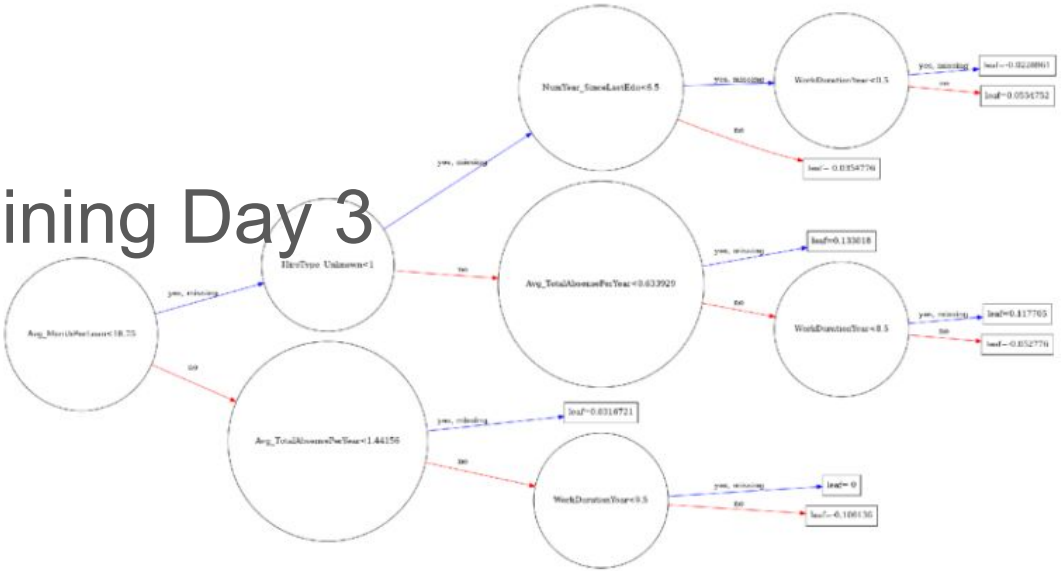
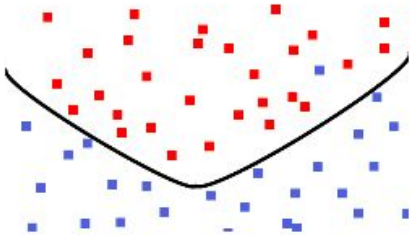
Classification and Regression

Exxon Training Day 3

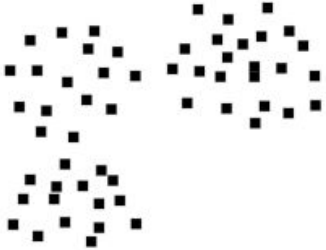


Types of Machine Learning



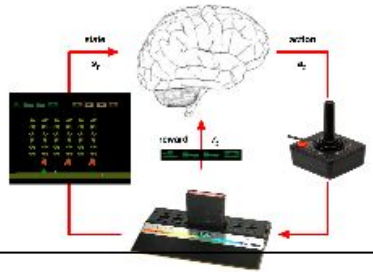
Supervised Learning

Reinforcement Learning



Unsupervised Learning

Learn the mapping function from input X to output Y

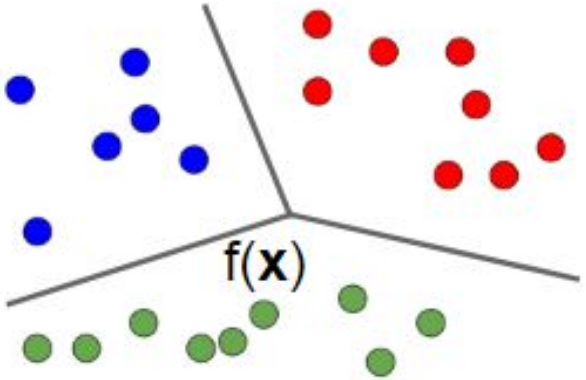


Learn by trial and error

Learn the structure of the data

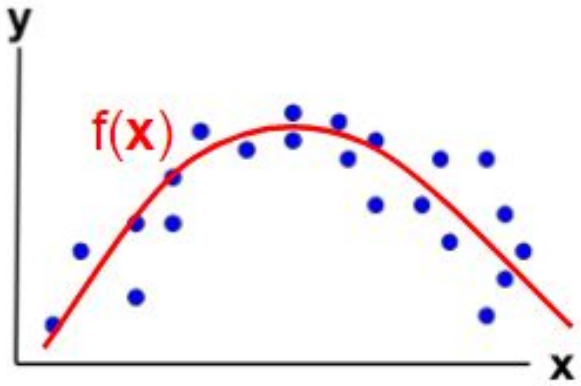
Classification and Regression

Supervised learning



Classification

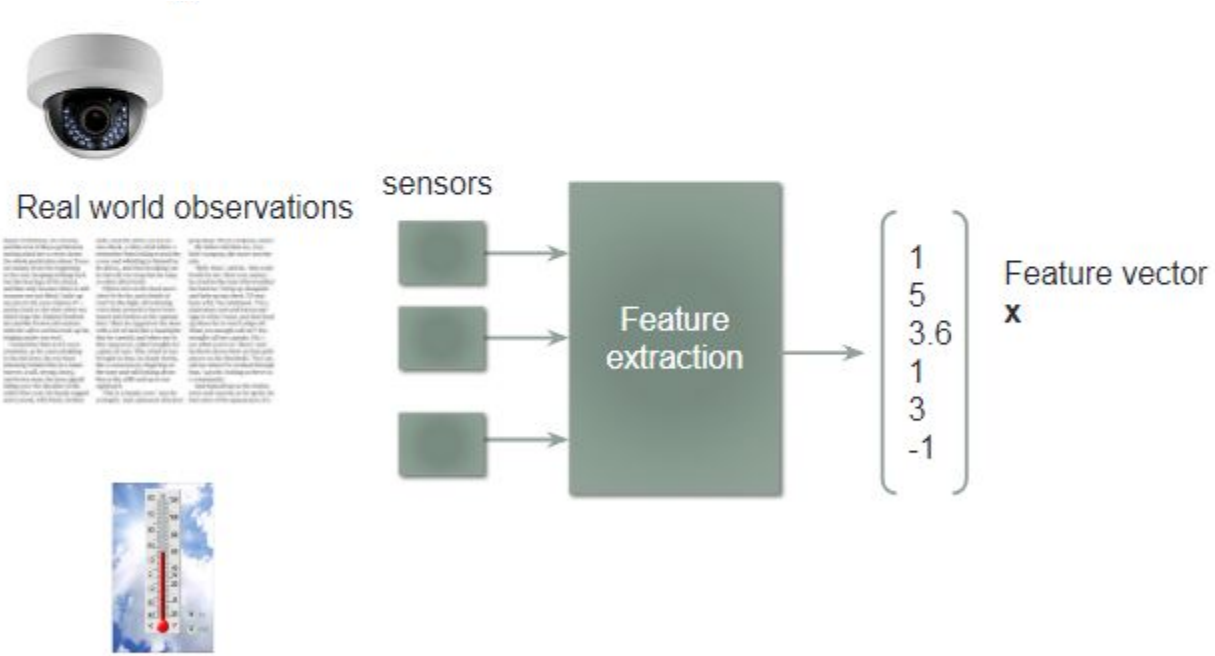
Cat vs dog
Age range of a person in the picture



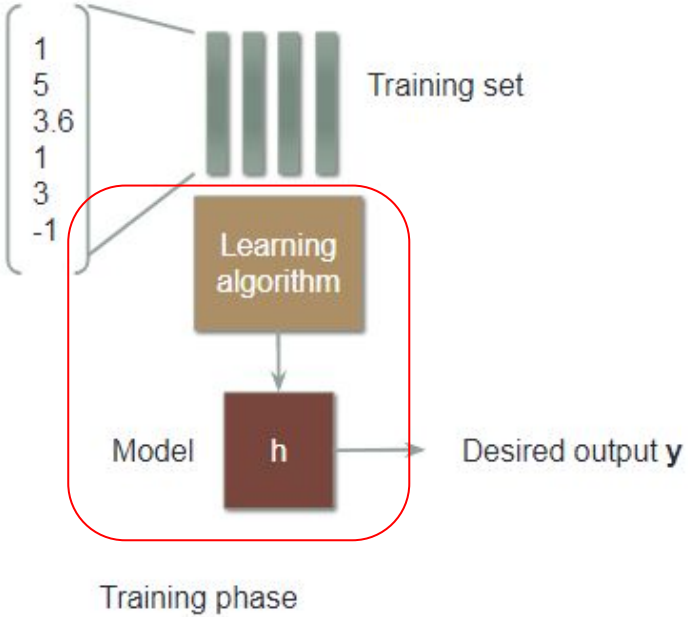
Regression

Rain amount
Age of a person in the picture

Typical workflow



Typical workflow



Agenda

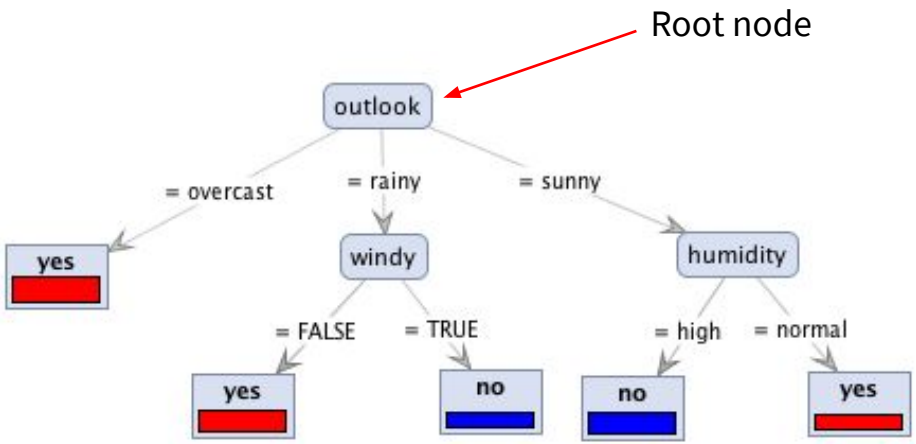
Decision Trees

- Tree Ensemble (Random forest)
- XGBoost

Intro to Neural Networks

Decision Trees

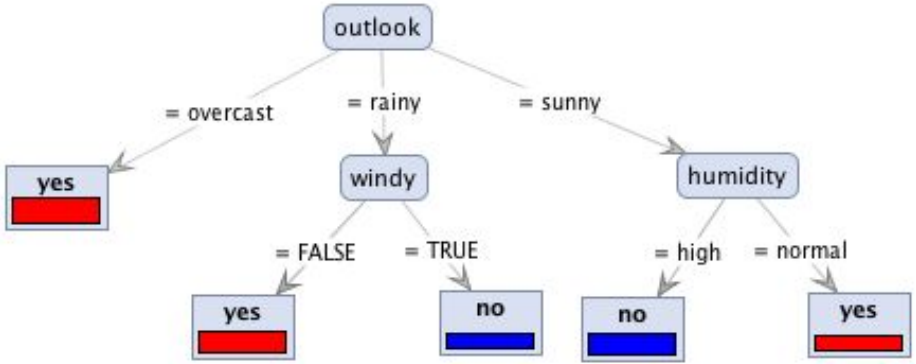
A tree structure that separates data into groups by the feature attributes
Can be used for classification and regression



https://www.xoriant.com/blog/product-engineering/decision-trees-machine-learning-algorithm.html

What's a good decision tree?

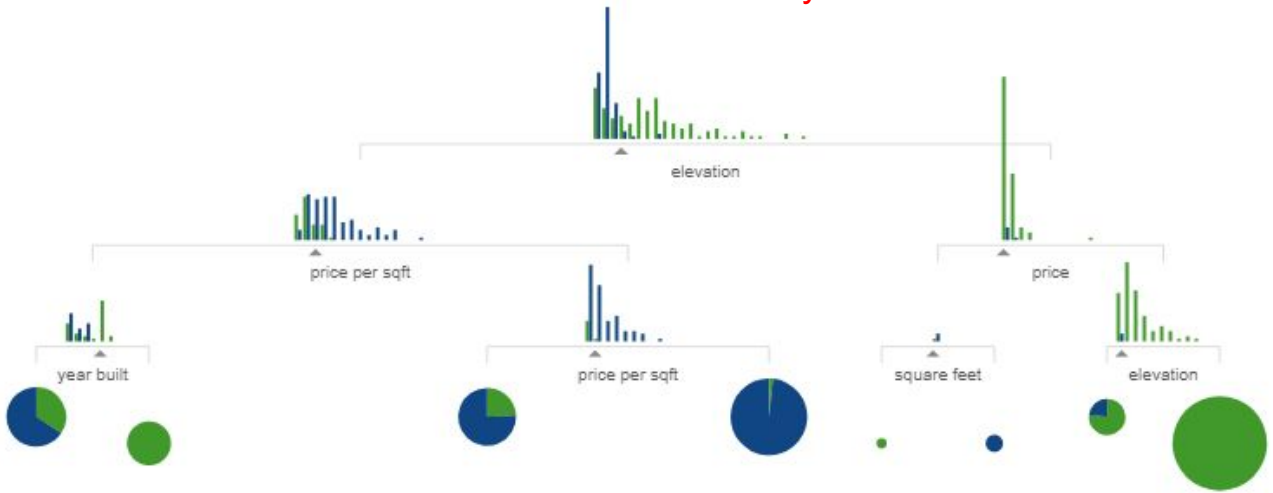
Separates the data nicely
Within a certain budget (smaller trees) - less overfitting



How to create a good decision tree?

Pick the attribute that best separates the classes
Keep doing it until a leave contains entirely one class or you decide it's not worth it to add more nodes

How to determine the best attribute automatically?



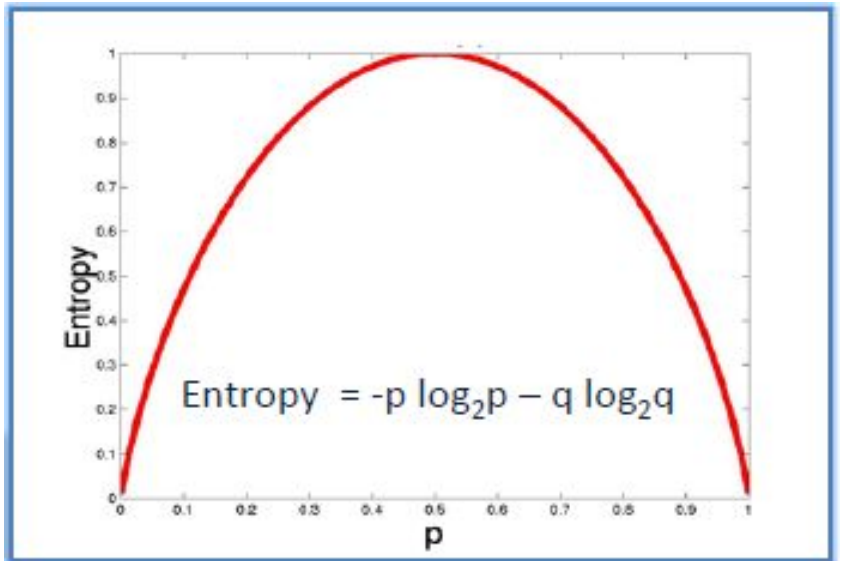
http://www.r2d3.us/visual-intro-to-machine-learning-part-1/

Entropy

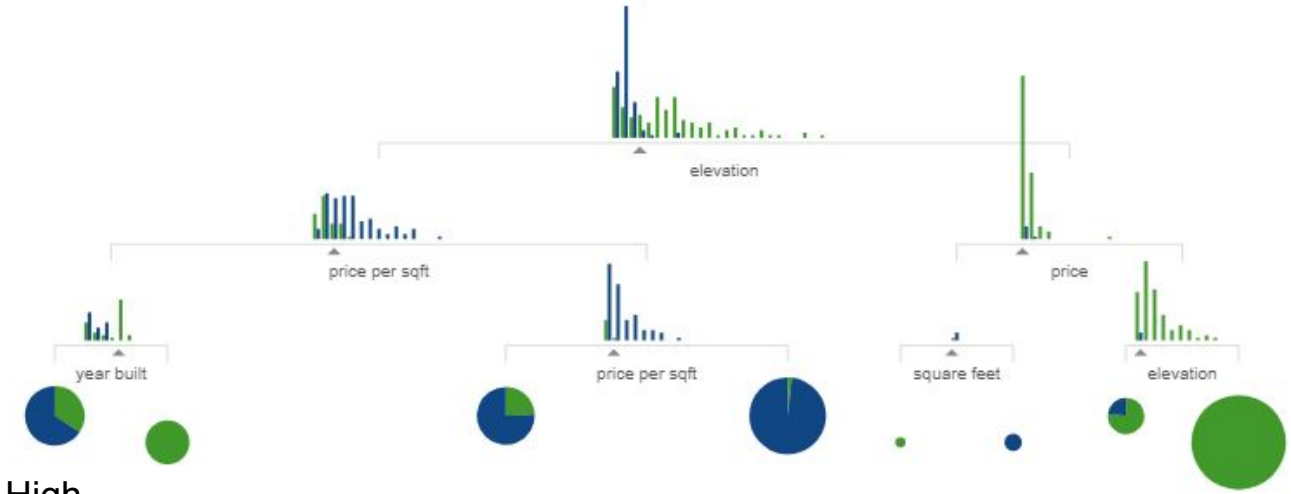
The whole sample space

A measure of randomness

$$E(S) = \sum_c -p(c)\, log_2 p(c) \text{ for } c \in C$$



Entropy



High entropy

Low entropy

Information Gain (IG)

The whole sample space

A measure of how much entropy is reduced

All child nodes by that attribute

$$E = \sum_c -p(c)\, log_2 p(c) \text{ for } c \in C$$

$$IG(parent,child) = E(parent) - \sum_t p(t)E(t) \text{ when } t \in T$$

Probability of going to that child node

Entropy of the child node

Information Gain (IG)

A measure of how much entropy is reduced

The whole sample space

All child nodes by that attribute

$$E = \sum_c -p(c)\, log_2 p(c) \text{ for } c \in C$$

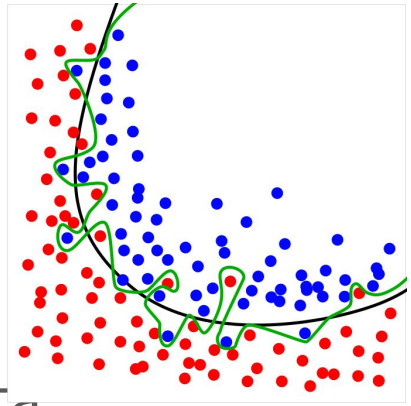
$$IG(parent,child) = E(parent) - \sum_t p(t)E(t) \text{ when } t \in T$$

Find the way to split that maximizes IG

Problems with Decision Trees

Can overfitting easily

Susceptible to noise or badly labelled data



TREE ENSEMBLE MODEL

Tree ensemble model

Ensemble types are models that combine multiple models together

A group of experts voting on a subject

Can lead to less overfitting

Tree ensemble = Multiple trees = Random Forest!

Bagging

Create multiple subsets of data
Each subset is used to train a different tree
The final answer is the average or mode

Less overfitting and can handle mislabeled data

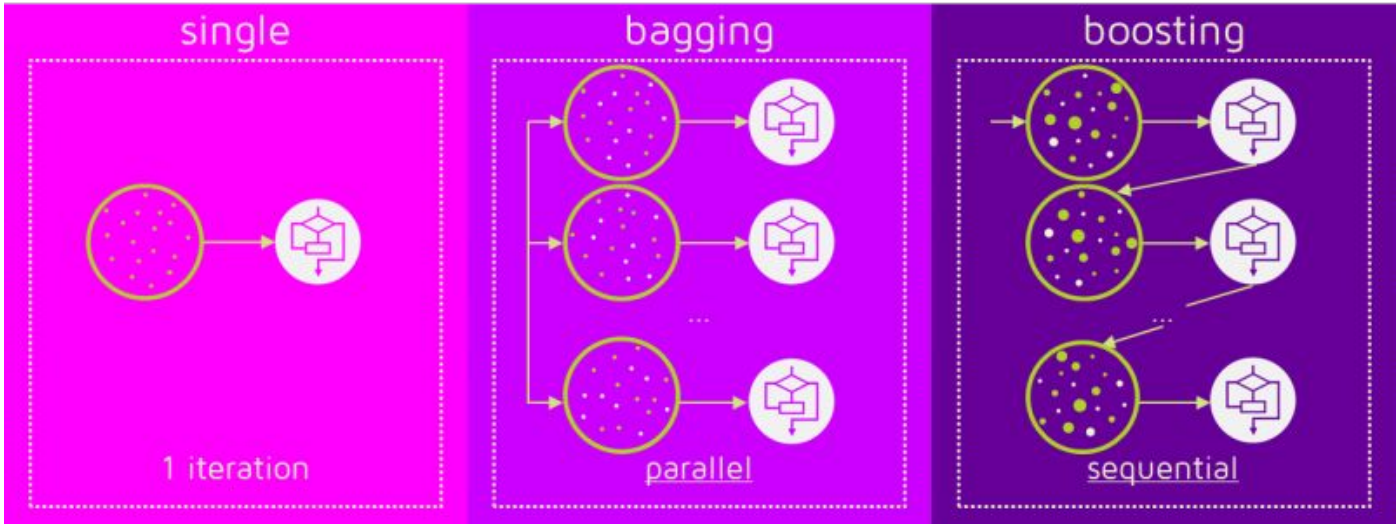
Random Forest

We can also use bagging on features
Each tree has different training samples AND set of features

Boosting vs Bagging

Boosting is another way to create multiple trees

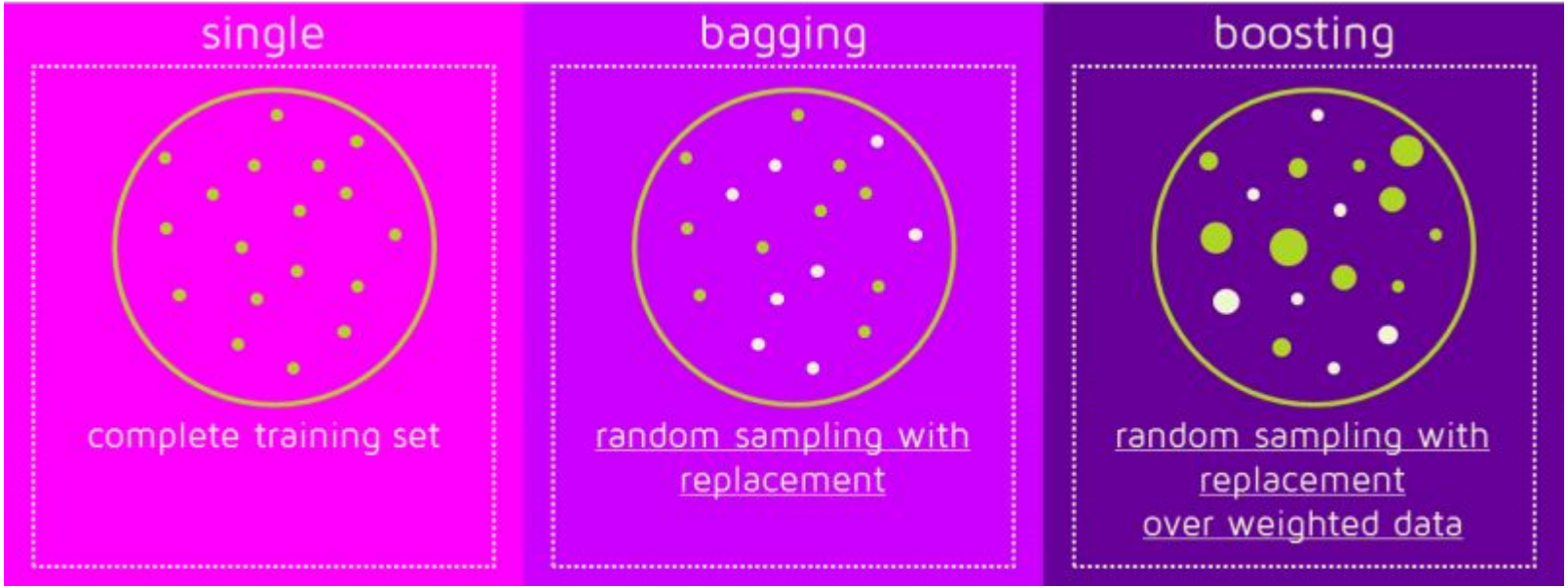
But boosting is iterative, the next tree is based on the errors from the previous trees



https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/

Boosting vs Bagging

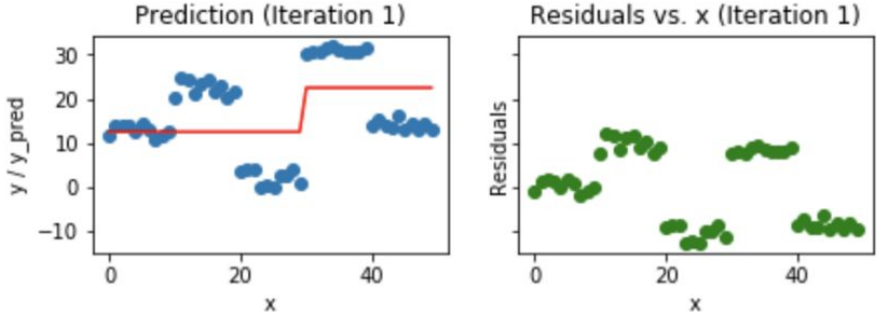
The selection of training data (bagging process) is based on the previous errors



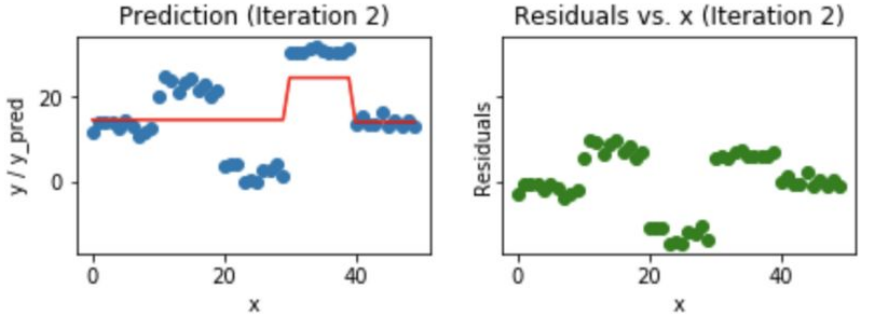
https://quantdare.com/what-is-the-difference-between-bagging-and-boosting/

Gradient Boosting

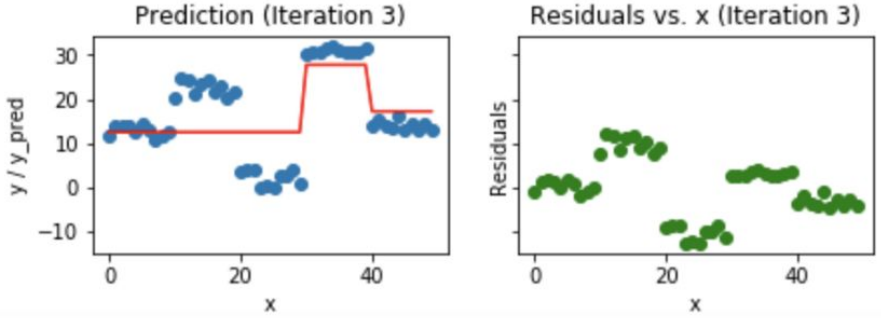
A method of boosting that use gradient-based methods


Prediction (Iteration 1)
y / y_pred
x

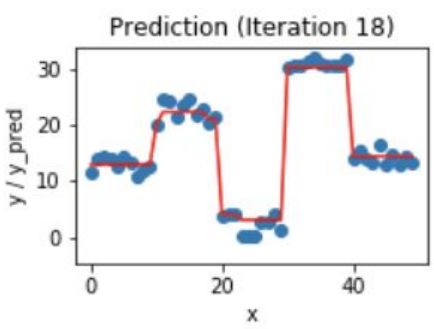
Residuals vs. x (Iteration 1)
Residuals
x

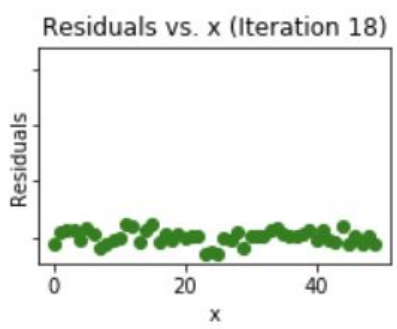

Prediction (Iteration 2)
y / y_pred
x

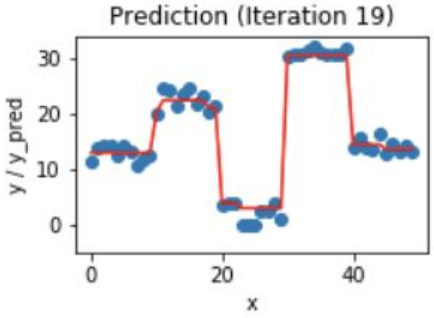
Residuals vs. x (Iteration 2)
Residuals
x

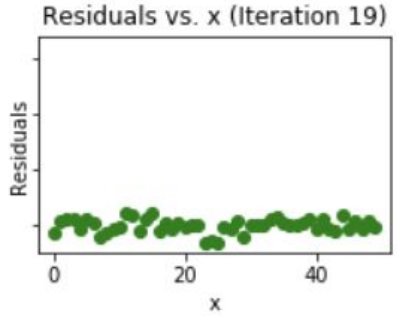

Prediction (Iteration 3)
y / y_pred
x

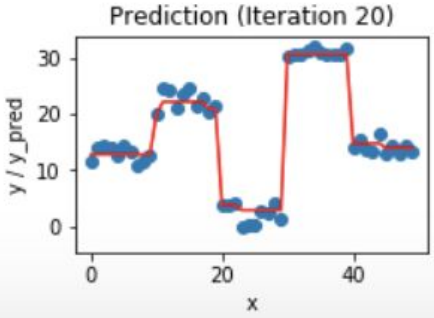
Residuals vs. x (Iteration 3)
Residuals
x

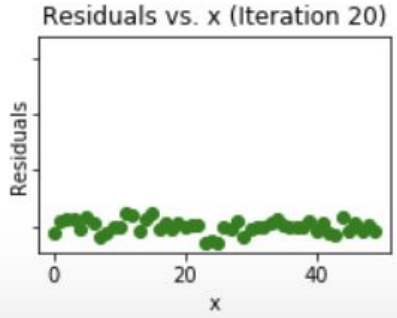

Prediction (Iteration 18)
y / y_pred
x


Residuals vs. x (Iteration 18)
Residuals
x


Prediction (Iteration 19)
y / y_pred
x


Residuals vs. x (Iteration 19)
Residuals
x

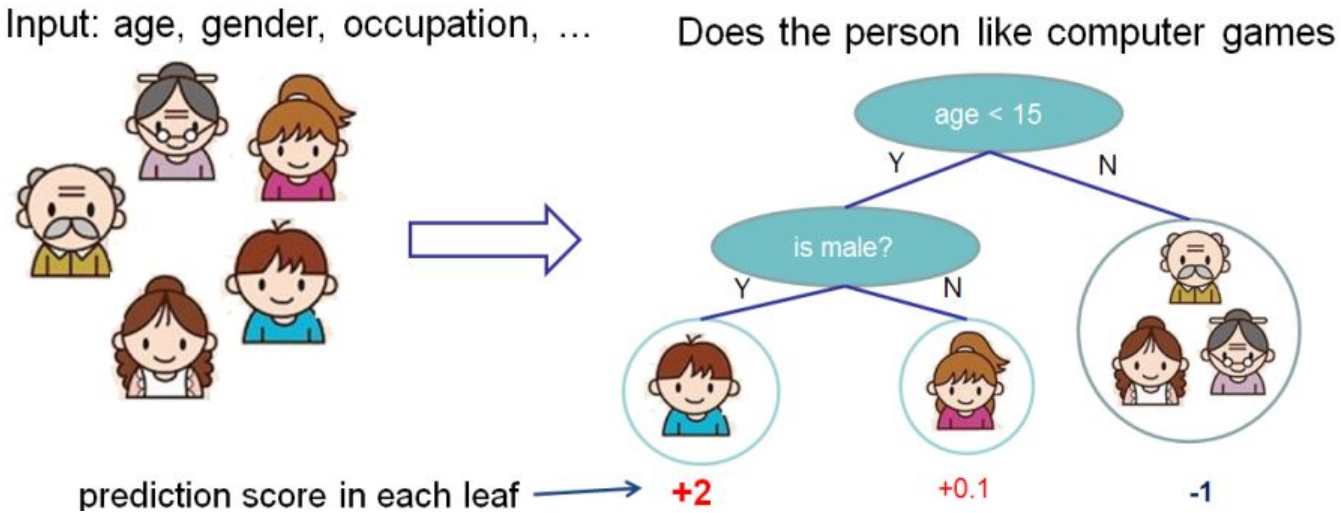

Prediction (Iteration 20)
y / y_pred
x


Residuals vs. x (Iteration 20)
Residuals
x

Tree Gradient Boosting

Similar to decision tree

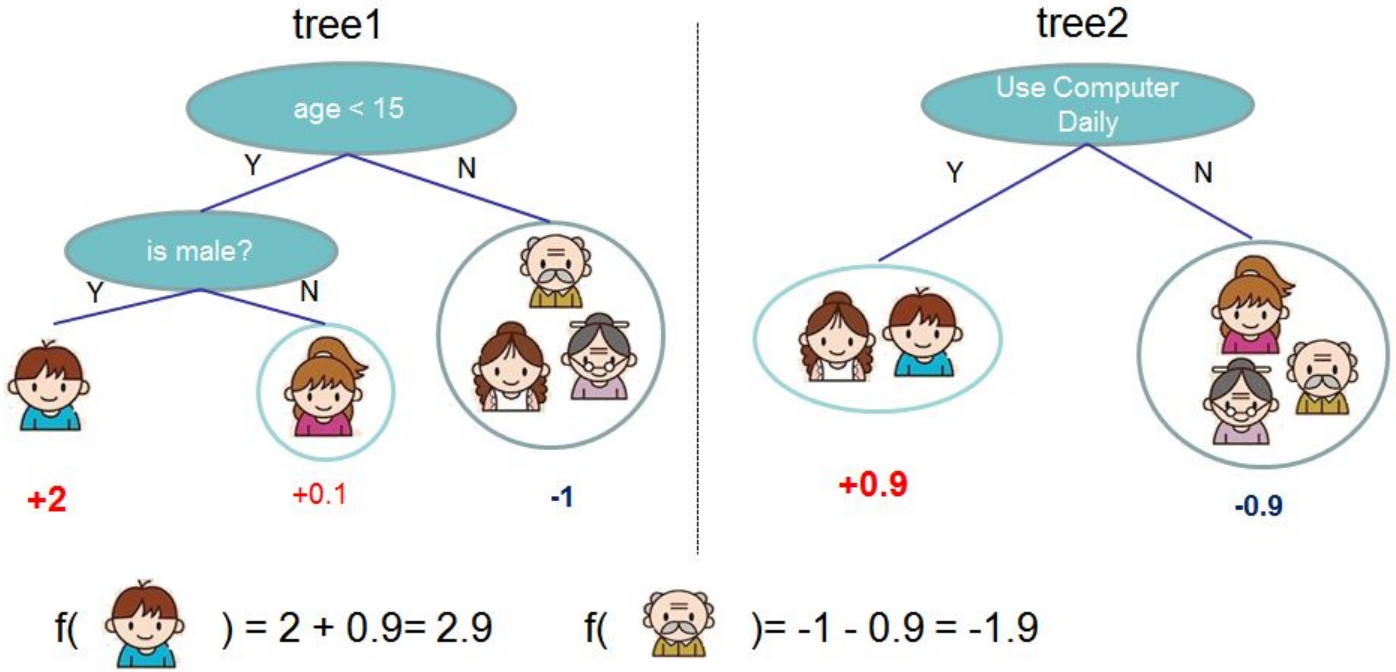
Different is the leaf node contains a score



https://xgboost.readthedocs.io/en/latest/tutorials/model.html

Tree Gradient Boosting

Multiple trees with different rules. The subsequent tree try to correct the errors from the previous trees



Extreme Gradient Boosting (XGBoost)

Super popular Tree Boosting library

Highly recommended for spreadsheets type of input data

```
model = XGBClassifier(
    n_jobs=16,
    n_estimators=400,
    max_depth=4,
    objective="binary:logistic",
    learning_rate=0.07,
    subsample=0.9,
    min_child_weight=6,
    colsample_bytree=.9,
    scale_pos_weight=0.8,
    gamma=8,
    reg_alpha=6,
    reg_lambda=1.3)
```

Objective <- type of problem you want to solve

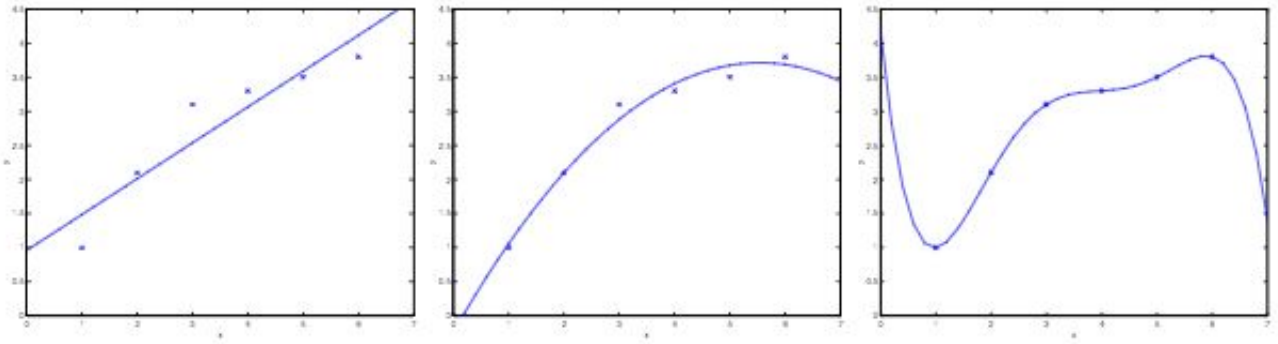
Max_depth <- max depth of tree, higher more overfitting
Min_child_weight <- how strong must the leave be, higher less overftting
Gamma <- when to stop splitting early

Reg_alpha, reg_lambda <- reduce overftting

Scale_pos_weight <- weight for class imbalance

https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

Notes on overfitting



Using too complex models can lead to overftting

If the curve fits too perfectly and doesn't generalize well to unseen data, it's overfitting
For the opposite case, having not enough parameters to model the data is called underfitting

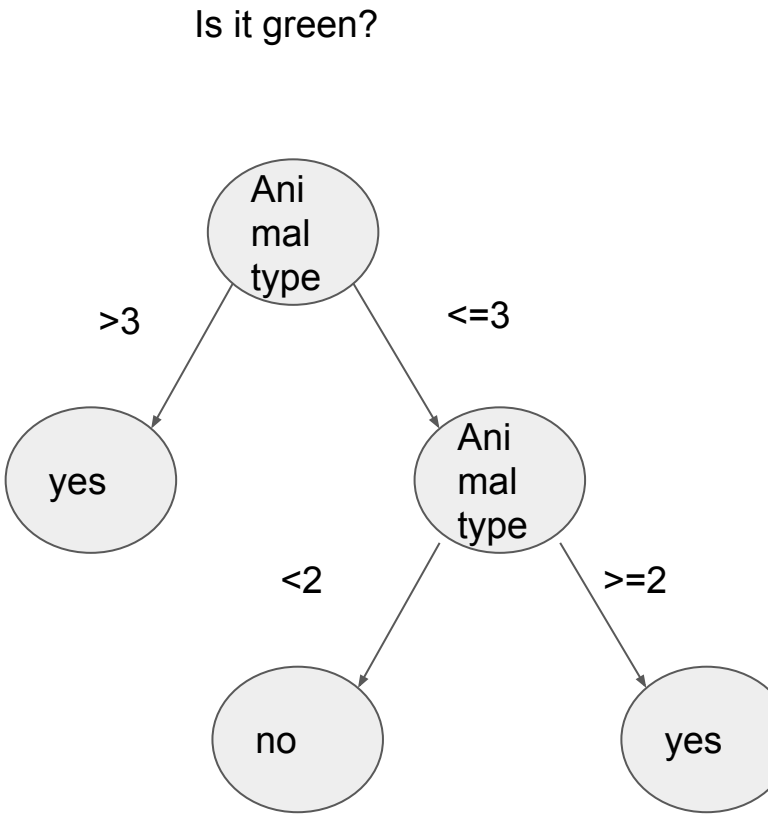
Notes on feature encoding

Categorical features does not mean anything

Type of animal

Animal type =
- 1 if mouse
- 2 if bird
- 3 if dog
- 4 if insect

Makes it hard to do decision trees



One hot encoding

Split categorical features into multiple binary features

Type of animal (as one hot)

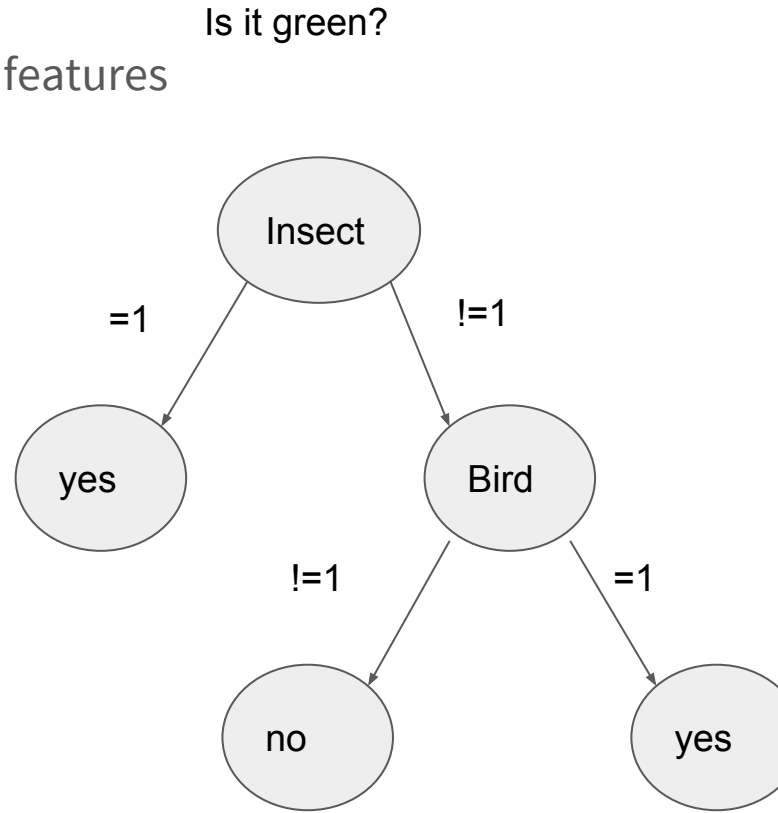
Is_mouse = (0,1)

Is_bird = (0,1)

Is_dog = (0,1)

Is_insect = (0,1)

Doesn’t change much



Target encoding

Encode information by looking at how the feature correlates with the final answer

Encoded feature = P(answer = yes| feature value)

Animal type =

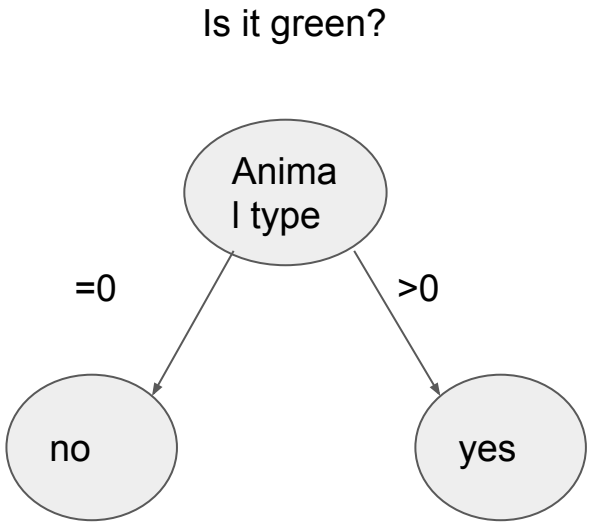
0 if mouse

0.3 if bird

0 if dog

0.5 if insect

Need some further smoothing to improve this.



https://dl.acm.org/citation.cfm?id=507538

Neural Networks

Agenda

Fully connected networks

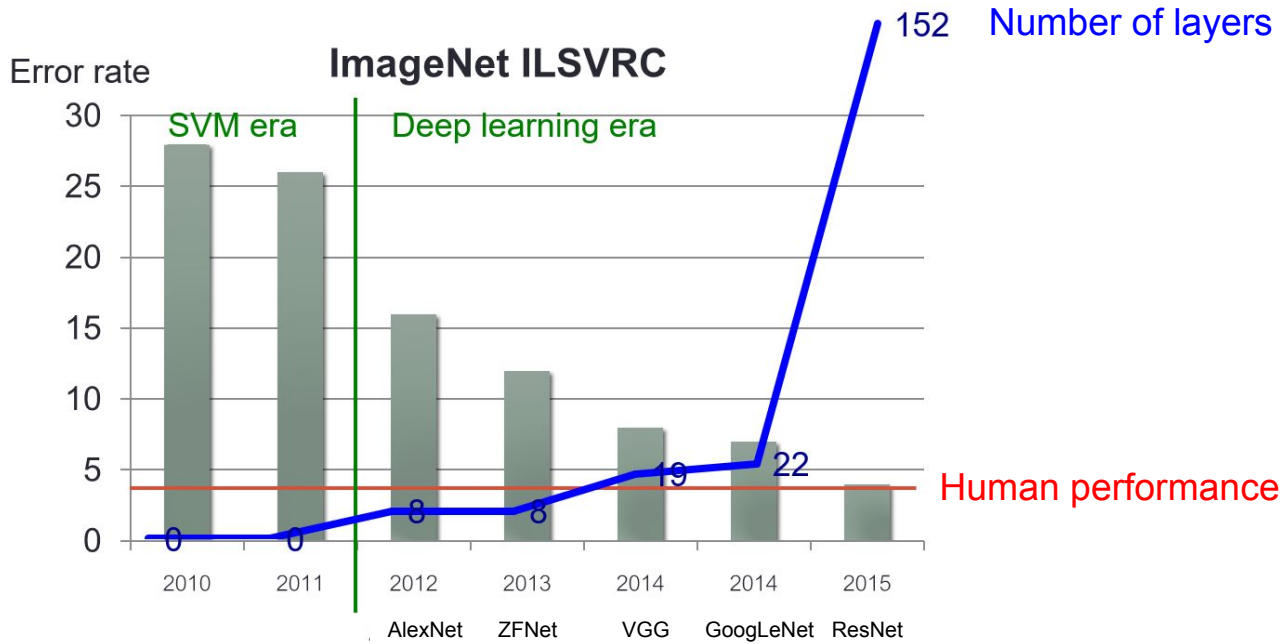
Convolutional neural networks

Deep learning

Artificial Neural Networks rebranded

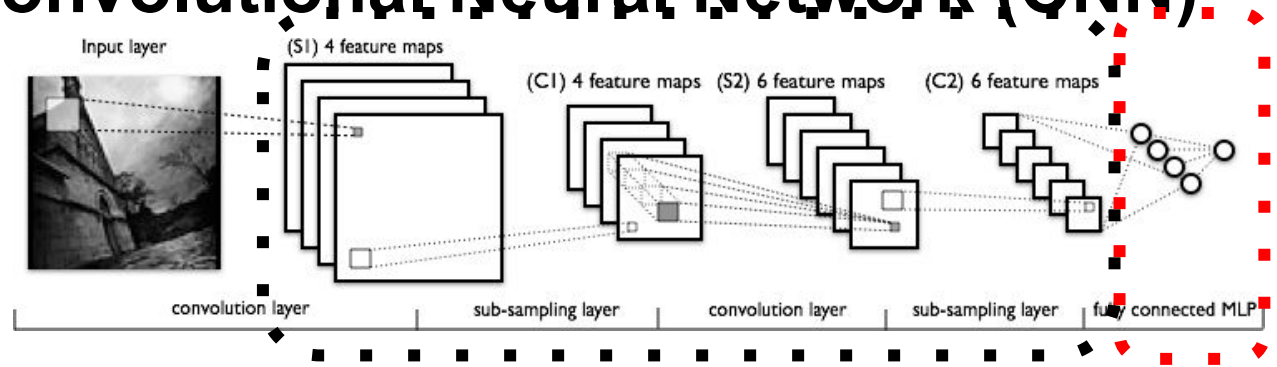
- Deeper models
- Bigger data
- Larger compute

A brief history of ImageNet



Olga Russakovsky, ImageNet Large Scale Visual Recognition Challenge, 2014 https://arxiv.org/abs/1409.0575

Convolutional Neural Network (CNN)

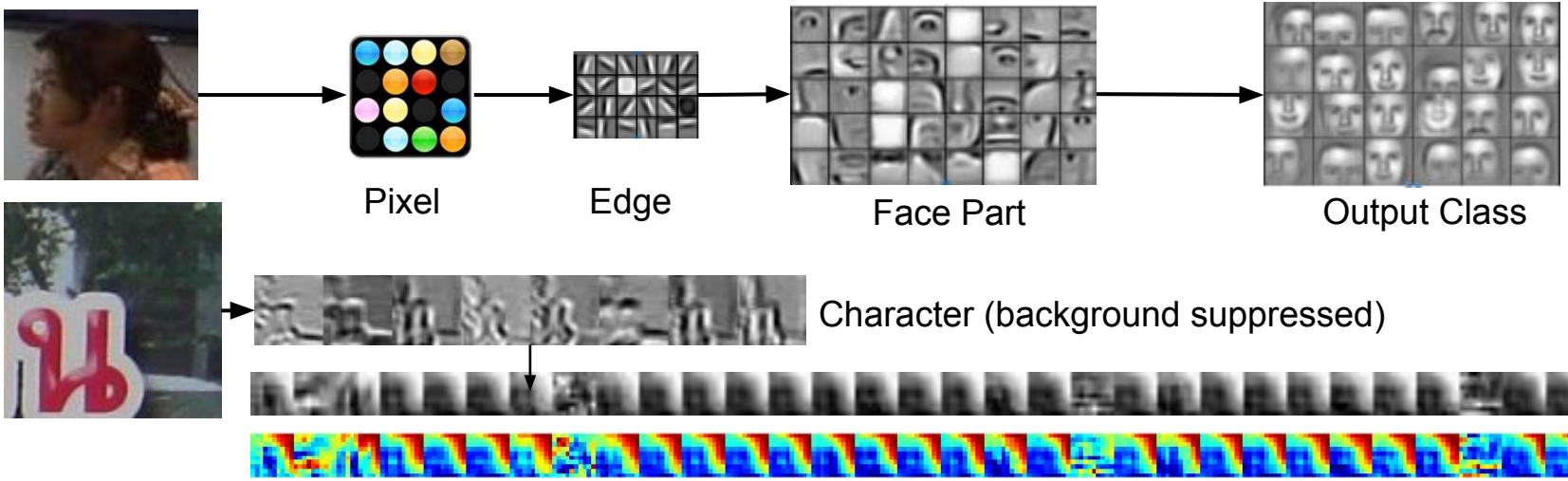


Trainable Feature Extractor Layer **Classification Layer**

Traditional CV: fixed representation but learnable classifier
Deep learning: jointly learn representations with the classifier

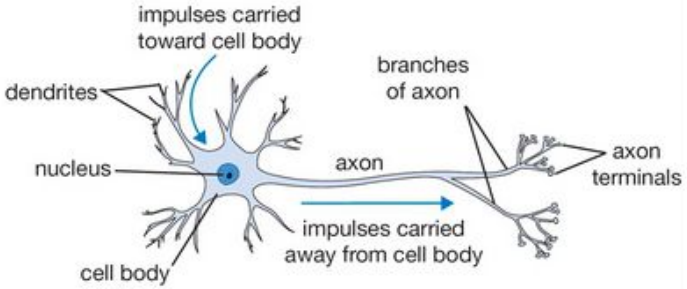
Convolutional Neural Network (CNN)

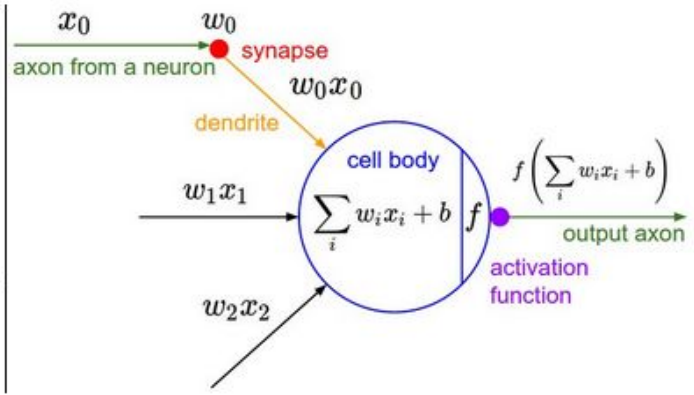
- Hierarchy of representations with increasing level of abstraction
- Each stage is a kind of trainable feature transform
- Image recognition: Pixel →edge → texture → part → object



Fully connected networks

- Many names: feedforward networks or deep neural networks or multilayer perceptron or artificial neural networks
- Composed of multiple neurons

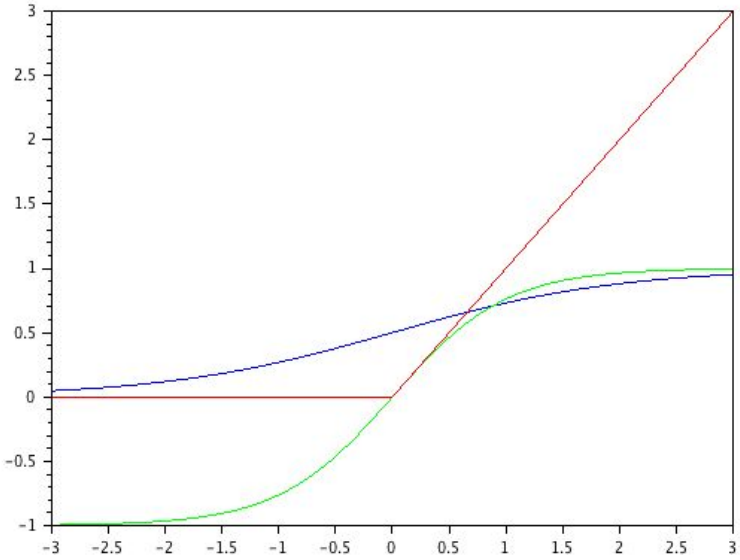




http://cs231n.github.io/neural-networks-1/

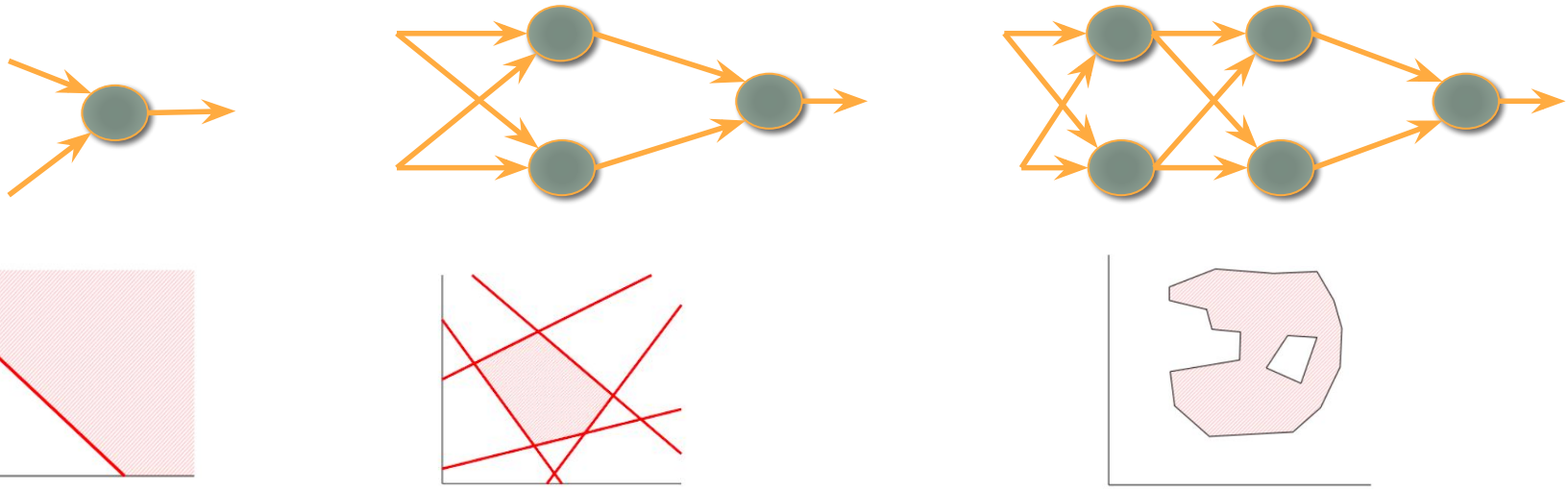
Non-linearity (Activation Function)

- The non-linearity is important in order to stack neurons
- If linear, a multi layered network can be collapsed to a single layer (by just multiplying weights together)
  - Sigmoid or logistic function
  - *T*anh
  - Rectified Linear Unit (ReLU)
- Most popular is ReLU and its variants (Fast to train, and more stable)



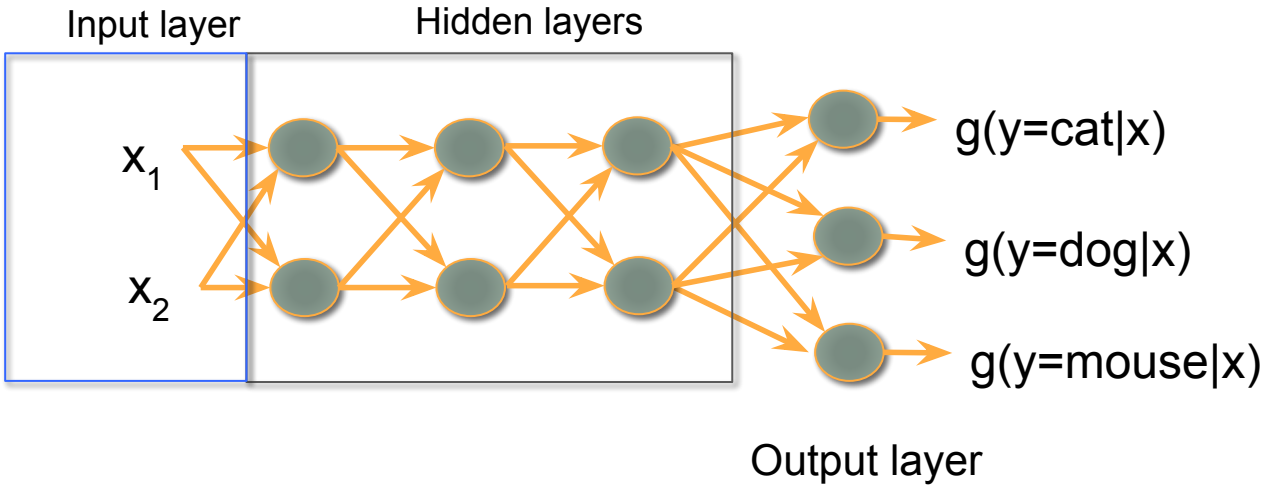
Combining neurons

- Each neuron splits the feature space with a hyperplane
- Stacking neuron creates more complicated decision boundaries
- More powerful but prone to overfitting



Terminology

Deep in Deep neural networks means many hidden layers

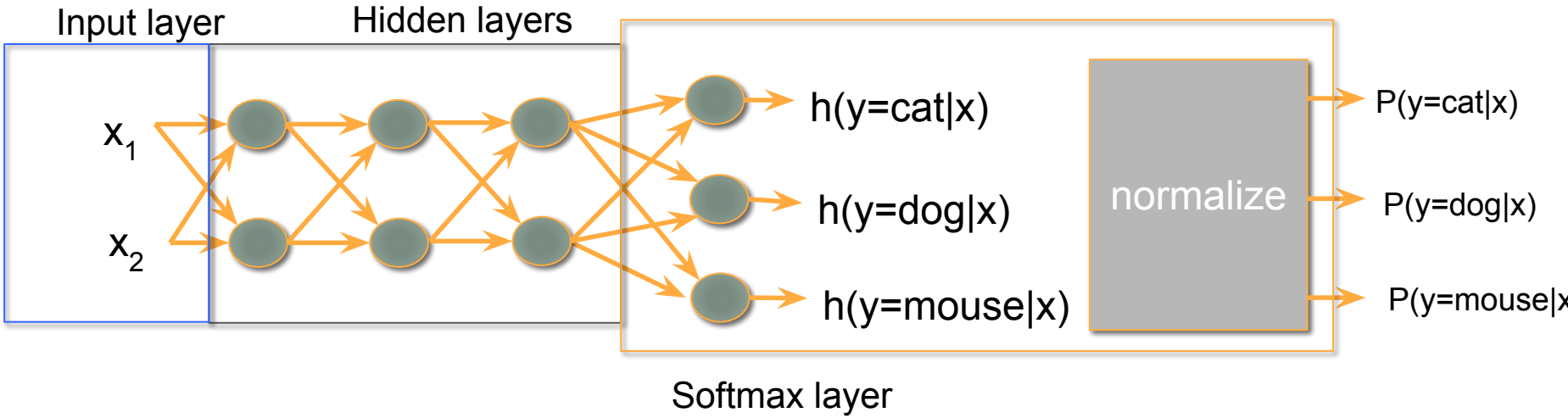


Function that map input to scores of being a particular class

Output layer – Softmax layer

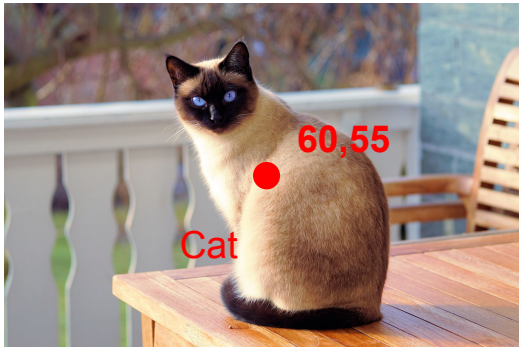
- We usually wants the output to mimic a probability function (0<=P<=1,sums to 1)
- Add a normalization layer called "softmax"

$$P(y=j|x) = \frac{e^{h(y=j|x)}}{\sum_y e^{h(y|x)}}$$



Objective function (Loss function)

- Can be any function that summarizes the performance into a single number
  - Cross entropy
    - Categorical tasks (cat vs dog, age range)
  - Sum of *squared* errors
    - Numerical tasks (location of cat, age)



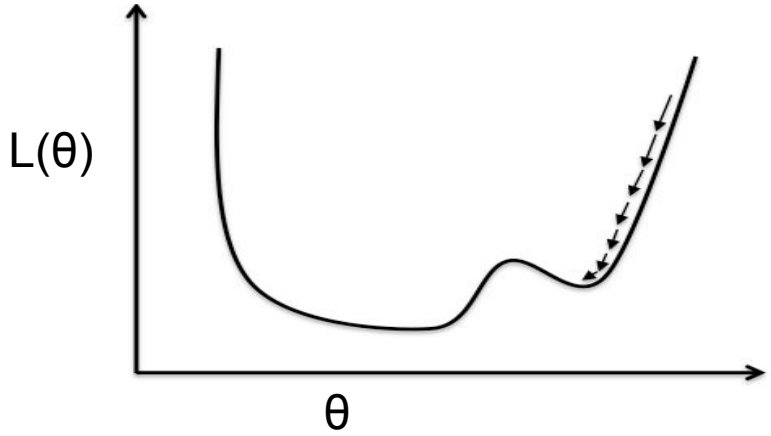
$q_1(x, \theta)$

$q_n(x, \theta)$

	Output type	
	Categorical	Numerical
	$y_1 = 1$	$y_x = 60$
	$y_n = 0$	$y_y = 55$

Minimization using gradient descent

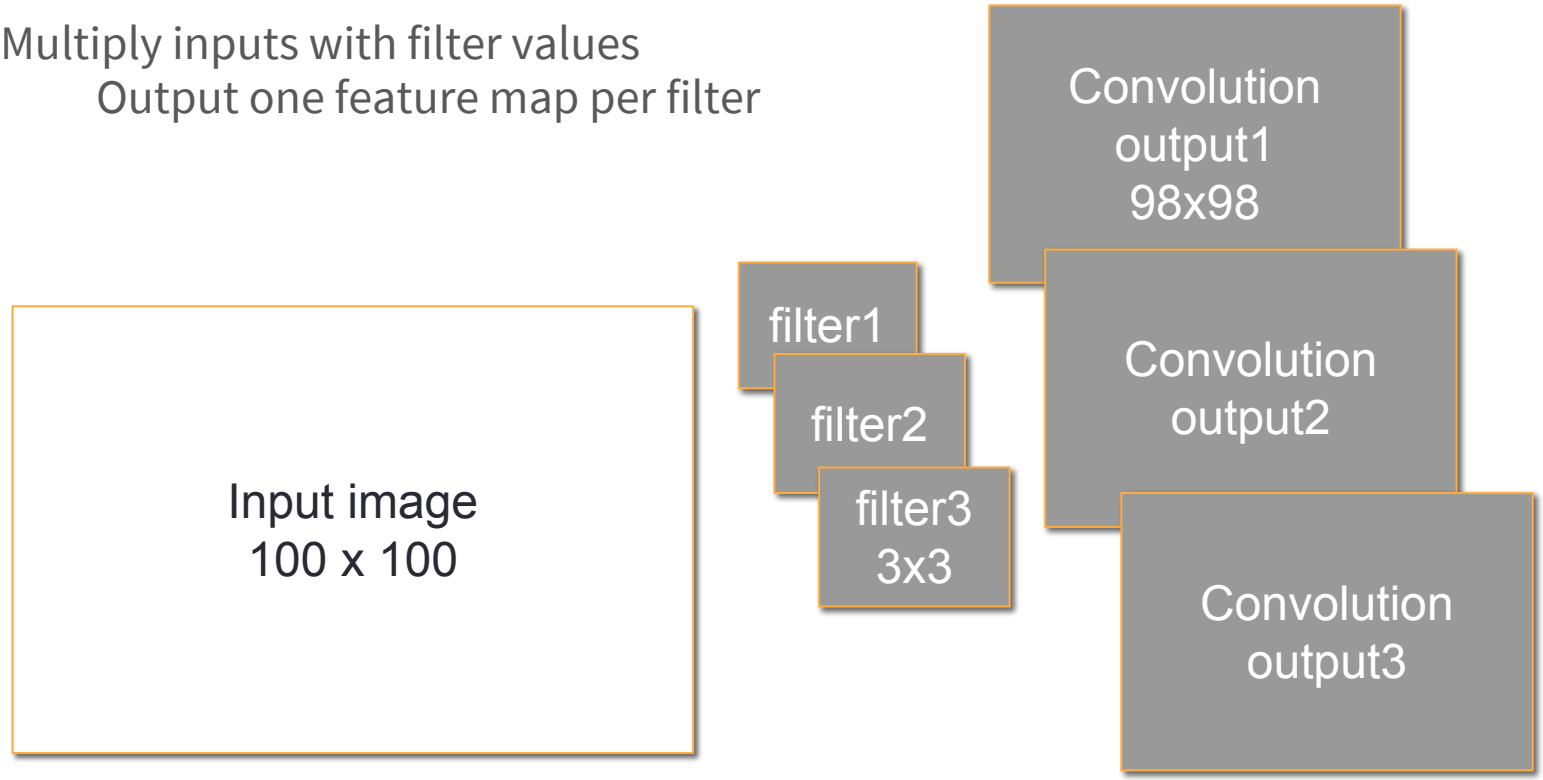
- We want to minimize L with respect to θ (weights and biases)
  - Differentiate with respect to θ
  - Gradients passes through the network by Back Propagation



Convolutional filters

Multiply inputs with filter values

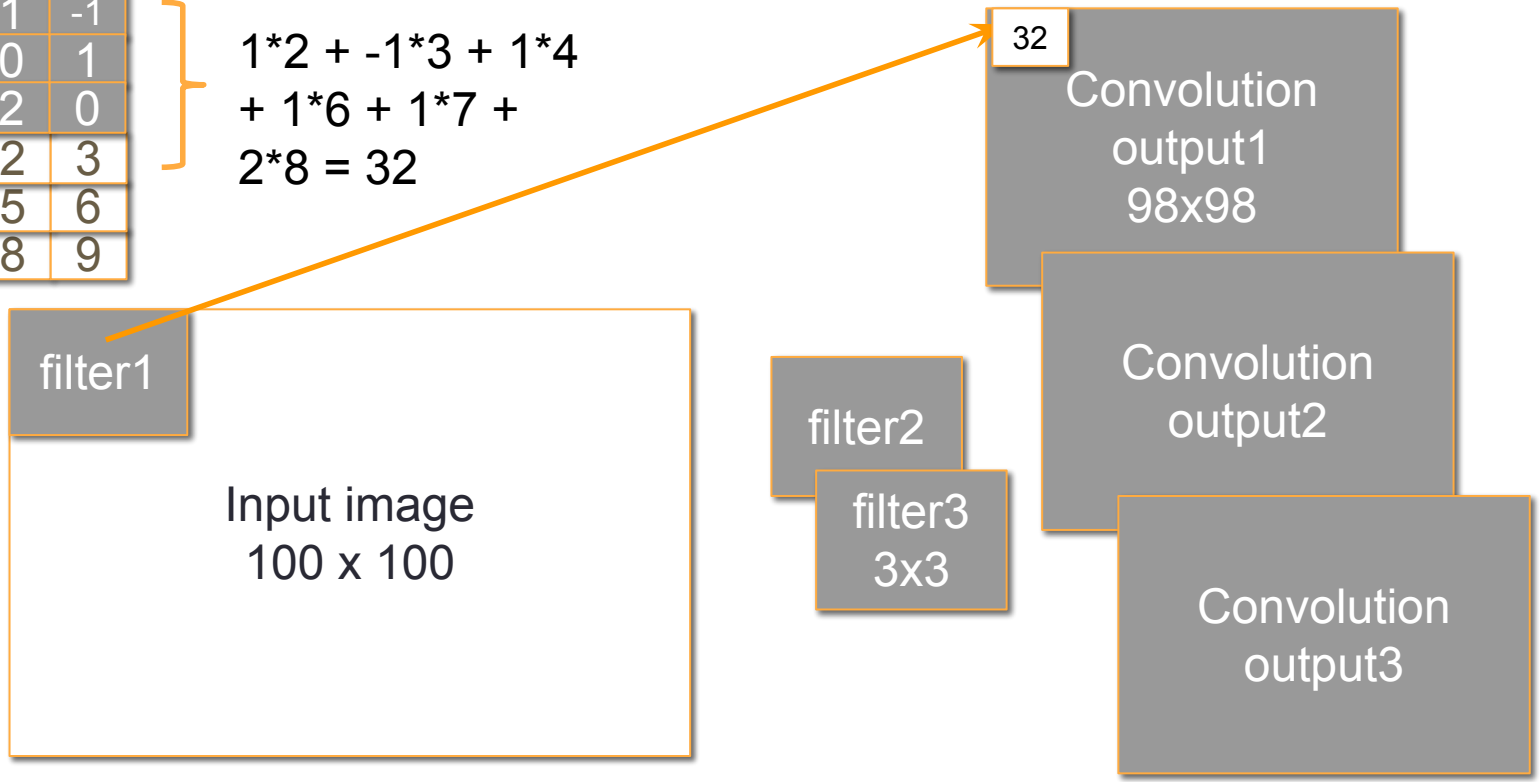
Output one feature map per filter



Convolutional filters

0	1	-1
1	0	1
1	2	0
1	2	3
4	5	6
7	8	9

1*2 + -1*3 + 1*4
+ 1*6 + 1*7 +
2*8 = 32



Convolutional filters

0	1	-1
1	0	1
1	2	0
2	3	1
5	6	3
8	9	8

1*3 + -1*1 + 1*5
+ 1*3 + 1*8 +
2*9 = 36

Stride of 1

33 36

Convolution output1 98x98

Convolution output2

Convolution output3

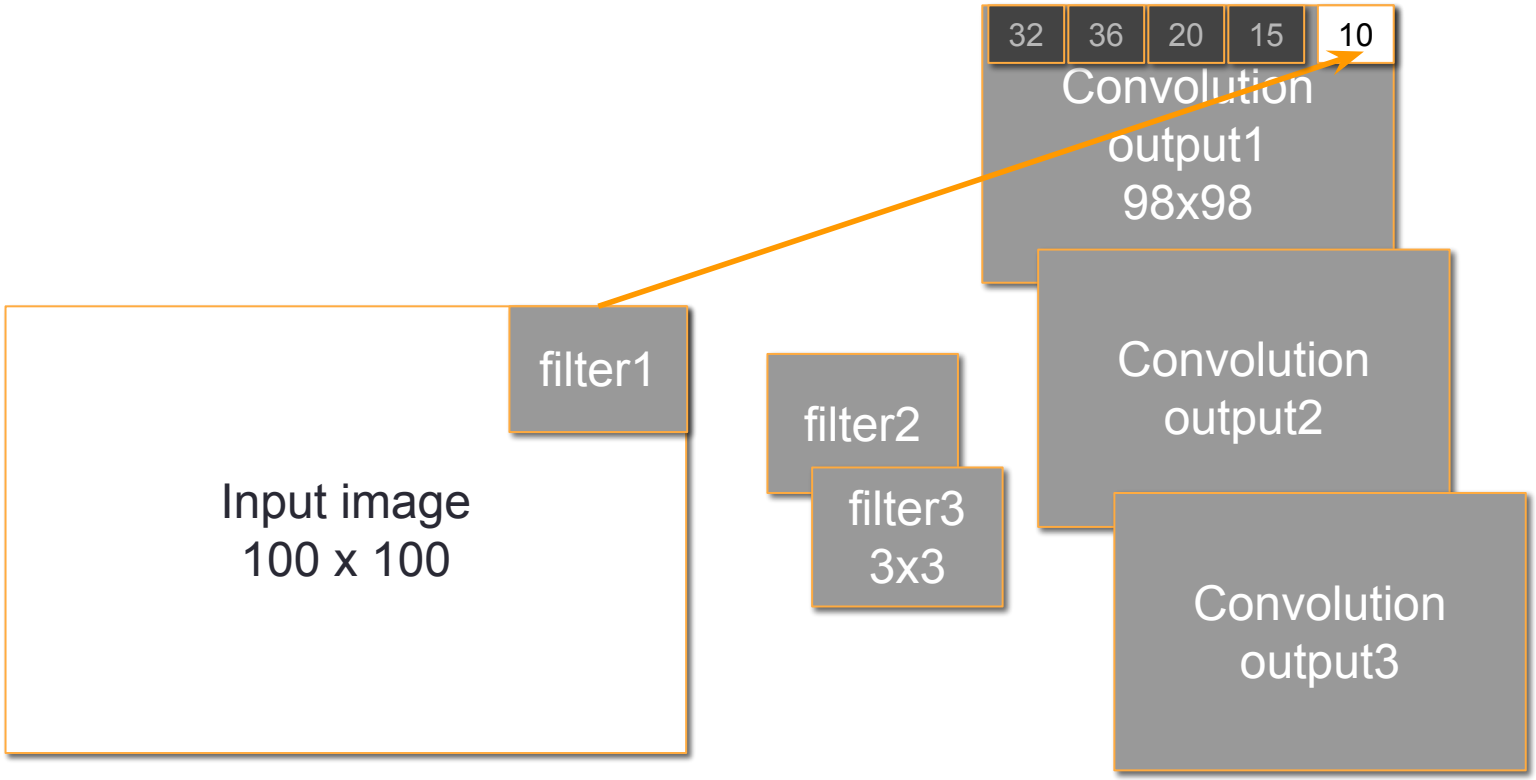
filter1

filter2

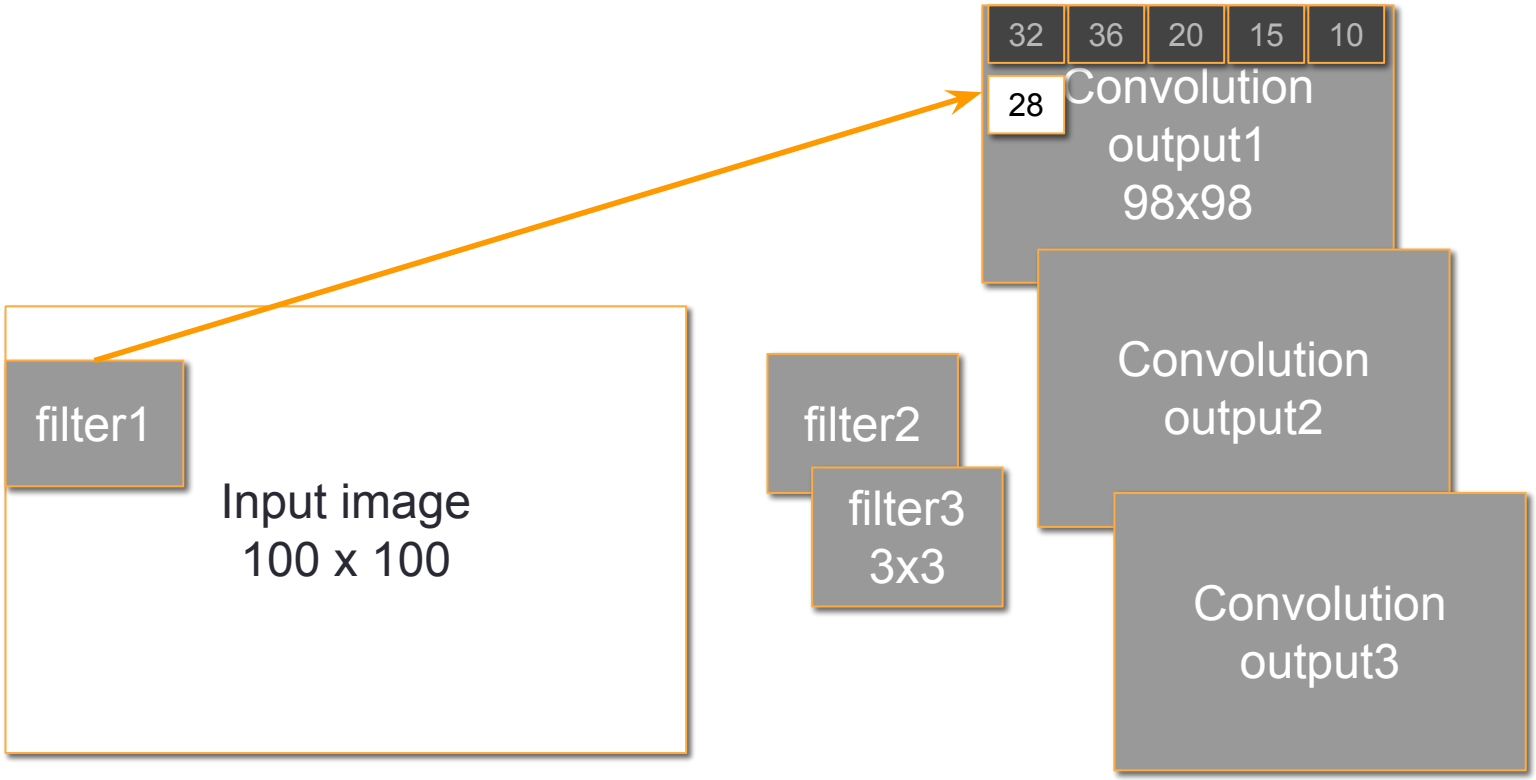
filter3 3x3

Input image
100 x 100

Convolutional filters

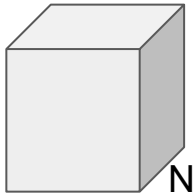


Convolutional filters



Convolutional filters

N filters means N feature maps
You get a 3 dimensional output



32	36	20	15	10
28	72	0	12	50
32	36	18	9	2
17	6	2	17	1

16

filter2

Input image
100 x 100

filter1

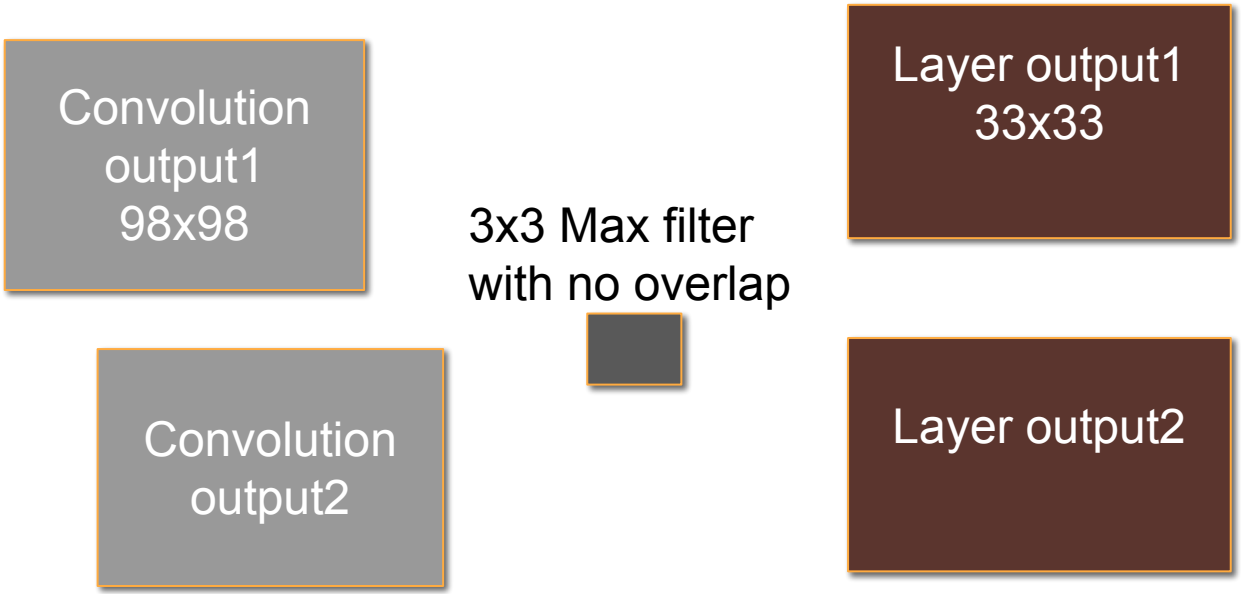
filter3
3x3

Convolution
output2

Convolution
output3

Pooling/subsampling

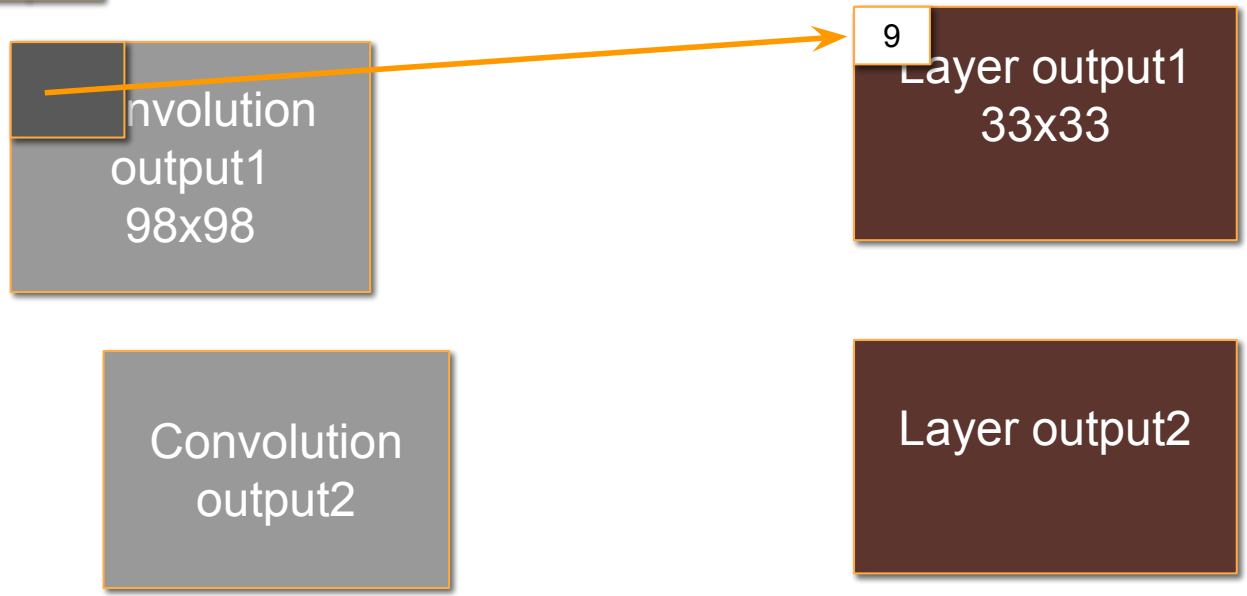
Reduce dimension of the feature maps



Pooling/subsampling

1	2	3
4	5	6
7	8	9

Max = 9

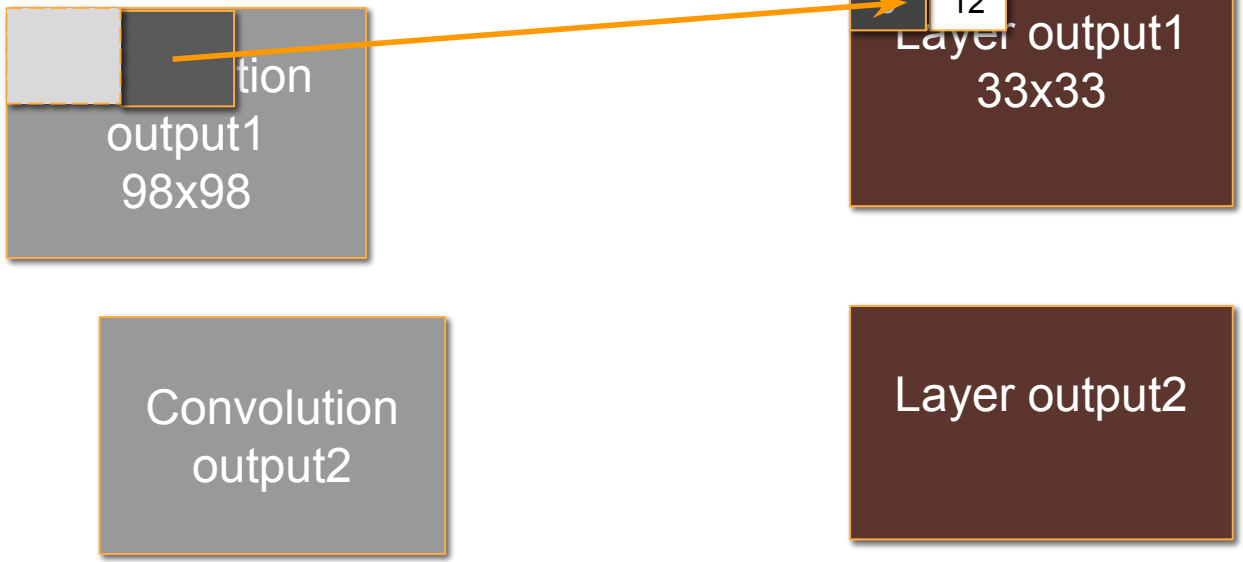


Pooling/subsampling

5	2	1
5	7	1
9	5	12

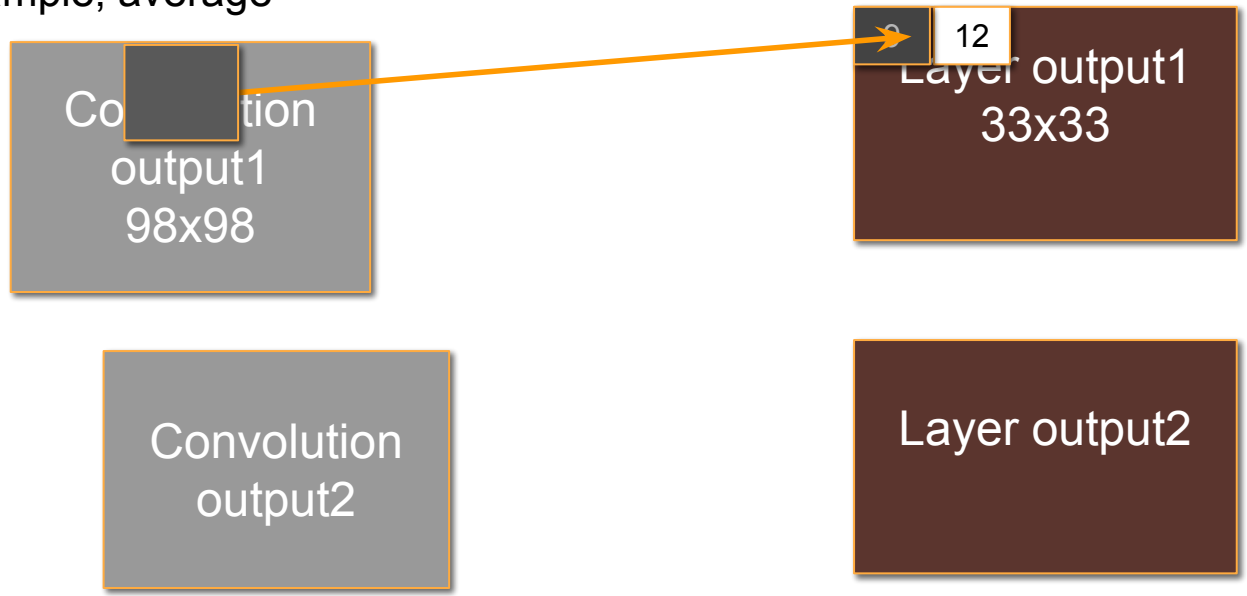
Max = 12

Stride = 3



Pooling/subsampling

Can use other functions besides max
Example, average



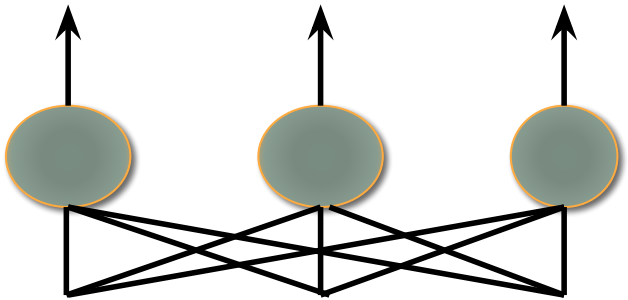
Dropout

A regularization technique for reducing overfitting

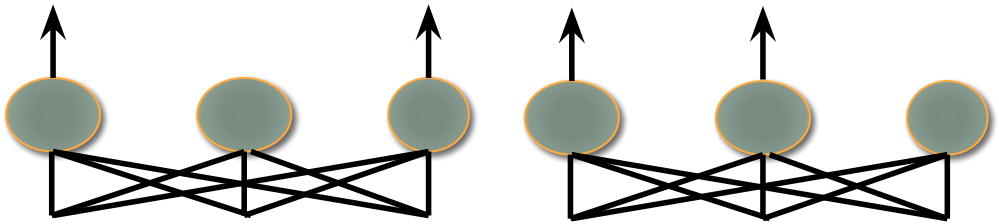
Randomly turn off different subset of neurons during training

- Network no longer depend on any particular neuron
- Force the model to have redundancy – robust to any corruption in input data

A form of performing model averaging (ensemble of experts)



Model

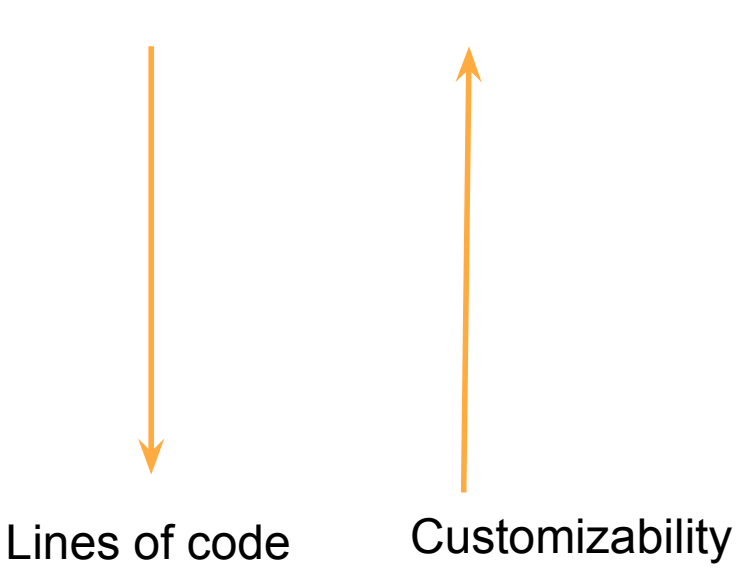


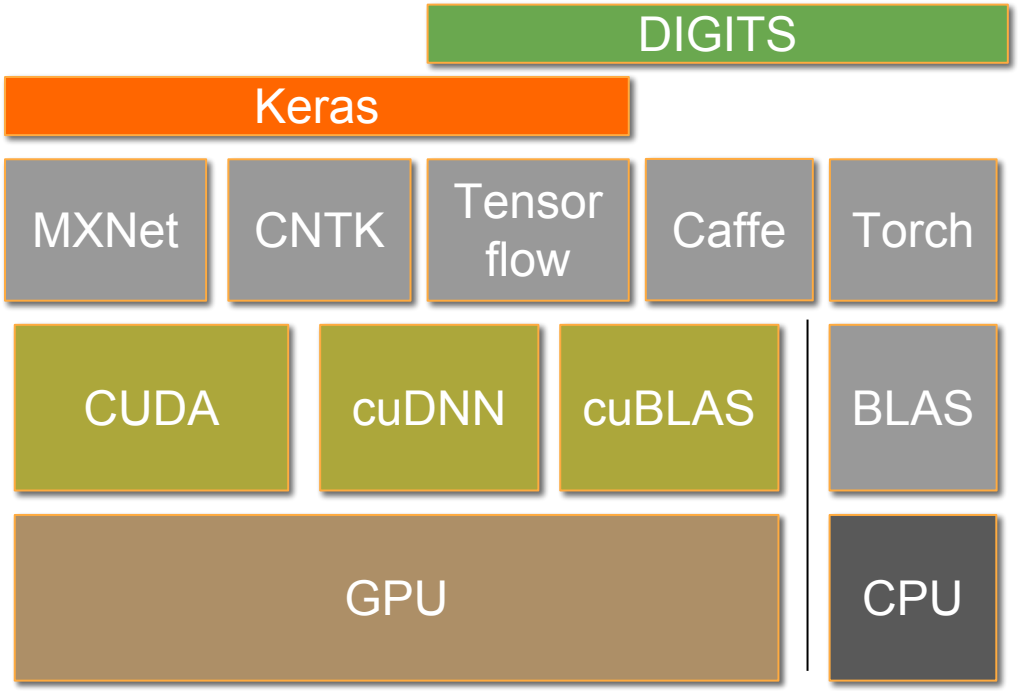
Dropout rate of ⅓

Hinton, Geoffrey "Improving neural networks by preventing co-adaptation of feature detectors" 2012

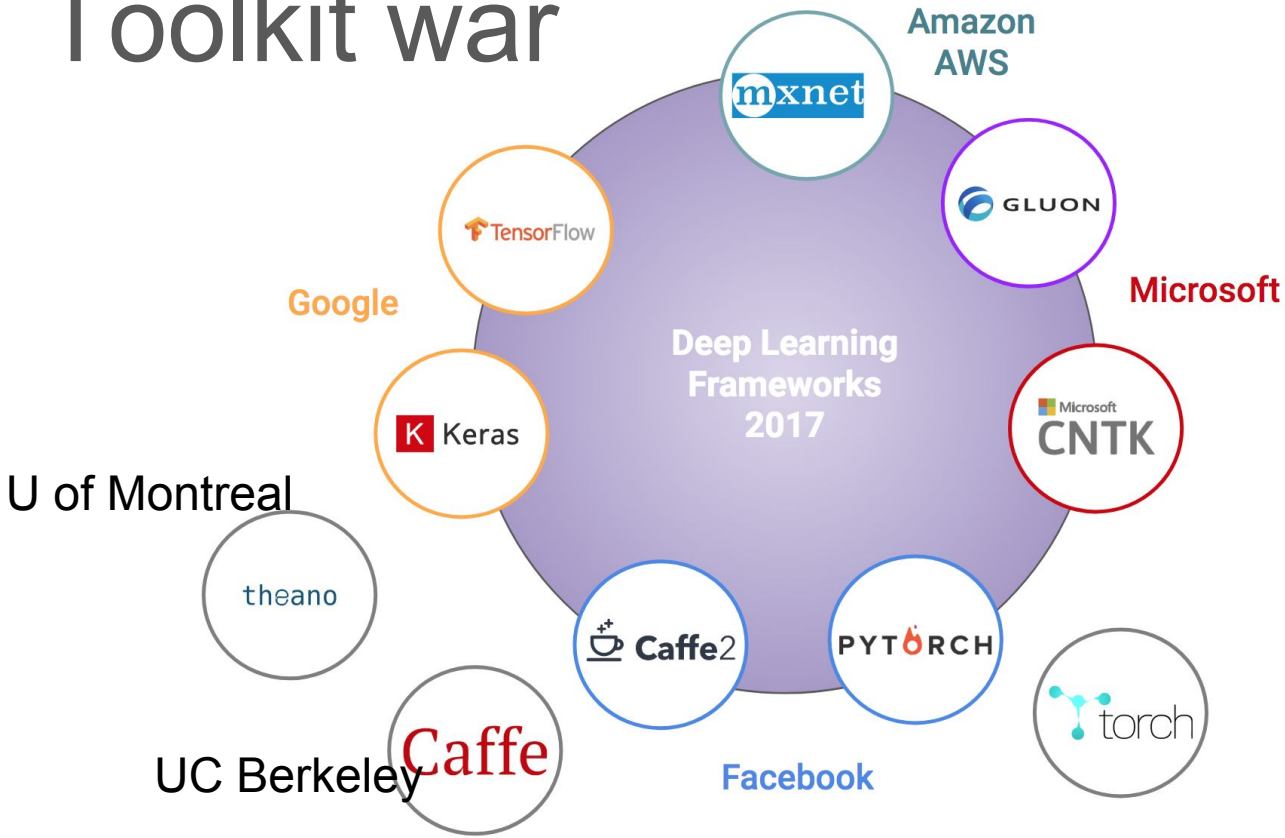
What toolkit

Tradeoff between customizability and ease of use





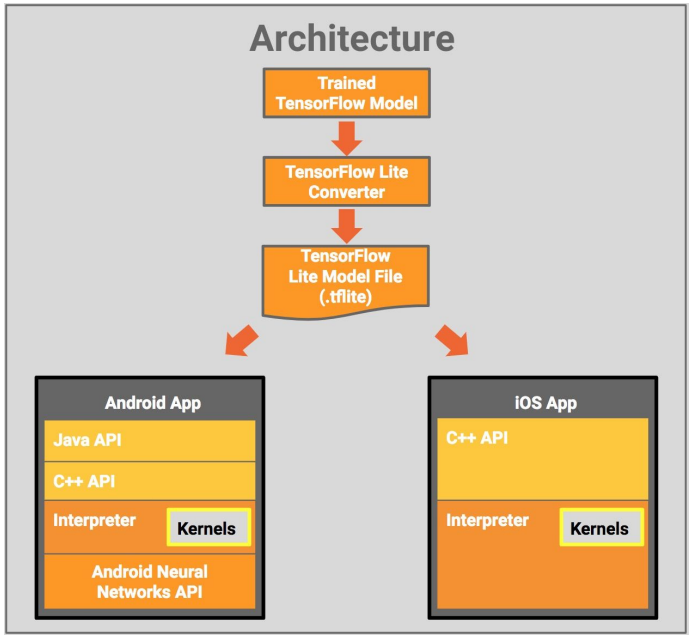
Toolkit war



https://towardsdatascience.com/battle-of-the-deep-learning-frameworks-part-i-cff0e3841750

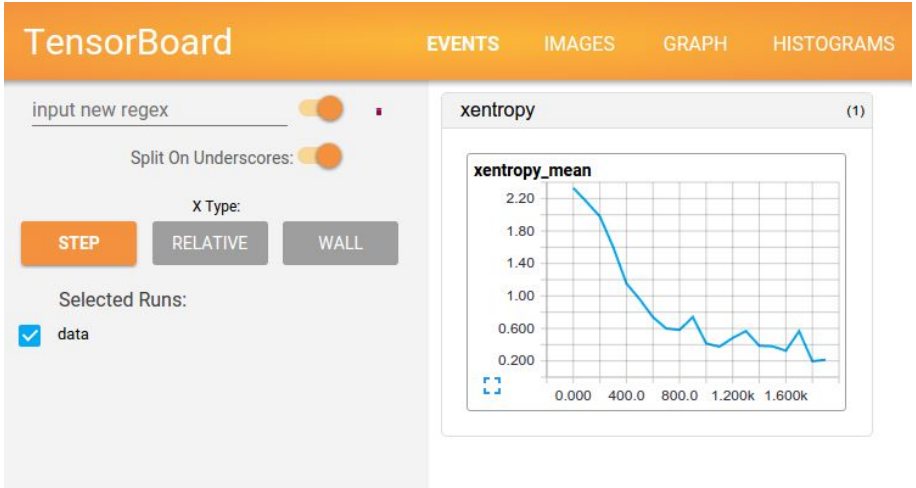
Which?

- Easiest to use and play with deep learning: Keras
- Easiest to use and tweak: pytorch
- Easiest to deploy: tensorflow
  - Tensorflow lite for mobile
  - *TensorRT support*
- Best tools: TensorFlow
  - *Tensorboard*



Which?

- Easiest to use and play with deep learning: Keras
- Easiest to use and tweak: pytorch
- Easiest to deploy: tensorflow
  - Tensorflow lite for mobile
  - *TensorRT support*
- Best tools: TensorFlow
  - *Tensorboard*
- Community: TensorFlow



Keras steps

- Define the network
- Compile the network
- Fit the network

```
def get_feedforward_nn():
  input1 = Input(shape=(21,))
  x = Dense(100, activation='relu')(input1)
  x = Dense(100, activation='relu')(x)
  x = Dense(100, activation='relu')(x)
  out = Dense(1, activation='sigmoid')(x)

  model = Model(inputs=input1, outputs=out)
  model.compile(optimizer=Adam(),
                loss='binary_crossentropy',
                metrics=['acc'])
  return model
```

```
  model_feedforward_nn.fit(x_train_char, y_train, epochs=epochs, batch_size=batch_size, verbose=ve
rbose,
                           callbacks=callbacks_list_feedforward_nn,
                           validation_data=(x_val_char, y_val))
```

Keras is easy!

Dense [source]

```
keras.layers.Dense(units, activation=None, use_bias=True, kernel_initializer='glorot_uniform', bias_initializ
```

Just your regular densely-connected NN layer.

`Dense` implements the operation: `output = activation(dot(input, kernel) + bias)` where `activation` is the element-wise activation function passed as the `activation` argument, `kernel` is a weights matrix created by the layer, and `bias` is a bias vector created by the layer (only applicable if `use_bias` is `True`).

- **Note**: if the input to the layer has a rank greater than 2, then it is flattened prior to the initial dot product with `kernel`.

Example

```
# as first layer in a sequential model:
model = Sequential()
model.add(Dense(32, input_shape=(16,)))
# now the model will take as input arrays of shape (*, 16)
# and output arrays of shape (*, 32)

# after the first layer, you don't need to specify
# the size of the input anymore:
model.add(Dense(32))
```

Dropout [source]

```
keras.layers.Dropout(rate, noise_shape=None, seed=None)
```

Applies Dropout to the input.

Dropout consists in randomly setting a fraction `rate` of input units to 0 at each update during training time, which helps prevent overfitting.

Arguments

- **rate**: float between 0 and 1. Fraction of the input units to drop.
- **noise_shape**: 1D integer tensor representing the shape of the binary dropout mask that will be multiplied with the input. For instance, if your inputs have shape `(batch_size, timesteps, features)` and you want the dropout mask to be the same for all timesteps, you can use `noise_shape=(batch_size, 1, features)`.
- **seed**: A Python integer to use as random seed.

Conv1D

[source]

Number of filters

```
keras.layers.Conv1D(filters, kernel_size, strides=1, padding='valid', dilation_rate=1, activation=None, use_b
```

Size of filter

1D convolution layer (e.g. temporal convolution).

This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs. If `use_bias` is True, a bias vector is created and added to the outputs. Finally, if `activation` is not `None`, it is applied to the outputs as well.

When using this layer as the first layer in a model, provide an `input_shape` argument (tuple of integers or `None`, e.g. `(10, 128)` for sequences of 10 vectors of 128-dimensional vectors, or `(None, 128)` for variable-length sequences of 128-dimensional vectors.

Arguments

- **filters**: Integer, the dimensionality of the output space (i.e. the number output of filters in the convolution).
- **kernel_size**: An integer or tuple/list of a single integer, specifying the length of the 1D convolution window.
- **strides**: An integer or tuple/list of a single integer, specifying the stride length of the convolution. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- **padding**: One of `"valid"`, `"causal"` or `"same"` (case-insensitive). `"valid"` means "no padding". `"same"` results in padding the input such that the output has the same length as the original input. `"causal"` results in causal (dilated) convolutions, e.g. output[t] does not depend on input[t+1:]. Useful when modeling temporal data where the model should not violate the temporal order. See WaveNet: A Generative Model for Raw Audio, section 2.1.