Convolutional Networks for text classification

Convolutional Networks

- A convolutional network is designed to identify indicative local indicators in a large structure, and combine them to produce a **fixed size** vector representation of the structure with a pooling function, capturing the local aspects that are most informative of the prediction task at hand.

- A convolutional network is not fully connected as a feedforward network is.

- It has been tremendously successful in image procession (or computer vision), where the input is the raw pixels of an image
- In NLP, it has been shown to be effective in sentence classification, etc.

Why it has been so effective in image processing

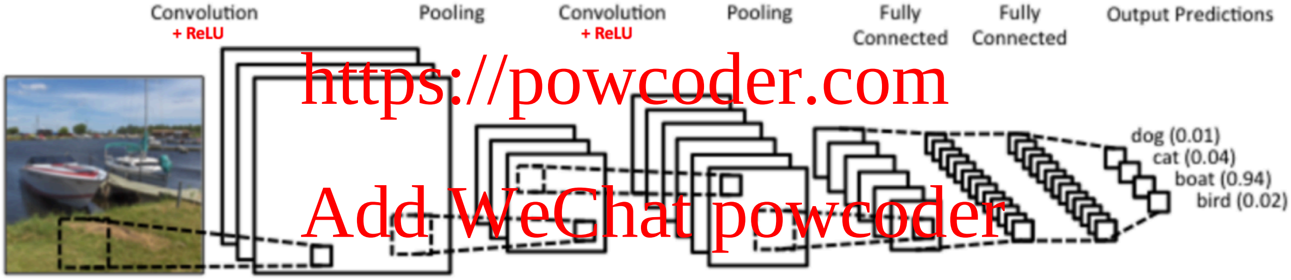
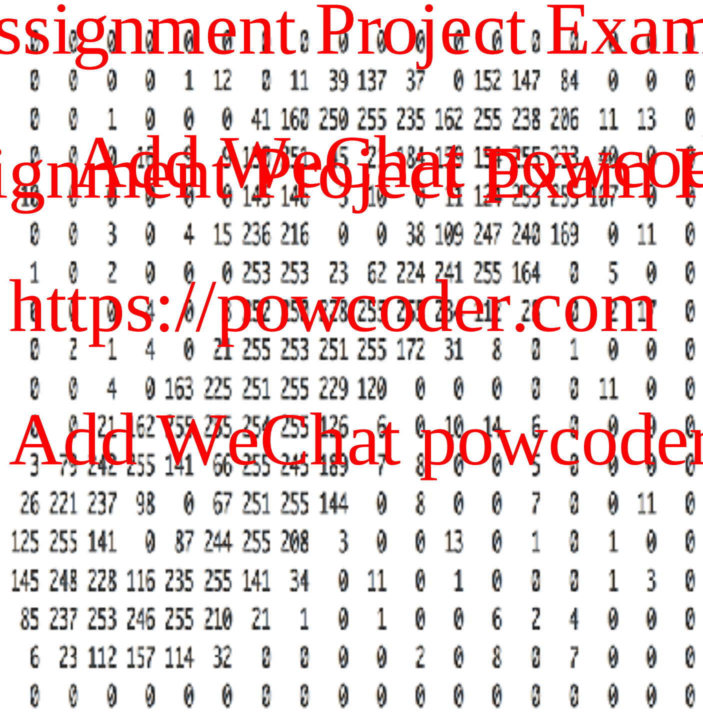


Image pixels

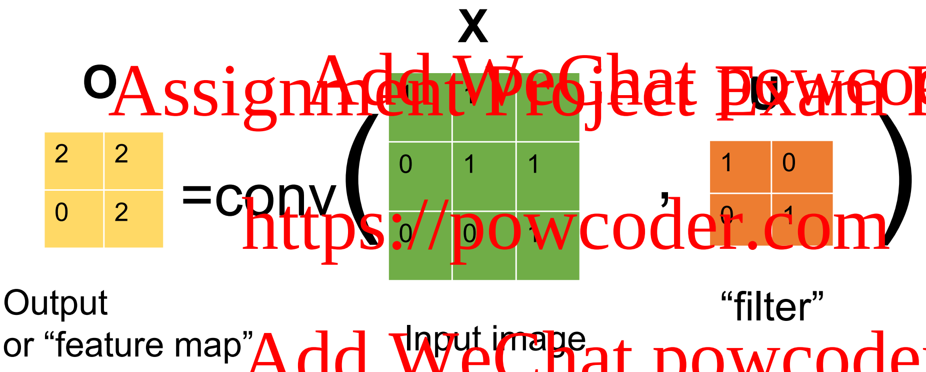


[illegible]	0	0	0	0	0	[illegible]	0	0	0	0	0	0	0	[illegible]	[illegible]	0	0
0	0	0	0	1	12	0	11	39	137	37	0	152	147	84	0	0	0
0	0	1	0	0	0	41	160	250	255	235	162	255	238	206	11	13	0
0	0	[illegible]	15	[illegible]	[illegible]	140	251	[illegible]	[illegible]	[illegible]	184	[illegible]	[illegible]	255	233	40	0
[illegible]	0	0	0	0	0	145	146	3	10	0	11	[illegible]	255	255	[illegible]	0	0
0	0	3	0	4	15	236	216	0	0	38	109	247	240	169	0	11	0
1	0	2	0	0	0	253	253	23	62	224	241	255	164	0	5	0	0
[illegible]	[illegible]	0	4	0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	0	[illegible]	[illegible]	0
0	2	1	4	0	21	255	253	251	255	172	31	8	0	1	0	0	0
0	0	4	0	163	225	251	255	229	120	0	0	0	0	0	11	0	0
[illegible]	0	21	162	255	255	254	255	126	6	0	10	14	6	0	0	0	0
3	79	242	255	141	66	255	245	189	7	8	0	0	5	0	0	0	0
26	221	237	98	0	67	251	255	144	0	8	0	0	7	0	0	11	0
125	255	141	0	87	244	255	208	3	0	0	13	0	1	0	1	0	0
145	248	228	116	235	255	141	34	0	11	0	1	0	0	0	1	3	0
85	237	253	246	255	210	21	1	0	1	0	0	6	2	4	0	0	0
6	23	112	157	114	32	0	0	0	0	2	0	8	0	7	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Four operations in a convolutional network

- Convolution
- Non-linear activation (ReLU)
- Pooling or subsampling (Max)
- Classification with a fully connected layer

Image convolution



Kernel size = (2,2), strides = 1

Image convolution

Kernel size = (2,2), strides = 1

Image convolution

Kernel size = (2,2), strides = 1

Image convolution

conv(X, U)

1	1	1
0x1	1x0	1
0x0	0x1	1

O

2	2
0	

Kernel size = (2,2), strides = 1

Image convolution

conv(X, U)

1	1	1
0	1x1	1x0
0	0x0	1x1

O

2	2
0	2

Kernel size = (2,2), strides = 1

Forward computation

$$o_{11} = x_{11}u_{11} + x_{12}u_{12} + x_{21}u_{21} + x_{22}u_{22}$$

$$o_{12} = x_{12}u_{11} + x_{13}u_{12} + x_{22}u_{21} + x_{23}u_{22}$$

$$o_{21} = x_{21}u_{11} + x_{22}u_{12} + x_{31}u_{21} + x_{32}u_{22}$$

$$o_{22} = x_{22}u_{11} + x_{23}u_{12} + x_{32}u_{21} + x_{33}u_{22}$$

ReLU

- Nonlinear transformation with ReLU

$$Output = ReLU(input) = \max(0, input)$$

- As we know, ReLU is an element-wise transformation that does not change the dimension of the feature map

- ReLU replaces all negative pixel values in the featuremap with 0

Image ReLU


Input Feature Map
Rectified Feature Map
ReLU
Black = negative; white = positive values
Only non-negative values

ReLU activation and Max pooling

- ReLU activation is a component-wise function and does not change the dimension of the input

$$\begin{bmatrix} 2 & 2 \\ 0 & 2 \end{bmatrix} = ReLU\left(\begin{bmatrix} 2 & 2 \\ 0 & 2 \end{bmatrix}\right)$$

- Max pooling does change the dimension of the input. Need to specify the pool size and strides.

$$\begin{bmatrix} 2 \end{bmatrix} = Max\left(\begin{bmatrix} 2 & 2 \\ 0 & 2 \end{bmatrix}\right)$$

$$\text{pool size} = (2, 2), \text{strides} = 2$$

Training a CNN

- Loss functions: Cross-entropy loss, squared error loss
- What are the parameters of a CNN?
  - The filters (kernels), weight matricies for the feedforward network on top of the convolution and pooling layers, biases
- Computing the gradient for the convolution layers is different from a feedforward neural network...

Computing the gradient on $\boldsymbol{U}$

$$
\begin{aligned}
\frac{\partial E}{\partial u_{11}} &= \frac{\partial E}{\partial o_{11}}\frac{\partial o_{11}}{\partial u_{11}} + \frac{\partial E}{\partial o_{12}}\frac{\partial o_{12}}{\partial u_{11}} + \frac{\partial E}{\partial o_{21}}\frac{\partial o_{21}}{\partial u_{11}} + \frac{\partial E}{\partial o_{22}}\frac{\partial o_{22}}{\partial u_{11}} \\
&= \frac{\partial E}{\partial o_{11}}x_{11} + \frac{\partial E}{\partial o_{12}}x_{12} + \frac{\partial E}{\partial o_{21}}x_{21} + \frac{\partial E}{\partial o_{22}}x_{22} \\
\frac{\partial E}{\partial u_{12}} &= \frac{\partial E}{\partial o_{11}}\frac{\partial o_{11}}{\partial u_{12}} + \frac{\partial E}{\partial o_{12}}\frac{\partial o_{12}}{\partial u_{12}} + \frac{\partial E}{\partial o_{21}}\frac{\partial o_{21}}{\partial u_{12}} + \frac{\partial E}{\partial o_{22}}\frac{\partial o_{22}}{\partial u_{12}} \\
&= \frac{\partial E}{\partial o_{11}}x_{12} + \frac{\partial E}{\partial o_{12}}x_{13} + \frac{\partial E}{\partial o_{21}}x_{22} + \frac{\partial E}{\partial o_{22}}x_{23} \\
\frac{\partial E}{\partial u_{21}} &= \frac{\partial E}{\partial o_{11}}\frac{\partial o_{11}}{\partial u_{21}} + \frac{\partial E}{\partial o_{12}}\frac{\partial o_{12}}{\partial u_{21}} + \frac{\partial E}{\partial o_{21}}\frac{\partial o_{21}}{\partial u_{21}} + \frac{\partial E}{\partial o_{22}}\frac{\partial o_{22}}{\partial u_{21}} \\
&= \frac{\partial E}{\partial o_{11}}x_{21} + \frac{\partial E}{\partial o_{12}}x_{22} + \frac{\partial E}{\partial o_{21}}x_{31} + \frac{\partial E}{\partial o_{22}}x_{32} \\
\frac{\partial E}{\partial u_{22}} &= \frac{\partial E}{\partial o_{11}}\frac{\partial o_{11}}{\partial u_{22}} + \frac{\partial E}{\partial o_{12}}\frac{\partial o_{12}}{\partial u_{22}} + \frac{\partial E}{\partial o_{21}}\frac{\partial o_{21}}{\partial u_{22}} + \frac{\partial E}{\partial o_{22}}\frac{\partial o_{22}}{\partial u_{22}} \\
&= \frac{\partial E}{\partial o_{22}}x_{11} + \frac{\partial E}{\partial o_{12}}x_{23} + \frac{\partial E}{\partial o_{21}}x_{32} + \frac{\partial E}{\partial o_{22}}x_{33}
\end{aligned}
$$

Summing up errors from all outputs that the filter component has contributed to.

Reverse Convolution

The computation of the gradient on the filter can be vectorized as a reverse convolution:

$$\begin{bmatrix} \frac{\partial E}{\partial u_{11}} & \frac{\partial E}{\partial u_{12}} \\ \frac{\partial E}{\partial u_{21}} & \frac{\partial E}{\partial u_{22}} \end{bmatrix} = conv\left(\begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix}, \begin{bmatrix} \frac{\partial E}{\partial o_{11}} & \frac{\partial E}{\partial o_{12}} \\ \frac{\partial E}{\partial o_{21}} & \frac{\partial E}{\partial o_{22}} \end{bmatrix} \right)$$

Computing the gradient on $\boldsymbol{X}$ (if this is not the input layer)

$$\frac{\partial E}{\partial x_{11}} = \frac{\partial E}{\partial o_{11}} u_{11} + \frac{\partial E}{\partial o_{12}} 0 + \frac{\partial E}{\partial o_{21}} 0 + \frac{\partial E}{\partial o_{22}} 0$$

$$\frac{\partial E}{\partial x_{12}} = \frac{\partial E}{\partial o_{11}} u_{12} + \frac{\partial E}{\partial o_{12}} u_{11} + \frac{\partial E}{\partial o_{21}} 0 + \frac{\partial E}{\partial o_{22}} 0$$

$$\frac{\partial E}{\partial x_{13}} = \frac{\partial E}{\partial o_{11}} 0 + \frac{\partial E}{\partial o_{12}} u_{12} + \frac{\partial E}{\partial o_{21}} 0 + \frac{\partial E}{\partial o_{22}} 0$$

$$\frac{\partial E}{\partial x_{21}} = \frac{\partial E}{\partial o_{11}} u_{21} + \frac{\partial E}{\partial o_{12}} 0 + \frac{\partial E}{\partial o_{21}} u_{11} + \frac{\partial E}{\partial o_{22}} 0$$

$$\frac{\partial E}{\partial x_{22}} = \frac{\partial E}{\partial o_{11}} u_{22} + \frac{\partial E}{\partial o_{12}} u_{21} + \frac{\partial E}{\partial o_{21}} u_{12} + \frac{\partial E}{\partial o_{22}} u_{11}$$

$$\frac{\partial E}{\partial x_{23}} = \frac{\partial E}{\partial o_{11}} 0 + \frac{\partial E}{\partial o_{12}} u_{22} + \frac{\partial E}{\partial o_{21}} 0 + \frac{\partial E}{\partial o_{22}} u_{12}$$

$$\frac{\partial E}{\partial x_{31}} = \frac{\partial E}{\partial o_{11}} 0 + \frac{\partial E}{\partial o_{12}} 0 + \frac{\partial E}{\partial o_{21}} u21 + \frac{\partial E}{\partial o_{22}} 0$$

$$\frac{\partial E}{\partial x_{32}} = \frac{\partial E}{\partial o_{11}} 0 + \frac{\partial E}{\partial o_{12}} 0 + \frac{\partial E}{\partial o_{21}} u_{22} + \frac{\partial E}{\partial o_{22}} u_{21}$$

$$\frac{\partial E}{\partial x_{12}} = \frac{\partial E}{\partial o_{11}} 0 + \frac{\partial E}{\partial o_{12}} 0 + \frac{\partial E}{\partial o_{21}} 0 + \frac{\partial E}{\partial o_{22}} u_{22}$$

Full convolution

$$\begin{bmatrix} \frac{\partial E}{\partial x_{11}} & \frac{\partial E}{\partial x_{12}} & \frac{\partial E}{\partial x_{13}} \\ \frac{\partial E}{\partial x_{21}} & \frac{\partial E}{\partial x_{22}} & \frac{\partial E}{\partial x_{23}} \\ \frac{\partial E}{\partial x_{31}} & \frac{\partial E}{\partial x_{32}} & \frac{\partial E}{\partial x_{33}} \end{bmatrix} = full\_conv\left(\begin{bmatrix} \frac{\partial E}{\partial o_{11}} & \frac{\partial E}{\partial o_{12}} \\ \frac{\partial E}{\partial o_{21}} & \frac{\partial E}{\partial o_{22}} \end{bmatrix}, \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \end{bmatrix} \right)$$

Gradient on $\boldsymbol{X}$ if it is not the inputs

Why convolutational networks for NLP?

- Even though bag-of-word models are simple and work well in some text classification tasks, they don't account for cases where multiple words combine to create meaning, such as "not interesting".
- The analogy with image processing is if the pixels are treated as separate features. (The analogy might be going too far).

Alternative text representations

- When using feed-forward networks for text classification, the input text is represented as a sparse vector that encodes bag-of-words features.

- Alternatively, a text can be represented as a sequence of word tokens $w_1, w_2, w_3, \cdots, w_M$. This view is useful for models such as **Convolutional Neural Networks**, or **ConvNets**, which processes text as a sequence.

- Each word token w_m is represented as a one-hot vector $\boldsymbol{e}_{w_m}$, with dimension V. The complete document can be represented by the horizontal concatenation of these one-hot vectors: $\boldsymbol{W} = [\boldsymbol{e}_{w_1}, \boldsymbol{e}_{w_2}, \cdots, \boldsymbol{e}_{w_m}] \in R^{V \times M}$.

- To show that this is equivalent to the bag-of-words model, we can recover the word count from the matrix-vector product $\boldsymbol{W}[1, 1, \cdots, 1]^\top \in R^V$.

Input to a convolutional network in a text classification task

When using conv nets for text classification, it is common to first “look up” the pretrained word embedding (e.g., the weight matrix produced by Word2Vec or GLOVE[1]) for each word in the sequence:

$$\boldsymbol{X}^{(0)} = \boldsymbol{\Theta}^{(x \to z)}[\boldsymbol{e}_{w_1}, \boldsymbol{e}_{w_2}, \cdots, \boldsymbol{e}_{w_M}]$$

$$\boldsymbol{X}^{(0)} \in \mathbb{R}^{K_e \times M}$$

where $\boldsymbol{e}_{w_m}$ is a column vector of zeros, with a 1 at position w_m, K_e is the size of embeddings

---

[1] https://nlp.stanford.edu/projects/glove

“Convolve” the input with a set of filters (kernels)

- A *filter* is a weight matrix of dimension $\boldsymbol{C}^{(k)} \in \mathbb{R}^{K_e \times h}$ where $\boldsymbol{C}^{(k)}$ is the kth filter. Note the first dimension of the filter is the same as the size of the embedding.
  - Unlike image processing, the filter doesn't have to cover the full width of the image.
- To merge adjacent words, we *convolve* $\boldsymbol{X}^{(0)}$ by sliding a set of filters across it:

$$\boldsymbol{X}^{(1)} = f(\boldsymbol{b} + \boldsymbol{C} * \boldsymbol{X}^{(0)})$$

where f is an activation function (e.g., tanh, ReLU), b is a vector of bias terms, and $*$ is the convolution operator.

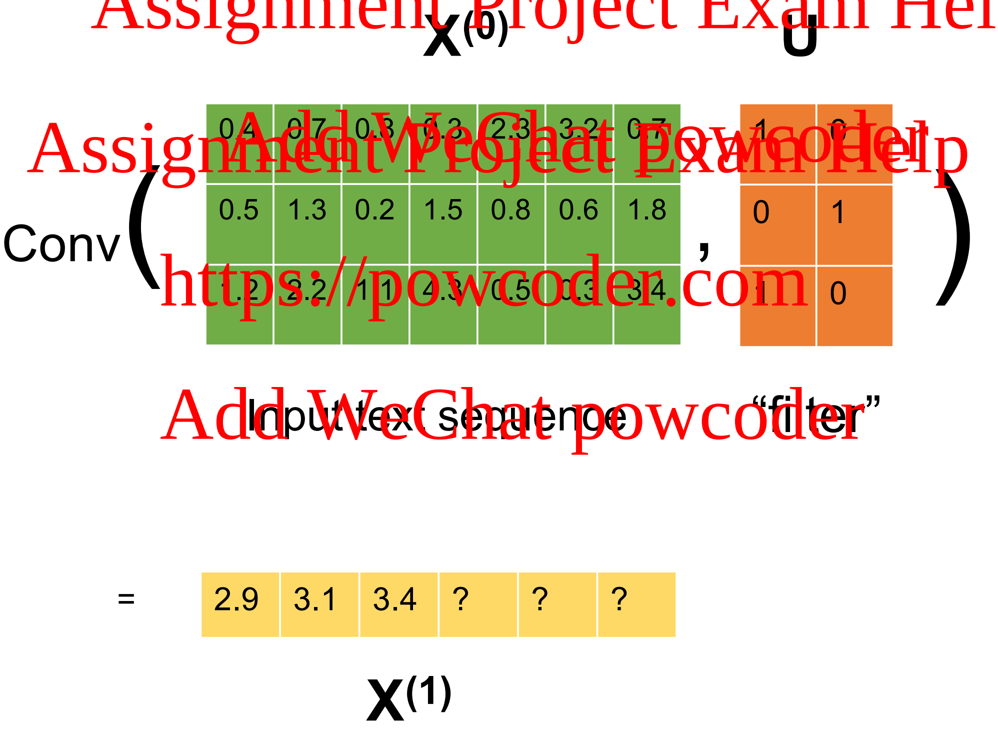
Computing the convolution

- At each position m (the mth word in the sequence), we compute the element-wise product of the kth filter and the sequence of words of window size h (think of it as an ngram of length h) starting at m and take its sum: $\boldsymbol{C}^{(k)} \odot \boldsymbol{X}^{(0)}_{m:m+h-1}$
- The value of the mth position with the kth filter can be computed as:

$$x_m^{(1)} = f\left(b_k + \sum_{k'=1}^{K_e} \sum_{n=1}^{h} c_{k',n}^{(k)} \times x_{k',m+n-1}^{(0)}\right)$$

- When we finish the convolution step, if we have K_f filters of dimension $\mathbb{R}^{K_e \times h}$, then $\boldsymbol{X}^{(1)} \in \mathbb{R}^{K_f \times M-h+1}$
- In practice, filters of different sizes are often used to captured ngrams of different lengths, so $\boldsymbol{X}^{(1)}$ will be K_f vectors of variable lengths, and we can write the size of each vector of h_k

Convolution step when processing text



$$\text{Conv}\left(\underbrace{\overset{\mathbf{X}^{(0)}}{\begin{bmatrix} 0.4 & 0.7 & 0.8 & 0.3 & 2.3 & 3.2 & 0.7 \\ 0.5 & 1.3 & 0.2 & 1.5 & 0.8 & 0.6 & 1.8 \\ 1.2 & 2.2 & 1.1 & 4.3 & 0.5 & 0.3 & 3.4 \end{bmatrix}}}_{\text{Input text sequence}}, \underbrace{\overset{\mathbf{U}}{\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}}}_{\text{“filter”}}\right)$$

$$= \underbrace{\begin{bmatrix} 2.9 & 3.1 & 3.4 & ? & ? & ? \end{bmatrix}}_{\mathbf{X}^{(1)}}$$

Padding

- To deal with the beginning and end of the input, the base matrix is often padded with $h-1$ column vectors of zeros at the beginning and end. this is called **wide convolution**
- If no padding is applied, then the output of each convolution layer will be $h-1$ units smaller than the input. This is known as **narrow convolution**.

Pooling

- After D convolutional layers, assuming filters have identical lengths, we have a representation of the document as a matrix $\boldsymbol{X}^{(D)} \in \mathbb{R}^{K_Z \times M}$.
- It is very likely that the documents will be of different lengths, so we need to turn them into matricies of the same length before feeding them to a feedward network to perform classification.
- This can done by **pooling** across times (over the sequence of words)

Prediction and training with CNN

- The CNN needs to be fed into a feedforward network to make a prediction $\hat{y}$ and compute the loss $\ell^{(i)}$ in training.

- Parameters of a CNN includes the weight mattics for the feedforward network and the filters $\mathbf{C}^{(k)}$ of the CNN, as well as the biases.

- The parameters can be updated with backpropagation, which may involve computing the gradient for the max pooling function.

$$\frac{\partial z_k}{\partial x_{k,m}^{(D)}} = \begin{cases} 1, & x_{k,m}^{(D)} = \max\left(x_{k,1}^{(D)}, x_{k,2}^{(D)}, \cdots, x_{k,M}^{(D)}\right) \\ 0, & \text{Otherwise} \end{cases}$$

Different pooling methods

- Max pooling

$$z_k = \max\left(x_{k,1}^{(D)}, x_{k,2}^{(D)}, \cdots, x_{k,M}^{(D)}\right)$$

- Average pooling

$$z_k = \frac{1}{M}\sum_{m=1}^{M} x_{k,m}^{(D)}$$

A graphic representation of a CNN

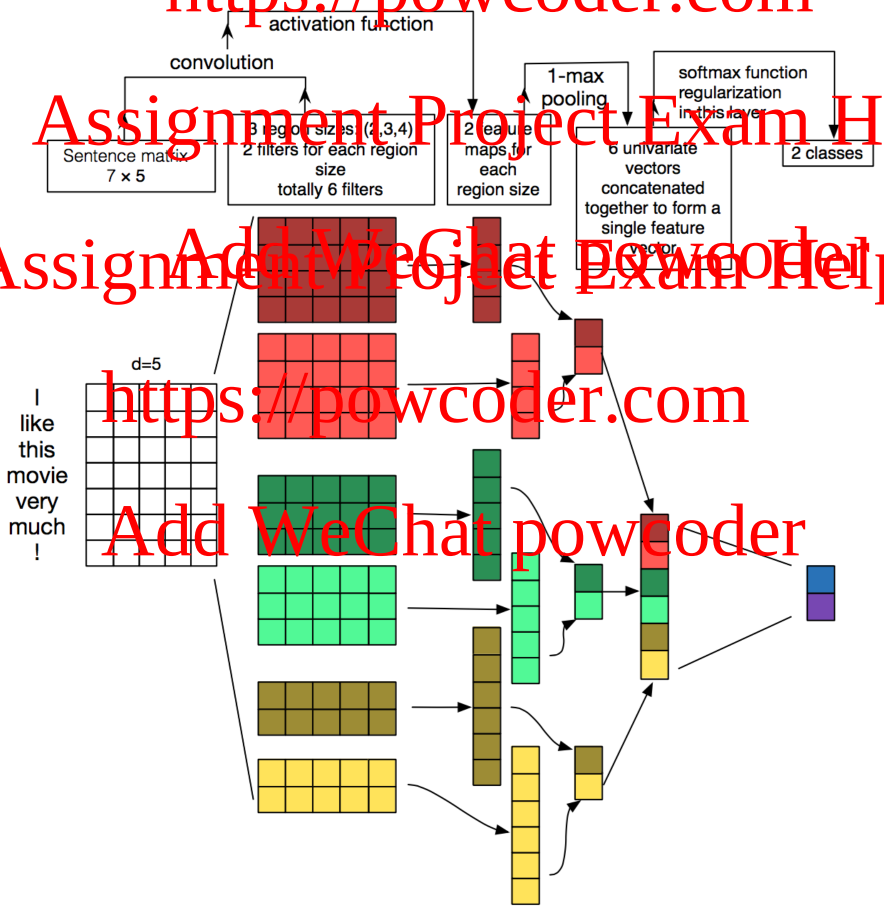
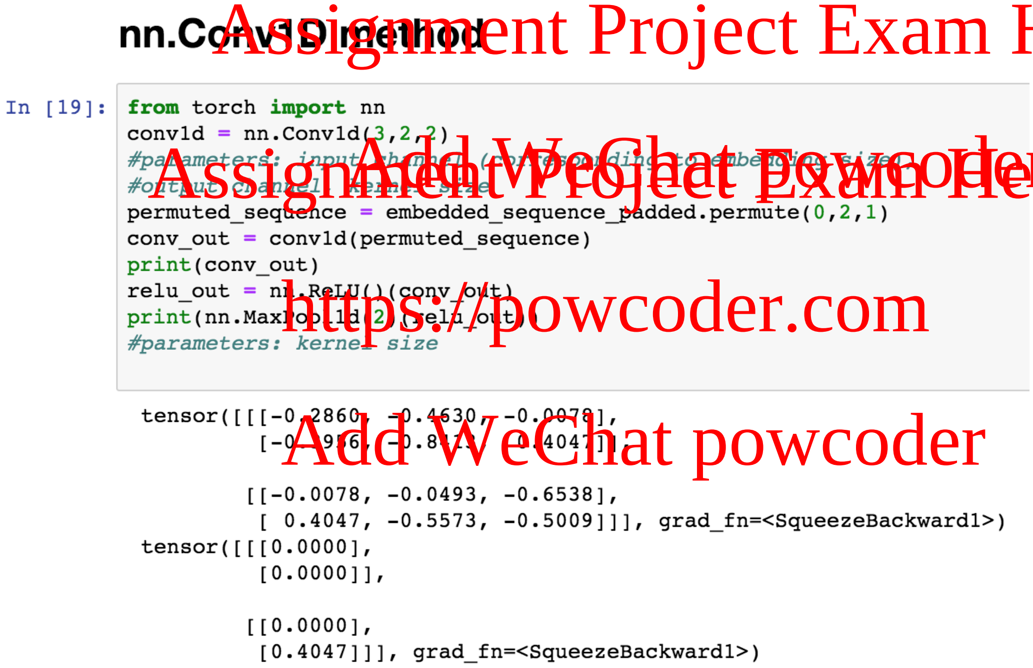


Figure 1: Caption

Using Conv Nets in PyTorch



nn.Conv1D method

In [19]:

```
from torch import nn
conv1d = nn.Conv1d(3,2,2)
#parameters: input channel (corresponding to embedding size),
#output channel, kernel size
permuted_sequence = embedded_sequence_padded.permute(0,2,1)
conv_out = conv1d(permuted_sequence)
print(conv_out)
relu_out = nn.ReLU()(conv_out)
print(nn.MaxPool1d(2)(relu_out))
#parameters: kernel size
```

```
tensor([[[-0.2860, -0.4630, -0.0078],
         [-0.2956, -0.8413,  0.4047]],

        [[-0.0078, -0.0493, -0.6538],
         [ 0.4047, -0.5573, -0.5009]]], grad_fn=<SqueezeBackward1>)
tensor([[[0.0000],
         [0.0000]],

        [[0.0000],
         [0.4047]]], grad_fn=<SqueezeBackward1>)
```