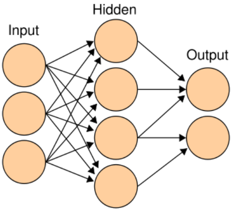
Neural Networks: Architecture Selection

11 March 2020

Architecture Selection

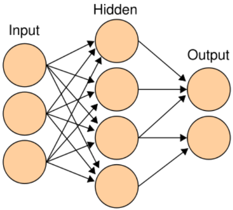
How many layers? How many neurons? How many weights?



- Occam's razor: the simplest network is always the best

Architecture Selection

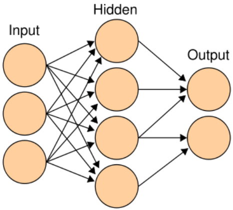
How many layers? How many neurons? How many weights?



- Occam's razor: the simplest network is always the best
- Too few neurons: insufficient complexity, poor (underfit) model

Architecture Selection

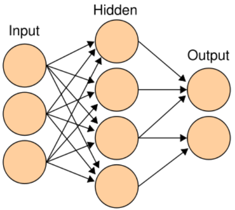
How many layers? How many neurons? How many weights?



- Occam's razor: the simplest network is always the best
- Too few neurons: insufficient complexity, poor (underfit) model
- Too many neurons: excessive complexity, poor (overfit) model

Architecture Selection

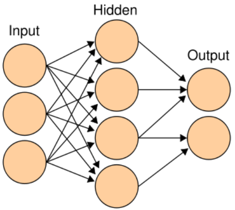
How many layers? How many neurons? How many weights?



- Occam's razor: the simplest network is always the best
- Too few neurons: insufficient complexity, poor (underfit) model
- Too many neurons: excessive complexity, poor (overfit) model
- What do we know about the complexity of any given problem?

Architecture Selection

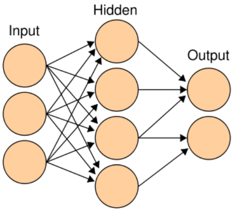
How many layers? How many neurons? How many weights?



- Occam's razor: the simplest network is always the best
- Too few neurons: insufficient complexity, poor (underfit) model
- Too many neurons: excessive complexity, poor (overfit) model
- What do we know about the complexity of any given problem?
    - Number of training patterns

Architecture Selection

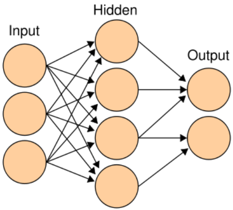
How many layers? How many neurons? How many weights?



- Occam's razor: the simplest network is always the best
- Too few neurons: insufficient complexity, poor (underfit) model
- Too many neurons: excessive complexity, poor (overfit) model
- What do we know about the complexity of any given problem?
  - Number of training patterns
  - Input/output dimensionality

Architecture Selection

How many layers? How many neurons? How many weights?



- Occam's razor: the simplest network is always the best
- Too few neurons: insufficient complexity, poor (underfit) model
- Too many neurons: excessive complexity, poor (overfit) model
- What do we know about the complexity of any given problem?
  - Number of training patterns
  - Input/output dimensionality
  - Complexity of input-output relationship (function)

Architecture Selection

Making guesstimates based on the number of data patterns

- Rule of thumb: The number of training patterns should always exceed the number of free parameters.

Architecture Selection

Making guesstimates based on the number of data patterns

- Rule of thumb: The number of training patterns should always exceed the number of free parameters.
- Why?

Architecture Selection

Making guesstimates based on the number of data patterns

- Rule of thumb: The number of training patterns should always exceed the number of free parameters.
- Why?
- Because otherwise overfitting may easily occur: free parameters will match exact patterns rather than extracting the big picture

Architecture Selection

Making guesstimates based on the number of data patterns

- Rule of thumb: The number of training patterns should always exceed the number of free parameters.
- Why?
- Because otherwise overfitting may easily occur: free parameters will match exact patterns rather than extracting the big picture
- *Food for thought: knowledge is a result of data compression*

Architecture Selection

Making guesstimates based on the number of data patterns

- Rule of thumb: The number of training patterns should always exceed the number of free parameters.
- Why?
- Because otherwise overfitting may easily occur: free parameters will match exact patterns rather than extracting the big picture
- *Food for thought: knowledge is a result of data compression*
- If the number of patterns is the same as the number of parameters, we might as well do k-nearest-neighbour classification (no learning)

Architecture Selection
Making guesstimates based on dimensionality

Let N be the input dimensionality, M be the output dimensionality, T be the number of training patterns, and N_h be the number of hidden neurons.

Architecture Selection

Making guesstimates based on dimensionality

Let N be the input dimensionality, M be the output dimensionality, T be the number of training patterns, and N_h be the number of hidden neurons.

1. $N_h = (N + M)/2$

Architecture Selection
Making guesstimates based on dimensionality

Let N be the input dimensionality, M be the output dimensionality, T be the number of training patterns, and N_h be the number of hidden neurons.

1. $N_h = (N + M)/2$
    - No mathematical justification
    - Used by default in Weka (scientific tool)

Architecture Selection
Making guesstimates based on dimensionality

Let N be the input dimensionality, M be the output dimensionality, T be the number of training patterns, and N_h be the number of hidden neurons.

1. $N_h = (N + M)/2$
   - No mathematical justification
   - Used by default in Weka (scientific tool)
2. $N_h = T/(5 * (N + M))$

Architecture Selection
Making guesstimates based on dimensionality

Let N be the input dimensionality, M be the output dimensionality, T be the number of training patterns, and N_h be the number of hidden neurons.

1. $N_h = (N + M)/2$
   - No mathematical justification
   - Used by default in Weka (scientific tool)
2. $N_h = T/\left(5 * (N + M)\right)$
   - Five training samples per weight (magic numbers!)
   - Used by default in Neuralware (commercial tool)

Architecture Selection

Making guesstimates based on dimensionality

Let N be the input dimensionality, M be the output dimensionality, T be the number of training patterns, and N_h be the number of hidden neurons.

1. $N_h = (N + M)/2$
   - No mathematical justification
   - Used by default in Weka (scientific tool)
2. $N_h = T/\big(5 * (N + M)\big)$
   - Five training samples per weight (magic numbers!)
   - Used by default in Neuralware (commercial tool)
3. $N_h = \sqrt{T/(N \log T)}$

Architecture Selection

Making guesstimates based on dimensionality

Let N be the input dimensionality, M be the output dimensionality, T be the number of training patterns, and N_h be the number of hidden neurons.

1. $N_h = (N + M)/2$
   - No mathematical justification
   - Used by default in Weka (scientific tool)
2. $N_h = T/\big(5 * (N + M)\big)$
   - Five training samples per weight (magic numbers!)
   - Used by default in Neuralware (commercial tool)
3. $N_h = \sqrt{T/(N \log T)}$
   - Optimises the mean integrated squared error for some classes of smooth functions
   - Smoothness assumption?

Architecture Selection

Making guesstimates based on dimensionality

Let N be the input dimensionality, M be the output dimensionality, T be the number of training patterns, and N_h be the number of hidden neurons.

1. $N_h = (N + M)/2$
    - No mathematical justification
    - Used by default in Weka (scientific tool)
2. $N_h = T/\big(5 * (N + M)\big)$
    - Five training samples per weight (magic numbers!)
    - Used by default in Neuralware (commercial tool)
3. $N_h = \sqrt{T/(N \log T)}$
    - Optimises the mean integrated squared error for some classes of smooth functions
    - Smoothness assumption?
4. $N_h = \sqrt{N * M}$

Architecture Selection

Making guesstimates based on dimensionality

Let N be the input dimensionality, M be the output dimensionality, T be the number of training patterns, and N_h be the number of hidden neurons.

1. $N_h = (N + M)/2$
   - No mathematical justification
   - Used by default in Weka (scientific tool)
2. $N_h = T/\big(5 * (N + M)\big)$
   - Five training samples per weight (magic numbers!)
   - Used by default in Neuralware (commercial tool)
3. $N_h = \sqrt{T/(N \log T)}$
   - Optimises the mean integrated squared error for some classes of smooth functions
   - Smoothness assumption?
4. $N_h = \sqrt{N * M}$
   - Pyramidal structures (# neurons reducing from layer to layer) have shown good generalisation performance

Architecture Selection

Making guesstimates based on dimensionality

- ...Out of the four rules, the second one gave the most reliably good results on a binary benchmark of varied complexity/dimensionality

Architecture Selection
Making guesstimates based on dimensionality

- ...Out of the four rules, the second one gave the most reliably good results on a binary benchmark of varied complexity/dimensionality
- Reason: Rule # 2 generated the largest architectures

Architecture Selection

Making guesstimates based on dimensionality

- ...Out of the four rules, the second one gave the most reliably good results on a binary benchmark of varied complexity/dimensionality
- Reason: Rule # 2 generated the largest architectures
- Larger architectures tend to make the problem easier for gradient descent

Architecture Selection
Making guesstimates based on dimensionality

- ...Out of the four rules, the second one gave the most reliably good results on a binary benchmark of varied complexity/dimensionality
- Reason: Rule # 2 generated the largest architectures
- Larger architectures tend to make the problem easier for gradient descent
  - "Blessing of dimensionality: mathematical foundations of the statistical physics of data", A.N. Gorban, I.Y. Tyukin

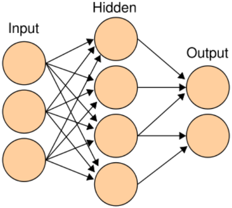
Architecture Selection
Making guesstimates based on dimensionality

- ...Out of the four rules, the second one gave the most reliably good results on a binary benchmark of varied complexity/dimensionality
- Reason: Rule # 2 generated the largest architectures
- Larger architectures tend to make the problem easier for gradient descent
  - "Blessing of dimensionality: mathematical foundations of the statistical physics of data", A.N. Gorban, I.Y. Tyukin
  - The number of local minima reduces with an increase in dimensionality

Architecture Selection
Making guesstimates based on dimensionality

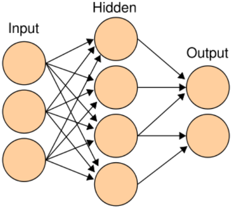
- ...Out of the four rules, the second one gave the most reliably good results on a binary benchmark of varied complexity/dimensionality
- Reason: Rule # 2 generated the largest architectures
- Larger architectures tend to make the problem easier for gradient descent
  - "Blessing of dimensionality: mathematical foundations of the statistical physics of data", A.N. Gorban, I.Y. Tyukin
  - The number of local minima reduces with an increase in dimensionality
  - Instead of geting out of local minima, gradient descent needs to get over multiple saddle points
  - Saddle points are easier to deal with than local minima!

Architecture Selection



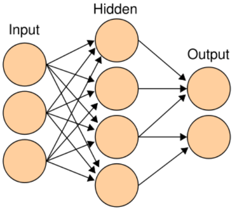
- What about the layers?

Architecture Selection



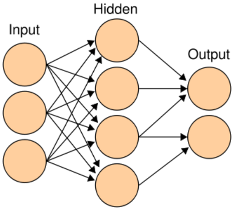
- What about the layers?
    - More can be better (deep learning), but is harder to train

Architecture Selection



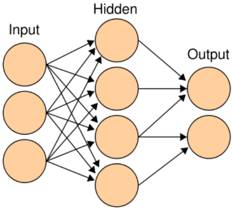
- What about the layers?
  - More can be better (deep learning), but is harder to train
  - How much is enough?

Architecture Selection



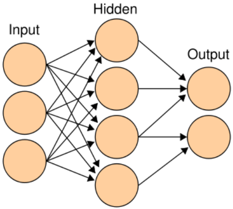
- What about the layers?
  - More can be better (deep learning), but is harder to train
  - How much is enough?
  - `https://playground.tensorflow.org`
- What about the weights?

Architecture Selection



- What about the layers?
  - More can be better (deep learning), but is harder to train
  - How much is enough?
  - `https://playground.tensorflow.org`
- What about the weights?
  - If the training algorithm is good, it should be capable of setting irrelevant weights to zero

Architecture Selection



- What about the layers?
  - More can be better (deep learning), but is harder to train
  - How much is enough?
  - https://playground.tensorflow.org
- What about the weights?
  - If the training algorithm is good, it should be capable of setting irrelevant weights to zero
  - Regularisation: minimise not only the error, but also the complexity

Regularisation

Penalizing complexity

- Add a penalty term to the objective function:
  - $E_{NN} = E + \lambda E_p$
- Now we are minimizing both the **error** and the **complexity**

Regularisation

Penalizing complexity

- Add a penalty term to the objective function:
  - $E_{NN} = E + \lambda E_p$
- Now we are minimizing both the **error** and the **complexity**

How do you measure complexity?

Regularisation

Penalizing complexity

- Add a penalty term to the objective function:
  - $E_{NN} = E + \lambda E_p$
- Now we are minimizing both the **error** and the **complexity**

How do you measure complexity?

- Weight decay:
  - $E_p = \sum_{i=1}^{W} w_i^2$
  - Minimize weight vector magnitude; only constantly reinforced weights will survive

Regularisation

Penalizing complexity

- Add a penalty term to the objective function:
  - $E_{NN} = E + \lambda E_p$
- Now we are minimizing both the **error** and the **complexity**

How do you measure complexity?

- Weight decay:
  - $E_p = \sum_{i=1}^{W} w_i^2$
  - Minimize weight vector magnitude; only constantly reinforced weights will survive
- Weight elimination:
  - $E_p = \sum_{i=1}^{W} \frac{w_i^2 / w_0^2}{1 + w_i^2 / w_0^2}$
  - w_0 determines the "significance" of weights
  - $|w_i| >> w_0$ => high complexity, penalize more
  - $|w_i| << w_0$ => low complexity, penalize less

Regularisation

Penalizing complexity

- Laplace: L1 regularisation
  - $E_p = \sum_{i=1}^{W} |w_i|$
  - Contribution of each w to the penalty term increases linearly with the increase of the weight

Regularisation

Penalizing complexity

- Laplace: L1 regularisation
  - $E_p = \sum_{i=1}^{W} |w_i|$
  - Contribution of each w to the penalty term increases linearly with the increase of the weight
- Multiple other penalty functions were proposed

Regularisation

Penalizing complexity

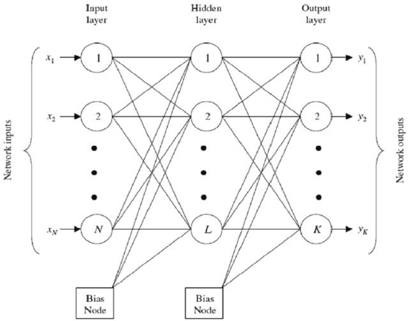
- Laplace: L1 regularisation
  - $E_p = \sum_{i=1}^{W} |w_i|$
  - Contribution of each w to the penalty term increases linearly with the increase of the weight
- Multiple other penalty functions were proposed
- Consider the objective function:
  - $E_{NN} = E + \lambda E_p$

Regularisation

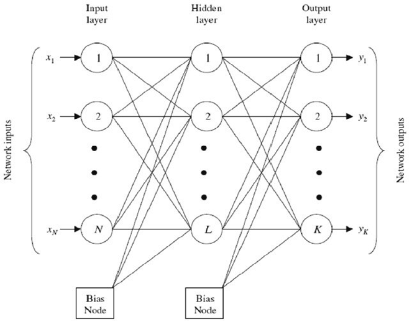
Penalizing complexity

- Laplace: L1 regularisation
  - $E_p = \sum_{i=1}^{W} |w_i|$
  - Contribution of each w to the penalty term increases linearly with the increase of the weight
- Multiple other penalty functions were proposed
- Consider the objective function:
  - $E_{NN} = E + \lambda E_p$
- How do we choose λ?
  - Cross-validation
  - Make it adaptive?

Regularisation and the Bias Weights



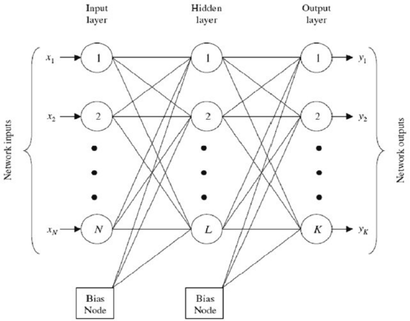
Regularisation and the Bias Weights



Should we penalise the biases?

- Regularisation makes the function smoother, i.e. less sensitive to changes in the input

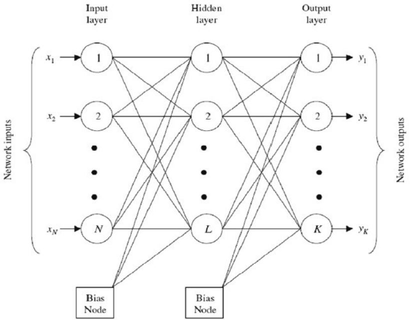
Regularisation and the Bias Weights



Should we penalise the biases?

- Regularisation makes the function smoother, i.e. less sensitive to changes in the input
- Biases provide constant input

Regularisation and the Bias Weights



Should we penalise the biases?

- Regularisation makes the function smoother, i.e. less sensitive to changes in the input
- Biases provide constant input
- In practice, we usually **do not** regularise the bias weights
- Regularising biases may cause underfitting

Regularisation

"Dropout: A Simple Way to Prevent Neural Networks from Overfitting", Srivastava et al. 2012

Dropout: a new form of regularization

- Hinton, 2012: overfitting occurs because the model is too complex, eg. each hidden unit relies on neighbour units to make the final prediction

Regularisation

"Dropout: A Simple Way to Prevent Neural Networks from Overfitting", Srivastava et al. 2012

Dropout: a new form of regularization

- Hinton, 2012: overfitting occurs because the model is too complex, eg. each hidden unit relies on neighbour units to make the final prediction
- "Dropout": On each presentation of each training pattern, each hidden unit is randomly omitted from the network with a probability of 0.5, so a hidden unit cannot rely on other hidden units being present

Regularisation

"Dropout: A Simple Way to Prevent Neural Networks from Overfitting", Srivastava et al. 2012

Dropout: a new form of regularization

- Hinton, 2012: overfitting occurs because the model is too complex, eg. each hidden unit relies on neighbour units to make the final prediction
- "Dropout": On each presentation of each training pattern, each hidden unit is randomly omitted from the network with a probability of 0.5, so a hidden unit cannot rely on other hidden units being present
- Force hidden units to "take responsibility"
- More robust models are obtained by preventing "co-adaptation"

Regularisation

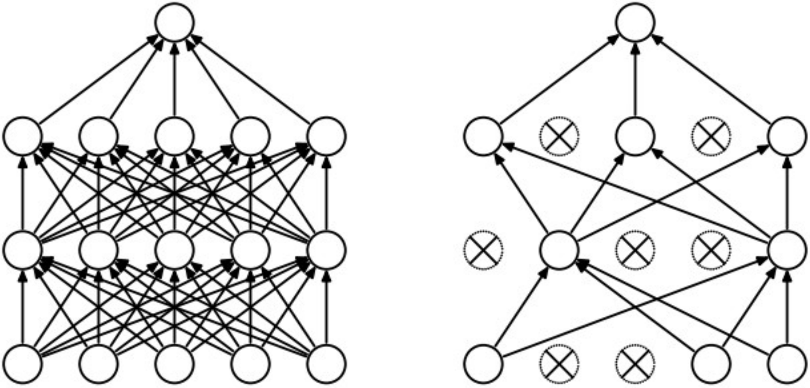
"Dropout: A Simple Way to Prevent Neural Networks from Overfitting", Srivastava et al. 2012

Dropout: a new form of regularization

- Hinton, 2012: overfitting occurs because the model is too complex, eg. each hidden unit relies on neighbour units to make the final prediction
- "Dropout": On each presentation of each training pattern, each hidden unit is randomly omitted from the network with a probability of 0.5, so a hidden unit cannot rely on other hidden units being present
- Force hidden units to "take responsibility"
- More robust models are obtained by preventing "co-adaptation"
- Can be combined with other regularisation methods

Regularisation

Dropout: a new form of regularization



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

Regularisation

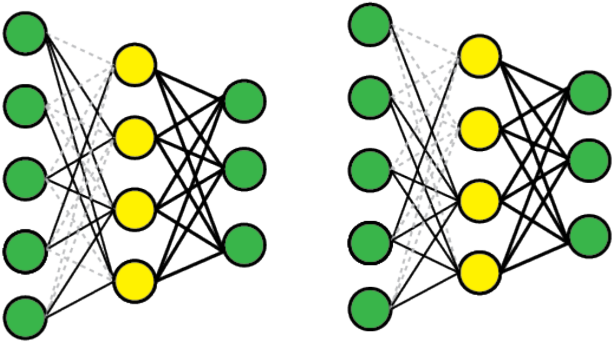
"Dropout: A Simple Way to Prevent Neural Networks from Overfitting", Srivastava et al. 2012

"A motivation for dropout comes from a theory of the role of sex in evolution (Livnat et al., 2010). Sexual reproduction involves taking half the genes of one parent and half of the other, adding a very small amount of random mutation, and combining them to produce an offspring. The asexual alternative is to create an offspring with a slightly mutated copy of the parent's genes. It seems plausible that asexual reproduction should be a better way to optimize individual fitness because a good set of genes that have come to work well together can be passed on directly to the offspring. On the other hand, sexual reproduction is likely to break up these co-adapted sets of genes, especially if these sets are large and, intuitively, this should decrease the fitness of organisms that have already evolved complicated coadaptations. However, sexual reproduction is the way most advanced organisms evolved."

Regularisation

Dropconnect: a generalisation of Dropout

You can also “disable” weights rather than neurons:



Essentially, we train an ensemble that looks like a single NN

Neural Network Construction

How to automate architecture selection

How do we automatically construct an optimal architecture?

Neural Network Construction

How to automate architecture selection

How do we automatically construct an optimal architecture?

Minimalistic approach

- Start with just a few neurons, add more when stagnation occurs

Neural Network Construction

How to automate architecture selection

How do we automatically construct an optimal architecture?

Minimalistic approach

- Start with just a few neurons, add more when stagnation occurs
  - The NN contains a working model when a new neuron is added => integrating the new neuron may slow down training

Neural Network Construction

How to automate architecture selection

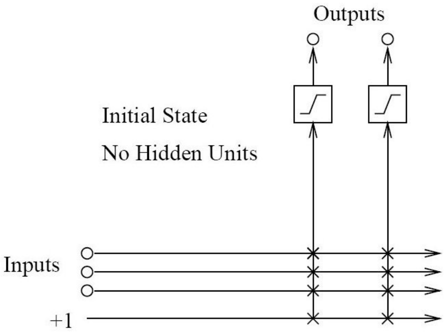
How do we automatically construct an optimal architecture?

Minimalistic approach

- Start with just a few neurons, add more when stagnation occurs
  - The NN contains a working model when a new neuron is added => integrating the new neuron may slow down training
  - How do we decide when to add a neuron, and when to stop growing?
  - How do we expand this to adding layers?

Cascade Correlation NNs

Goal: Minimial effective architecture, easy training. Fahlman & Lebiere, 1990.

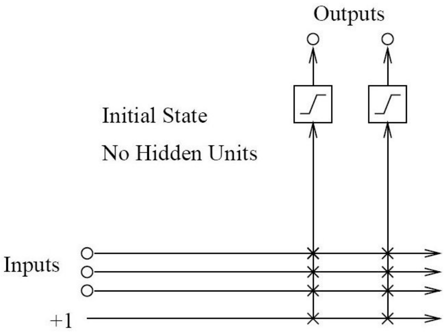


First iteration

- Occam's Razor: if you can do it with a perceptron, you don't need a NN

Cascade Correlation NNs

Goal: Minimial effective architecture, easy training. Fahlman & Lebiere, 1990.

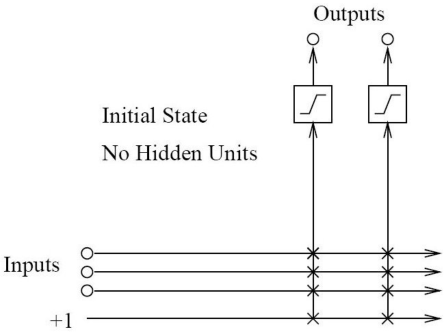


First iteration

- Occam's Razor: if you can do it with a perceptron, you don't need a NN
- Every input is connected to every output

Cascade Correlation NNs

Goal: Minimial effective architecture, easy training. Fahlman & Lebiere, 1990.

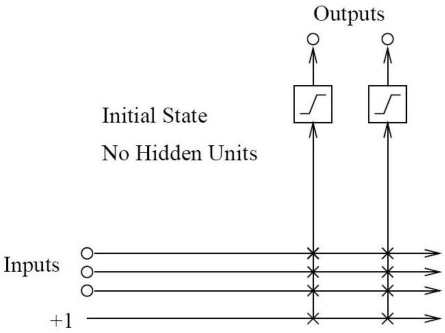


First iteration

- Occam's Razor: if you can do it with a perceptron, you don't need a NN
- Every input is connected to every output
- Training starts with no hidden units

Cascade Correlation NNs

Goal: Minimial effective architecture, easy training. Fahlman & Lebiere, 1990.

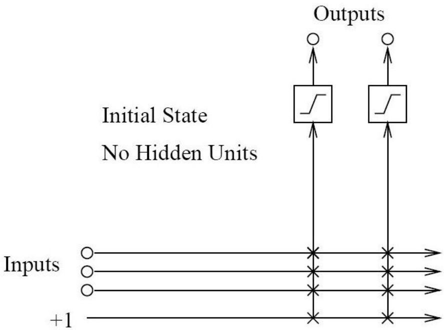


First iteration

- Occam's Razor: if you can do it with a perceptron, you don't need a NN
- Every input is connected to every output
- Training starts with no hidden units
- Train the perceptrons till training stagnates

Cascade Correlation NNs

Goal: Minimial effective architecture, easy training. Fahlman & Lebiere, 1990.

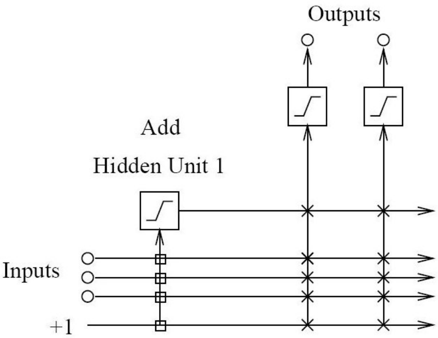


First iteration

- Occam's Razor: if you can do it with a perceptron, you don't need a NN
- Every input is connected to every output
- Training starts with no hidden units
- Train the perceptrons till training stagnates
- If the accuracy after training is unacceptable, a single hidden neuron is added

Cascade Correlation NNs

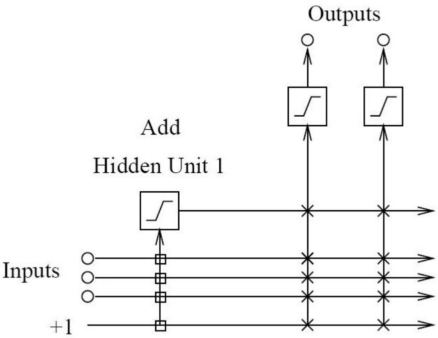
Goal: Minimial effective architecture, easy training



- All previously trained connections remain

Cascade Correlation NNs

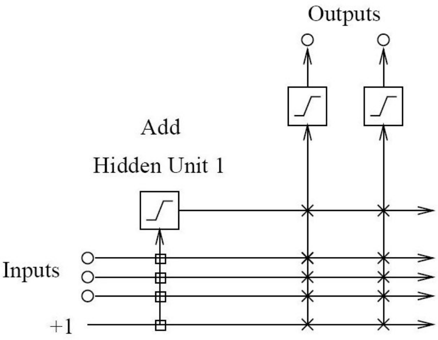
Goal: Minimial effective architecture, easy training



- All previously trained connections remain
- All inputs are connected to the hidden neuron

Cascade Correlation NNs

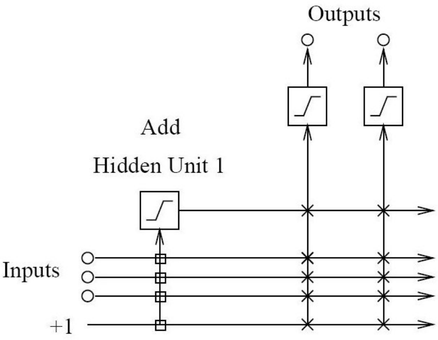
Goal: Minimial effective architecture, easy training



- All previously trained connections remain
- All inputs are connected to the hidden neuron
- Hidden neuron is connected to all output neurons

Cascade Correlation NNs

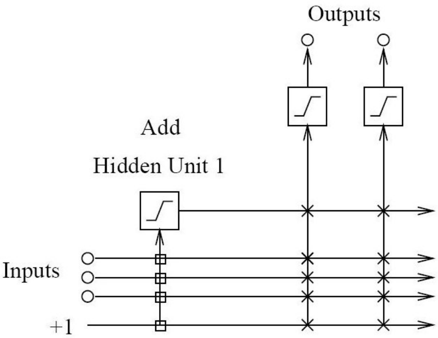
Goal: Minimial effective architecture, easy training



- All previously trained connections remain
- All inputs are connected to the hidden neuron
- Hidden neuron is connected to all output neurons
- Hidden neuron's input weights are adjusted to maximize covariance between the new neuron's output and the NN error

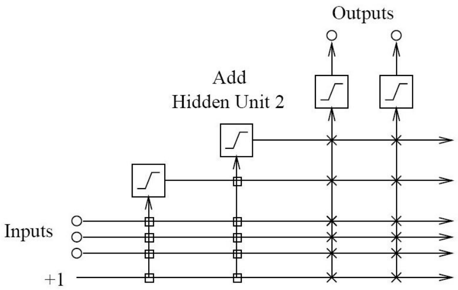
Cascade Correlation NNs

Goal: Minimial effective architecture, easy training



- All previously trained connections remain
- All inputs are connected to the hidden neuron
- Hidden neuron is connected to all output neurons
- Hidden neuron's input weights are adjusted to maximize covariance between the new neuron's output and the NN error
- The input weights are then frozen, and only the output weights are trained

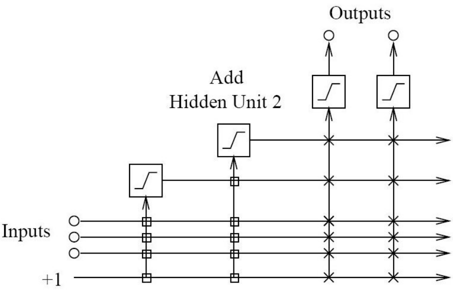
Cascade Correlation NNs
Goal: Minimial effective architecture, easy training



- Train input weights first

Cascade Correlation NNs

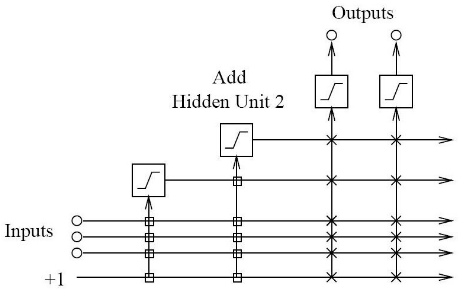
Goal: Minimial effective architecture, easy training



- Train input weights first
- Once trained, freeze the input weights

Cascade Correlation NNs

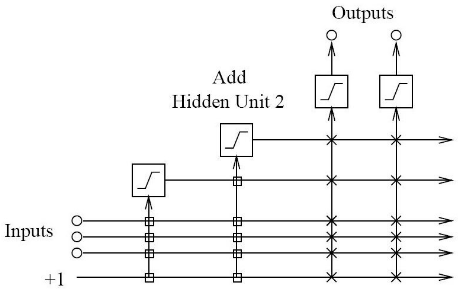
Goal: Minimial effective architecture, easy training



- Train input weights first
- Once trained, freeze the input weights
- Apply perceptron learning rules to train the output weights

Cascade Correlation NNs

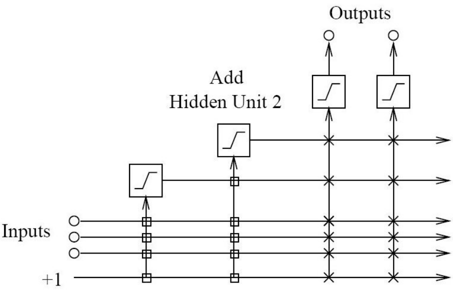
Goal: Minimial effective architecture, easy training



- Train input weights first
- Once trained, freeze the input weights
- Apply perceptron learning rules to train the output weights
- In the figure, boxed connections are frozen, x connections are trained iteratively

Cascade Correlation NNs

Goal: Minimial effective architecture, easy training



- Train input weights first
- Once trained, freeze the input weights
- Apply perceptron learning rules to train the output weights
- In the figure, boxed connections are frozen, x connections are trained iteratively
- We always deal with only a single layer of modifiable weights

Neural Network Construction

Evolutionary approach

- Optimise the weights and/or the architecture using a genetic algorithm

Neural Network Construction

Evolutionary approach

- Optimise the weights and/or the architecture using a genetic algorithm
  - Evolutionary algorithms were successfully used to “evolve” NN architectures
  - If you can represent it, you can evolve it
  - Probably the best “growing” approach

Neural Network Construction

Evolutionary approach

- Optimise the weights and/or the architecture using a genetic algorithm
  - Evolutionary algorithms were successfully used to “evolve” NN architectures
  - If you can represent it, you can evolve it
  - Probably the best “growing” approach
  - Start with a perceptron-like architecture: inputs + outputs

Neural Network Construction

Evolutionary approach

- Optimise the weights and/or the architecture using a genetic algorithm
  - Evolutionary algorithms were successfully used to “evolve” NN architectures
  - If you can represent it, you can evolve it
  - Probably the best “growing” approach
  - Start with a perceptron-like architecture: inputs + outputs
  - Allow the algorithm to establish new neurons and connections

Neural Network Construction

Evolutionary approach

- Optimise the weights and/or the architecture using a genetic algorithm
  - Evolutionary algorithms were successfully used to “evolve” NN architectures
  - If you can represent it, you can evolve it
  - Probably the best “growing” approach
  - Start with a perceptron-like architecture: inputs + outputs
  - Allow the algorithm to establish new neurons and connections
  - Evolving NNs is a slow process

Neural Network Construction

Evolutionary approach

- Optimise the weights and/or the architecture using a genetic algorithm
  - Evolutionary algorithms were successfully used to “evolve” NN architectures
  - If you can represent it, you can evolve it
  - Probably the best “growing” approach
  - Start with a perceptron-like architecture: inputs + outputs
  - Allow the algorithm to establish new neurons and connections
  - Evolving NNs is a slow process

Evolutionary pruning

- Make different architectures compete for survival

Neural Network Construction

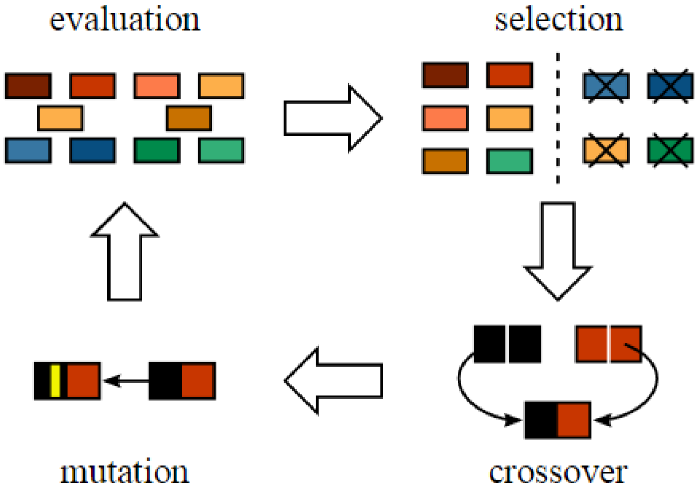
Evolutionary approach

- Optimise the weights and/or the architecture using a genetic algorithm
  - Evolutionary algorithms were successfully used to “evolve” NN architectures
  - If you can represent it, you can evolve it
  - Probably the best “growing” approach
  - Start with a perceptron-like architecture: inputs + outputs
  - Allow the algorithm to establish new neurons and connections
  - Evolving NNs is a slow process

Evolutionary pruning

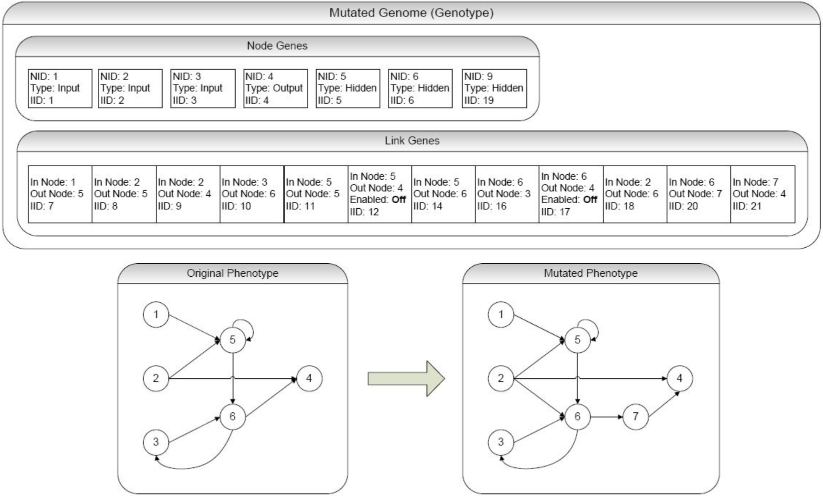
- Make different architectures compete for survival
- Assign higher fitness to smaller architectures

Genetic Algorithms in a Nutshell



Neural Network Construction

NEAT: NeuroEvolution of Augmenting Topologies



Neural Network Construction

- NEAT uses direct encoding: every node/connection is explicitly stored in the representation

Neural Network Construction

- NEAT uses direct encoding: every node/connection is explicitly stored in the representation
- Indirect encoding:
  - Define high-level primitves (layers, neuron types, activation functions...)

Neural Network Construction

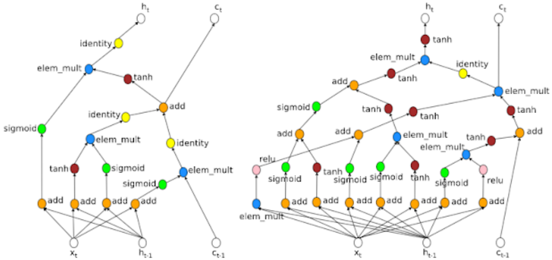
- NEAT uses direct encoding: every node/connection is explicitly stored in the representation
- Indirect encoding:
  - Define high-level primitves (layers, neuron types, activation functions...)
  - Representation is a graph of primitives

Neural Network Construction

- NEAT uses direct encoding: every node/connection is explicitly stored in the representation
- Indirect encoding:
  - Define high-level primitves (layers, neuron types, activation functions...)
  - Representation is a graph of primitives
  - Allows to re-use the primitives/subgraphs iteratively/recursively

Neural Network Construction

- NEAT uses direct encoding: every node/connection is explicitly stored in the representation
- Indirect encoding:
  - Define high-level primitves (layers, neuron types, activation functions...)
  - Representation is a graph of primitives
  - Allows to re-use the primitives/subgraphs iteratively/recursively
  - The research is ongoing!



Neural Network Pruning

To grow or to prune?

Applying Occam's Razor

- Start with an oversized architecture, remove unnecessary parameters

Neural Network Pruning

To grow or to prune?

Applying Occam's Razor

- Start with an oversized architecture, remove unnecessary parameters
  - Weights
  - Hidden units
  - Input units
  - Need a way of quantifying relevance of each parameter

Neural Network Pruning

To grow or to prune?

Applying Occam's Razor

- Start with an oversized architecture, remove unnecessary parameters
  - Weights
  - Hidden units
  - Input units
  - Need a way of quantifying relevance of each parameter
- Large architectures have large functional flexibility => a lot of potential for a good fit

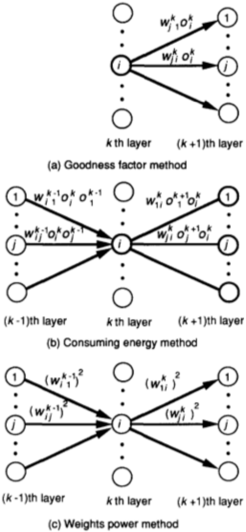
Neural Network Pruning

To grow or to prune?

Applying Occam's Razor

- Start with an oversized architecture, remove unnecessary parameters
    - Weights
    - Hidden units
    - Input units
    - Need a way of quantifying relevance of each parameter
- Large architectures have large functional flexibility => a lot of potential for a good fit
- And a lot of potential for an over-fit?..

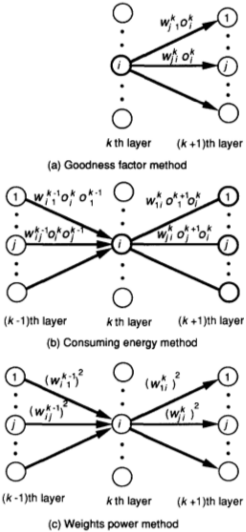
Neural Network Pruning



Intuitive pruning

- Determine the “active” neurons, remove inactive ones

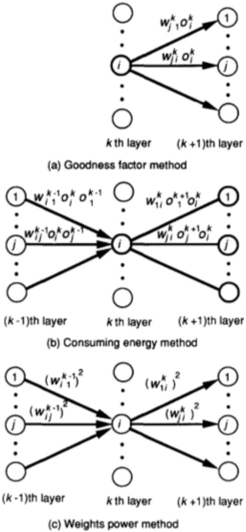
Neural Network Pruning



Intuitive pruning

- Determine the “active” neurons, remove inactive ones
  - “An important unit is the one that fires frequently and has strong connections to other units”

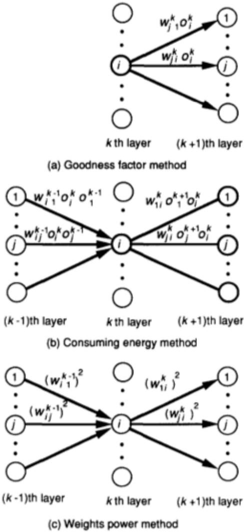
Neural Network Pruning



Intuitive pruning

- Determine the “active” neurons, remove inactive ones
  - “An important unit is the one that fires frequently and has strong connections to other units”
  - $G_i = \sum_P \sum_j (w_{ji} o_i)^2$ - Goodness factor
  - $E_i = \sum_P \sum_j (w_{ji}^l o_i^l o_j^{l+1})$ - Consuming energy

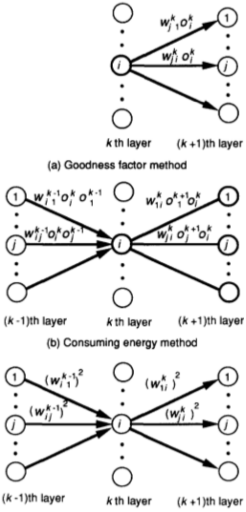
Neural Network Pruning



Intuitive pruning

- Determine the “active” neurons, remove inactive ones
  - “An important unit is the one that fires frequently and has strong connections to other units”
  - $G_i = \sum_P \sum_j (w_{ji} o_i)^2$ - Goodness factor
  - $E_i = \sum_P \sum_j (w_{ji}^l o_i^l o_j^{l+1})$ - Consuming energy
- Units that output 0 more often than 1 are considered irrelevant - is it fair?

Neural Network Pruning



Intuitive pruning

- Determine the “active” neurons, remove inactive ones
  - “An important unit is the one that fires frequently and has strong connections to other units”
  - $G_i = \sum_P \sum_j (w_{ji} o_i)^2$ - Goodness factor
  - $E_i = \sum_P \sum_j (w_{ji}^l o_i^l o_j^{l+1})$ - Consuming energy
- Units that output 0 more often than 1 are considered irrelevant - is it fair?
- Weight magnitude pruning: remove small weights

Neural Network Pruning

Information Matrix pruning

- Fisher information: a way of measuring the amount of information that an observable random variable X carries about an unknown parameter θ upon which the probability of X depends.
  - $I = \frac{1}{P}\sum_{p=1}^{P} \frac{\partial f_{NN}}{\partial w}(\frac{\partial f_{NN}}{\partial w})^T$ - approx. information matrix
  - Calculates the covariance of the weights
  - Captures curvature, just like the Hessian
  - May be time- (and memory-) consuming to compute
  - Prune the weights that bear the least information

Neural Network Pruning

Information Matrix pruning

- Fisher information: a way of measuring the amount of information that an observable random variable X carries about an unknown parameter θ upon which the probability of X depends.
  - $I = \frac{1}{P}\sum_{p=1}^{P} \frac{\partial f_{NN}}{\partial w}\left(\frac{\partial f_{NN}}{\partial w}\right)^T$ - approx. information matrix
  - Calculates the covariance of the weights
  - Captures curvature, just like the Hessian
  - May be time- (and memory-) consuming to compute
  - Prune the weights that bear the least information
- Principal Component Analysis (PCA): prune parameters (weights) that do not account for data variance

Neural Network Pruning

Information Matrix pruning

- Fisher information: a way of measuring the amount of information that an observable random variable X carries about an unknown parameter θ upon which the probability of X depends.
    - $I = \frac{1}{P}\sum_{p=1}^{P} \frac{\partial f_{NN}}{\partial w}(\frac{\partial f_{NN}}{\partial w})^T$ - approx. information matrix
    - Calculates the covariance of the weights
    - Captures curvature, just like the Hessian
    - May be time- (and memory-) consuming to compute
    - Prune the weights that bear the least information
- Principal Component Analysis (PCA): prune parameters (weights) that do not account for data variance
- All of these techniques do not scale very well to large NNs

Neural Network Pruning

Hypothesis Testing

- Use statistical tests to calculate the significance of weights/hidden units
  - Null hypothesis: a subset of weights is equal to zero
  - If weights associated with a neuron are not statistically different from zero, prune the neuron

Neural Network Pruning

Hypothesis Testing

- Use statistical tests to calculate the significance of weights/hidden units
  - Null hypothesis: a subset of weights is equal to zero
  - If weights associated with a neuron are not statistically different from zero, prune the neuron
- Input pruning: Inject a noisy input
  - If the statistical significance of an original parameter is not higher than that of random noise, prune the parameter

Neural Network Pruning

Hypothesis Testing

- Use statistical tests to calculate the significance of weights/hidden units
  - Null hypothesis: a subset of weights is equal to zero
  - If weights associated with a neuron are not statistically different from zero, prune the neuron
- Input pruning: Inject a noisy input
  - If the statistical significance of an original parameter is not higher than that of random noise, prune the parameter
- Assume that weights are $\approx$ normally distributed
  - Remove the weights that are in the distribution tails

Neural Network Pruning

Sensitivity analysis pruning

- Saliency: the influence small perturbations to a parameter have on the approximated error/output function
- Prune parameters with low saliency

Neural Network Pruning

Sensitivity analysis pruning

- Saliency: the influence small perturbations to a parameter have on the approximated error/output function
- Prune parameters with low saliency
- Optimal Brain Damage (OBD), introduced by Yann LeCun:
  1. Choose a reasonable NN architecture
  2. Train until a reasonable solution is obtained
  3. Compute second order derivatives h_{kk} for each parameter (diagonal of the Hessian matrix)
  4. Compute the saliencies for each parameter: $s_k = h_{kk} w^2 / 2$
  5. Sort parameters by saliency and delete low-saliency ones
  6. Go back to step 2

Neural Network Pruning

Sensitivity analysis pruning

- Saliency: the influence small perturbations to a parameter have on the approximated error/output function
- Prune parameters with low saliency
- Optimal Brain Damage (OBD), introduced by Yann LeCun:
    1. Choose a reasonable NN architecture
    2. Train until a reasonable solution is obtained
    3. Compute second order derivatives h_{kk} for each parameter (diagonal of the Hessian matrix)
    4. Compute the saliencies for each parameter: $s_k = h_{kk} w^2/2$
    5. Sort parameters by saliency and delete low-saliency ones
    6. Go back to step 2
- Optimal Brain Surgeon (OBS) - adjust weights
- Optimal Cell Damage (OCD) - prune inputs

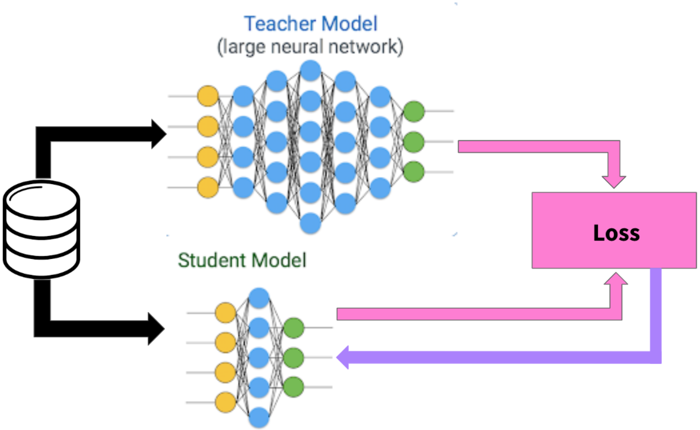
Neural Network Pruning

Sensitivity analysis pruning

- Saliency: the influence small perturbations to a parameter have on the approximated error/output function
- Prune parameters with low saliency
- Optimal Brain Damage (OBD), introduced by Yann LeCun:
  1. Choose a reasonable NN architecture
  2. Train until a reasonable solution is obtained
  3. Compute second order derivatives h_{kk} for each parameter (diagonal of the Hessian matrix)
  4. Compute the saliencies for each parameter: $s_k = h_{kk} w^2 / 2$
  5. Sort parameters by saliency and delete low-saliency ones
  6. Go back to step 2
- Optimal Brain Surgeon (OBS) - adjust weights
- Optimal Cell Damage (OCD) - prune inputs
- Hessians are expensive to calculate

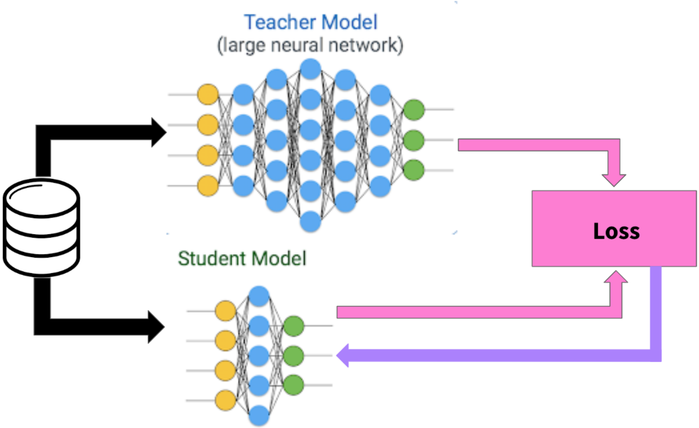
Neural Network Knowledge Distillation

Alternative to pruning: use a large NN to train a smaller NN.



Neural Network Knowledge Distillation

Alternative to pruning: use a large NN to train a smaller NN.



How do we ensure that the students learns from the teacher?

The End

- Questions?
- Next lecture: Deep Learning