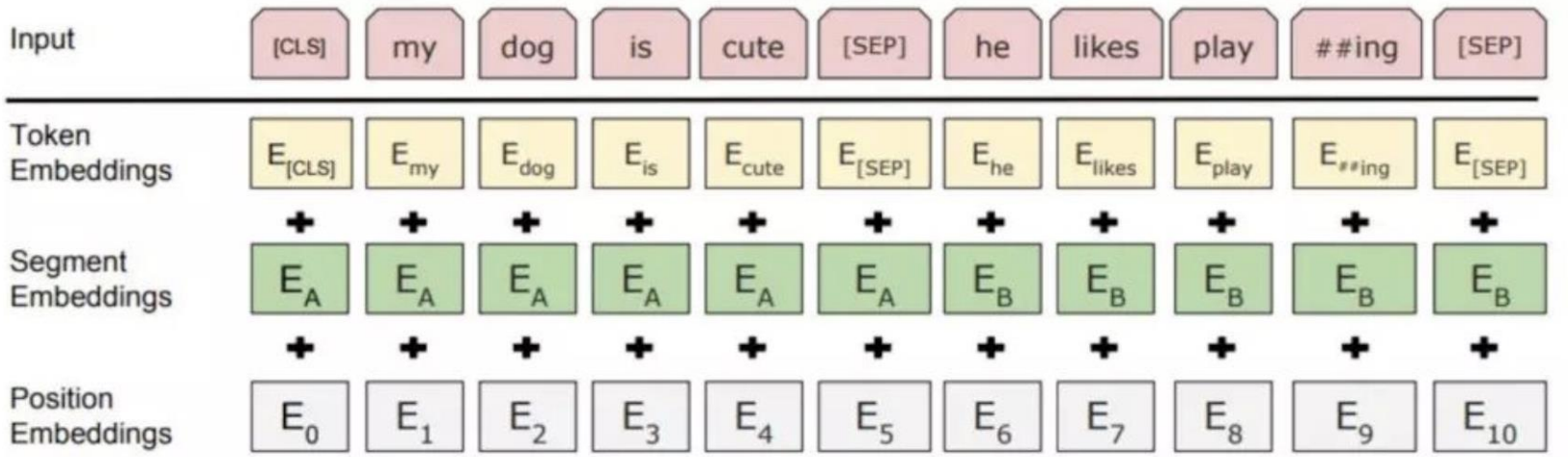
pre-training:确实存在通用的语言模型，先用文章预训练通用模型，然后再根据具体应用，用 supervised 训练数据，精加工（fine tuning）模型，使之适用于具体应用。

Deep : 模型非常深，12层，中间层是1024，而之前的Transformer-base模型中间层有2048, 深度为64。

Bi-directional: 一般的双向语言模型把从前往后，与从后往前的两个预测，拼接在一起 [mask1/mask2]，更好的办法是用上下文全向来预测[mask]，也就是用 “能/实现/语言/表征/../的/模型”，来预测[mask]。BERT 作者把上下文全向的预测方法，称之为 deep bi-directional.

Transformer: bert整体是使用Transformer Encoder实现的

输入模块



- （1）使用WordPiece嵌入（Wu et al., 2016）和30,000个token的词汇表。用##表示分词。
  （2）使用学习的positional embeddings，支持的序列长度最多为512个token。
  每个序列的第一个token始终是特殊分类嵌入（[CLS]）。对应于该token的最终隐藏状态（即，Transformer的输出）被用作分类任务的聚合序列表示。对于非分类任务，将忽略此向量。
  （3）句子对被打包成一个序列。以两种方式区分句子。首先，用特殊标记（[SEP]）将它们分开。其次，添加一个learned sentence A嵌入到第一个句子的每个token中，一个sentence B嵌入到第二个句子的每个token中。
  （4）对于单个句子输入，只使用 sentence A嵌入

预训练任务

- Masked Language Model
  - 随机屏蔽（masking）15%的输入token，然后只预测那些被屏蔽的token,其中：
    - 80%：用[MASK]标记替换单词，例如，my dog is hairy → my dog is [MASK]
      10%：用一个随机的单词替换该单词，例如，my dog is hairy → my dog is apple
      10%：保持单词不变，例如，my dog is hairy → my dog is hairy. 这样做的目的是将表示偏向于实际观察到的单词。
- Next Sentence Prediction
  - 为了训练一个理解整句的模型，预先训练一个二进制化的下一句测任务，这一任务可以从任何单语语料库中生成。具体地说，当选择句子A和B作为预训练样本时，B有50%的可能是A的下一个句子，也有50%的可能是来自语料库的随机句子

Input = [CLS] the man went to [MASK] store [SEP]

he bought a gallon [MASK] milk [SEP]

Label = IsNext

Input = [CLS] the man [MASK] to the store [SEP]

penguin [MASK] are flight ##less birds [SEP]

Label = NotNext

参考

Pre-trained Models for Natural Language Processing: A Survey

Xipeng Qiu*, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai & Xuanjing Huang

School of Computer Science, Fudan University, Shanghai 200433, *China;*
Shanghai Key Laboratory of Intelligent Information Processing, Shanghai 200433, *China*

Recently, the emergence of pre-trained models (PTMs)* has brought natural language processing (NLP) to a new era. In this survey, we provide a comprehensive review of PTMs for NLP. We first briefly introduce language representation learning and its research progress. Then we systematically categorize existing PTMs based on a taxonomy from four different perspectives. Next, we describe how to adapt the knowledge of PTMs to downstream tasks. Finally, we outline some potential directions of PTMs for future research. This survey is purposed to be a hands-on guide for understanding, using, and developing PTMs for various NLP tasks.

Deep Learning, Neural Network, Natural Language Processing, Pre-trained Model, Distributed Representation, Word Embedding, Self-Supervised Learning, Language Modelling

RoBERTa: A Robustly Optimized BERT Pretraining Approach

Yinhan Liu[*§] **Myle Ott**[*§] **Naman Goyal**[*§] **Jingfei Du**[*§] **Mandar Joshi**[†]
Danqi Chen[§] **Omer Levy**[§] **Mike Lewis**[§] **Luke Zettlemoyer**[†§] **Veselin Stoyanov**[§]

[†] Paul G. Allen School of Computer Science & Engineering,
University of Washington, Seattle, WA
{mandar90,lsz}@cs.washington.edu

[§] Facebook AI
{yinhanliu,myleott,naman,jingfeidu,
danqi,omerlevy,mikelewis,lsz,ves}@fb.com

Abstract

Language model pretraining has led to significant performance gains but careful comparison between different approaches is challenging. Training is computationally expensive, often done on private datasets of different sizes, and, as we will show, hyperparameter choices have significant impact on the final results. We present a replication study of BERT pretraining (Devlin et al., 2019) that carefully measures the impact of many key hyperparameters and training data size. We find that BERT was significantly undertrained, and can match or exceed the performance of every model published after it. Our best model achieves state-of-the-art results on GLUE, RACE and SQuAD. These results highlight the importance of previously overlooked design choices, and raise questions about the source of recently reported improvements. We release our models and code.[1]

We present a replication study of BERT pretraining (Devlin et al., 2019), which includes a careful evaluation of the effects of hyperparmeter tuning and training set size. We find that BERT was significantly undertrained and propose an improved recipe for training BERT models, which we call RoBERTa, that can match or exceed the performance of all of the post-BERT methods. Our modifications are simple, they include: (1) training the model longer, with bigger batches, over more data; (2) removing the next sentence prediction objective; (3) training on longer sequences; and (4) dynamically changing the masking pattern applied to the training data. We also collect a large new dataset (CC-NEWS) of comparable size to other privately used datasets, to better control for training set size effects.

When controlling for training data, our improved training procedure improves upon the published BERT results on both GLUE and SQuAD.

- 训练时间更长
- batch size更大
- 训练数据更多
- 动态调参

2.roberta-wwm

2.1 wwm策略介绍

Whole Word Masking (wwm)，暂翻译为 `全词Mask` 或 `整词Mask`，是谷歌在2019年5月31日发布的一项BERT的升级版本，主要更改了原预训练阶段的训练样本生成策略。

在原BERT、RoBERTa中，由于是**英语**语料，切词采用的是**WordPiece**, 是比词更小的粒度，词根切词法，如predict 这个词被切分成pre、##di、##ct三个token。这种切分的方式能有效的减小预训练中词典|V|的大小，缓解未见词问题（可由词根组装），能有效表达更多的词。但中文语料是不存在这种词根的说法的。后续就有采用了一种改进此问题的措施，**整词的mask方法--Whole word masking（wwm）**，如predict 这个词被切分成pre、##di、##ct三个token，则将三个token全部mask，这也是后续的bert-wwm、roberta-wwm版本。

需要注意的是，这里的mask指的是广义的mask（替换成[MASK]；保持原词汇；随机替换成另外一个词），并非只局限于单词替换成 `[MASK]` 标签的情况。

同理，由于谷歌官方发布的 `BERT-base, Chinese` 中，中文是以**字**为粒度进行切分，没有考虑到传统NLP中的中文分词（CWS）。 我们将全词Mask的方法应用在了中文中，使用了中文维基百科（包括简体和繁体）进行训练，并且使用了哈工大LTP作为分词工具，即对组成同一个**词**的汉字全部进行Mask。下述文本展示了 `全词Mask` 的生成样例。 **注意：为了方便理解，下述例子中只考虑替换成[MASK]标签的情况。**

TRANSFORMER

https://pytorch.org/docs/stable/generated/torch.nn.Transformer.html

CLASS torch.nn.Transformer(*d_model=512, nhead=8, num_encoder_layers=6, num_decoder_layers=6, dim_feedforward=2048, dropout=0.1, activation=<function relu>, custom_encoder=None, custom_decoder=None, layer_norm_eps=1e-05, batch_first=False, norm_first=False, bias=True, device=None, dtype=None*) [SOURCE]

A transformer model. User is able to modify the attributes as needed. The architecture is based on the paper "Attention Is All You Need". Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Advances in Neural Information Processing Systems, pages 6000-6010.

Parameters

- **d_model** (*int*) – the number of expected features in the encoder/decoder inputs (default=512).
- **nhead** (*int*) – the number of heads in the multiheadattention models (default=8).
- **num_encoder_layers** (*int*) – the number of sub-encoder-layers in the encoder (default=6).
- **num_decoder_layers** (*int*) – the number of sub-decoder-layers in the decoder (default=6).
- **dim_feedforward** (*int*) – the dimension of the feedforward network model (default=2048).
- **dropout** (*float*) – the dropout value (default=0.1).
- **activation** (*Union*[*str*, *Callable*[[*Tensor*], *Tensor*]]) – the activation function of encoder/decoder intermediate layer, can be a string ("relu" or "gelu") or a unary callable. Default: relu
- **custom_encoder** (*Optional*[*Any*]) – custom encoder (default=None).
- **custom_decoder** (*Optional*[*Any*]) – custom decoder (default=None).

PYTORCH-TRANSFORMERS

https://pytorch.org/hub/huggingface_pytorch-transformers/

Usage

The available methods are the following:

- `config`: returns a configuration item corresponding to the specified model or pth.
- `tokenizer`: returns a tokenizer corresponding to the specified model or path
- `model`: returns a model corresponding to the specified model or path
- `modelForCausalLM`: returns a model with a language modeling head corresponding to the specified model or path
- `modelForSequenceClassification`: returns a model with a sequence classifier corresponding to the specified model or path
- `modelForQuestionAnswering`: returns a model with a question answering head corresponding to the specified model or path

Model Description

PyTorch-Transformers (formerly known as `pytorch-pretrained-bert`) is a library of state-of-the-art pre-trained models for Natural Language Processing (NLP).

The library currently contains PyTorch implementations, pre-trained model weights, usage scripts and conversion utilities for the following models:

1. BERT (from Google) released with the paper BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding by Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova.
2. GPT (from OpenAI) released with the paper Improving Language Understanding by Generative Pre-Training by Alec Radford, Karthik Narasimhan, Tim Salimans and Ilya Sutskever.
3. GPT-2 (from OpenAI) released with the paper Language Models are Unsupervised Multitask Learners by Alec Radford, *Jeffrey Wu*, Rewon Child, David Luan, Dario Amodei** and Ilya Sutskever**.
4. Transformer-XL (from Google/CMU) released with the paper Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context by Zihang Dai, *Zhilin Yang*, Yiming Yang, Jaime Carbonell, Quoc V. Le, Ruslan Salakhutdinov.
5. XLNet (from Google/CMU) released with the paper XLNet: Generalized Autoregressive Pretraining for Language Understanding by Zhilin Yang, *Zihang Dai*, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, Quoc V. Le.
6. XLM (from Facebook) released together with the paper Cross-lingual Language Model Pretraining by Guillaume Lample and Alexis Conneau.
7. RoBERTa (from Facebook), released together with the paper a Robustly Optimized BERT Pretraining Approach by Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, Veselin Stoyanov.
8. DistilBERT (from HuggingFace), released together with the blogpost Smaller, faster, cheaper, lighter: Introducing DistilBERT, a distilled version of BERT by Victor Sanh, Lysandre Debut and Thomas Wolf.

TOKENIZER

The tokenizer object allows the conversion from character strings to tokens understood by the different models. Each model has its own tokenizer, and some tokenizing methods are different across tokenizers. The complete documentation can be found here.

```
import torch
tokenizer = torch.hub.load('huggingface/pytorch-transformers', 'tokenizer', 'bert-base-unc
tokenizer = torch.hub.load('huggingface/pytorch-transformers', 'tokenizer', './test/bert_s
```

MODELS

The model object is a model instance inheriting from a `nn.Module`. Each model is accompanied by their saving/loading methods, either from a local file or directory, or from a pre-trained configuration (see previously described `config`). Each model works differently, a complete overview of the different models can be found in the documentation.

```
import torch
model = torch.hub.load('huggingface/pytorch-transformers', 'model', 'bert-base-uncased')
model = torch.hub.load('huggingface/pytorch-transformers', 'model', './test/bert_model/')
model = torch.hub.load('huggingface/pytorch-transformers', 'model', 'bert-base-uncased', o
assert model.config.output_attentions == True
# Loading from a TF checkpoint file instead of a PyTorch model (slower)
config = AutoConfig.from_json_file('./tf_model/bert_tf_model_config.json')
model = torch.hub.load('huggingface/pytorch-transformers', 'model', './tf_model/bert_tf_ch
```

零、Setup

1）安装一个非常轻量级的 Transformers

```
!pip install transformers
```

```
import transformers
transformers.__version__
```

```
'4.35.2'
```

pip install --upgrade transformers==4.36.2

https://zhuanlan.zhihu.com/p/448852278

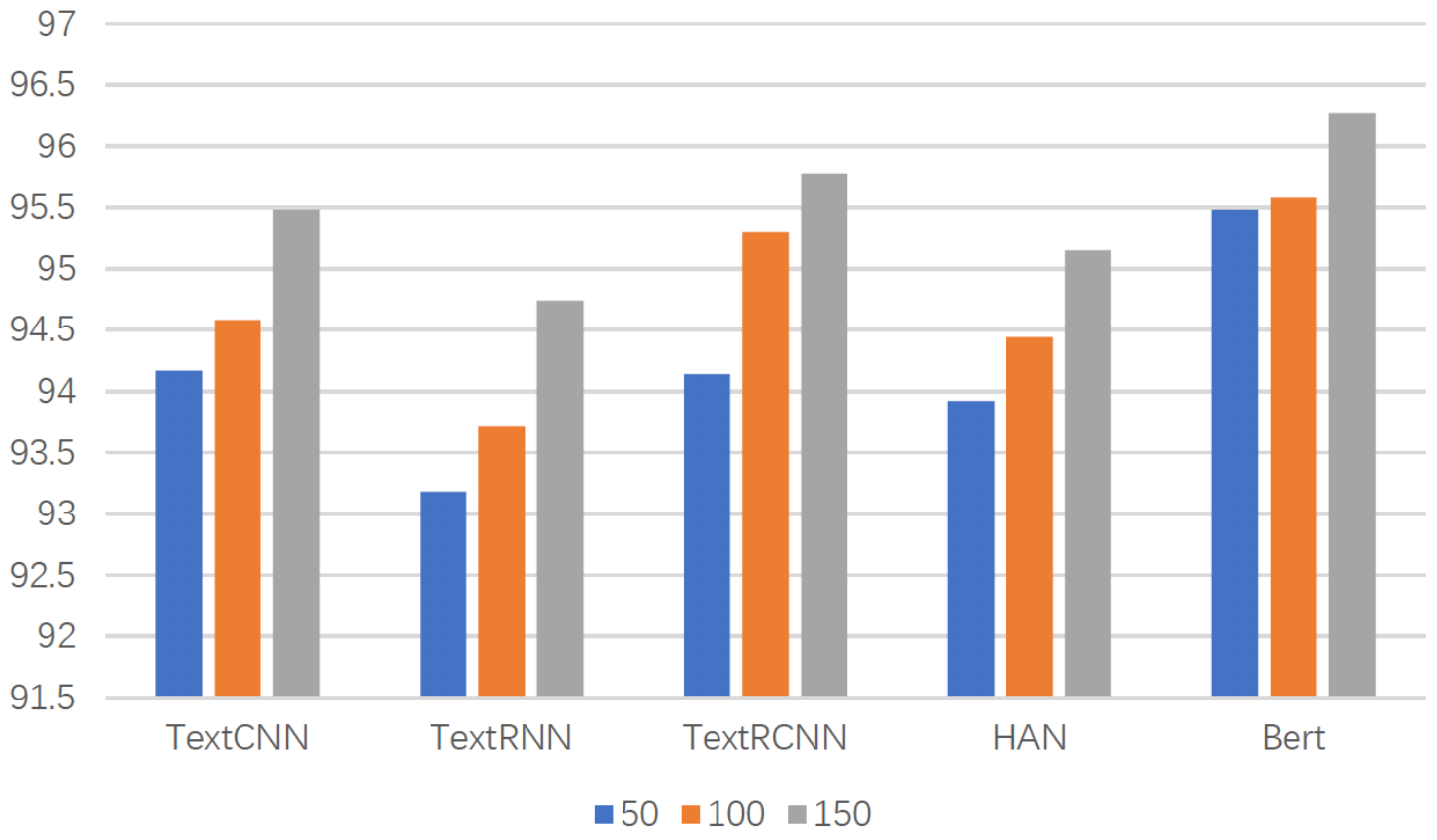
1. pipelines 简单的小例子

> Transformers 库中最基本的对象是 `pipeline()` 函数。它将模型与其必要的预处理和后处理步骤连接起来，使我们能够**直接输入任何文本并获得答案**：

当第一次运行的时候，它会下载预训练模型和分词器(tokenizer)并且缓存下来。

```
from transformers import pipeline

classifier = pipeline("sentiment-analysis")  # 情感分析
classifier("I've been waiting for a HuggingFace course my whole life.")

# 输出
# [{'label': 'POSITIVE', 'score': 0.9598047137260437}]
```

任务综合效果:


97
96.5
96
95.5
95
94.5
94
93.5
93
92.5
92
91.5
TextCNN
TextRNN
TextRCNN
HAN
Bert
50
100
150

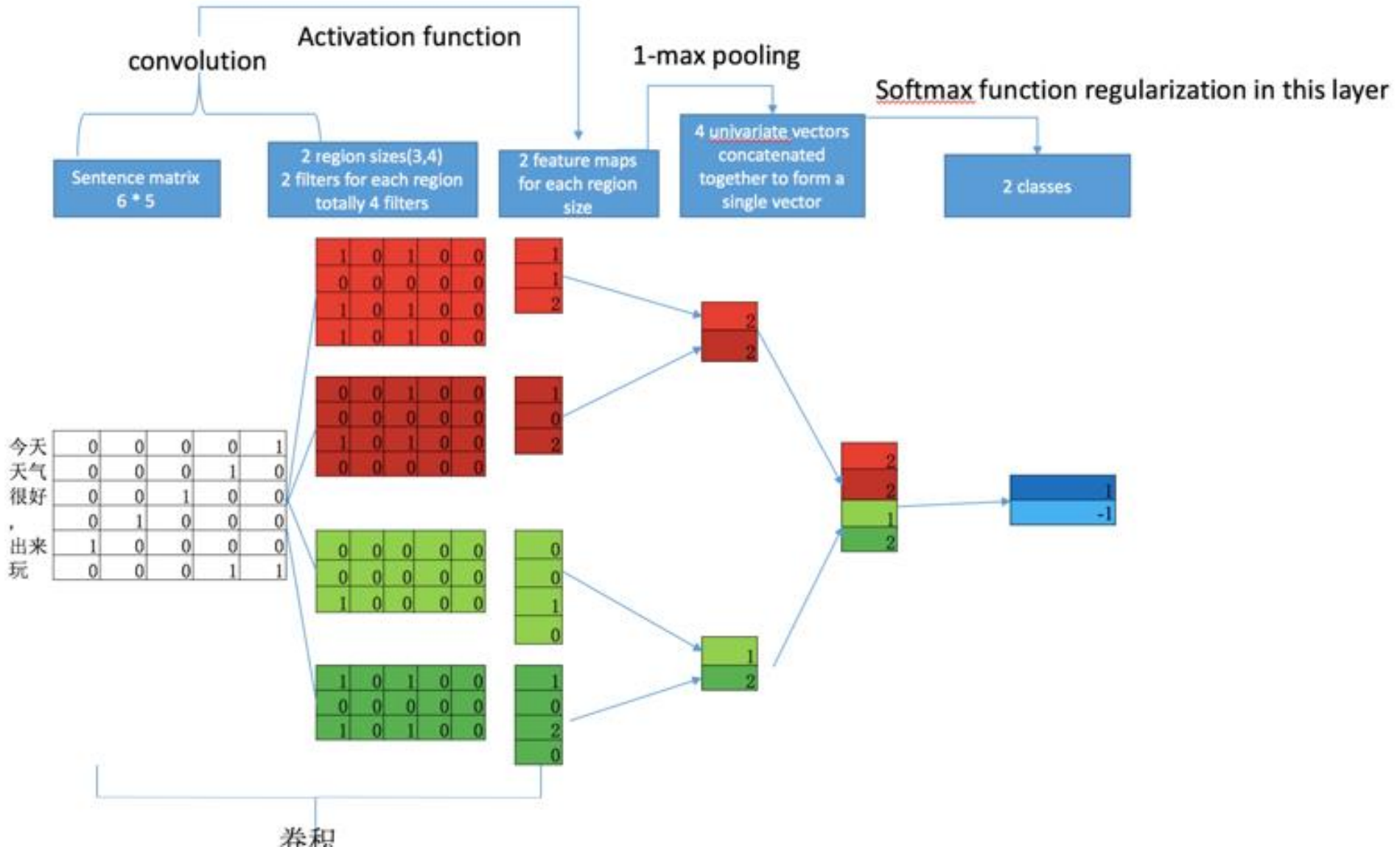
BERT的使用参考

- Tensorflow: https://github.com/google-research/bert
- Pytorch: https://github.com/huggingface/transformers
  - 集成了各类Transformer-based的预训练模型
  - 具体的例子在examples/目录下,
  - Token classification任务, token tagging类任务
    https://github.com/huggingface/transformers/tree/master/examples/token-classification
  - Text classification, 文本分类任务
  - Entailment, 文本蕴含推理

```
{
    "data_dir": ".",
    "labels": "./labels.txt",
    "model_name_or_path": "bert-base-multilingual-cased",
    "output_dir": "germeval-model",
    "max_seq_length": 128,
    "num_train_epochs": 3,
    "per_device_train_batch_size": 32,
    "save_steps": 750,
    "seed": 1,
    "do_train": true,
    "do_eval": true,
    "do_predict": true
}
```

t must be saved with a `.json` extension and can be used by running `python3 run_ner.py config.json`

Textcnn结构


convolution
Activation function
1-max pooling
Softmax function regularization in this layer
Sentence matrix
6 * 5
2 region sizes(3,4)
2 filters for each region
totally 4 filters
2 feature maps
for each region
size
4 univariate vectors
concatenated
together to form a
single vector
2 classes
今天
天气
很好
,
出来
玩
卷积

```
CLASS torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
      dilation=1, groups=1, bias=True, padding_mode='zeros')
```

in_channels —— 输入的channels数

out_channels —— 输出的channels数

kernel_size ——卷积核的尺寸，可以是方形卷积核、也可以不是，下边example可以看到

stride —— 步长，用来控制卷积核移动间隔

padding ——输入边沿扩边操作

padding_mode ——扩边的方式

bias ——是否使用偏置(即out = wx+b中的b)

```
#textcnn
class Mish(nn.Module):
    def __init__(self):
        super(Mish, self).__init__()
    def forward(self,x):
        x = x * (torch.tanh(F.softplus(x)))
        return x
class TextCNN(nn.Module): #定义TextCNN模型
    # TextCNN类继承了nn.Module类，在该类中定义的网络层列表必须要使用nn.ModuleList进行转化，才可以被TextCNN类识别。
    # 如果直接使用列表的话，在训练模型时无法通过TextCNN类对象的parameters方法获得权重。
    # 定义初始化方法
    def __init__(self, vocab_size, embedding_dim, n_filters, filter_sizes, output_dim, pad_idx,pretrained_embedding=None):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_dim, padding_idx = pad_idx) # 定义词向量权重
        if pretrained_embedding is not None:
            self.embedding.from_pretrained(pretrained_embedding,freeze=False)
        # 定义多分支卷积层
        # 将定义好的多分支卷积层以列表形式存放，以便在前向传播方法中使用。
        # 每个分支中卷积核的第一个维度由参数filter_sizes设置，第二个维度都是embedding_dim，即只在纵轴的方向上实现了真正的卷积操作，
        self.convs = nn.ModuleList([nn.Conv2d(in_channels = 1,out_channels = n_filters,kernel_size = (fs, embedding_dim))
                                                                for fs in filter_sizes])  #########注意不能用list
        # 定义输出层
        self.fc = nn.Linear(len(filter_sizes) * n_filters, output_dim)

        self.mish = Mish()  # 实例化激活函数对象
```