Comparison and contrast models same? Different points? No need to know equations, identify key differences, keywords, what is the same? What is difference?

no need to memorize as long as know to use them

https://www.slideshare.net/balveenchugh/neural-networks-12102680

Many models

Feedforward network & recurrent network

Feed-forward ANNs allow signals to travel one way only: from input to output. There are no feedback (loops); i.e., the output of any layer does not affect that same layer. Feed-forward ANNs tend to be straightforward networks that associate inputs with outputs. They are extensively used in pattern recognition. This type of organization is also referred to as bottom-up or top-down.

Feedforward neural networks are ideally suitable for modeling relationships between a set of predictor or input variables and one or more response or output variables. In other words, they are appropriate for any functional mapping problem where we want to know how a number of input variables affect the output variable. The multilayer feedforward neural networks, also called multi-layer perceptrons (MLP), are the most widely studied and used neural network model in practice.

Feedback (or recurrent or interactive) networks can have signals traveling in both directions by introducing loops in the network. Feedback networks are powerful and can get extremely complicated. Computations derived from earlier input are fed back into the network, which gives them a kind of memory. Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found.

As an example of feedback network, I can recall Hopfield's network. The main use of Hopfield's network is as associative memory. An associative memory is a device which accepts an input pattern and generates an output as the stored pattern which is most closely associated with the input. The function of the associate memory is to recall the corresponding stored pattern, and then produce a clear version of the pattern at the output. Hopfield networks are typically used for those problems with binary pattern vectors and the input pattern may be a noisy version of one of the stored patterns. In the Hopfield network, the stored patterns are encoded as the weights of the network.

https://stats.stackexchange.com/questions/2213/whats-the-difference-between-feed-forward-and-recurrent-neural-networks

Deterministic & stochastic

A deterministic model is used in that situation wherein the result is established straightforwardly from a series of conditions. In a situation wherein the cause and effect relationship is stochastically or randomly determined the stochastic model is used.

A deterministic model has no stochastic elements and the entire input and output relation of the model is conclusively determined. A dynamic model and a static model are included in the deterministic model.

A stochastic model has one or more stochastic element. The system having stochastic element is generally not solved analytically and, moreover, there are several cases for which it is difficult to build an intuitive perspective. In the case of simulating a stochastic model, a random number is normally generated by some method or the other to execute trial. Such a simulation is called the Monte Carlo method or Monte Carlo simulation.

A deterministic model assumes that its outcome is certain if the input to the model is fixed. No matter how many times one recalculates, one obtains exactly the same result. It is arguable that the stochastic model is more informative than a deterministic model since the former accounts for the uncertainty due to varying behavioral characteristics. In nature, a deterministic model is one where the model parameters are known or assumed. Deterministic models describe behavior on the basis of some physical law.

https://www.slideshare.net/sohail40/deterministic-vs-stochastic https://www.researchgate.net/post/What_is_the_difference_among_Deterministic_model_Stochastic_model_and_Hybrid_model

In deterministic models, the output of the model is fully determined by the parameter values and the initial conditions.

Stochastic models possess some inherent randomness. The same set of parameter values and initial conditions will lead to an ensemble of different outputs.

https://www4.stat.ncsu.edu/~gross/BIO560%20webpage/slides/Jan102013.pdf

https://www.linkedin.com/pulse/understanding-differences-between-deterministic-stochastic-paul-dalen

Single layer & multilayer

https://www.tandfonline.com/doi/abs/10.1080/10916466.2018.1425717

The main advantage of the multi-layer models is that they can fit training examples better than single-layer models can. This means they can give better generalization, provided there are sufficient training examples. Conversely single-layer models give better relative generalization. By relative generalization we mean testing classification rate divided by training classification rate. Thus a model that correctly classifies 82% of the training examples and 81% of the test examples has better relative generalization (.81/.82) than a model that trains 100% and tests 90% (.90/1.00). The principle factor for determining relative generalization is the inverse of the number of trainable weights set by the algorithm [1, 6]. Therefore single-layer models have better relative generalization than corresponding multi-layer models. This means that if a single-layer model can fit the training examples almost as well as a multi-layer model then the single-layer model can be expected to give better performance on unseen test data. Conversely if a multi-layer model fits the examples significantly better, then the multi-layer model can be expected to give better performance. The deciding factor for determining how much better the multi-layer network must fit the training examples is the number of training examples. If there are few training examples then a greater difference in fit is required before a larger (ie. multi-layer) model is preferred, and conversely if there are many training examples. Note that in practice it is easy to compare generalization of single-layer and multi-layer models; we can simply test the models on unseen training examples. Single-layer models also have faster learning speed and faster execution of the final network because they are smaller than multi-layer models. The bottom line is: if speed is critical or if a multilayer model doesn't fit the training examples a lot better than a single-layer model {or a little better if a large number of training examples were used}, then prefer the single-layer model.

https://www.researchgate.net/publication/312689257_Multi-Layer_Versus_Single-Layer_Neural_Networks_and_an_Application_to_Reading_Hand-Stamped_Characters

https://www.quora.com/Why-do-neural-networks-with-more-layers-perform-better-than-a-single-layer-MLP-with-a-number-of-neurons-that-leads-to-the-same-number-of-parameters

https://stats.stackexchange.com/questions/182734/what-is-the-difference-between-a-neural-network-and-a-deep-neural-network-and-w

Different activation functions

Binary Step Function

It is defined as f(x) = 1, x>=0 else f(x) = 0

The binary function is extremely simple. It can be used while creating a binary classifier. When we simply need to say yes or no for a single class, step function would be the best choice, as it would either activate the neuron or leave it to zero. The function is more theoretical than practical since in most cases we would be classifying the data into multiple classes than just a single class. The step function would not be able to do that. Moreover, the gradient of the step function is zero. This makes the step function not so useful since during back-propagation when the gradients of the activation functions are sent for error calculations to improve and optimize the results. The gradient of the step function reduces it all to zero and improvement of the models doesn’t really happen.

f '(x) = 0, for all x

Linear Function

We saw the problem with the step function, the gradient being zero, it was impossible to update gradient during the backpropagation. Instead of a simple step function, we can try using a linear function. We can define the function as f(x)=ax

We have taken a as 4 in the figure above. Here the activation is proportional to the input. The input x, will be transformed to ax. This can be applied to various neurons and multiple neurons can be activated at the same time. Now, when we have multiple classes, we can choose the one which has the maximum value. But we still have an issue here. Let’s look at the derivative of this function.

f'(x) = a

The derivative of a linear function is constant i.e. it does not depend upon the input value x. This means that every time we do a back propagation, the gradient would be the same. And this is a big problem, we are not really improving the error since the gradient is pretty much the same. And not just that suppose we are trying to perform a complicated task for which we need multiple layers in our network. Now if each layer has a linear transformation, no matter how many layers we have the final output is nothing but a linear transformation of the input. Hence, linear function might be ideal for simple tasks where interpretability is highly desired.

Sigmoid

Sigmoid is a widely used activation function. It is of the form - f(x)=1/(1+e^-x)

This is a smooth function and is continuously differentiable. The biggest advantage that it has over step and linear function is that it is non-linear. This is an incredibly cool feature of the sigmoid function. This essentially means that when I have multiple neurons having sigmoid function as their activation function – the output is non linear as well. The function ranges from 0-1 having an S shape. Let’s take a look at the shape of the curve. The gradient is very high between the values of -3 and 3 but gets much flatter in other regions. How is this of any use? This means that in this range small changes in x would also bring about large changes in the value of Y. So the function essentially tries to push the Y values towards the extremes. This is a very desirable quality when we’re trying to classify the values to a particular class. Let’s take a look at the gradient of the sigmoid function as well.

It’s smooth and is dependent on x. This means that during backpropagation we can easily use this function. The error can be backpropagated and the weights can be accordingly updated. Sigmoids are widely used even today but we still have a problem that we need to address. As we saw previously – the function is pretty flat beyond the +3 and -3 region. This means that once the function falls in that region the gradients become very small. This means that the gradient is approaching to zero and the network is not really learning. Another problem that the sigmoid function suffers is that the values only range from 0 to 1. This means that the sigmoid function is not symmetric around the origin and the values received are all positive. So not all times would we desire the values going to the next neuron to be all of the same sign. This can be addressed by scaling the sigmoid function. That’s exactly what happens in the tanh function. let’s read on.

Tanh

The tanh function is very similar to the sigmoid function. It is actually just a scaled version of the sigmoid function.

tanh(x)=2sigmoid(2x)-1

It can be directly written as – tanh(x)=2/(1+e^(-2x)) -1

Tanh works similar to the sigmoid function but is symmetric over the origin. it ranges from -1 to 1.

It basically solves our problem of the values all being of the same sign. All other properties are the same as that of the sigmoid function. It is continuous and differentiable at all points. The function as you can see is non linear so we can easily backpropagate the errors. Let’s have a look at the gradient of the tan h function.

The gradient of the tanh function is steeper as compared to the sigmoid function. Our choice of using sigmoid or tanh would basically depend on the requirement of gradient in the problem statement. But similar to the sigmoid function we still have the vanishing gradient problem. The graph of the tanh function is flat and the gradients are very low.

ReLU

The ReLU function is the Rectified linear unit. It is the most widely used activation function. It is defined as-
f(x)=max(0,x)

ReLU is the most widely used activation function while designing networks today. First things first, the ReLU function is non linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function. The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time. What does this mean? If you look at the ReLU function if the input is negative it will convert it to zero and the neuron does not get activated. This means that at a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

But ReLU also falls a prey to the gradients moving towards zero. If you look at the negative side of the graph, the gradient is zero, which means for activations in that region, the gradient is zero and the weights are not updated during back propagation. This can create dead neurons which never get activated. When we have a problem, we can always engineer a solution.

https://www.analyticsvidhya.com/blog/2017/10/fundamentals-deep-learning-activation-functions-when-to-use-them/

Learning rules, activation rule- active rule

Learning rule: Perceptron learning algorithm is not gradient-descent and can operate in either sequential or batch training mode, whereas Adaline learning (LMS) algorithm is gradient descent, but can only operate in batch mode.

Architecture: Perceptron uses bipolar or unipolar hardlimiter activation function, Adaline uses linear activation function.

What Adaline and the Perceptron have in common

- they are classifiers for binary classification
- both have a linear decision boundary
- both can learn iteratively, ~~sample by sample~~ (the Perceptron naturally, and Adaline via stochastic gradient descent)
- ~~both use a threshold function~~

The differences between the Perceptron and Adaline

- the Perceptron uses the class labels to learn model coefficients
- Adaline uses continuous predicted values (from the net input) to learn the model coefficients, which is more “powerful” since it tells us by “how much” we were right or wrong

So, in the perceptron, as illustrated below, we simply use the predicted class labels to update the weights, and in Adaline, we use a continuous response

https://sebastianraschka.com/faq/docs/diff-perceptron-adaline-neuralnet.html

There are two differences between the perceptron and the delta rule. The perceptron is based on an output from a step function, whereas the delta rule uses the linear combination of inputs directly. The perceptron is guaranteed to converge to a consistent hypothesis assuming the data is linearly separable. The delta rules converges in the limit but it does not need the condition of linearly separable data.

http://nptel.ac.in/courses/106105078/pdf/Lesson%2038.pdf

https://www.slideshare.net/MohammedBennamoun/artificial-neural-networks-lect3-neural-network-learning-rules

https://towardsdatascience.com/machine-learning-101-supervised-unsupervised-reinforcement-beyond-f18e722069bc

Weighted sum-most output rules almost the same

But activation functions could be quite different, as a result, also there are different learning rules

So that's one type of questions

Another question is ask you to derive, no need for calculator, very simple, some derivation, but you need to look at your calculus. Algebra calculus

Simple differentiation (tut3) or maybe simple integration

Sigmoid activation function unipolar

f(u) = 1/1+exp(-u)

df(u)/du = ((1)'(1+exp(-u)) – (1)(1+exp(-u))')/(1+exp(-u))^2 = (0-((1)'+(exp(-u))')/(1+exp(-u))^2 = -(exp(-u))'/(1+exp(-u))^2 = -(-exp(-u))/ (1+exp(-u))^2 = exp(-u)/(1+exp(-u))^2 = 1/(1+exp(-u)) * exp(-u)/(1+exp(-u)) = 1/(1+exp(-u)) * [1+exp(-u)-1/(1+exp(-u))] = f(u) * [1 – f(u)]

Some question with yes or no, then you should why yes, why no (unknown)

Some involved some calculations, for example, I did it last time, Hopfield network, active function, mean point, new cycle,

https://www.slideshare.net/MohammedBennamoun/artificial-neural-network-lecture-6-associative-memories-discrete-hopfield-networks-62024366

know the rule, know how to use the rule, for example, this rule, how do you use it?

MAXNET, ask you backward, ask you to find w

Nonlinear SVM: optimization, the solution of the discriminant function with the kernel, if you find alpha and b, can find the discriminant function g(x)
MP model

Problems:

Optimization problem, there's noise

Problem local minima, RBF is one solution, get the sigmoid,

Overfitting or underfitting

Generalization – maximum margin

	CI	
NN (BU)	FS(TD)	EC(BU)

NN and FS can do modelling

NN and EC can do computation

FNN – give directly, no way to go back to think yes or no

Classification/regression, could be image, not only label class

supervised

TLU

Perceptron

Adaline

MLP ~~– deep learning~~

RBF

SVM

FLN

ELM – extreme learning machine

RNN – we don't know the output. If we do, we don't have to computer the optimal solution- unsupervised

Sth abstract given, need to compute longer to find solution

Finding solution- optimization

Have to think about it- then finally settle down with an answer solution

TLU

deep learning

HN- Hopfield network

ART – adaptive resonance theory

SOM (self-organizing map)

Perceptron

Perceptron Convergence Theorem

If two sets of data are linearly separable, the perceptron learning algorithm converge to a set of weights and a threshold in a finite steps.

Limitations of Perceptrons

 Only linearly separable data can be classified

 The convergence rate may be low for high-dimensional or large number of data.

ADALINE

 A single adaptive layer of feedforward network of linear elements.

 Trained using a learning algorithm called Delta Rule or Least Mean Squares (LMS) Algorithm.

Training Modes

Sequential mode: input training sample pairs one by one orderly or randomly.

Batch mode: input training sample pairs in the whole training set at each iteration.

Perceptron learning: either sequential or batch mode.

ADALINE training: batch mode only.

Perceptron vs. Adaline

Architecture: Perceptron uses bipolar or unipolar hardlimiter activation function, Adaline uses linear activation function.

Learning rule: Perceptron learning algorithm is not gradient-descent and can operate in either sequential or batch training mode, whereas Adaline learning (LMS) algorithm is gradient descent, but can only operate in batch mode.

Perceptron & Adaline

Pros

Only deal with linearly separable

Cons

Cannot deal with high dimension cases

Mutilayer perceptron (MLP)

Pros

have universal approximations

Cons

Local minima problem – local oscillation

Generalization problem

no details for finding solutions -> What are the hyperparameters? How many layers? How many hidden neurons?

Deep learning

Pros

High complexity – high redundant – much better than MLP

Could capture a lot of details

Very sophisticated architecture

Can deal with complex model

Low generalization error

Cons

Price to pay is can only do on terminal computer device, not in mobile cellphone

convergence rate has slowed down as more number of weights when layers are added and it is also due to vanishing gradient problem – gradient of loss function become smaller and the error propagate very slowly down the network

no theories behind – how many layers you need

MLP vs Deep learning

Similarities

Both can be multilayer neuron networks (MLP is subset of DNN)

MLP is a feed-forward neuron network and Deep learning can also be a feed-forward neuron network.

No theories behind – how many layers you need, How many hidden neurons?

Differences

MLP	Deep Learning
Generalization problem	Low generalization error
-MLP are always feed-forward, there is no cycle between connections -the multilayer perceptron is a specific feed-forward neural network architecture	-DNN can have loops, have cyclic connections could be feed-forward, recurrent -For nature of the connections, we could have fully connected layers, convolutional layers, recurrence, etc.
A stack of multiple fully-connected layers (so, no convolution layers at all), where the activation functions of the hidden units are often a sigmoid or a tanh. The nodes of the output layer usually have softmax activation functions (for classification) or linear activation functions (for regression).	
typical MLP architectures are not "deep" , i.e., we don't have many hidden layers. You usually have, say, 1 to 5 hidden layers	has many layers (19, 22, 152,...even > 1200, very extreme)

http://boards.4chan.org/pol/thread/170254609/you-cant-tell-me-racemixing-isnt-fetishization-it#p170279456

RBF

Pros

Can avoid local minima problem using reachability function error/ output layer

feedforward networks (both RBF and MLP) with randomly fixed hidden neurons (RHN) have previously been proposed and discussed by other authors in papers and textbooks. These RHN net-works have been shown, both theoretically and experimentally, to be fast and accurate

Cons

Transfer problem to another problem. For example, how to determine the center c_i of RBF. Also, randomly generated hidden neurons may not be good like tossing a coin. They are also fixed and only output weights/layers can be adjusted.

ELM Extreme Learning Machine

Pros

Can avoid local minima problem

incremental “ELM” is a universal approximator.

Cons

Similar to RBF. The hidden layer is not adjustable and it is randomly generated. Could only tune output weights/layer during training.

a single-hidden-layer feedforward neural network with the hidden neuron parameters [i.e., weights and bias for a multilayer percep-tron (MLP) network, centers and width’s in a radial basis function(RBF) network] randomly assigned and only the output weights adjusted during training.

FLN . functional-link neural net-works

Pros

a universal approximator

learning and generalization with a one-hidden-layer feedforward neural network consisting of hetero-geneous and randomly prescribed nodes. Pao et al. showed that the RVFL is fast and accurate

Cons

Similar to RBF. The hidden layer is not adjustable and it is randomly generated. Could only tune output weights/layer during training.

FLN vs ELM

Similarity

the hidden neurons are randomly selected and only the weights of the output layer need to be trained

Differences

the only difference from the “ELM” being that the RVFL allows for direct connections from the input nodes to the output neurons, whereas the “ELM” does not

~~As a special case of their more general functional-link neural net-works (FLN) described in Pao’s popular textbook [9] (cited over 700times in the ISI Web of Science), Pao et al. (e.g., [10]–[15]) proposed the random vector functional-link (RVFL) network where the hidden neurons in an FLN are randomly selected and only the weights of the output layer need to be trained (e.g., with pseudoinverse or gra-dient descent), with the only difference from the “ELM” being that the RVFL allows for direct connections from the input nodes to the output neurons, whereas the “ELM” does not. Igelnik and Pao [12] proved that the RVFL network is a universal approximator. Igelnik, Pao, LeClair, and Shen [14] discussed learning and generalization with a one-hidden-layer feedforward neural network consisting of hetero-geneous and randomly prescribed nodes. Pao et al. showed that the RVFL is fast and accurate, In particular, Lewis and co-workers [21]–[24] demonstrated that the RVFL makes efficient neural controllers. Husmeier [18] and Taylor[19], [20] used the RVFL with the expectation–maximization (EM) al-gorithm for probability–density estimation. In conclusion, feedforward networks (both RBF and MLP) with ran-domly fixed hidden neurons (RHN) have previously been proposed and discussed by other authors in papers and textbooks. These RHN net-works have been shown, both theoretically and experimentally, to be fast and accurate. Hence, it is not necessary to introduce a new name “ELM.”~~

~~Comments on The Extreme Learning Machine (PDF Download Available). Available from: https://www.researchgate.net/publication/3304187_Comments_on_The_Extreme_Learning_Machine [accessed May 03 2018].~~

Fuzzy set

Pro

~~Uncertainty comes in many guises and is independent of the kind of FS or methodology one uses to handle it.~~ Two important kinds of uncertainties are linguistic and random. The former is associated with words, a~~nd the fact that words can mean different things to different people,~~ and the latter is associated with unpredictability. Probability theory is used to handle random uncertainty and FSs are used to handle linguistic uncertainty, and sometimes FSs can also be used to handle both kinds of uncertainty, because a fuzzy system may use noisy measurements or operate under random disturbances.

Type 1 fuzzy set

Pros

Not only have T1 FSs been around since 1965, they have also been successfully used in many applications

Cons

However, such FSs have limited capabilities to directly handle data uncertainties, where handle means to model and minimize the effect of. ~~That a T1 FS cannot do this sounds paradoxical because the word fuzzy has the connotation of uncertainty. This paradox has been known for a long time, but it is questionable who first referred to "fuzzy" being paradoxical, e.g.~~

~~Just as variance provides a measure of dispersion about the mean,~~ an FS also needs some measure of dispersion to capture more about linguistic uncertainties than just a single membership function (MF), which is all that is obtained when a T1 FS is used. A T2 FS provides this measure of dispersion.

Type 2 fuzzy set

Pros

Just as variance provides a measure of dispersion about the mean, an FS also needs some measure of dispersion to capture more about linguistic uncertainties than just a single membership function (MF), which is all that is obtained when a T1 FS is used. A T2 FS provides this measure of dispersion.

Cons

Type 1 vs Type 2 fuzzy set

Similarities

If the value of membership function is given by a fuzzy set, it is a type-2 fuzzy set.

Two important kinds of uncertainties are linguistic and random.

FSs are used to handle linguistic uncertainty, and sometimes FSs can also be used to handle both kinds of uncertainty, because a fuzzy system may use noisy measurements or operate under random disturbances.

Uncertainty comes in many guises and is independent of the kind of FS or methodology one uses to handle it. Two important kinds of uncertainties are linguistic and random. The former is associated with words, and the fact that words can mean different things to different people, and the latter is associated with unpredictability. Probability theory is used to handle random uncertainty and FSs are used to handle linguistic uncertainty, and sometimes FSs can also be used to handle both kinds of uncertainty, because a fuzzy system may use noisy measurements or operate under random disturbances.

Differences

T1 FS	T2 FS
~~Just as variance provides a measure of dispersion about the mean,~~ an FS also needs some measure of dispersion to capture more about linguistic uncertainties than just a single membership function (MF), which is all that is obtained when a T1 FS is used.	A T2 FS provides this measure of dispersion.
A T1 FS has a grade of membership that is crisp	a T2 FS has grades of membership that are fuzzy, so it could be called a "fuzzy-fuzzy set." Such a set is useful in circumstances where it is difficult to determine the exact MF for an FS, as in modeling a word by an FS.
An FLS that is described completely in terms of T1 FSs is called a T1 FLS. T1 FLSs are unable to directly handle these uncertainties because they use T1 FSs that are certain.	an FLS that is described using at least one T2 FS is called a T2 FLS. T2 FLSs, on the other hand, are very useful in circumstances where it is difficult to determine an exact MF for an FS; hence, they can be used to handle these uncertainties.
The output processor for a T1 FLS is just a defuzzifier	the output processor of a T2 FLS contains two components: the first maps a T2 FS into a T1 FS and is called a type-reducer [that performs type-reduction (TR)], and the second performs defuzzification on the type-reduced set.

~~What is a T2 FS and how is it different from a T1 FS? A T1 FS has a grade of membership that is crisp, whereas a T2 FS has grades of membership that are fuzzy, so it could be called a "fuzzy-fuzzy set." Such a set is useful in circumstances where it is difficult to determine the exact MF for an FS, as in modeling a word by an FS.~~

~~An FLS that is described completely in terms of T1 FSs is called a T1 FLS, whereas an FLS that is described using at least one T2 FS is called a T2 FLS. T1 FLSs are unable to directly handle these uncertainties because they use T1 FSs that are certain. T2 FLSs, on the other hand, are very useful in circumstances where it is difficult to determine an exact MF for an FS; hence, they can be used to handle these uncertainties.~~

~~The output processor for a T1 FLS is just a defuzzifier; however, the output processor of a T2 FLS contains two components: the first maps a T2 FS into a T1 FS and is called a type-reducer [that performs type-reduction (TR)], and the second performs defuzzification on the type-reduced set.~~

SVM

Instead of minimizing the error, it maximizes the margin between two classes.

Solve the QPP, could be solved as a linearly equation, but generally you cannot

Pros

Easy computation

Boost generalization problem

Lots of theories

LSSVM

Linear equation

Pros

The least squares SVM solution on the other hand can be found with low computational cost and is free of many local minima, being the solution to a convex optimization problem. For two-spiral classification problems the method gives good results over a wide parameter range of σ and γ values.

For a complicated two-spiral classification problem it is illustrated that a least squares SVM with RBF kernel is readily found with excellent generalization performance and low computational cost.

SVM vs LSSVM

the support values for a linearly separable problem of two classes in a two dimensional space. A linear SVM has been taken with $\gamma = 1$. Clearly, points located close and far from the decision line have the largest support values. This is different from SVM's based on inequality constraints, where only points that are near the decision line have nonzero support values.

For SVM, it is to solve the QPP.

For LSSVM, the classifier is found by solving the linear set of equations instead of quadratic programming.

Similarity

Instead of minimizing the error, it maximizes the margin between two classes.

Mercer's condition is applied

Difference

SVM	LSSVM
While in classical SVM's many support values are zero (nonzero values correspond to support vectors)	in least squares SVM's, the support values are proportional to the errors.
To solve the QPP, could be solved as a linearly equation, but generally you cannot	-Linear equation - the classifier is found by solving the linear set of equations instead of quadratic programming -Due to the equality constraints in the formulation, a set of linear equations has to be solved instead of a quadratic programming problem.
For the support values for a linearly separable problem of two classes in a two dimensional space. This is different for SVM's based on inequality constraints, where only points that are near the decision line have nonzero support values.	For the support values for a linearly separable problem of two classes in a two dimensional space. Points located close and far from the decision line have the largest support values.
Not as excellent nor low cost	For a complicated two-spiral classification problem it is illustrated that a least squares SVM with RBF kernel is readily found with excellent generalization performance and low computational cost.

Genetic Algorithm

Discrete binary

Binary coding
real number to binary number

Pros

No derivatives needed

Easy to parallelize

Can escape local minima - GA uses both crossover and mutation operators which makes its population more diverse and thus more immune to be trapped in a local optima

The diversity also helps the algorithm to be faster in reaching the global optima

Works on a wide range of problems

Faster (and lower memory requirements) than searching a very large search space.

Easy, in that if your candidate representation and fitness function are correct, a solution can be found without any explicit analytical work.

Cons

Need much more function evaluations than linearized methods

No guaranteed convergence even to local minimum

Have to discretize parameter space

Long Time taken for convergence

No guarantee of finding global maxima

Fine tuning all the parameters for the GA, like mutation rate, elitism percentage, crossover parameters, fitness normalisation/selection parameters, etc, is often just trial and error.

Randomized – not optimal or even complete.

Can get stuck on local maxima, though crossover can help mitigate this.

It can be hard to work out how best to represent a candidate as a bit string (or otherwise).

Ant Colony Optimization

social plausibility

Advantages of the Ant Colony Optimization

1. Inherent parallelism
2. Positive Feedback accounts for rapid discovery of good solutions
3. Efficient for Traveling Salesman Problem and similar problems
4. Can be used in dynamic applications (adapts to changes such as new distances, etc)

Disadvantages of the Ant Colony Optimization

1. Theoretical analysis is difficult
2. Sequences of random decisions (not independent)
3. Probability distribution changes by iteration

4. Research is experimental rather than theoretical

5. Time to convergence uncertain (but convergence is guaranteed!)

Particle Swarm optimization

Advantages of the basic particle swarm optimization algorithm:

(1)PSO is based on the intelligence. It can be applied into both scientific research and engineering use.

(2)PSO have no overlapping and mutation calculation. The search can be carried out by the speed of the particle. During the development of several generations, only the most optimist particle can transmit information onto the other particles, and the speed of the researching is very fast.

(3)The calculation in PSO is very simple. Compared with the other developing calculations, it occupies the bigger optimization ability and it can be completed easily.

(4) PSO adopts the real number code, and it is decided directly by the solution. The number of the dimension is equal to the constant of the solution.

Disadvantages of the basic particle swarm optimization algorithm:

(1)The method easily suffers from the partial optimism, which causes the less exact at the regulation of its speed and the direction.

(2)The method cannot work out the problems of scattering and optimization.

(3)The method cannot work out the problems of non-coordinate system, such as the solution to the energy field and the moving rules of the particles in the energy field.

GA vs ACO

Similarities

They have common grounds on carrying out Traveling Salesman Problem (TSP):

1) They are all prone to premature convergence, so as to get into local optimum value

2) Algorithms itself in the choice of parameters and its' values cannot be rigorous proved in theory, only through the experimental method to determine the optimum combination.

3) They have not special requirements to searching space, such as derivative, continuity, concavo-convex and other auxiliary information, at the same time their practical ranges are more widespread.

Difference

ACO converges to the optimal path through the accumulation and update of information, but the lack of pheromone in initial stages leads to slower speed of convergence. GA have rapid global searching capability, but the feedback of information in the system has not been utilized, sometimes leading to do-nothing redundant iteration and inefficient solution.

Therefore, if the number of the cities is more than 30, GA's searching capability will gradually decline to a certain extent. When the number of the cities is too big, it cannot obtain the optimal solution in finite iteration, because iterative times are too long and unbearable. Here ACO is better than GA, and it can reach the optimum value. However, when the size of cities is too big, ACO may appear stagnation. Thereby it cannot obtain optimum value.

GA vs PSO

Similarity

Techniques such as PSO and GA are inspired by nature, and have proved themselves to be effective solutions to optimization problems.

Difference

It is observed that, from an evolutionary point of view, the performance of the PSO is better than that of GA. PSO seems to arrive at its final parameter values in fewer generations than the GA.

It is also noticed that, even though the cumulative computational time increases linearly with the number of generations for both PSO and GA, the computational time for GA is low compared to the PSO optimization algorithm. ~~The higher computational time for PSO is due to the communication between the particles after each generation. Additionally, control parameters and objective function are involved in these optimization techniques, and appropriate selection of these is a key point for success.~~

ACO vs PSO

Similarity

PSO and ACO are considered as a global optimization method. In particular, these algorithms manage very well combinatorial and mixed problems. Unlike gradient search methods, they are less susceptible to be trapped in local optima. When GA mimics the natural biological evolution, ACO and PSO draw inspiration from the interactive behavior of social species [1, 2].

~~GA has a relatively old history since the first work of its author John Holland backs to 1962. ACO was proposed by Dorigo et al. in 1990 [1]. It is inspired by the collaborative behavior of the ants [1]. PSO is the most recent one of both. It was developed by Eberhart and Kennedy in 1995 [2]. PSO is inspired by social behavior of bird flocking.~~ Both algorithms (ACO, PSO) are population based stochastic optimization techniques.

PSO and ACO have many similarities with GA. All are initialized with a randomly generated population and updating solution following the fitness values at each generation. However, unlike GA, ACO and PSO have no evolution operators like crossover and mutation. But both have memory, which is important to the algorithms [1, 2]. Compared with GA where individuals are in rivalry, the efficiency of PSO and ACO is in the cooperative behavior of agents (particles or ants).

~~In the following, a version of ACO called Generalize Ant Colony Optimization (GACO) algorithm [3], and the algorithm implemented in Matlab Genetic Algorithms and Direct Search Toolbox [4] are used for comparison. But first, PSO and ACO are briefly presented below.~~

In this paper, the multi-agent algorithms PSO and ACO have proved to be effective to handle hard optimization problems. For this purpose both algorithms are compared together and with Matlab's GA on the optimization of the brushless DC wheel motor with constraints and a single objective. The accuracy of the solution and the computation time are compared. ACO is faster and more accurate. The mixed-integer NSGA-II is employed to find the optimal tuning of ACO and PSO taking into account the accuracy of the solution and the computation time. On this benchmark, ACO and PSO are more computationally effective than Matlab's GA. PSO is the less powerful but research about it is still ongoing. However, compared to GA and ACO, the advantages of PSO are its easy implementation and the few parameters to tune.

https://link.springer.com/content/pdf/10.1007%2F978-3-540-78490-6_1.pdf

Table VII shows a comparison between computing times for both algorithms where we clearly notice that PSO is faster than the ACO algorithm, as it was the case for the first circuit. On the other hand, the zoom shown in Figure 5 shows that in term of robustness, ACO gives a relative better result than the PSO algorithm.

The presented work presents a comparison between two swarm intelligence techniques:

Particle Swarm Optimization and Ant Colony

Optimization. Both techniques were used for the optimal sizing of two analog circuits; a CMOS second generation current conveyor and an operational amplifier. It has been shown that ACO algorithm offers better results in terms of robustness, whereas PSO is faster and requires less algorithm-parameters to handle. Regarding rapidity, PSO has an intrinsic better communication system between its particles when compared to the communication system inside ACO. On the other hand, ACO is relatively more robust due to the fact that it is less dependent on a random technique. Accordingly, the choice between these algorithms will depend on the desiderata of the designer. Our future work will focus on combining benefits of both algorithms in order to propose a hybrid metaheuristics.

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6177367

	Objective function	Constraint
ADALINE	Yes	No
MLP	Yes	No
RBF	Yes	No
SVM	Yes	Yes

Discrete Hopfield network vs Continuous Hopfield Network

30:47

MAXNET

http://teaching.csse.uwa.edu.au/units/CITS4210/lectureNotes/Lect7-UWA.pdf

http://boards.4chan.org/pol/thread/165173134/shills-get-out-while-you-can#p165177049

http://boards.4chan.org/pol/thread/165560647