A Thorough Walk Through Deep Models

Xinyan Zhao
School of Information
University of Michigan

January 29, 2018

Abstract

This is an in-progress document to help people understand various deep learning models. So far, it only includes RNN, LSTM and CNN. More models are to be added!

1 Recurrent Neural Network

1.1 Basic Concepts

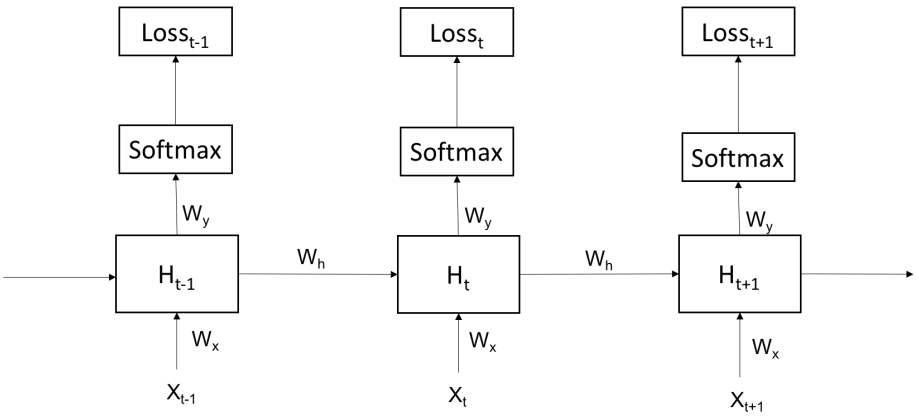

Figure 1: RNN model

Figure 1 shows a classic RNN model. Each X_t is t is a representation vector that represents a word. They can be one-hot vectors. Each H_t is called hidden state (also a vector) that the corresponding X_t will be transformed to. Softmax functions will transform the output of hidden states into a vector P_t that has the same dimension as X_t. Each entry in P_t represents the probability of the predicted word being the corresponding one-hot vector. W_x, W_h, W_y are coefficient matrices that we are trying to find.

Let's first make some notations:

$$
\begin{aligned}
&\text{(a) } Z_t = W_x X_t + W_h H_{t-1} \\
&\text{(b) } H_t = \tanh(Z_t) \\
&\text{(c) } Y_t = W_y H_t \\
&\text{(d) } P_t = Softmax(Y_t), \ \left(P_{t_i} = \frac{e^{Y_{t_i}}}{\Sigma_k e^{Y_{t_k}}} \right) \\
&\text{(e) } Loss_t = CrossEntropy(P_t, Label_t) = -\Sigma_j Label_{t_j} \log P_{t_j} \\
&\text{(f) } L = \Sigma_t Loss_t
\end{aligned}
\tag{1}
$$

X_t is input vector, H_t is hidden vector, P_t is predicted result, $Label_t$ is the true value.

1.2 Derivation

Now the do derrivations. Please note that during the derivation, I am being careless about the dimensions here. For example, sometime a derivative should be X^T, but I will just put X here. So if you are implementing the algorithm, be make sure the dimension matches!

Because of equation(f), we know that

$$
\begin{aligned}
\frac{\partial L}{\partial W_y} &= \frac{\partial \Sigma Loss_t}{\partial W_y} = \Sigma_t \frac{\partial Loss_t}{\partial W_y} \\
\frac{\partial L}{\partial W_y} &= \frac{\partial \Sigma Loss_t}{\partial W_h} = \Sigma_t \frac{\partial Loss_t}{\partial W_h} \\
\frac{\partial L}{\partial W_y} &= \frac{\partial \Sigma Loss_t}{\partial W_x} = \Sigma_t \frac{\partial Loss_t}{\partial W_x}
\end{aligned}
$$

So we only have to find all the derivatives of $Loss_t$, then when we implement the backpropagation algorithm, we only need to sum up all the derivatives at each step t.

When doing derivative in backpropagation, one trick to keep in mind is that our loss is a numeric value, which means that the result of the derivative of the loss w.r.t some variable should always has the same dimension of that variable. And when we do derivative of a vector w.r.t another vector, we will have a Jacobian matrix.

First let's do $\frac{\partial L_t}{\partial P_t}$.

$$
\frac{\partial L_t}{\partial P_t} = \begin{bmatrix} \vdots \\ \frac{\partial L_t}{\partial P_{t_j}} \\ \vdots \end{bmatrix}
$$

Thus, we have

$$\begin{aligned}\frac{\partial L_t}{\partial_{t_j}} &= \frac{\partial\left(-\Sigma_k Label_{t_k}\log P_{t_k}\right)}{\partial P_{t_j}} \\ &= -\frac{1}{P_{t_j}} Label_{t_j} \\ &= -\frac{Label_{t_j}}{P_{t_j}}\end{aligned}$$

Therefore, we have

$$\frac{\partial L_t}{\partial P_t} = \begin{bmatrix} \vdots \\ -\frac{Label_{t_j}}{P_{t_j}} \\ \vdots \end{bmatrix}$$

Next let's do $\frac{\partial L_t}{\partial Y_t} = \frac{\partial L_t}{\partial P_t}\frac{\partial P_t}{\partial Y_t}$.

Apparently, $\frac{\partial P_t}{\partial Y_t}$ is an Jacobian matrix, and $\frac{\partial P_t}{\partial Y_t} = \begin{bmatrix} \frac{\partial P_{t_1}}{\partial Y_{t_1}} & \frac{\partial P_{t_1}}{\partial Y_{t_2}} & \cdots \\ \frac{\partial P_{t_2}}{\partial Y_{t_1}} & \frac{\partial P_{t_2}}{\partial Y_{t_2}} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}$ or $\left(\frac{\partial P_t}{\partial Y_t}\right)_{ij} = \frac{\partial P_{t_i}}{\partial Y_{t_j}}$. There are two ways to do this.
Method 1:
When $i = j$,

$$\begin{aligned}\frac{\partial P_{t_i}}{\partial Y_{t_i}} &= \frac{e^{Y_{t_i}}\left(\Sigma_k e^{Y_{t_k}}\right) - e^{Y_{t_i}}\left(e^{Y_{t_i}}\right)}{\left(\Sigma_k e^{Y_{t_k}}\right)^2} \\ &= \frac{e^{Y_{t_i}}}{\Sigma_k e^{Y_{t_k}}} - \frac{e^{Y_{t_i}}}{\Sigma_k e^{Y_{t_k}}}\frac{e^{Y_{t_i}}}{\Sigma_k e^{Y_{t_k}}} \\ &= P_i \odot P_i P_i\end{aligned}$$

($\odot$ here indicates that we are doing element-wise production)
When $i \neq j$,

$$\frac{\partial P_{t_i}}{\partial Y_{t_i}} = \frac{-e^{Y_{t_i}} e^{Y_{t_j}}}{\left(\Sigma_k e^{Y_{t_k}}\right)^2} = -P_i \odot P_j$$

Thus we have

$$\frac{\partial P_t}{\partial Y_t} = \begin{bmatrix} P_{t_1} - P_{t_1} \odot P_{t_1} & P_{t_1} P_{t_2} & \cdots \\ P_{t_2} P_{t_1} & P_{t_2} - P_{t_2} \odot P_{t_2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}, \text{where } \left(\frac{\partial P_t}{\partial Y_t}\right)_{ij} = \begin{cases} P_{t_i} - P_{t_i} \odot P_{t_i} & \text{if } i = j, \\ P_{t_i} P_{t_j} & \text{else} \end{cases}$$

$$
\begin{aligned}
\frac{\partial L_t}{\partial Y_t} &= \frac{\partial L_t}{\partial P_t}\frac{\partial P_t}{\partial Y_t} \\
&= \begin{bmatrix} P_{t_1} - P_{t_1} \odot P_{t_1} & P_{t_1}P_{t_2} & \cdots \\ P_{t_2}P_{t_1} & P_{t_2} - P_{t_2} \odot P_{t_2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \begin{bmatrix} \vdots \\ -\frac{Label_{t_j}}{P_{t_j}} \\ \vdots \end{bmatrix} \\
&= \begin{bmatrix} \cdots & P_{t_{j-1}} & P_{t_j} - 1 & P_{t_{j+1}} & \cdots \end{bmatrix}^T \\
&= \begin{bmatrix} \cdots & P_{t_{j-1}} - 0 & P_{t_j} - 1 & P_{t_{j+1}} - 0 & \cdots \end{bmatrix}^T \\
&= P_t - Label_t
\end{aligned}
$$

(Note that in $Label_t$, since it's the true label, there is exactly one element which is 1, all the other elements are 0!)

Method 2:

$$
\begin{aligned}
\frac{\partial L_t}{\partial Y_{t_i}} &= \frac{-\partial \Sigma_k Label_{t_k} \log P_{t_k}}{\partial Y_{t_i}} \\
&= -\Sigma_k \frac{\partial Label_{t_k} \log P_{t_k}}{\partial Y_{t_i}} \\
&= -\Sigma_k Label_{t_k} \frac{1}{P_{t_k}} \frac{\partial P_{t_k}}{\partial Y_{t_i}} \\
&= -Label_{t_i} \frac{1}{P_{t_i}} \frac{\partial P_{t_i}}{\partial Label_{t_i}} - \Sigma_{k \neq i} \frac{Label_{t_k}}{P_{t_k}} \frac{\partial P_{t_k}}{\partial Y_{t_i}} \\
&= -\frac{Label_{t_i}}{P_{t_i}} \left(P_{t_i} - P_{t_i} \odot P_{t_i}\right) - \Sigma_{k \neq i} \frac{Label_{t_k}}{P_{t_k}} \left(-P_{t_k} \odot P_{t_i}\right) \\
&= -Label_{t_i} \left(1 - P_{t_i}\right) + \Sigma_{k \neq i} Label_{t_k} Y_{t_i} \\
&= -Label_{t_i} + Label_{t_i} P_{t_i} + \Sigma_{k \neq i} Label_{t_k} P_{t_i} \\
&= -Label_{t_i} + P_{t_i} \Sigma_k Label_{t_k} \\
&= P_{t_i} - Label_{t_i} (\text{because } \Sigma_k label_{t_k} = 1)
\end{aligned}
$$

Now we move on to compute other components!

$$
\frac{\partial L_t}{\partial W_y} = \frac{\partial L_t}{\partial Y_t}\frac{\partial Y_t}{\partial W_y} = \left(P_t - Label_t\right) H_t
$$

$$
\frac{\partial L_t}{\partial H_t} = \frac{\partial L_t}{\partial Y_t}\frac{\partial Y_t}{\partial H_t} = \left(P_t - Label_t\right) W_y
$$

Deriving $\frac{\partial L_t}{\partial W_h}$ is a bit different because W_y affects L_t only at time t and W_h affects L_t through $t, t-1, ..., 0$. However, the impact of W_h towards L_t at step $t-1$ can be computed when we move to step $t-1$, we just need to add the impact of W_h from step t to the step $t-1$, and so on and so forth. When we finally move to step 0, we would've added the impact of W_h from all steps.

Thus,

$$\begin{aligned}\frac{\partial L_t}{\partial W_h} &= \frac{\partial L_t}{\partial H_t}\frac{\partial H_t}{\partial W_h} \\ &= \frac{\partial L_t}{\partial H_t}\Sigma_{k=0}^{t}\frac{\partial H_t}{\partial H_k}\frac{\partial H_k}{\partial W_h} \\ &= \frac{\partial L_t}{\partial H_t}\Sigma_{k=0}^{t}\frac{\partial H_t}{\partial H_k}\frac{\partial H_k}{\partial Z_k}\frac{\partial Z_k}{\partial W_h}\end{aligned}$$

$$\begin{aligned}\frac{\partial L_t}{\partial W_x} &= \frac{\partial L_t}{\partial H_t}\frac{\partial H_t}{\partial W_x} \\ &= \frac{\partial L_t}{\partial H_t}\Sigma_{k=0}^{t}\frac{\partial H_t}{\partial H_k}\frac{\partial H_k}{\partial Z_k}\frac{\partial Z_k}{\partial W_x}\end{aligned}$$

To compute these derivatives when we implement the backpropagation, all we have to do are the following derivatives:

$$\frac{\partial H_t}{\partial Z_t} = 1 - \tanh^2(Z_t) = 1 - H_t H_t^T \text{ (Jacobian Matrix)}$$

$$\begin{aligned}\frac{\partial Z_t}{\partial Z_{t-1}} &= W_h \\ \frac{\partial Z_t}{\partial W_x} &= X_t \\ \frac{\partial Z_t}{\partial W_h} &= H_{t-1}\end{aligned}$$

2 Long-Short Term Memory

2.1 Basic Concepts

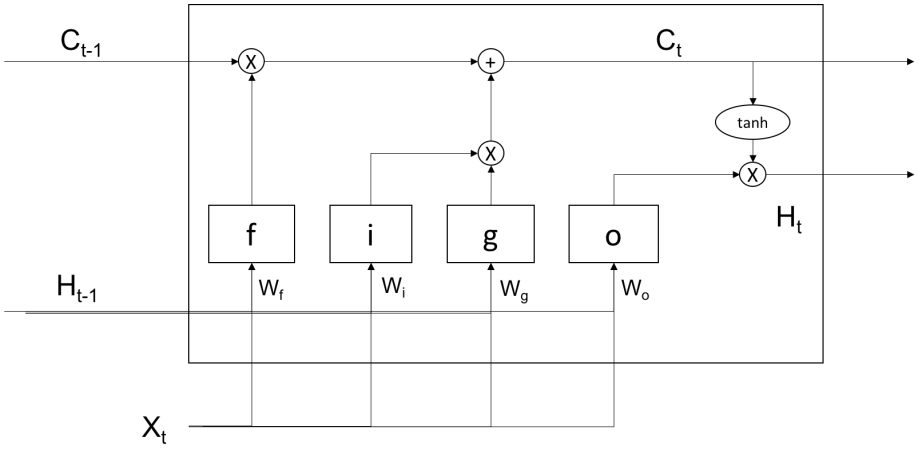

Figure 2: LSTM model

LSTM solves the gradient exploding/vanishing problem by setting up IFOG gates, namely, input gate, forget gate, output gate, memory gate (g gate).

Let's first make some notations:

$$
\begin{aligned}
&\text{(a)}\ f_t = W_f\left[H_{t-1}, X_t\right] + b_f \\
&\text{(b)}\ i_t = W_i\left[H_{t-1}, X_t\right] + b_i \\
&\text{(c)}\ o_t = W_o\left[H_{t-1}, X_t\right] + b_o \\
&\text{(d)}\ g_t = W_g\left[H_{t-1}, X_t\right] + b_g \\
&\text{(e)}\ f_t^* = \sigma(f_t) \\
&\text{(f)}\ i_t^* = \sigma(i_t) \\
&\text{(g)}\ o_t^* = \sigma(o_t) \\
&\text{(h)}\ g_t^* = \tanh(g_t) \\
&\text{(i)}\ C_t = f_t^* \odot C_{t-1} + i_t^* \odot g_t^* \\
&\text{(j)}\ H_t = o_t^* \odot \tanh(C_t) \\
&\text{(k)}\ Y_t = W_y H_t + b_y \\
&\text{(l)}\ P_t = Softmax(Y_t) \\
&\text{(m)}\ Loss_t = CrossEntropy(P_t, Label_t) = -\Sigma_j Label_{t_j} \log P_{tj}
\end{aligned}
\tag{2}
$$

As you can see, for each step t, we have two kinds of outputs, C_t (cell state) and H_t (hidden state). Current cell state is computed by using forget gate to forget some previous cell state and using input gate to input some current memory from memory gate. Current hidden state is computed by doing a tanh transformation on current cell state and let output gate to control how much we want to be current hidden state. $W_f, W_i, W_o, W_g, b_f b_i, b_o, b_g$ are parameters we want

to find!

The process of $H_t \to Y_t \to P_t \to L_t$ is the same as what we have in RNN. If you are confused, please refer to the RNN section.

2.2 Derivation

1. $\frac{\partial L_t}{\partial Y_t} = P_t - Label_t$ (same as in RNN).

2. $\frac{\partial L_t}{\partial W_y} = \frac{\partial L_t}{\partial Y_t}\frac{\partial Y_t}{\partial W_y} = (P_t - Label_t)H_t^T$

3. $\frac{\partial L_t}{\partial b_y} = \frac{\partial L_t}{\partial Y_t}\frac{\partial Y_t}{\partial b_y} = (P_t - Label_t)I$

4. $\frac{\partial L_t}{\partial H_t}$ is a bit different because H_t depends on all gates, current and previous cell states. Let's assume that we know $\frac{\partial L_t}{\partial H_t}$ for now.

5. $\frac{\partial L_t}{\partial o_t^*} = \frac{\partial L_t}{\partial H_t}\frac{\partial H_t}{\partial o_t^*} = \frac{\partial L_t}{\partial H_t}\tanh(C_t)$

6. $\frac{\partial L_t}{\partial C_t} = \frac{\partial L_t}{\partial H_t}\frac{\partial H_t}{\partial C_t} = \frac{\partial L_t}{\partial H_t}o_t^*(1 - \tanh^2(C_t))$

7. $\frac{\partial L_t}{\partial i_t^*} = \frac{\partial L_t}{\partial C_t} = \frac{\partial C_t}{\partial i_t^*} = \frac{\partial L_t}{\partial C_t} \odot g_t^*$.

8. $\frac{\partial L_t}{\partial f_t^*} = \frac{\partial L_t}{\partial C_t} \odot C_{t-1}$

9. $\frac{\partial L_t}{\partial g_t^*} = \frac{\partial L_t}{\partial C_t} \odot i_t^*$

10. $\frac{\partial L_t}{\partial i_t} = \frac{\partial L_t}{\partial i_t^*}\frac{\partial i_t^*}{\partial i_t} = \frac{\partial L_t}{\partial i_t^*} \odot i_t^* \odot (1 - i_t^*)$

11. $\frac{\partial L_t}{\partial o_t} = \frac{\partial L_t}{\partial o_t^*}\frac{\partial o_t^*}{\partial o_t} = \frac{\partial L_t}{\partial o_t^*} \odot o_t^* \odot (1 - o_t^*)$

12. $\frac{\partial L_t}{\partial f_t} = \frac{\partial L_t}{\partial f_t^*}\frac{\partial f_t^*}{\partial f_t} = \frac{\partial L_t}{\partial f_t^*} \odot f_t^* \odot (1 - f_t^*)$

13. $\frac{\partial L_t}{\partial g_t} = \frac{\partial L_t}{\partial f_g^*}\frac{\partial g_t^*}{\partial g_t} = \frac{\partial L_t}{\partial g_t^*} \odot (1 - g_t^* \odot g_t^*)$

(Equation (10)-(12) used the fact that $\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$)

14. $\frac{\partial L_t}{\partial W_f} = \frac{\partial L_t}{\partial f_t}\frac{\partial f_t}{\partial W_f} = \frac{\partial L_t}{\partial f_t}X_t$

15. $\frac{\partial L_t}{\partial b_f} = \frac{\partial L_t}{\partial f_t}\frac{\partial f_t}{\partial b_f} = \frac{\partial L_t}{\partial f_t}X_t$

16. $\frac{\partial L_t}{\partial W_i} = \frac{\partial L_t}{\partial i_t}\frac{\partial i_t}{\partial W_i} = \frac{\partial L_t}{\partial i_t}X_t$

17. $\frac{\partial L_t}{\partial b_i} = \frac{\partial L_t}{\partial i_t}\frac{\partial i_t}{\partial b_i} = \frac{\partial L_t}{\partial i_t}X_t$

18. $\frac{\partial L_t}{\partial W_o} = \frac{\partial L_t}{\partial o_t}\frac{\partial o_t}{\partial W_o} = \frac{\partial L_t}{\partial o_t}X_t$

19. $\frac{\partial L_t}{\partial b_o} = \frac{\partial L_t}{\partial o_t}\frac{\partial o_t}{\partial b_o} = \frac{\partial L_t}{\partial o_t}X_t$

20. $\frac{\partial L_t}{\partial W_g} = \frac{\partial L_t}{\partial g_t}\frac{\partial g_t}{\partial W_g} = \frac{\partial L_t}{\partial g_t}X_t$

21. $\frac{\partial L_t}{\partial b_g} = \frac{\partial L_t}{\partial g_t}\frac{\partial g_t}{\partial b_g} = \frac{\partial L_t}{\partial g_t}X_t$

22.
$$\begin{aligned}\frac{\partial L_t}{\partial X_t} &= \frac{\partial L_t}{\partial f_t}\frac{\partial f_t}{\partial x_t} + \frac{\partial L_t}{\partial i_t}\frac{\partial i_t}{\partial x_t} + \frac{\partial L_t}{\partial o_t}\frac{\partial o_t}{\partial x_t} + \frac{\partial L_t}{\partial g_t}\frac{\partial g_t}{\partial x_t} \\ &= \frac{\partial L_t}{\partial f_t}W_f + \frac{\partial L_t}{\partial i_t}W_i + \frac{\partial L_t}{\partial o_t}W_o + \frac{\partial L_t}{\partial g_t}W_g\end{aligned}$$

The reason why we are summing up four components in $\frac{\partial L_t}{\partial X_t}$ is because the relation between X_t and L_t is something like this:

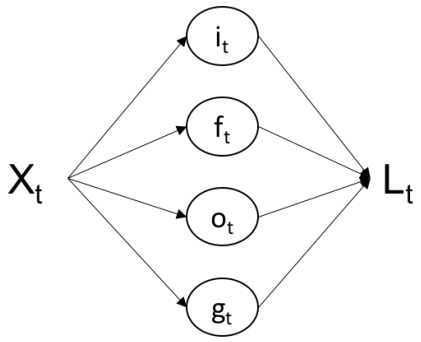

Figure 3: Relation between X and Loss

X_t impacts L_t though four gates. Thus when do derivative, we need to sum up the partial derivative from four sources.

Finally, let me explain how find $\frac{\partial L_t}{\partial H_t}$. H_t also depends on H_{t-1} which depends on H_{t-2}, so on and so forth. Since we are doing backpropagation, which means that we can find $\frac{\partial L_t}{\partial H_{t-1}}$ by computing $\frac{\partial H_t}{\partial H_{t-1}}$, we can find this at the step $t-1$. When we at step t, all we have to do is to find the impact of H_t on L_t at step t which is $\left(\frac{\partial L_t}{\partial H_t}\right)^*$ and pass this to the derivation work to the step $t-1$. We keep doing this until we reach the step 0.

Now we have worked out all the details in LSTM!

3 Convolutional Neural Network

The derivation work in CNN is fairly simple and straightforward, I won't be doing derivation here. Instead, I will fully explain the mechanism of the convolution layer and pooling layer which are the two main components in CNN.

3.1 Convolution Layer

In image classification work, the purpose of convolution layer is to extract spatial feature of a image. To do this, we use a component called filter.

First, let's say we have a image to feed into a CNN network. The first layer is usually a convolution layer. The size of the image is $D \times H \times W$. For simplicity, let's say it's $3 \times 32 \times 32$, where the dimension D or 3 represents the three channels (red, green, blue).

Now we want to use convolution layer to extract features of this image. We create a filter, which essentially is nothing but a matrix with dimension $D \times H^* \times W^*$. The filter must have the same channel size D as the image, and in most cases, H^* and W^* are equal. Here we use a filter $F_{3\times3\times3}$.

To extract features, we overlay the filter F to the most left-top part of the $3 \times 32 \times 32$ image matrix. Now we do a element-wise production of the F and the correspond overlaying part in image matrix.

Figure 4 gives an example of how a filter works. Three bigger matrices on

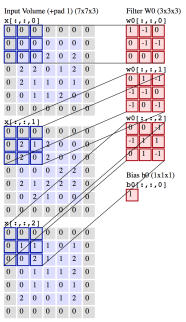

Figure 4: Convolution Filter step 1

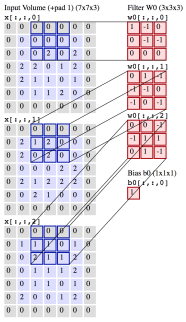

Figure 5: Convolution Filter step 2

the left are three channels a image, and three smaller matrices on the right are three channels of a filter. And there is also a numeric bias value. The image size in this case is $3 \times 7 \times 7$. As I explained above, we do a element-wise multiplication of the filter matrix and a overlaying part in the image matrix, which is highlighted in blue. we do element-wise multiplication in each channel and we will have three numeric numbers. Then we sum up the three numbers and the bias and use it as one feature.

Now we move to Figure 5. As we can see, the three blue parts move 2 strides towards the right. This will give us another feature value. If we keep move

2 strides to the right, we will have the third value. Obviously, we can move downwards. Usually the strides on each direction are equal, in this case, we let the stride be 2. At the end, we will have a 3×3 matrix. This 3×3 matrix is our first convolution layer.

In most times, we want more than just one feature layer because we want to use filters with different values but size sames to catch features with different focus in a same image region. So we create more filters, let's say 16, all with the dimension $3 \times 3 \times 3$. In this case, we will have a output matrix from the convolution layer with dimension $16 \times 3 \times 3$.

A trick to calculate the size of the filtered feature matrix:
Assuming input size is $D \times H \times W$, we use a filter with size $D \times H^* \times W^*$. The dimension of the corresponding feature matrix will be

$$D \times (\frac{H - H^*}{S_H} + 1) \times (\frac{W - W^*}{S_W} + 1)$$

, where S_H and S_W are strides on the H and W direction, respectively.

3.2 Pooling Layer

Pooling layers are used to downsample our representation of spatial features (the output of convolution layers). Is it periodically used in between convolution layers to reduce parameter space and computation.

The most commonly used pooling operation is call Max Pooling.

Figure 6 shows how max pooling works. We usually keep a pooling kernel

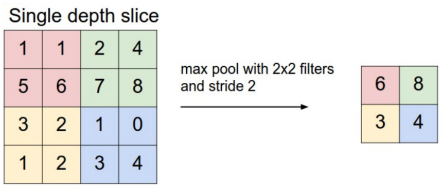

Figure 6: Max Pooling Operation

in 2 by 2 size, and overlay it on the feature matrix and move it with a stride of 2. For each overlaying region, we only take the max value of that region. As represented in the Figure 6, it shrinks the feature size dramatically. Since it operation on each channel, pooling operation only reduces H and W. D will be unchanged. One thing to keep in mind is that in pooling layers, there will not be any new parameters.

3.3 Activation Function, Regularization and Final Output

There are different types of activation functions such as Relu, Sigmoid, tanh, etc. In image tasks, ReLU function is most commonly used.

As for regularization, people usually in Dropout to avoid overfitting. Dropout is usually applied after we feed the output of a series of convolution layers and pooling layers into a 2-layer fully connected neural network. The way it works is for each neuron, it sample a 0-1 Bernoulli random variable with some parameter P. Then each neuron has a probability P to keep its original value and (1-P) to be change to zero. If it is changed to zero, it means that this neuron will no longer part of the contribution to final output, which means it is dropped out.