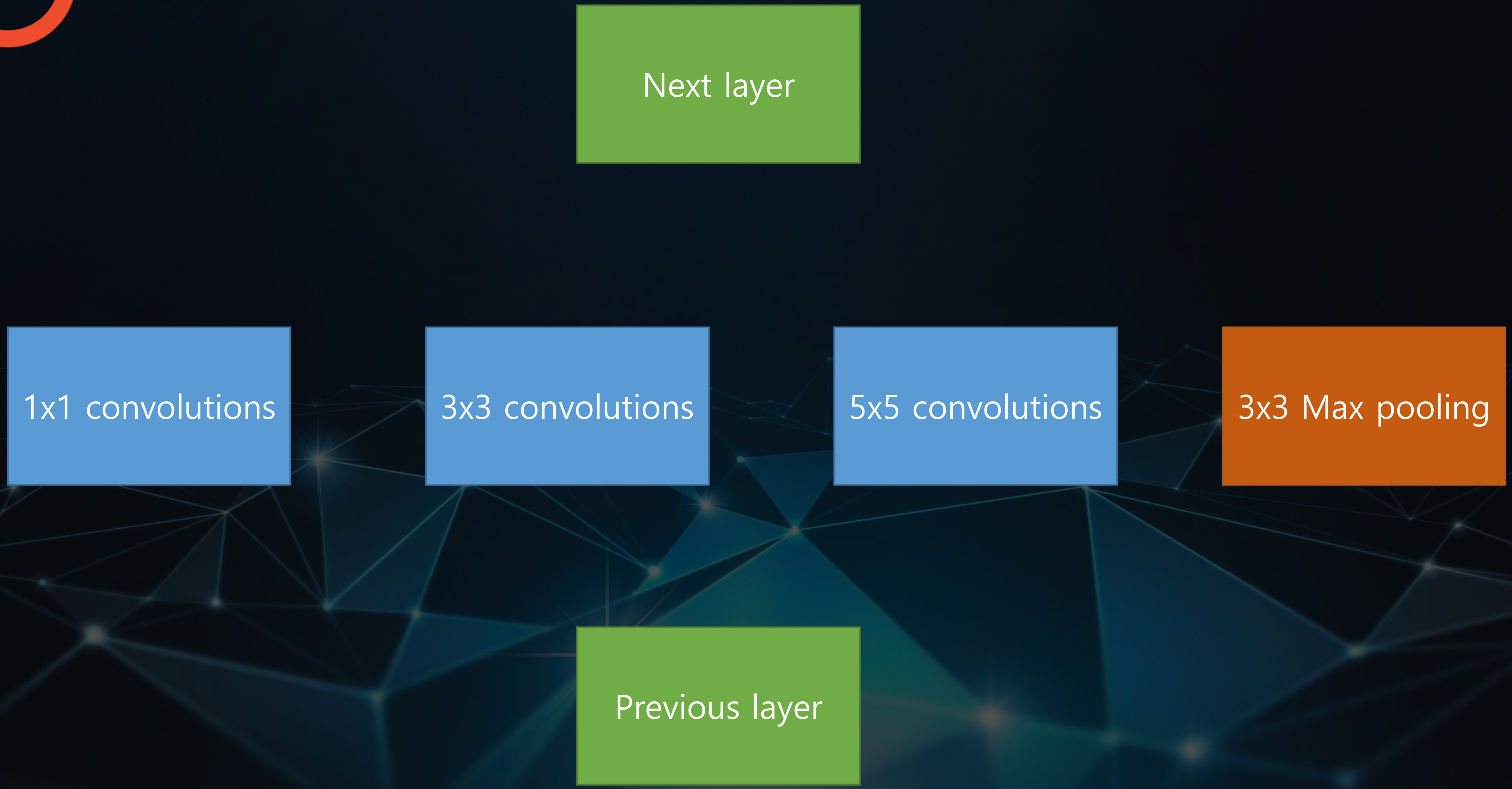
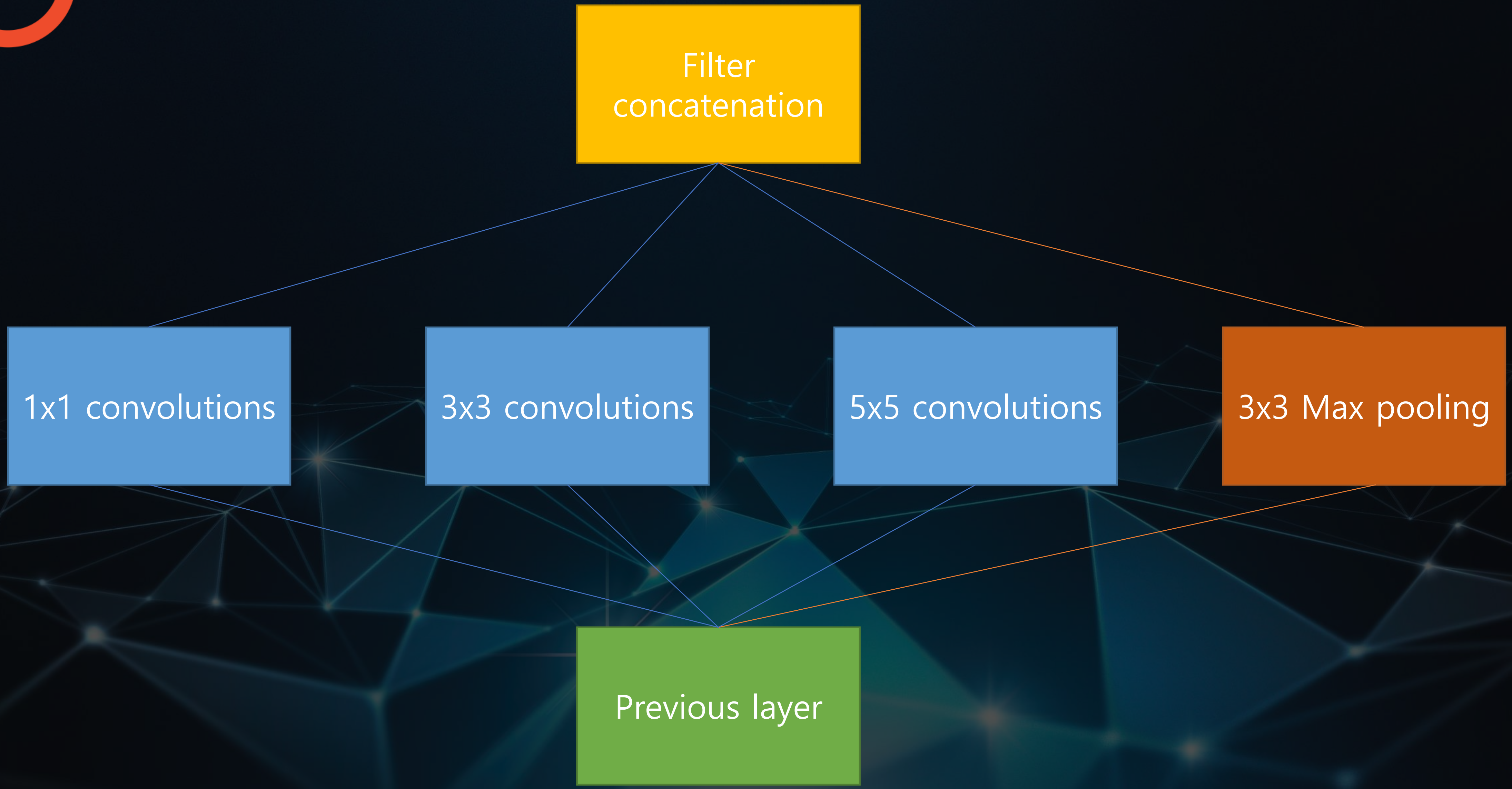
D-ai-ving

Advanced CNN & RNN Basic

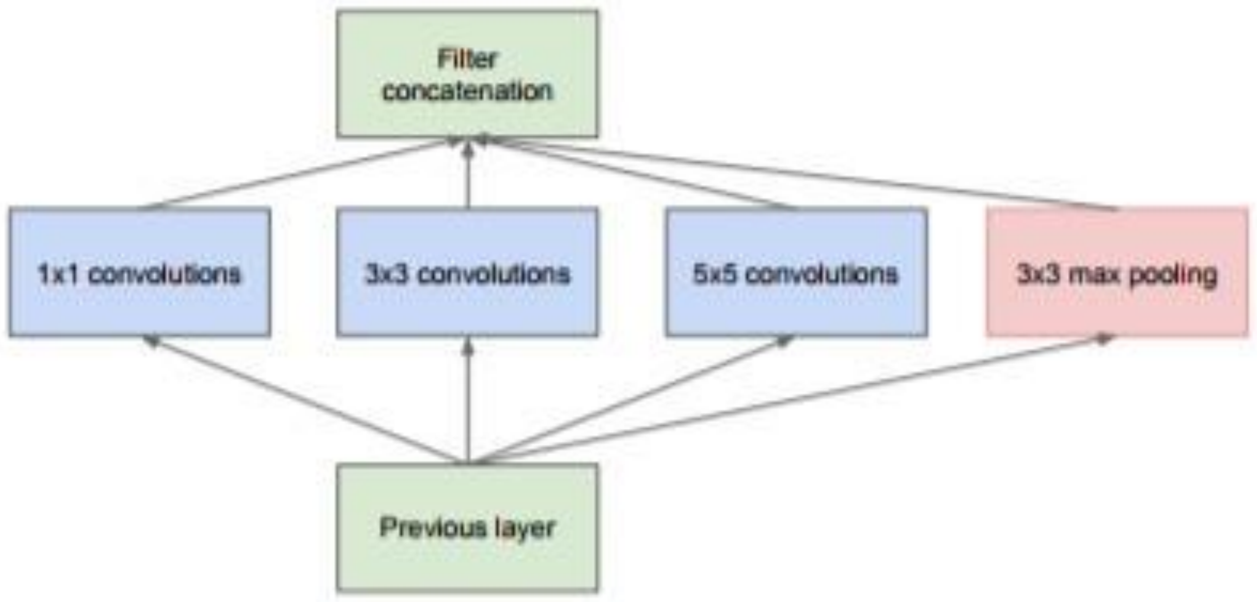
Advanced CNN



Advanced CNN

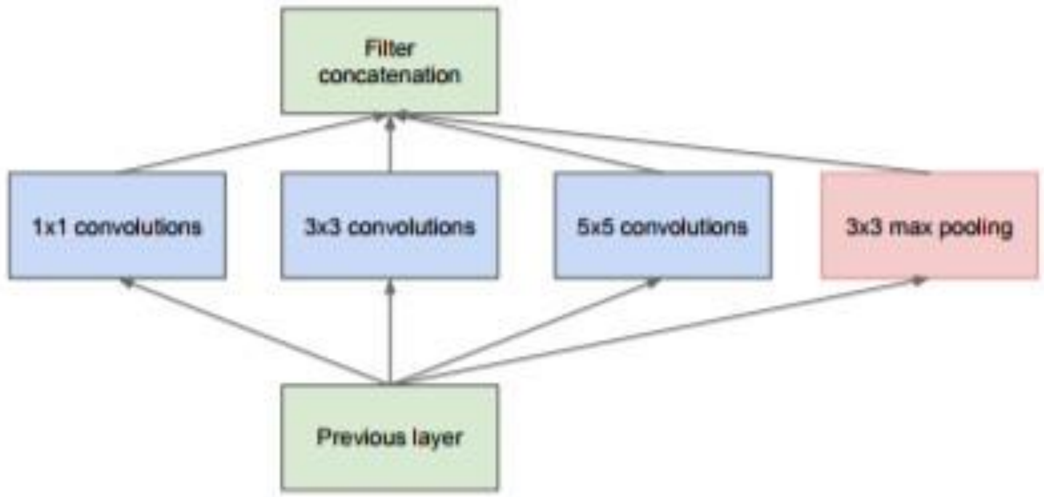


Advanced CNN

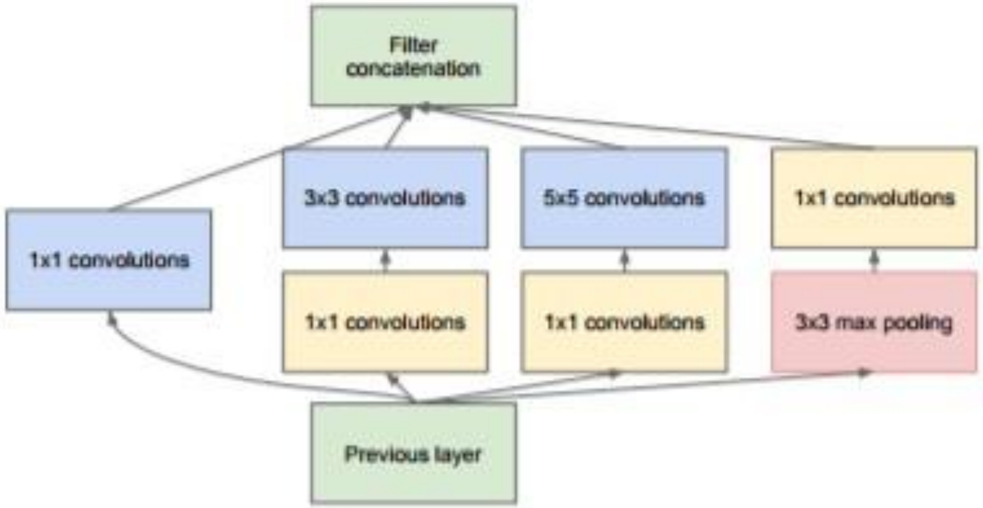


(a) Inception module, naïve version

Advanced CNN

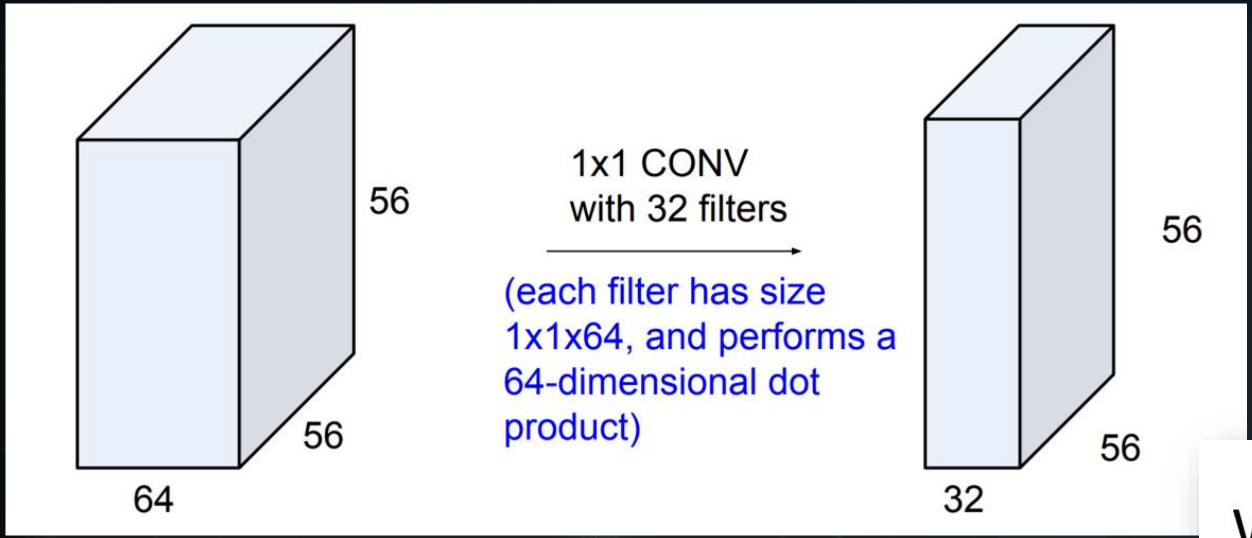


(a) Inception module, naïve version



(b) Inception module with dimension reductions

Advanced CNN



Why 1x1 convolution

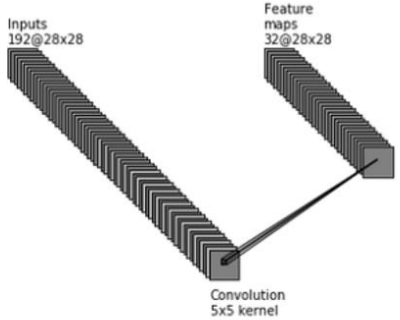


Figure 6. 5×5 convolutions inside the Inception module using the naive model

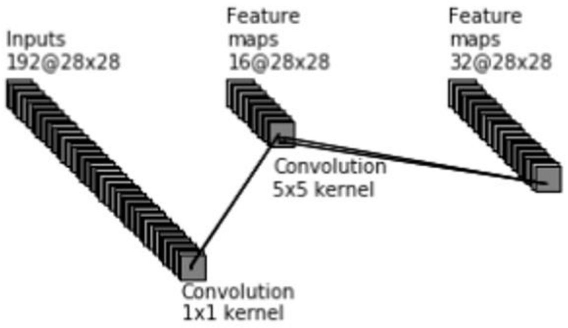
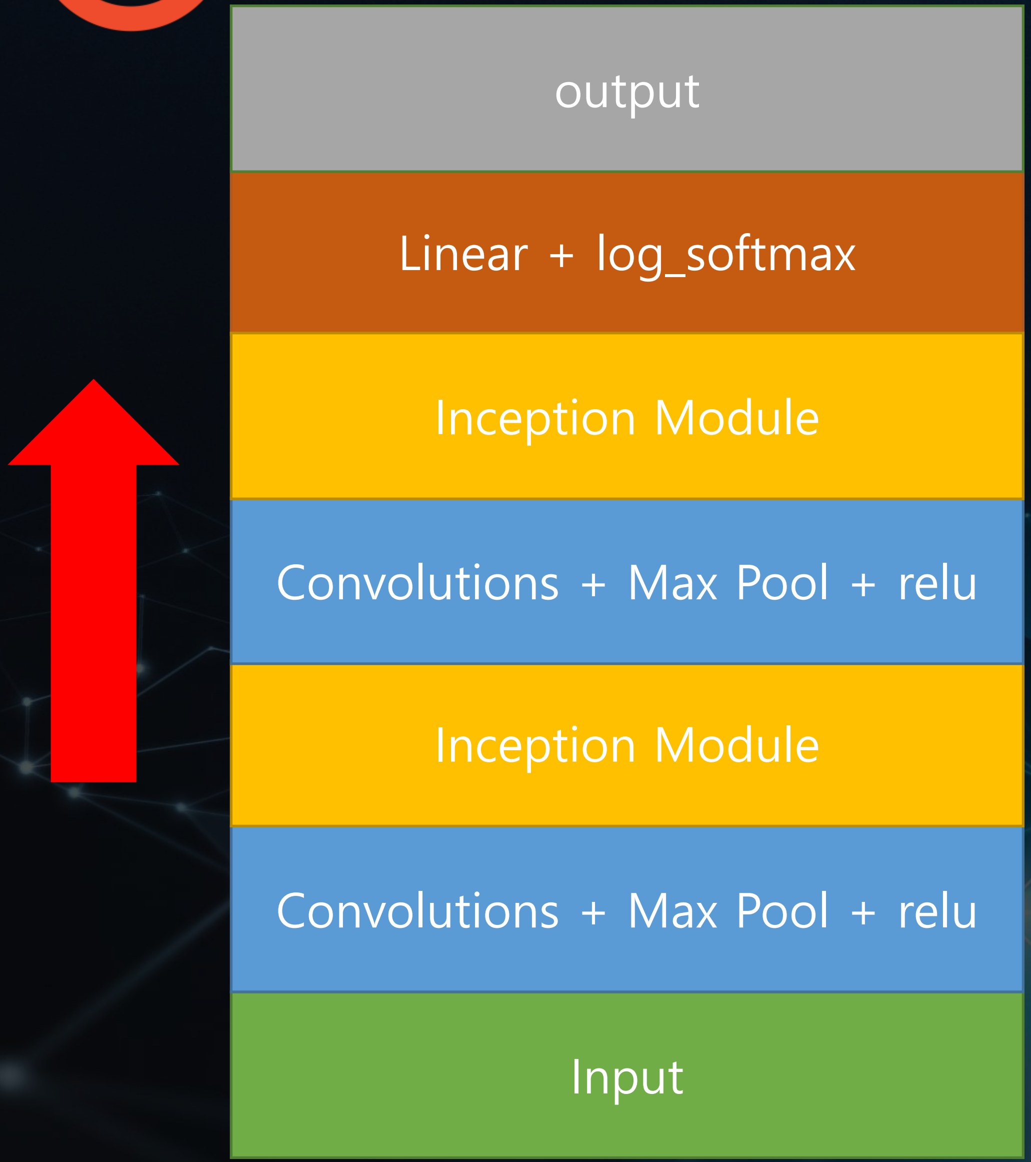


Figure 7. 1×1 convolutions serve as the dimensionalit reducers that limit the number of expensive 5×5 convolutions that follow

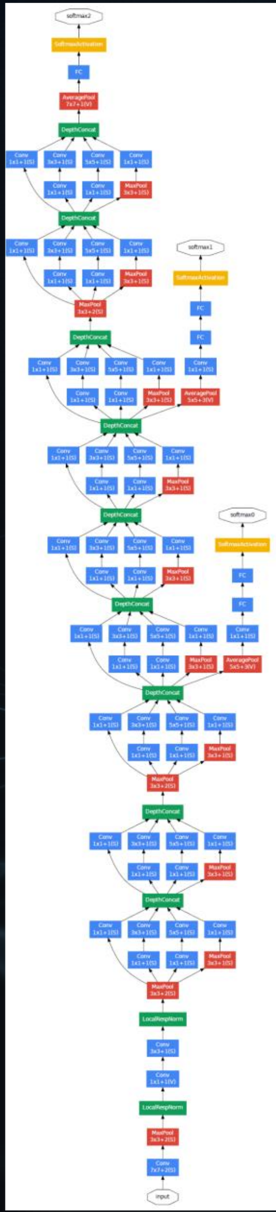
Advanced CNN

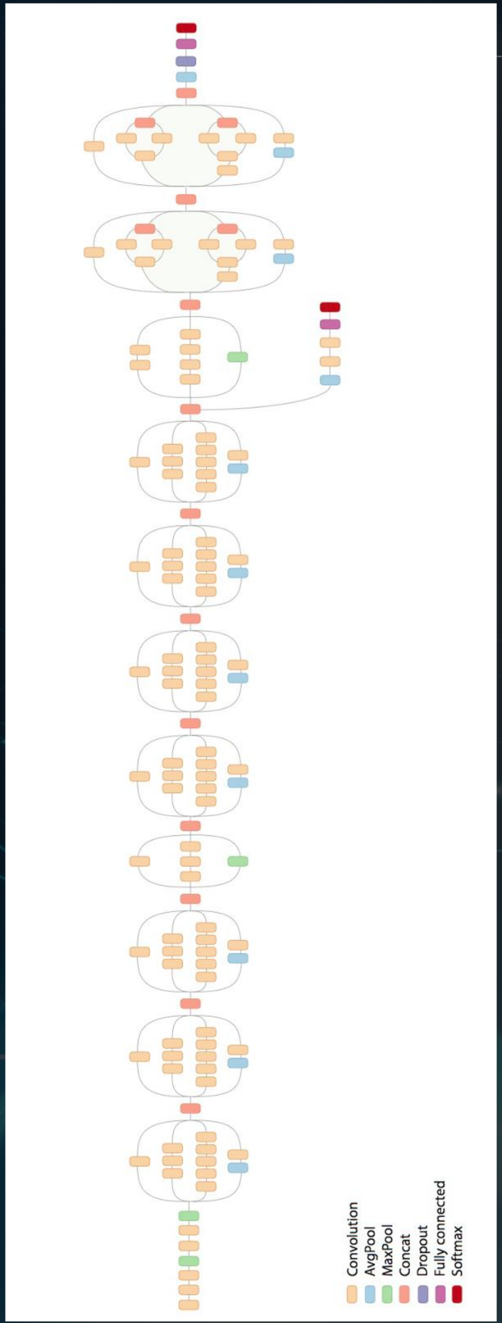


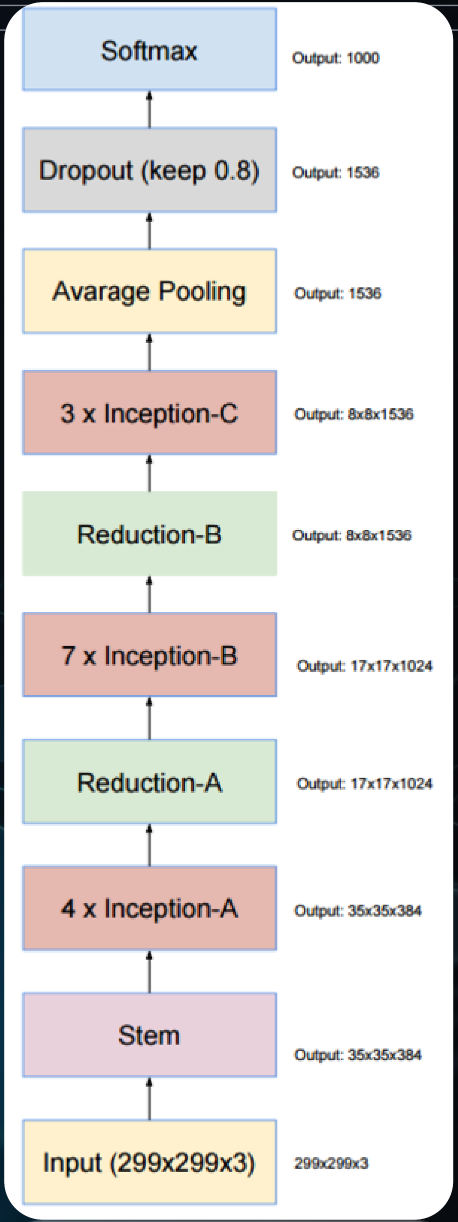

```python
class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(88, 20, kernel_size=5)

        self.incept1 = InceptionA(in_channels=10)
        self.incept2 = InceptionA(in_channels=20)

        self.mp = nn.MaxPool2d(2)
        self.fc = nn.Linear(1408, 10)

    def forward(self, x):
        in_size = x.size(0)
        x = F.relu(self.mp(self.conv
        x = self.incept1(x)
        x = F.relu(self.mp(self.conv
        x = self.incept2(x)
        x = x.view(in_size, -1)  # f
        x = self.fc(x)
        return F.log_softmax(x, -1)
```

```
Train Epoch: 9 [48000/60000 (80%)]	Loss: 0.049192
Train Epoch: 9 [48640/60000 (81%)]	Loss: 0.035681
Train Epoch: 9 [49280/60000 (82%)]	Loss: 0.006367
Train Epoch: 9 [49920/60000 (83%)]	Loss: 0.035510
Train Epoch: 9 [50560/60000 (84%)]	Loss: 0.165726
Train Epoch: 9 [51200/60000 (85%)]	Loss: 0.050424
Train Epoch: 9 [51840/60000 (86%)]	Loss: 0.164085
Train Epoch: 9 [52480/60000 (87%)]	Loss: 0.008275
Train Epoch: 9 [53120/60000 (88%)]	Loss: 0.006872
Train Epoch: 9 [53760/60000 (90%)]	Loss: 0.048391
Train Epoch: 9 [54400/60000 (91%)]	Loss: 0.019315
Train Epoch: 9 [55040/60000 (92%)]	Loss: 0.027699
Train Epoch: 9 [55680/60000 (93%)]	Loss: 0.025706
Train Epoch: 9 [56320/60000 (94%)]	Loss: 0.042459
Train Epoch: 9 [56960/60000 (95%)]	Loss: 0.071335
Train Epoch: 9 [57600/60000 (96%)]	Loss: 0.040346
Train Epoch: 9 [58240/60000 (97%)]	Loss: 0.047501
Train Epoch: 9 [58880/60000 (98%)]	Loss: 0.089792
Train Epoch: 9 [59520/60000 (99%)]	Loss: 0.062169

Test set: Average loss: 0.0446, Accuracy: 9864/10000 (99%)
```

Advanced CNN







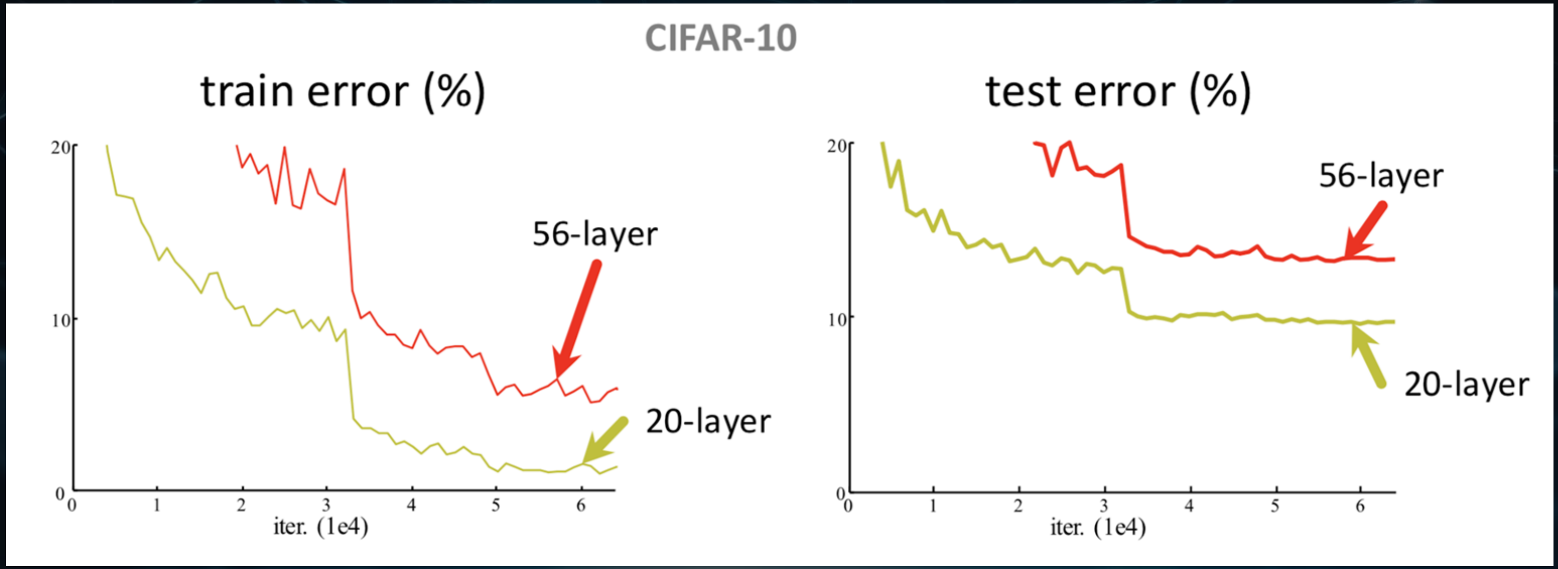
Advanced CNN

버전	특징
Inception. v1.	Inception module을 제시한 최초의 모델
Inception. v2	파라미터 수, 연산량, 복잡도를 줄이기 위해 convolution 필터를 3x3만으로 제한함.(VGG style 도입)
Inception. v3	v2에서 구조는 바뀌지 않고 일부 optimizer 종류와 convolution argument만 바뀜, 우리가 주로 사용하는 모델
Inception. v4	2015년에 Resnet을 추가하기 위해 발표, 세부적으로 달라진 inception layer 3개 사용(A, B, C)

Advanced CNN

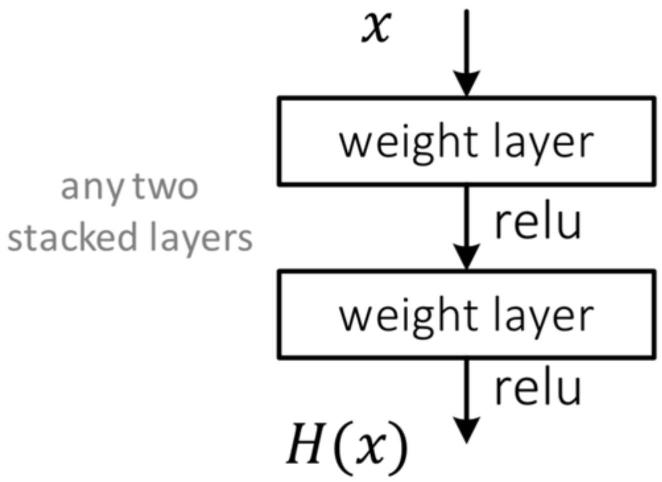
Problems with stacking layers

① Vanishing gradients problem
② Back propagation kind of gives up…
③ Degradation problem
  – with increased network depth accuracy gets saturated and then rapidly degrades

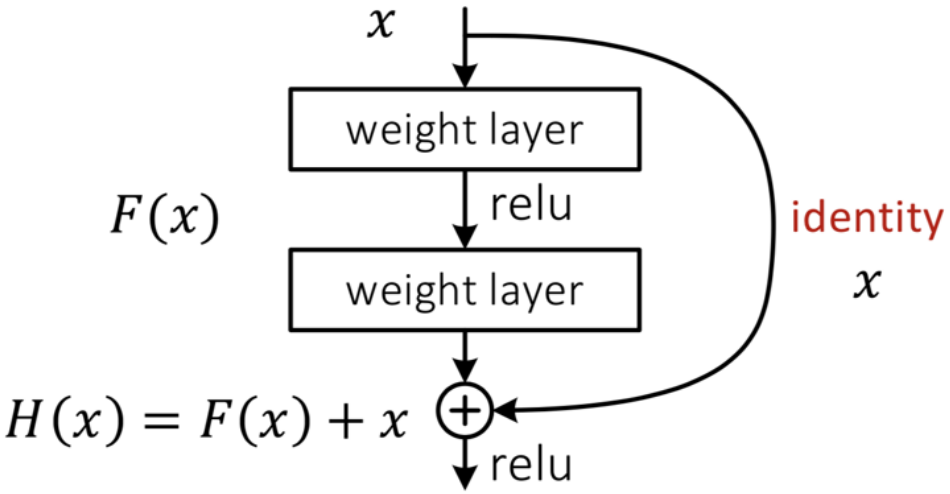


Deep Residual Learning

- Plaint net



- Residual net



Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". CVPR 2016.

Advanced CNN

```python
        self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
        self.conv2 = nn.Conv2d(88, 20, kernel_size=5)
        self.conv3 = nn.Conv2d(88, 88, kernel_size=9)

        self.incept1 = InceptionA(in_channels=10)
        self.incept2 = InceptionA(in_channels=20)

        self.mp = nn.MaxPool2d(2)
        self.fc = nn.Linear(1408, 10)

    def forward(self, x):
        in_size = x.size(0)
        x = F.relu(self.mp(self.conv1(x)))
        x = self.incept1(x)
        jump = self.conv3(x)
        x = F.relu(self.mp(self.conv2(x)))
        x = self.incept2(x)
        x = x+jump
        x = x.view(in_size, -1)  # flatten the tensor
```

Inception

```
Train Epoch: 4 [50560/60000 (84%)]      Loss: 0.066574
Train Epoch: 4 [51200/60000 (85%)]      Loss: 0.069952
Train Epoch: 4 [51840/60000 (86%)]      Loss: 0.156167
Train Epoch: 4 [52480/60000 (87%)]      Loss: 0.276394
Train Epoch: 4 [53120/60000 (88%)]      Loss: 0.126287
Train Epoch: 4 [53760/60000 (90%)]      Loss: 0.046774
Train Epoch: 4 [54400/60000 (91%)]      Loss: 0.061152
Train Epoch: 4 [55040/60000 (92%)]      Loss: 0.068414
Train Epoch: 4 [55680/60000 (93%)]      Loss: 0.038474
Train Epoch: 4 [56320/60000 (94%)]      Loss: 0.017419
Train Epoch: 4 [56960/60000 (95%)]      Loss: 0.128696
Train Epoch: 4 [57600/60000 (96%)]      Loss: 0.036454
Train Epoch: 4 [58240/60000 (97%)]      Loss: 0.053295
Train Epoch: 4 [58880/60000 (98%)]      Loss: 0.032368
Train Epoch: 4 [59520/60000 (99%)]      Loss: 0.092593

Test set: Average loss: 0.0682, Accuracy: 9771/10000 (98%)
```

VS

4 epoch

Inception Resnet

```
Train Epoch: 4 [51840/60000 (86%)]      Loss: 0.025530
Train Epoch: 4 [52480/60000 (87%)]      Loss: 0.068905
Train Epoch: 4 [53120/60000 (88%)]      Loss: 0.023693
Train Epoch: 4 [53760/60000 (90%)]      Loss: 0.101106
Train Epoch: 4 [54400/60000 (91%)]      Loss: 0.196795
Train Epoch: 4 [55040/60000 (92%)]      Loss: 0.045218
Train Epoch: 4 [55680/60000 (93%)]      Loss: 0.167126
Train Epoch: 4 [56320/60000 (94%)]      Loss: 0.393767
Train Epoch: 4 [56960/60000 (95%)]      Loss: 0.035607
Train Epoch: 4 [57600/60000 (96%)]      Loss: 0.078126
Train Epoch: 4 [58240/60000 (97%)]      Loss: 0.076608
Train Epoch: 4 [58880/60000 (98%)]      Loss: 0.068349
Train Epoch: 4 [59520/60000 (99%)]      Loss: 0.100903

Test set: Average loss: 0.0712, Accuracy: 9760/10000 (98%)
```

Advanced CNN

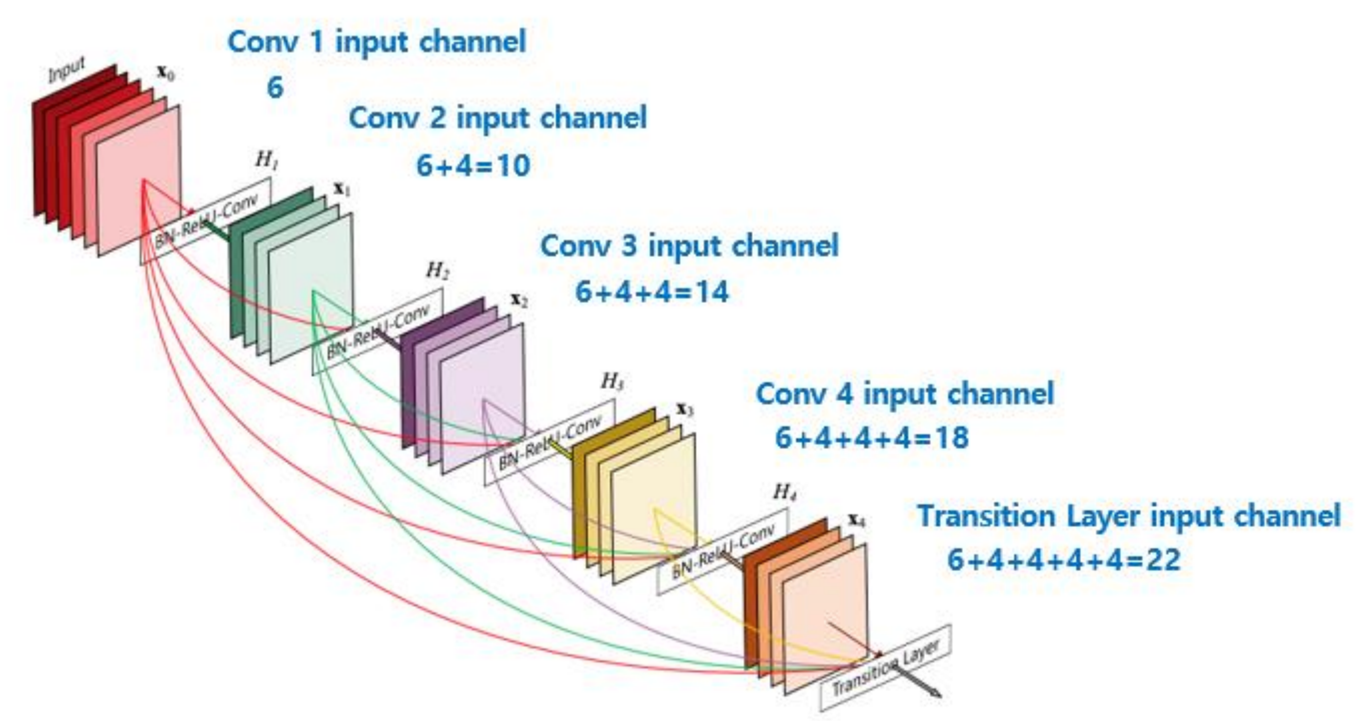
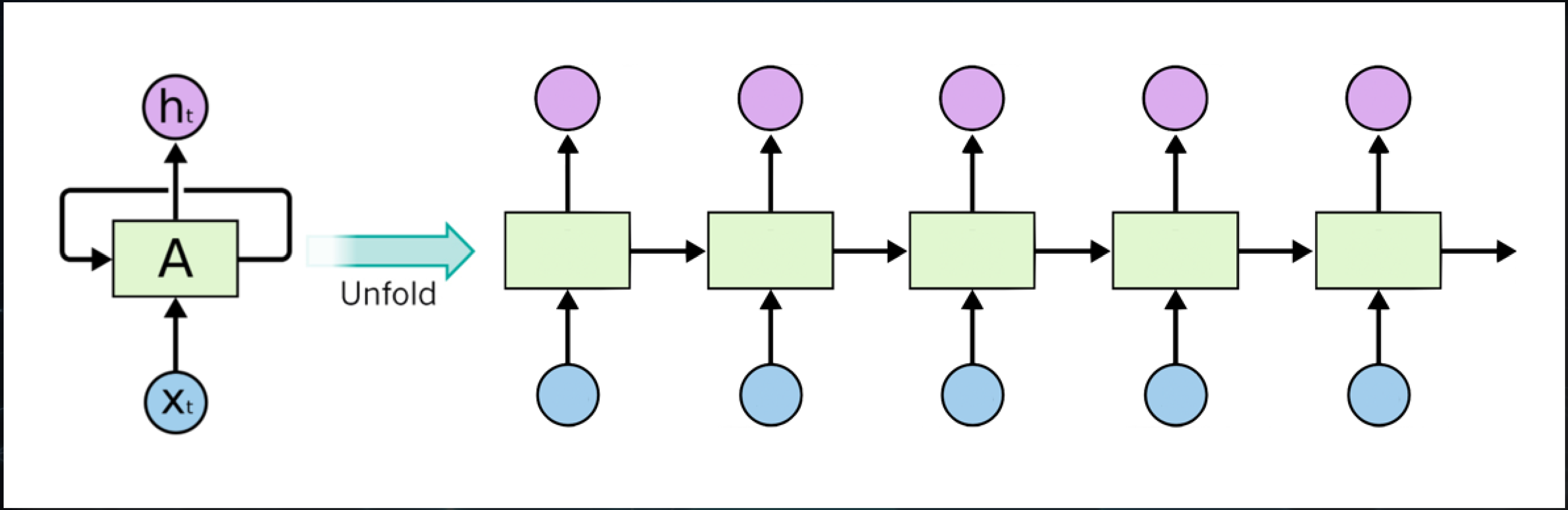


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

RNN Basic

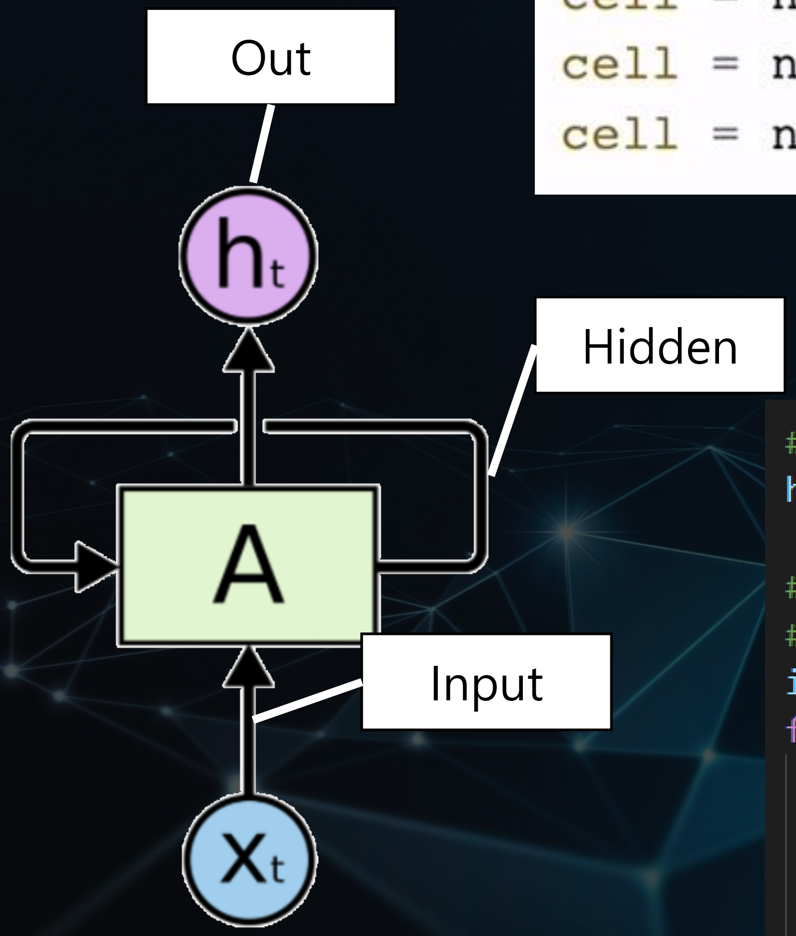


RNN Basic

	x - input		y - output
Speech recognition		⟹	"Fuzzy Wuzzy was a bear. Fuzzy Wuzzy had no hair."
Music generation	∅	⟹	
Sentiment classification	"Decent effort. The plot could have been better."	⟹	★★★☆☆
DNA sequence analysis	ACTGTACCCATGTGACTGCCC	⟹	ACTGTACCCATGTGACTGCCC
Machine translation	"El que no arriesga, no gana."	⟹	"If you don't take risks, you cannot win."
Video activity recognition		⟹	Running
Name entity recognition	"Ygritte says Jon Snow knows nothing."	⟹	"Ygritte says Jon Snow knows nothing."

RNN Basic

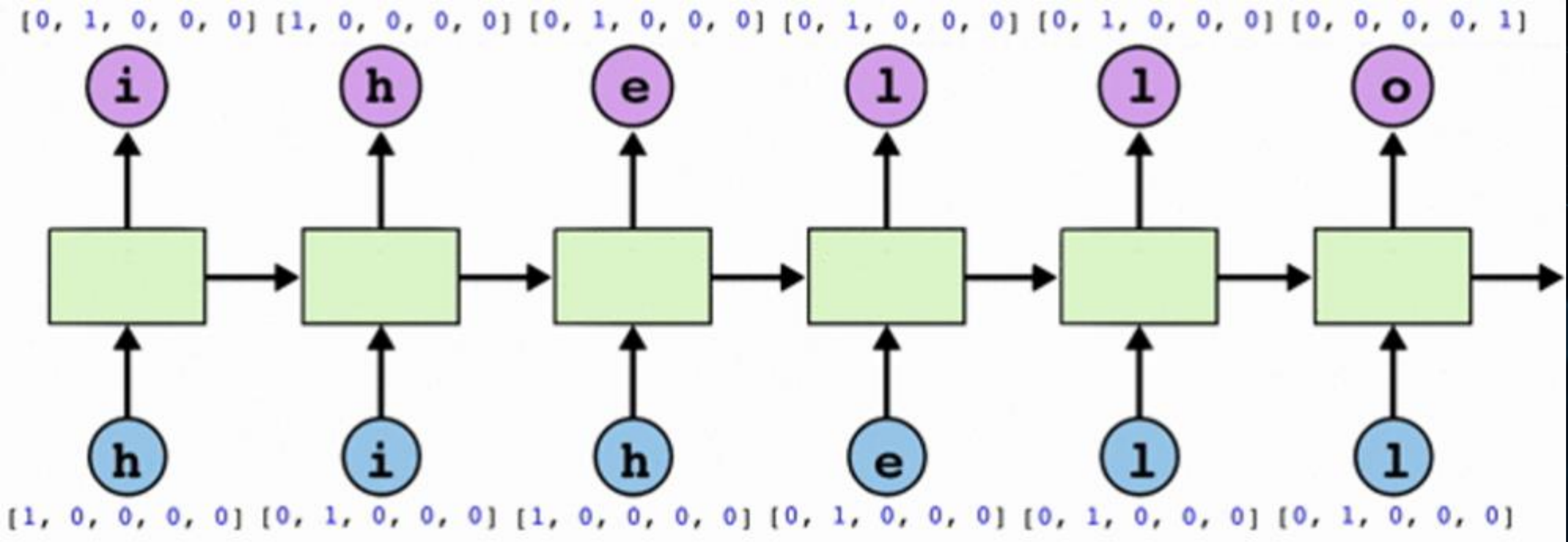
```
cell = nn.RNN(input_size=4, hidden_size=2, batch_first=True)
cell = nn.GRU(input_size=4, hidden_size=2, batch_first=True)
cell = nn.LSTM(input_size=4, hidden_size=2, batch_first=True)
```



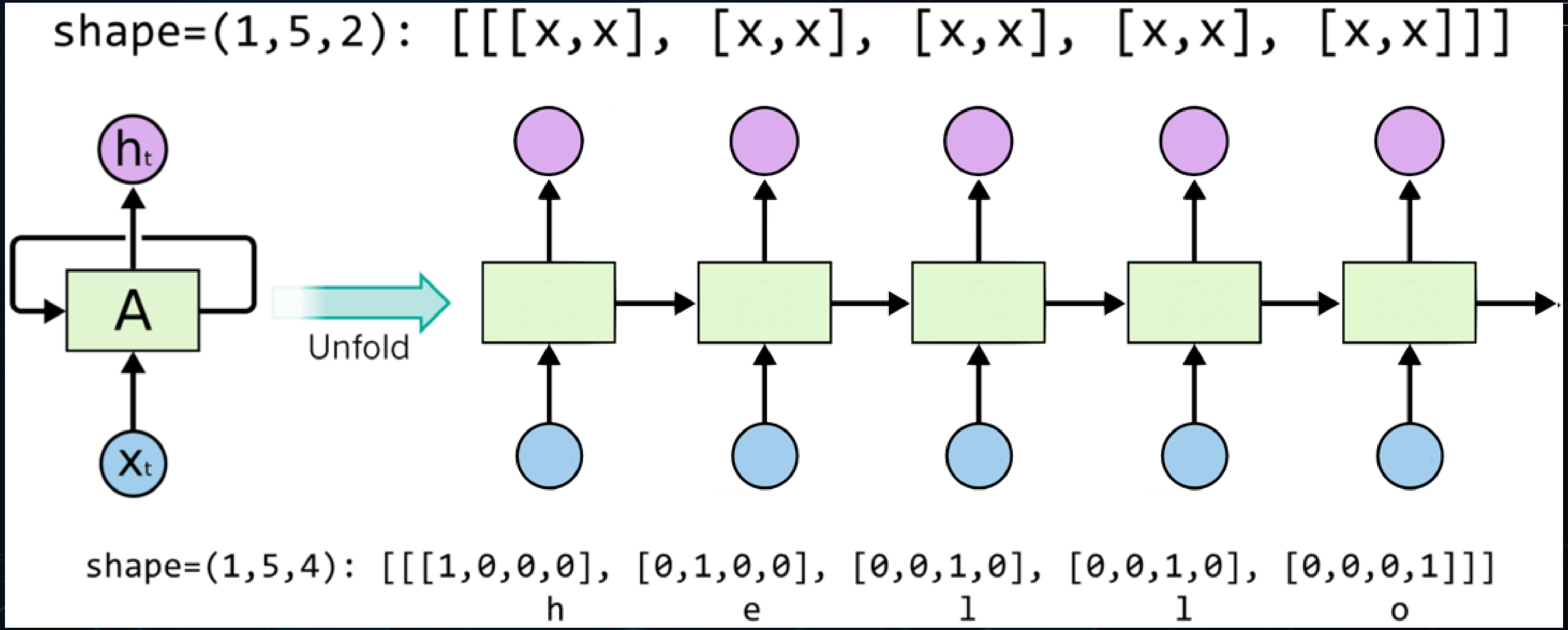

```
# (num_layers * num_directions, batch, hidden_size) whether batch_first=True or False
hidden = Variable(torch.randn(1, 1, 2))

# Propagate input through RNN
# Input: (batch, seq_len, input_size) when batch_first=True
inputs = Variable(torch.Tensor([h, e, l, l, o]))
for one in inputs:
    one = one.view(1, 1, -1)
    # Input: (batch, seq_len, input_size) when batch_first=True
    out, hidden = cell(one, hidden)
    print("one input size", one.size(), "out size", out.size())
```

RNN Basic

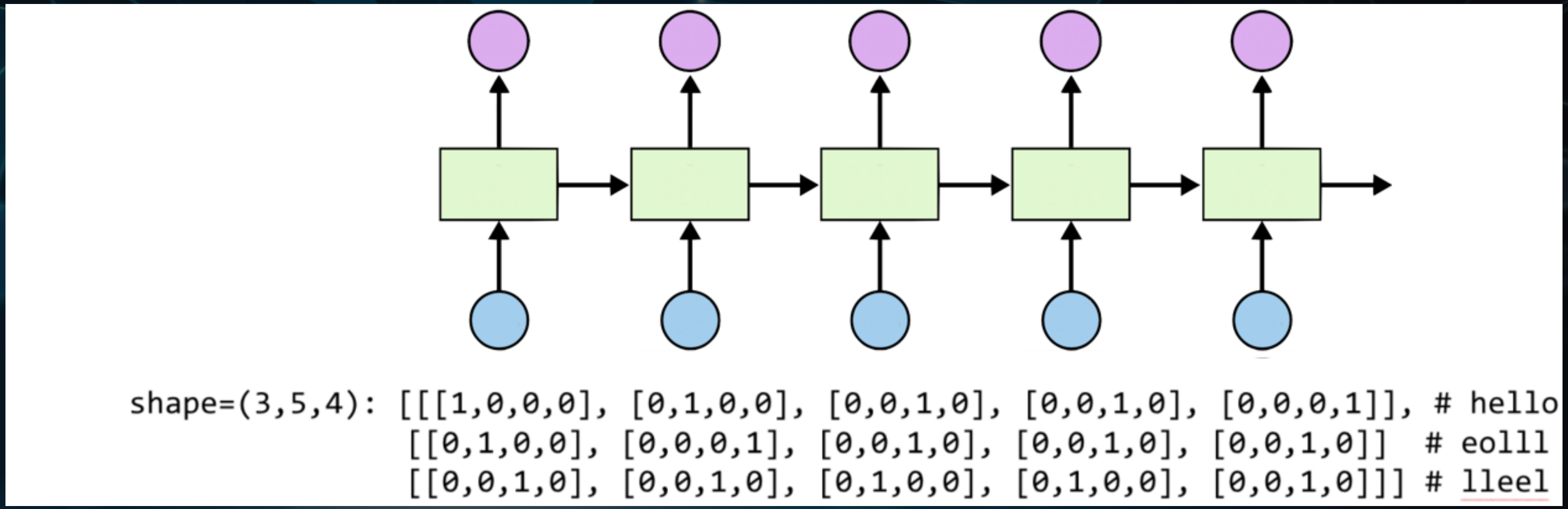
```python
x_data = [0, 1, 0, 2, 3, 3]   # hihell
one_hot_lookup = [[1, 0, 0, 0, 0],  # 0
                  [0, 1, 0, 0, 0],  # 1
                  [0, 0, 1, 0, 0],  # 2
                  [0, 0, 0, 1, 0],  # 3
                  [0, 0, 0, 0, 1]]  # 4

y_data = [1, 0, 2, 3, 3, 4]    # ihello
x_one_hot = [one_hot_lookup[x] for x in x_data]
```



RNN Basic



Shape =
(배치 수, 시퀀스 크기,
input 크기)

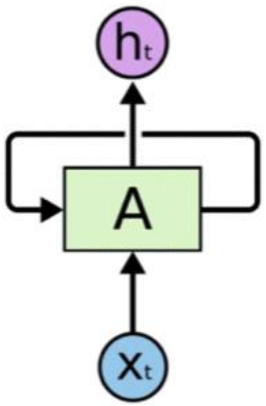


RNN Basic

```
l before view: torch.Size([1, 1, 5])
after view:  torch.Size([1, 5])
```

```python
out = out.view(-1, num_classes)
```

```python
# CrossEntropyLoss = LogSoftmax + NLLLoss
criterion = nn.CrossEntropyLoss()
```




```python
loss = 0
hidden = model.init_hidden()

sys.stdout.write("predicted string: ")
for input, label in zip(inputs, labels):
    hidden, output = model(hidden, input)
    loss += criterion(output, label)

print(", epoch: %d, loss: %1.3f" %
                    (epoch + 1, loss.data[0]))

loss.backward()
optimizer.step()
```

RNN Basic

시퀀스 사용 코드

```
model = Model()

# Set loss and optimizer function
# CrossEntropyLoss = LogSoftmax + NLLLoss
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.1)

# Train the model
for epoch in range(100):
    optimizer.zero_grad()
    loss = 0
    hidden = model.init_hidden()

    for input, label in zip(inputs, labels):
        hidden, output = model(hidden, input)
        loss += criterion(output, torch.LongTensor([label]))

    print(", epoch: %d, loss: %1.3f" % (epoch + 1, loss.item()))

    loss.backward()
    optimizer.step()
```

```
for epoch in range(100):
    outputs = rnn(inputs)
    optimizer.zero_grad()
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()
    _, idx = outputs.max(1)
    idx = idx.data.numpy()
    result_str = [idx2char[c] for c in idx.squeeze()]
    print("epoch: %d, loss: %1.3f" % (epoch + 1, loss.item()))
    print("Predicted string: ", ''.join(result_str))
```

```
Predicted string:  ihello
epoch: 96, loss: 0.458
Predicted string:  ihello
epoch: 97, loss: 0.458
Predicted string:  ihello
epoch: 98, loss: 0.458
Predicted string:  ihello
epoch: 99, loss: 0.458
Predicted string:  ihello
epoch: 100, loss: 0.458
Predicted string:  ihello
Learning finished!
```

RNN Basic

소프트 맥스 적용

```
        self.sm = nn.Softmax(dim=1)
        self.rnn = nn.RNN(input_size=5, hidden_size=5, batch_first=True)
        # self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        h_0 = Variable(torch.zeros(
            self.num_layers, x.size(0), self.hidden_size))
        x.view(x.size(0), self.sequence_length, self.input_size)
        out, _ = self.rnn(x, h_0)
        return self.sm(out.view(-1, num_classes))
```

```
Predicted string:  ihello
epoch: 96, loss: 0.458
Predicted string:  ihello
epoch: 97, loss: 0.458
Predicted string:  ihello
epoch: 98, loss: 0.458
Predicted string:  ihello
epoch: 99, loss: 0.458
Predicted string:  ihello
epoch: 100, loss: 0.458
Predicted string:  ihello
Learning finished!
```

```
Predicted string:  ihello
epoch: 96, loss: 1.201
Predicted string:  ihello
epoch: 97, loss: 1.201
Predicted string:  ihello
epoch: 98, loss: 1.201
Predicted string:  ihello
epoch: 99, loss: 1.201
Predicted string:  ihello
epoch: 100, loss: 1.201
Predicted string:  ihello
Learning finished!
```

"e/(e+(k−1)/e)) = e^2/(e^2+k−1) = 0.71 이상의 값을 얻을 수 없다."

```
Predicted string:  ihello
tensor([[0.0886, 0.6469, 0.0883, 0.0879, 0.0883],
        [0.6478, 0.0879, 0.0877, 0.0888, 0.0878],
        [0.0878, 0.0879, 0.6483, 0.0881, 0.0878],
        [0.0878, 0.0879, 0.0878, 0.6487, 0.0878],
        [0.0811, 0.0811, 0.0811, 0.5808, 0.1759],
        [0.0879, 0.0880, 0.0879, 0.0995, 0.6368]], grad_fn=<SoftmaxBackward>)
epoch: 100, loss: 1.201
Predicted string:  ihello
Learning finished!
```

RNN Basic

```
        self.sm = nn.Softmax(dim=1)
        self.rnn = nn.RNN(input_size=5, hidden_size=5, batch_first=True)
        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        h_0 = Variable(torch.zeros(
            self.num_layers, x.size(0), self.hidden_size))
        x.view(x.size(0), self.sequence_length, self.input_size)
        out, _ = self.rnn(x, h_0)
        # return self.sm(out.view(-1, num_classes))
        return self.fc(out.view(-1, num_classes))
```

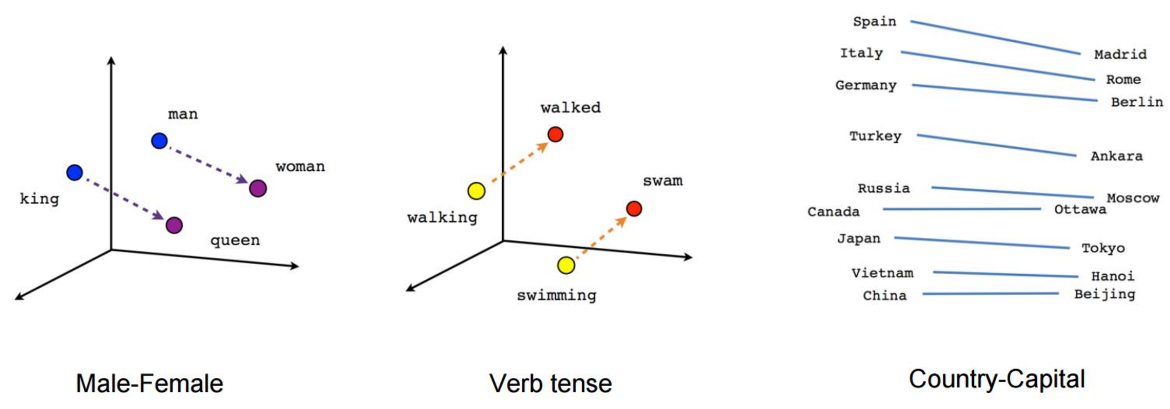
```
epoch: 96, loss: 0.001
Predicted string:  ihello
epoch: 97, loss: 0.001
Predicted string:  ihello
epoch: 98, loss: 0.001
Predicted string:  ihello
epoch: 99, loss: 0.001
Predicted string:  ihello
epoch: 100, loss: 0.001
Predicted string:  ihello
Learning finished!
```

RNN Basic

One hot VS embedding

Index (ex:2) → One hot → [0, 0, 1, 0, 0]
One-hot: len (index)

Index (ex:2) → Embedding (lookup table) → [0.2, 0.4, 0.6, 0.8, 0.5, 0.9]
Embedding
n size (typically 100~300)



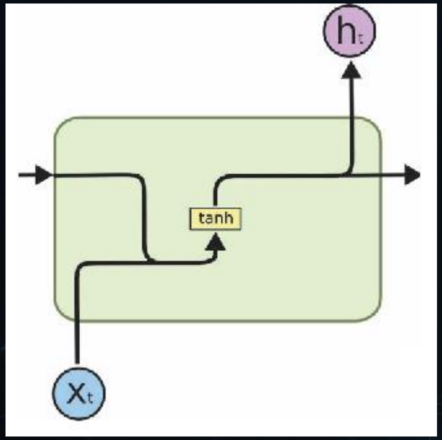
RNN Basic

```
num_classes = 5
input_size = 5
embedding_size = 10
hidden_size = 5
batch_size = 1
sequence_length = 6
num_layers = 1
```

```
class Model(nn.Module):

    def __init__(self, num_layers, hidden_size):
        super(Model, self).__init__()
        self.num_layers = num_layers
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(input_size, embedding_size)
        self.rnn = nn.RNN(input_size=embedding_size,
                          hidden_size=5, batch_first=True)
        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        h_0 = Variable(torch.zeros(
            self.num_layers, x.size(0), self.hidden_size))

        emb = self.embedding(x)
        emb = emb.view(batch_size, sequence_length, -1)
        out, _ = self.rnn(emb, h_0)
        return self.fc(out.view(-1, num_classes))
```

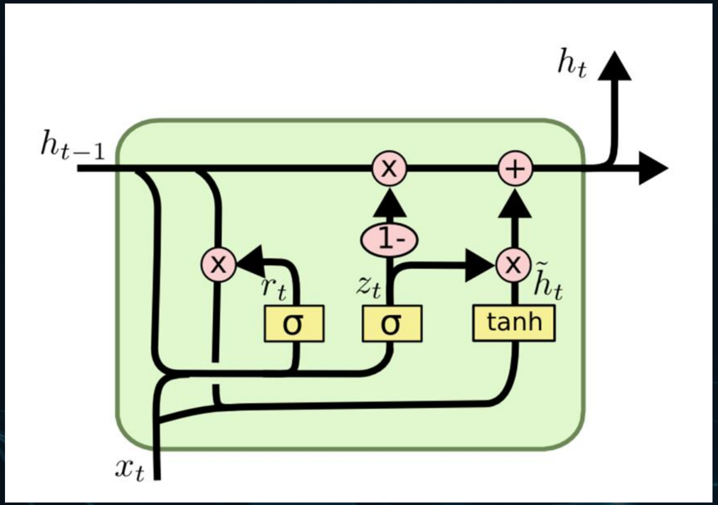
```
Predicted string:  ihello
epoch: 93, loss: 0.007
Predicted string:  ihello
epoch: 94, loss: 0.007
Predicted string:  ihello
epoch: 95, loss: 0.007
Predicted string:  ihello
epoch: 96, loss: 0.007
Predicted string:  ihello
epoch: 97, loss: 0.007
Predicted string:  ihello
epoch: 98, loss: 0.007
Predicted string:  ihello
epoch: 99, loss: 0.007
Predicted string:  ihello
epoch: 100, loss: 0.007
Predicted string:  ihello
Learning finished!
```

RNN Basic



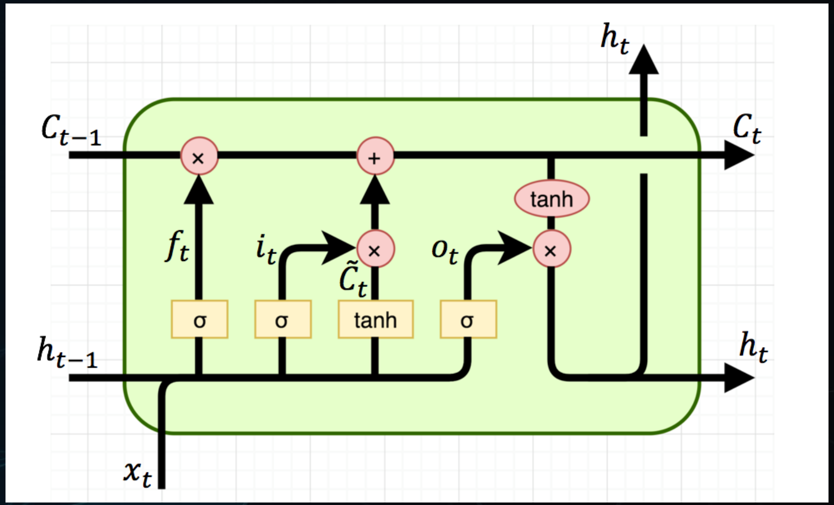
RNN

$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}), \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \\
\hat{\boldsymbol{y}}^{(t)} &= \mathrm{softmax}(\boldsymbol{o}^{(t)}),
\end{aligned}
$$



GRU(Gated Recurrent Unit)

$$
\begin{aligned}
z_t &= \sigma\left(W_z \cdot [h_{t-1}, x_t]\right) \\
r_t &= \sigma\left(W_r \cdot [h_{t-1}, x_t]\right) \\
\tilde{h}_t &= \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right) \\
h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t
\end{aligned}
$$



LSTM(Long Short Term Memory Unit)

$$
\begin{aligned}
f_t &= \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \\
i_t &= \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \\
\tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \\
C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\
o_t &= \sigma\left(W_o [h_{t-1}, x_t] + b_o\right) \\
h_t &= o_t * \tanh\left(C_t\right)
\end{aligned}
$$

D-ai-ving

감사합니다!

https://excelsior-cjh.tistory.com/185
https://towardsdatascience.com/word-embeddings-and-the-chamber-of-secrets-lstm-gru-tf-keras-de3f5c21bf16
https://hoya012.github.io/blog/DenseNet-Tutorial-1/
https://curaai00.tistory.com/1
https://arxiv.org/pdf/1512.03385v1.pdf
https://sotudy.tistory.com/13
https://blueskyvision.tistory.com/539
https://ikkison.tistory.com/86