● Classification :Predicting class membership

● Numeric Prediction :Predicting a numeric value

● Association :Determining associations between arbitrary features

● Clustering :Grouping of objects based on their similarity

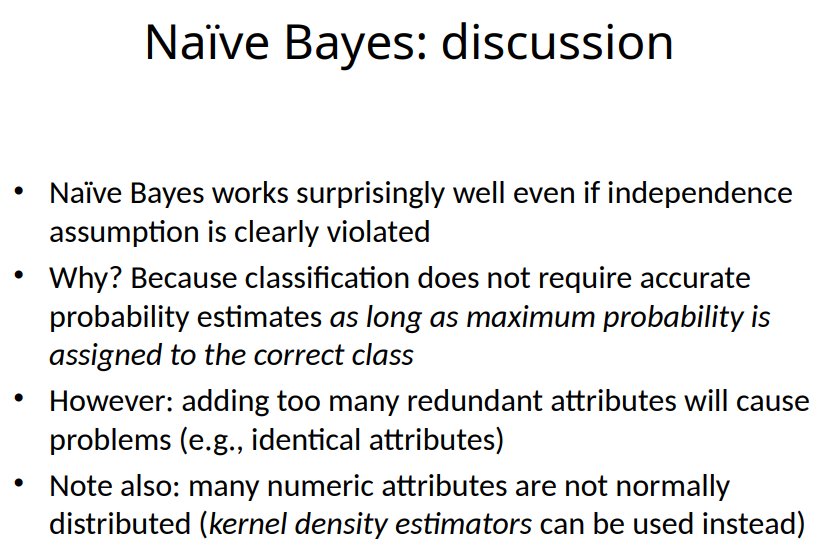
**Types of Learning**

• Supervised (inductive) learning – Training data includes desired outputs

• Unsupervised learning – Training data does not include desired outputs

• Semi-supervised learning – Training data includes a few desired outputs

• Reinforcement learning – Rewards from sequence of actions



**Linear models: linear regression** 4 77 3 5

Predict a numeric value

Inputs (attribute values) and output are all numeric

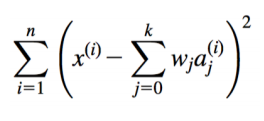
• Output is the sum of the weighted input attribute values



• The trick is to find good values for the weights

Weights are calculated from the training data

• the most famous one is to minimize the squared error



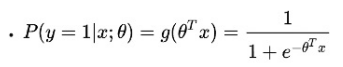
**Linear models: logistic regression** 4 80

All regressions can be used for classification. The output value of linear regression is not in the [0,1] interval, so the output value of linear regression does not represent the classification probability -> logistic regression (linear regression mainly solves regression problems, logistic regression solves classification problems )

Logistic regression mainly solves the problem of binary classification, which is used to express the possibility of something happening.

The label is not directly predicted, but the probability that the label is A is predicted. Generally, if the probability of the label being A is greater than 0.5, we consider it to be category A, otherwise it is category B. The label is not directly predicted, but the probability that the label is A is predicted. Generally, if the probability of the label being A is greater than 0.5, we consider it to be category A, otherwise it is category B.

The assumptions made by the logistic regression model are:



The meaning of this function is the probability of y=1 given x and θ

Line separates the two classes

• Decision boundary

- defines where the decision changes from one class value to the other

Decision boundary for two-class logistic regression is where probability equals 0.5

• Prediction is made by plugging in observed values of the attributes into the expression • Predict one class if output ≧ 0, and the other class if output < 0

• Boundary becomes a high-dimensional plane (hyperplane) when there are multiple attributes

Maximum likelihood

• Aim: maximize probability of observed training data with respect to final parameters of the logistic regression model

• Any regression technique can be used for classification

• Training: perform a regression for each class, setting the output to 1 for training instances that belong to class, and 0 for those that don't

• Prediction: predict class corresponding to model with largest output value (membership value)

• For linear regression this method is also known as multi-response linear regression

• Problem: membership values are not in the [0,1] range, so they cannot be considered proper probability estimates

Logistic regression for two classes is also called binomial logistic regression

• What do we do when have a problem with k classes?

The multi-classification task is divided into several two-classification tasks to solve. We train a classifier for each binary classification task. During the test, each classifier will vote for a new sample, and the category with the highest number of votes is the predicted category

The strategy have three, OvO, OvR, MvM

Linear models: the perceptron

• Observation: we do not actually need probability estimates if all we want to do is classification

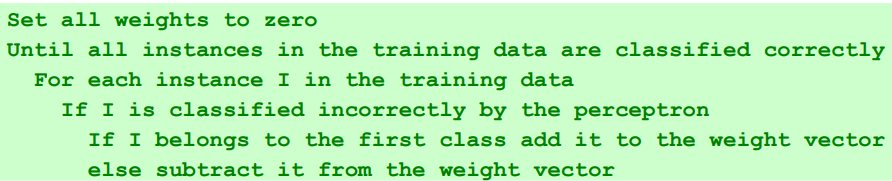
In that case there is a simple algorithm for learning a separating hyperplane called the perceptron learning rule

• Hyperplane:



where we again assume that there is a constant attribute with value 1 (bias)

• If the weighted sum is greater than zero we predict the first class, otherwise the second class





The perceptron is driven by mistakes because the classifier only changes when a mistake is made

• Another mistake-driven algorithm for finding a separating hyperplane is known as Winnow

• Assumes binary data (i.e., attribute values are either zero or one)

• Difference to perceptron learning rule: multiplicative updates instead of additive updates

• Weights are multiplied by a user-specified parameter a > 1 (or its inverse)

Another difference: user-specified threshold parameter θ

• Predict first class if



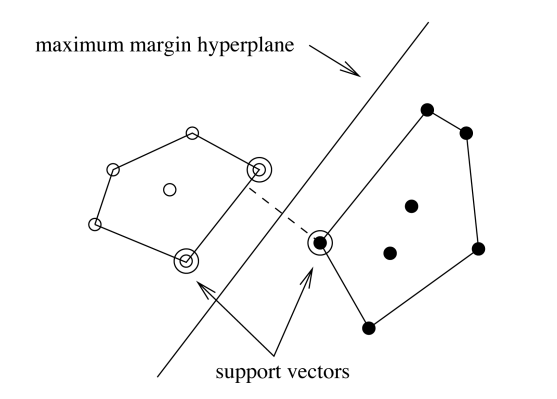
Balanced Winnow

• Winnow does not allow negative weights and this can be a drawback in some applications

• Balanced Winnow maintains two weight vectors, one for each class:



SVM



The most basic idea of classification learning is to find a hyperplane in the sample space based on the training set to separate samples of different categories. But there may be many such hyperplanes. To solve this problem, we use svm to find the best hyperplane.

The classification result produced by the maximum margin hyperplane is the most robust and has the strongest generalization ability for unclassified instances

SVM maps training examples to points in space so as to maximise the width of the gap between the two categories.

The instances closest to the maximum margin hyperplane are called support vectors

The support vectors define the maximum margin hyperplane

In machine learning, a “kernel” is usually used to refer to the kernel trick, a method of using a linear classifier to solve a non-linear problem. It entails transforming linearly inseparable data like to linearly separable ones. The kernel function is what is applied on each data instance to map the original non-linear observations into a higher-dimensional space in which they become separable.

C parameter in SVM is **Penalty parameter of the error term**. It controls the tradeoff between margin maximization and error minimization.

Hard margin: All samples must be divided correctly.

Soft margin: Allow some samples not to satisfy the constraints.

Support vector regression

• Maximum margin hyperplane only applies to classification

• However, idea of support vectors and kernel functions can be used for regression

• Basic method is the same as in linear regression: want to minimize error

• Difference A: ignore errors smaller than e and use absolute error instead of squared error

• Difference B: simultaneously aim to maximize flatness of function

• User-specified parameter e defines “tube”

From trees to rules

• Easy: converting a tree into a set of rules

• One rule for each leaf:

• Antecedent contains a condition for every node on the path from the root to the leaf

• Consequent is class assigned by the leaf

• Produces rules that are unambiguous

But: resulting rules are unnecessarily complex

Pruning to remove redundant tests/rules

From rules to trees

• More difficult: transforming a rule set into a tree

Tree cannot easily express disjunction between rules

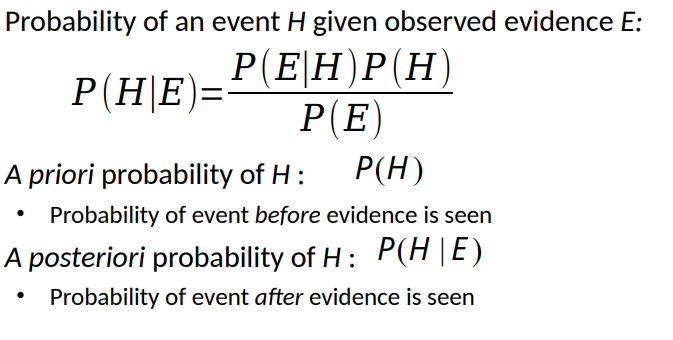
• Example: rules which test different attributes

• Symmetry needs to be broken

• Corresponding tree contains identical subtrees (→ “replicated subtree problem”)

• Support: number of instances predicted correctly

• Confidence: number of correct predictions, as proportion of all instances that rule applies to



Constructing decision trees

Strategy: top down learning using recursive divide-andconquer process

• First: select attribute for root node Create branch for each possible attribute value

• Then: split instances into subsets One for each branch extending from the node

• Finally: repeat recursively for each branch, using only instances that reach the branch

• Stop if all instances have the same class

Which is the best attribute?

• Want to get the smallest tree

• Heuristic: choose the attribute that produces the “purest” nodes

• Popular selection criterion: information gain

• Information gain increases with the average purity of the subsets

• Strategy: amongst attributes available for splitting, choose attribute that gives greatest information gain

• Information gain requires measure of impurity

Information gain: information before splitting – information after splitting(ID3)

• Problematic: attributes with a large number of values (extreme case: ID code)

• Subsets are more likely to be pure if there is a large number of values

• Information gain is biased towards choosing attributes with a large number of values

• This may result in overfitting (selection of an attribute that is non-optimal for prediction)

• An additional problem in decision trees is data fragmentation

Gain ratio (C4.5)

• Gain ratio is a modification of the information gain that reduces its bias towards attributes with many values

• Gain ratio takes number and size of branches into account when choosing an attribute

• It corrects the information gain by taking the intrinsic information of a split into account

The gain ratio is defined as the information gain of the attribute divided by its intrinsic information

• Problem with gain ratio: it may overcompensate

• May choose an attribute just because its intrinsic information is very low

• Standard fix: only consider attributes with greater than average information gain

C4.5 applies method of fractional instances:

• Split instances with missing values into pieces

• A piece going down a branch receives a weight proportional to the popularity of the branch

• weights sum to 1

CART tree learner • Uses Gini index rather than entropy to measure impurity

Pruning

• Prevent overfitting the training data: “prune” the decision tree

• Two strategies:

• Postpruning take a fully-grown decision tree and discard unreliable parts

• Prepruning stop growing a branch when information becomes unreliable

Prepruning

• Based on statistical significance test

• Stop growing the tree when there is no statistically significant association between any attribute and the class at a particular node

• Most popular test: chi-squared test

chi-squared test is used to determine whether there is a statistically significant difference between the expected frequencies and the observed frequencies in one or more categories of a contingency table.

Postpruning

Two pruning operations:

• Subtree replacement • Subtree raising

If the subtree corresponding to the non-leaf node is replaced with a leaf node to improve generalization ability, then Subtree replaced with leaf nodes

Possible strategies: • error estimation • significance testing • MDL principle

Post-pruned decision trees usually retain more branches than pre-pruned decision trees. In general, the risk of underfitting of the post-pruning decision tree is small, and the generalization ability is better than that of the pre-pruning decision tree. However, the post-pruning process is performed after the complete decision tree is generated, and all non-leaf nodes in the tree are inspected one by one from the bottom up, so the training time cost is larger than that of the pre-pruning decision tree.

• Prune only if it does not increase the estimated error

Decision tree characteristics

The axis is parallel, that is, its classification boundary consists of several segments parallel to the coordinate axis

Multivariate decision tree (oblique decision tree)

Its classification boundary can use oblique segmentation

association rules

we can look for association rules with high support and accuracy directly

• Support: number of instances correctly covered by association rule

• The same as the number of instances covered by all tests in the rule (LHS and RHS!)

• Item: one test/attribute-value pair

• Item set : all items occurring in a rule

• Goal: find only rules that exceed pre-defined support • => Do it by finding all item sets with the given minimum support and generating rules from them!

How can we efficiently find all frequent item sets?

Idea: use one-item sets to generate two-item sets, two-item sets to generate three-item sets, …

We are looking for all high-confidence rules

• Support of antecedent can be obtained from item set hash table

• But: brute-force method is (2N -1) for an N-item set

• Better way: building (c + 1)-consequent rules from c-consequent ones

• Observation: (c + 1)-consequent rule can only hold if all corresponding c-consequent rules also hold

k-means算法主要步骤

1. Randomly choose k cluster centers

2. Assign each object to the closest cluster center

3. Determine the cluster a new center for each cluster

4. Assign objects to the new nearest cluster center

5. If at least one data object has been reassigned, go to 3，if not , then stop.

split the data into K fold, and train on (K–1) folds and test on 1 fold as validation set, which is called K-fold cross validation.

K=1, The training data will be perfectly predicted.The bias will be 0 when K=1, however, when it comes to new data (in test set), it has higher chance to be an error, which causes high variance. When we increase K, the training error will increase (increase bias), but the test error may decrease at the same time (decrease variance). We can think that when K becomes larger, since it has to consider more neighbors, its model is more complex.

What are the two main problems of k-means and how can we tackle them?

1) the number of K à Cross validation

2)stack in a local minimal à try several runs

The KNN classification algorithm includes the following 4 steps:

① Prepare data and preprocess the data.

②Calculate the distance between the test sample point and each other sample point.

③ Sort each distance, and then select the K points with the smallest distance.

④according to the principle that the minority obeys the majority, the test sample points are classified into the category with the highest proportion among the K points.

The main shortcoming of this algorithm in classification is that when the sample is unbalanced, for example, the sample size of one class is very large, while the sample size of other classes is very small, which may cause when a new sample is input, K of the sample Large-volume samples in the neighbors account for the majority.

Another shortcoming of this method is the large amount of calculation, because for each sample to be classified, the distance to all known samples must be calculated to find its K nearest neighbors.

ensemle learning has good strategies on data sets of all sizes.

Combining multiple models

Large data set: Divide into multiple small data sets, learn multiple models for combination

Small data set: use Bootstrap method to sample, get multiple data sets, train multiple models separately and then combine them

In the Boosting algorithm, there is a strong dependency between individual learners and a serialization method that must be generated serially, that is, the next learner must rely on the previous learner for learning, and cannot be parallelized.

There is no strong dependency between individual learners of the Bagging algorithm and a parallelization method that can be generated at the same time, that is, it can be parallelized.

The prediction function in Bagging is uniform and equal, but the prediction function in Boosting is weighted

From the perspective of bias-variance, boosting mainly focuses on reducing bias, while bagging mainly focuses on reducing variance, which means that boosting performs better on weak learners, and reducing variance can reduce the risk of overfitting, so Bagging Usually performs well on strong classification and complex models.

Comparison of Random Forest and Bagging:

The convergence of the two is similar, but the initial performance of RF is relatively poor, especially when there is only one base learner. As the number of base learners increases, random forests usually converge to lower generalization errors. The training efficiency of random forest is often better than Bagging, because Bagging is a "deterministic" decision tree, while random forest uses a "random" decision tree.

In the Bagging method, N data sets are obtained from the overall data set with replacement sampling using the bootstrap method, a model is learned from each data set, and the final prediction result is obtained using the output of the N models, specifically: classification The problem uses N models to predict voting, and the regression problem uses N models to predict the average.

For example, Random Forest (Random Forest) belongs to Bagging. Random forest is simply to build a forest in a random way. The forest is composed of many decision trees. There is no correlation between each decision tree in the random forest.

We need to use the Bootstrap method when we learn each decision tree. In random forest, there are two random sampling processes: sampling both rows (number of data) and columns (characteristics of data) of the input data. For row sampling, the method of replacement is used. If there are N data, N data (may be repeated) are sampled, so that each tree is not all samples during training, and it is relatively difficult to appear overfitting; then perform column sampling to select m from M features (m<<M). Finally, learn the decision tree.

When predicting, each tree in the random forest predicts the input, and finally votes, which category is more, the input sample belongs to which category. This is equivalent to what I said earlier, each classifier (each tree) is weaker, but when combined (voting) is stronger.

Boosting is an algorithm that can promote a weak learner to a strong learner. Its working mechanism is: first train a base learner from the initial training set, and then adjust the sample distribution according to the performance of the base learner, so that the training samples that the previous base learner did wrong will receive more attention in the follow-up. Then the next base learner is trained based on the adjusted sample distribution; this is repeated until the number of base learners reaches the specified value T, or the entire integration result reaches the exit condition, and then these learners are weighted and combined.

The stacking method refers to training a model to combine other models. First, we train a number of different models, and then use the output of each model previously trained as input to train a model to get a final output.

Semi-Supervised Learning (SSL) is a learning method that combines supervised learning and unsupervised learning. Semi-supervised learning uses a large amount of unlabeled data and a small amount of labeled data.

Reinforcement learning is to guide the behavior of the agent by interacting with the environment to obtain rewards. The goal is to make the agent get the greatest reward

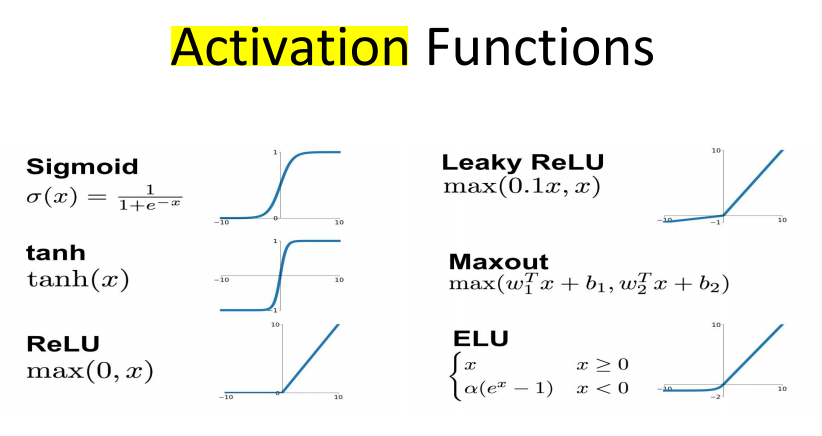
Precision p: Proportion of relevant among retrieved, it is defined as the degree of exactness.

Recall r: Proportion of retrieved among relevant, it is defined as the degree of completeness.

Normal search engine: recall is more important than precision.

Email Spam detection: precision is more important than recall.

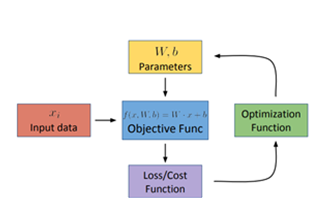
system detecting cancer: recall is very important then do the check again



**motivation**: use for non-linear separating.

The activation after the linear classifier gives us an idea of how much the neuron “supports” the feature

• Activations also helps us map linear spaces into non-linear spaces



**Input data**: the data that we provide to our machine learning model.

**Objective function**: mathematical representation of our modle, reflects how our machine learns from the input data. We will optimize the function during training. E.g. for a simple linear regression model, the objective founction can be represent as f(x)=wx+b(wàweight, bà bias).

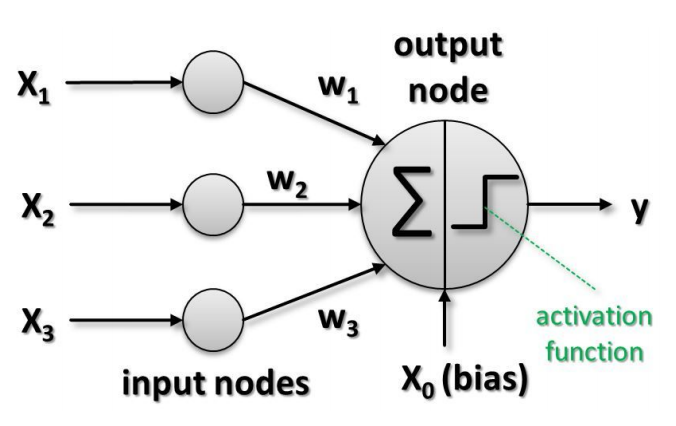
**Loss/cost function**: evaluate how good our model performs. E.g. Mean squared error, cross entropy loss.

**Optimization function:** defines how our model learns, in order to minimize the error. We can update our objective function based on our optimization function. E.g Gradient Descent

Neural network: a complex structure that aims to process information and use the results to learn, like a human brain

• Structure:large number of highly interconnected processing elements (neurons) working together

An artificial neural network (ANN) is a computational model based on the structure and functions of biological neural networks.



Single Neuron

A Neuron can be thought of as a linear classifier plus an activation function

Gradient Descent

We have k parameters w1 ,…,wk we would like to train for a model (representation, objective function) -- with respect to an error/loss J(w1 ,…,wk ) to be minimized

Gradient Descent:

1. Initialize w1 ,…,wk randomly

2. Update w1 ,…,wk to reduce J(w1 ,…,wk )

3. J’(w1 ,…,wk ) tells us in which direction J(w1 ,…,wk ) increases the most

4. Use opposite direction of J’(w1 ,…,wk ) to minimize J(w1 ,…,wk )

The step size or learning rate defines how big a step we should take in the direction of the gradient

Larger learning rates/step size (α) can speed up

-- but with too large α the optimum can be skipped or jumped over

- To small α will slow down the process

- To combine the gradient descent with line search to optimize the process (on every iteration the α value is determined using the line search)

Problem: local minimum of a function

we’ve computed the objective function and gradient for every instance, adjusted our parameters and then continued with the next instance. This approach is known as stochastic gradient descent (SGD)

Another approach is to accumulate the gradients over all instances, and then do a single update to the parameters - this approach is known as Batch gradient descent

SGD will be affected by outliers

Batch GD will drown out the effect of the outliers

But Batch GD look over the entire dataset before making any progress - so it’s much slower.

Solution

- Minibatch SGD: Perform updates after looking at a “minibatch” - Much faster than Batch GD, but largely avoids the issues with SGD.

Parameter Initialization

• Zero initialization

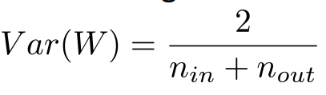
• Random initialization

• Avoids symmetry in the network by having different random weights

• Xavier Initialization

Different for each layer - depends on the number of connections coming in and going out!

Xavier initialization says we should pick the random numbers from a distribution with zero mean and the following variance



Q: Why should we never use zero initialization with neural networks?

A: All the neurons see the same input

- and with the same weight matrices, they will make the exact same decisions!

L2 regularization

Idea: Introduce to the cost functions some extra weights to penalize extreme parameter weights

Dropout

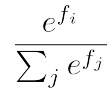
“Randomly drop neurons from the network during training”

In order to avoid overfitting the training data and make the training of the model more robust and improve generalization.

For multi-class classification, these scores are not interpretable. Their absolute values don’t give us any insight, we can only compare them relatively.

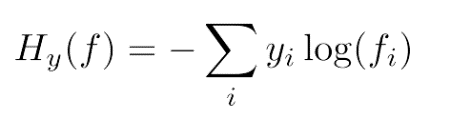
The softmax function helps us transform these values into probability distributions:

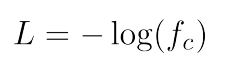
each output can be treated as the probability of that class



Cross Entropy Loss

It generally performs better for more complex models.





y represents the true probability distribution (so y i = 1 for the correct class i, and 0 otherwise)

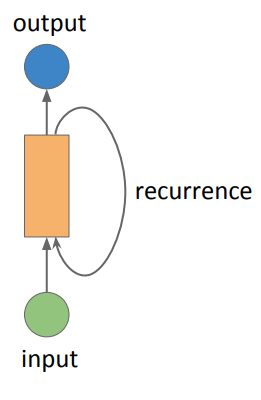
f i represents the score of class i from our classifier

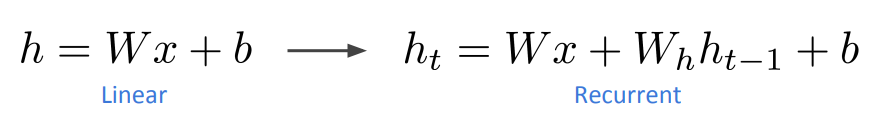
A more principled reason arises from the underlying mathematics of MSE and Cross Entropy

MSE causes the gradients to become very small as the network scores become better, so learning slows down

Recurrent Neural Network

RNN is mainly used for sequence data processing, such as text translation, which has a high degree of correlation between input and output sequences.

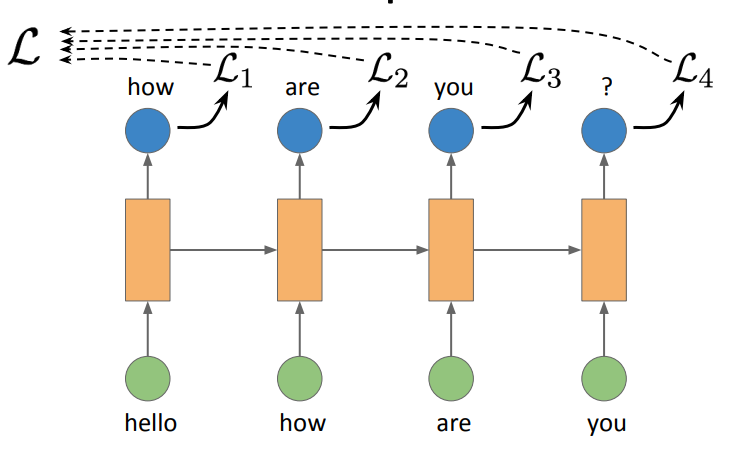




Loss Computation

Each of these outputs can be used to get one loss per timestep

We add all of these losses together to get a single loss for our optimization algorithm



Issues with Vanilla RNN

• Information decay – long-term dependencies

The last timestep remembers very little about older timesteps, since it needs to remember information from recent history and the current timestep

• Vanishing gradients

• Exploding gradient

One potential solution: Long short-term memory (LSTM)

at each timestep we can choose some information to be “important” and tell the network to remember it for longer

Intuition: We have a “memory cell” or “cell state” that is passed along the time steps. At each timestep, the unit decides to forget some information from this cell and add some new information from the current input!

This effectively helps us solve the Information decay and potentially reduces the vanishing/exploding gradient problem

Advantages

• Can process any input length

• Model size is independent of the sequence length Disadvantages

• High Computation time

Task Variations

• Sequence classification

• Per timestep classification

• Sequence generation

Sequence generation

During training time, next word/label to predict is available because we have the entire gold target sequence. gold words are used as input irrespective of the previously predicted word

At test time, we do not have gold labels, and our history is made up of predicted labels in the past.

A prediction error at one step can affect next predictions

Architecture Variations

• Bidirectional RNN

• Averaging vs. Summary vector

• Multilayer RNN

Bidirectional RNN

a word can be dependent on both the left context and the right context

Solution: Read the sequence from left to right and from right to left

A summary vector may have forgotten the information seen at the very beginning of the sequence

An alternate is to take an average of vectors at every time step

One Hot Vector Representation

• Every word can be represented as a one hot vector

Only index that represents the input word will be one

bag of words approach

Vector can represent multiple words

Problem: order information is lost!

Context-aware approach

Solution: for N words, concatenate one-hot vectors for each of the words in the correct order

In one-hot vector representation, a word is represented as one large sparse vector

only one element is 1 in the entire vector

vectors of different words do not give us any information about the potential relations between the words

word embeddings are dense vectors in some vector space

word vectors are continuous representations of words

vectors of different words give us information about the potential relations between the words - words closer together in meaning have vectors closer to each other

“Representation of words in continuous space”

Inherit benefits

• Reduce dimensionality

• Semantic relatedness

• Increase expressiveness – one word is represented in the form of several features (numbers)

Padding

• Input sentences are of varied length

• Need a fixed length to define a fixed size of weight matrices

Solution: Pad smaller sentences with 0’s

Problem: What if one sentence is very long in a batch?

Alternatively, sort all sentences by length

• Create minibatches by putting sentences of similar length together

We are inducing a bias so that the model sees all short sentences early and all long sentences later

Solution: Sort sentences in a maxi-batch

Now choose minibatches from each maxibatch

CNN

Stride: The number of pixels to shift at each step

Depth: The depth of a Conv layer is the number of filters in it - If we have N filters, we will have N activation maps

Padding: Since we are sliding the filters within the image boundaries, the activation map will be smaller. Each image is padded with “zero" pixels to maintain the image size in the map.

Full padding same padding valid padding