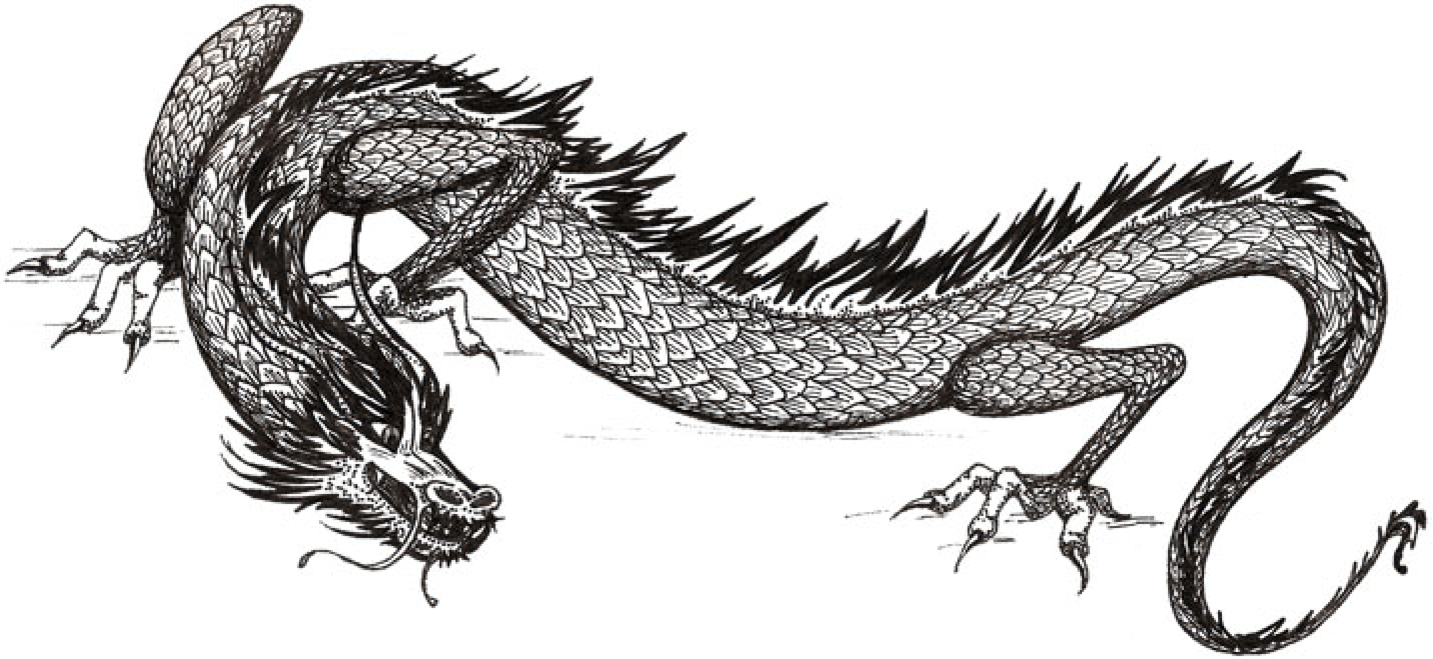
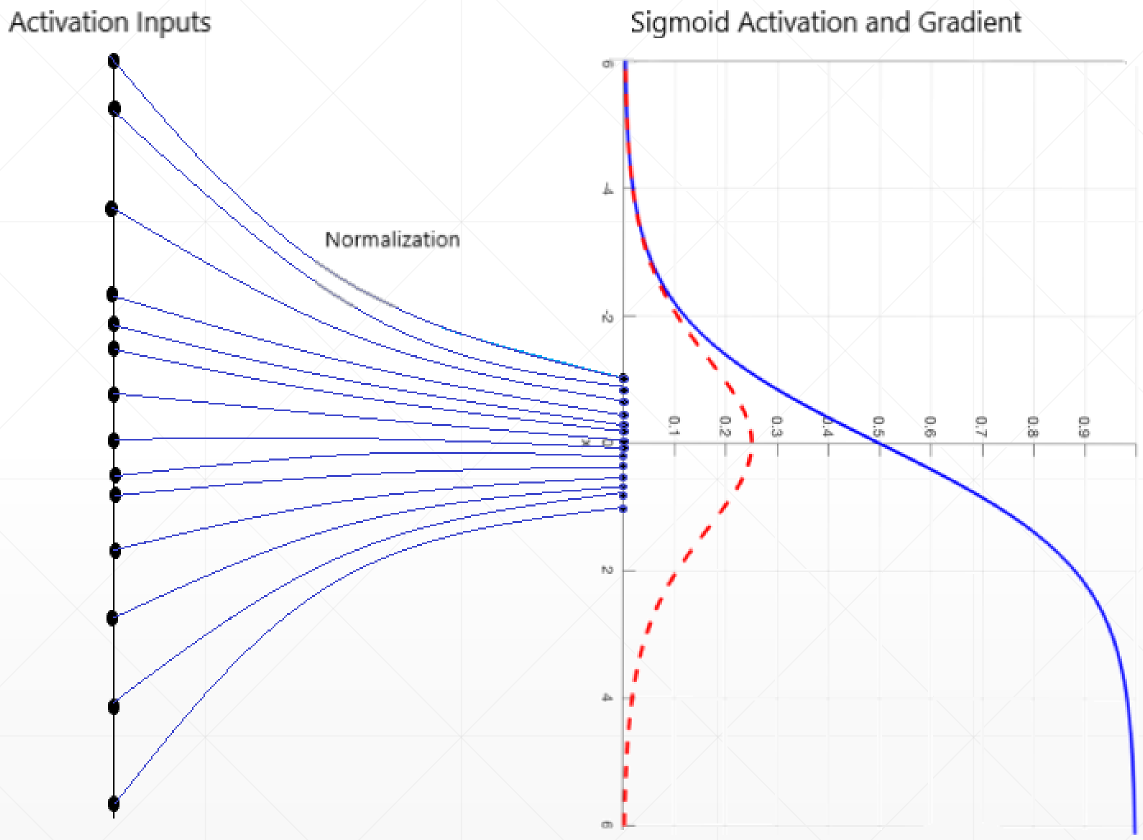


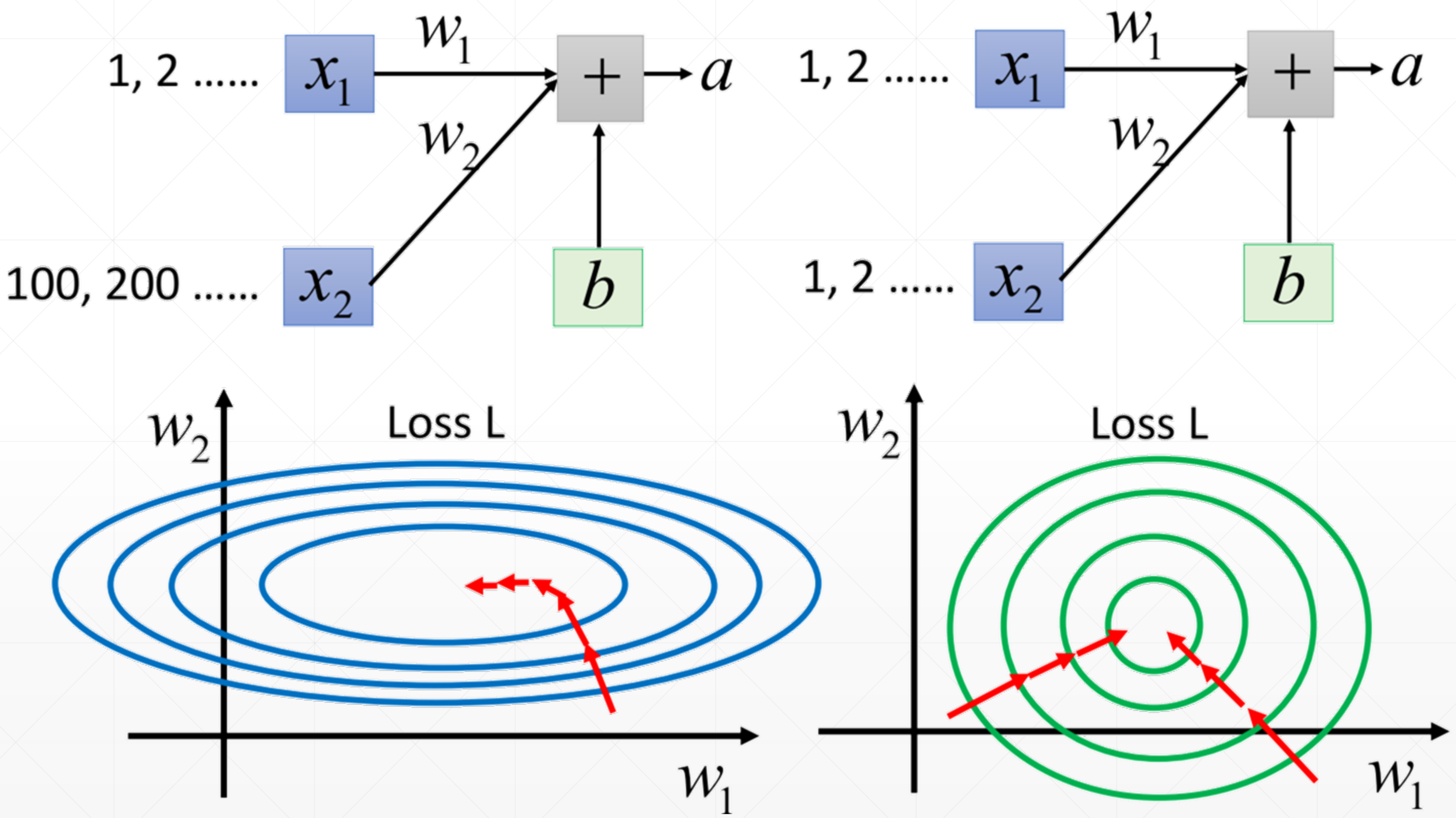
Batch Normalization

主讲：龙良曲

Gradient Vanishing



Intuitive explanation



Feature scaling

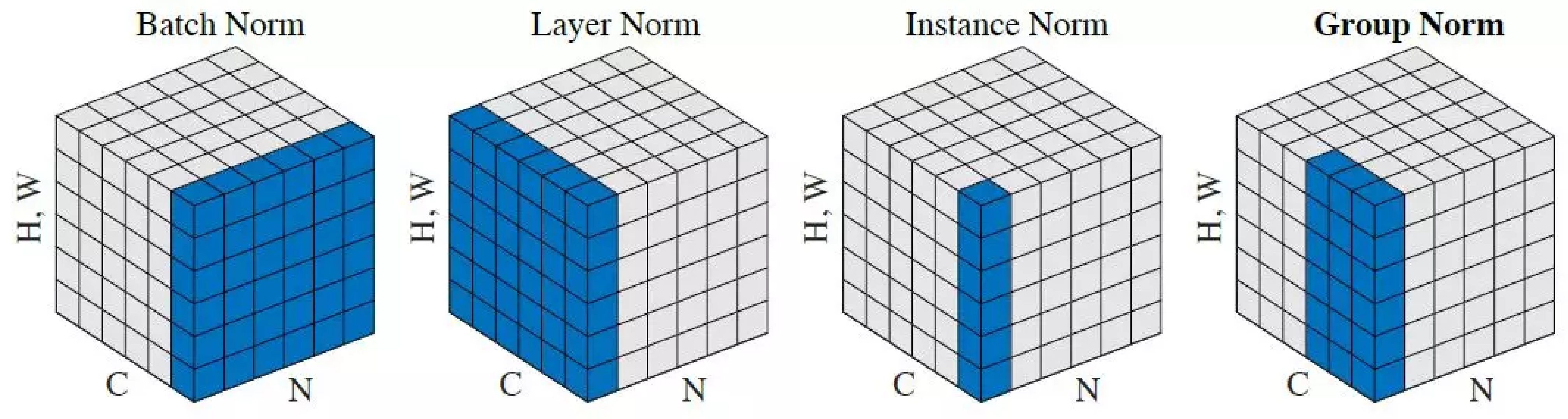
- Image Normalization

```
def normalize(x, mean,std):
  # x: [b, h, w, c]
  x = x - mean
  x = x / std
  return x
```

```
(mean=[0.485, 0.456, 0.406],
 std=[0.229, 0.224, 0.225])
```

- Batch Normalization
  - dynamic mean/std

Batch Norm

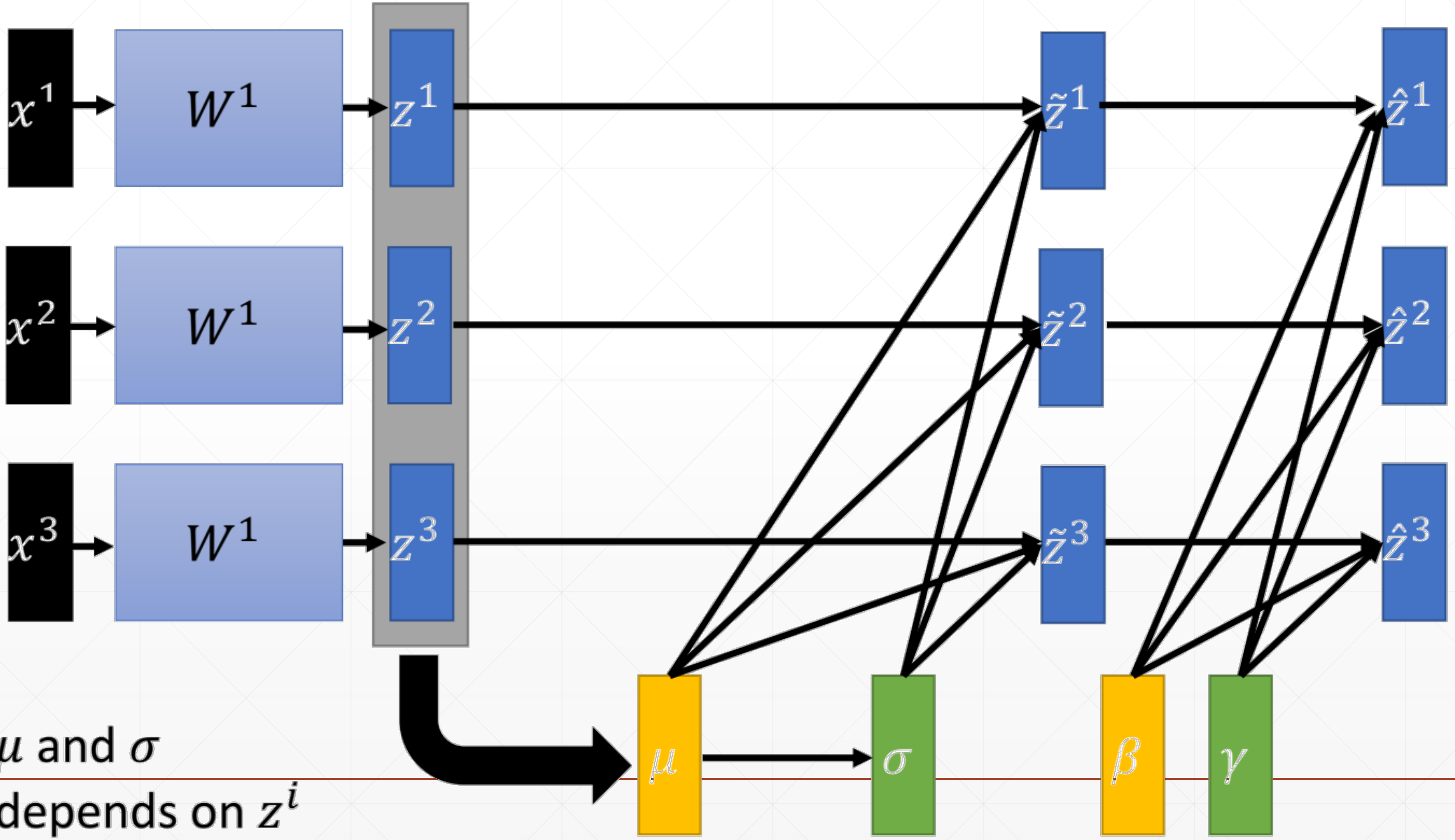


https://medium.com/syncedreview/facebook-ai-proposes-group-normalization-alternative-to-batch-normalization-fb0699bffae7

Batch normalization

$$\tilde{z}^i = \frac{z^i - \mu}{\sigma}$$

$$\hat{z}^i = \gamma \odot \tilde{z}^i + \beta$$



μ and σ depends on z^i

layers.BatchNormalization

- net = layers.BatchNormalization()
  - axis=-1,
  - center=True,
  - scale=True
  - trainable=True

- net(x, training=None)

```
In [3]: net=layers.BatchNormalization()
In [5]: x=tf.random.normal([2,3])
In [6]: out=net(x)

In [7]: net.trainable_variables
[<tf.Variable 'batch_normalization/gamma:0' shape=(3,) dtype=float32, numpy=array([1., 1., 1.],
dtype=float32)>,
 <tf.Variable 'batch_normalization/beta:0' shape=(3,) dtype=float32, numpy=array([0., 0., 0.],
dtype=float32)>]

In [8]: net.variables
[<tf.Variable 'batch_normalization/gamma:0' shape=(3,) dtype=float32, numpy=array([1., 1., 1.],
dtype=float32)>,
 <tf.Variable 'batch_normalization/beta:0' shape=(3,) dtype=float32, numpy=array([0., 0., 0.],
dtype=float32)>,
 <tf.Variable 'batch_normalization/moving_mean:0' shape=(3,) dtype=float32, numpy=array([0., 0.,
0.], dtype=float32)>,
 <tf.Variable 'batch_normalization/moving_variance:0' shape=(3,) dtype=float32, numpy=array([1.,
1., 1.], dtype=float32)>]
```

Pipeline

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

BatchNorm for Image

```
In [15]: x=tf.random.normal([2,4,4,3],mean=1.,stddev=0.5)
In [16]: net=layers.BatchNormalization(axis=3)
In [17]: out=net(x)

In [18]: net.variables
[<tf.Variable 'batch_normalization_2/gamma:0' shape=(3,) dtype=float32, numpy=array([1., 1.,
1.], dtype=float32)>,
 <tf.Variable 'batch_normalization_2/beta:0' shape=(3,) dtype=float32, numpy=array([0., 0., 0.],
dtype=float32)>,
 <tf.Variable 'batch_normalization_2/moving_mean:0' shape=(3,) dtype=float32, numpy=array([0.,
0., 0.], dtype=float32)>,
 <tf.Variable 'batch_normalization_2/moving_variance:0' shape=(3,) dtype=float32,
numpy=array([1., 1., 1.], dtype=float32)>]
```

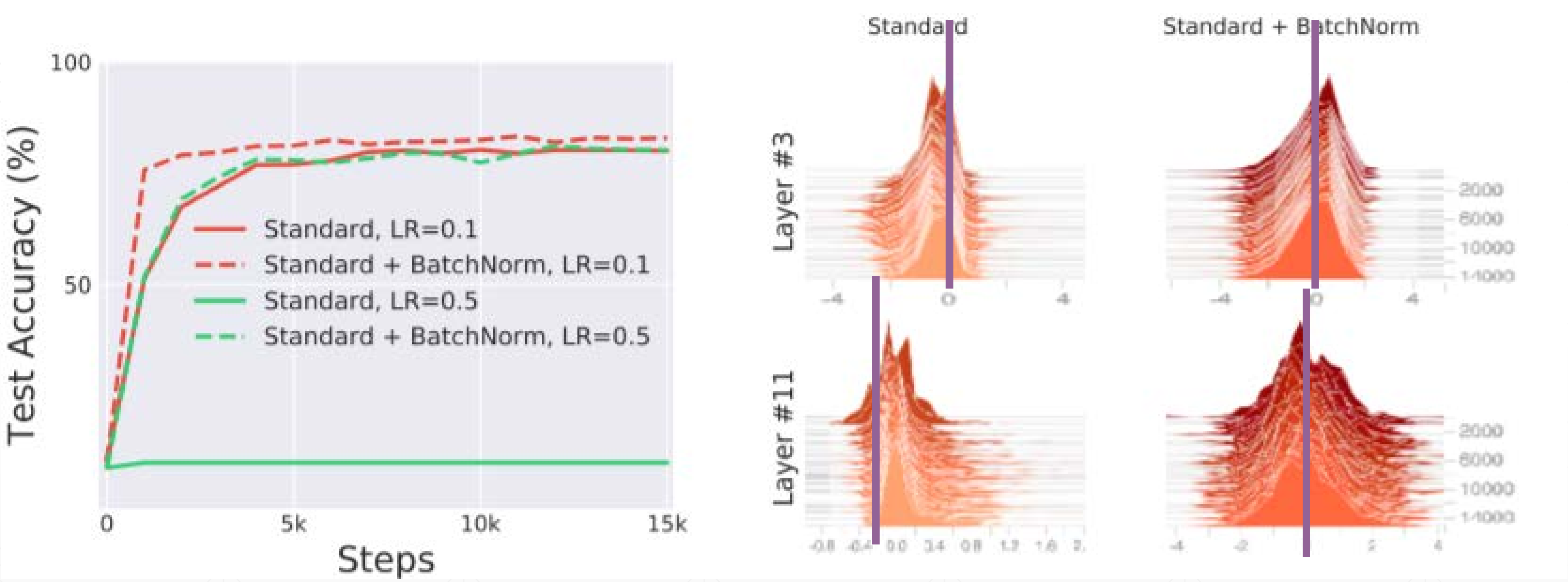
Forward update

```
In [15]: x=tf.random.normal([2,4,4,3],mean=1.,stddev=0.5)
In [16]: net=layers.BatchNormalization(axis=3)
In [19]: out=net(x,training=True)
In [20]: net.variables
[<tf.Variable 'batch_normalization_2/gamma:0' shape=(3,) dtype=float32, numpy=array([1., 1.,
1.], dtype=float32)>,
 <tf.Variable 'batch_normalization_2/beta:0' shape=(3,) dtype=float32, numpy=array([0., 0., 0.],
dtype=float32)>,
 <tf.Variable 'batch_normalization_2/moving_mean:0' shape=(3,) dtype=float32,
numpy=array([0.00992113, 0.00976678, 0.00942467], dtype=float32)>,
 <tf.Variable 'batch_normalization_2/moving_variance:0' shape=(3,) dtype=float32,
numpy=array([0.9923687 , 0.9918039 , 0.99186534], dtype=float32)>]

In [21]: for i in range(100):out=net(x,training=True)
In [22]: net.variables
[<tf.Variable 'batch_normalization_2/gamma:0' shape=(3,) dtype=float32, numpy=array([1., 1.,
1.], dtype=float32)>,
 <tf.Variable 'batch_normalization_2/beta:0' shape=(3,) dtype=float32, numpy=array([0., 0., 0.],
dtype=float32)>,
 <tf.Variable 'batch_normalization_2/moving_mean:0' shape=(3,) dtype=float32,
numpy=array([0.63259894, 0.6227575 , 0.60094345], dtype=float32)>,
 <tf.Variable 'batch_normalization_2/moving_variance:0' shape=(3,) dtype=float32,
numpy=array([0.5134074, 0.4773918, 0.4813124], dtype=float32)>]
```

Backward update

```
for i in range(10):
    with tf.GradientTape() as tape:
        out = net(x, training=True)
        loss = tf.reduce_mean(tf.pow(out,2)) - 1
    grads = tape.gradient(loss, net.trainable_variables)
    optimizer.apply_gradients(zip(grads, net.trainable_variables))

backward(10 steps):
  [<tf.Variable 'batch_normalization/gamma:0' shape=(3,) dtype=float32, numpy=array([0.93549937,
0.9356556 , 0.9355564 ], dtype=float32)>,
   <tf.Variable 'batch_normalization/beta:0' shape=(3,) dtype=float32, numpy=array([ 1.3411044e-
09,  1.3411045e-08, -4.0978188e-10], dtype=float32)>...]
```

Visualization



Advantages

- Converge faster
- slightly Better performance
- Robust
  - stable
  - larger learning rate

下一课时

深度残差网络

Thank You.