Improving Neural Networks

- In our linear regression example, we trained the model by minimizing the training error,

$$\frac{1}{m^{(train)}} || X^{(train)} w - y^{(train)} ||_2^2$$

- but we actually care about the test error,

$$\frac{1}{m^{(test)}} || X^{(test)} w - y^{(test)} ||_2^2$$

Improving Neural Networks

Optimization

$$\frac{1}{m^{(train)}} || X^{(train)} w - y^{(train)} ||_2^2$$

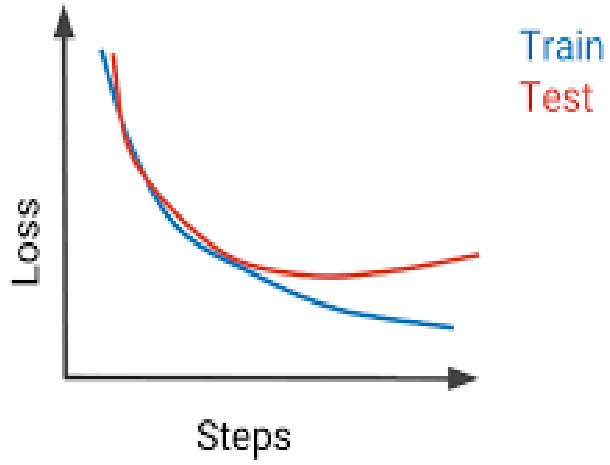
Generalization

$$\frac{1}{m^{(test)}} || X^{(test)} w - y^{(test)} ||_2^2$$

Improving Neural Networks

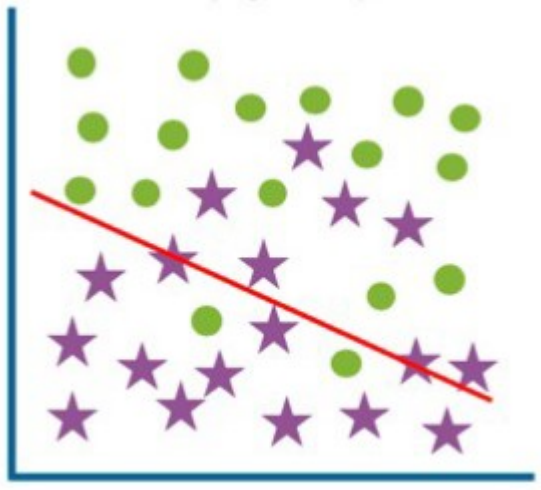
- At the beginning of training, optimization and generalization are correlated
- The factors determining how well a machine learning algorithm will perform are its ability to:

1. Make the training error small.
2. Make the gap between training and test error small.

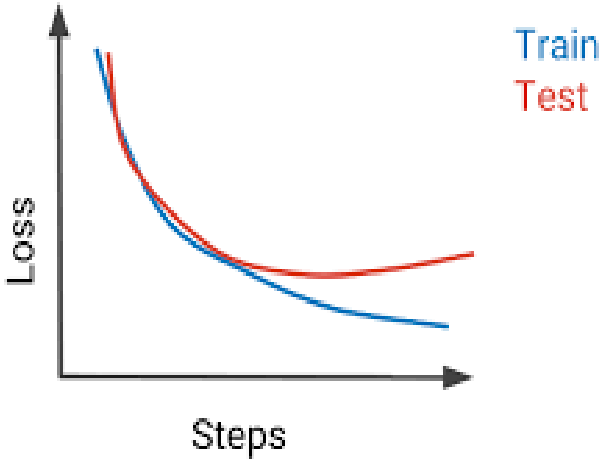


Improving Neural Networks

- The high loss on training and test data then your model is said to be **underfit**

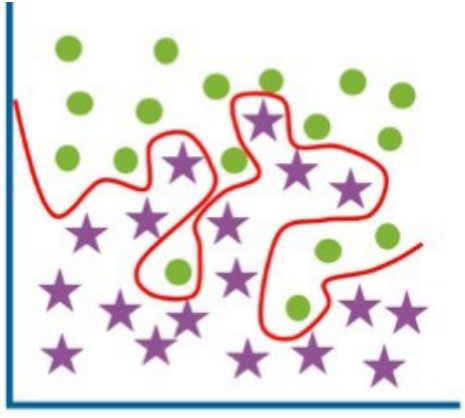


underfit

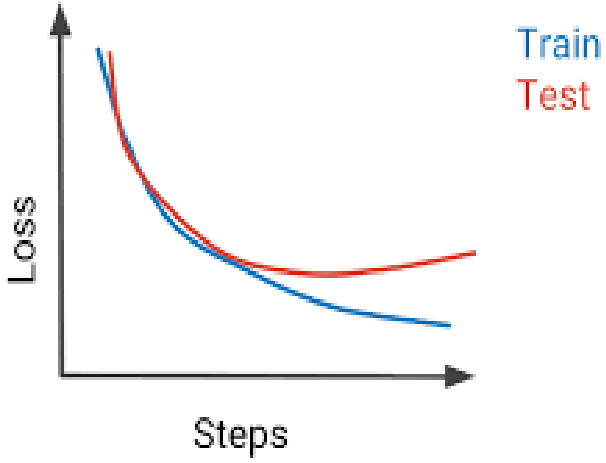


Improving Neural Networks

- But after a certain number of iterations on the training data, generalization stops improving, and begin to degrade
- The low loss on training data and high loss on test data then your model is said to be **Overfit.**



overfit



Improving Neural Networks

data

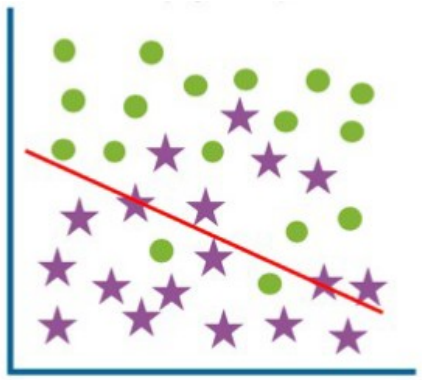
- 1, 2, 3 => satisfies rule
- 4, 5, 6 => satisfies rule
- 7, 8, 9 => satisfies rule
- 9,2,35 => does not satisfy rule

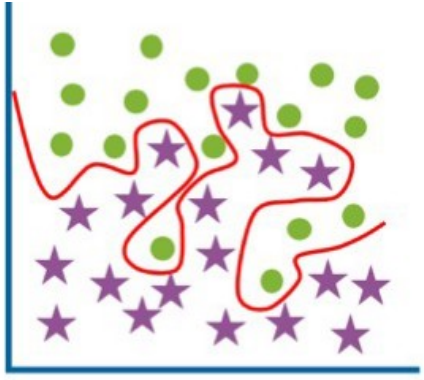
rule

- 3 consecutive single digits
- 3 consecutive integers
- 3 numbers in ascending order
- 3 numers whose sum is less than 25
- 3 numbers < 10
- 1, 4 or 7 as first number

Changing the model's capacity

- Models with low capacity may struggle to fit the training set.

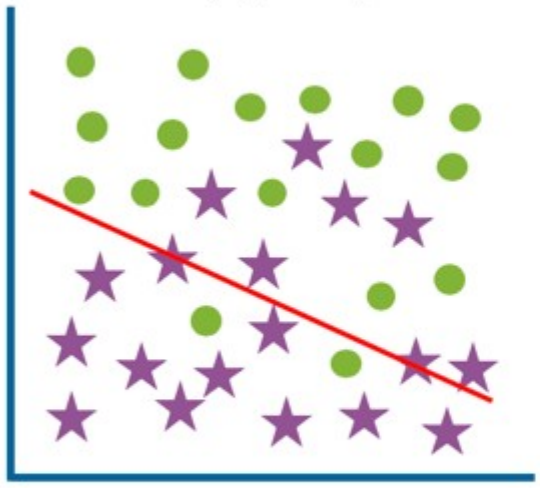




- Models with high capacity can overfit by memorizing properties of the training set that do not serve them well on the test set.

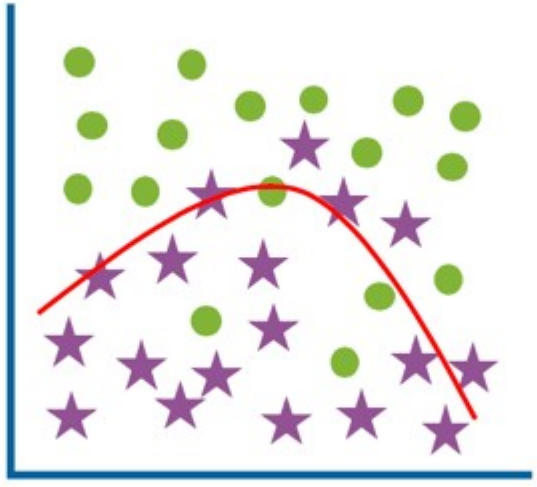
Changing the model's capacity

Underfit
(high bias)



High training error
High test error

Optimum



Low training error
Low test error

Overfit
(high variance)

Low training error
High test error

Fighting overfitting

1. **More training data.**
2. **Reduce the size of our network**

- However, large networks have the potential to be more powerful than small networks, and so this is an option we'd only adopt reluctantly.

Fighting overfitting-Regularization

- **Regularization techniques**: Techniques which can reduce overfitting, even when we have a fixed network and fixed training data.
- regularization place **constraints on the quantity and type of information** your model can store.
- If a network can only afford to memorize a small number of patterns, the optimization process will force it to focus on the most prominent patterns
- which have a better chance of generalizing well.

Regularization Techniques

- Weight regularization
  - L2 Regularization
  - L1 Regularization
- Dropout
- Data-augmentation
- Early stopping
- Batch Normalization

L2-Regularization

- The idea of L2 regularization is to add an extra term to the cost function, a term called the regularization term.

$$C = \frac{1}{2n} \sum_x \| y - \hat{y} \|^2 + \frac{\lambda}{2n} \sum_w w^2$$

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2$$

- where C_0 is the original, unregularized cost function.

L2-Regularization

- Regularization is to make it so the network prefers to **learn small weights**, all other things being equal.
- 1. finding small weights
- 2. minimizing the original cost function.

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2$$

- The relative importance of the two elements of the compromise depends on the value of λ:
- when λ is small we prefer to minimize the original cost function,
- but when λ is large we prefer small weights.

L2-Regularization

- Effect of regularization

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2$$

$$\frac{\partial C}{\partial w} = \frac{\partial C_0}{\partial w} + \frac{\lambda}{n} w$$

$$\frac{\partial C}{\partial b} = \frac{\partial C_0}{\partial b}$$

- Find $\partial C_0 / \partial w$ with backprop then add $(\lambda/n)w$ to the partial derivative of all the weight terms.
- The partial derivatives with respect to the biases are unchanged

L2-Regularization

- The learning rule for the weights becomes

$$w' \rightarrow w - \alpha \frac{\partial C_0}{\partial w} - \frac{\alpha \lambda}{n} w$$

$$w' = \left(1 - \frac{\alpha \lambda}{n}\right) w - \alpha \frac{\partial C_0}{\partial w}$$

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2$$

- This is exactly the same as the usual gradient descent learning rule, except we first rescale the weight w by a factor $1-\alpha\lambda/n$.
- This rescaling is sometimes referred to as weight decay, since it makes the weights smaller.

L1-Regularization

- L1 regularization: In this approach we modify the unregularized cost function by adding the sum of the absolute values of the weights:

  $C=C_0+(\lambda/n)\sum|w|$.

$$\frac{\partial C}{\partial w}=\frac{\partial C_0}{\partial w}+\frac{\lambda}{n}sgn(w)$$

$$w' \rightarrow w-\alpha\frac{\partial C}{\partial w}$$

$$w' \rightarrow w-\alpha\frac{\partial C_0}{\partial w}-\frac{\alpha\lambda}{n}$$

$$w' \rightarrow w-\alpha\frac{\partial C_0}{\partial w}+\frac{\alpha\lambda}{n}$$

L1-Regularization

- In L1 regularization, the weights shrink by a constant amount toward 0.

$$w \rightarrow w' - \alpha \frac{\partial C_0}{\partial w} - \frac{\alpha \lambda}{n}$$

$$w \rightarrow w' - \alpha \frac{\partial C_0}{\partial w} + \frac{\alpha \lambda}{n}$$

- In L2 regularization, the weights shrink by an amount which is proportional to w.

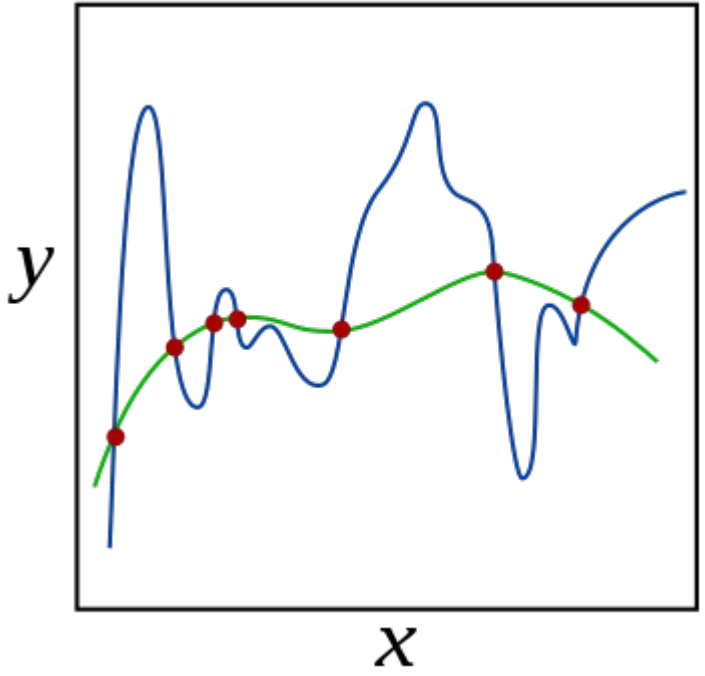
$$w \rightarrow w' = \left(1 - \frac{\alpha \lambda}{n}\right) w - \alpha \frac{\partial C_0}{\partial w}$$

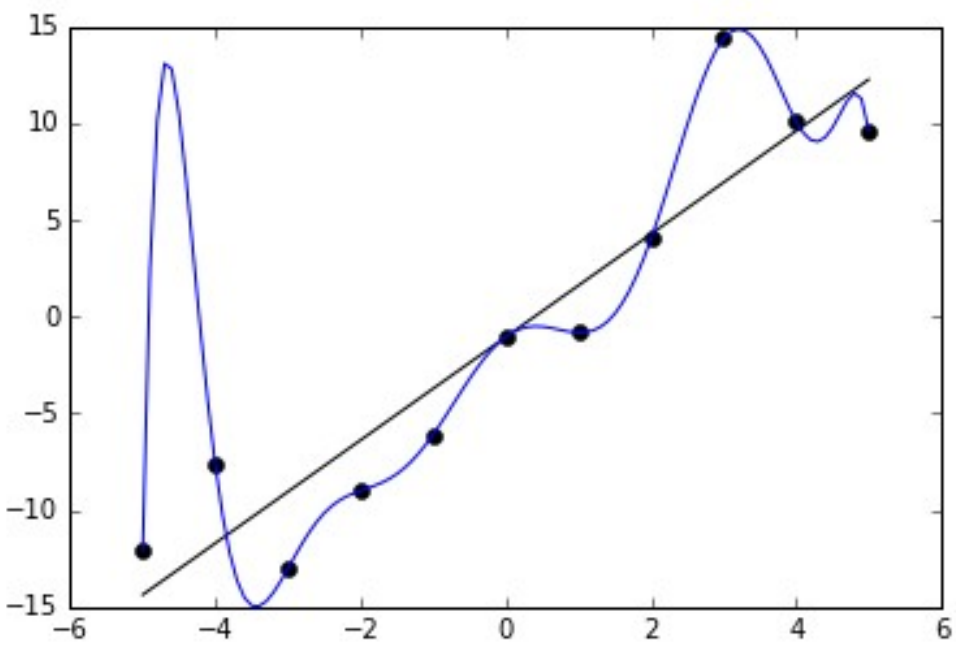
Weight Regularization

- If **large** weight |w|, L1 regularization shrinks the weight **much less** than L2 regularization does.
- when |w| is **small**, L1 regularization shrinks the weight **much more** than L2 regularization.
- The net result is that L1 regularization tends to concentrate the weight of the network in a relatively **small number of high-importance connections**, while the other weights are driven toward zero.

Regularization

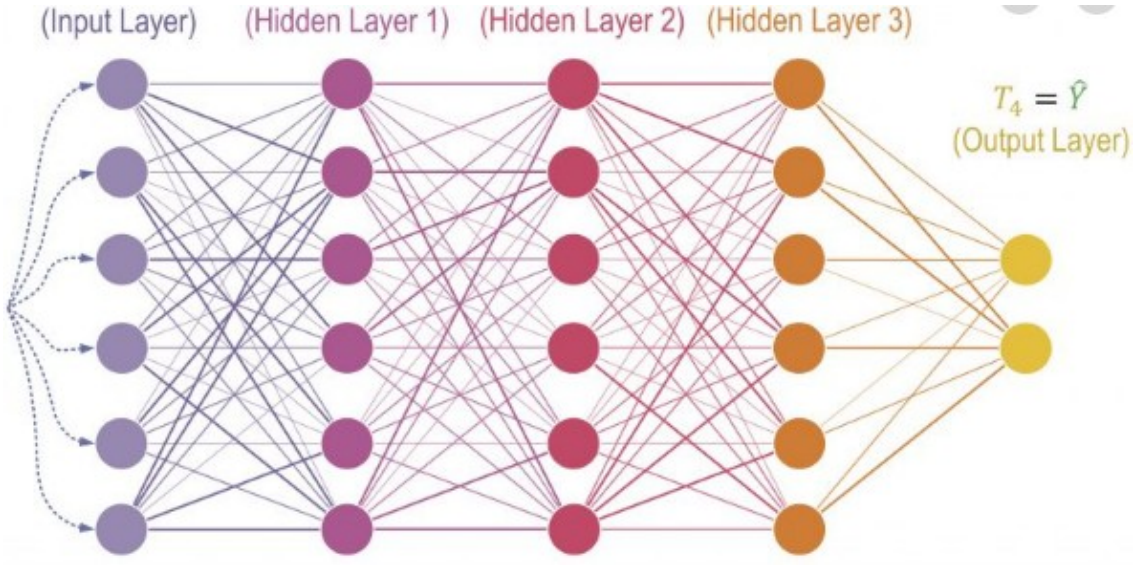
- Does regularization really reduce overfitting?
- Prevents excessive fluctuation of the coefficients. Thereby, reducing the chances of overfitting.

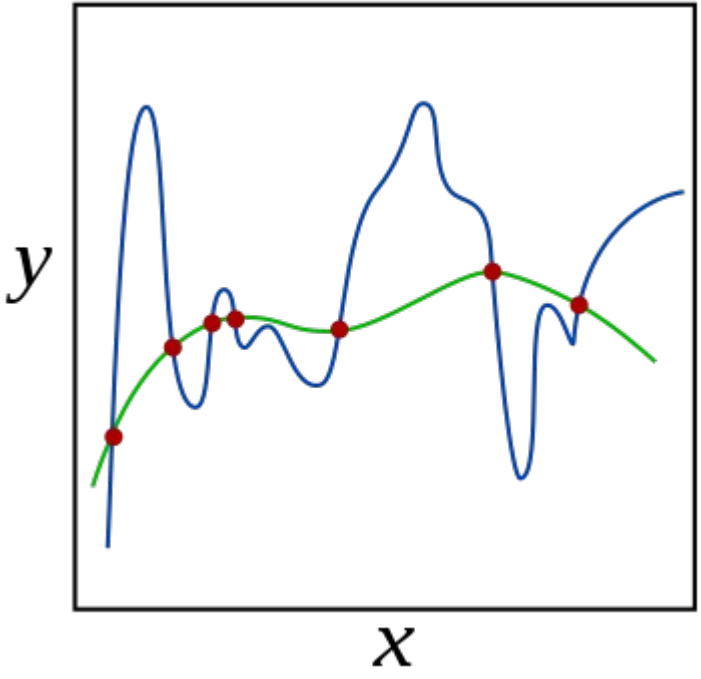




Regularization

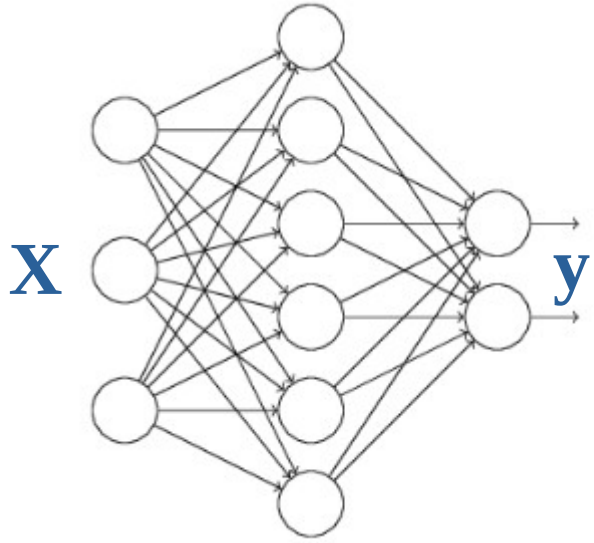
- If some weights are zero or closer to zero some neurons in network does not contribute much in the caliculation of output.





Dropout

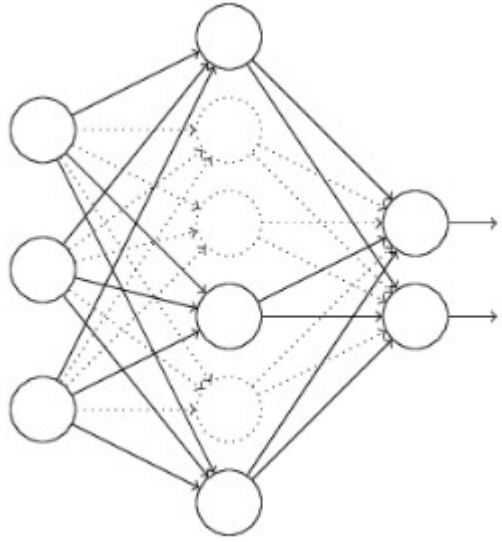
- Unlike L1 and L2 regularization, dropout doesn't rely on modifying the cost function.
- Instead, in dropout we modify the network itself.



- Train by forward-propagating x through the network, and then backpropagating to determine the contribution to the gradient.

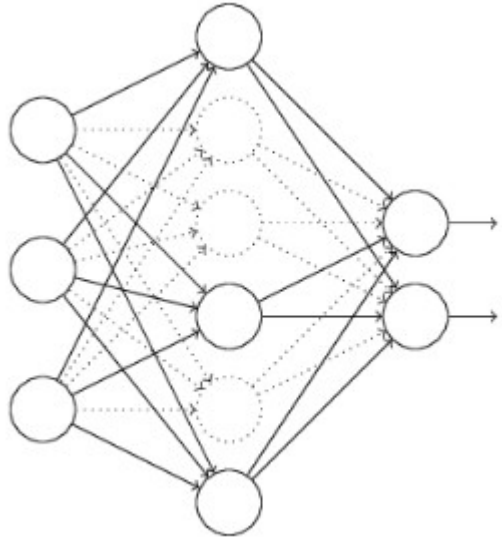
Dropout

- We start by randomly (and temporarily) deleting half the hidden neurons in the network, while leaving the input and output neurons untouched.
- Note that the dropout neurons, i.e., the neurons which have been temporarily deleted, are still ghosted in:



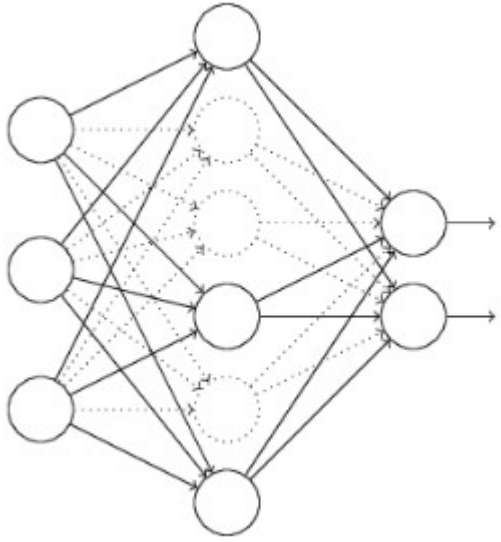
Dropout

- We forward-propagate the input x through the modified network, and then backpropagate the result, also through the modified network.
- We then repeat the process, first restoring the dropout neurons, then choosing a new random subset of hidden neurons to delete, estimating the gradient for a different mini-batch, and updating the weights and biases in the network.



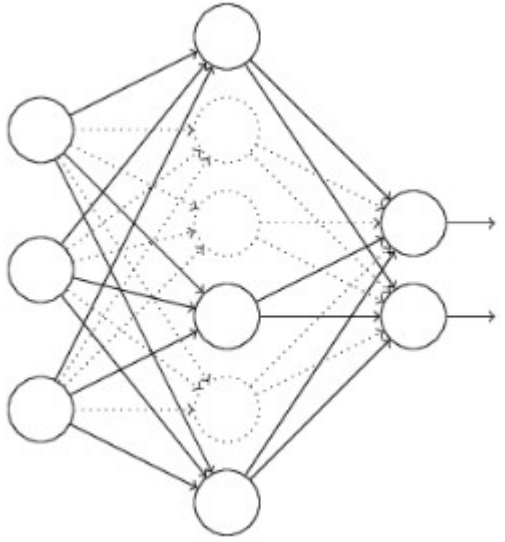
Dropout

- By repeating this process over and over, our network will learn a set of weights
- Those weights will have been learnt under conditions in which half the hidden neurons were dropped out.
- when we dropout different sets of neurons,
- it's rather like we're training different neural networks.



Dropout

- So the dropout procedure is like averaging the effects of a very large number of different networks.
- The different networks will overfit in different ways
- So, the net effect of dropout will be to reduce overfitting.



Data augmentation

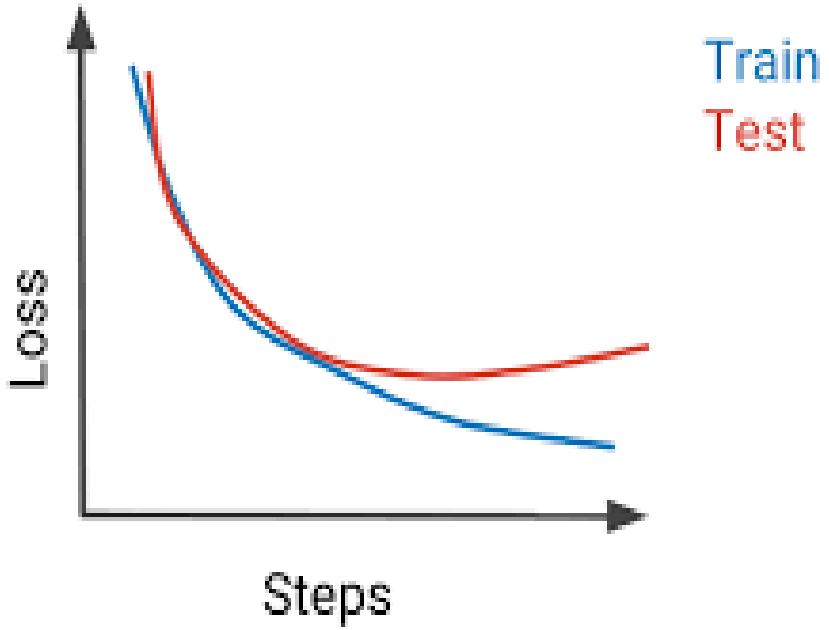
- **Artificially expanding the training data:**
- Data augmentation in data analysis are techniques used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data.
- It acts as a regularizer and helps reduce overfitting when training a machine learning model.

Data augmentation for image classifation

- **Transformations of images**
- Geometric transformations, flipping, color modification, cropping, rotation, noise injection and random erasing are used to augment image in deep learning
- **Introducing new synthetic images**
- If a dataset is too small, then a transformed image set via rotation and mirroring etc. may still be too small for a given problem. Another solution is the sourcing of entirely new and synthetic images through various techniques, for example the use of Generative adversarial networks to create new synthetic images for data augmentation.

Early stoping

- Early stopping rules provide guidance as to how many iterations can be run before the model begins to over-fit.



Early stoping

- Stop training when a monitored metric has stopped improving.
- A model.fit() training loop will check at end of every epoch whether the loss is no longer decreasing, considering the patience if applicable.
- Once it's found no longer decreasing, model.stop_training is marked True and the training terminates.

Batch Normalization

Weight Regularization

- https://towardsdatascience.com/intuitions-on-l1-and-l2-regularisation-235f2db4c261