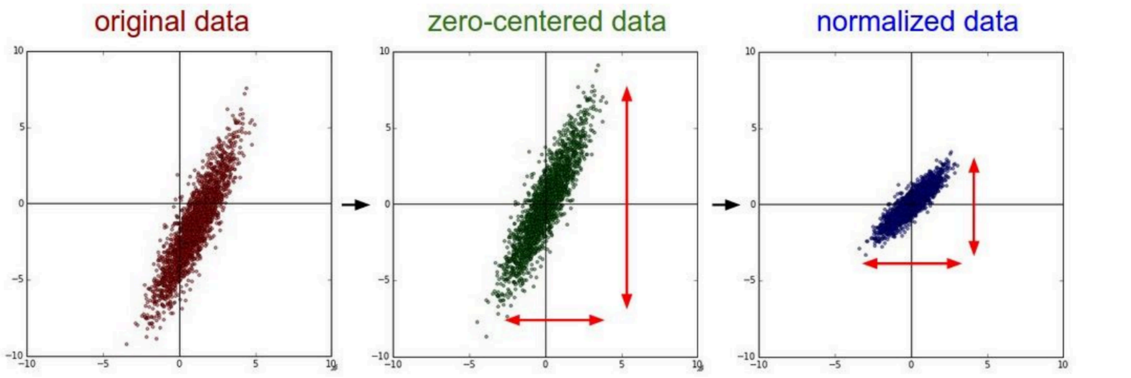
Processing data

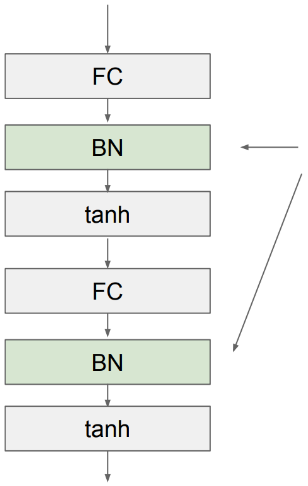



```
X -= np.mean(X, axis = 0)
```

```
X /= np.std(X, axis = 0)
```

In practice for Images: **center only**
In practice, the current recommendation is to use ReLU units and use the
`w = np.random.randn(n) * sqrt(2.0/n)`

Batch normalization



Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\boxed{\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}}$$

为什么要用Batch Normalization?

- Improves gradient flow through the network
- Allows higher learning rates

- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout

---

Regularization

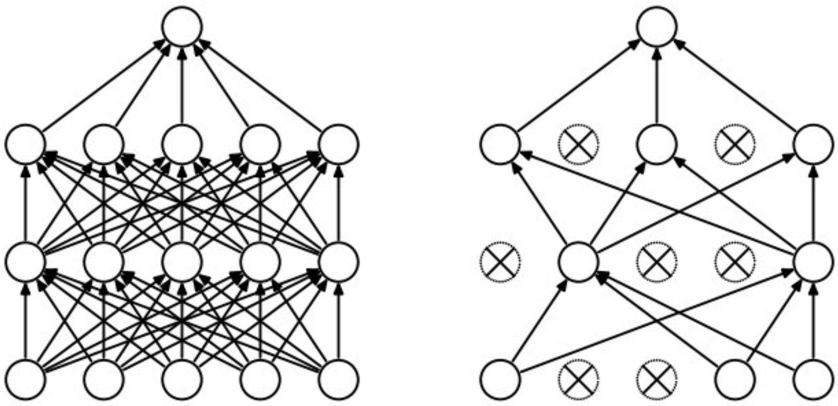
为什么要用Regularization?

- 避免overfitting
    - 回忆 w0 = [1,0,0,0] 与 w1 = [0.25,0.25,0.25,0.25]的例子，w0在这个训练集上表现很好，但是过拟合了，换一个训练集就不行啦

Regularization的主要方法
L2 regularization
L1 regularization
Dropout

Dropout

随机扔掉一些neuron



(a) Standard Neural Net (b) After applying dropout.

```
"""
Inverted Dropout: Recommended implementation example.
We drop and scale at train time and don't do anything at test time.
"""

p = 0.5 # probability of keeping a unit active. higher = less dropout

def train_step(X):
  # forward pass for example 3-layer neural network
  H1 = np.maximum(0, np.dot(W1, X) + b1)
  U1 = (np.random.rand(*H1.shape) < p) / p # first dropout mask. Notice /p!
  H1 *= U1 # drop!
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  U2 = (np.random.rand(*H2.shape) < p) / p # second dropout mask. Notice /p!
  H2 *= U2 # drop!
  out = np.dot(W3, H2) + b3

  # backward pass: compute gradients... (not shown)
  # perform parameter update... (not shown)

def predict(X):
  # ensembled forward pass
  H1 = np.maximum(0, np.dot(W1, X) + b1) # no scaling necessary
  H2 = np.maximum(0, np.dot(W2, H1) + b2)
  out = np.dot(W3, H2) + b3
```

p = 0.5 意味着概率上会扔掉一半