Module 8.11 : Dropout

Other forms of regularization

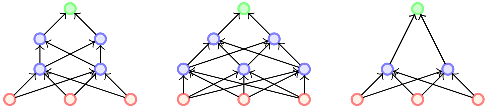
- l_2 regularization
- Dataset augmentation
- Parameter Sharing and tying
- Adding Noise to the inputs
- Adding Noise to the outputs
- Early stopping
- Ensemble methods
- Dropout

Other forms of regularization

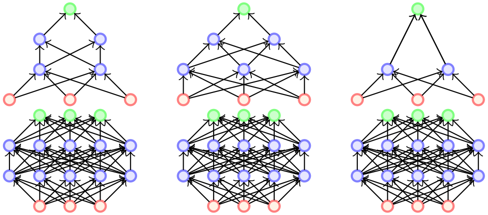
- l_2 regularization
- Dataset augmentation
- Parameter Sharing and tying
- Adding Noise to the inputs
- Adding Noise to the outputs
- Early stopping
- Ensemble methods
- Dropout

- Typically model averaging(bagging ensemble) always helps

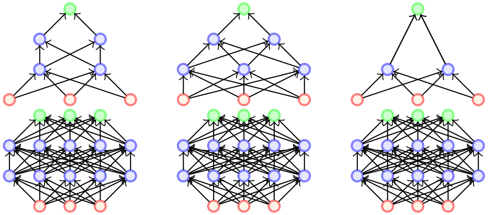
- Typically model averaging(bagging ensemble) always helps
- Training several large neural networks for making an ensemble is prohibitively expensive



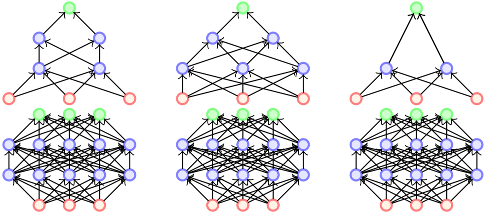
- Typically model averaging(bagging ensemble) always helps
- Training several large neural networks for making an ensemble is prohibitively expensive
- Option 1: Train several neural networks having different architectures(obviously expensive)



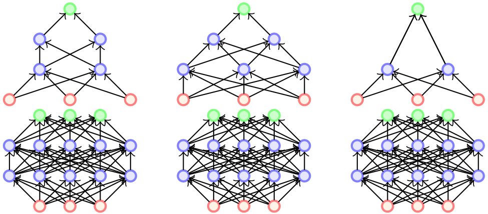
- Typically model averaging(bagging ensemble) always helps
- Training several large neural networks for making an ensemble is prohibitively expensive
- Option 1: Train several neural networks having different architectures(obviously expensive)
- Option 2: Train multiple instances of the same network using different training samples (again expensive)



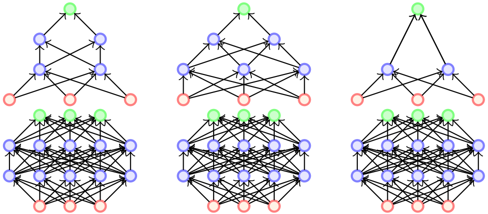
- Typically model averaging(bagging ensemble) always helps
- Training several large neural networks for making an ensemble is prohibitively expensive
- Option 1: Train several neural networks having different architectures(obviously expensive)
- Option 2: Train multiple instances of the same network using different training samples (again expensive)
- Even if we manage to train with option 1 or option 2, combining several models at test time is infeasible in real time applications



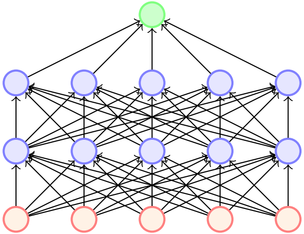
- Dropout is a technique which addresses both these issues.



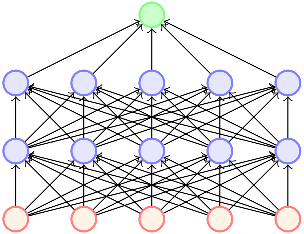
- Dropout is a technique which addresses both these issues.
- Effectively it allows training several neural networks without any significant computational overhead.

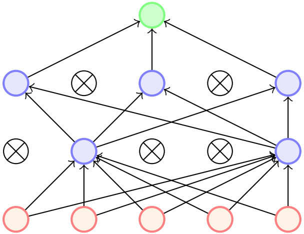


- Dropout is a technique which addresses both these issues.
- Effectively it allows training several neural networks without any significant computational overhead.
- Also gives an efficient approximate way of combining exponentially many different neural networks.

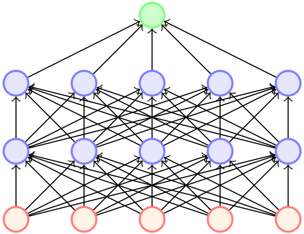
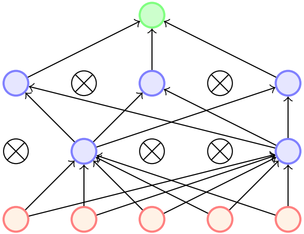


- Dropout refers to dropping out units

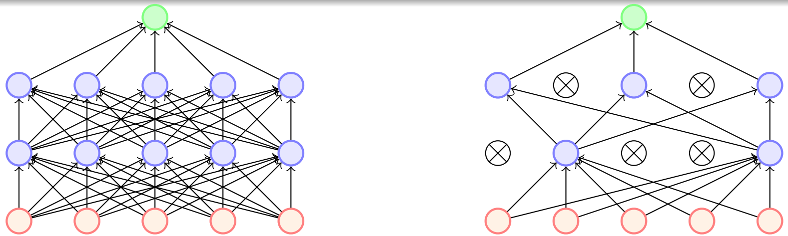


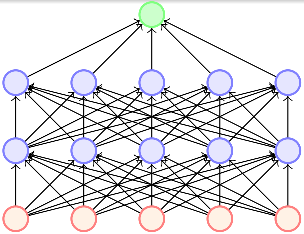


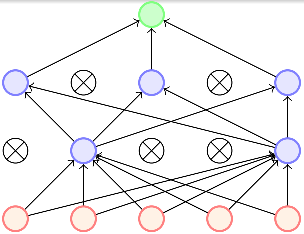
- Dropout refers to dropping out units
- Temporarily remove a node and all its incoming/outgoing connections resulting in a thinned network

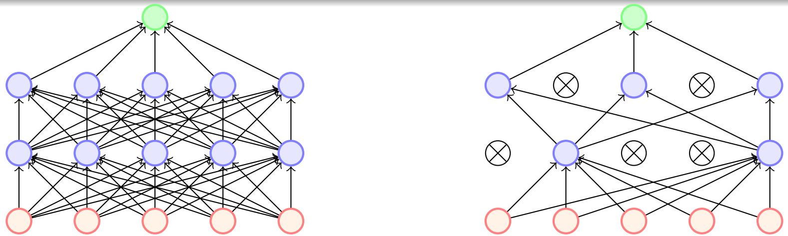
- Dropout refers to dropping out units
- Temporarily remove a node and all its incoming/outgoing connections resulting in a thinned network
- Each node is retained with a fixed probability (typically $p = 0.5$) for hidden nodes and $p = 0.8$ for visible nodes



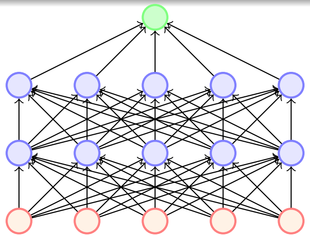


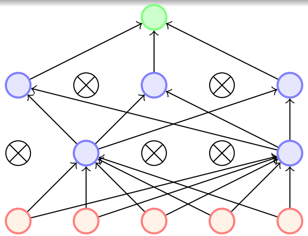


- Suppose a neural network has n nodes

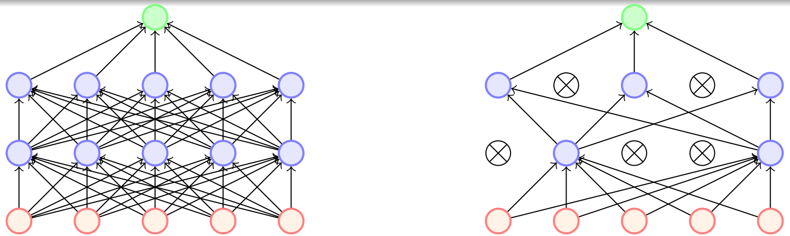


- Suppose a neural network has n nodes
- Using the dropout idea, each node can be retained or dropped

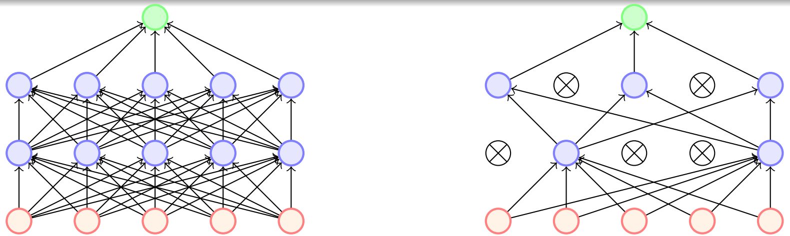




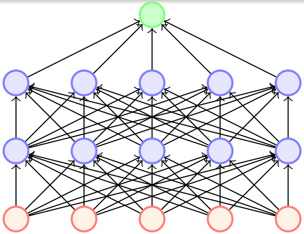
- Suppose a neural network has n nodes
- Using the dropout idea, each node can be retained or dropped
- For example, in the above case we drop 5 nodes to get a thinned network

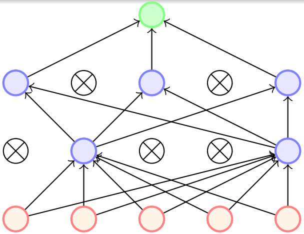


- Suppose a neural network has n nodes
- Using the dropout idea, each node can be retained or dropped
- For example, in the above case we drop 5 nodes to get a thinned network
- Given a total of n nodes, what are the total number of thinned networks that can be formed?

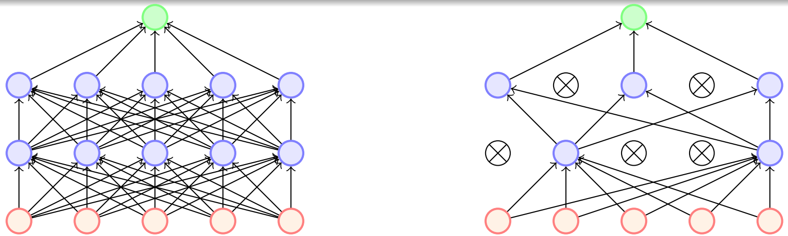


- Suppose a neural network has n nodes
- Using the dropout idea, each node can be retained or dropped
- For example, in the above case we drop 5 nodes to get a thinned network
- Given a total of n nodes, what are the total number of thinned networks that can be formed? 2^n

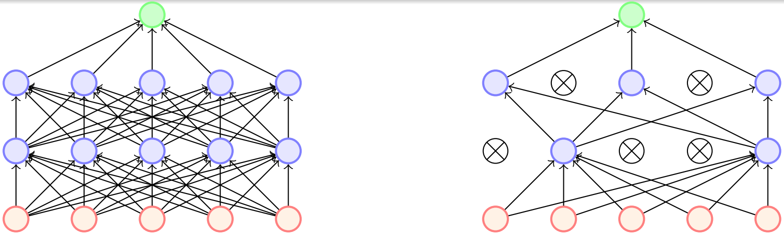




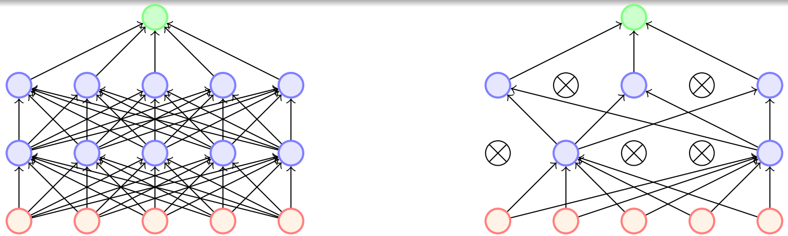
- Suppose a neural network has n nodes
- Using the dropout idea, each node can be retained or dropped
- For example, in the above case we drop 5 nodes to get a thinned network
- Given a total of n nodes, what are the total number of thinned networks that can be formed? 2^n
- Of course, this is prohibitively large and we cannot possibly train so many networks



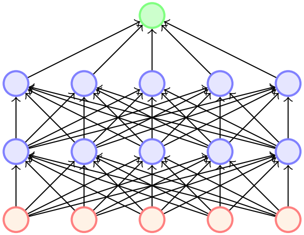
- Suppose a neural network has n nodes
- Using the dropout idea, each node can be retained or dropped
- For example, in the above case we drop 5 nodes to get a thinned network
- Given a total of n nodes, what are the total number of thinned networks that can be formed? 2^n
- Of course, this is prohibitively large and we cannot possibly train so many networks
- **Trick:** (1) Share the weights across all the networks

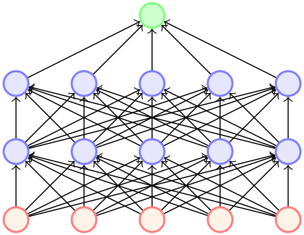


- Suppose a neural network has n nodes
- Using the dropout idea, each node can be retained or dropped
- For example, in the above case we drop 5 nodes to get a thinned network
- Given a total of n nodes, what are the total number of thinned networks that can be formed? 2^n
- Of course, this is prohibitively large and we cannot possibly train so many networks
- **Trick:** (1) Share the weights across all the networks
  (2) Sample a different network for each training instance

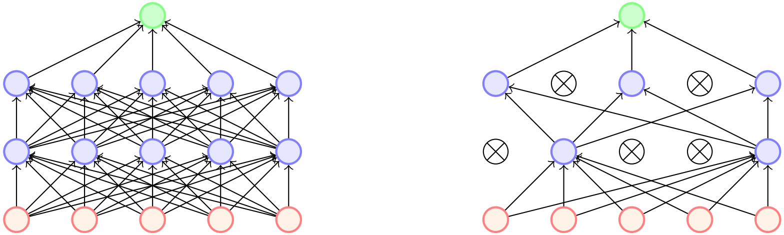


- Suppose a neural network has n nodes
- Using the dropout idea, each node can be retained or dropped
- For example, in the above case we drop 5 nodes to get a thinned network
- Given a total of n nodes, what are the total number of thinned networks that can be formed? 2^n
- Of course, this is prohibitively large and we cannot possibly train so many networks
- **Trick:** (1) Share the weights across all the networks
  (2) Sample a different network for each training instance
- Let us see how?

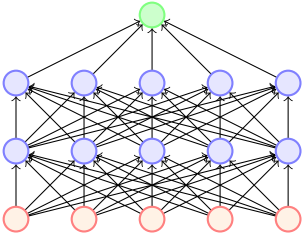


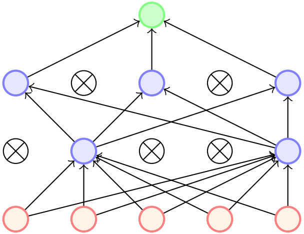


- We initialize all the parameters (weights) of the network and start training

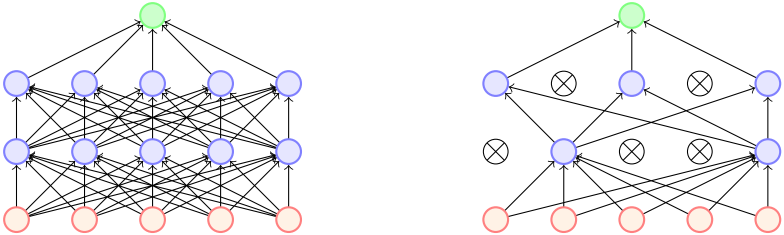


- We initialize all the parameters (weights) of the network and start training
- For the first training instance (or mini-batch), we apply dropout resulting in the thinned network

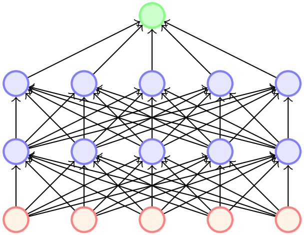


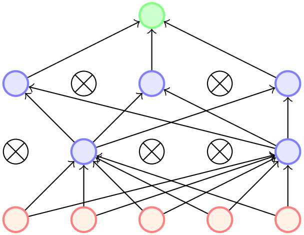


- We initialize all the parameters (weights) of the network and start training
- For the first training instance (or mini-batch), we apply dropout resulting in the thinned network
- We compute the loss and backpropagate

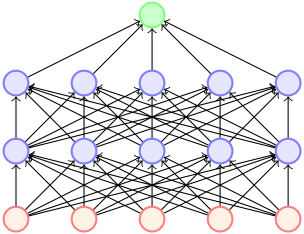


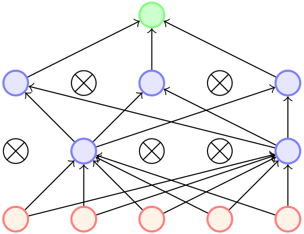
- We initialize all the parameters (weights) of the network and start training
- For the first training instance (or mini-batch), we apply dropout resulting in the thinned network
- We compute the loss and backpropagate
- Which parameters will we update?



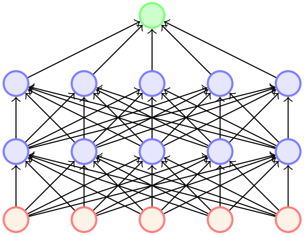


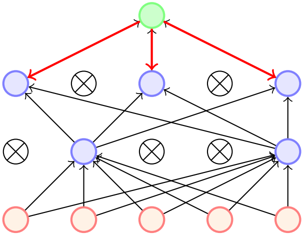
- We initialize all the parameters (weights) of the network and start training
- For the first training instance (or mini-batch), we apply dropout resulting in the thinned network
- We compute the loss and backpropagate
- Which parameters will we update? Only those which are active



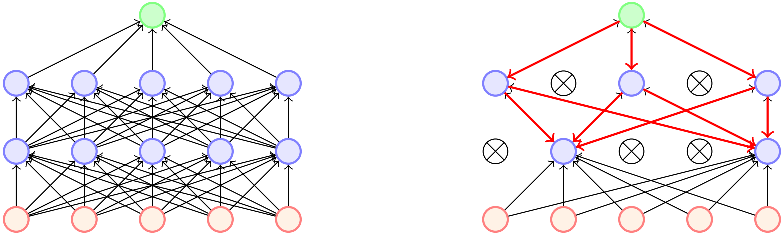


- We initialize all the parameters (weights) of the network and start training
- For the first training instance (or mini-batch), we apply dropout resulting in the thinned network
- We compute the loss and backpropagate
- Which parameters will we update? Only those which are active

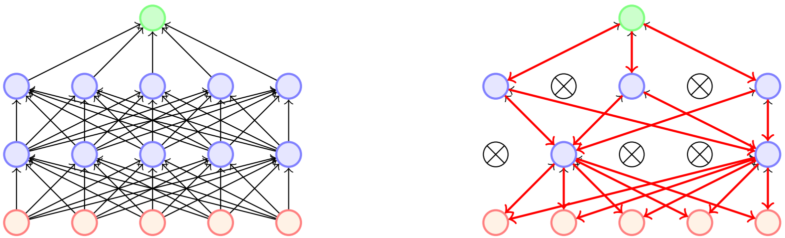




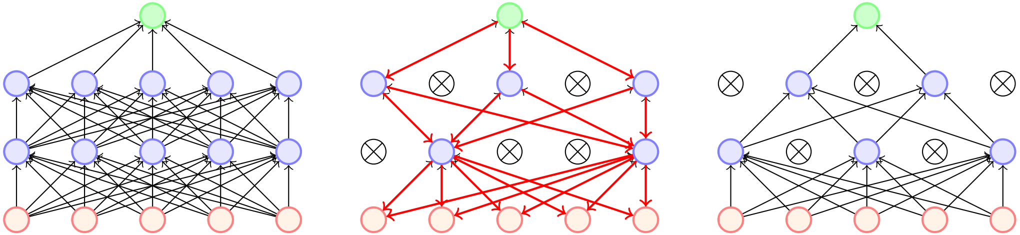
- We initialize all the parameters (weights) of the network and start training
- For the first training instance (or mini-batch), we apply dropout resulting in the thinned network
- We compute the loss and backpropagate
- Which parameters will we update? Only those which are active



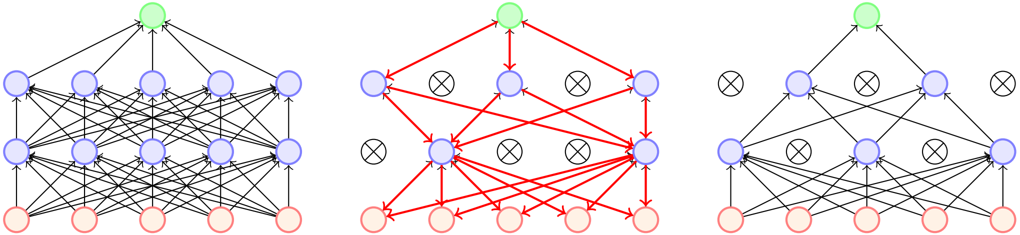
- We initialize all the parameters (weights) of the network and start training
- For the first training instance (or mini-batch), we apply dropout resulting in the thinned network
- We compute the loss and backpropagate
- Which parameters will we update? Only those which are active



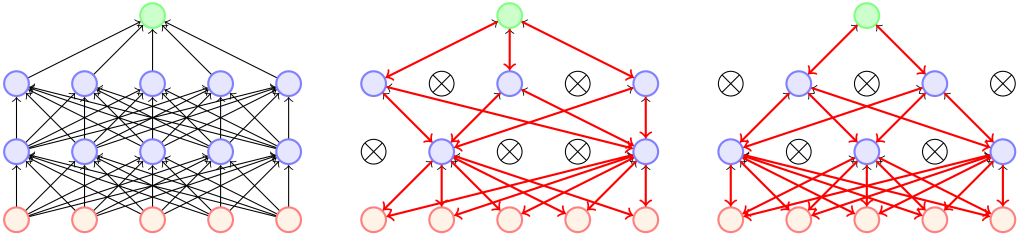
- We initialize all the parameters (weights) of the network and start training
- For the first training instance (or mini-batch), we apply dropout resulting in the thinned network
- We compute the loss and backpropagate
- Which parameters will we update? Only those which are active



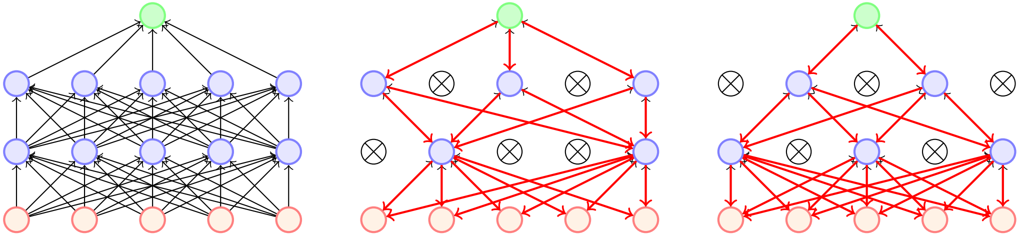
- For the second training instance (or mini-batch), we again apply dropout resulting in a different thinned network



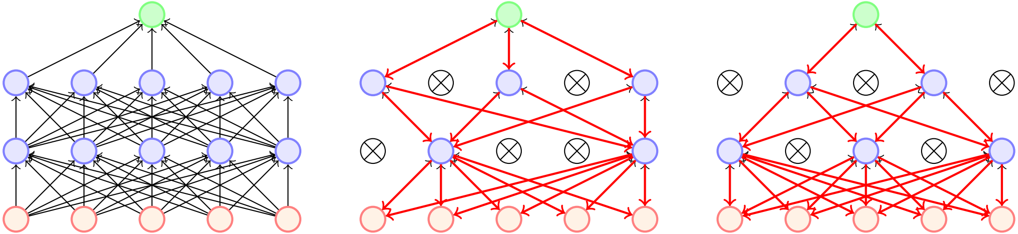
- For the second training instance (or mini-batch), we again apply dropout resulting in a different thinned network
- We again compute the loss and backpropagate to the active weights



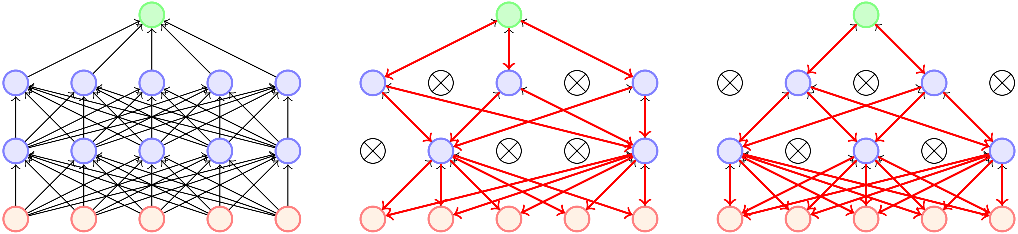
- For the second training instance (or mini-batch), we again apply dropout resulting in a different thinned network
- We again compute the loss and backpropagate to the active weights



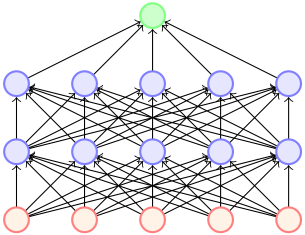
- For the second training instance (or mini-batch), we again apply dropout resulting in a different thinned network
- We again compute the loss and backpropagate to the active weights
- If the weight was active for both the training instances then it would have received two updates by now

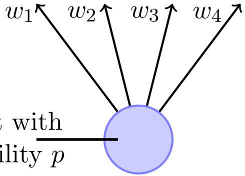


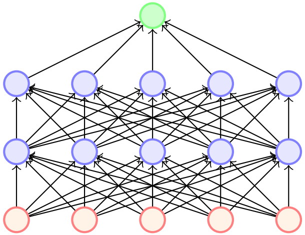
- For the second training instance (or mini-batch), we again apply dropout resulting in a different thinned network
- We again compute the loss and backpropagate to the active weights
- If the weight was active for both the training instances then it would have received two updates by now
- If the weight was active for only one of the training instances then it would have received only one updates by now

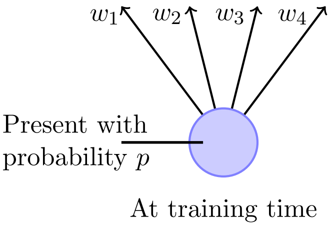


- For the second training instance (or mini-batch), we again apply dropout resulting in a different thinned network
- We again compute the loss and backpropagate to the active weights
- If the weight was active for both the training instances then it would have received two updates by now
- If the weight was active for only one of the training instances then it would have received only one updates by now
- Each thinned network gets trained rarely (or even never) but the parameter sharing ensures that no model has untrained or poorly trained parameters

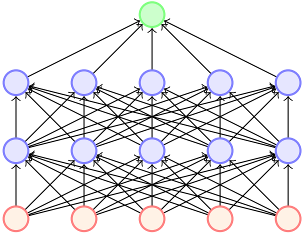


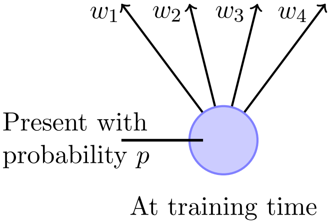

w_1
w_2
w_3
w_4
Present with
probability p
At training time



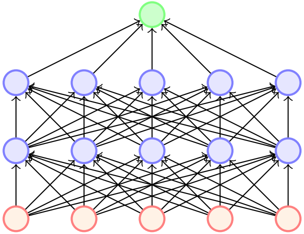


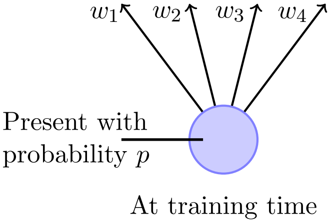
- What happens at test time?

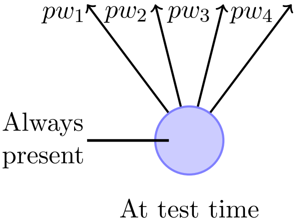




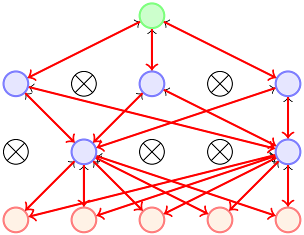
- What happens at test time?
- Impossible to aggregate the outputs of 2^n thinned networks



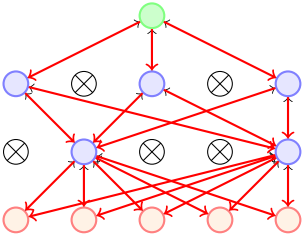




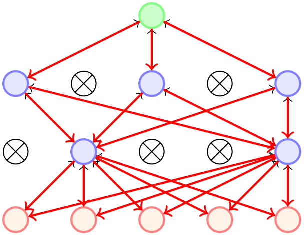
- What happens at test time?
- Impossible to aggregate the outputs of 2^n thinned networks
- Instead we use the full Neural Network and scale the output of each node by the fraction of times it was on during training



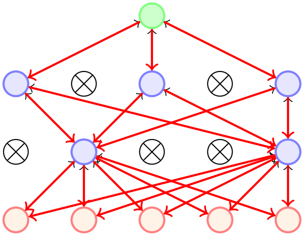
- Dropout essentially applies a masking noise to the hidden units



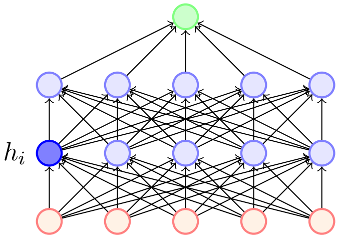
- Dropout essentially applies a masking noise to the hidden units
- Prevents hidden units from co-adapting

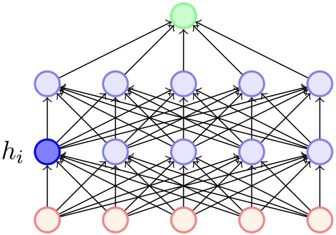


- Dropout essentially applies a masking noise to the hidden units
- Prevents hidden units from co-adapting
- Essentially a hidden unit cannot rely too much on other units as they may get dropped out any time

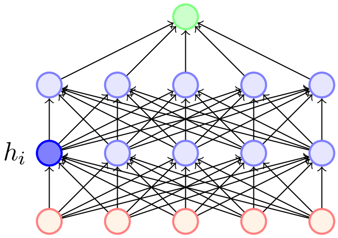


- Dropout essentially applies a masking noise to the hidden units
- Prevents hidden units from co-adapting
- Essentially a hidden unit cannot rely too much on other units as they may get dropped out any time
- Each hidden unit has to learn to be more robust to these random dropouts

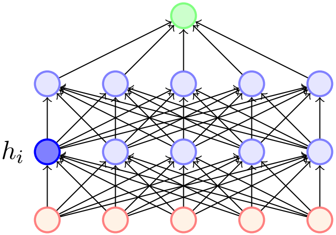

h_i



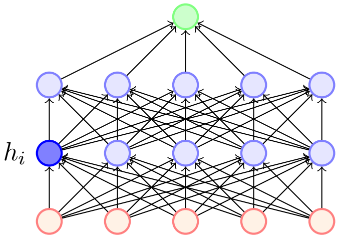
- Here is an example of how dropout helps in ensuring redundancy and robustness



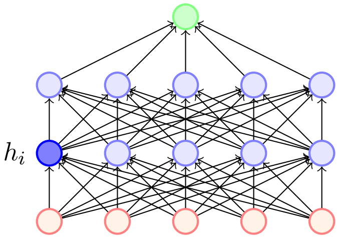
- Here is an example of how dropout helps in ensuring redundancy and robustness
- Suppose h_i learns to detect a face by firing on detecting a nose



- Here is an example of how dropout helps in ensuring redundancy and robustness
- Suppose h_i learns to detect a face by firing on detecting a nose
- Dropping h_i then corresponds to erasing the information that a nose exists



- Here is an example of how dropout helps in ensuring redundancy and robustness
- Suppose h_i learns to detect a face by firing on detecting a nose
- Dropping h_i then corresponds to erasing the information that a nose exists
- The model should then learn another h_i which redundantly encodes the presence of a nose



- Here is an example of how dropout helps in ensuring redundancy and robustness
- Suppose h_i learns to detect a face by firing on detecting a nose
- Dropping h_i then corresponds to erasing the information that a nose exists
- The model should then learn another h_i which redundantly encodes the presence of a nose
- Or the model should learn to detect the face using other features

Recap

- l_2 regularization
- Dataset augmentation
- Parameter Sharing and tying
- Adding Noise to the inputs
- Adding Noise to the outputs
- Early stopping
- Ensemble methods
- Dropout