Machine Learning

Part V

Convolutional Networks, Recurrent Networks, and Ensemble Learning

A Biswas

Syllabus

Introduction

Learning Problems, Well-posed learning problems, Designing learning systems.

Concept Learning

Concept learning task, Inductive hypothesis, Ordering of Hypothesis, General-to-specific ordering of hypotheses. Version spaces, Inductive Bias.

Learning Rule Sets

Sequential Covering Algorithm, First Order Rules, Induction, First Order Resolution, Inverting Resolution.

Regression

Linear regression, Notion of cost function, Logistic Regression, Cost function for logistic regression, application of logistic regression to multi-class classification.

Continued …

Syllabus (continued)

Supervised Learning

Support Vector Machine, Decision tree Learning, Representation, Problems, Decision Tree Learning Algorithm, Attributes, Inductive Bias, Overfitting.

Bayes Theorem, Bayesian learning, Maximum likelihood, Least squared error hypothesis, Gradient Search, Naive Bayes classifier, Bayesian Network, Expectation Maximization Algorithm.

Unsupervised learning

Clustering, K-means clustering, hierarchical clustering.

Instance-Based Learning

k-Nearest Neighbour Algorithm, Radial Basis Function**, Locally Weighted Regression, Locally Weighted Function.**

Neural networks

Linear threshold units, Perceptrons, Multilayer networks and back-propagation, recurrent networks. Probabilistic Machine Learning, Maximum Likelihood Estimation.

Regularization, Preventing Overfitting, Ensemble Learning: Bagging and Boosting, Dimensionality reduction

Convolutional Networks

Earlier networks discussed used fully connected neural net.

However, to classify images, it may not be appropriate as it does not take into account the spatial structure of the images.

Convolutional Networks

For instance, it treats input pixels which are far apart and close together on exactly the same footing. Such concepts of spatial structure must instead be inferred from the training data.

We can try to use a network architecture which is tabula rasa, which tries to take advantage of the spatial structure.

Convolutional Neural Networks

Convolutional neural networksuse a special architecture which is particularly well-adapted to classify images.

Using this architecture makes convolutional networks fast to train. This, in turn, helps us train deep, many-layer networks, which are very good at classifying images.

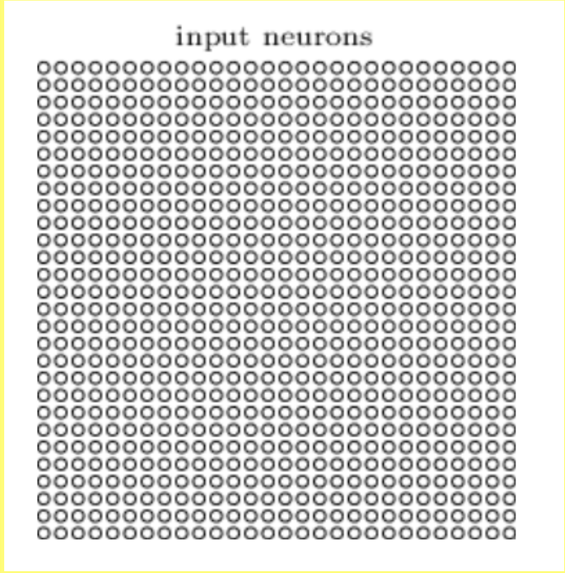
Convolutional Neural Networks

Today, deep convolutional networks or some close variant are used in most neural networks for image recognition.

Convolutional neural networks use three basic ideas:

- local receptive fields,
- shared weights, and
- pooling

Local receptive fields:

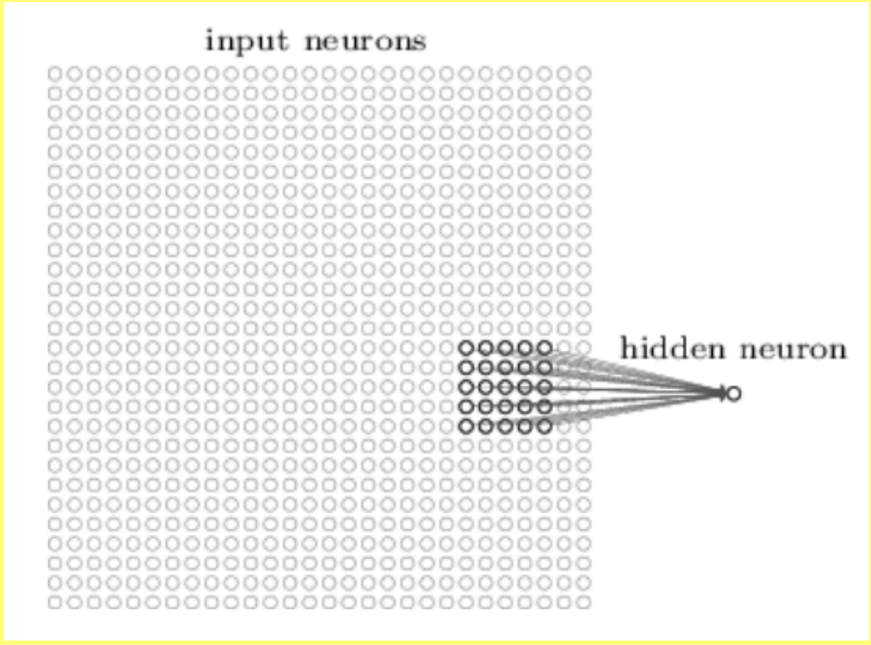


We shall connect the input pixels to a layer of hidden neurons.

But we won't connect every input pixel to every hidden neuron.

Instead, we only make connections in small, localized regions of the input image.

To be more precise, each neuron in the first hidden layer will be connected to a small region of the input neurons, say, for example, a 5 × 5 region, corresponding to 25 input pixels.



That region in the input image is called the local receptive field for the hidden neuron.
It's a little window on the input pixels.

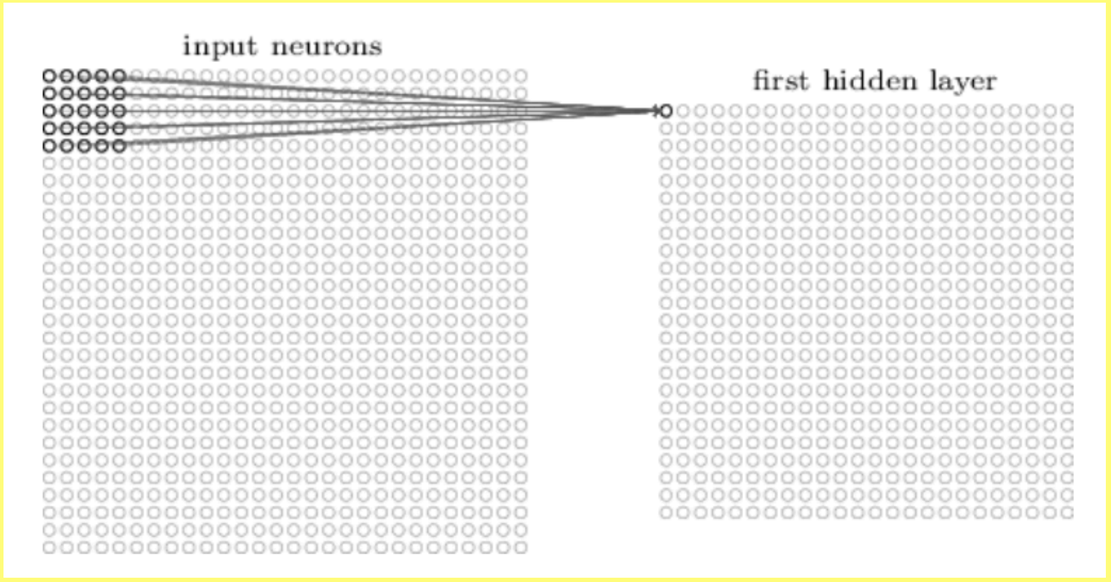
Each connection learns a weight.

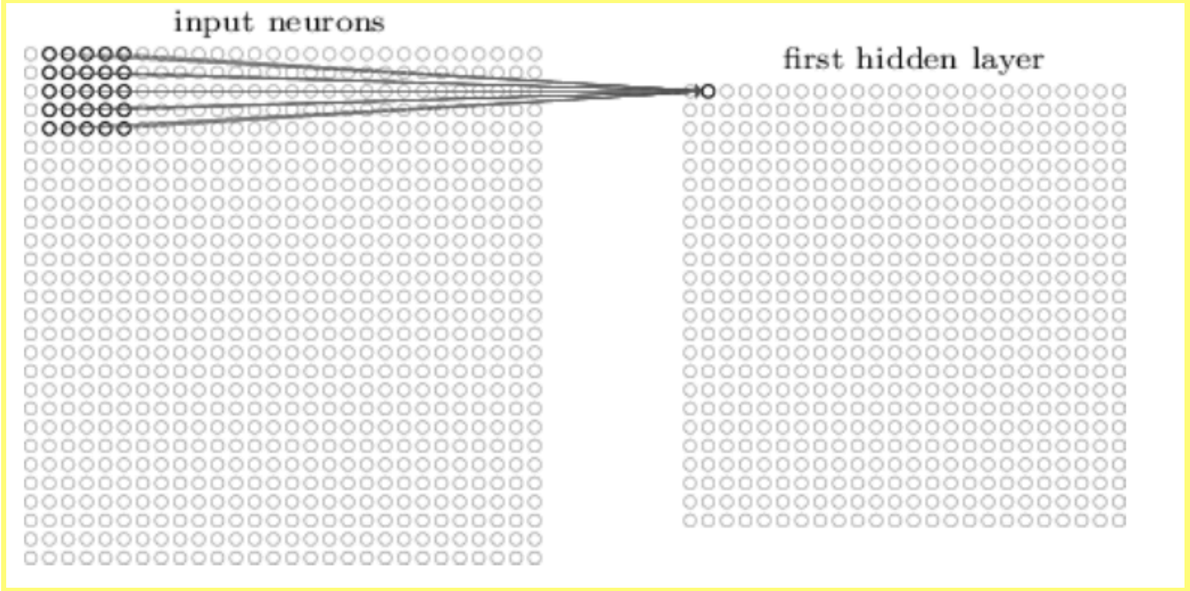
And the hidden neuron learns an overall bias as well.

That is the particular hidden neuron as learning to analyze its particular local receptive field.

We then slide the local receptive field across the entire input image.

For each local receptive field, there is a different hidden neuron in the first hidden layer.





Note that if we have a 28 × 28 input image, and 5 × 5 local receptive fields, then there will be 24 × 24 neurons in the hidden layer.

Because we can only move the local receptive field 23 neurons across (or 23 neurons down), before colliding with the right-hand side (or bottom) of the input image.

Stride length:

Here, the local receptive field has been moved by one pixel at a time. In fact, sometimes a different stride length is used. For instance, the local receptive field moved 2 pixels to the right (or down), (a stride length of 2).

Shared weights and biases:

Each hidden neuron has a bias and 5 × 5 weights connected to its local receptive field.

Also, we use the same weights and bias for each of the 24 × 24 hidden neurons

So, the output of j, k-th hidden neuron:

$$\sigma\left(b + \sum_{l=0}^{4}\sum_{m=0}^{4} w_{l,m} a_{j+l,k+m}\right)$$

b is the shared value for the bias

$w_{l,m}$ is a 5 × 5 array of shared weights

$a_{x,y}$ - input activation at position (x,y)

This means that all the neurons in the first hidden layer detect exactly the same feature, just at different locations in the input image. To see why this makes sense, suppose the weights and bias are such that the hidden neuron can pick out, say, a vertical edge in a particular local receptive field.

That ability is also likely to be useful at other places in the image. And so it is useful to apply the same feature detector everywhere in the image.

Shared weights and biases:

To put it in slightly more abstract terms, convolutional networks are well adapted to the translation invariance of images: move a picture of a cat (say) a little ways, and it's still an image of a cat.

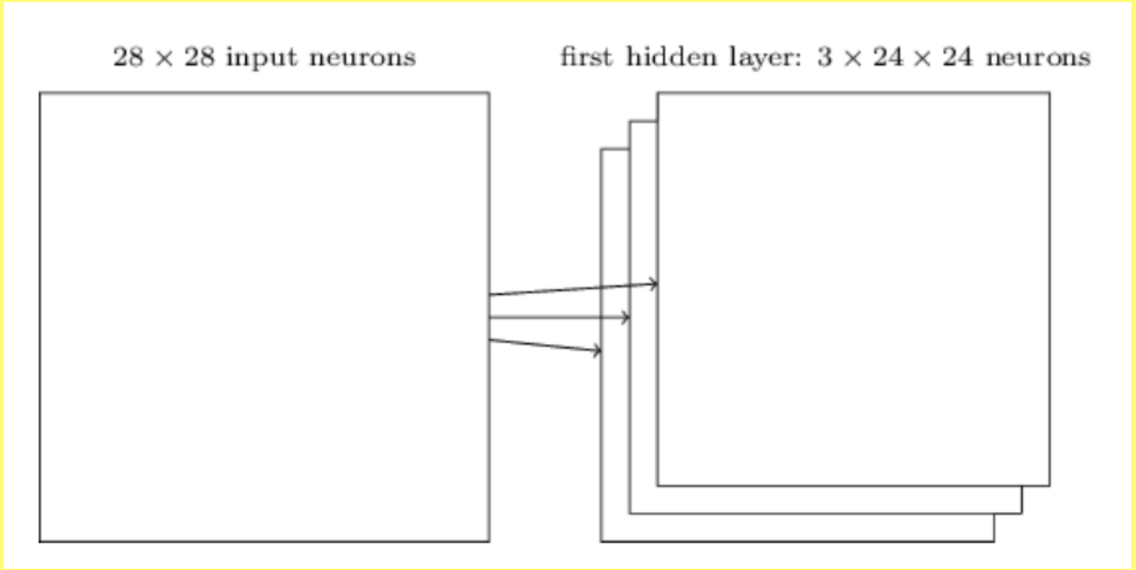
For this reason, we call the map from the input layer to the hidden layer a **feature map**.

We call the weights defining the feature map the **shared weights**. And we call the bias defining the feature map in this way the **shared bias**.

The shared weights and bias are often said to define a *kernel* or *filter*.

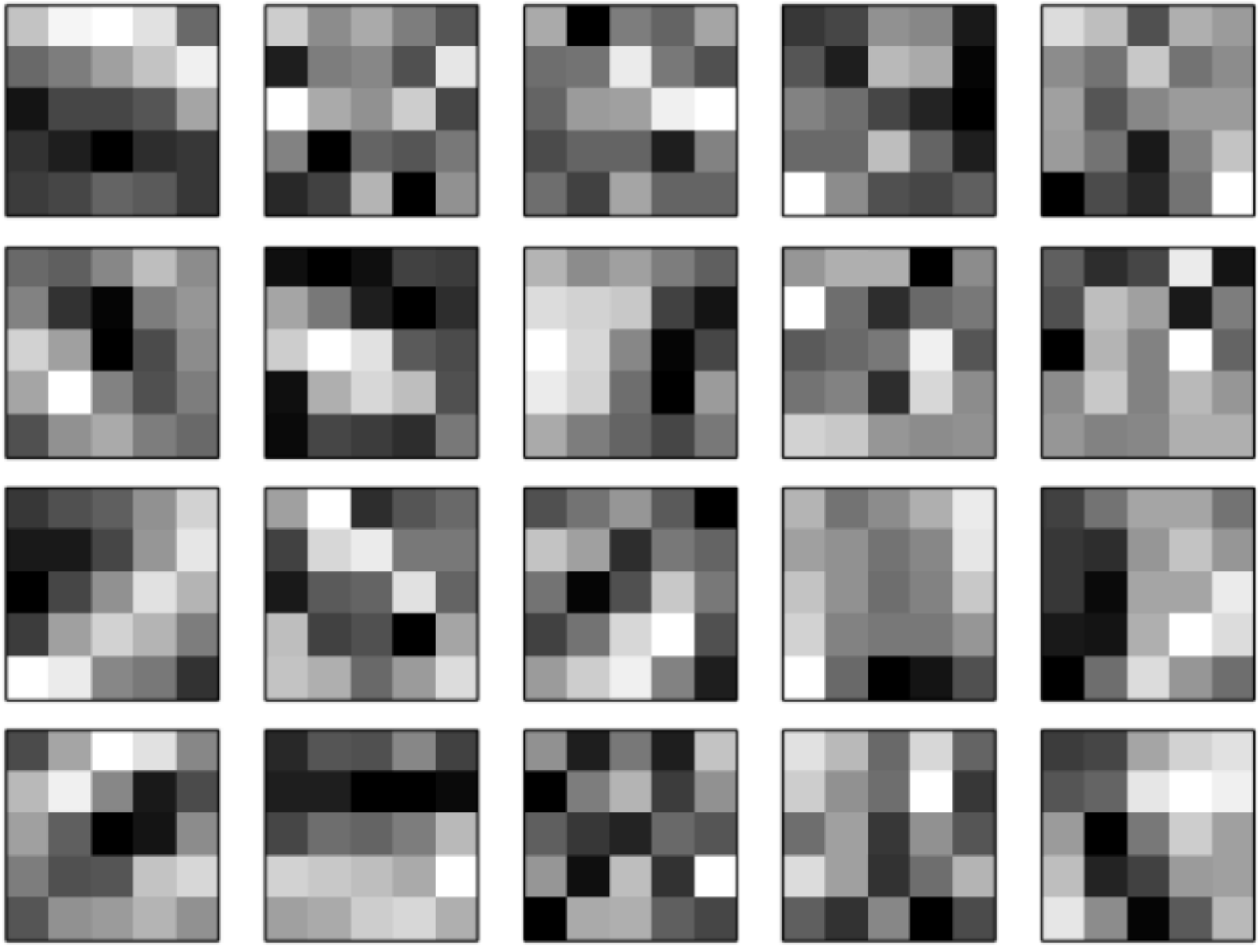
To do image recognition we'll need more than one feature map.

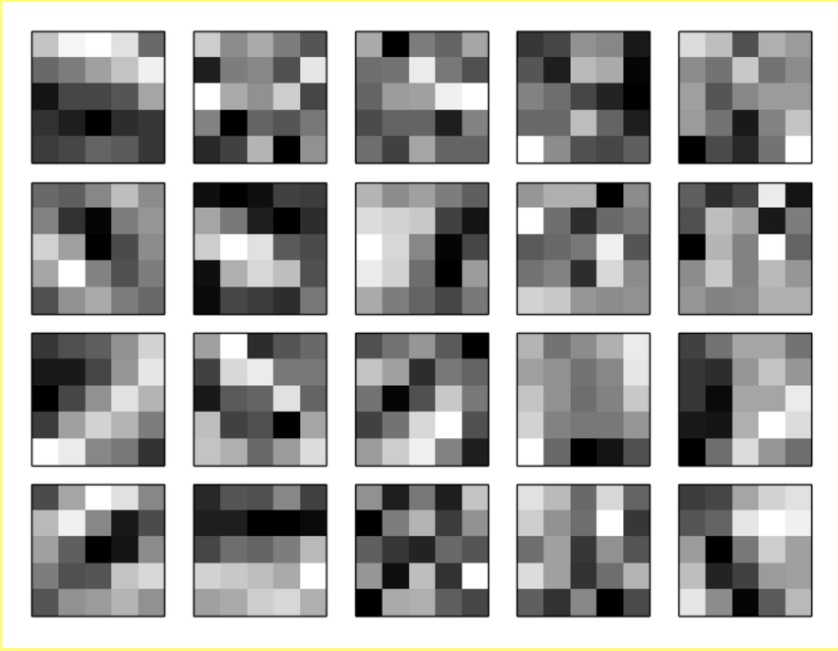
And so a complete convolutional layer consists of several different feature maps.



There are 3 feature maps. Each feature map is defined by a set of 5 × 5 shared weights, and a single shared bias. The result is that the network can detect 3 different kinds of features, with each feature being detectable across the entire image.

LeNet-5, used 6 feature maps, each associated to a 5 × 5 local receptive field, to recognize MNIST digits.





The 20 images correspond to 20 different feature maps (or filters, or kernels). Each map is represented as a 5 × 5 block image, corresponding to the 5 × 5 weights in the local receptive field.

Whiter blocks mean a smaller (typically, more negative) weight, so the feature map responds less to corresponding input pixels. Darker blocks mean a larger weight, so the feature map responds more to the corresponding input pixels.

Very roughly speaking, the images above show the type of features the convolutional layer responds to.

Ref: Visualizing and Understanding Convolutional Networks by Matthew Zeiler and Rob Fergus (2013).

A big advantage of sharing weights and biases is that it greatly reduces the number of parameters involved in a convolutional network.

For each feature map we need $25 = 5 \times 5$ shared weights, plus a single shared bias. So each feature map requires 26 parameters. If we have 20 feature maps that's a total of $20 \times 26 = 520$ parameters defining the convolutional layer.

Suppose we had a fully connected first layer, with $784 = 28 \times 28$ input neurons, and a relatively modest 30 hidden neurons.

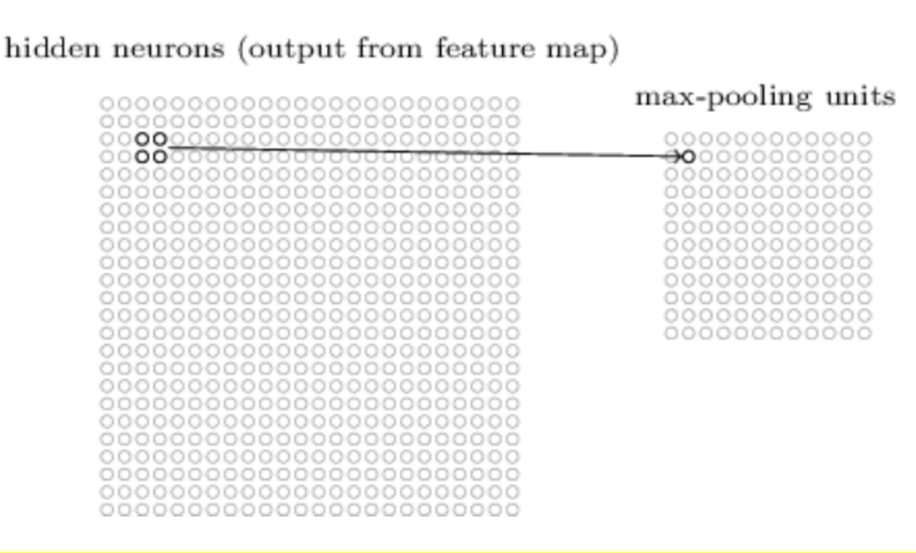
That's a total of 784×30 weights, plus an extra 30 biases, for a total of 23,550 parameters.

In other words, the fully-connected layer would have more than 40 times as many parameters convolutional layer.

Pooling layers:
Convolutional neural networks also contain pooling layers.

Pooling layers are usually used immediately after convolutional layers.

It simplifies the information in the output from the convolutional layer.

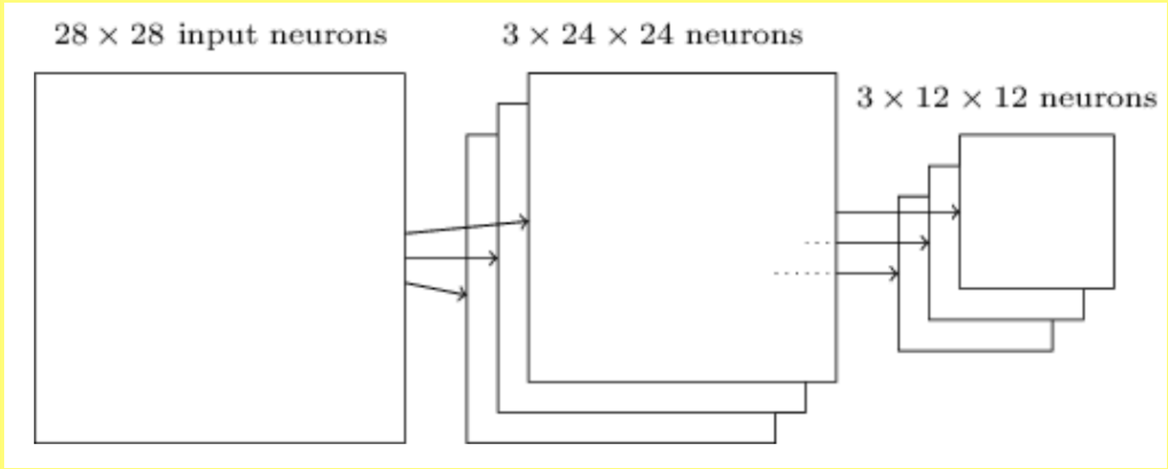


After pooling we have 12 × 12 neurons.

The convolutional layer usually involves more than a single feature map.

We apply max-pooling to each feature map separately.

So if there were three feature maps



Max-pooling is a way for the network to ask whether a given feature is found anywhere in a region of the image.

It then throws away the exact positional information.

The intuition is that once a feature has been found, its exact location isn't as important as its rough location relative to other features.

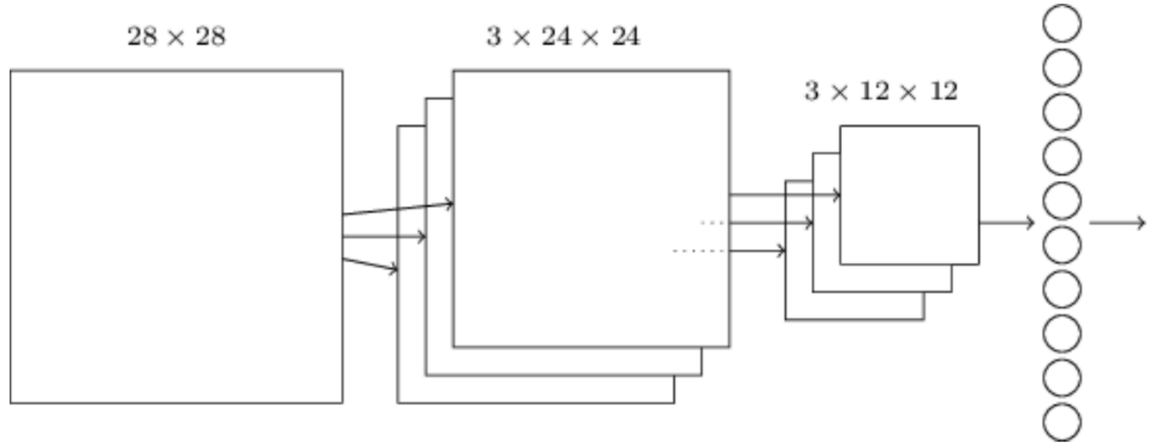
Alternative to max-pooling:

L2 pooling

Instead of max,

the square root of the sum of the squares of the activations in the 2 × 2 region is taken.

Putting it all together



The final layer of connections in the network is a fully-connected layer. That is, this layer connects every neuron from the max-pooled layer to every one of the 10 output neurons.

However, we do need to make a few modifications to the backpropagation procedure as earlier derivation of backpropagation was for networks with fully-connected layers. Fortunately, it's straightforward to modify the derivation for convolutional and max-pooling layers.

Key characteristics of ConvNet:

The patterns they learn are translation-invariant. After learning a certain pattern in the lower-right corner of a picture, a convnet can recognize it anywhere: for example, in the upper-left corner.

A densely connected model would have to learn the pattern anew if it appeared at a new location.

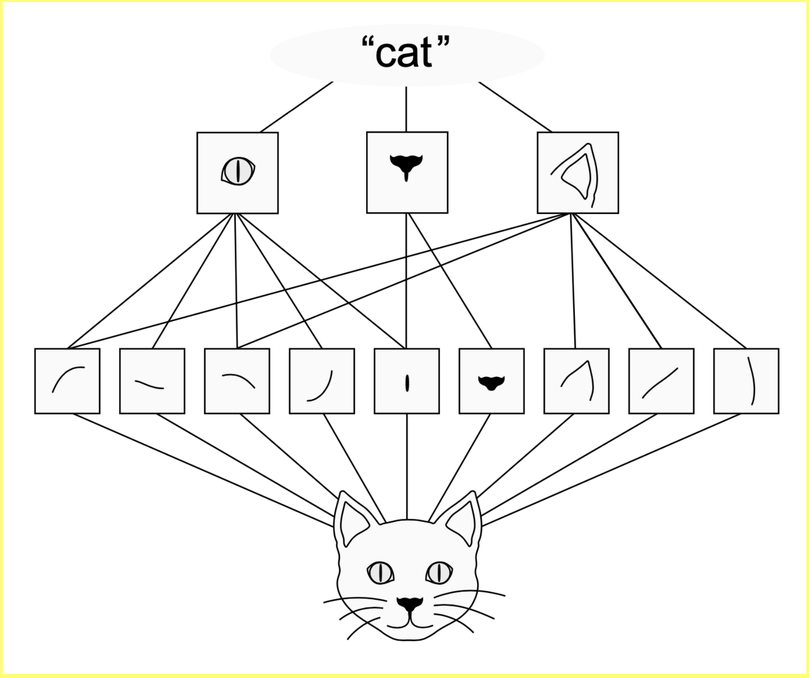
This makes convnets data-efficient when processing images (because the visual world is fundamentally translation-invariant): they need fewer training samples to learn representations that have generalization power.

Key characteristics of ConvNet:

They can learn spatial hierarchies of patterns.

A first convolution layer will learn small local patterns such as edges, a second convolution layer will learn larger patterns made of the features of the first layers, and so on.

This allows convnets to efficiently learn increasingly complex and abstract visual concepts, because the visual world is fundamentally spatially hierarchical.

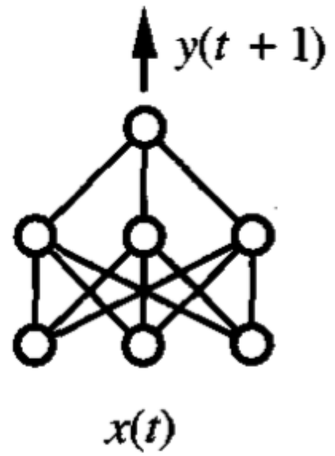


Recurrent Networks

Recurrent networks are artificial neural networks that apply to time series data and that use outputs of network units at time t as the input to other units at time t+1. In this way, they support a form of directed cycles in the network.

To illustrate, consider the time series prediction task of predicting the next day's stock market average y(t+1) based on the current day's economic indicators x (t) .

Given a time series of such data, one obvious approach is to train a feedforward network to predict y(t+1) as its output, based on the input values x(t).



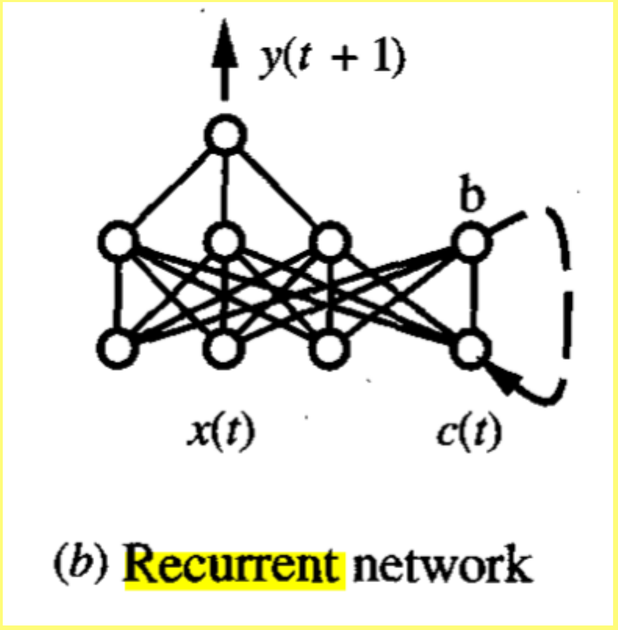
(a) Feedforward network

One limitation of such a network is that the prediction of y(t+1) depends only on x (t) and cannot capture possible dependencies of y(t+1) on earlier values of x.

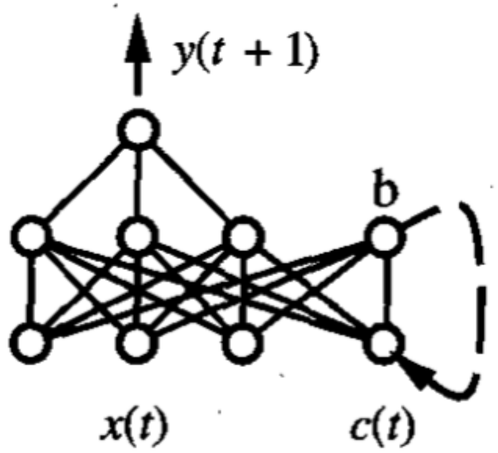
This might be necessary, for example, if tomorrow's stock market average y(t+1) depends on the difference between today's economic indicator values x(t)and yesterday's values x(t-1).

Remedy: by making both x(t) and x(t - 1) inputs to the feedforward network.

However, if we wish the network to consider an arbitrary window of time in the past when predicting y(t +1), then a different solution is required.



(b) Recurrent network



(b) Recurrent network

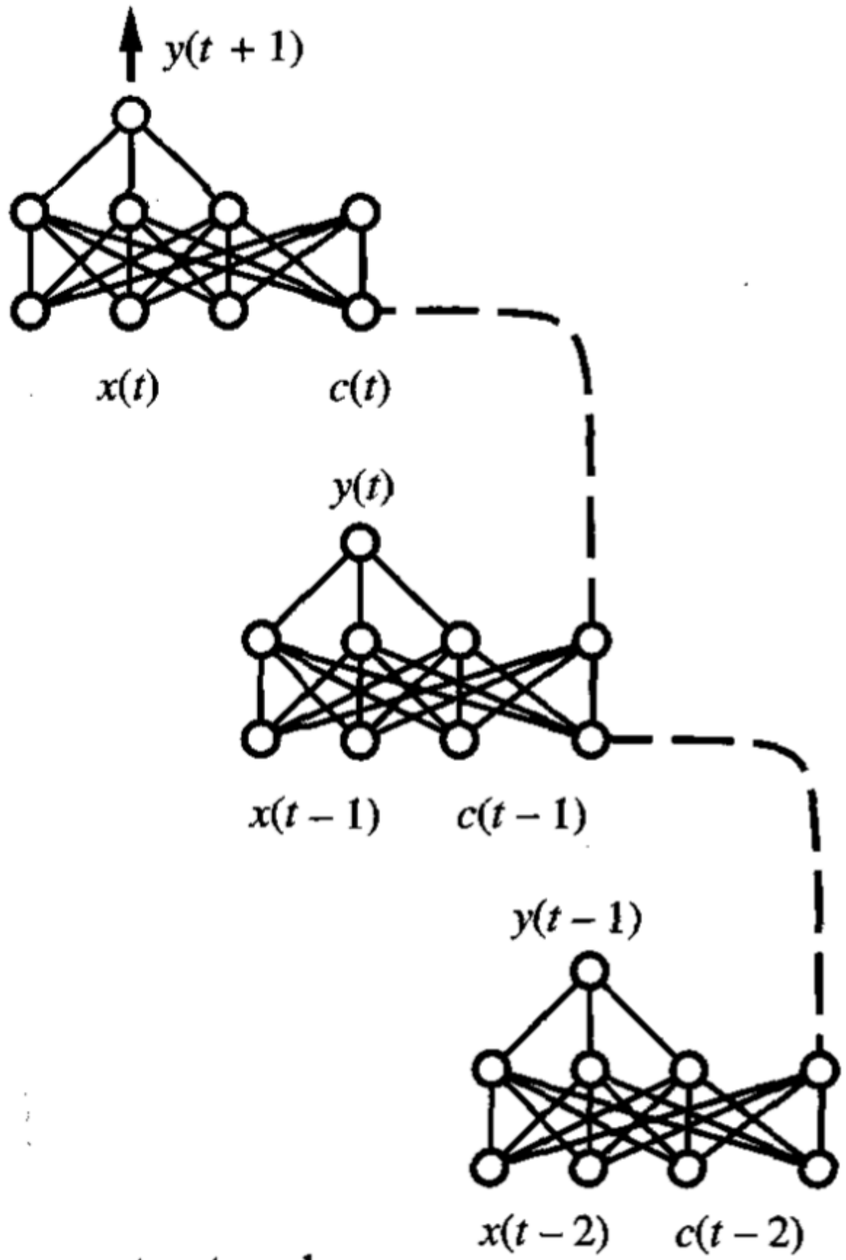
Here, we have added a new unit b to the hidden layer, and new input unit c(t). The value of c(t) is defined as the value of unit b at time t-1; that is, the input value c(t) to the network at one time step is simply copied from the value of unit b on the previous time step.

Notice this implements a recurrence relation, in which b represents information about the history of network inputs. Because b depends on both x(t) and on c(t), it is possible for b to summarize information from earlier values of x that are arbitrarily distant in time.

How to train recurrent network?

Shows the data flow of the recurrent network unfolded in time.

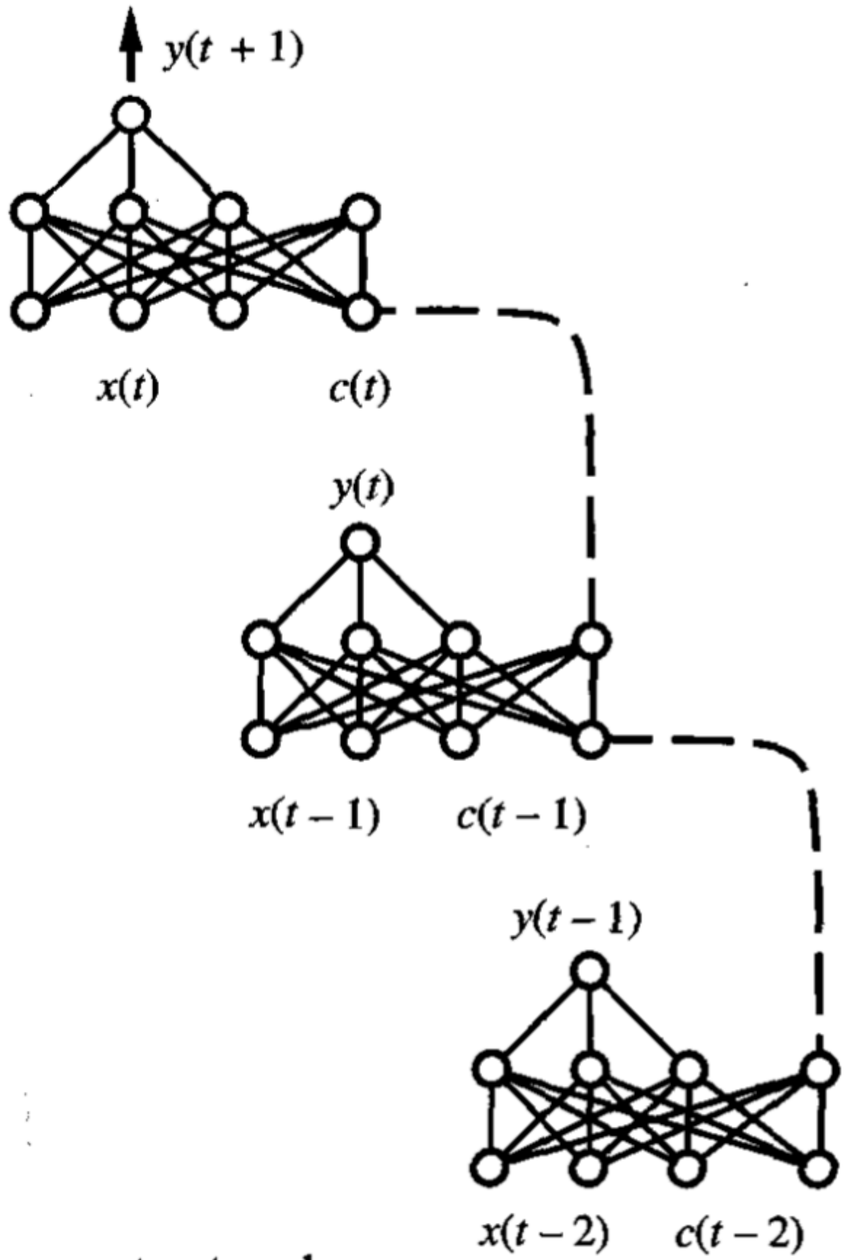
Here we have made several copies of the recurrent network, replacing the feedback loop by connections between the various copies. Notice that this large unfolded network contains no cycles. Therefore, the weights in the unfolded network can be trained directly using BACKPROPAGATION.



(c) Recurrent network unfolded in time

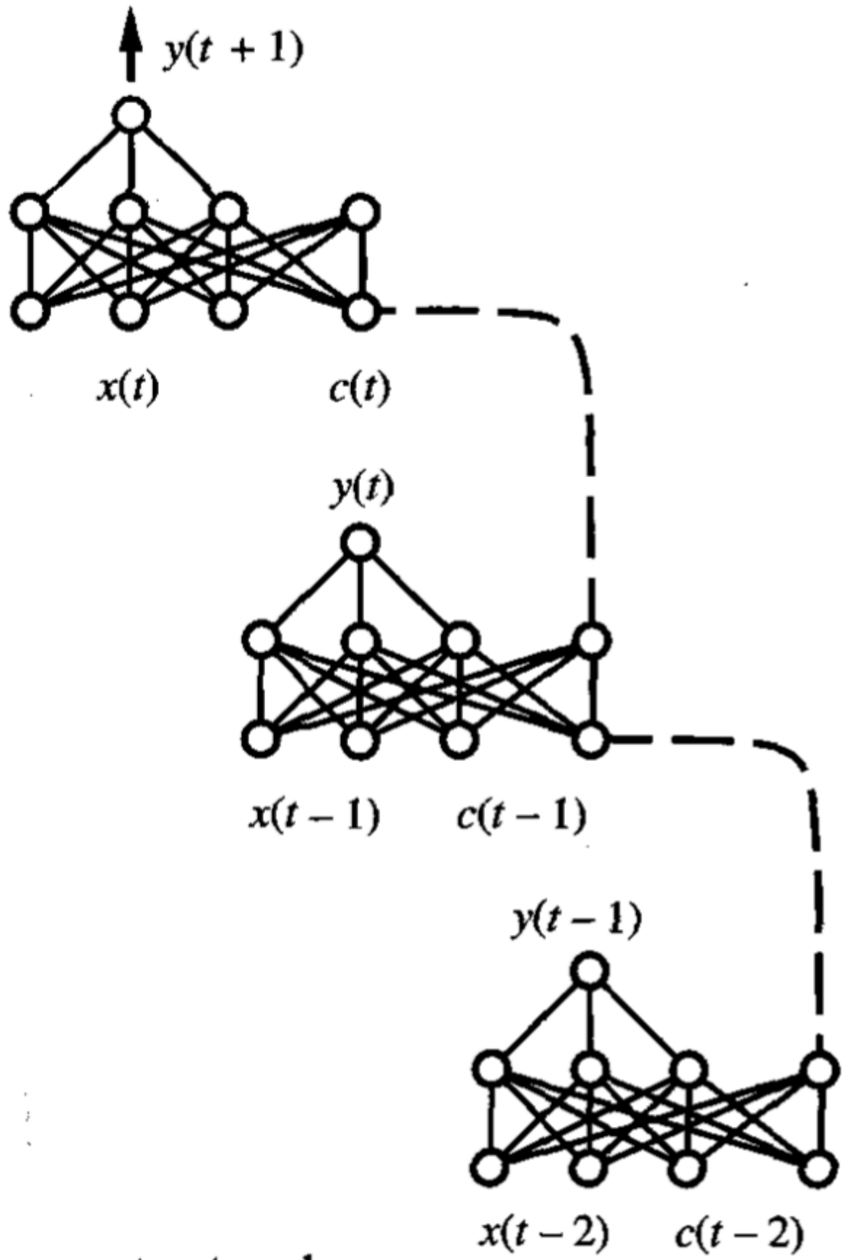
Of course, in practice, we wish to keep only one copy of the recurrent network and one set of weights.
weights in the various copies

Therefore, after training the unfolded network, the final weight w_{ji} in the recurrent network can be taken to be the mean value of the corresponding w_{ji} in different copies.



(c) Recurrent network unfolded in time

In practice, recurrent networks are more difficult to train than networks with no feedback loops and do not generalize as reliably. However, they remain important due to their increased representational power.



(c) Recurrent network unfolded in time

Class Test: 27/4/22, 20 min

Answer all questions

1. What is the difference between a perceptron and a sigmoid neuron? [5]

2. Can you implement a NAND gate using perceptrons? [5]

3. Explain one characteristics of sigmoid function such that it is usable for Neural Networks. [5]

4. Explain the role of hidden layer in a neural network. [5]

Ensemble Learning:

Instead of trying to learn one super-accurate model, focuses on training a large number of low-accuracy models and then combining the predictions given by those weak models to obtain a high-accuracy meta-model.

Low-accuracy models are usually learned by weak learners, that is learning algorithms that cannot learn complex models, and thus are typically fast at the training and at the prediction time.

The most frequently used weak learner is a decision tree learning algorithm in which we often stop splitting the training set after just a few iterations.

The obtained trees are shallow and not particularly accurate.

If the trees are not identical and each tree is at least slightly better than random guessing, then we can obtain high accuracy by combining a large number of such trees.

To obtain the prediction for input x, the predictions of each weak model are combined using some sort of weighted voting.

The specific form of vote weighting depends on the algorithm, but, independently of the algorithm, the idea is the same.

"If the council of weak models predicts that the message is spam, then we assign the label spam to x."

Two most widely used

1. random forest and
2. gradient boosting.

Random Forest:

Two ensemble learning paradigms:

1. bagging and
2. boosting.

Bagging:

1. creating many “copies” of the training data (each copy is slightly different from another) and
2. then apply the weak learner to each copy to obtain multiple weak models and then combine them.

The bagging paradigm is behind the random forest learning algorithm.

"Vanilla" bagging algorithm:

1. Given a training set
2. create B random samples S_b (for each b = 1,…,B)
3. build a decision tree model f_b using each sample S_b as the training set.

To sample S_b for some b, we do the sampling with replacement.

Sampling with replacement:

1. Start with an empty set
2. then pick at random an example from the training set
3. put its exact copy to S_b by keeping the original example in the original training set.
4. keep picking examples at random until the $|S_b|$ = N

After training, there are B decision trees. The prediction for a new example x is obtained as the average of B predictions:

$$y \leftarrow \hat{f}(\mathbf{x}) \stackrel{\text{def}}{=} \frac{1}{B} \sum_{b=1}^{B} f_b(\mathbf{x})$$

for regression

Or by taking the majority vote in the case of classification.

The random forest algorithm is different from the vanilla bagging in just one way.

It uses a modified tree learning algorithm that inspects, at each split in the learning process, a random subset of the features.

Why a random subset of features?

To avoid the correlation of the trees: if one or a few features are very strong predictors for the target, these features will be selected to split examples in many trees.

This would result in many correlated trees in our "forest." Correlated predictors cannot help in improving the accuracy of prediction.

The main reason behind a better performance of model ensembling is that models that are good will likely agree on the same prediction, while bad models will likely disagree on different ones.

Correlation will make bad models more likely to agree, which will hamper the majority vote or the average.

The most important hyperparameters to tune are the number of trees, B, and the size of the random subset of the features to consider at each split.

Random forest is very effective. Why?

By using multiple samples of the original dataset, we reduce the variance of the final model.

Remember that the low variance means low overfitting.

Overfitting happens when our model tries to explain small variations in the dataset because our dataset is just a small sample of the population of all possible examples of the phenomenon we try to model.

If we were unlucky with how our training set was sampled, then it could contain some undesirable (but unavoidable) artifacts: noise, outliers and over- or underrepresented examples.

By creating multiple random samples with replacement of our training set, we reduce the effect of these artifacts.

Gradient Boosting:

First, gradient boosting for regression:

To build a strong regressor

Start with a constant model $f = f_0$:

$$f = f_0(\mathbf{x}) \stackrel{\text{def}}{=} \frac{1}{N}\sum_{i=1}^{N} y_i$$

Then modify labels of each example i = 1, . . . , N in our training set like follows:

$$\hat{y}_i \leftarrow y_i - f(\mathbf{x}_i)$$

$\hat{y}_i$, called the residual, is the new label for example x_i

Now we use the modified training set, with residuals instead of original labels, to build a new decision tree model, f_1.

The boosting model is now:

$$f \stackrel{\text{def}}{=} f_0 + \alpha f_1$$

α is the learning rate, a hyperparameter.

Then we recompute the residuals using and replace the labels in the training data once again, train the new decision tree model f_2, redefine the boosting model as

$$f \stackrel{\text{def}}{=} f_0 + \alpha f_1 + \alpha f_2$$

the process continues until the maximum of M (another hyperparameter) trees are combined.

Intuitively —

By computing the residuals, we find how well (or poorly) the target of each training example is predicted by the current model f.

We then train another tree to fix the errors of the current model (this is why we use residuals instead if real labels) and add this new tree to the existing model with some weight α.

Therefore, each additional tree added to the model partially fixes the errors made by the previous trees until the maximum number of trees are combined.

However, instead of getting the gradient directly, we use its proxy in the form of residuals: they show us how the model has to be adjusted so that the error (the residual) is reduced.

Three principal hyperparameters to tune in gradient boosting are

1. the number of trees,
2. the learning rate, and
3. the depth of trees — all three affect model accuracy.

The depth of trees also aects the speed of training and prediction: the shorter, the faster.

Difference with bagging:

boosting reduces the bias (or underfitting) instead of the variance.

As such, boosting can overfit. However, by tuning the depth and the number of trees, overfitting can be largely avoided.

i) Bagging —

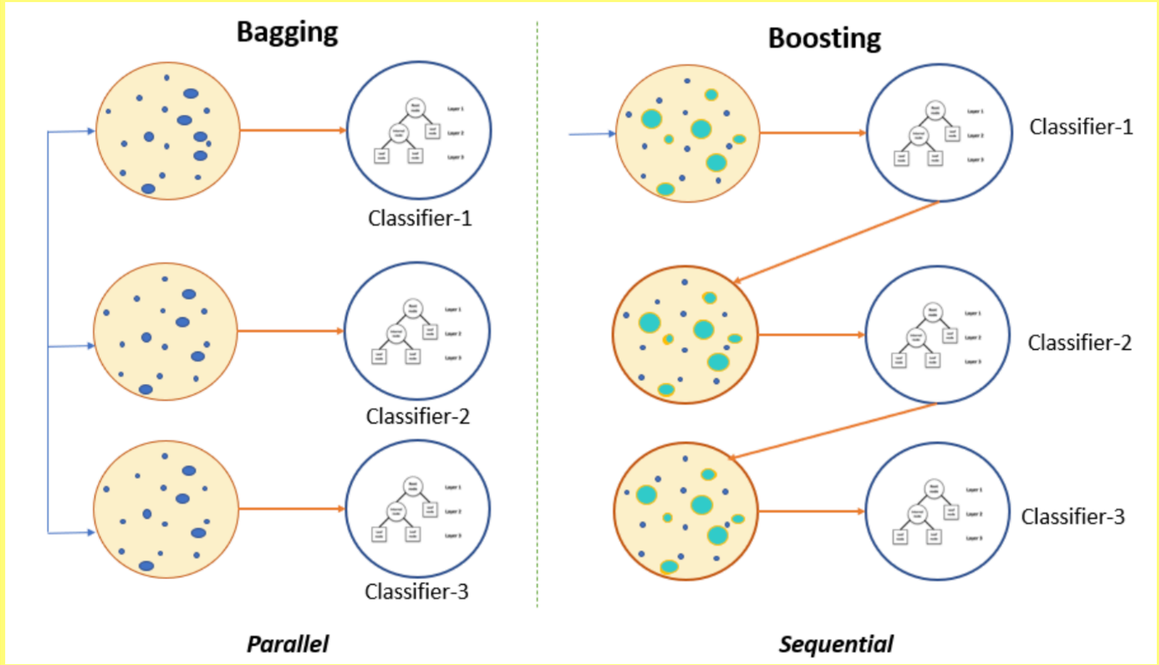
- a number of independent predictors are built by taking samples with replacement.
- The individual outcomes are then combined by average (Regression) or majority voting (Classification) to derive the final prediction.

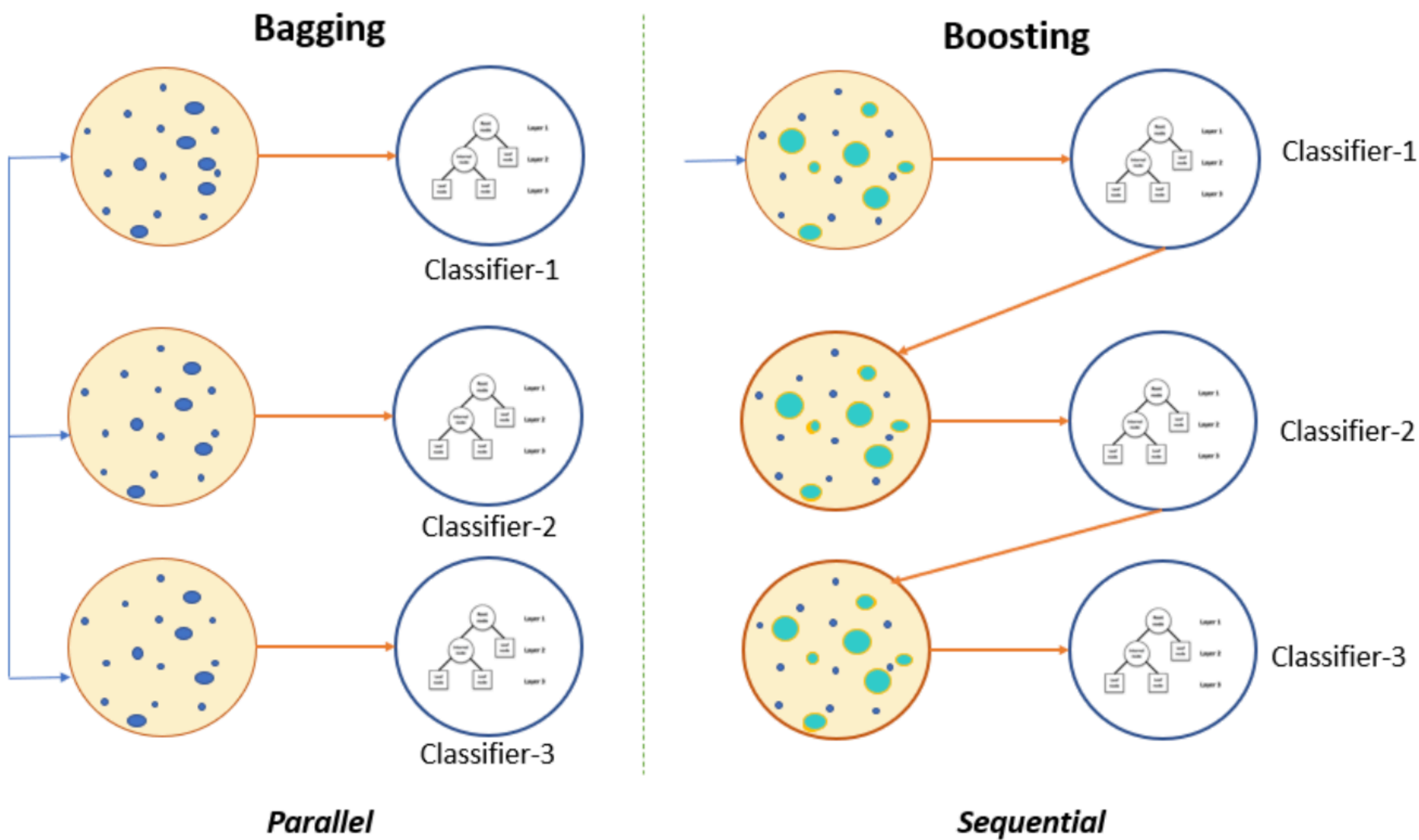
Example: Random Forest.

ii) Boosting —

- the weak learners are converted into strong learners.
- Weak learners are classifiers which always perform slightly better than chance irrespective of the distribution over the training data.
- In Boosting, the predictions are sequential wherein each subsequent predictor learns from the errors of the previous predictors.

- Gradient Boosting Trees (GBT) is a commonly used method in this category.




Bagging
Classifier-1
Classifier-2
Classifier-3
Parallel
Boosting
Classifier-1
Classifier-2
Classifier-3
Sequential

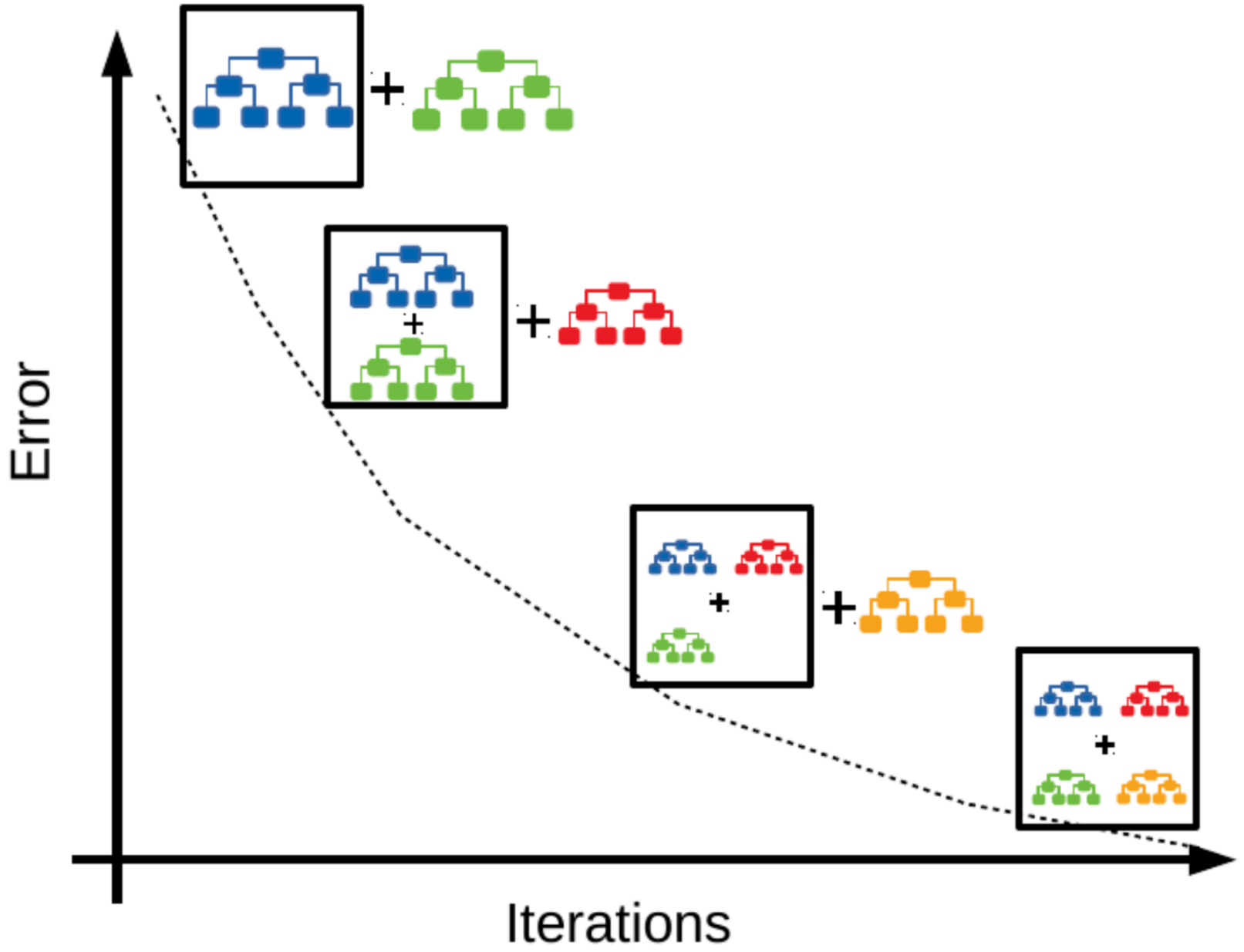
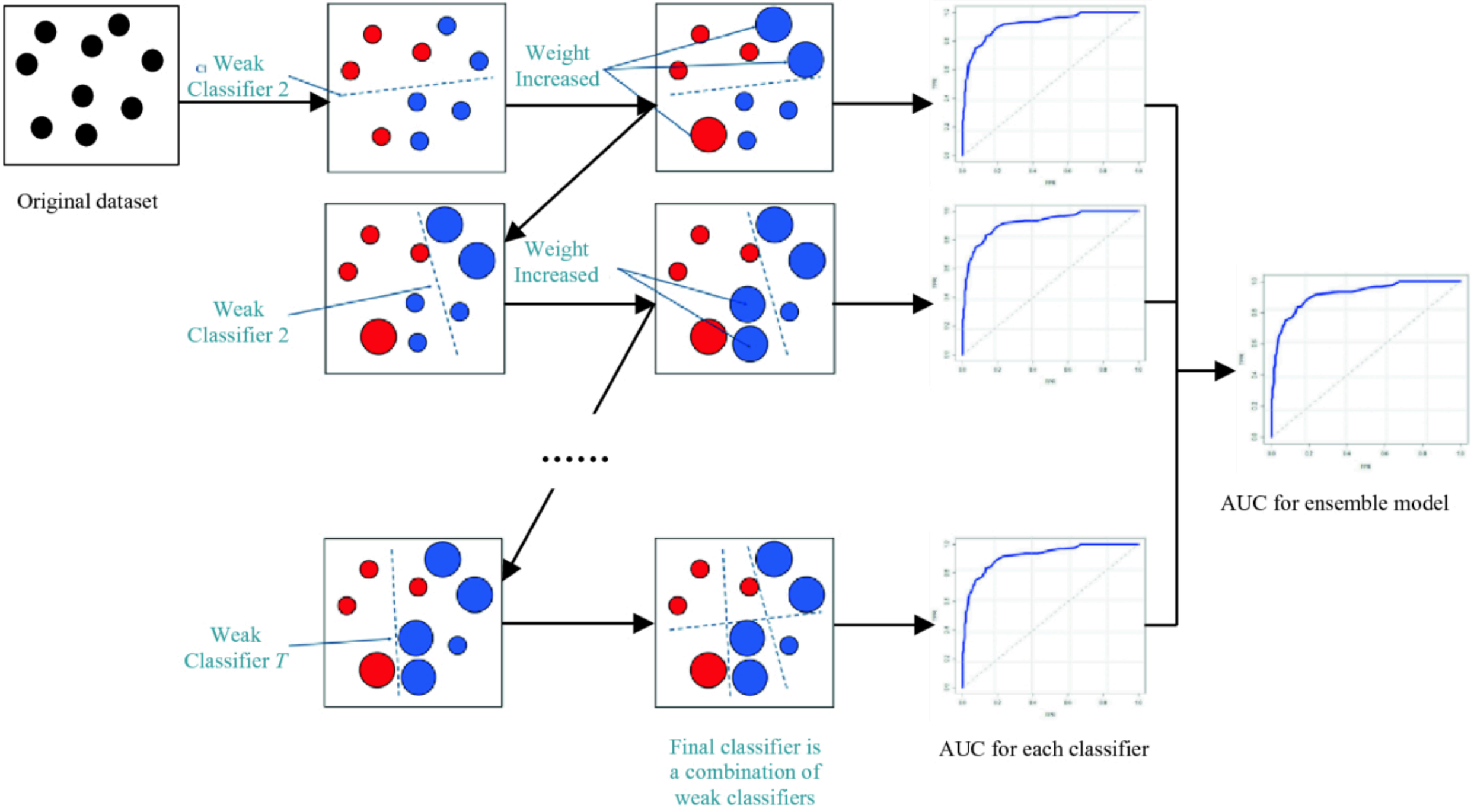


Fig 2. Gradient Boosting Algorithm


Original dataset
Weak Classifier 2
Weight Increased
Weak Classifier 2
Weight Increased
Weak Classifier T
Final classifier is a combination of weak classifiers
AUC for each classifier
AUC for ensemble model

After analyzing the first tree,

- raise the weights of every observation that they find complicated to classify.
- decrease the weights for the ones in which classification is not an issue.

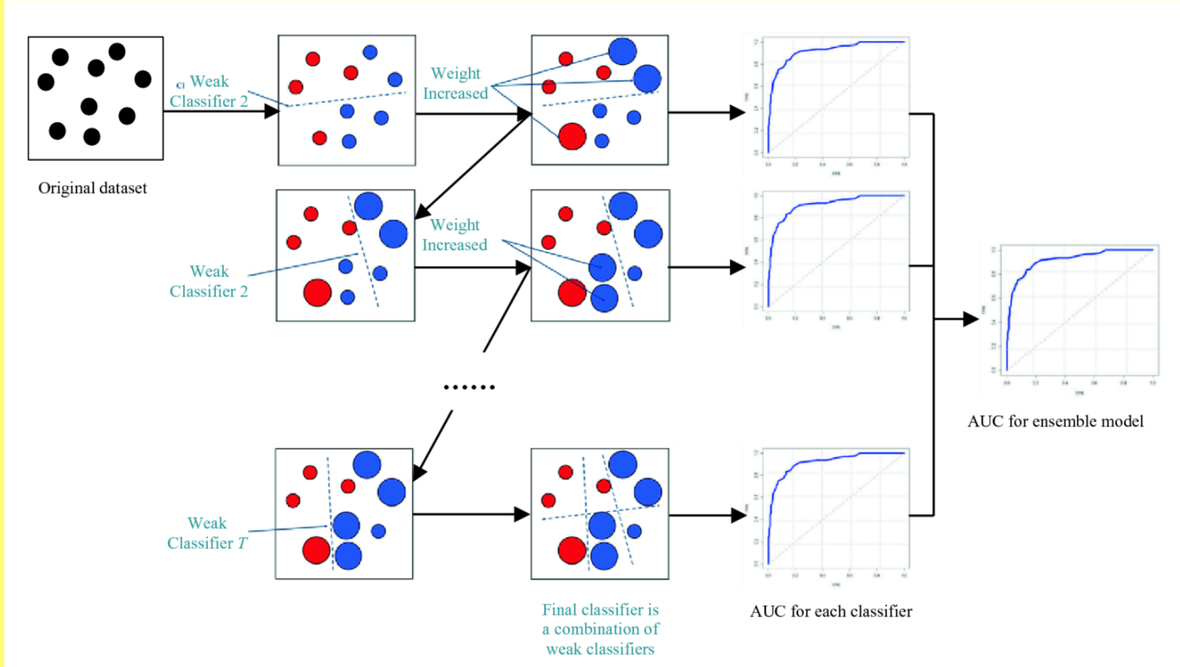
Therefore, the second tree grows on the weighted data to make improvements upon the first tree's predictions.

So, the new model uses tree one plus tree two.

- calculate the classification errors from the new ensemble model and
- develop a third tree for predicting the amended residuals.
- repeat this procedure for a particular amount of iterations.
- Upcoming trees will help us determine each observation where the previous trees failed or showed errors.

Therefore, the last ensemble model's predictions will be the overall weighted predictions provided by former tree models.

Gradient boosting provides training to several models in sequential, additive, and gradual manners.

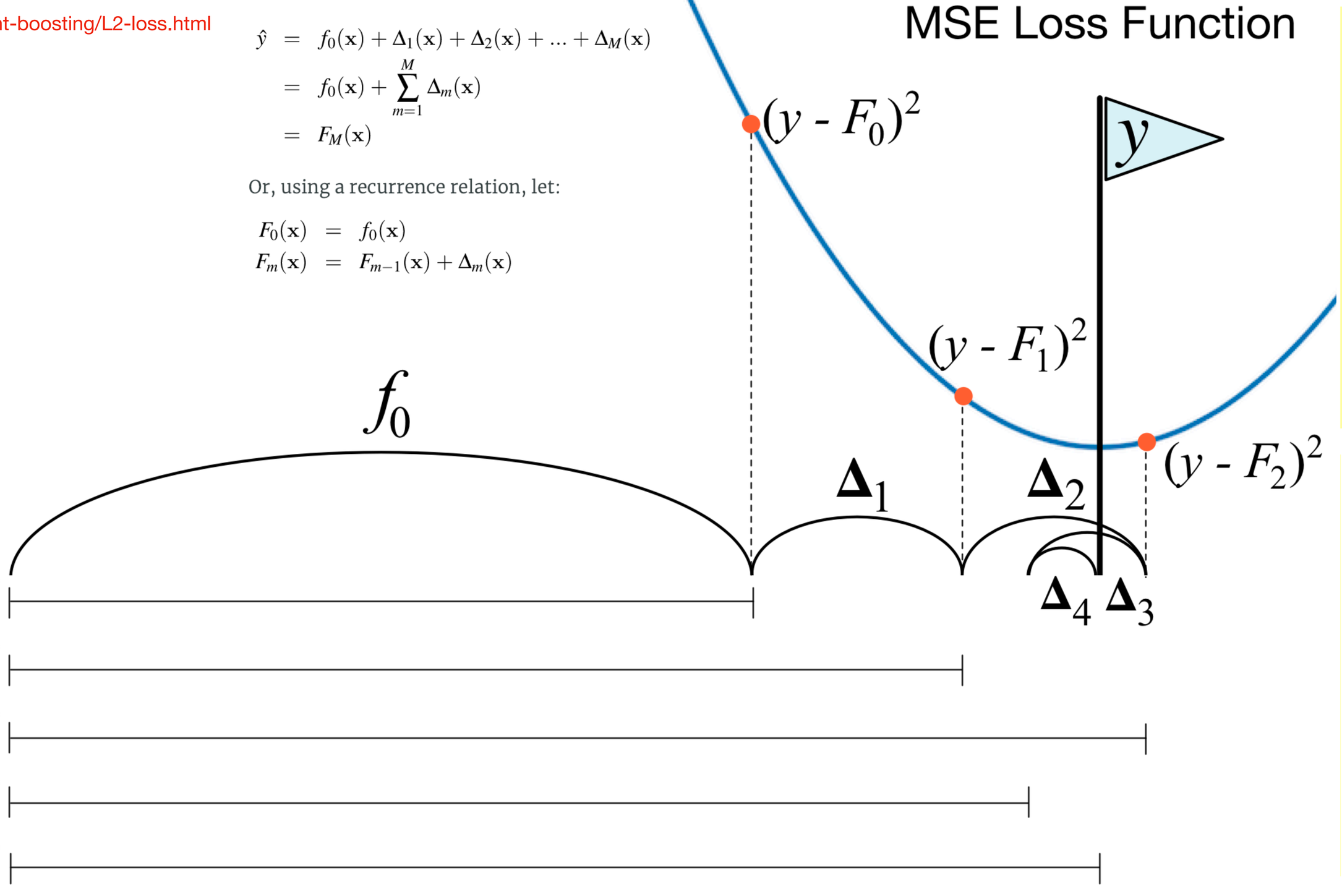


https://explained.ai/gradient-boosting/L2-loss.html

$$
\begin{aligned}
\hat{y} &= f_0(\mathbf{x}) + \Delta_1(\mathbf{x}) + \Delta_2(\mathbf{x}) + \ldots + \Delta_M(\mathbf{x}) \\
&= f_0(\mathbf{x}) + \sum_{m=1}^{M} \Delta_m(\mathbf{x}) \\
&= F_M(\mathbf{x})
\end{aligned}
$$

Or, using a recurrence relation, let:

$$
\begin{aligned}
F_0(\mathbf{x}) &= f_0(\mathbf{x}) \\
F_m(\mathbf{x}) &= F_{m-1}(\mathbf{x}) + \Delta_m(\mathbf{x})
\end{aligned}
$$



- Enough !!!
- We sign off, as of now !
- Thanks for your patience !

- Good luck and good health - all the way !
  (Unconditional wish for all)