深度學習
之
Keras
高級技巧

林澤佑

政治大學應用數學系 博士候選人

Outline

- Sequential Review
- Model (Functional API)
- Final Challenge

Sequential Review

Syntax to Neural Network

- When enter a Keras syntax and execute it, what would you think?

Syntax to Neural Network

- When enter a Keras syntax and execute it, what would you think?

```
In [2]:   model = Sequential()
```

Syntax to Neural Network

- When enter a Keras syntax and execute it, what would you think?

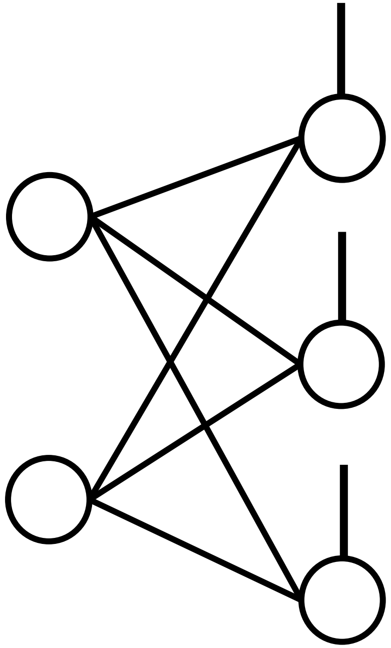
model

```
In [2]:   model = Sequential()

In [3]:
```

Syntax to Neural Network

- When enter a Keras syntax and execute it, what would you think?

model

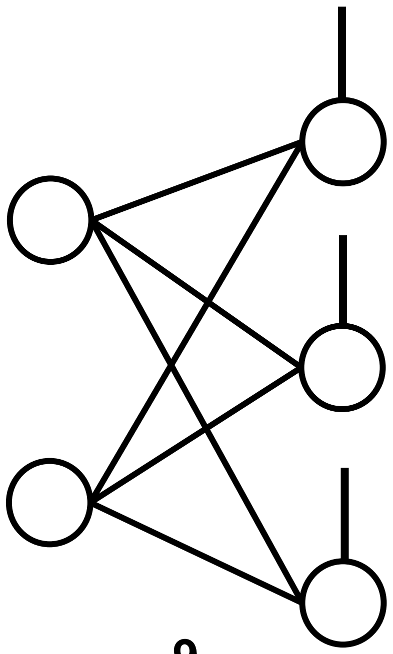


```
In [2]:   model = Sequential()

In [3]:   model.add(Dense(3, input_shape=(2, )))
```

Syntax to Neural Network

- When enter a Keras syntax and execute it, what would you think?

model

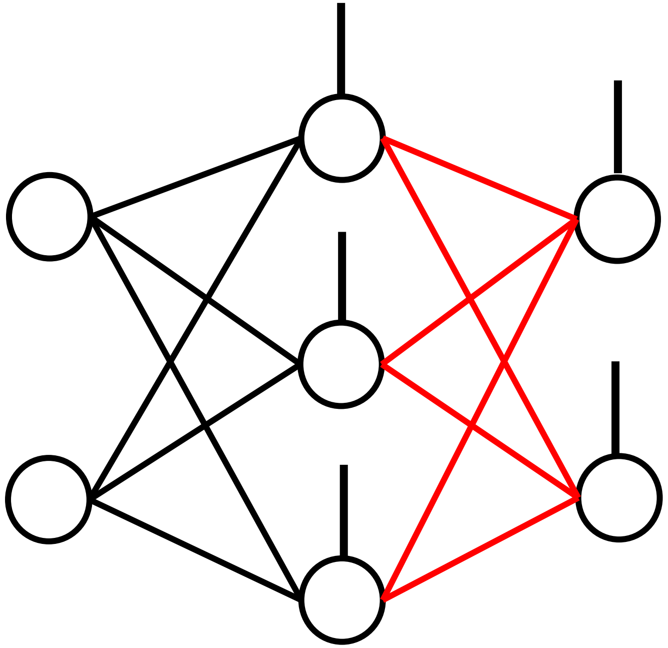



```
In [2]:   model = Sequential()

In [3]:   model.add(Dense(3, input_shape=(2, )))

In [4]:   model.summary()
```

Syntax to Neural Network

- When enter a Keras syntax and execute it, what would you think?

model

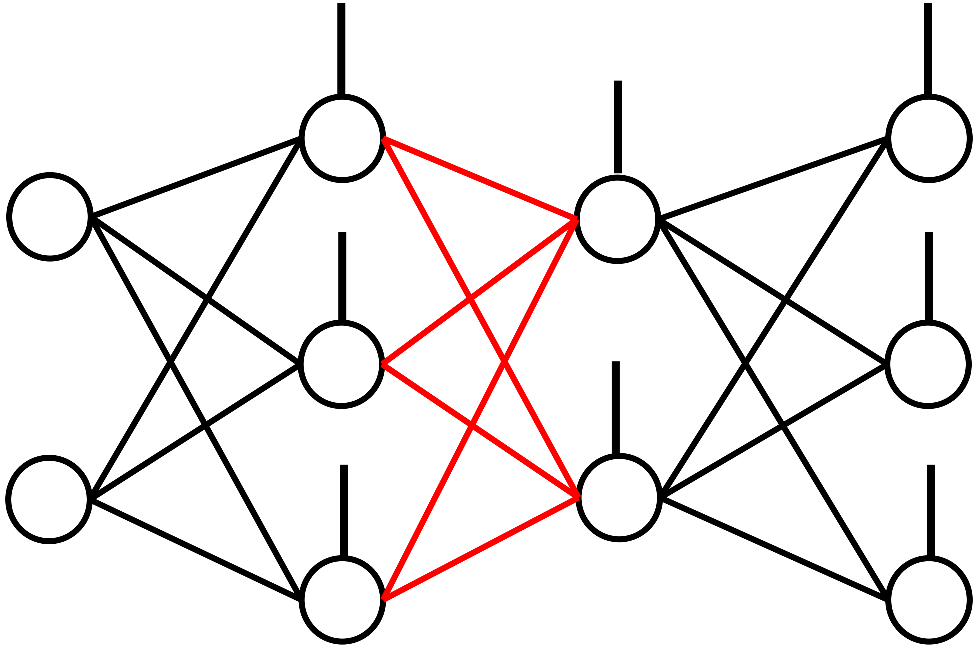


```
In [2]:   model = Sequential()

In [3]:   model.add(Dense(3, input_shape=(2, )))

In [4]:   model.summary()

In [5]:   model.add(Dense(2))
```

Syntax to Neural Network

- When enter a Keras syntax and execute it, what would you think?

model

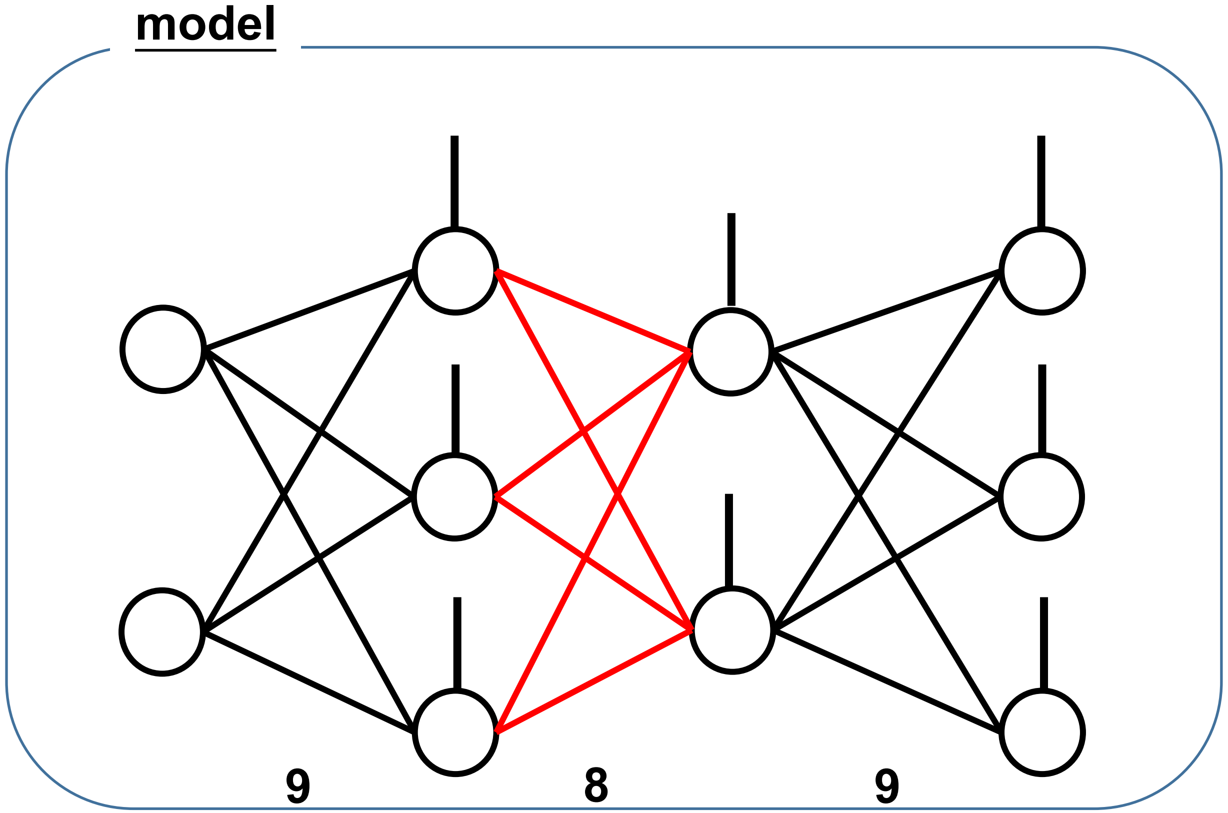


```
In [2]:   model = Sequential()

In [3]:   model.add(Dense(3, input_shape=(2, )))

In [4]:   model.summary()

In [5]:   model.add(Dense(2))
```

In [6]:   model.add(Dense(3))

Syntax to Neural Network

- When enter a Keras syntax and execute it, what would you think?




```
In [2]:   model = Sequential()

In [3]:   model.add(Dense(3, input_shape=(2, )))

In [4]:   model.summary()

In [5]:   model.add(Dense(2))

In [6]:   model.add(Dense( 3 ))

In [7]:   model.summary()
```

Parameter Counting

- How many parameters we have when a layer is added?
  - e.g., Dense, Conv1D, Conv2D, SimpleRNN, LSTM

- To simplify your observation, use simple model w/o hidden layer and set unit = 1, 2 or 3 to see how the # of parameters changed?

Parameter Counting

- How many parameters we have when a layer is added?
  - e.g., Dense, Conv1D, Conv2D, SimpleRNN, LSTM

- To simplify your observation, use simple model w/o hidden layer and set unit = 1, 2 or 3 to see how the # of parameters changed?

```
In [ ]:   model _1 = Sequential()
In [ ]:   model_1.add(Dense(2, input_shape=(2, )))
In [ ]:   model_1.summary()

In [ ]:   model_2 = Sequential()
In [ ]:   model_2.add(Dense(3, input_shape=(2, )))
In [ ]:   model_2.summary()
```

Parameter Counting

- How many parameters we have when a layer is added?
  - e.g., Dense, Conv1D, Conv2D, SimpleRNN, LSTM

- To simplify your observation, use simple model w/o hidden layer and set unit = 1, 2 or 3 to see how the # of parameters changed?

```
In [ ]:   model _3 = Sequential()
In [ ]:   model_3.add(Dense(3, input_shape=(2, )))
In [ ]:   model_3.summary()

In [ ]:   model_4 = Sequential()
In [ ]:   model_4.add(Dense(2, input_shape=(3, )))
In [ ]:   model_4.summary()
```

Parameter Counting

- You must try conv1D and Conv2D to see how they work in your model.

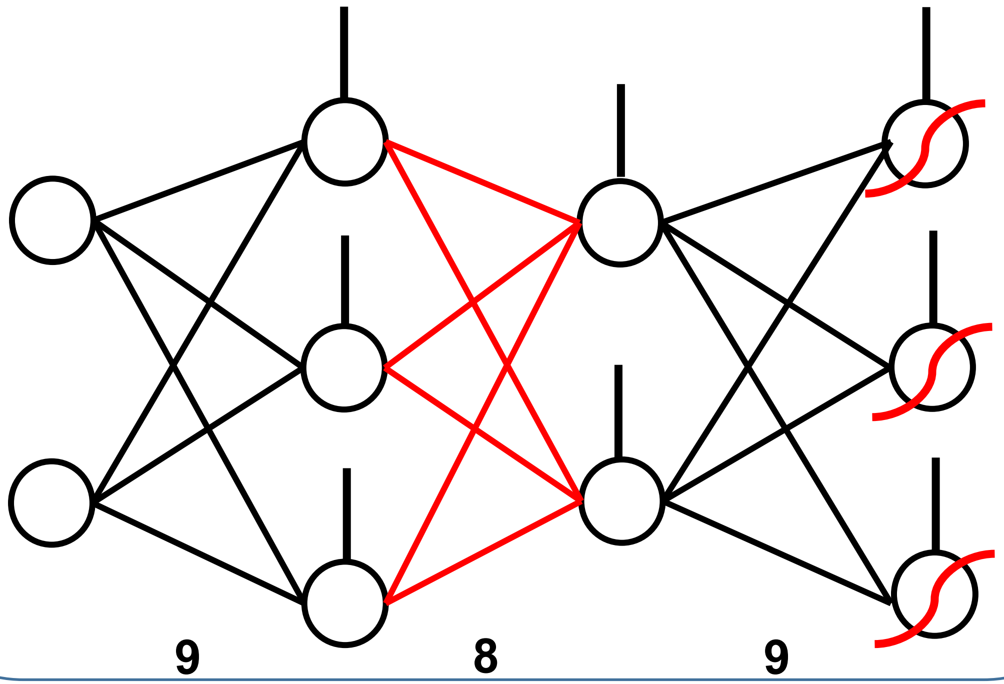
- SimpelRNN and LSTM is relative easy to understand.

- However, I suggest you to go through it by yourself.

Parameter Counting

- Activation unction doesn't affect the # of parameters.

model



In [ ]: model = Sequential()

In [ ]: model.add(Dense(3, input_shape=(2,)))

In [ ]: model.add(Dense(2))

In [ ]: model.add(Dense(3))

In [ ]: model.add(Activation('sigmoid'))

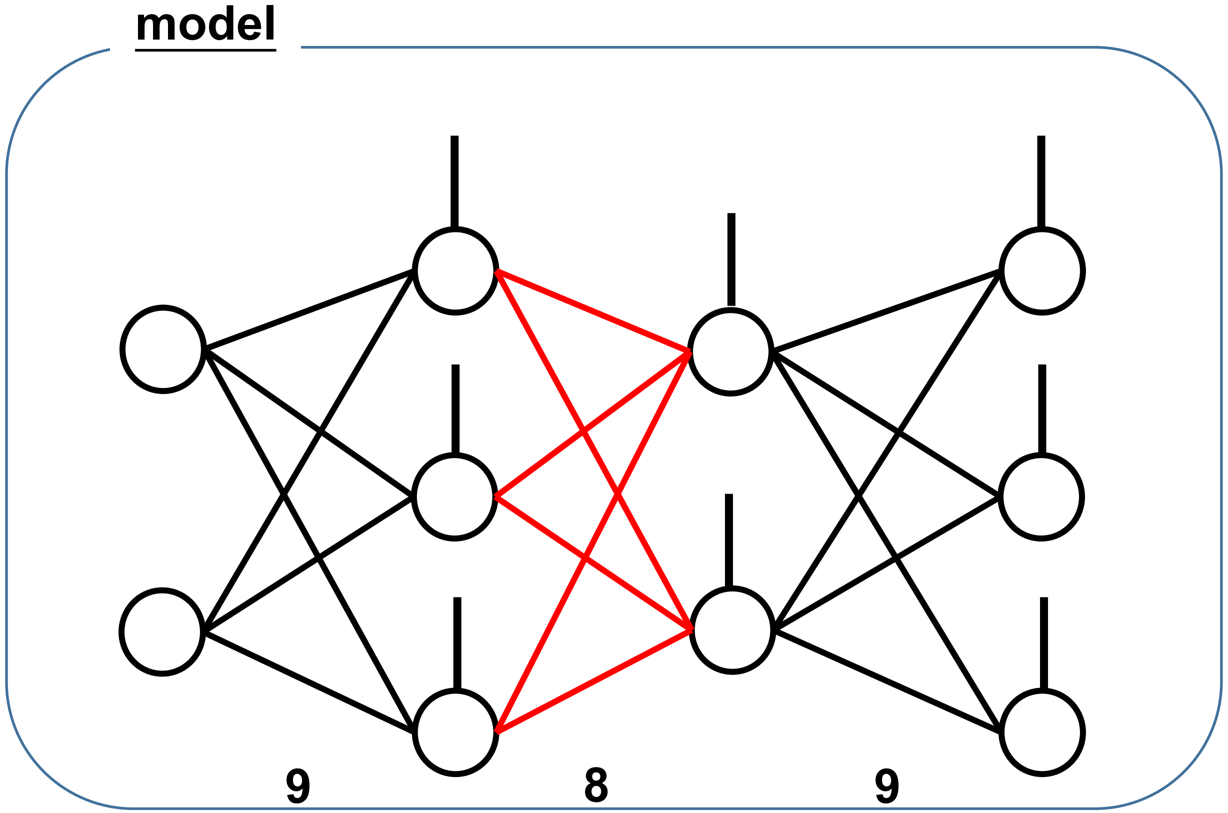
In [ ]: model.summary()

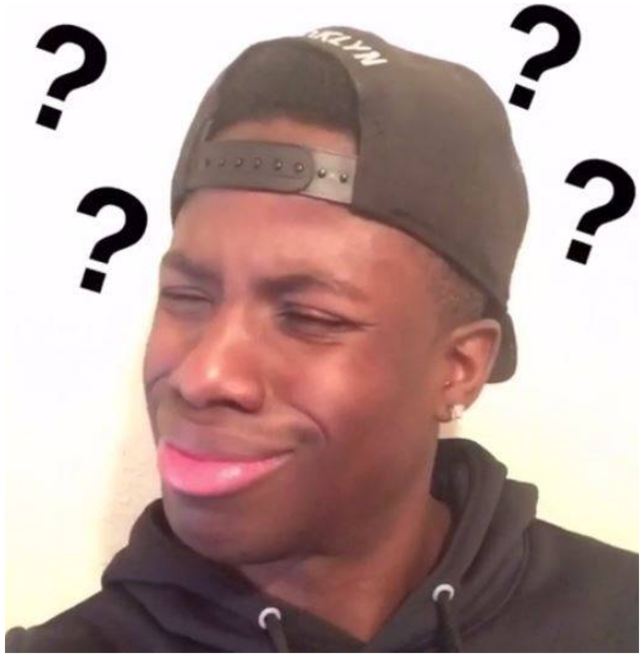
Conclusion

- **The Sequential model is a linear stack of layers.**
- Pros: You can define your network layer-by-layer, you don’t need to take care those layers we have already defined.
- Cons: When you want to modified some layers, you need to re-defined EVERYTHING in your model from the very beginning！

Some Black Magic – Shared Layer

- If we want first 9 weights and last 9 weights to be the same.
- We have no idea how to formulate using Sequential API.



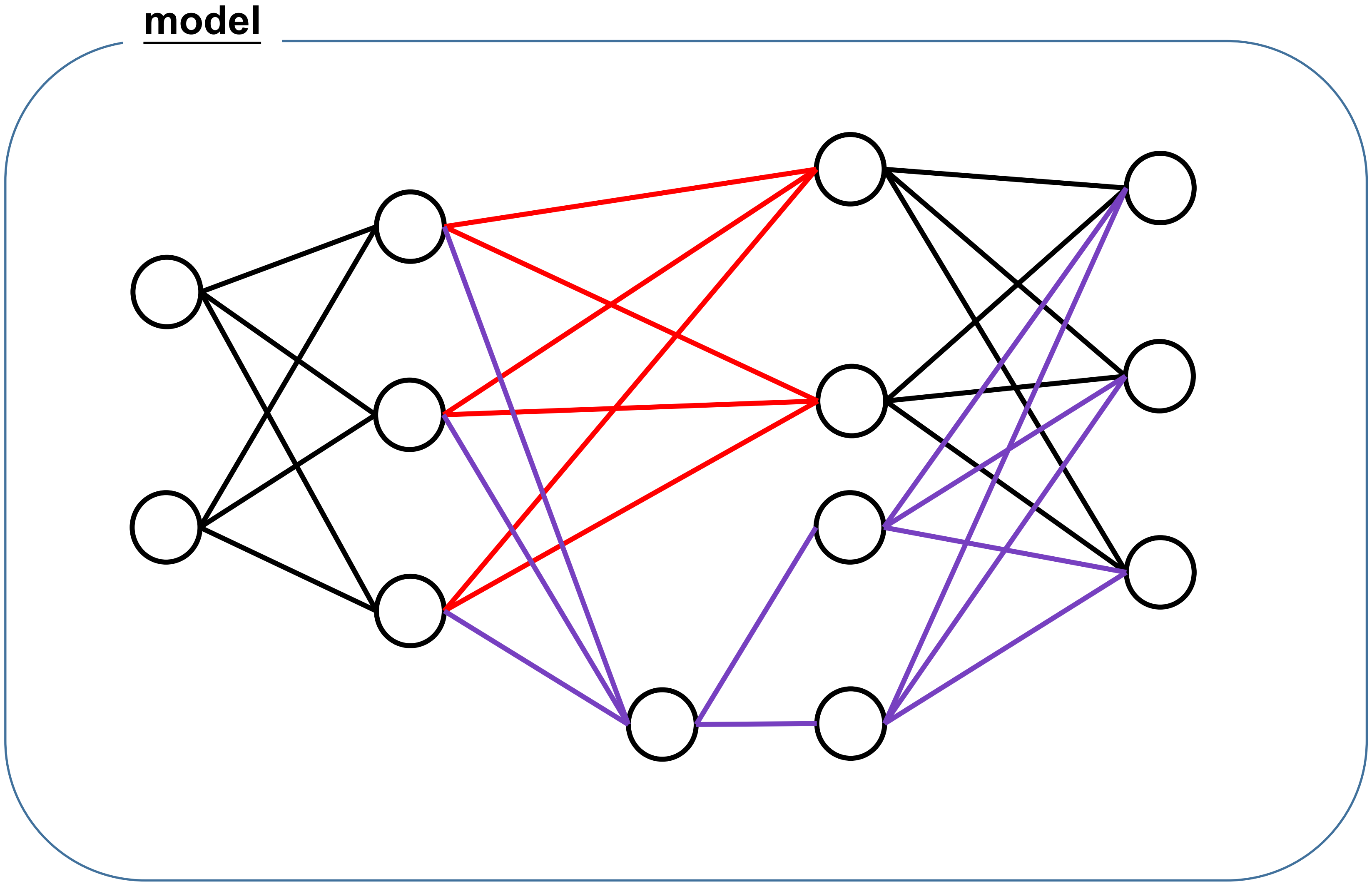


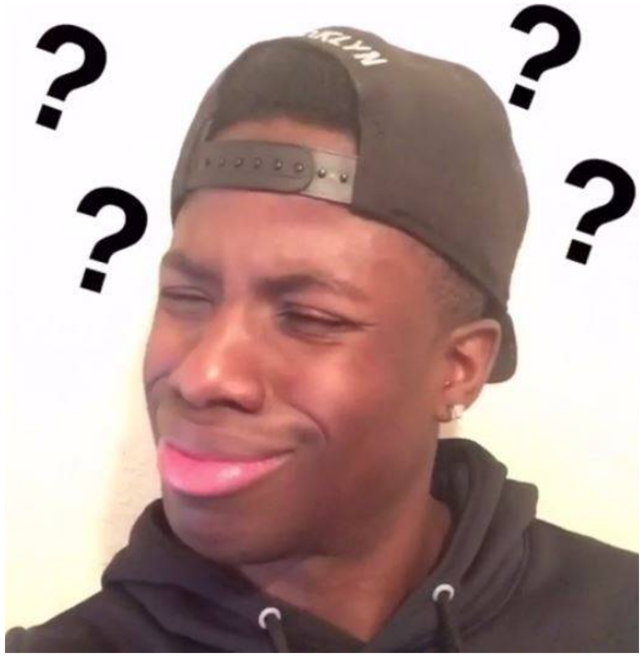
Some Black Magic – Branch or Merge

- If we want our model has branch in some place and merge together in other place.

- We have no idea how to formulate using Sequential API.

Some Black Magic – Branch or Merge





Model
(Functional API)

Sequential v.s. Model

- Sequential: create a “sandbox”, then build your NN model layer-by-layer.
- Model: create the connection between layers (and I/O on each layer), then “model” them as a NN model.
- Some Pros: only need to define the relation of I/O on each layer, easy to create sub-models.

What does Functional means?

- We can use Keras syntax as well as function composition!
- If you like mathematical notations (especial, functions) more than Keras syntax.

You’re welcome !

Syntax to Neural Network Revisit

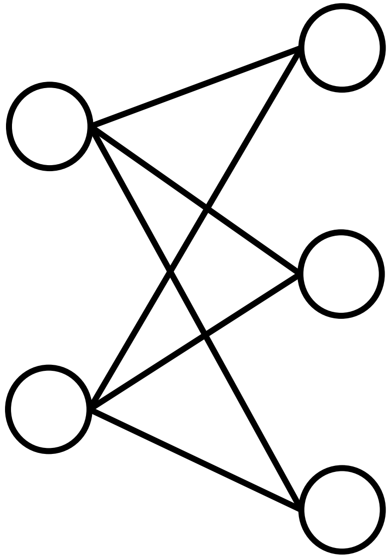
- When enter a Keras syntax and execute it, what would you think?

$\mathbb{R}^2$

Syntax to Neural Network Revisit

- When enter a Keras syntax and execute it, what would you think?

$$\mathbb{R}^2 \xrightarrow{f} \mathbb{R}^3$$

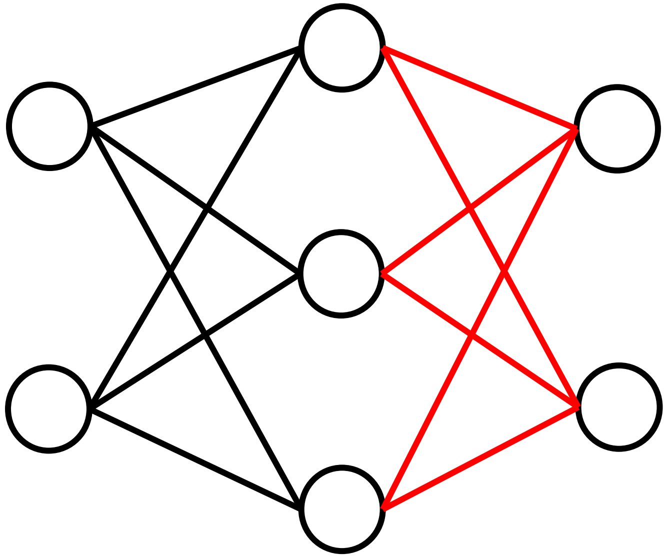


$$f: \mathbb{R}^2 \longrightarrow \mathbb{R}^3$$
$$u = f(x) = \sigma(W^{(1)}x + b^{(1)})$$

Syntax to Neural Network Revisit

- When enter a Keras syntax and execute it, what would you think?

$$\mathbb{R}^2 \xrightarrow{f} \mathbb{R}^3 \xrightarrow{g} \mathbb{R}^2$$



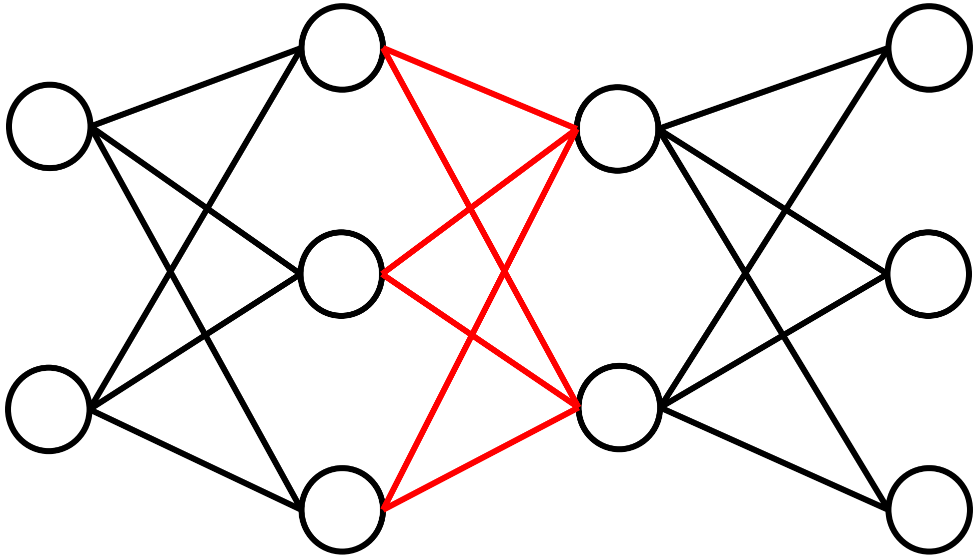
$$g: \mathbb{R}^3 \longrightarrow \mathbb{R}^2$$

$$v = g(u) = \sigma(W^{(2)}u + b^{(2)})$$

Syntax to Neural Network Revisit

- When enter a Keras syntax and execute it, what would you think?

$$\mathbb{R}^2 \xrightarrow{f} \mathbb{R}^3 \xrightarrow{g} \mathbb{R}^2 \xrightarrow{h} \mathbb{R}^3$$



$$h\colon \mathbb{R}^2 \longrightarrow \mathbb{R}^3$$
$$y = h(v) = \sigma\left(W^{(3)}v + b^{(3)}\right)$$

Let’s see how it works!

Syntax to Neural Network Revisit

$\mathbb{R}^2$

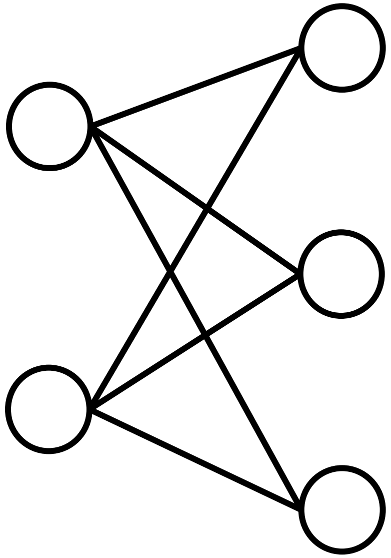
x

```
In [ ]:    x = Input(shape=(2,))
```

Syntax to Neural Network Revisit

$$\mathbb{R}^2 \xrightarrow{f} \mathbb{R}^3$$

$$x \quad \mapsto \quad u$$



$$f: \mathbb{R}^2 \longrightarrow \mathbb{R}^3$$

$$u = f(x) = \sigma(W^{(1)}x + b^{(1)})$$

```
In [ ]:    f = Dense(3, activation='relu')
In [ ]:    u = f(x)
```

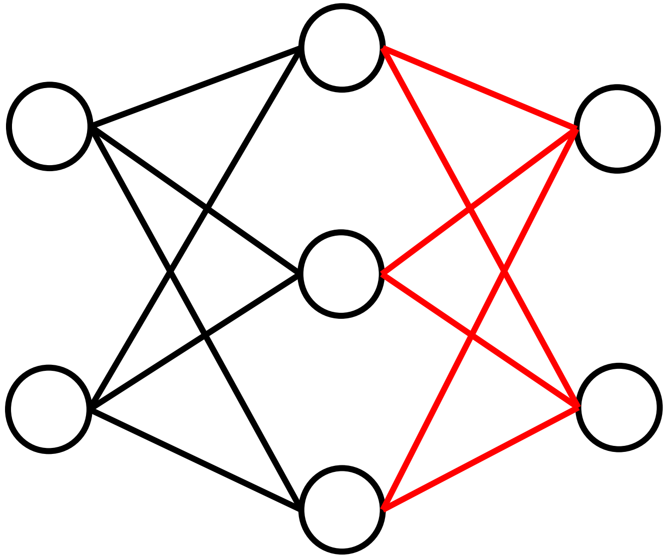
or

```
In [ ]:    u = Dense(3, activation='relu')(x)
```

Syntax to Neural Network Revisit

$$\mathbb{R}^2 \xrightarrow{f} \mathbb{R}^3 \xrightarrow{g} \mathbb{R}^2$$

$$x \mapsto u \mapsto v$$



$$g\colon \mathbb{R}^3 \longrightarrow \mathbb{R}^2$$

$$v = g(u) = \sigma(W^{(2)}u + b^{(2)})$$

```
In [ ]:   g = Dense(2, activation='relu')
In [ ]:   v = g(u)
```

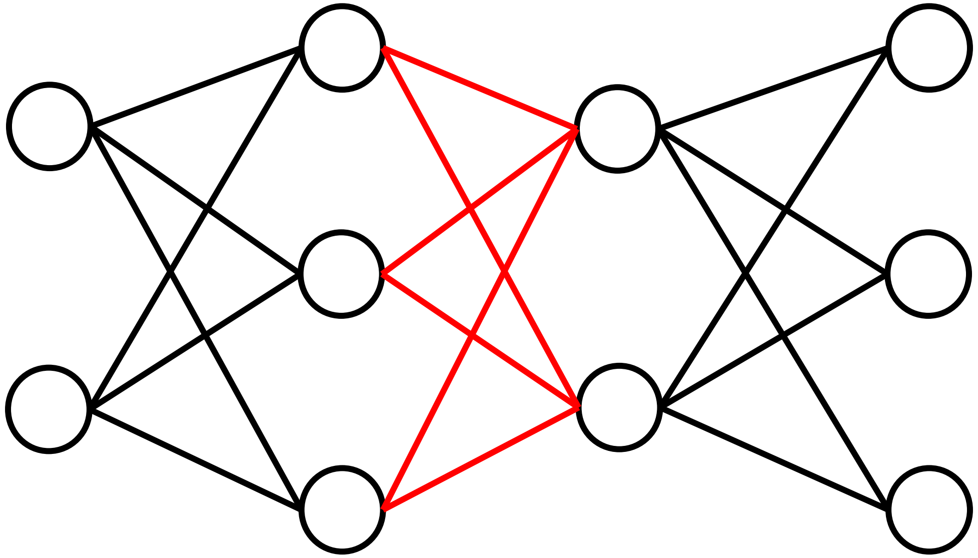
or

```
In [ ]:   v = Dense(2, activation='relu')(u)
```

Syntax to Neural Network: Revisit

$$\mathbb{R}^2 \xrightarrow{f} \mathbb{R}^3 \xrightarrow{g} \mathbb{R}^2 \xrightarrow{h} \mathbb{R}^3$$

$$x \mapsto u \mapsto v \mapsto y$$



$$h: \mathbb{R}^2 \longrightarrow \mathbb{R}^3$$

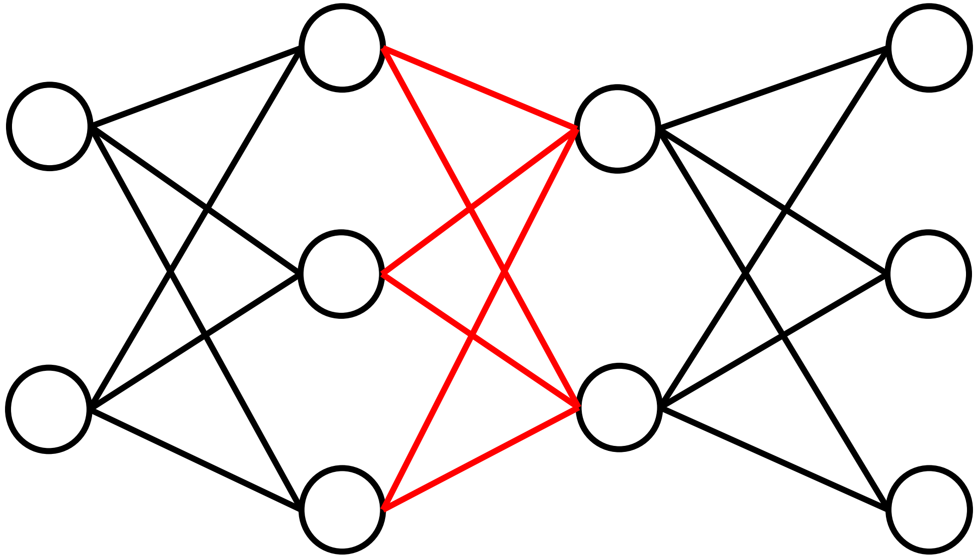
$$y = h(v) = \sigma\left(W^{(3)}v + b^{(3)}\right)$$

```
In [ ]:   h = Dense(3, activation='sigmoid')
In [ ]:   y = h(v)
```

or

```
In [ ]:   y = Dense(3, activation='sigmoid')(v)
```

Syntax to Neural Network Revisit



$$\mathbb{R}^2 \xrightarrow{f} \mathbb{R}^3 \xrightarrow{g} \mathbb{R}^2 \xrightarrow{h} \mathbb{R}^3$$

$$x \mapsto u \mapsto v \mapsto y$$

$$h\colon \mathbb{R}^2 \longrightarrow \mathbb{R}^3$$

$$y = h(v) = \sigma\left(W^{(3)}v + b^{(3)}\right)$$

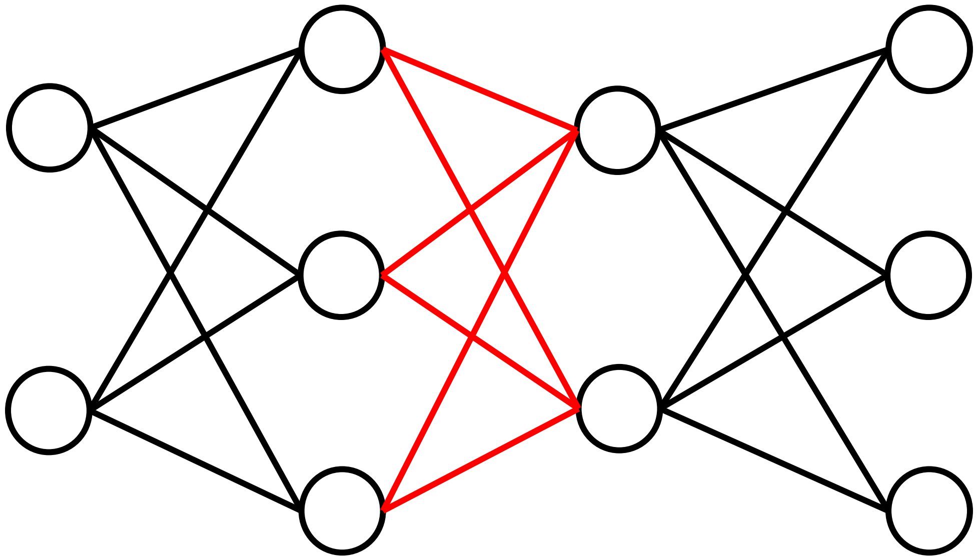
```
In [ ]:   model = Model(x, y)
```

Syntax to Neural Network Revisit

model

$$\mathbb{R}^2 \xrightarrow{f} \mathbb{R}^3 \xrightarrow{g} \mathbb{R}^2 \xrightarrow{h} \mathbb{R}^3$$

$$x \mapsto u \mapsto v \mapsto y$$



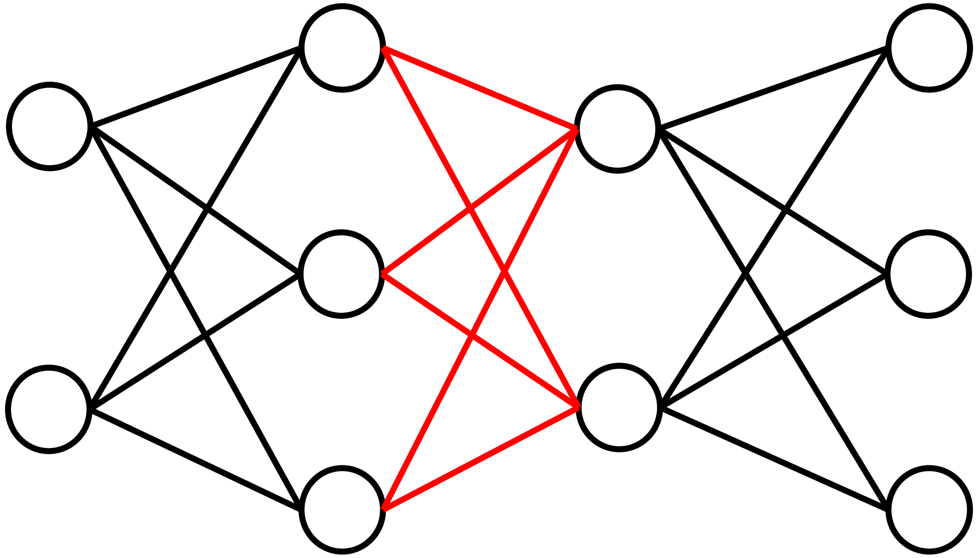
```
In [ ]:   x = Input(shape=(2,))

In [ ]:   f = Dense(3, activation='relu')
In [ ]:   u = f(x)

In [ ]:   g = Dense(2, activation='relu')
In [ ]:   v = g(u)

In [ ]:   h = Dense(3, activation='sigmoid')
In [ ]:   y = h(v)

In [ ]:   model = Model(x, y)
```

Syntax to Neural Network Revisit

model

$$\mathbb{R}^2 \xrightarrow{f} \mathbb{R}^3 \xrightarrow{g} \mathbb{R}^2 \xrightarrow{h} \mathbb{R}^3$$

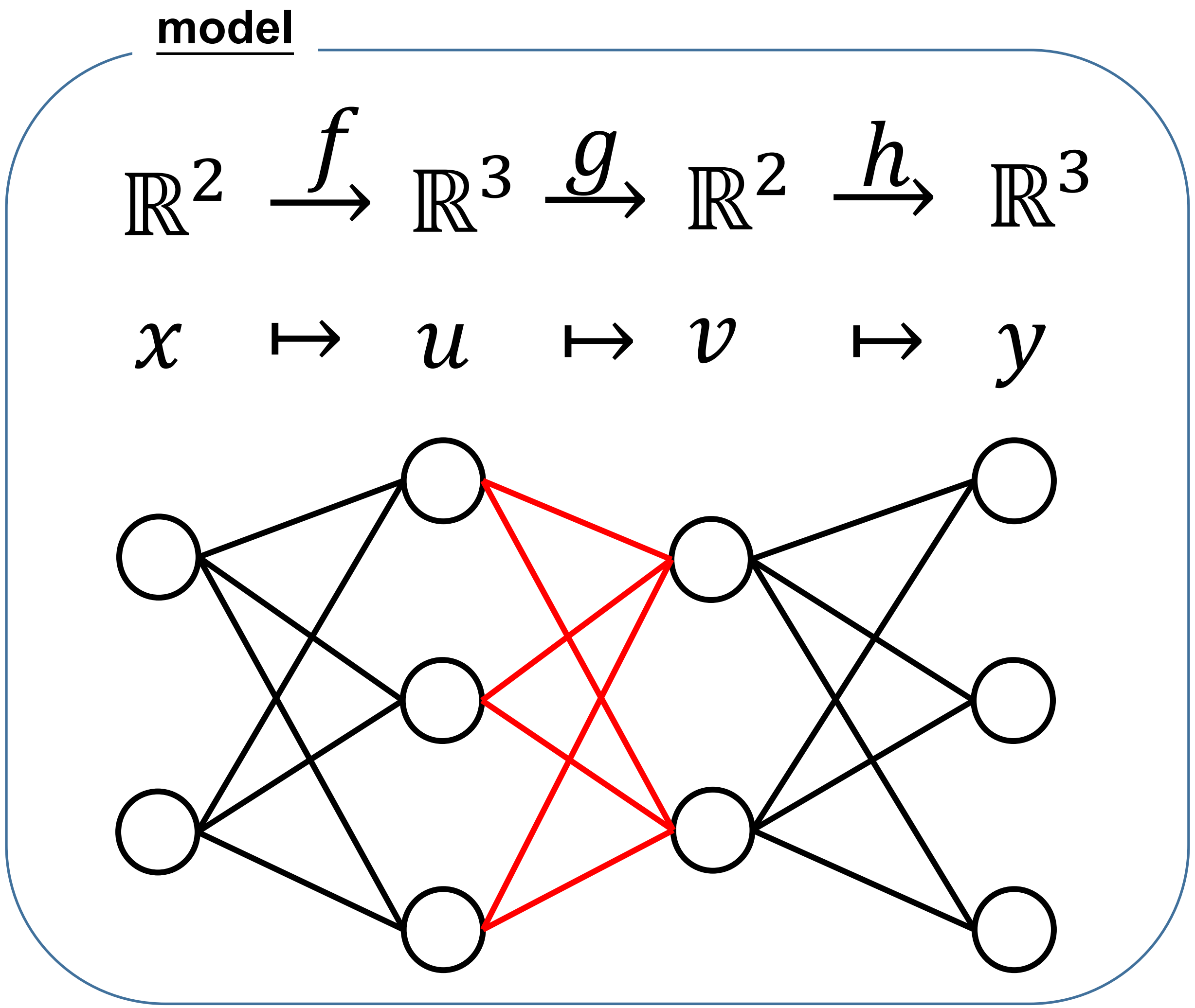
$$x \mapsto u \mapsto v \mapsto y$$



```
In [ ]:   x = Input(shape=(2,))

In [ ]:   u = Dense(3, activation='relu')(x)

In [ ]:   v = Dense(2, activation='relu')(u)

In [ ]:   y = Dense(3, activation='sigmoid')(v)

In [ ]:   model = Model(x, y)
```

This is the same as what you did when using Sequential API

Syntax to Neural Network Revisit



In [ ]: x = Input(shape=(2,))

In [ ]: u = Dense(3, activation='relu')(x)

In [ ]: v = Dense(2, activation='relu')(u)

In [ ]: y = Dense(3, activation='sigmoid')(v)

In [ ]: model = Model(x, y)

This is the same as what you did when using Sequential API

Conclusion

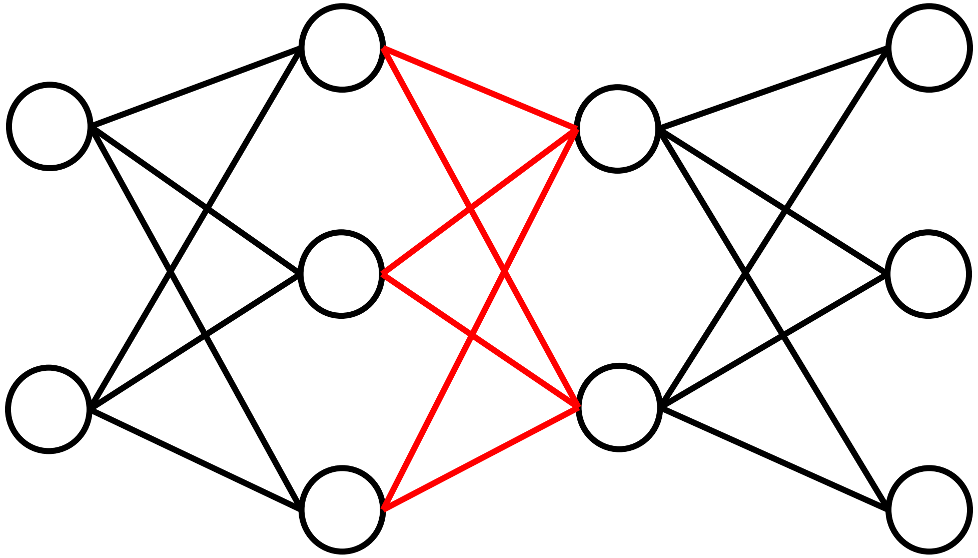
- Model Functional API defines the relation of I/O of each layer as well as a function.
- Variables from each layer are all available in Model case other than only I/O in Sequential case.
- Latent variables can be manipulated by user.

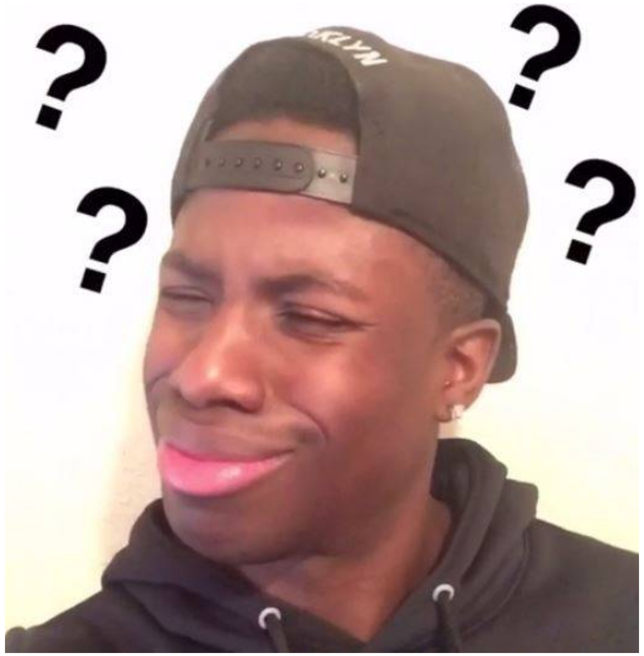
Black Magic Revisit – Shared Layer

- If we want first 9 weights and last 9 weights to be the same.
- How to formulate using Model API?

model

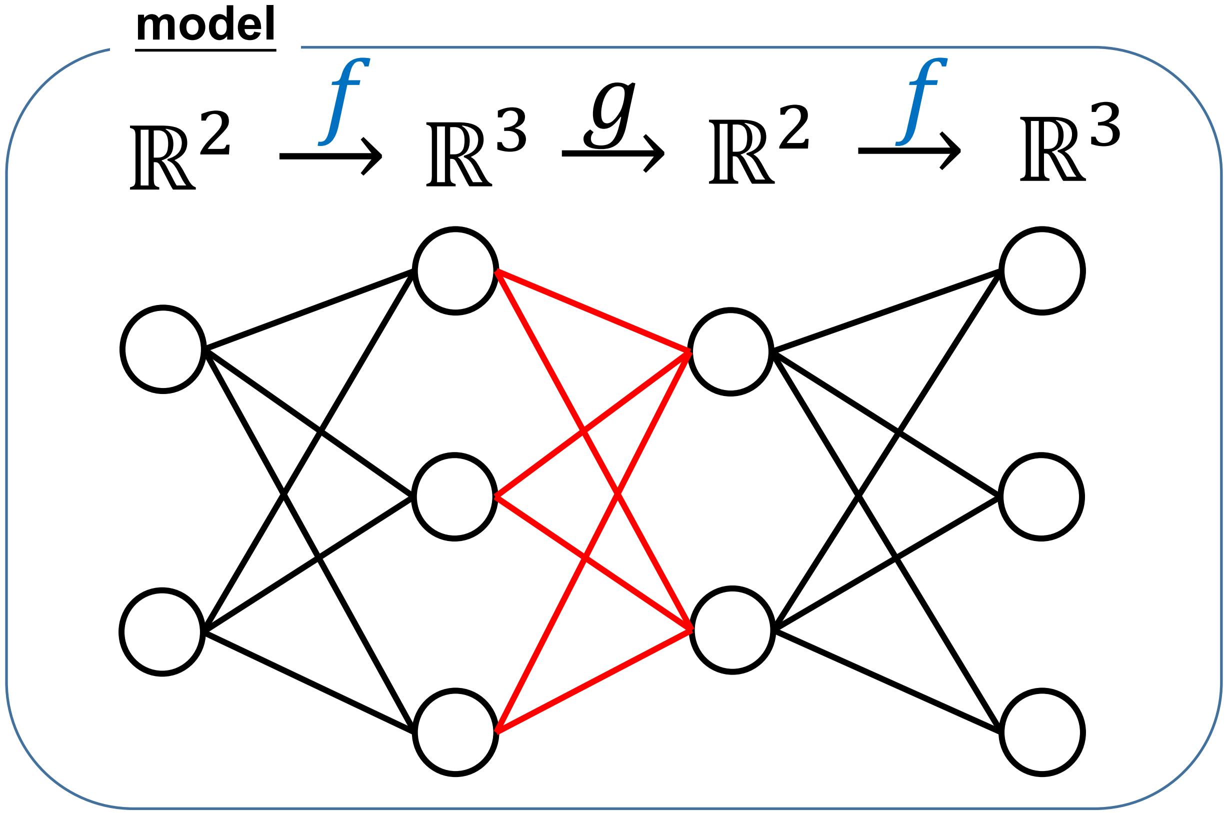
$$\mathbb{R}^2 \xrightarrow{f} \mathbb{R}^3 \xrightarrow{g} \mathbb{R}^2 \xrightarrow{f} \mathbb{R}^3$$





Black Magic Revisit – Shared Layer

- If we want first 9 weights and last 9 weights to be the same.
- How to formulate using Model API?



$$f : \mathbb{R}^2 \longrightarrow \mathbb{R}^3$$
$$u = f(x) = \sigma\left(W^{(1)}x + b^{(1)}\right)$$

$$g : \mathbb{R}^3 \longrightarrow \mathbb{R}^2$$
$$v = g(u) = \sigma\left(W^{(2)}u + b^{(2)}\right)$$

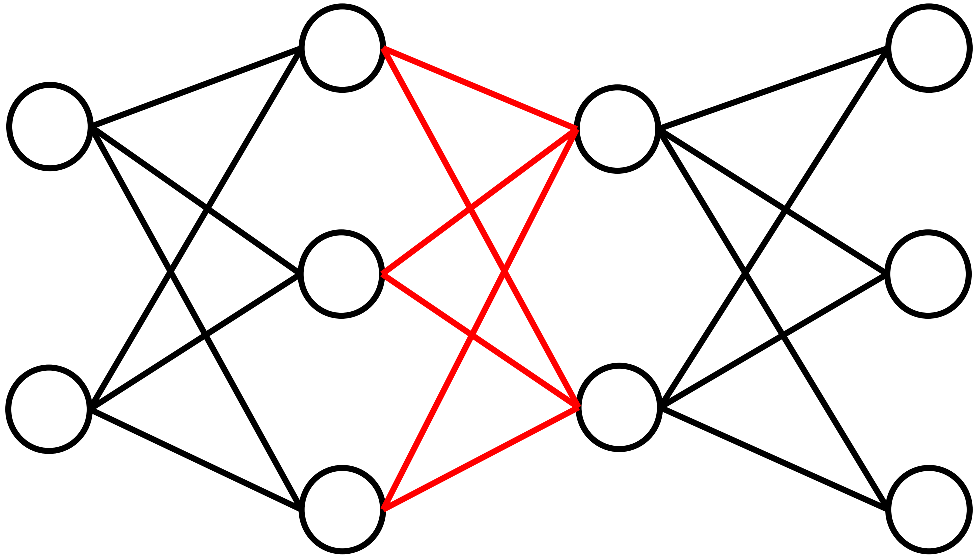
$$y = f(v) = \sigma\left(W^{(3)}v + b^{(3)}\right)$$

Black Magic Revisit – Shared Layer

- If we want first 9 weights and last 9 weights to be the same.
- How to formulate using Model API?

model

$$\mathbb{R}^2 \xrightarrow{f} \mathbb{R}^3 \xrightarrow{g} \mathbb{R}^2 \xrightarrow{f} \mathbb{R}^3$$



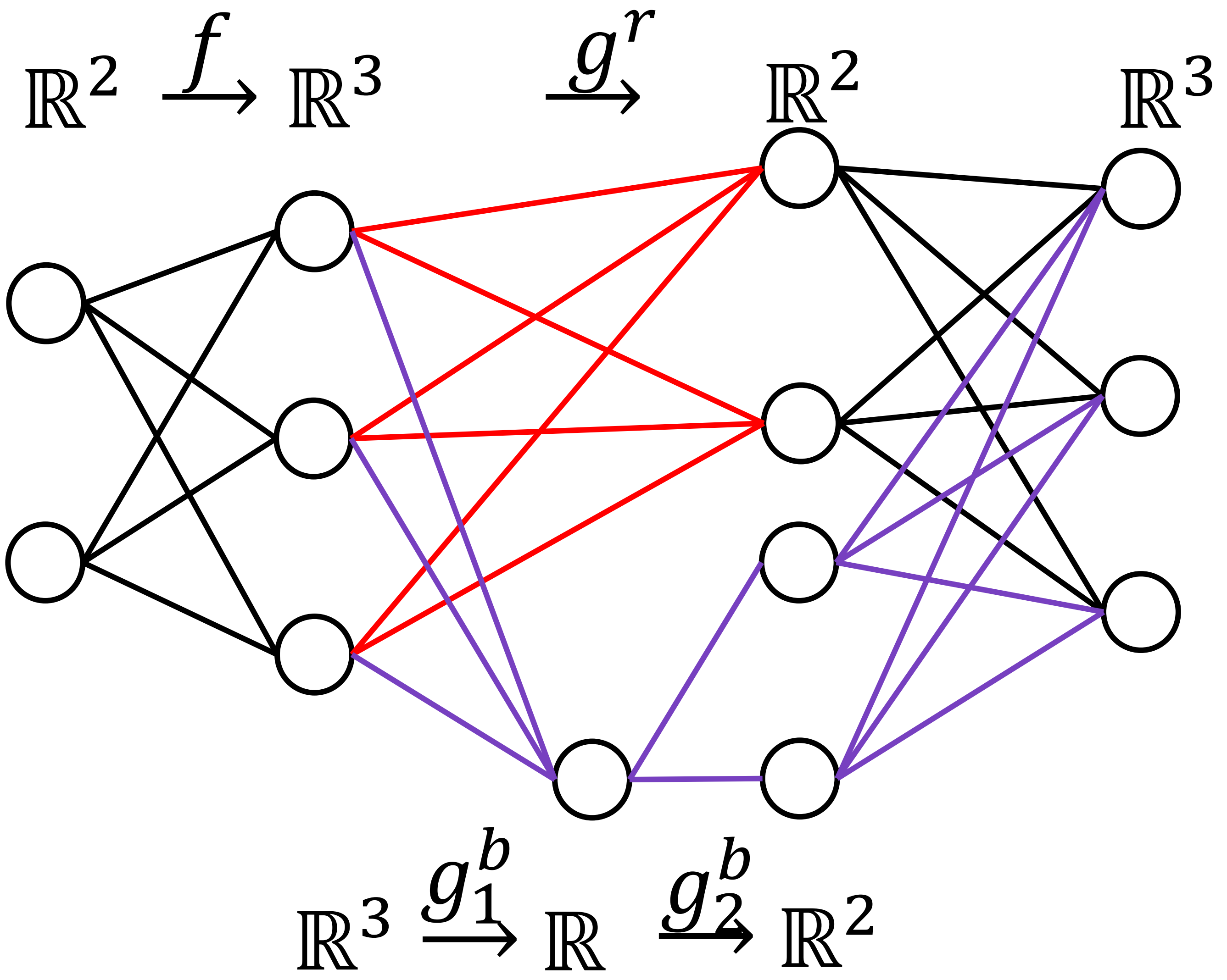
In [ ]: x = Input(shape=(2,))

In [ ]: f = Dense(3, activation='relu')
In [ ]: u = f(x)

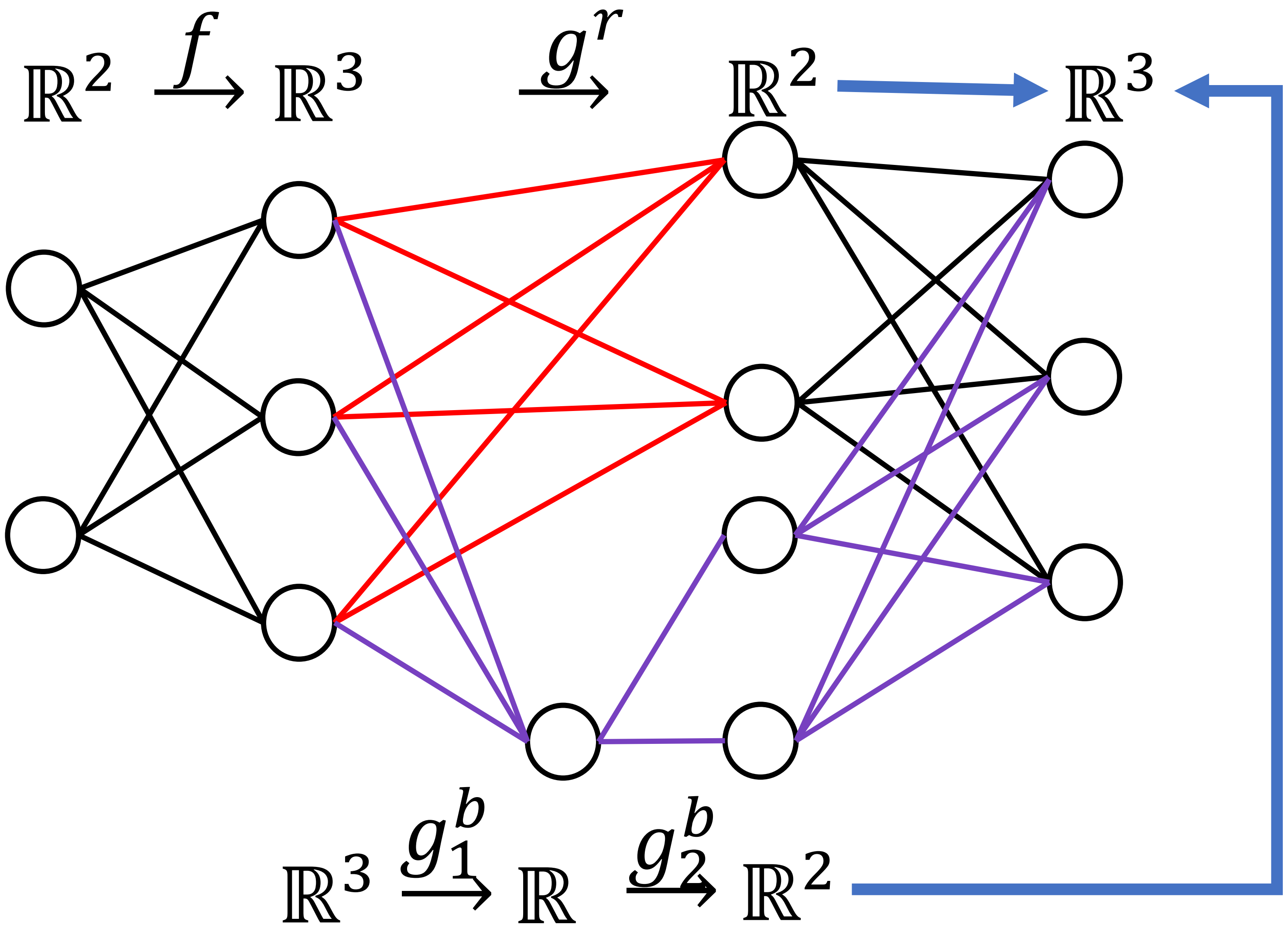
In [ ]: g = Dense(2, activation='relu')
In [ ]: v = g(u)

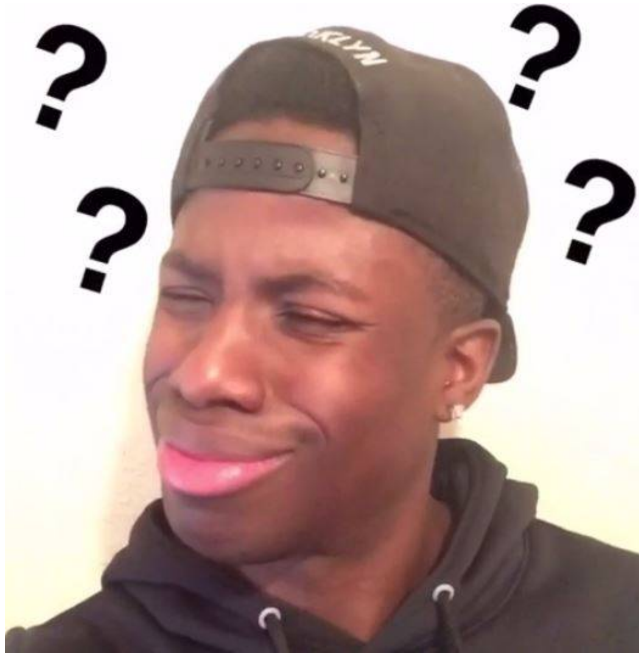
~~In [ ]: h = Dense(3, activation='sigmoid')~~
~~In [ ]: y = h(v)~~
In [ ]: y = f(v)
In [ ]: model = Model(x, y)

Black Magic Revisit – Branch or Merge

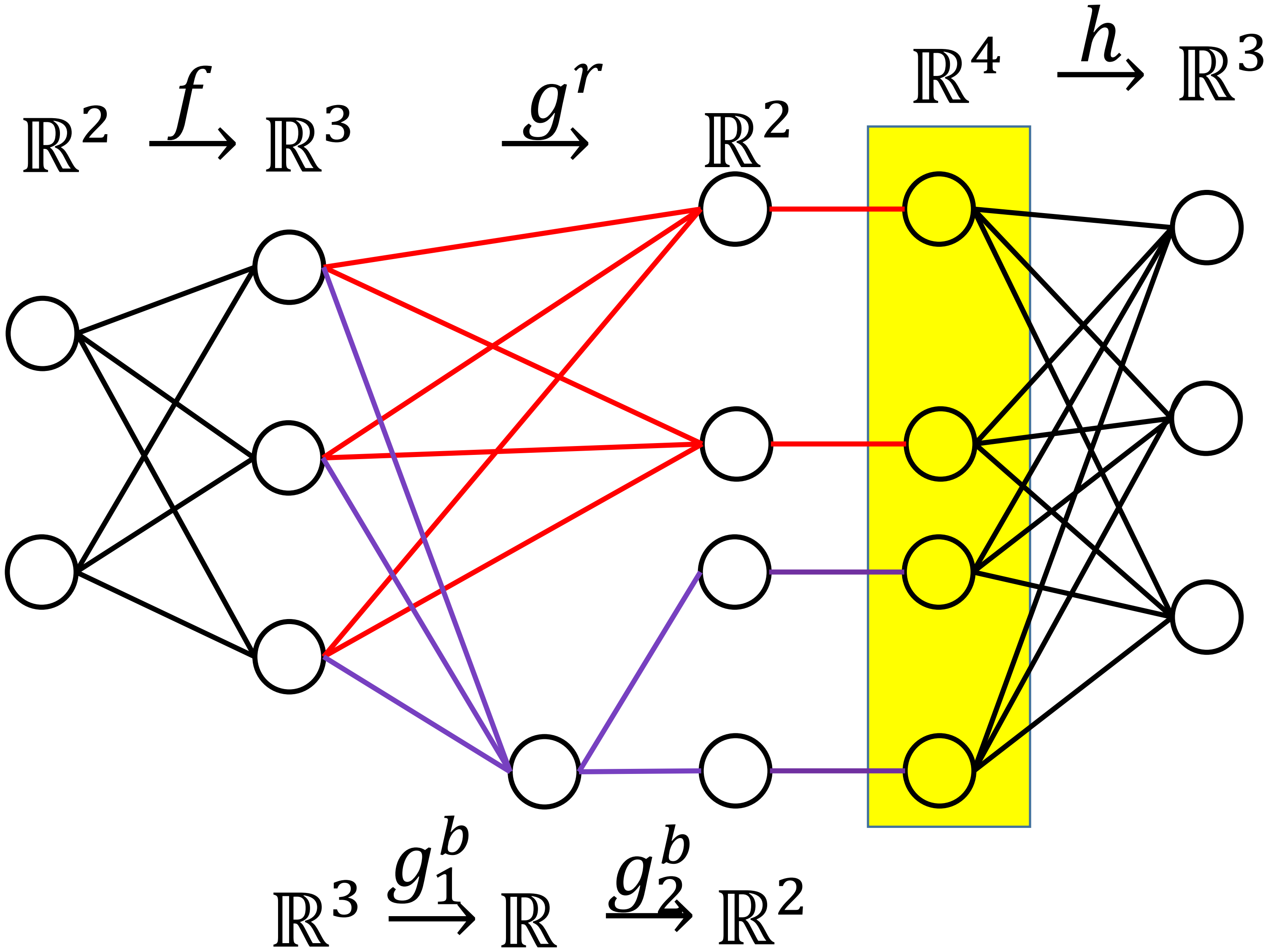


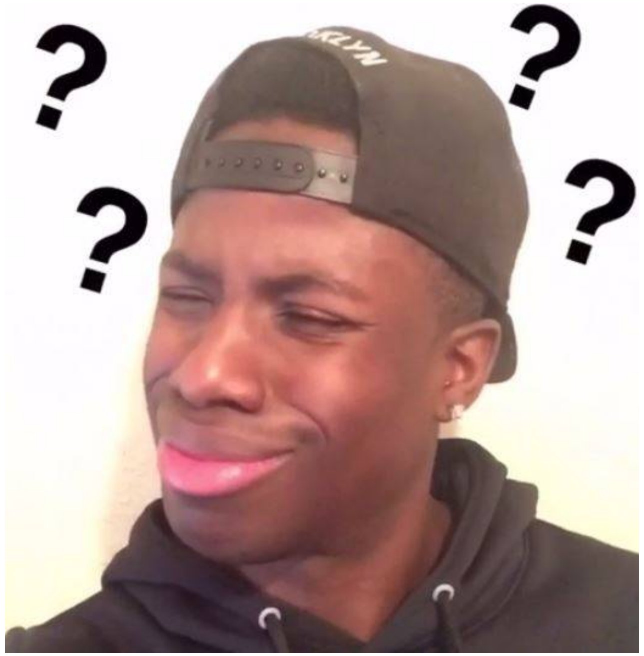
Black Magic Revisit – Branch or Merge





Black Magic Revisit – Branch or Merge

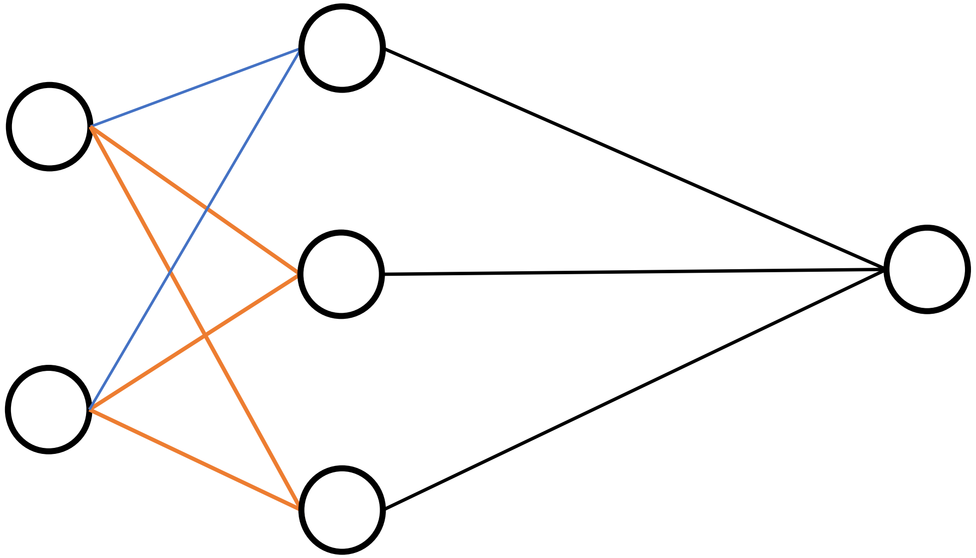


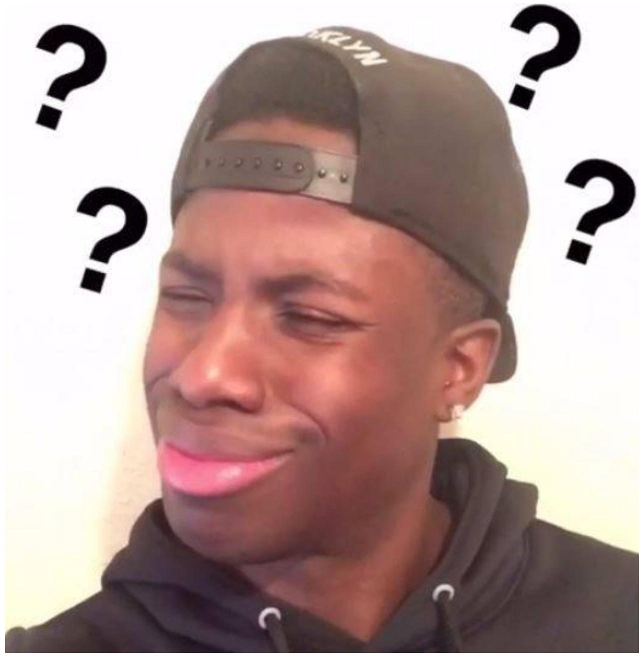


Black Magic Revisit – Branch or Merge

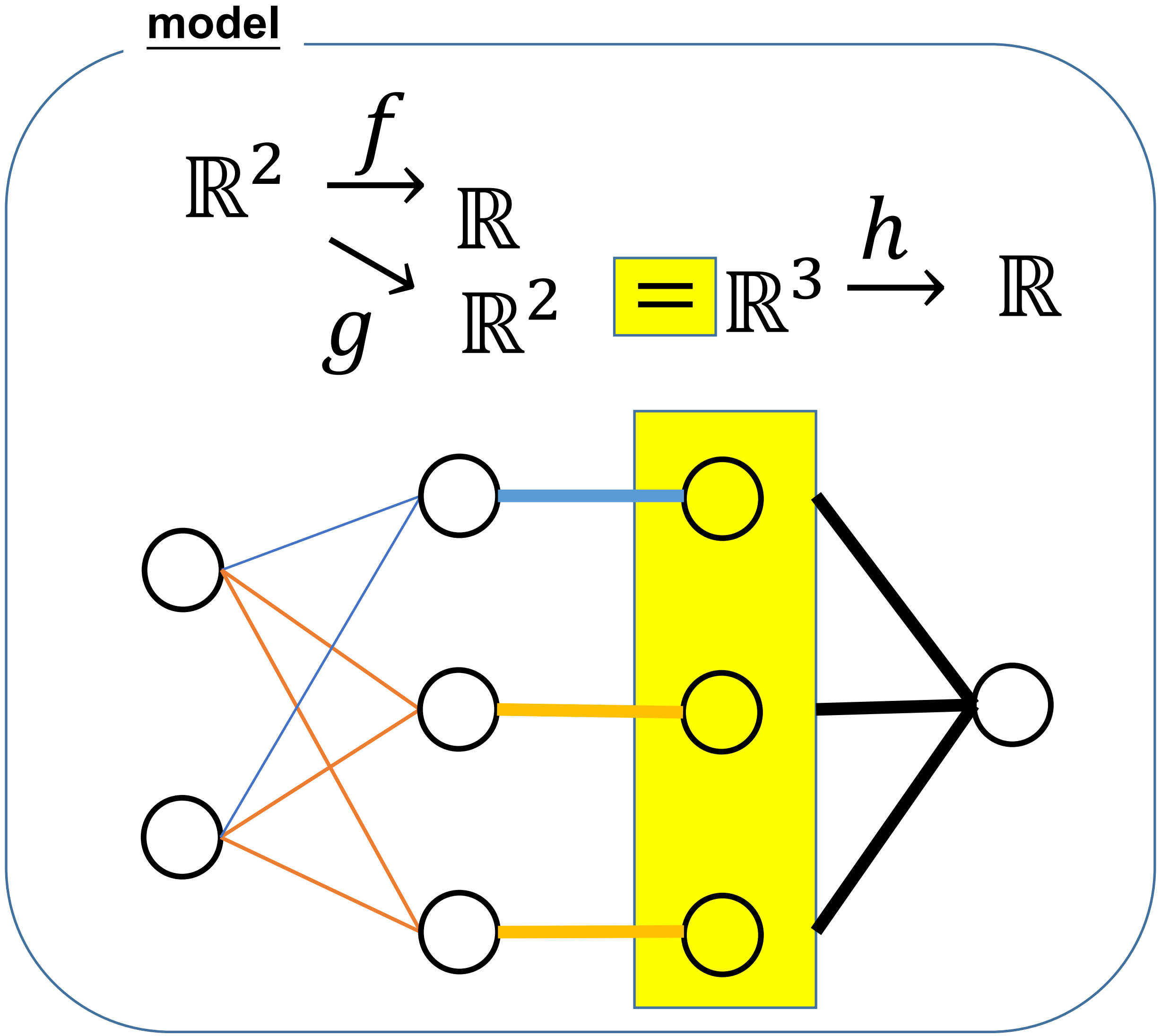
model

$$\begin{tikzcd} \mathbb{R}^2 \arrow[r, "f"] \arrow[dr, "g"'] & \mathbb{R} \\ & \mathbb{R}^2 \end{tikzcd} \qquad \xrightarrow{h} \mathbb{R}$$





Black Magic Revisit – Branch or Merge



In [ ]: x = Input(shape=(2,))

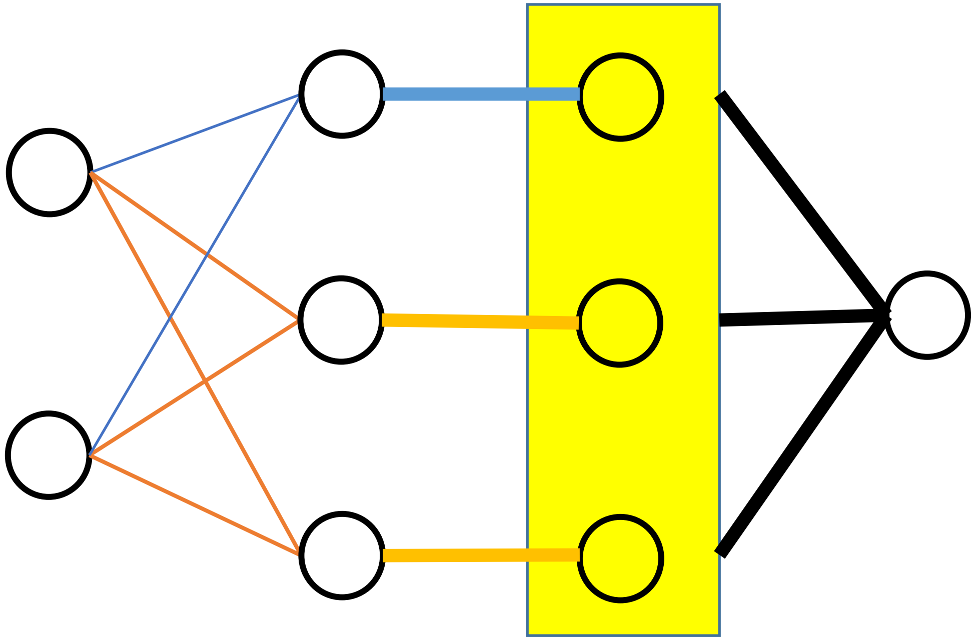
In [ ]: f = Dense(3, activation='relu')
In [ ]: u_1 = f(x)

In [ ]: g = Dense(2, activation='relu')
In [ ]: u_2= g(u)

Black Magic Revisit – Branch or Merge

model

$$\begin{tikzcd} \mathbb{R}^2 \arrow[r, "f"] \arrow[dr, "g"'] & \mathbb{R} & & \\ & \mathbb{R}^2 & \boxed{=} \mathbb{R}^3 \arrow[r, "h"] & \mathbb{R} \end{tikzcd}$$

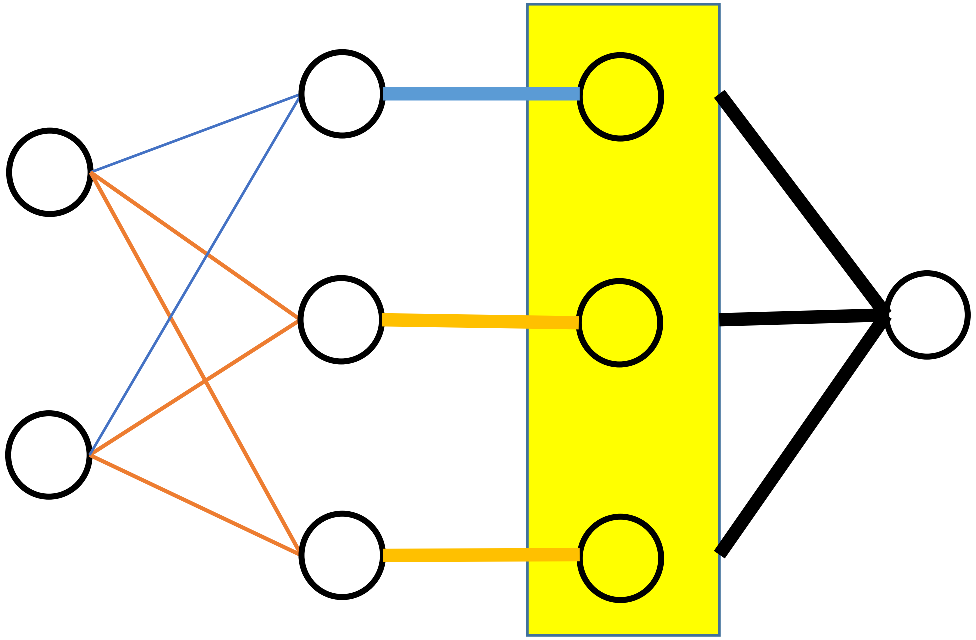


```
In [ ]:   x = Input(shape=(2,))

In [ ]:   f = Dense(3, activation='relu')
In [ ]:   u_1 = f(x)

In [ ]:   g = Dense(2, activation='relu')
In [ ]:   u_2= g(u)

In [ ]:   merged_v = concatenate([u_1, u_2])
```

Black Magic Revisit – Branch or Merge

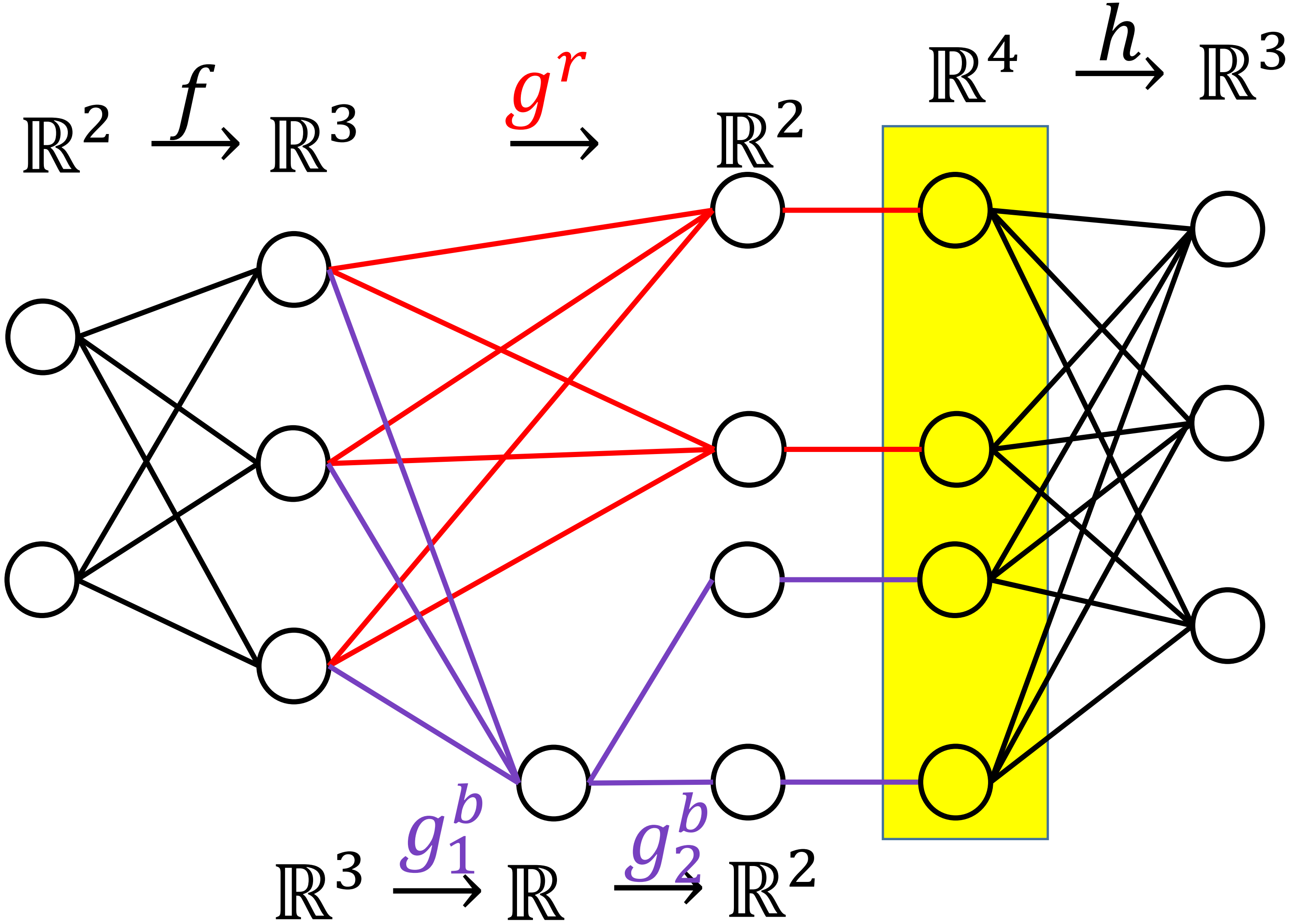
<u>**model**</u>

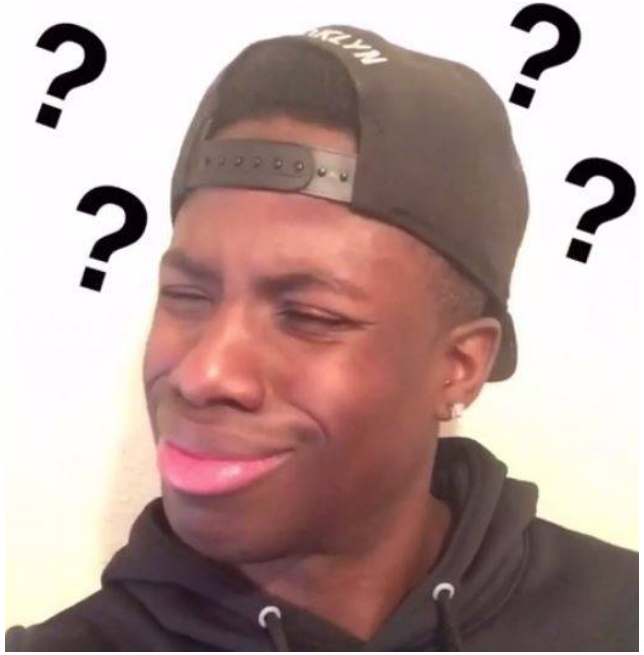
$$\mathbb{R}^2 \begin{array}{l} \xrightarrow{f} \mathbb{R} \\ \underset{g}{\searrow} \mathbb{R}^2 \end{array} \boxed{=} \mathbb{R}^3 \xrightarrow{h} \mathbb{R}$$



```
In [ ]:   x = Input(shape=(2,))

In [ ]:   f = Dense(1, activation='relu')
In [ ]:   u_1 = f(x)

In [ ]:   g = Dense(2, activation='relu')
In [ ]:   u_2= g(u)

In [ ]:   merged_v = concatenate([u_1, u_2])

In [ ]:   h = Dense(1, activation='softmax')
In [ ]:   y = h(merged_v)

In [ ]:   model = Model(x, y)
```

Black Magic Revisit – Challenge



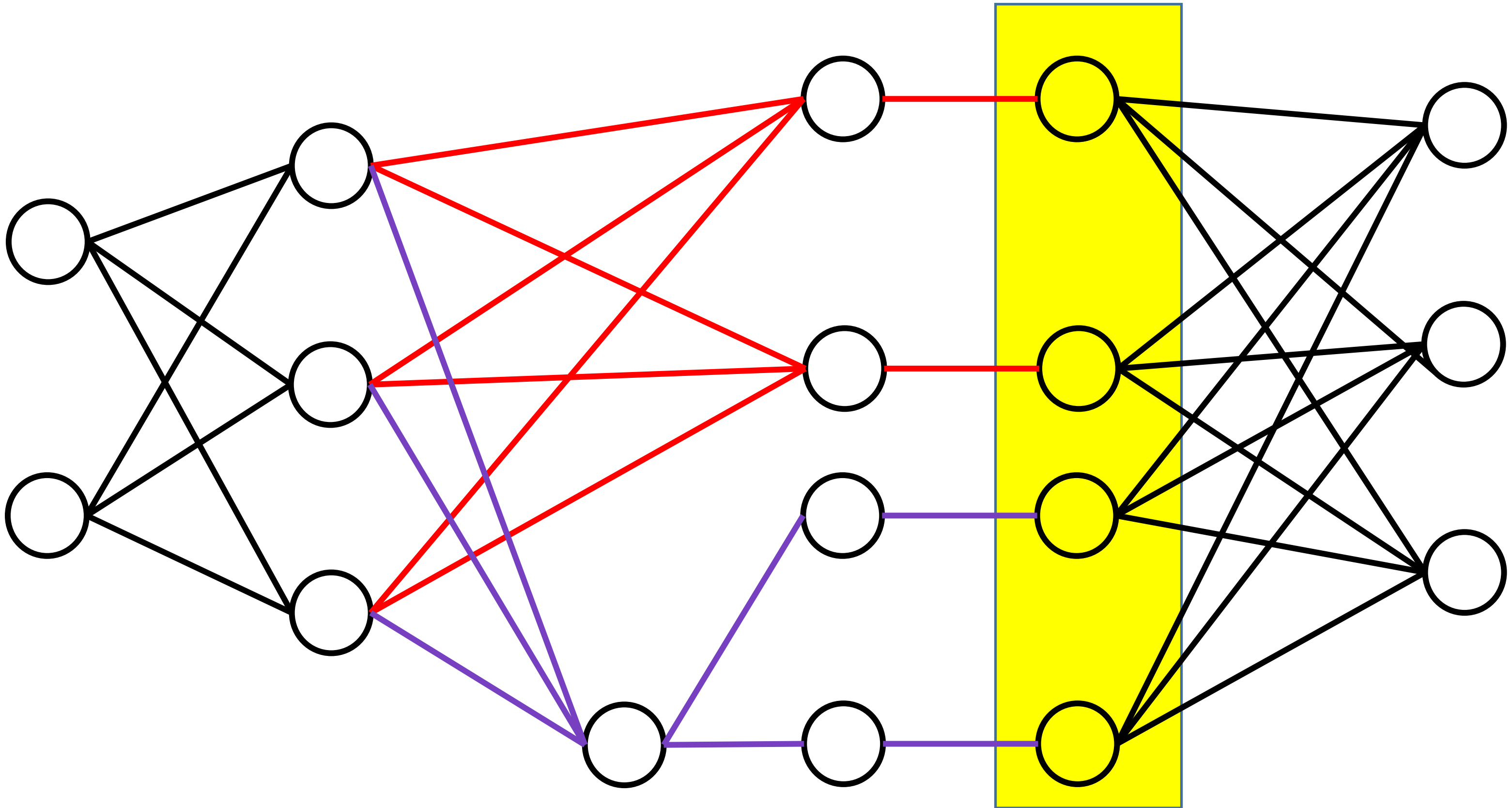


Hint:

$$\xrightarrow{g^r} \quad \mathbb{R}^2$$

$$\mathbb{R}^2 \xrightarrow{f} \mathbb{R}^3 \qquad \boxed{=} \mathbb{R}^4 \xrightarrow{h} \mathbb{R}^3$$

$$\xrightarrow{g_1^b} \mathbb{R} \xrightarrow{g_2^b} \mathbb{R}^2$$



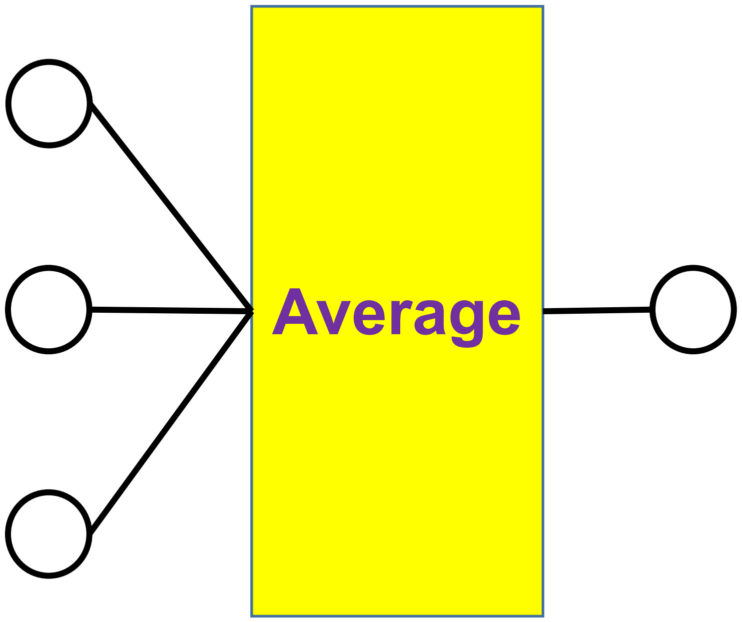
Black Magic Revisit – Conclusion

- A common application of Brach-and-Merge is to build Multi-input and multi-output models.

Ref: https://keras.io/getting-started/functional-api-guide/#multi-input-and-multi-output-models

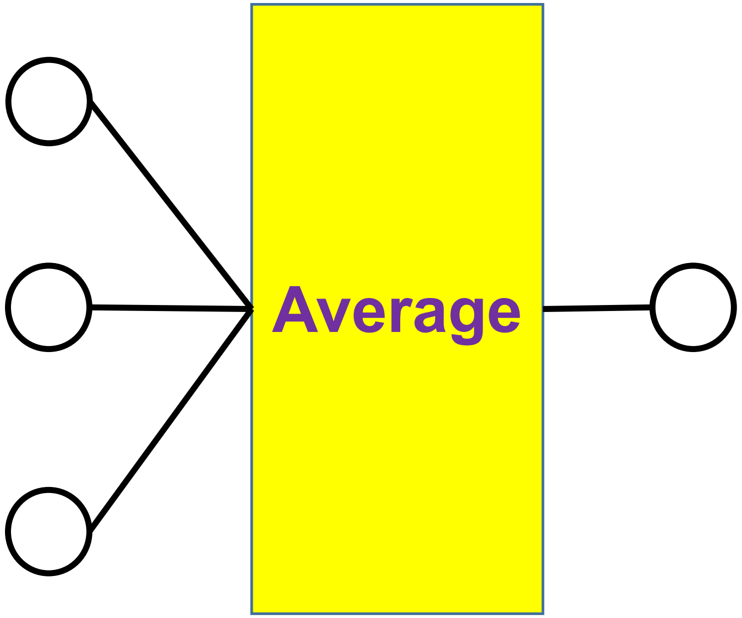
Customized Layer

- For example, we want a layer which output the average of the input neurons



Customized Layer

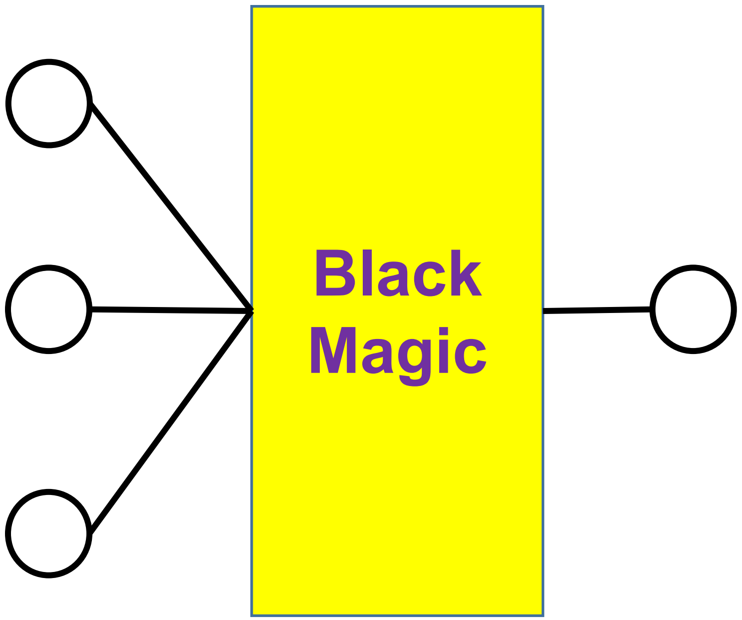
- Lambda layer transfers customized function as a layer.
- This could be also used when using Sequential API.




```
def my_avergae(args):
    return K.mean(args)

x = Input(shape=(3,), name='x_is_input')

f = Lambda(my_avergae, output_shape=(1,))

y = f(x)
model = Model(x, y)
```

Customized Layer

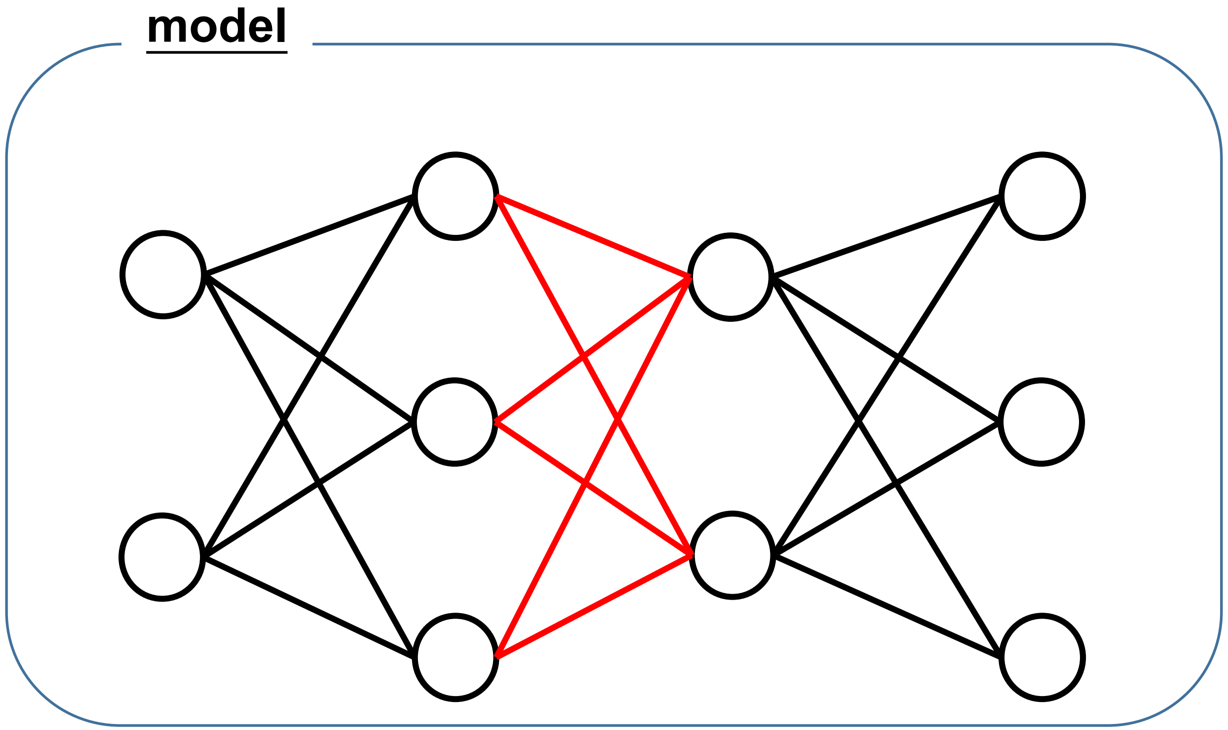
- Lambda layer is used for customized function as layer behavior.
- This could be also used when using Sequential API.




```
def my_spell(args):
    xxx
    return black_magic
x = Input(shape=(3,), name='x_is_input')

f = Lambda(my_spell, output_shape=(1,))

y = f(x)
model = Model(x, y)
```

Customized Loss Function

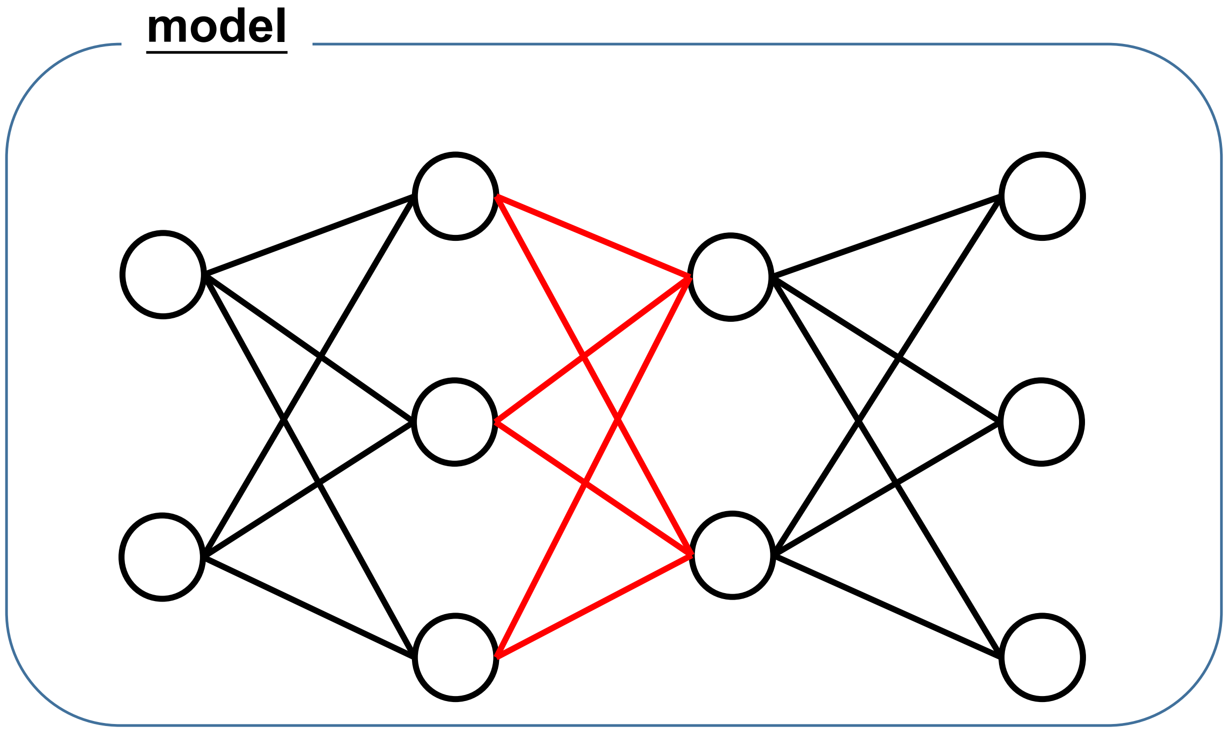
- Can we use user-defined loss function?




```
In [ ]:   x = Input(shape=(2,))
In [ ]:   f = Dense(3, activation='relu')
In [ ]:   u = f(x)
In [ ]:   g = Dense(2, activation='relu')
In [ ]:   v = g(u)
In [ ]:   h = Dense(3, activation='sigmoid')
In [ ]:   y = h(v)
In [ ]:   model = Model(x, y)
In [ ]:   model.compile(loss='mse')
```

Customized Loss Function

- Can we use user-defined loss function?
- Yes, as shown below. Another example is VAE.



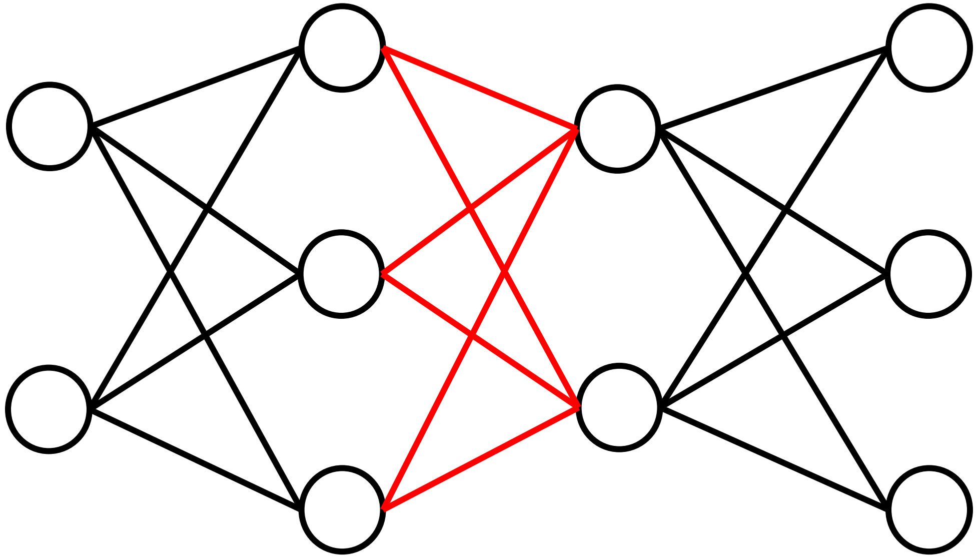

```
In [ ]:  x = Input(shape=(2,))
In [ ]:  f = Dense(3, activation='relu')
In [ ]:  u = f(x)
In [ ]:  g = Dense(2, activation='relu')
In [ ]:  v = g(u)
In [ ]:  h = Dense(3, activation='sigmoid')
In [ ]:  y = h(v)
In [ ]:  model = Model(x, y)
```

```
In [ ]:  def my_loss(y_true, y_pred):
             return K.mean(y_true - y_pred)

In [ ]:  model.compile(loss=my_loss)
```

Customized Loss Function

- In most case, the order of y_true and y_pred is not important
- However, for cross-entropy-like loss function, it's IMPORTANT

model

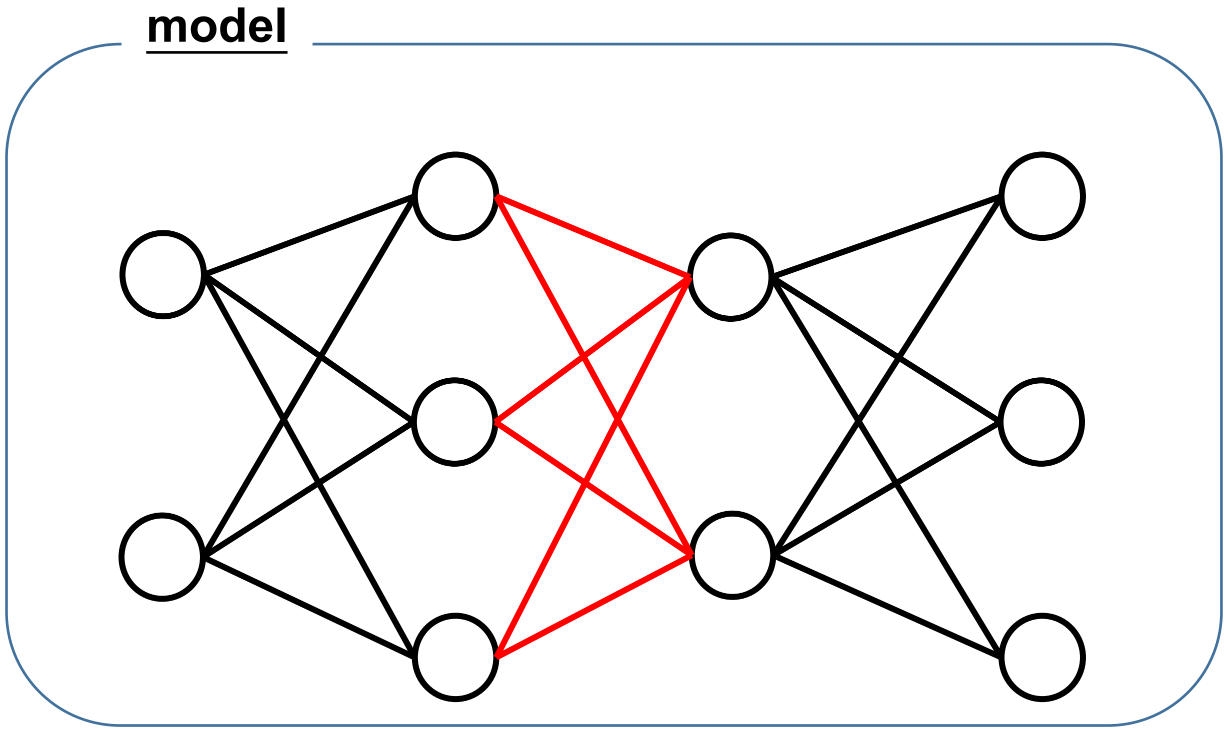


```
In [ ]:   x = Input(shape=(2,))
In [ ]:   f = Dense(3, activation='relu')
In [ ]:   u = f(x)
In [ ]:   g = Dense(2, activation='relu')
In [ ]:   v = g(u)
In [ ]:   h = Dense(3, activation='sigmoid')
In [ ]:   y = h(v)
In [ ]:   model = Model(x, y)
```

```
In [ ]:   def my_loss(y_true, y_pred):
              return K.mean(y_true - y_pred)

In [ ]:   model.compile(loss=my_loss)
```

Customized Loss Function

- Now, try to define some loss function using functions provided from backend.



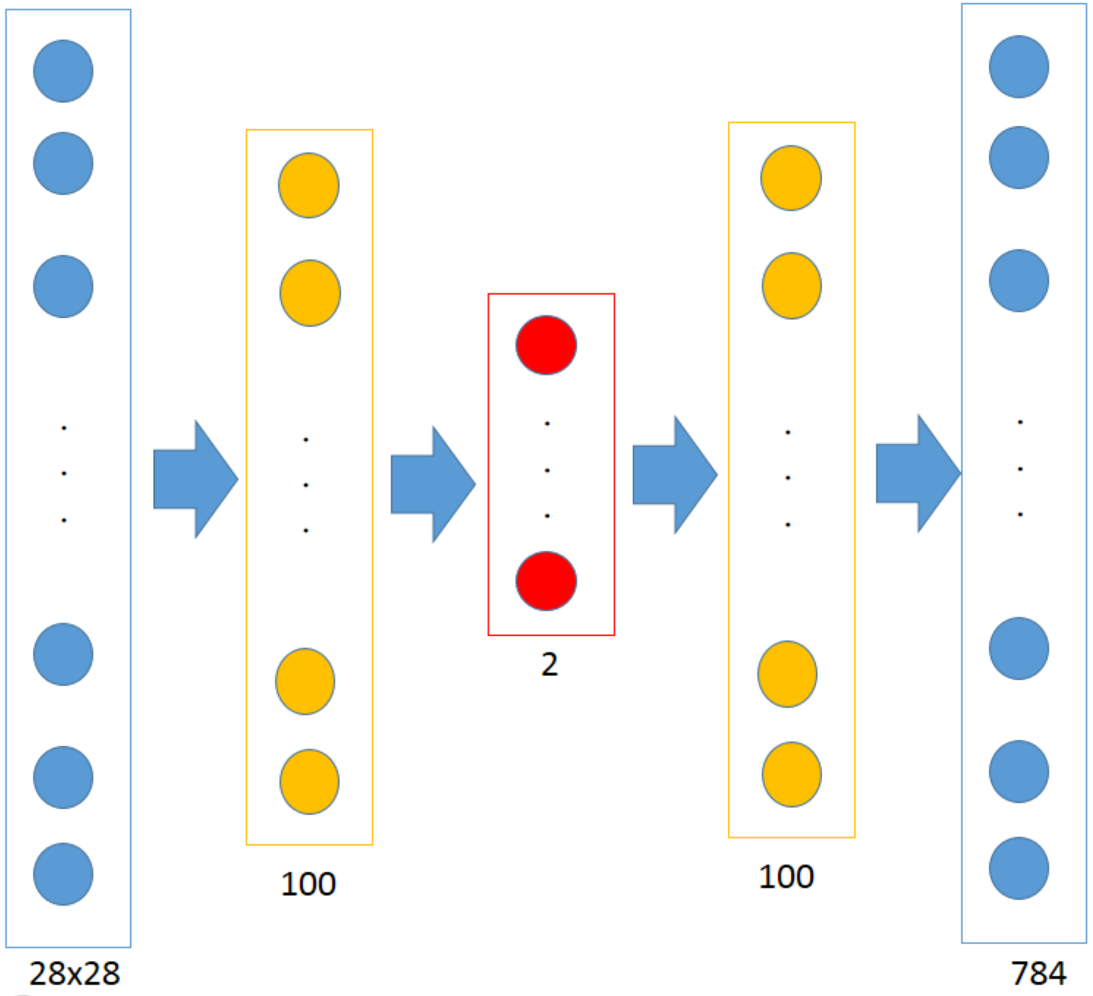

```
In [ ]:   x = Input(shape=(2,))
In [ ]:   f = Dense(3, activation='relu')
In [ ]:   u = f(x)
In [ ]:   g = Dense(2, activation='relu')
In [ ]:   v = g(u)
In [ ]:   h = Dense(3, activation='sigmoid')
In [ ]:   y = h(v)
In [ ]:   model = Model(x, y)
```

```
In [ ]:   def my_loss(y_true, y_pred):
              xxx
              return ooo
In [ ]:   model.compile(loss=my_loss)
```

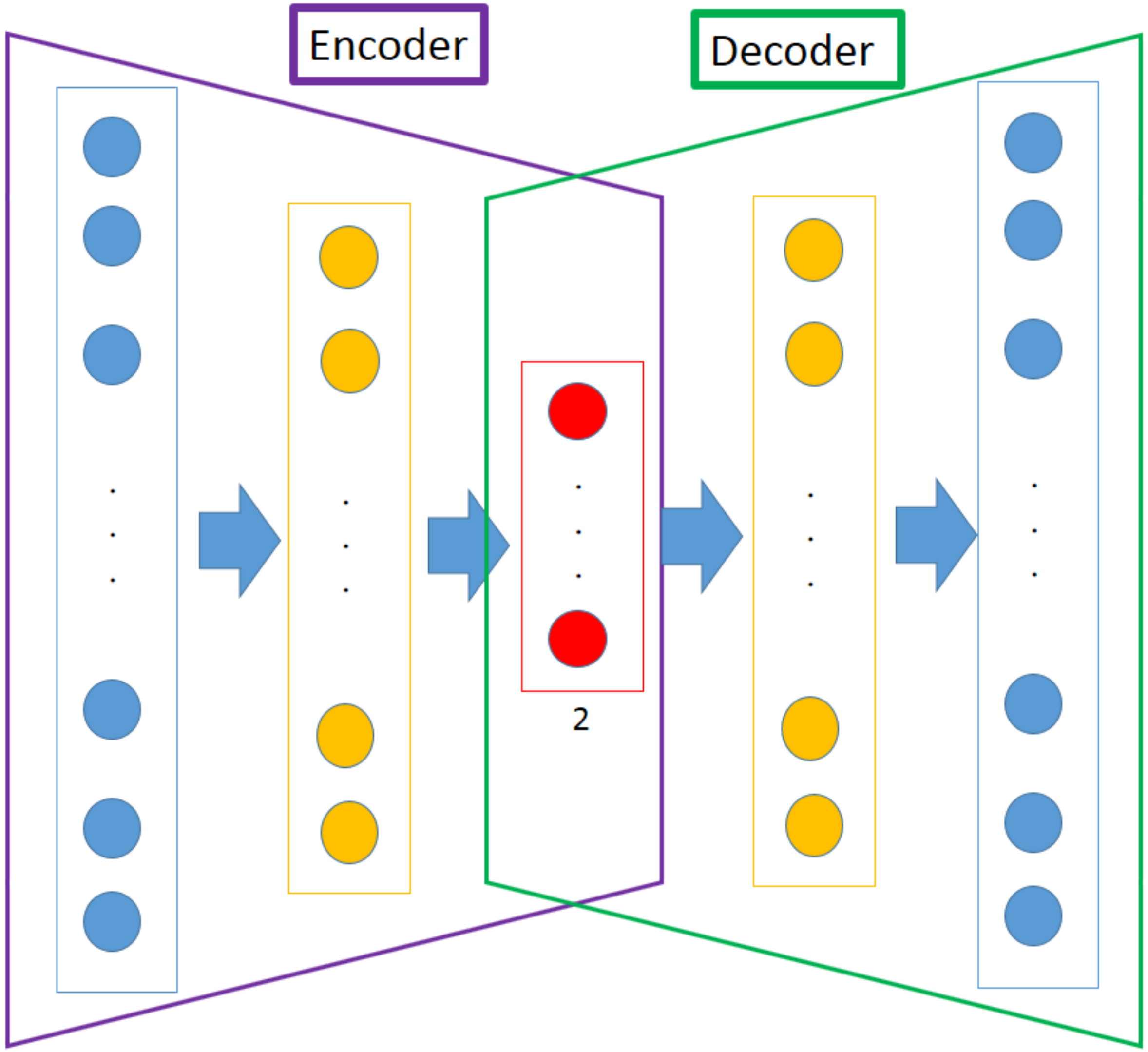
Submodels

- When we define a NN model, sometimes, we don’t care about output value or class.
- Usually, neurons in the hidden layers are of interesting.

Submodels – Autoencoder as Example



Submodels – Autoencoder as Example

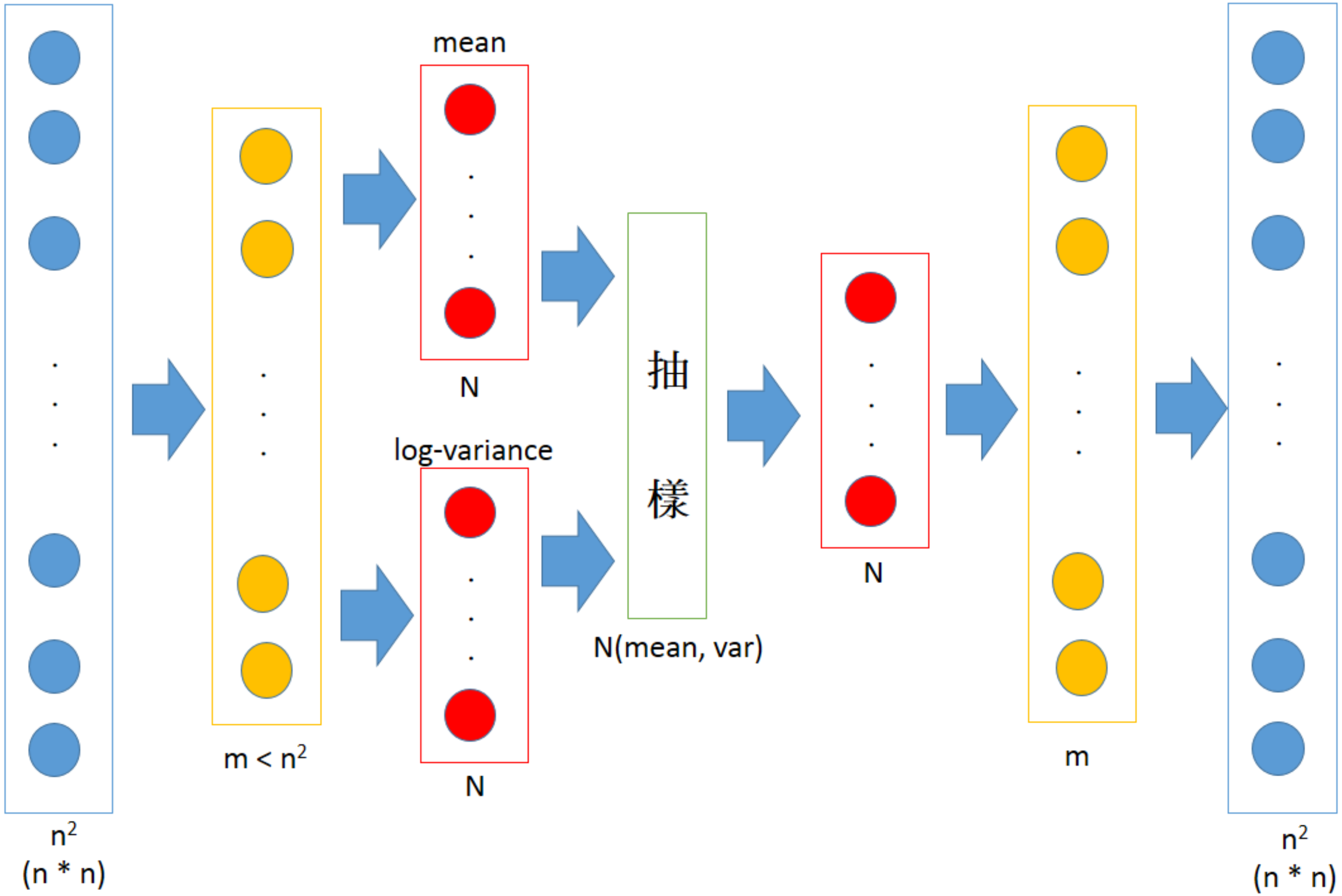


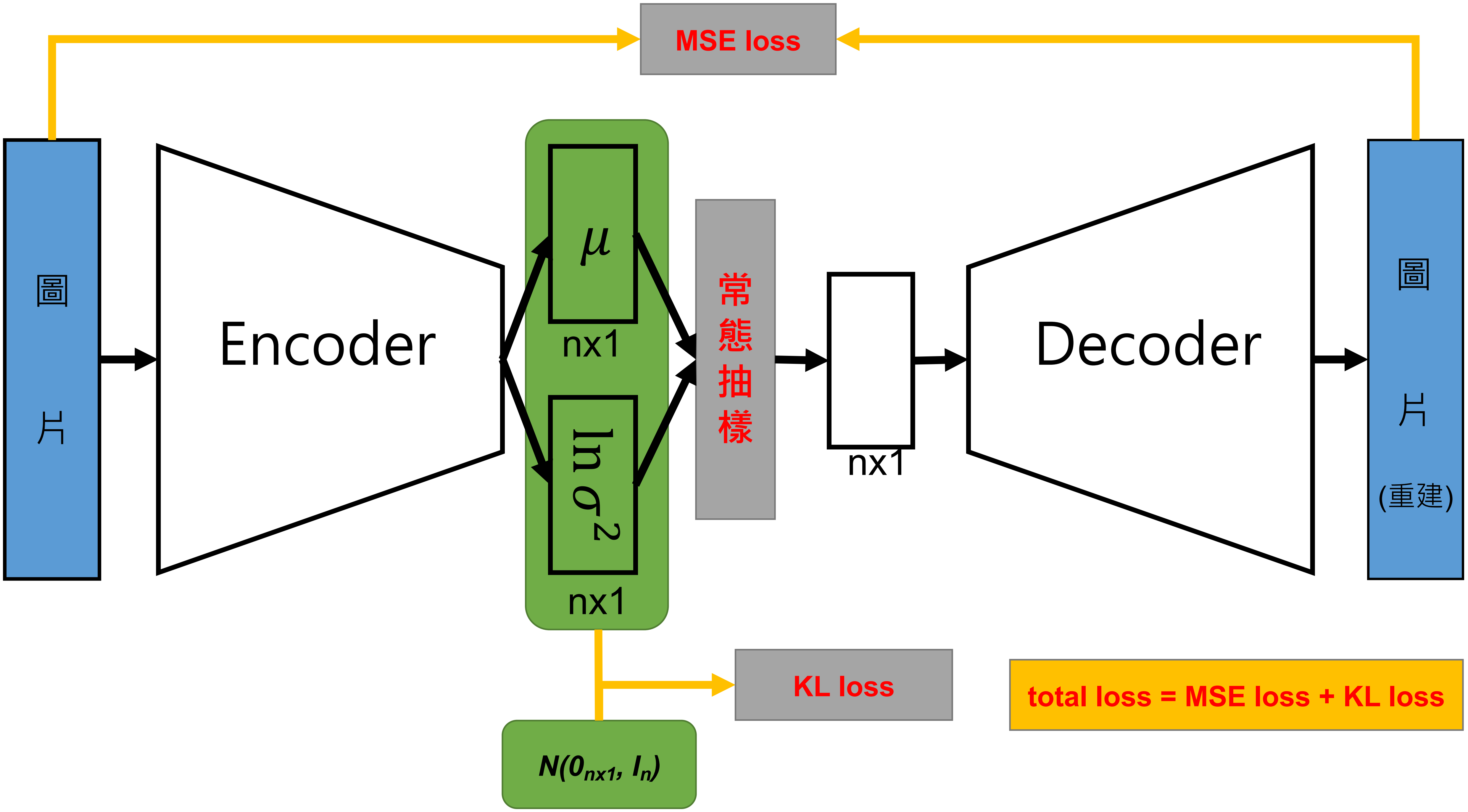
Assignment Revisit – VAE

- Variational Autoencoder (VAE) is a variant of AE which randomize latent vector from deterministic one. (Don't worry, I will explain what it means)

- VAE can be constructed using all tips we learned above: Model, Branch-and-Merge, Customized Layers, Customized Loss Function.

Assignment Revisit – VAE





MSE loss
圖片
Encoder
μ
nx1
$\ln \sigma^2$
nx1
常態抽樣
nx1
Decoder
圖片(重建)
KL loss
$N(0_{nx1}, I_n)$
total loss = MSE loss + KL loss

Assignment Revisit – VAE

- So, we need to define two things:
  - Sampling from Normal distribution where mean and covariance are given by previous two layers.
  - Loss function defined by sum of MSE between I/O and KL-divergence between $N(\mu, \sigma^2)$ and $N(0, I_n)$.

Assignment Revisit – VAE

- So, we need to define two things:
  - **Sampling from Normal distribution where mean and covariance are given by previous two layers.**
  - Loss function defined by sum of MSE between I/O and KL-divergence between N(μ, σ^2) and N(0,I_n).

- Recall in probability, if X~N(0,1), then μ+σX~N(μ,σ^2).
- For higher dimensional Normal distribution, we have similar result.

Assignment Revisit – VAE

- So, we need to define two things:
  - **Sampling from Normal distribution where mean and covariance are given by previous two layers.**
  - Loss function defined by sum of MSE between I/O and KL-divergence between $N(\mu, \sigma^2)$ and $N(0, I_n)$.

```
def sampling(args):
    z_mean, z_log_var = args
    epsilon = K.random_normal(shape=(batch_size, latent_dim), mean=0., stddev=epsilon_std)
    return z_mean + K.exp(z_log_var / 2) * epsilon
```

Assignment Revisit – VAE

- So, we need to define two things:
  - Sampling from Normal distribution where mean and covariance are given by previous two layers.
  - **Loss function defined by sum of MSE between I/O and KL-divergence between N(μ, σ^2) and N(0,I_n).**

```
def vae_loss(y_true, y_pred):
    mse_loss = K.square(y_true - y_pred)
    kl_loss = - 0.5 * K.sum(1 + z_log_var - K.square(z_mean) - K.exp(z_log_var), axis=-1)
    return K.mean(xent_loss + kl_loss)
```

Assignment Revisit – VAE

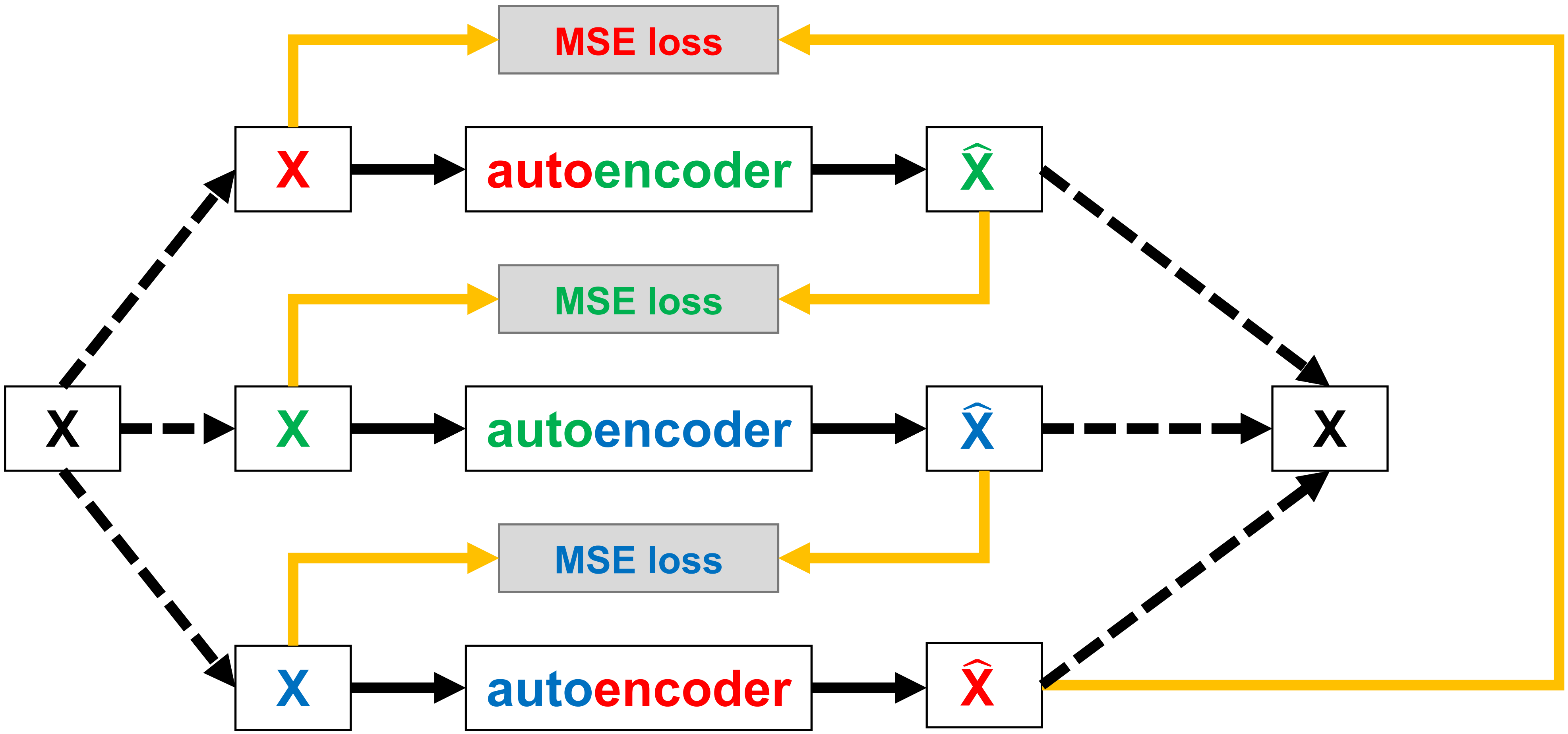
- Review the code of VAE!

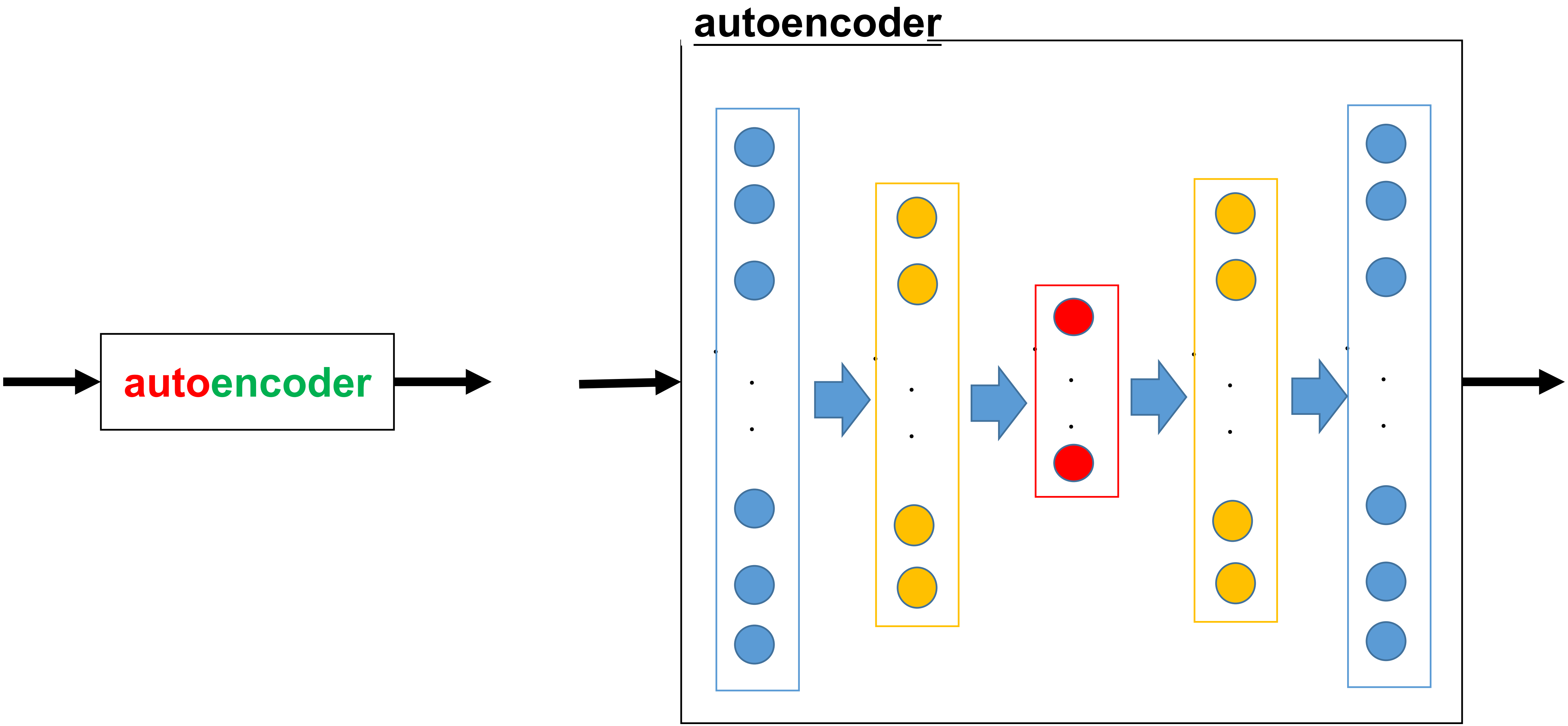
Final Challenge

- For the remaining time, I want you all to build up a model which is a combination of what we learned today.

- You might use Model, Branch-and-Merge, Customized Loss Function in this model.

Final Challenge - Structure





autoencoder
autoencoder

Congratulations!

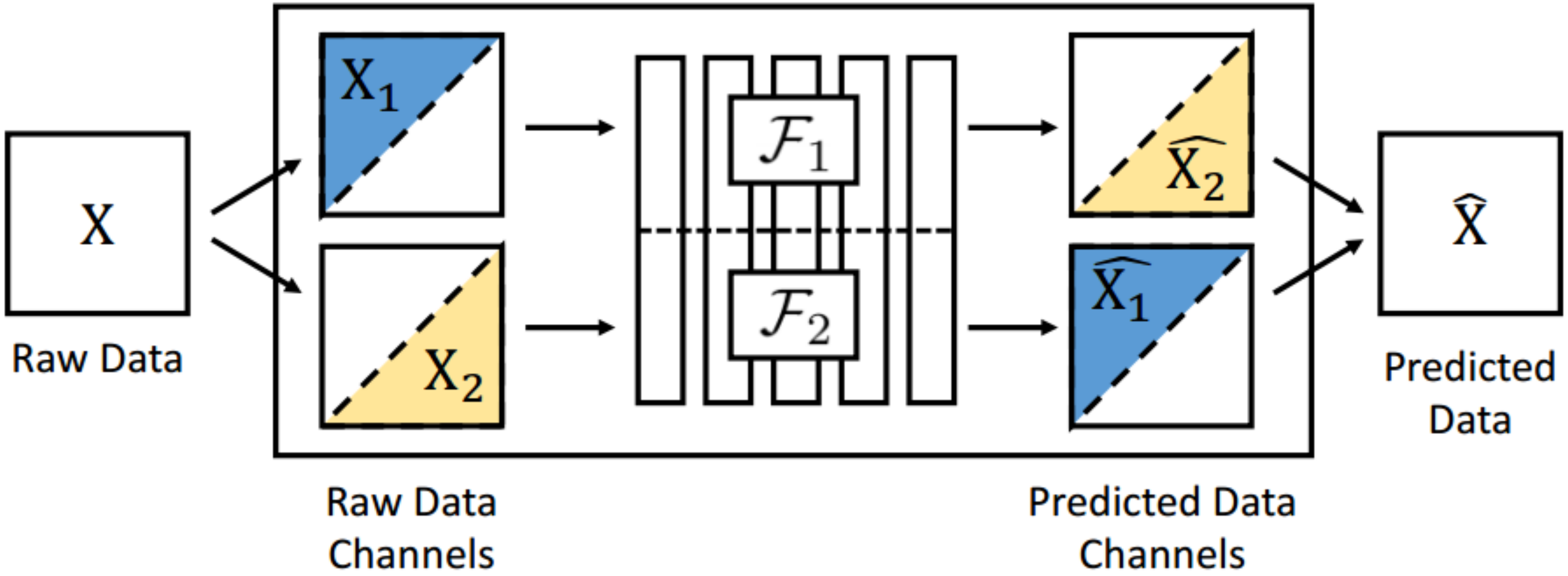
- If you can finish this final challenge, then you are one step closer to …

Congratulations!

- If you can finish this final challenge, then you are one step closer to …

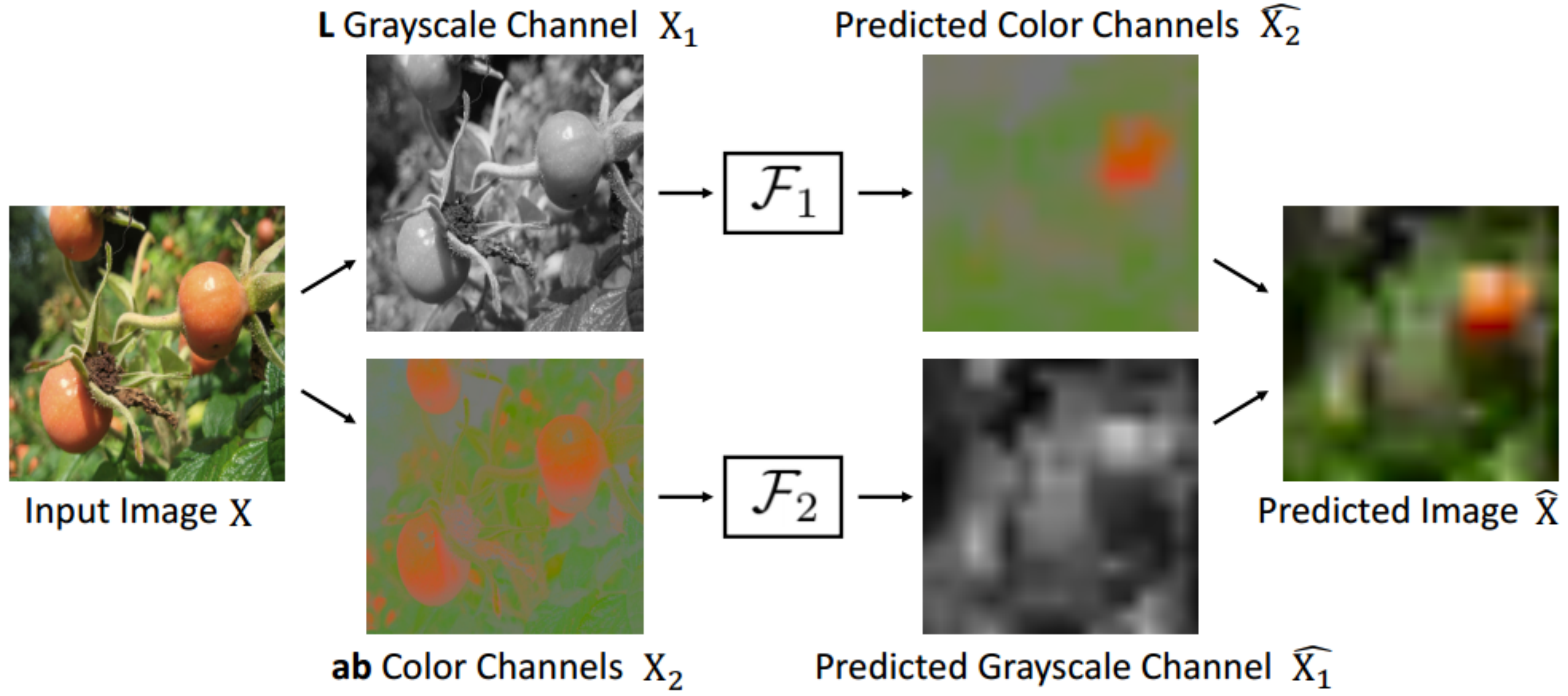
Berkeley AI Research (BAIR) Laboratory！

Real World Implementation



Ref: R. Zhang, P. Isola, and A. A. Efros. Split-brain autoencoders: Unsupervised learning by cross-channel prediction. CoRR, abs/1611.09842, 2016.

Real World Implementation



Acknowledgement

- Special thanks to Yen Jan who provides me some suggestion about this talk.

всё