Machine Learning Interview Study Guide

Alec Lau

Contents

1 Linear Regression

Linear regression is a linear model, i.e. it assumes a linear relationship between the input and output variables. Suppose we have a data point y_i dependent on data $\{x_{i1}, ..., x_{in}\}$. In linear regression, we assume a linear dependence, and add in some error ϵ_i, so we get

$$\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & ... & x_{1p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & ... & x_{np} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_n \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

To fit a linear model to a data set means minimizing $\boldsymbol{\epsilon}$ in $\boldsymbol{\epsilon} = \mathbf{y} - \mathbf{x} \cdot \square$.

Example 1. The height y of a ball thrown into the air aat time t is modeled as

$$h = \beta_0 + \beta_1 t + \beta_2 t^2 + \epsilon$$

Let $\mathbf{x} = (t, t^2)$. The following are assumptions of the variables and their relationships:

1. The errors are independently drawn from a distribution of the same variance.
2. The predictor variables $\mathbf{x}$ are fixed, i.e. not random.
3. Linearity (go figure).

1.1 Least Squares Estimation

We seek to find $\boldsymbol{\beta}$ such that the loss function

$$L(\boldsymbol{\beta}) = ||\mathbf{X}\boldsymbol{\beta} - \mathbf{y}||^2$$

is minimized. Thus we get

$$\begin{aligned} L(\boldsymbol{\beta}) &= (\mathbf{X}\boldsymbol{\beta} - \mathbf{Y})^T(\mathbf{X}\boldsymbol{\beta} - Y) \\ \frac{\partial L(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} &= \frac{\partial(\mathbf{Y}^T\mathbf{Y} - \mathbf{Y}^T\mathbf{X}\boldsymbol{\beta} - \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{Y} + \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} \\ &= -2\mathbf{Y}^T\mathbf{X} + 2\boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X} \\ 2\mathbf{Y}^T\mathbf{X} &= 2\boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X} \\ \mathbf{X}^T\mathbf{Y} &= \mathbf{X}^T\mathbf{X}\boldsymbol{\beta} \\ \boldsymbol{\beta} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \end{aligned}$$

Since this method is based on example input-output pairs, this is an example of supervised machine learning.

2 Logistic Regression

Suppose we have one binary variable y. In linear regression we'd want to our guess h_β of y based on $\mathbf{x}$ as

$$\begin{aligned} 0 \leq h_\beta(\mathbf{x}) \leq 1 \\ h_\beta(\mathbf{x}) = \beta^T\mathbf{x} \end{aligned}$$

In logistic regression, we instead use the sigmoid, or logistic function for this linear function:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

This looks like arctan but is always positive. This gives us

$$P(y = 1|\mathbf{x}) = p = \frac{1}{1 + e^{-(\beta_0 + \boldsymbol{\beta}\cdot\mathbf{x})}} \tag{1}$$

If y's log odds (without loss of generality) are linearly related between the predictor vector $\mathbf{x}$, i.e.

$$\log(\frac{p}{1-p}) = \beta_0 + \boldsymbol{\beta}\cdot\mathbf{x}, p := P(y = 1)$$

Then by some easy manipulation, we have (1).

2.1 Model Fitting

Say we want to find the best $\boldsymbol{\beta}$ that matches the data y vs $\mathbf{x}$. We have

$$P(y = 1|\mathbf{x}, \boldsymbol{\beta}) = \frac{1}{1 + e^{-\boldsymbol{\beta}\cdot\mathbf{x}}} := h_\beta(\mathbf{x}) \Rightarrow$$
$$P(y|\mathbf{x}, \boldsymbol{\beta}) = h_\beta^y \cdot (1 - h_\beta)^{1-y}$$

We define the likelihood function as

$$\begin{aligned} L(\beta|y, \mathbf{x}) &= P(Y|\mathbf{x}, \beta) \\ &= \prod_i P(y_i|x_i; \beta) \\ &= \prod_i h_\beta(x_i)^{y_i}(1 - h_\beta(x_i))^{1-y_i} \end{aligned}$$

To fit the model, we use gradient descent to maximize normalized log likelihood function:

$$\frac{1}{N} \log L(\beta|y; \mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} \log P(y_i|x_i; \beta)$$

2.2 Rule of 10

A rule of thumb states that regression values are stable after about 10 instances of the less frequent category be explanatory variable x_i.

3 R^2

R^2, or the Coefficient of determination tells us the proportion of variance in the data with the variance in our model:

$$\overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

$$SS_{tot} = \sum_{i=1}^{n}(y_i - \overline{y})^2$$

$$SS_{res} = \sum_{i=1}^{n}(y_i - f_i)^2 = \sum_{i=1}^{n} e_i^2$$

$$R^2 := 1 - \frac{SS_{res}}{SS_{tot}}$$

We can see that the optimal value has $R^2 = 1$. A baseline model has $R^2 = 0$. The ratio $\frac{SS_{res}}{SS_{tot}}$ is known sometimes as the unexplained variance, or Fraction of Variance Unexplained (FVU).

4 F_1 Score

In binary classification, *Precision* is the number of correctly classified positive results divided by the number of all positively classified results. *Recall* is the number of correctly classified positive results divided by the true number of positive results.

$$F_1 = \frac{2}{\text{recall}^{-1} + \text{precision}^{-1}}$$

5 KNN

k^{th}-nearest-neighbor is a nonparametric classification algorithm. One is given a set of training data with the classification. With each data point we want to classify, we find the k-nearest neighboring training data points, and have them vote on the classification of said point, where k is a hyperparameter. k is often effective when it is more than 1. Higher values of k smooth the boundaries of the categories, but eventually lead to oversimplification and higher computational cost.

6 K-means clustering

Say you have some data. A *centroid* is the center of a *cluster*, which is a set of data with the same label. Choose k centroids. The k-means clustering algorithm takes k centroids, then assigns every data point the label of the nearest cluster, while keeping the centroids as small as possible. The algorithm terminates when the centroids stabilize, or a set number of iterations is completed.

7 Regularization

In model fitting, the coefficients β_i won't generalize well if the training data has noise, i.e. the model will overfit. If the function we come up with after model fitting is excessively fluctuating, e.g. a polynomial with a large contribution of high-degree terms, the model is likely overfitted. Thus, in our error function, we add an additional error term for controlling excessively fluctuating parts of the function. Typically the error term added is used to shrink the β_is:

$$L(\mathbf{x}, \beta) + \lambda \sum_{i}^{N} |\beta_i|^q$$

where λ and q are hyperparameters.

8 L^1 vs L^2

8.1 As Loss Functions

L^2:

1. Less robust. It penalizes distance more heavily, so if an outlier appears, L^2 is more sensitive to this.

2. More stable. For any small adjustment of a data point, the regression line won't change too much. This is not the case with L^1.

3.

8.2 As Regularization

L^2:

1. Has solution uniqueness. L^2 norm has a unique solutions, but in general L^1 doesn't.

2. Has analytic solutions, and so is computationally efficient. However, L^1 has sparse properties, and so can be calculated more efficiently than one might think.

3. Shrinks coefficients evenly, whereas L^1 shrinks coefficients to zero. Since L^1 is sparse, this is built-in feature selection, as we can drop any coefficients that are zero.

9 SVM

The (Multiclass) support vector machine (SVM) loss is a loss function for the classification of data. The SVM "wants" the correct class to have a score higher than the incorrect classes by a fixed margin Δ. Suppose we take the data point x with label y, and calculate its label score function $f(\beta, x) = \beta x + \beta_0$; the score for the j^{th} class is s_j. The Multiclass SVM loss function is then

$$L = \sum_{j \neq y} \max(0, s_j - s_y + \Delta)$$

We should probably regularize this function to avoid excessive fluctuation, so lets set our regularized SVM loss function as

$$L' = \sum_{j \neq y} \max(0, s_j - s_y + \Delta) + \lambda \sum_j |\beta_j|$$

10 Softmax

Another loss function is the Softmax classifier. This is the generalization to binary logist regression to many classes. Given our score function $f(\beta, x)$ as in the last section, we set our loss function as

$$L = -\log(\frac{e^y}{\sum_j e^{f_j}}) = -f_y + \log \sum_j e^{f_j}$$

10.1 Connection to information theory

The cross entropy between a true distribution p and an estimated distribution q is

$$H(p,q) = -\sum_x p(x) \log q(x)$$

The softmax classifier minimizes the cross entropy between the estimated class probabilities $q = e^{f_y} / \sum_j e^{f_j}$ and the true distribution, which is $[0, ..., 0, 1, 0, ..., 0]$.

10.2 Numerical stability

Since softmax has exponentials, things can get out of hand fast. Thus we set

$$\frac{e^{f_y}}{\sum_j e^{f_j}} = \frac{e^{f_y - \log C}}{\sum_j e^{f_j - \log C}}$$

Setting $\log C = -\max f_j$ takes care of this issue.

11 SVM vs Softmax

Softmax allows to compute probabilities for each class. SVM is more local - it's happy once the margin is set, and doesn't micromanage beyond this objective, whereas Softmax is always trying to get better.

12 Stochastic Gradient Descent

If W is our set of weights (in matrix form) that give a score function, then stochastic gradient descent updates weights based on a timestep Δt:

$$W_i = W_{i-1} - (\Delta t) \frac{dL}{dW}$$

where $\frac{dL}{dW}$ is the matrix where the $(i,j)^{th}$ entry is $\frac{\partial L}{\partial W_{ij}}$. Typically we use either stochastic gradient descent with Nesterov momentum or ADAM.

12.1 Nesterov Momentum

We introduce a velocity variable v initialized at 0. We recursively update this as

$$v = v * mu - \text{learning rate} * dx$$
$$x = x + v$$

where *mu* is a hyperparameter. In Nesterov momentum, the gradient we use is the gradient at the next step we're going:

$$x_a = x + mu * v$$
$$v = mu * v - \text{learning rate} * dx_a$$
$$x = x + v$$

12.2 ADAM

We introduce another technique:

$$m = \beta_1 * m + (1 - \beta_1) * dx \tag{2}$$
$$v = \beta_2 * v + (1 - \beta_2) * dx^2 \tag{3}$$
$$x = x - \frac{\text{learning rate} * m}{\sqrt{v} + \epsilon} \tag{4}$$

Recommended values are $\epsilon = 1^{-8}, \beta_1 = 0.9, \beta_2 = 0.999$, and, to quote Andrej Karpathy,

"3e-4 is the best learning rate for Adam, hands down."

(Nov. 23, 2016)

13 Bayes' Theorem

Bayes' Theorem gives the probability of an event based on past information.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

$P(A), P(B)$ are the probabilities of A, B respectively of just happening, without consideration of past events. $P(B|A)$ is the probability of B given that A is true. In words,

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$$

14 Naive Bayes

We have data $\mathbf{x}$ with features x_i that we want to classify into categories y_i. In Bayesian probabiliy, denote the prior of a class $p(y_k)$. In *naive* Bayes we assume that all features x_i are mutually independent:

$$\begin{aligned} p(y_k, \mathbf{x}) &= p(\mathbf{x}, y_k) \\ &= p(x_1|x_2, ..., x_n, y_k)p(x_2, ..., x_n, y_k) \\ &\vdots \\ &\xrightarrow{naive} p(x_1|y_k)p(x_2|y_k)...p(x_n|y_k)p(y_k) \end{aligned}$$

Thus we have

$$p(y_k|\mathbf{x}) = \frac{1}{Z}p(y_k)\prod_i^n p(x_i|y_k)$$

where $Z = \sum_k p(y_k)p(\mathbf{x}|y_k)$.

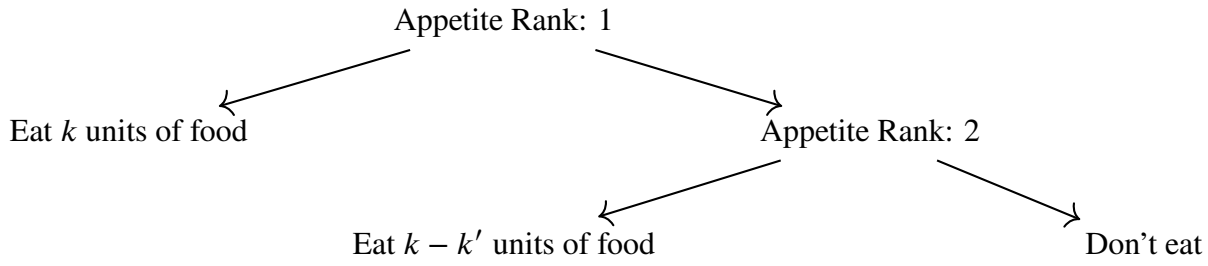
A Bayes classifier assigns the label

$$argmax_{k\in\{1,...,K\}}p(y_k)\prod_{i=1}^n p(x_i|y_k)$$

15 Decision Trees

A decision tree consists of a logical flow of the properties of an input. It's pretty easy to visualize a decision tree; see the example.

Example 1.



How do we design decision trees? Well, we want to gauge the importance of each numerical value of potential candidates for the root, or top, of the tree. We determine this by finding which feature is the least "impure," i.e. which feature does the best in properly classifying the inputs based on the training data.

For example, we could use the *Gini impurity* of a feature classification, with binary features:

$$\begin{aligned}
&[1 - (\text{Probability of Yes})^2 - (\text{Probability of No})^2]_{\text{Left branch}} \\
&\qquad \times \frac{\text{Data points that went left}}{\text{Total data points}} \\
+&[1 - (\text{Probability of Yes})^2 - (\text{Probability of No})^2]_{\text{Right branch}} \\
&\qquad \times \frac{\text{Data points that went right}}{\text{Total data points}}
\end{aligned}$$

Then we can determine the root node. Then we can go down a branch and check which feature has the lowest Gini impurity for the data separated by the root, and this continues recursively. Using Gini impurity, we take all features that we want to use in our decision tree and calculate their Gini impurities. If a node itself has the lowest score, it becomes a leaf node. If separating the data yields a lower impurity score, then pick the separation with the lowest impurity score.

If instead we have a numerical feature input, we find a cutoff for the numerical value to turn this into a binary output.

16 Bagging

Bootstrap AGGregatING is a machine learning *ensemble* tecnique used to improve stability and accuracy of machine learning algorithms.

We start with our data. Divide it up into training and validation data, with n training examples. We then create m data "bags" D_i, each with n' training examples chosen randomly with replacement (repeats are allowed). We almost always want $n' < n$. We use each of these collections of data to train a different model, i.e. each D_i data collection trains its own model. Then, we query each model in the same way and average their outputs.

For an example, see the previous and next sections.

17 Random Forests

Decision trees aren't all that flexible when it comes to new samples. Thus we can use random forests, which take advantage of the ease of building decision trees with flexibility. The result turns out to be much more accurate.

To start, we create a bootstrapped dataset (here with $n' = n$, with repetition). Then, we randomly select a subset of these features, and use Gini impurity to determine the root of a new tree. Then, we randomly select (without repetition, of course) a new set of features from which we'll pick the next node. Continuing recursively, we've created the first tree in our forest. Then we repeat. In summation:

1. Bootstrap a random dataset with repetition.
2. Choose the nodes from a randomly selected subset of variables, constructing a decision tree this way.
3. Repeat.

We've constructed our random tree. With a new data point, we run this data through each of our random trees, and have each random tree vote on the outcome of the data. This is an example of bagging: we've bootstrapped our data, and aggregated the results based on models trained on said data.

18 Boosting

We use decision trees and random forests to illustrate adaboost, which is a main example of boosting. A *stump* is a tree with two leaves, and are generally weak classifiers. However, boosting likes weak classifiers. In a random forest, all trees get a vote, i.e. they are weighted equally. In a forest of stumps in adaboost, each stump gets weighted differently. In a random forest, order of trees is irrelevant, but with adaboost, the error of the first stump influences how the second tree is made.

Adaboost consists of:

1. A combination of weak learners, which are almost always stumps, that make classifications.
2. Some stumps are weighted higher than others.
3. Each stump is made by taking the previous stumps' error into account.

We start by weighing all features equally. Calculate the Gini index for all features, and let the stump with the lowest Gini index be the first stump. The total error is the number of incorrectly classified data points times the weight. The amount of say A that this stump has is given by

$$A = \frac{1}{2}\log(\frac{1 - \text{total error}}{\text{total error}})$$

The next stump is built by the need to correctly classify the error that the previous stump made. Thus we increase the weight of a stump that correctly classified the data point that the previous stump missed,

$$w_i = w_{i-1}e^{A}$$

and decrease all other weights:

$$w_i = w_{i-1}e^{-A}$$

Then we normalize. We use these updated weights to produce a weighted Gini index, and repeat. At the end of it wall, we have a bunch of stumps with an amount of say. To determine the classification, we add up the amounts of say for all classes, and the winner is our classification.

19 Stacking

Stacking is an ensemble machine learning algorithm that learns which model to trust.

1. Unlike bagging, the models are typically different, e.g. not all decision trees.
2. Unlike boosting, the model is used to learn how to best combine the predictions of the other models, e.g. not a sequence of models correcting the previous model's mistakes.

Stacking consists of two levels:

1. Level-0 models: the models that fit on the training data and output the predictions to be compiled.
2. Level-1 model (meta model): the model that figures out how to compile the outputs of the level-0 models.

Thus the models are "stacked" on top of one another. The data used to train the meta-model is NOT the training data used to train the level-0 models.

Level-0 models are often complex and diverse. Typically it is good use models with very different assumptions about the prediction task: linear models, decision trees, SVMs, neural networks, etc.

Usually the meta-model is simple, typically using linear regression for regression tasks and logistic regression for classification tasks.

20 Supervised vs Unsupervised

Supervised learning has a ground truth, i.e. we have information about what the output values for our samples should be.

Example 1. Regression, classification.

Unsupervised learning doesn't have this; unsupervised learning is to infer a structure in the data.

Example 2. Clustering, representation learning (feature learning), density estimation.

21 Neural Networks

Suppose we have an input data vector x with weight matrix W resulting in a score vector s (each entry representing a score for a class). We modify the score function in neural networks. An example neural network would be

$$s = W_2 \max(0, W_1 x)$$

The $\max(0, -)$ layer is a nonlinearity which allows us to model nonlinear functions. A 3-layer neural network would look like

$$s = W_3 \max(0, W_2 \max(0, W_1 x))$$

where the three layers W_1, W_2, W_3 are learned.

The $\max(0, -)$ (ReLU, Rectified Linear Unit) is an example of an *activation function* (source of nonlinearity). Other activation functions are used, like tanh or leakey ReLU, which, instead of having a zero slope for $x < 0$, has a small negative slope in this domain (the sigmoid function doesn't work, because it isn't

0-centered, which makes the gradients either all positive or all negative, which produces costly zig-zagging during gradient descent. Also, its slope can get very *very* close to 0, effectively making gradient descent uselessly slow). Add too few layers, the model's not going to be good enough. Add too many, the model will overfit, and it'll perform like garbage. Also, the loss function should have a regularization term to avoid overfitting as well.

22 Data preprocessing

22.1 Mean Subtraction

Mean subtraction is a very common form of data preprocessing. It subtracts the mean of all features $x_i \in \mathbf{x}$ from each feature, which centers the cloud of data along every dimension.

22.2 Normalization

$\mathbf{x} \mapsto \frac{\mathbf{x}}{std(\mathbf{x})}$. This makes the data cloud's dimensions on the same scale, and is only necessary if different input features x_i have different scales.

22.3 PCA & Whitening

Principal Components Analysis is a technique used to reduce the dimensions of our data to speed up our algorithm. First we center, and then rotating the data by the first 100 or so eigenvectors of our data's covariance matrix. For example, if we are training our data on images, adjacent pixels are very highly correlated, and so PCA allows us to greatly reduce the dimension of the input data.

1. $X = X - \text{mean}(X, \text{axis=0})$
2. $cov = \frac{X^T \cdot X}{0^{th} \text{ axis length}}$
3. $X_{red} = X \cdot \text{eig}(cov)[:, 100]$

To whiten the data, we add one more step:

$$X_{red,white} = \frac{X_{red}}{\sqrt{\text{eigvalues}}[:, 1000] + e^{-5}}$$

i.e. we divide by the square root of the eigenvalues, and add e^{-5} or something to avoid dividing by zero. The only problem is, this could amplify noise. If our input data is a Gaussian, post-whitening our data is a Gaussian with zero mean and identity covariance matrix.

23 BatchNorm

Small changes in parameters in the first few layers of a model lead to huge amplifications in inputs in later models. The change in the distribution of inputs to each new layer slows things down because the layers constantly need to adapt to new distributions. This change in distribution is called *covariate shift*. To increase stability batchnorm normalizes the output $\mathbf{x}$ of a previous layer via the following: Let $\{x_1, ..., x_m\}$ be the values of a *single feature* for m pieces of data in our batch (our batch has m data points). of our input data.

$$\mu_B = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\sigma_B^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_B)^2$$

$$\hat{x}_i := \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$y_i := \gamma \hat{x}_i + \beta$$

where γ and β are hyperparameters.

Usually batchnorm is inserted after fully connected layers.

24 LayerNorm

LayerNorm does the same centering and normalization as batchnorm, but this time across features, not a single feature across a batch. This is thus a centering and normalization of the output of a full data vector.

25 InstanceNorm

This is the same centering and normalization as above, but across channels. Channels are sort of an extra dimension of the output. For example, for a color photo (which isn't an output, but works in the sense that it's an input) the channel depth is 3, for RGB.

26 Bias vs Variance

1. Bias is the simplifying assumption made by the model to make function approximation easier. It is the error due to the simplification assumption.

2. Variance is the amount that the estimates change between input data. It is the error from sensitivity to small fluctuations in the input data.

27 ROC

Receiver Operator Characteristic graphs provide a simple way to evaluate a classifier. The $y-$axis is the *true positive rate*, or *sensitivity*:

$$\text{sensitivity} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}}$$

The $x-$axis is the *false positive rate*, or $(1-\text{specificity})$:

$$(-\text{specificity}) = \frac{\text{false positive}}{\text{false positive} + \text{true negative}}$$

Example 1. For logistic regression, say we want a threshold cutoff for the logistic function for a binary classification. Each threshold gives us a point on this ROC graph. We want the threshold that corresponds to the highest and furthest left point on the threshold-ROC point graph.

28 Exploding/Vanishing Gradients

If your model has exploding gradients, the following can occur:

1. The model is not learning well on the training data.
2. The model has large changes in loss on each update due to instability.

If your model has vanishing gradients, the following can occur:

1. The model improves very slowly, or the model stops training very early on.
2. The weights on layers closer to the output change more than layers closer to the input.

These problems can be solved with taking fewer layers, clipping gradients, and more careful weight initialization.

29 MLP

A MultiLayer Perceptrons are the classic neural networks. Data is fed into the input layer, hidden layers act on it, and we get a loss function. These are good for classification prediction problems.

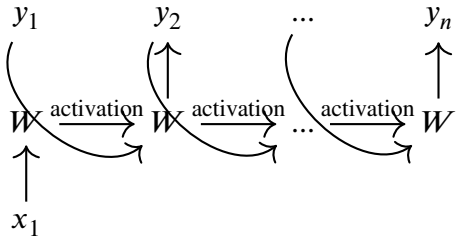
30 CNNs

Convolutional Neural Networks have a smaller window that convolves across a data point. This window could select the maximum value in each place in its convolution (maxpool) or just multiply the values in the window and add them up. These are good for image stuff.

31 RNNs and LSTMs

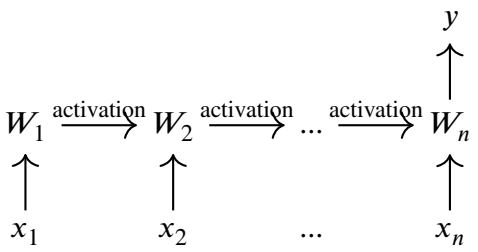
Recurrent Neural Networks are good at modelling sequence data, e.g. text, audio, etc. They use past inputs in the feed-forward of the next input.

Example 1. Below is a "one-to-many" RNN:



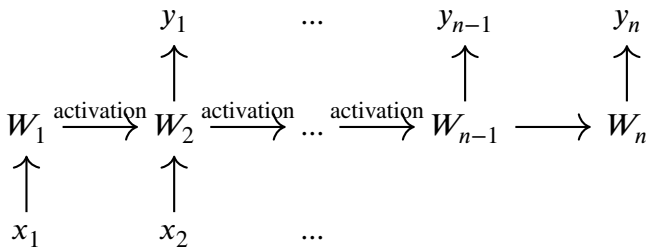
This type of RNN can be used for music generation.

Example 2. Below is a "many-to-one" RNN:



This type of RNN can be used for sentiment classification.

Example 3. Below is an example of a "many-to-many" RNN:



This type of RNN can be used for name entry recognition or machine translation.

Long Short-Term Memory blocks are a way to deal with the vanishing gradient problem that is prevalent in traditional RNNs. An i^{th} LSTM unit has outputs c^i, a^i and inputs from the previous LSTM unit c^{i-1}, a^{i-1}. These are calculated by the following:

$$\tilde{c}^i = \tanh W_c[\Gamma_r \cdot a^{i-1}, x^t] + b_c$$
$$c^i = \Gamma_u \cdot \tilde{C}^i + \Gamma_f \cdot c^{i-1}$$
$$a^i = \Gamma_o \cdot c^i$$

32 Generative vs Discriminative

In supervised learning, a discriminative model models the decision boundary between classes. It learns the conditional probability distribution $p(y|x)$.

A generative model learns the actual probability distribution of each class. It learns the joint probability distribution $p(y, x)$ using Bayes' Theorem.

33 Parametric vs Nonparametric

Parametric algorithms make assumptions about the input data. Nonparametric algorithms don't.