Welcome

Overview of machine learning and discussion

Introduction

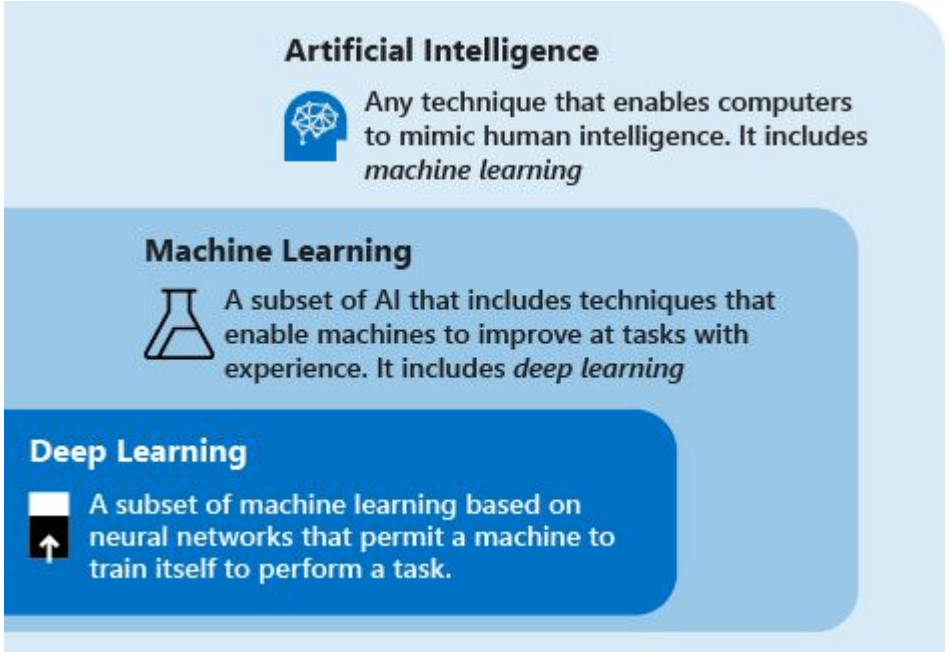
Intro - self, course material and teaching methodology: top down vs. bottom up.

Topics:

- Machine learning/ deep learning quick overview. What can and can’t be achieved?
- Random Forests, Naive Bayes, Neural Nets, Collaborative filtering.
- Data Leakage
- Overfitting
- The importance of picking a good validation set: cross validation? Random sample?
- How much data is enough? Data augmentation
- Loss functions: Mean Squared Error, cross entropy, binary crossentropy.
- Learning rate finder - paper.
- Domain transfer.

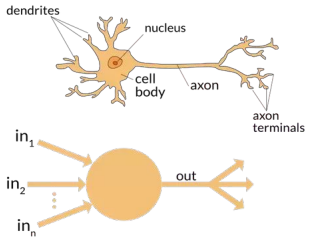
Machine learning / deep learning definition

- Deep learning is a computer technique to extract and transform data.
- Using multiple layers of neural networks.
- Each of these layers takes its inputs from previous layers and progressively refines them. The layers are trained by algorithms that minimize their errors and improve their accuracy. In this way, the network learns to perform a specified task.



https://learn.microsoft.com/en-us/azure/machine-learning/concept-deep-learning-vs-machine-learning

Brief history of neural nets:



- **1943 McCulloch and Pitts develop a mathematical model of an artificial neuron.** “Because of the “all-or-none” character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms”
- **1959-1969**, Perceptrons published, in which Marvin Minsky proved that **a single layer of neurons was unable to learn X-OR** (e.g).
- He also showed that **using multiple layers would allow this issue to be addressed**.
- Only the first of these insights was recognized, and global research on neural nets stalled for the next 2 decades!
- **1986** James McClellan & David Rumelhart publish Parallel Distributed Processing

Parallel distributed Processing (1986)

- **Basic premise:** computer programs work very differently to brains, which is why they're so bad at things like image recognition and other human/animal intelligence solvable problems.
- PDP provided "a computational framework which seems closer than other frameworks to the style of computation as it might be done in the brain.
- **PDP requires:**

1. A set of *processing units*
2. A *state of activation*
3. An *output function* for each unit
4. A *pattern of connectivity* among units
5. A *propagation rule* for propagating patterns of activities through the network of connectivities
6. An *activation rule* for combining the inputs impinging on a unit with the current state of that unit to produce an output for the unit
7. A *learning rule* whereby patterns of connectivity are modified by experience
8. An *environment* within which the system must operate

80s and 90s

Neural networks used for practical purposes, but a misunderstanding of the theoretical issues held back the field:

Universal approximation theorem: a 2 layer neural network (with a nonlinearity) can solve any computable problem to an arbitrarily high level of accuracy, provided the right set of parameters, and enough of them to represent the problem.

People worked with 2 layer nets and one nonlinearity, and tried to scale this laterally.

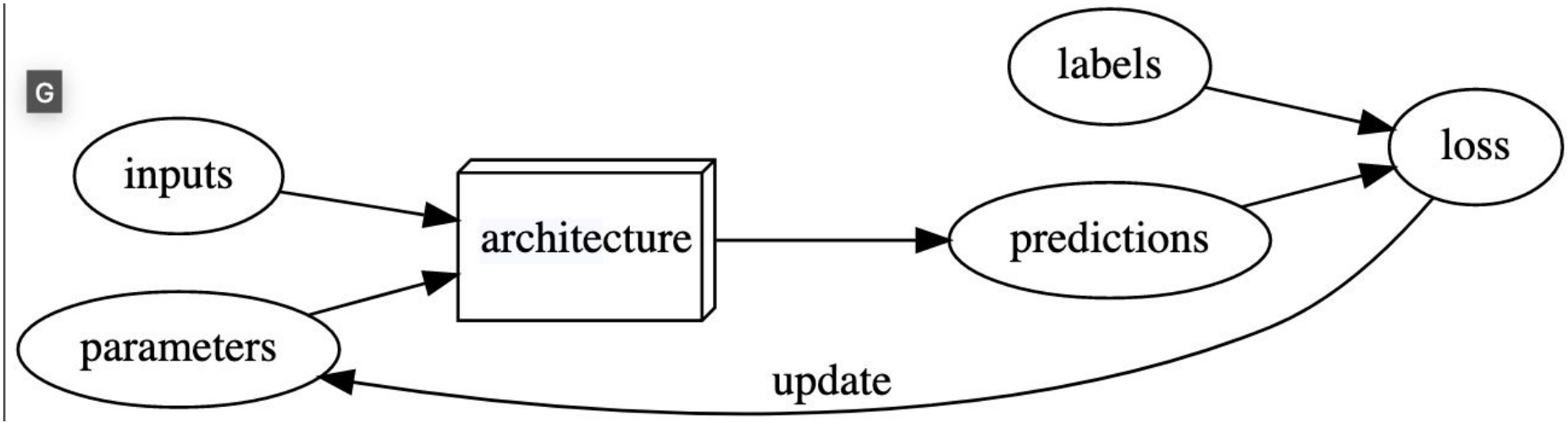
In practice, deeper nets train more quickly and use less computation. This was first discovered through experimentation in the 90s.

Deep Learning is now best in the world in the following areas:

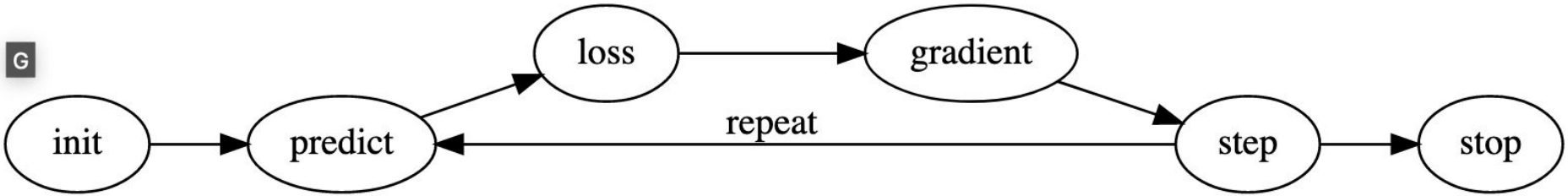
- Natural language processing (NLP):: Answering questions; speech recognition; summarizing documents; classifying documents; finding names, dates, etc. in documents; searching for articles mentioning a concept
- Computer vision:: Satellite and drone imagery interpretation (e.g., for disaster resilience); face recognition; image captioning; reading traffic signs; locating pedestrians and vehicles in autonomous vehicles
- Medicine:: Finding anomalies in radiology images, including CT, MRI, and X-ray images; counting features in pathology slides; measuring features in ultrasounds; diagnosing diabetic retinopathy
- Biology:: Folding proteins; classifying proteins; many genomics tasks, such as tumor-normal sequencing and classifying clinically actionable genetic mutations; cell classification; analyzing protein/protein interactions
- Image generation:: Colorizing images; increasing image resolution; removing noise from images; converting images to art in the style of famous artists
- Recommendation systems:: Web search; product recommendations; home page layout
- Playing games:: Chess, Go, most Atari video games, and many real-time strategy games
- Robotics:: Handling objects that are challenging to locate (e.g., transparent, shiny, lacking texture) or hard to pick up
- Other applications:: Financial and logistical forecasting, text to speech, and much more...

Normal programming paradigm:

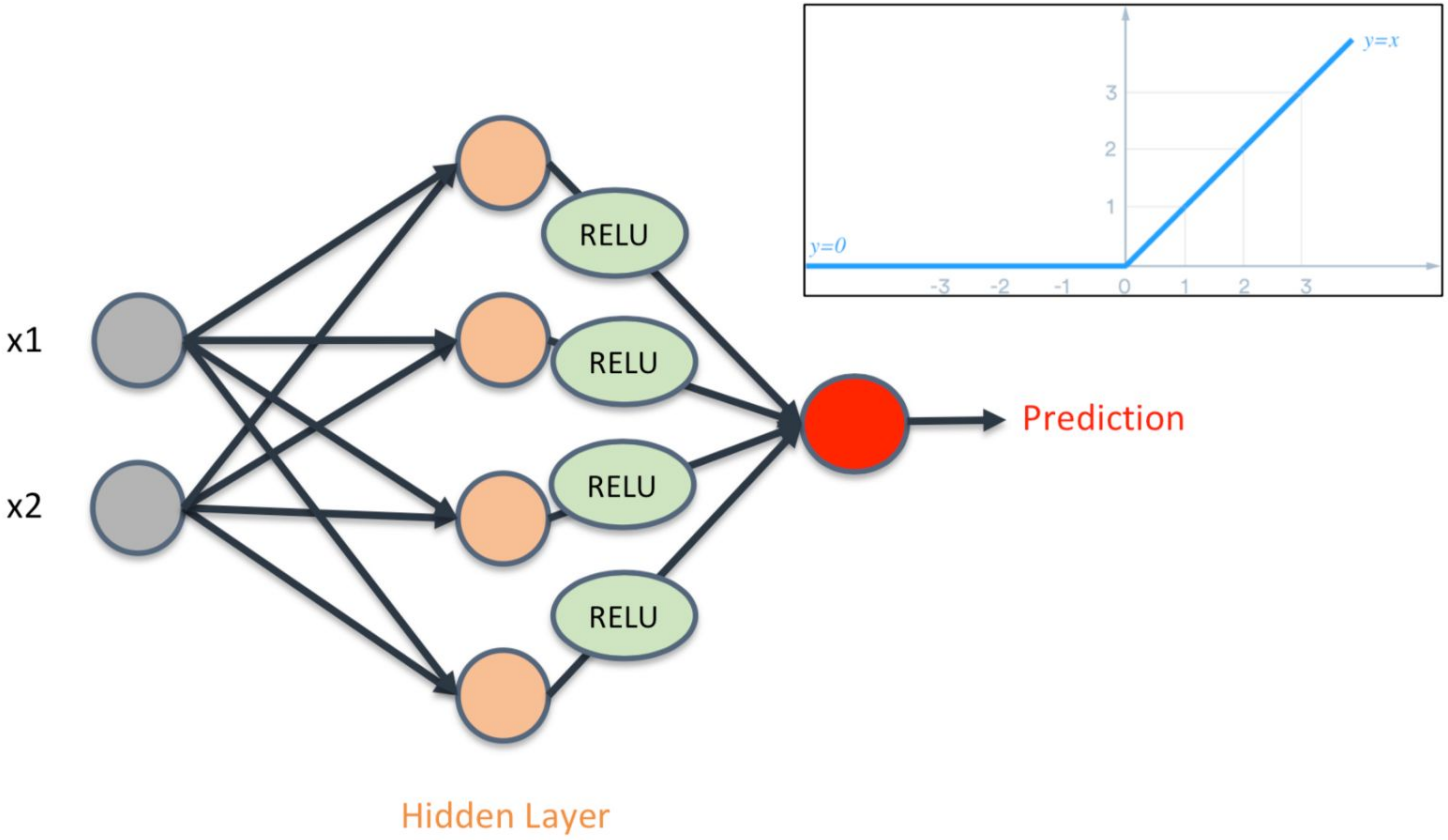
Components for training a model



Process for training a model



The Next Simplest Possible Neural Net



Limitations Inherent To Machine Learning (supervised)

- A model cannot be created without data.

- A model can only learn to operate on the patterns seen in the input data used to train it.

- This learning approach only creates *predictions*, not recommended *actions*.

- It's not enough to just have examples of input data; we need *labels* for that data too (e.g., pictures of dogs and cats aren't enough to train a model; we need a label for each one, saying which ones are dogs, and which are cats).

Out of domain data

- Models used to make predictions on types of data not in the training set will be less reliable.
- E.g. If we built a classifier trained on images of dogs from a google search, then tried to apply it to cctv camera footage, we would expect it to perform poorly.
- The model specifically is optimized to make predictions based on patterns it has seen in the past.
- https://www.flawedfacedata.com/

Has anybody else come across a case where a model is used on out of domain data?

Domain Shift

- Over time things in the world change. People's behaviour, the economy, technology etc.
- A model which was once providing accurate and valid predictions can become outdated as the features which were used to train it no longer represent the features found in the real world.

Has anyone had to deal with domain shift in their work? Or does anyone know any interesting examples?

Feedback loops

Once a model is in production, it can become an influencing component of the environment it is trying to make predictions of.

This can create a feedback loop, where the model causes unintended and often undesirable changes to the system it is embedded in.

Example: YouTube's recommendation system. Designed to increase the amount of time a user spends watching YouTube, the recommender would promote videos 'light on facts and heavy on speculation', and promote conspiracy theories.

Feedback loops: predictive policing example

- A *predictive policing* model is created based on where arrests have been made in the past. In practice, this is not actually predicting crime, but rather predicting arrests, and is therefore partially simply reflecting biases in existing policing processes.

- Law enforcement officers then might use that model to decide where to focus their police activity, resulting in increased arrests in those areas.

- Data on these additional arrests would then be fed back in to retrain future versions of the model

Data

- Split into independent and dependent variable.

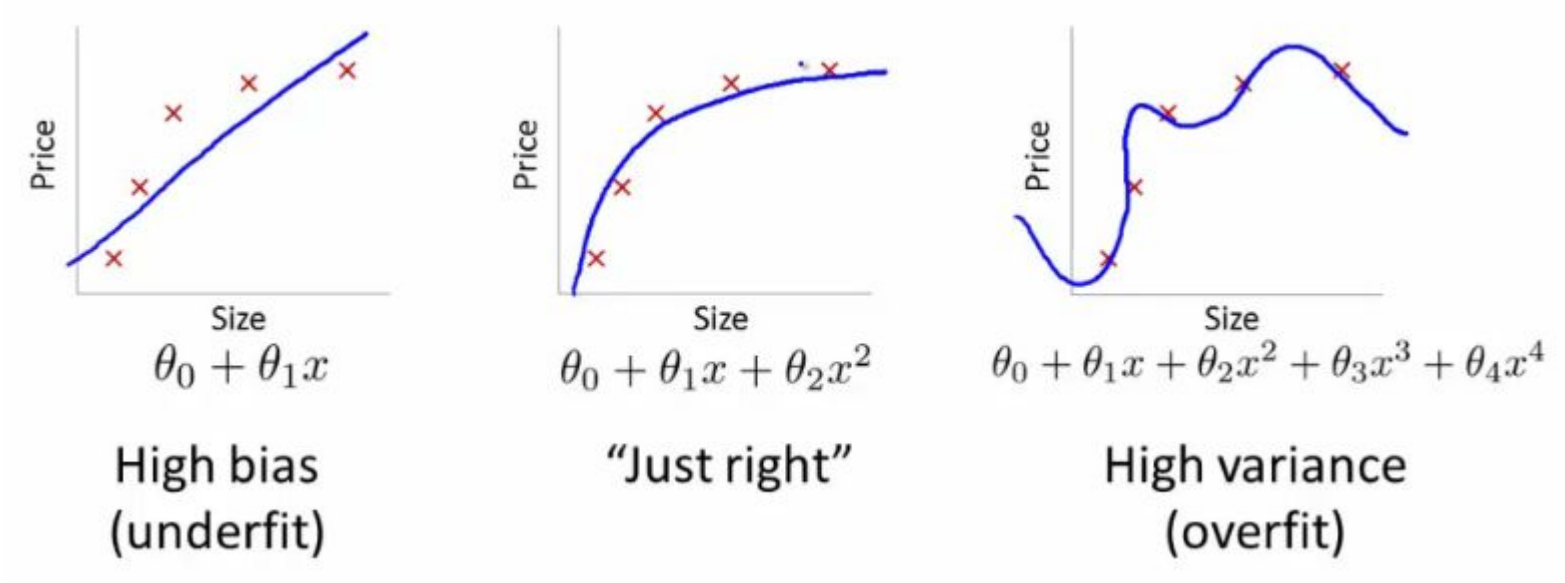
Independent var (x) (features) E.g. an image, lat/long, num of rooms	Dependent var (y) (labels) E.g. ‘cat’ ‘dog’ for categorical, Price,

Data

- Split into training and validation set
- We do this so that we can evaluate the model's performance on data it has never seen before. E.g.

<table>
<tr><td colspan="4">Data</td></tr>
<tr><td colspan="2">Training set</td><td colspan="2">Validation set</td></tr>
<tr><td>Features</td><td>Labels</td><td>Features</td><td>Labels</td></tr>
</table>

Overfitting

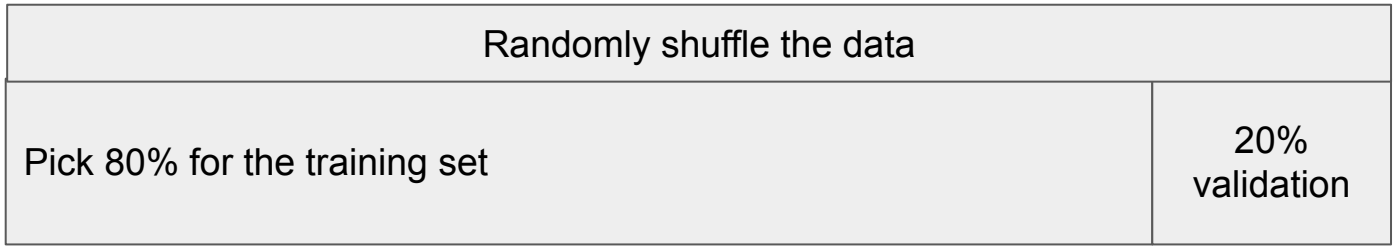


Source: Andrew Ng's Machine Learning Coursera class

The importance of picking a good validation set

- The validation data is used to measure how well the model is doing on unseen data.
- The validation set should be a measure of how well the model will perform in the real world.

How to pick?



This is ok sometimes but…

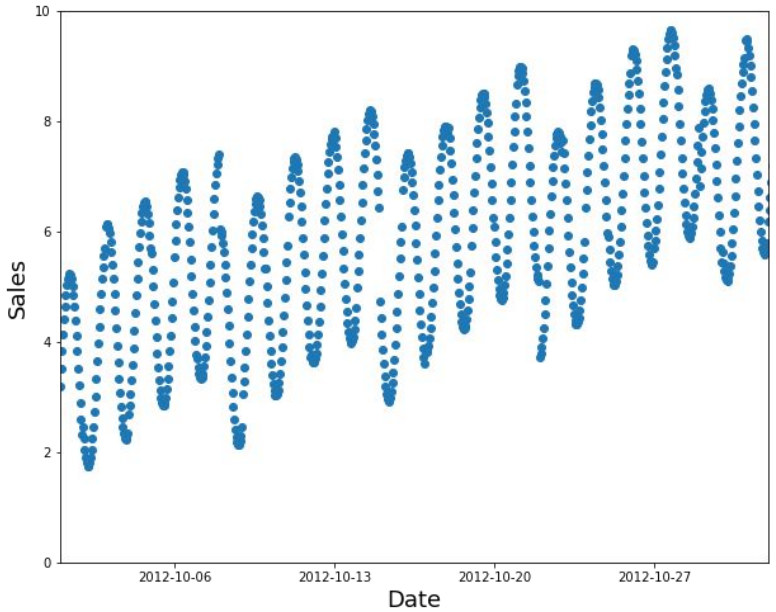
- Advantages / disadvantages / considerations

Source: How (and why) to create a good validation set

Validation set e.g. time series

In this scenario we'd like to predict sales at a point in the future.

If we just shuffled the data and took a random sample for the validation set…



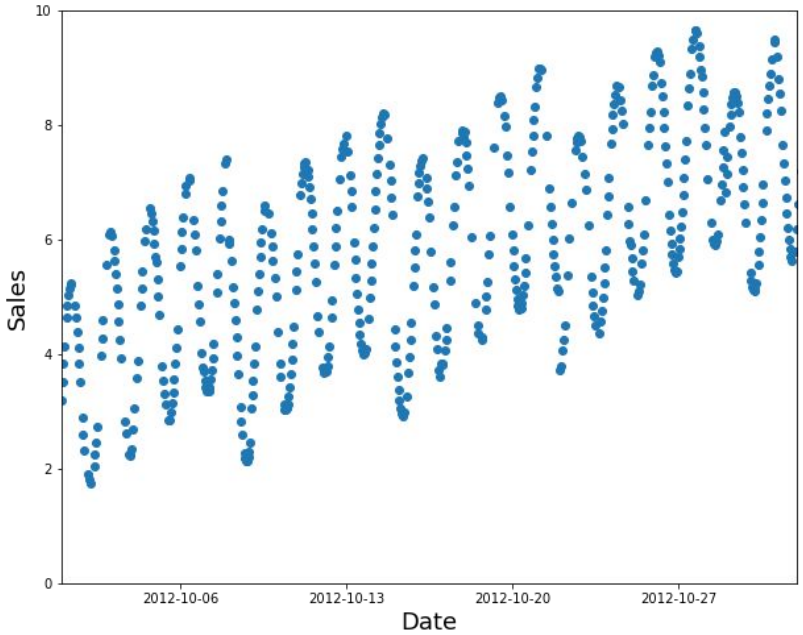
Source: [How (and why) to create a good validation set]

Validation set e.g. time series

The validation samples would always lie pretty close to samples seen in the training data. The model would be able to ‘fill in the gaps’.

This isn’t really what we are trying to validate.

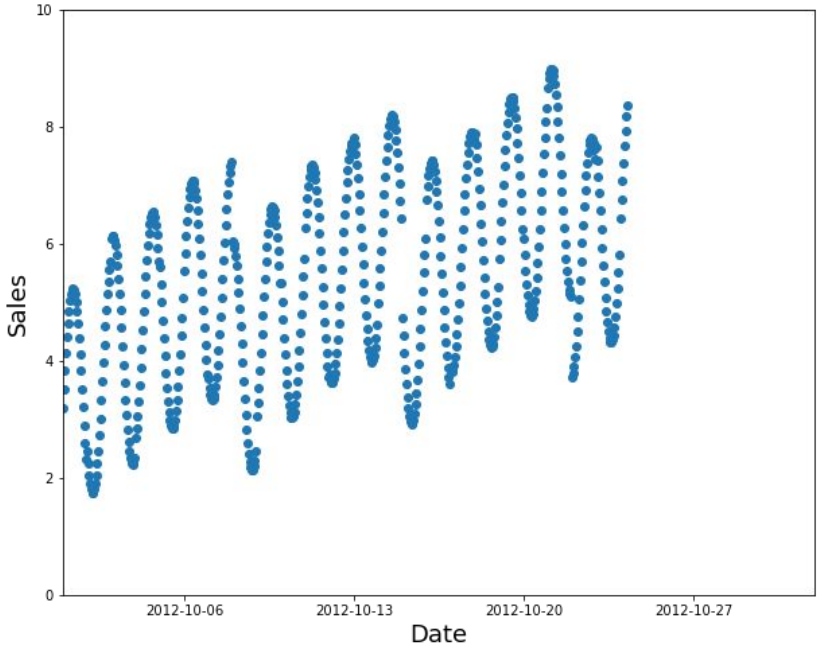
What we’d actually like to find out is how well does our model do at predicting future values.



Source: How (and why) to create a good validation set

Validation set e.g. time series

A better way to construct the validation set would be to pick values in the future relative to the training data. This way the validation score will indicate how well the model might perform on future events.



Source: [How (and why) to create a good validation set]

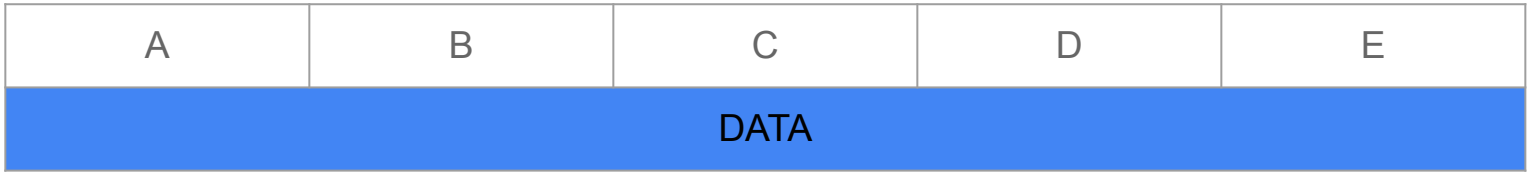
Cross Validation

Discarding 20% of the data just to validate the model seems like a waste of that 20%.

Cross validation provides a way to use 100% of the data for training, whilst still providing a validation score.

Cross-validation only works in the same cases where you can randomly shuffle your data to choose a validation set.

Fold	Training	Validation
1	BCDE	A
2	ACDE	B
3	ABDE	C
4	ABCE	D
5	ABCD	E



Validation set discussion: example

Distracted driver recognition:

In this kaggle competition you are given driver images, each taken in a car with a driver doing something in the car (texting, eating, talking on the phone, makeup, reaching behind, etc). Your goal is to predict the likelihood of what the driver is doing in each picture.

The training set contains pictures from 10 different drivers

How should the validation set be constructed?

Data Leakage - Example 1 - fields as a proxy

Data leakage occurs when information is present in the training set which would not be available in production.

Example 1: A company builds a model to predict which interviewees to hire, using past interview data as training data.

Data leakage could occur when the cell phone field of a candidate is only filled in for candidates invited back for a second interview.

The cell phone field becomes a proxy for the interviewer’s confidence in that candidate. Consider what would happen for years 2019, 2020, next year.

How would the validation score look?

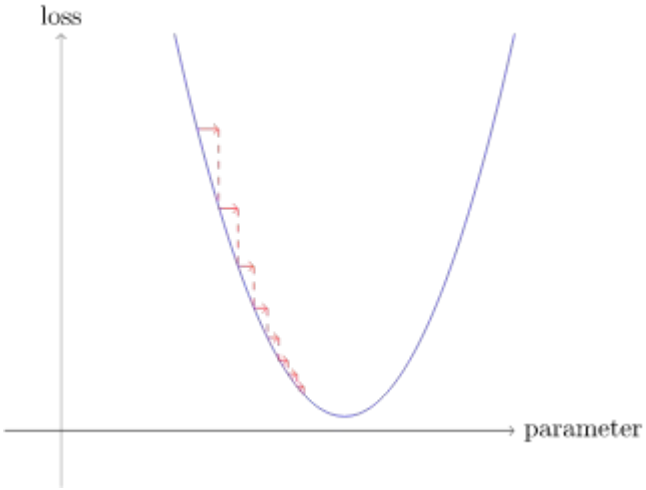
Data Leakage - Example 2 - leaking distribution

- Consider a time series trend where the validation set contains features with a different distribution than the training set.
- We might want to fill in missing values in the training set with the mean value, or scale the values to lie between zero and 1 by dividing by the maximum value.
- If we do this before making the train/valid split, the values in the validation set will affect the distribution of the values in the training set.
- The validation set is supposed to represent unseen data - so its having an effect on the training set is not ok!

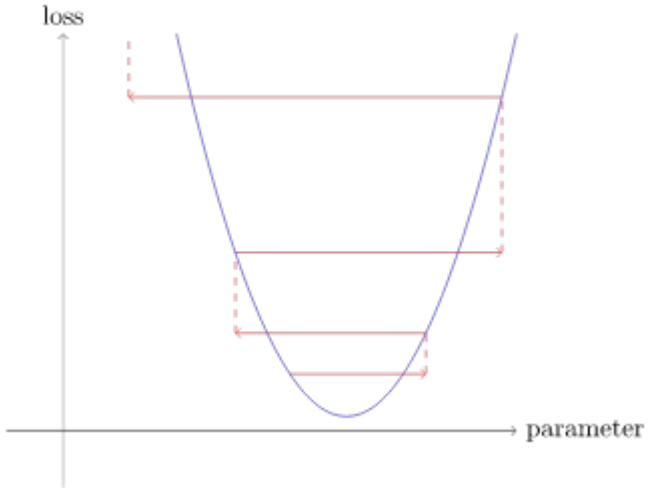
https://docs.google.com/spreadsheets/d/1c-5BRVJq3tfVEIlsM53Pa6hsDxZZ0hpK3PCpAVuHNwM/edit#gid=0

Learning rate

Low



High



Learning rate finder

One way of finding the optimal learning rate:

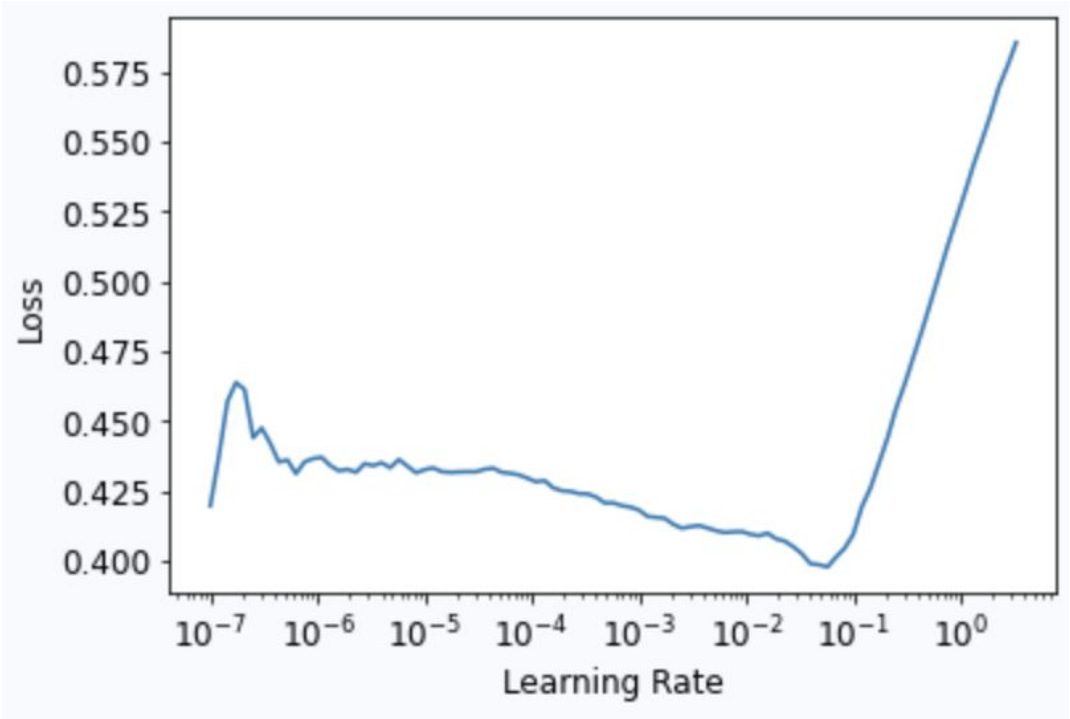
Start with a super low (10e-7) lr

Increment the lr a tiny bit each mini batch

Plot the loss for each learning rate.

Pick the LR one order of magnitude before the loss starts to diverge.

Works well for SGD



ADAM optimizer

ADAM optimizer has an adaptive learning rate for every parameter in the model.

It works by keeping track of the gradient for each parameter across previous updates, and calculates a momentum based on this gradient.

One Cycle Policy

Starts with a very low learning rate - this is to initialize the activations. A high learning rate on randomly initialized activations would risk the model diverging rather than converging.

Large learning rates are required to allow the model to train quickly, but also to jump out of local minima and traverse large saddles in the loss landscape.

Small learning rates are required to allow the fine tuning of parameters to reach the very bottom of minima in the loss landscape.

Discussion

Domain transfer?

Pretrained models vs. from scratch?