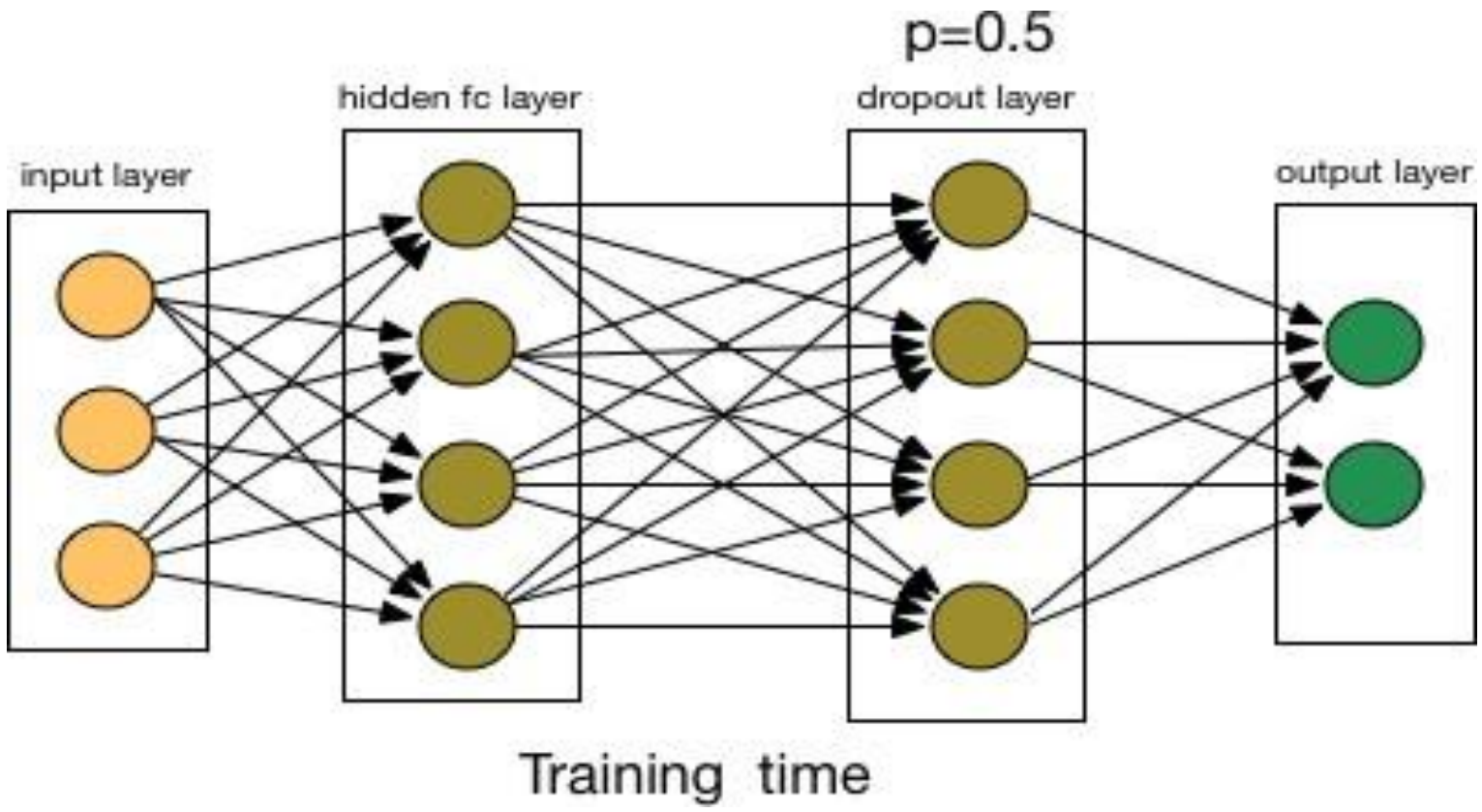
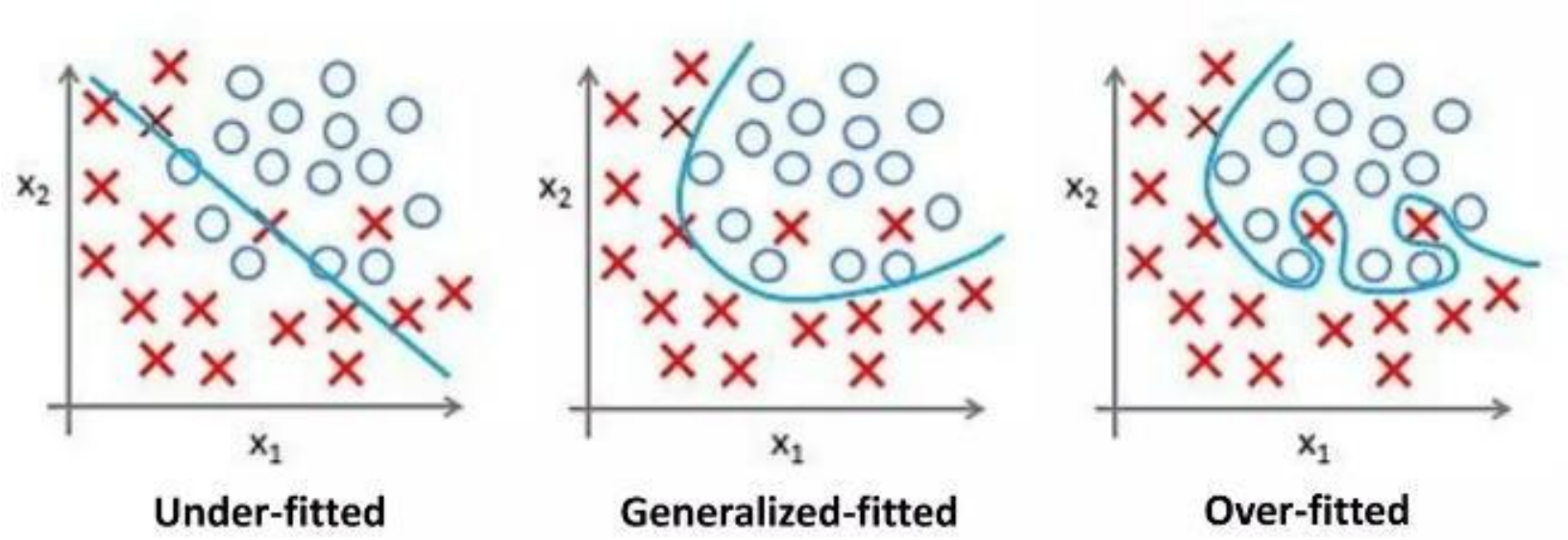
Dropouts and Batch Normalization in CNN



Recap – overfitting and underfitting



- Due to a large number of parameters, they can learn extremely complex functions. But this also makes them very prone to overfitting the training data.
- To prevent the overfitting problem, many regularization techniques have been introduced.
  - Ex. Dropout, L1/L2 regularization, and Max-Norm Constraints.

Regularization with Unlimited Computation

Best way to regularize a fixed-size model is

- Average the **predictions of all possible settings** of the parameters.
- Weighting each setting with the posterior probability given the training data.

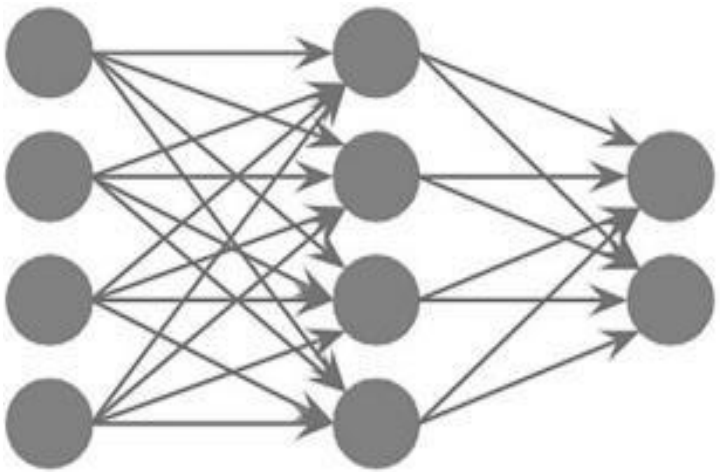
  -This would be the **Bayesian approach**

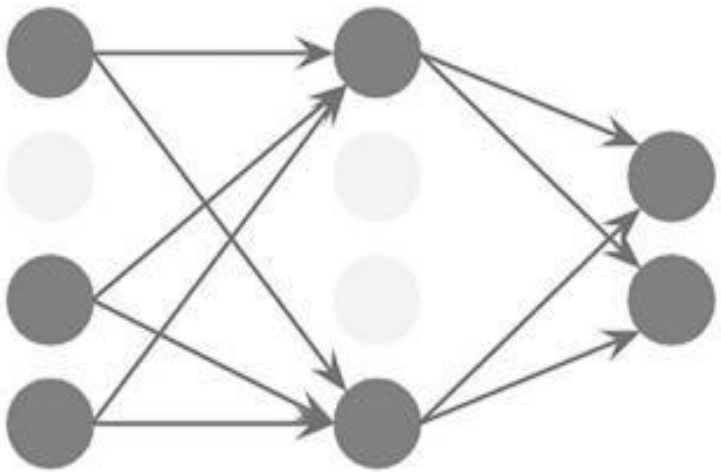
Dropout does this using considerably **less computation**

- By approximating an equally weighted geometric mean of the predictions of an exponential number of learned models that share parameters.

Dropout

- Introduced by Nitish Srivastava, Geoffrey Hinton Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov in 2014.
- It prevents overfitting and provides a way of approximately combining exponentially many different neural network architectures efficiently.
- The term "dropout" refers to dropping out units (hidden and visible) in a neural network. By dropping a unit out, that means temporarily removing it from the network, along with all its **incoming and outgoing connections.**





Dropout

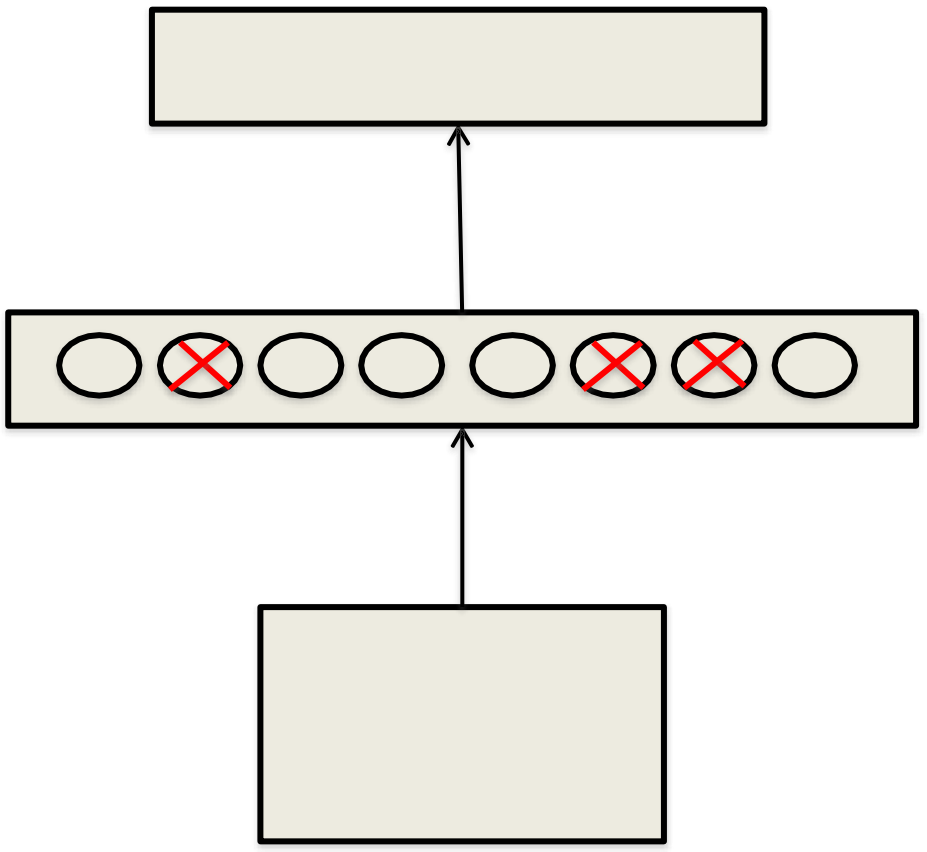
Dropout is a bagging method.

- Bagging is a method of **averaging over several models** to improve generalization.
- Impractical to train many neural networks since it is expensive in time and memory.
  - Dropout makes it practical to apply bagging to very many large neural networks.
    - It is a method of bagging applied to neural networks

Dropout is an inexpensive but powerful method of regularizing a broad family of models

Dropout: An efficient way to average many large neural nets

- Consider a neural net with one hidden layer.
- Each time we present a training example, we **randomly omit each hidden unit** with probability 0.5.
- So we are randomly sampling from 2^H different architectures.
  - All architectures share weights.



Dropout as a form of **model averaging**

- We sample from 2^H models. So only a few of the models ever get trained, and they only get one training example.
  - This is as extreme as bagging can get.
- The sharing of the weights means that every model is very strongly regularized.
  - It's a much better regularizer than L2 or L1 penalties that pull the weights towards zero.

But what do we do at test time?

- We could sample many different architectures and take the geometric mean of their output distributions.
- It is better to use all of the hidden units, but to halve their outgoing weights.
  - This exactly computes the geometric mean of the predictions of all 2^H models.

What if we have more hidden layers?

- Use dropout of 0.5 in every layer.
- At test time, use the "mean net" that has all the outgoing weights halved.
  - This is not exactly the same as averaging all the separate dropped out models, but it's a pretty good approximation, and it's fast.
- Alternatively, run the stochastic model several times on the same input.
  - This gives us an idea of the uncertainty in the answer.

What about the input layer?

- It helps to use dropout there too, but with a **higher probability of keeping an input unit.**
  - This trick is already used by the "denoising autoencoders" developed by Pascal Vincent, Hugo Larochelle and Yoshua Bengio.

Dropout

- Example:
  - Suppose a given input x: {1, 2, 3, 4, 5} to the fully connected layer.
  - We have a dropout layer with probability p = 0.2 (or keep probability = 0.8).
  - During the **forward propagation** (training) from the input x, 20% of the nodes would be dropped.
  - ie., the x could become {1, 0, 3, 4, 5} or {1, 2, 0, 4, 5} and so on. Similarly, it applied to the hidden layers.

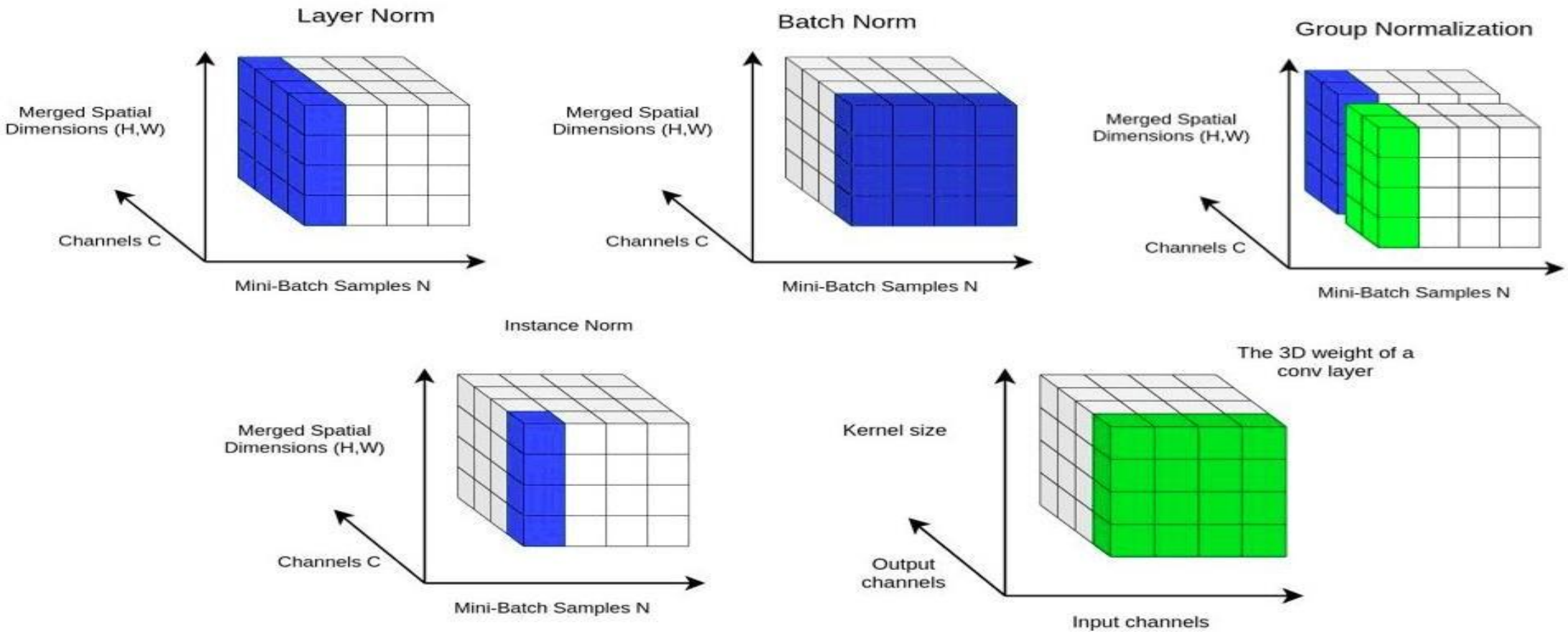
- For instance, if the hidden layers have 1000 neurons (nodes) and a dropout is applied with drop probability = 0.5, then 500 neurons would be randomly dropped in every iteration (batch).

- Standard practice is to set the retention probability to 0.5 for hidden layers and to something close to 1, like 0.8 or 0.9 on the input layer. Output layers generally do not apply dropout.

How well does dropout work?

- The record-breaking object recognition net developed by Alex Krizhevsky uses dropout and it helps a lot.
- If your deep neural net is significantly overfitting, dropout will usually reduce the number of errors by a lot.
  - Any net that uses "early stopping" can do better by using dropout (at the cost of taking quite a lot longer to train).
- If your deep neural net is not overfitting you should be using a bigger one!

Normalization



Normalization

- Normalizing a set of data transforms the set of data to be on a similar scale.
- In Deep Learning, Normalization is a pre-processing technique used to standardize data.
- In other words, having different sources of data inside the same range.
- Not normalizing the data before training can cause problems in our network, making it drastically harder to train and decreasing its learning speed.

Types of Normalization

- There are two main methods to normalize our data.

The most straightforward method is to scale it to a range from 0 to 1:

$$x_{normalized} = \frac{x - m}{x_{max} - x_{min}}$$

- x the data point to normalize, m the mean of the data set, x_{max} the highest value, and x_{min} the lowest value.
- This technique is generally used in the inputs of the data.
- The non-normalized data points with wide ranges can cause instability in Neural Networks. The relatively large inputs can cascade down to the layers, causing problems such as exploding gradients.

Types of Normalization

The other technique used to normalize data is forcing the data points to have a mean of 0 and a standard deviation of 1, using the following formula:

$$x_{normalized} = \frac{x - m}{s}$$

- *being x the data point to normalize, m the mean of the data set, and s the standard deviation of the data set. Now, each data point mimics a standard normal distribution. Having all the features on this scale, none of them will have a bias, and therefore, our models will learn better.*

Batch Normalization

- Batch Norm is a normalization technique done between the layers of a Neural Network instead of in the raw data.
- It is done along mini-batches instead of the full data set. It serves to speed up training and use higher learning rates, making learning easier.

$$z^N = \left(\frac{z - m_z}{s_z} \right)$$

- m_z the mean of the neurons' output and s_z the standard deviation of the neurons' output.

1. **Linear Transformation (Dense or Conv layer):**

$$z = Wx + b$$

2. **Batch Normalization (BN):**

$$\hat{z} = \frac{z - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$y = \gamma \hat{z} + \beta$$

3. **Activation Function (ReLU, Sigmoid, etc.):**

$$a = f(y)$$

Step 1: Compute the Mean & Variance of the Mini-Batch

For a given feature x in a batch, BN calculates:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \quad \text{(mean of batch)}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \quad \text{(variance of batch)}$$

Step 2: Normalize the Batch (Standardization)

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

- This makes the batch have a **mean of 0** and **standard deviation of 1**.
- ϵ (a small value) prevents division by zero.

Step 3: Scale and Shift Using Learnable Parameters

Instead of keeping the values strictly between $[-1, 1]$, BN allows the network to learn new parameters:

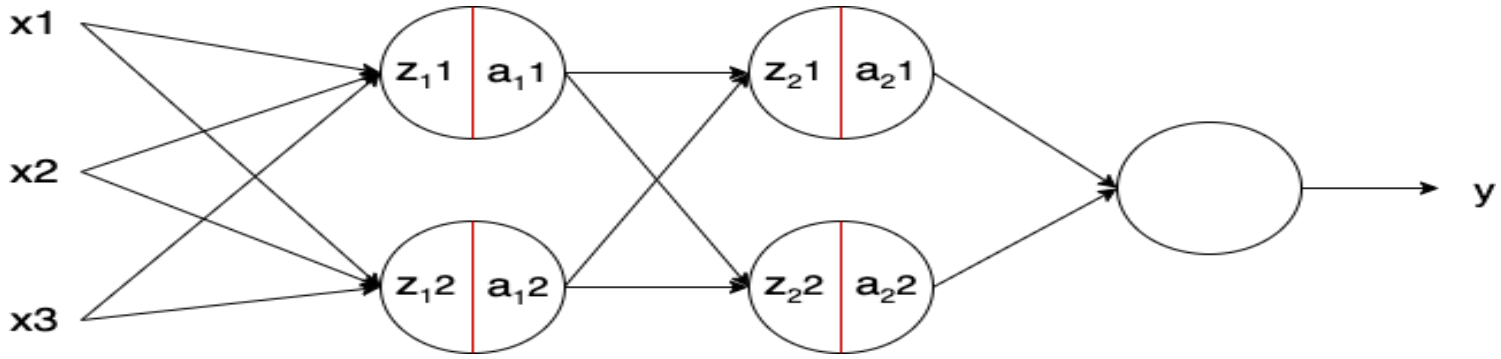
$$y_i = \gamma \hat{x}_i + \beta$$

where:

- γ (scale) and β (shift) are **learnable parameters** that allow the network to restore any useful distribution.
- If $\gamma = \sigma_B$ and $\beta = \mu_B$, BN can learn to **undo the normalization** if needed.

Batch Normalization -How Is It Applied

- Consider a regular feed-forward Neural Network - x_i are the inputs, z the output of the neurons, a the output of the activation functions, and y the output of the network:



- Batch Norm – in the image represented with a red line – is applied to the neurons' output just before applying the activation function.
- Usually, a neuron without Batch Norm would be computed as : $z = g(w, x) + b; \qquad a = f(z)$
- being g() the linear transformation of the neuron, w the weights of the neuron, b the bias of the neurons, and f() the activation function.

Batch Normalization -How Is It Applied

- The model learns the parameters *w* and *b*. Adding Batch Norm, it looks as:

$$z = g(w, x); \qquad z^N = \left(\frac{z - m_z}{s_z}\right) \cdot \gamma + \beta; \qquad a = f(z^N)$$

- m_z the mean of the neurons' output, s_z the standard deviation of the output of the neurons, and γ and β learning parameters of Batch Norm.
- Note that the bias of the neurons (b) is removed. This is because as we subtract the mean m_z, any constant over the values of *z* – such as *b* – can be ignored as it will be subtracted by itself.
- The parameters γ and β shift the mean and standard deviation. These values are learned over epochs and other learning parameters, such as the weights of the neurons, aiming to decrease the loss of the model.

Batch Normalization Advantages

- Batch Normalization accelerates the training of deep neural networks.

- For every input mini-batch we calculate different statistics. This introduces some sort of regularization. Regularization refers to any form of technique/constraint that restricts the complexity of a deep neural network during training.

- Every mini-batch has a different mini-distribution. We call the change between these mini-distributions Internal Covariate Shift.

- BN also has a beneficial effect on the gradient flow through the network. It reduces the dependence of gradients on the scale of the parameters or of their initial values. This allows us to use much higher learning rates.

Batch Normalization Disadvantages

- Inaccurate estimation of batch statistics with small batch size, which increases the model error.
- In tasks such as video prediction, segmentation and 3D medical image processing the batch size is usually too small. BN needs a sufficiently large batch size.
- Problems when batch size is varying. Example showcases are training VS inference, pretraining VS fine-tuning, and backbone architecture VS head.

Different Recent Batch Normalization

- Weight normalization (2016)
- Layer normalization (2016)
- Instance Normalization: The Missing Ingredient for Fast Stylization (2016)
- Adaptive Instance Normalization (2017)
- Group normalization (2018)
- Synchronized Batch Normalization (2018)
- Weight Standardization (2019)
- SPADE (2019)