Goal:

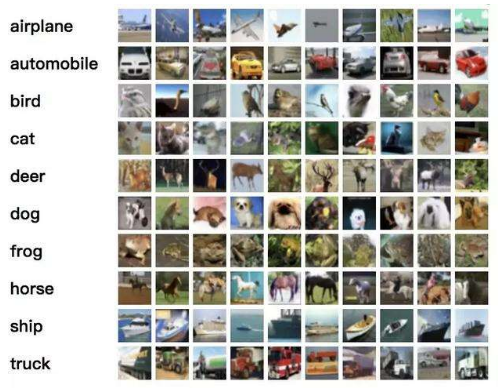
Provide some aspects of solutions to improve the performance of neural networks.

Outlines

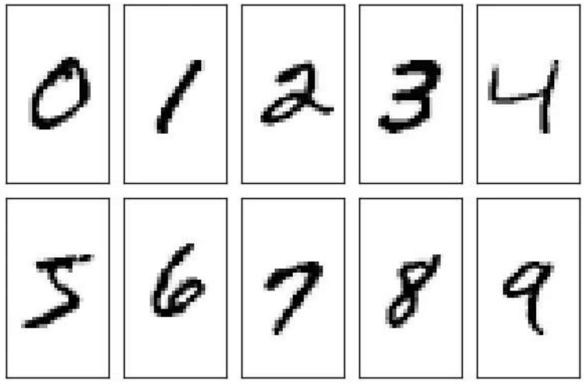
- Data
- Models
- Models Tuning
- Ensembles

Data

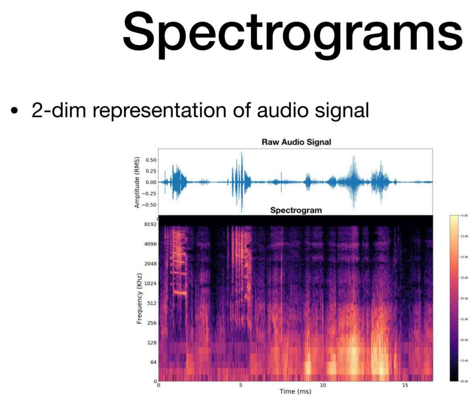
Data is the KEY to machine learning and deep learning. What machine learning or deep learning do is to analyze the features of data and make predictions based on it. There are thousands of dataset right now, for every kind of supervised learning tasks. No matter computer vision tasks or natural language processing, data is always the key.



Cifar 10



MNIST

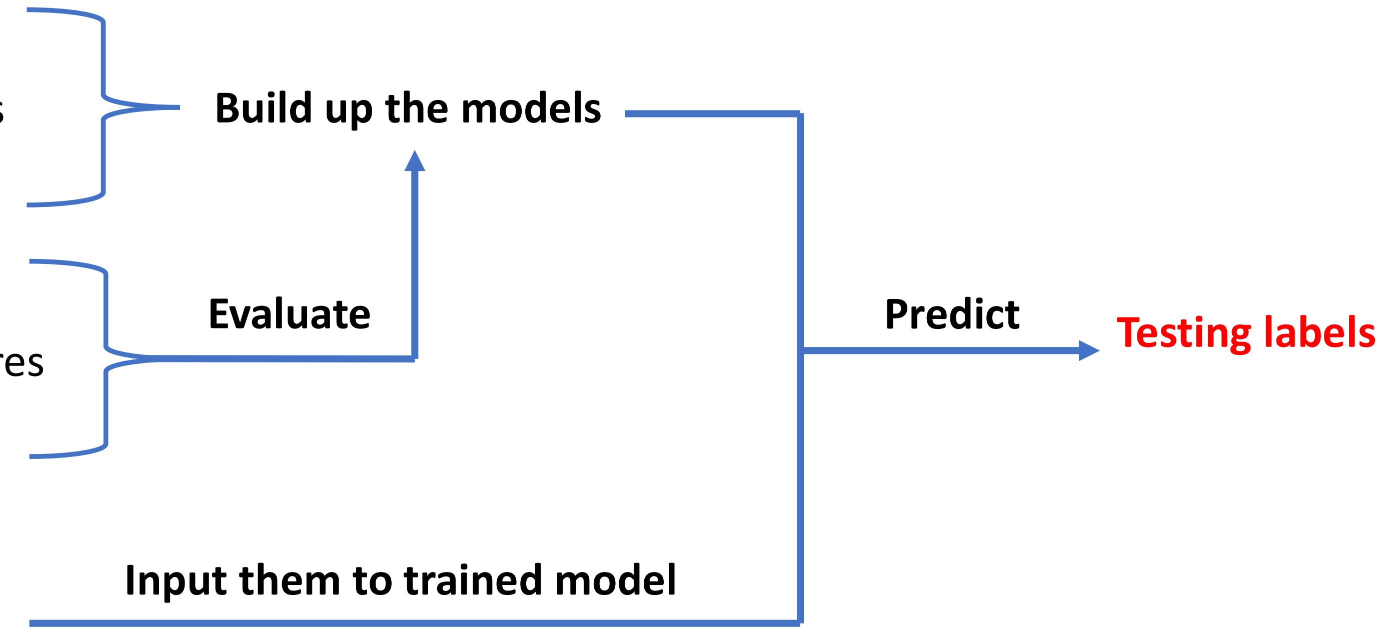


Sound Data

Data

Architecture of data

- Training data
    - -- Training Features
    - -- Training labels
- Validation data
    - -- Validation Features
    - -- Validation labels
- Testing data
    - -- Testing Features



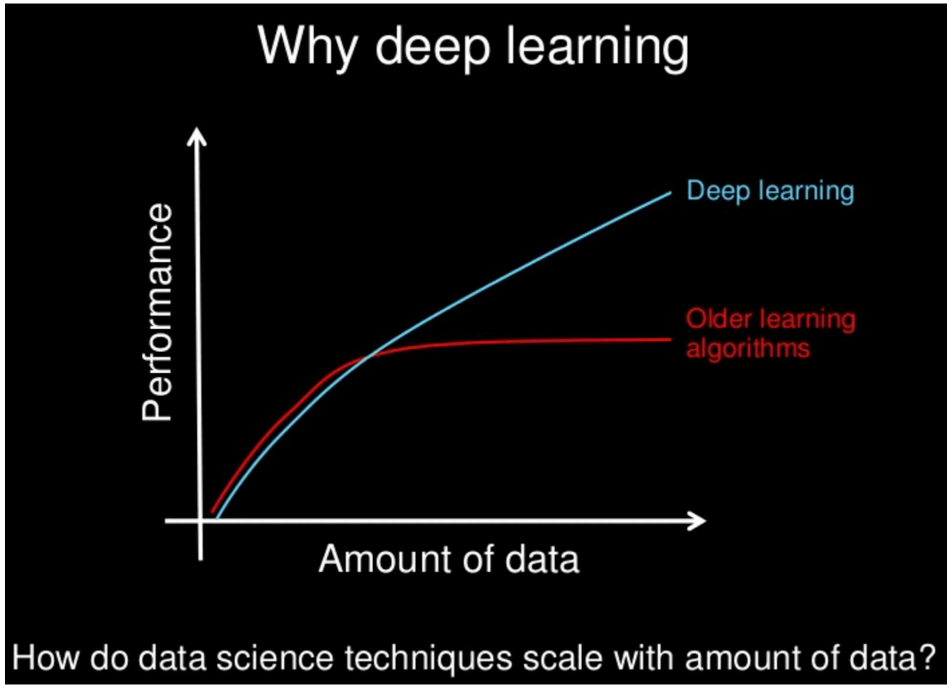
Data

1. Get more data

The number of data will determine the performance of your algorithm. Especially for deep learning.

However, in our dataset, the size of the training dataset is fixed.

ATTENTION: you can never use validation or test data for training. It is cheat. And cannot use other datasets

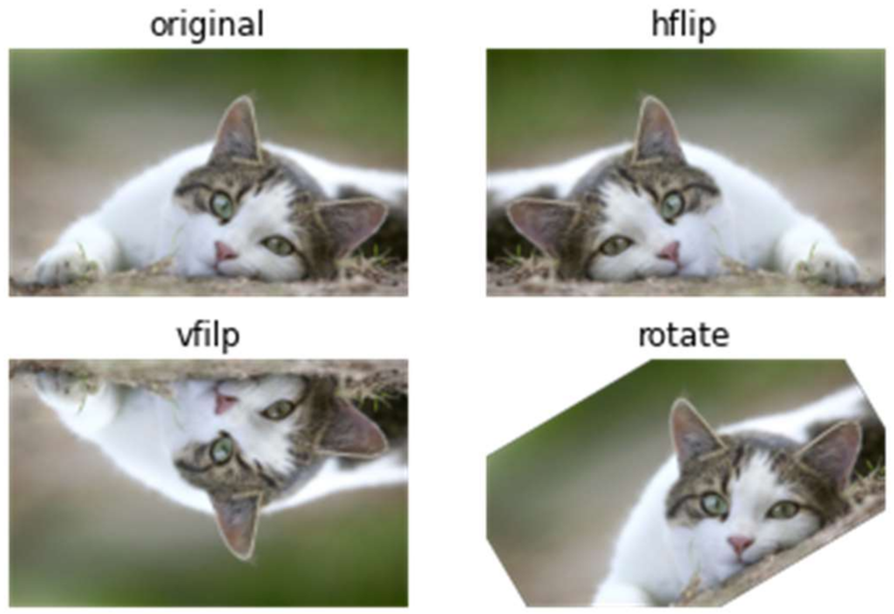


Data

2. Invent more data

As we mentioned before, the size of the training dataset is of great importance, and we have to get more data. That's why we do **data augmentation**.

Refer to: https://pytorch.org/vision/stable/transforms.html



Data

3. Resize data

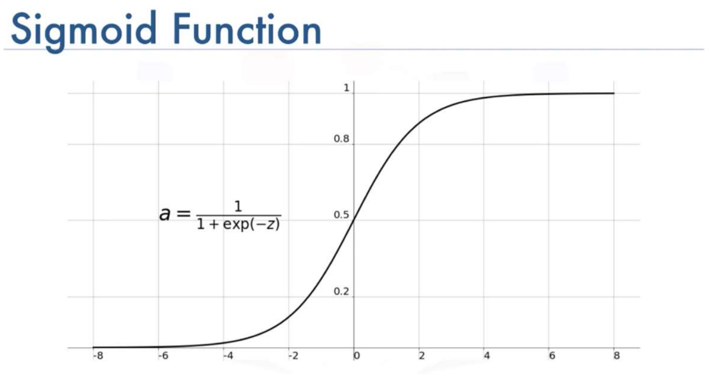
A traditional rule of thumb when working with neural networks is:

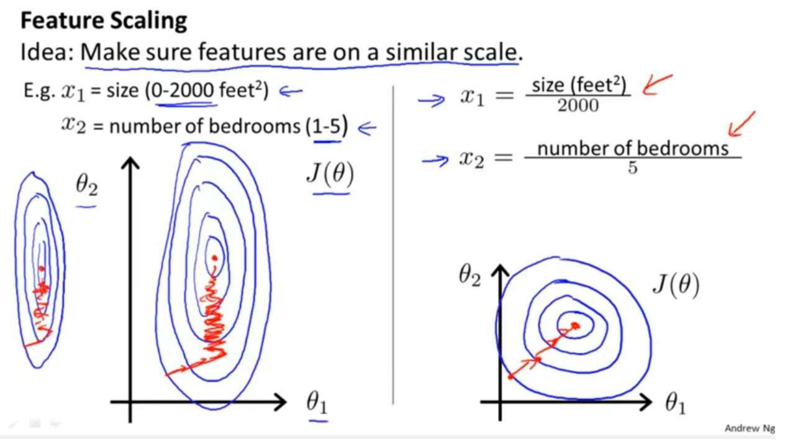
Rescale your data to the bounds of your activation functions.

Why: i) scales between features may well be different; ii) speed up convergence of gradient descent

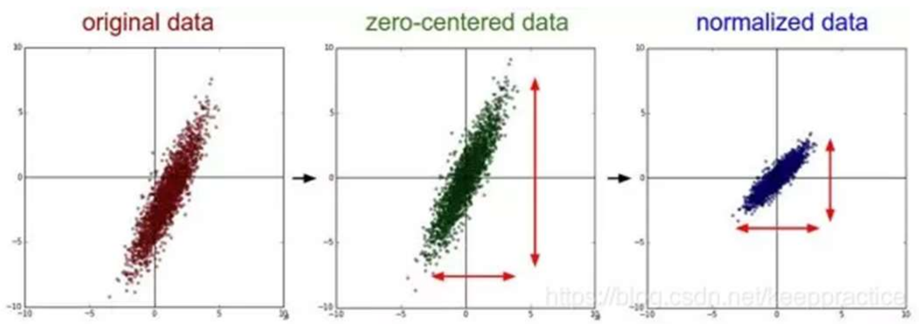
Normalization: a rescaling of the data from the original range so that all values are within the range of 0 and 1.

Standardization: rescaling the distribution of values so that the mean of observed values is 0 and the standard deviation is 1. It is sometimes referred to as "whitening."





Feature Scaling
Idea: Make sure features are on a similar scale.
E.g. x_1 = size (0-2000 feet^2)
x_2 = number of bedrooms (1-5)
$x_1 = \frac{\text{size (feet}^2)}{2000}$
$x_2 = \frac{\text{number of bedrooms}}{5}$
θ_1
θ_2
$J(\theta)$
Andrew Ng



Models

1. Choose a better model for your task

- It means you can 'steal' ideas from published research, which is highly optimized.
- You can also check piazza, papers, books, posts and every source permitted.

Like for image classification, VGG, Alex net, Resnet, Google Net are all great models

For seq2seq task, LAS is one of the best models. You can also search for other architectures.

HINT: Try to search for some variants of ResNet and use them in your homework! CNN can also be used in NLP question.

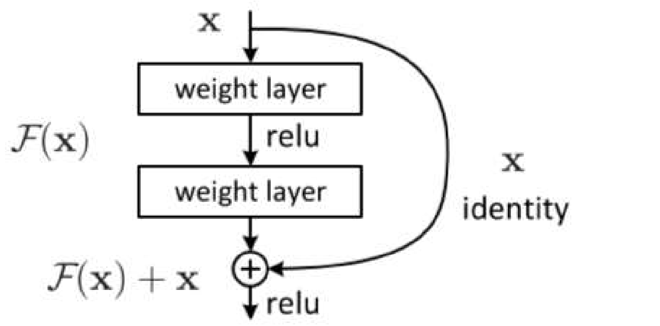
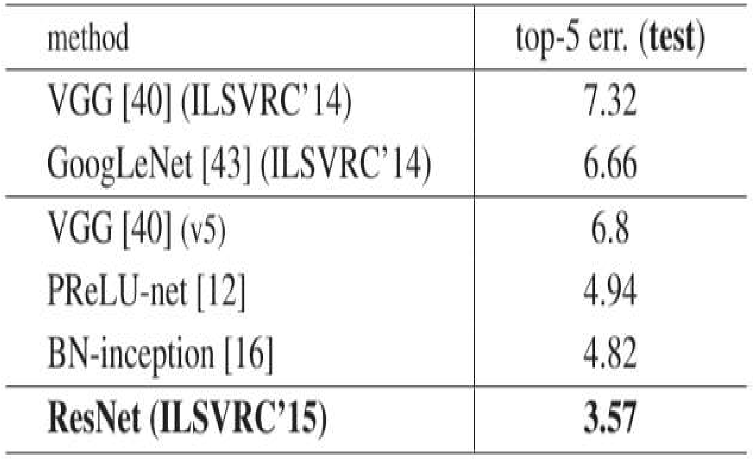

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2. Residual learning: a building block.



method	top-5 err. (**test**)
VGG [40] (ILSVRC'14)	7.32
GoogLeNet [43] (ILSVRC'14)	6.66
VGG [40] (v5)	6.8
PReLU-net [12]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

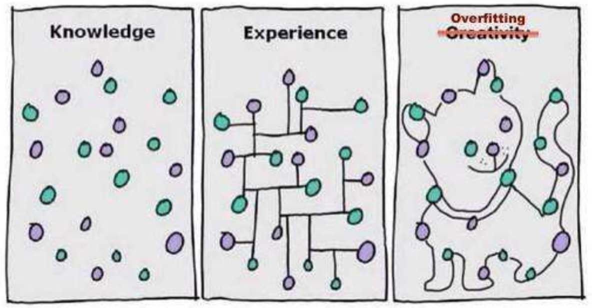
Table 5. Error rates (%) of **ensembles**. The top-5 error is on the test set of ImageNet and reported by the test server.

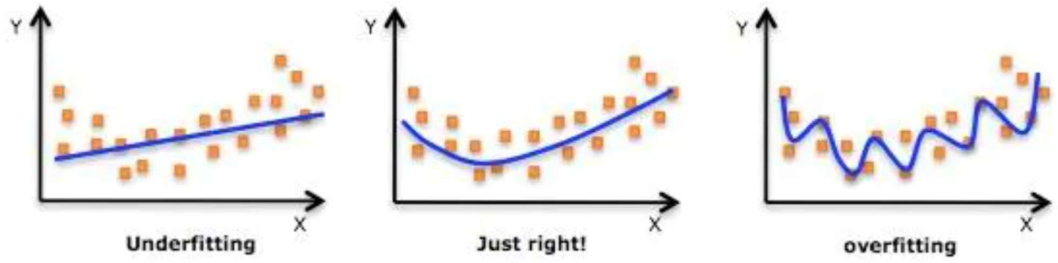
Models

2. Build a stronger model

Deeper and Wider layers tend to have a better performance, because they have more parameters. However, it may cause **"overfitting"** problem and computation time will be much longer.

Most tricks we will talk about aim at solving overfitting problem.





If I've learned anything from my studies of Deep Learning, it's that one can solve most problems by just adding more layers

From memes thread in spring2021

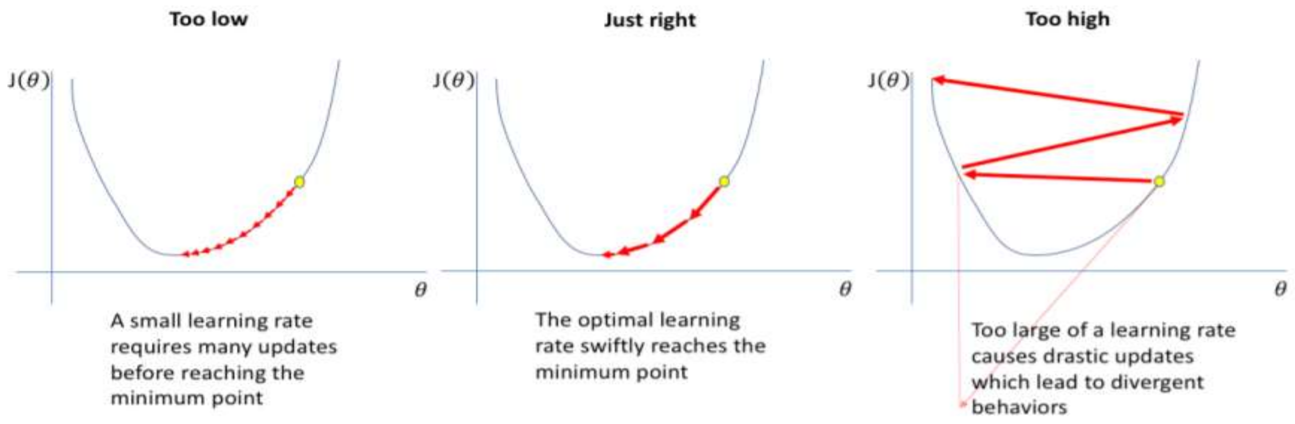
Models Tuning

1. Learning rate and its scheduler

Among all the hyperparameters in NN, learning rate is one of the most important ones that affect your performance. So you can:

- Experiment with different learning rates. (give a good initial LR)
- Anneal the learning rate over epochs. (LR scheduler)

Learning rate is coupled with the number of training epochs, batch size $(LR_{new} = LR_{old} * \sqrt{\frac{BS_{new}}{BS_{old}}})$ and optimization methods.


Too low
Just right
Too high
$J(\theta)$
θ
A small learning rate requires many updates before reaching the minimum point
The optimal learning rate swiftly reaches the minimum point
Too large of a learning rate causes drastic updates which lead to divergent behaviors

Models Tuning

2. Optimizer

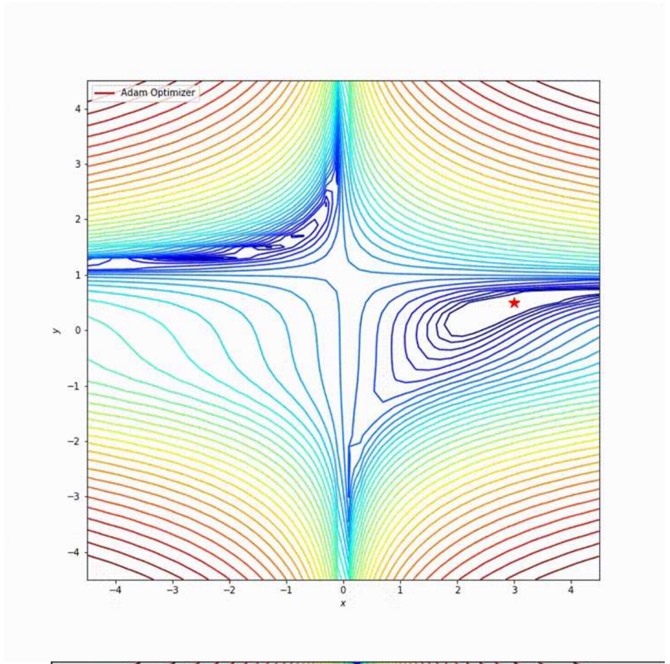
It used to be stochastic gradient descent (aka SGD), but now there are a ton of optimizers.

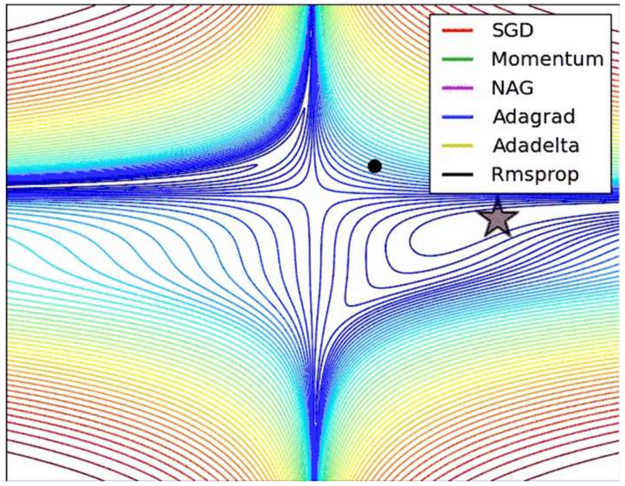
NAG, Adagrad, Rmsprop, Adam, etc. Considering more parameters, like momentum, to make optimizer strong.

HINT: SGD tends to have better performance if we can find the best parameters of SGD. But Adam is the efficient enough to achieve good performance without searching for parameters.

Adam converges fast and fine-tuned SGD can have a better performance. How about firstly use Adam and switch to SGD?

Refer to this paper: https://arxiv.org/abs/1712.07628





Models Tuning

3. Weight Initialization

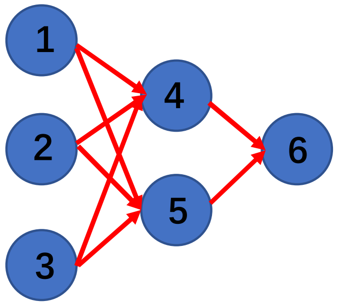
Why is initialization important?

- Resolves the issue of exploding/vanishing gradients/activations(to some extent)
- Faster convergence
- Helps reach better minima

Why cannot we initialize them to zero or constant?

Gradient descent will backpropagate same value to weights so that model can hardly converge.

It doesn’t suit the case of RNN. Zero or constant initialization are important for recurrent neural network.



Initialize

$$First\ Layer\ W_1 = \begin{bmatrix} w_{41} & w_{42} & w_{43} \\ w_{51} & w_{52} & w_{53} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$Second\ Layer\ W_2 = [w_{64} \quad w_{65}] = [0 \quad 0]$$

$$Input\ X = [x_1 \quad x_2 \quad x_3]$$

Forward

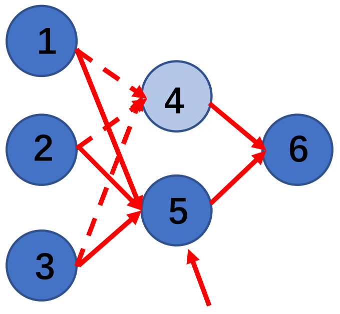
$$First\ Layer\ Output\ Z = [z_4 \quad z_5]$$
$$z_4 = w_{41} * x_1 + w_{42} * x_2 + w_{43} * x_3 = 0$$
$$z_5 = w_{51} * x_1 + w_{52} * x_2 + w_{53} * x_3 = 0$$

$$Second\ Layer\ Output$$
$$a_6 = w_{64} * a_4 + w_{65} * a_5$$
$$where\ a_4 = f(a_4), a_5 = f(a_5)$$

$$Loss = \frac{1}{2}(y - a_6)^{\ 2}$$

Backward

$$\frac{\delta L}{\delta w}\ will\ be\ the\ same. Cannot\ update\ weights$$

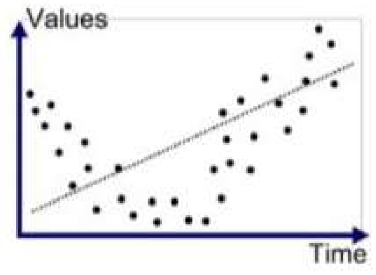


Only one neuron works

Models Tuning

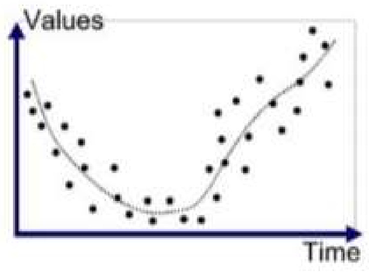
4. Regularization

Overfitting is a modeling error that occurs when a fusion is too closely fit to a limited set of data points.



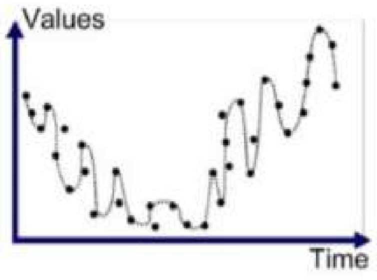
Underfitted

$y = a + w_0 x$



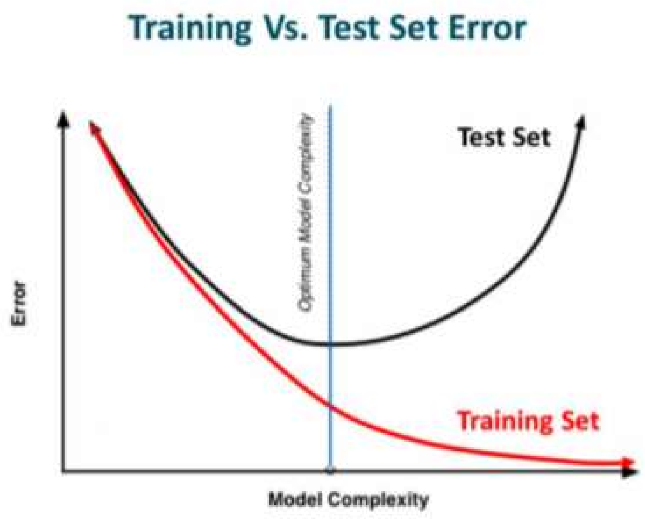
Good Fit/Robust

$y = a + w_0 x + w_1 x^2 + w_2 x^3$



Overfitted

$y = a + w_0 x + w_1 x^2 + w_2 x^3 + w_3 x^4 + \cdots + w_{10} x^{11}$



Models Tuning

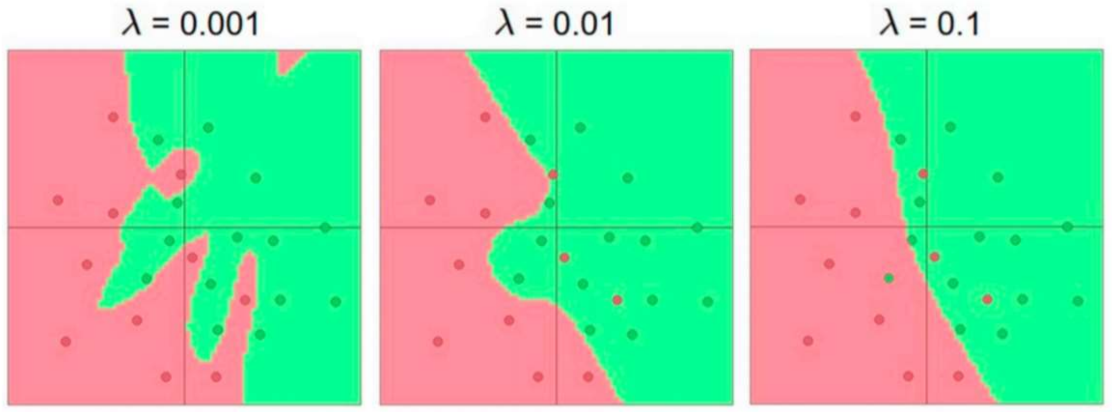
- **L1/L2 Regularization**

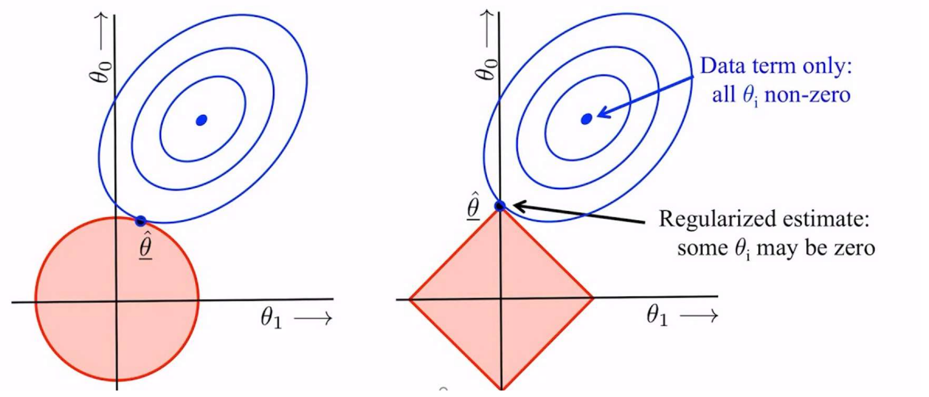
It involves adding an extra element to the loss function, which punishes our model for being too complex or, in simple words, for using too high values in the weight matrix (usually, lambda is 1e-4 or 1e-5)

L0 Norm $Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} ||w_i||, stands\ for\ \#\ of\ zero$

L1 Norm $Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i|$

L2 Norm $Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2$




$\theta_0 \longrightarrow$
$\hat{\theta}$
$\theta_1 \longrightarrow$
Data term only: all θ_i non-zero
Regularized estimate: some θ_i may be zero

Models Tuning

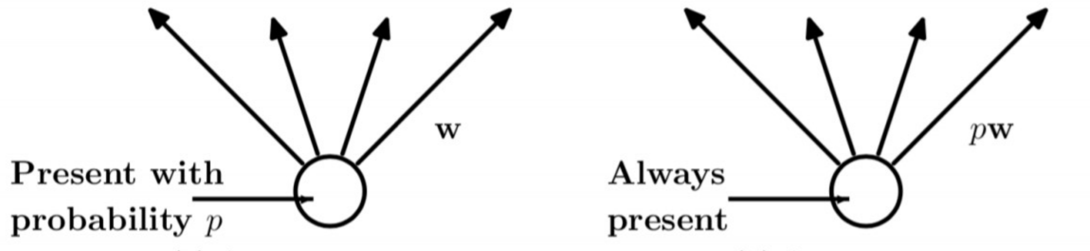
- **Dropout**

For each training batch, you turn off some neurons with a probability.

Motivation: With unlimited computation, the best way to "regularize" a fixed-sized model to average the predictions of all possible settings of the parameters. Practically, it's computationally prohibitive. So dropout provides a method to use O(n) neural network to approximate O(2^n) different architectures with shared O(n^2) parameters.

Implementation:

- **Train Time:** Mask some neuron outputs as 0 with a probability
- **Test Time:** No parameters masked at test time but need to multiply with the dropout probability to approximate the expected output
- **Hyper-parameter:** dropout rate, usually from 0.1 to 0.5



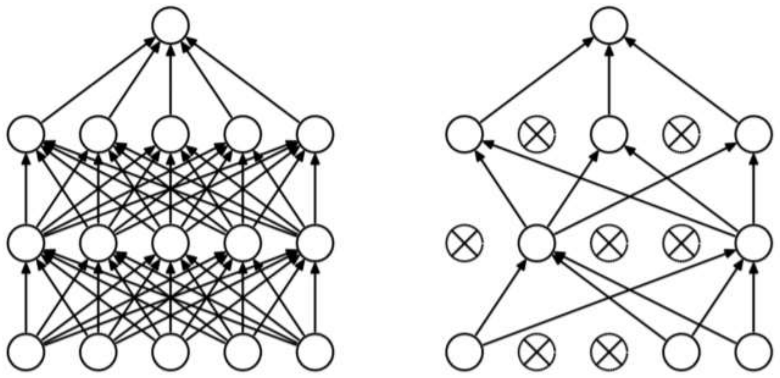
(a) At training time (b) At test time

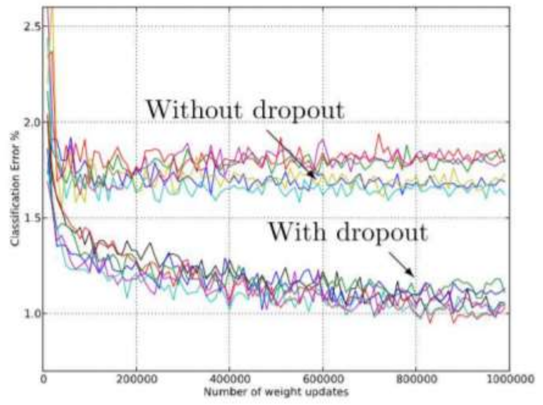
Models Tuning

- **Dropout**

Two problems resolved:

- **Overfitting:** disable some outputs so that the layers cannot overfit the data. Or we can see dropout is like generating many models, have different "overfitting" effect but some of them are opposite. We average them and can prevent overfitting in general
- **Generalization:** let model abandon some specific features with probability and decrease co-adaptation between the neurons


(a) Standard Neural Net (b) After applying dropout.



Models Tuning

- **Batch-Norm**

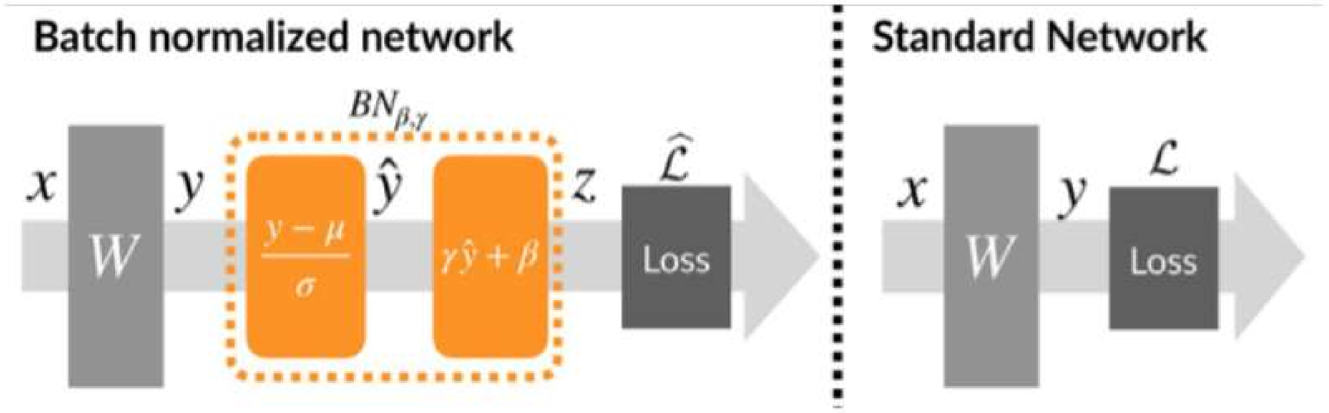
Wildly successful and simple technique for accelerating training and learning better neural network representations

Motivation: The general motivation of BatchNorm is the non-stationary of unit activity during training that requires downstream units to adapt to a non-stationary input distribution. This co-adaptation problem, which the paper authors refer to as ICS (internal covariate shift), significantly slows learning.

ICS: In neural networks, the output of the first layer feeds into the second layer, the output of the second layer feeds into the third, and so on. When the parameters of a layer change, so does the distribution of inputs to subsequent layers.

These shifts in input distributions can be problematic for neural networks, especially deep neural networks that could have a large number of layers.

Models Tuning

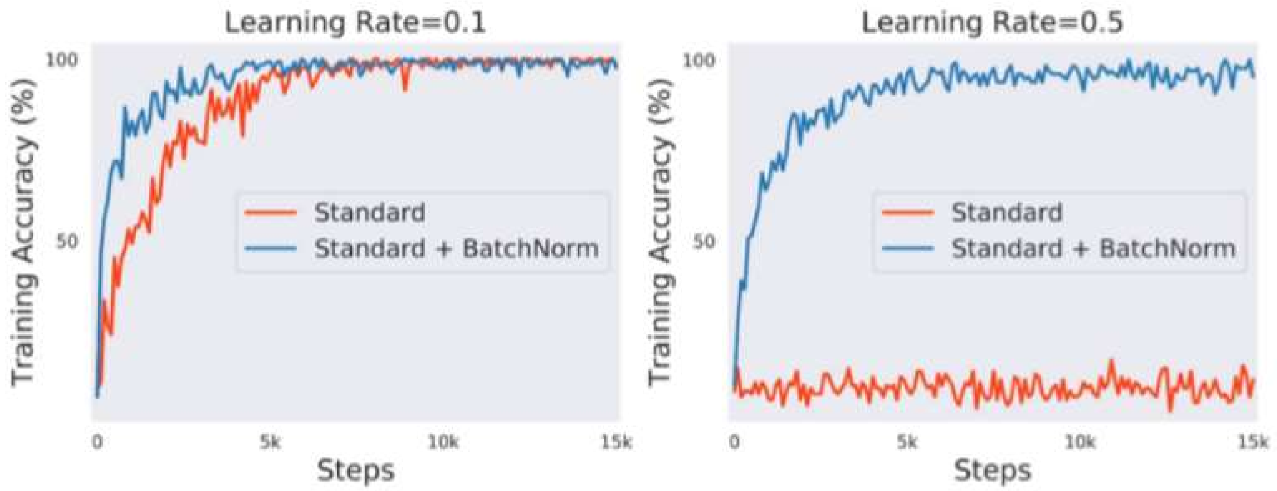


1.

$$u_{\mathcal{B}}^{(k)} \leftarrow \frac{1}{m}\sum_{i=1}^{m} \mathbf{x}_i^{(k)}$$

$$(\sigma_{\mathcal{B}}^2)^{(k)} \leftarrow \frac{1}{m}\sum_{i=1}^{m} \left(\mathbf{x}_i^{(k)} - u_{\mathcal{B}}^{(k)}\right)^2$$

$$\hat{\mathbf{x}}_i \leftarrow \frac{\mathbf{x}_i - u_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$



2.

$$\mathbf{y}_i \leftarrow \gamma\hat{\mathbf{x}}_i + \beta$$

Models Tuning

- **Batch-Norm**

Benefits:

a) **BN enables higher training rate:** Normally, large learning rates may increase the scale of layer parameters, which then amplify the gradient during back propagation and lead to the model explosion. However, with Batch Normalization, BP through a layer is unaffected by the scale of its parameters

$$BN(Wu) = BN\big((aW)u\big) \quad \frac{\partial BN((aW)u)}{\partial u} = \frac{\partial BN(Wu)}{\partial u}$$

a) **Faster Convergence.**

b) **BN regularizes the models:** a training example is seen in conjunction with other examples in the mini-batch, and the training network no longer producing deterministic values for a given training example. In our experiments, we found this effect to be advantageous to the generalization of the network.

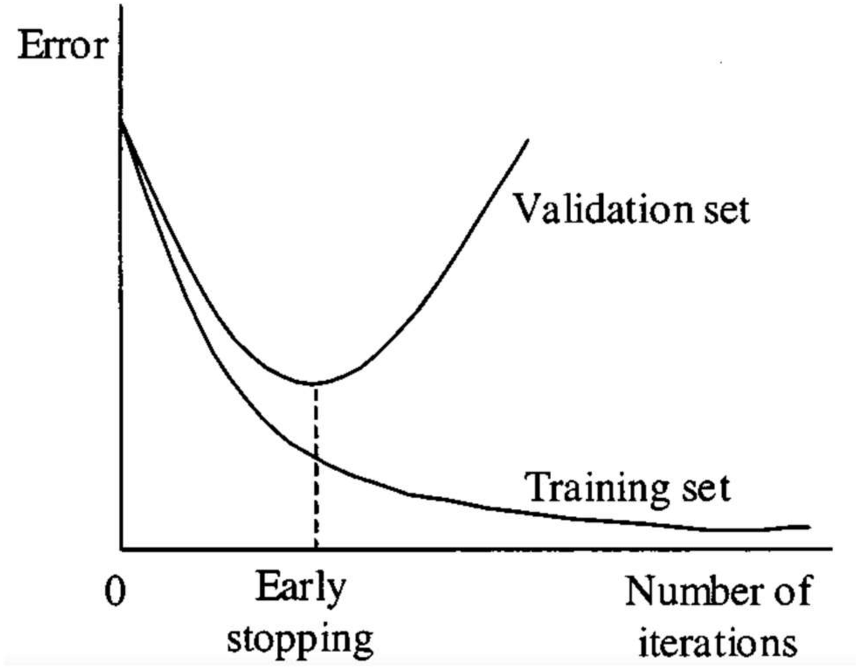
Models Tuning

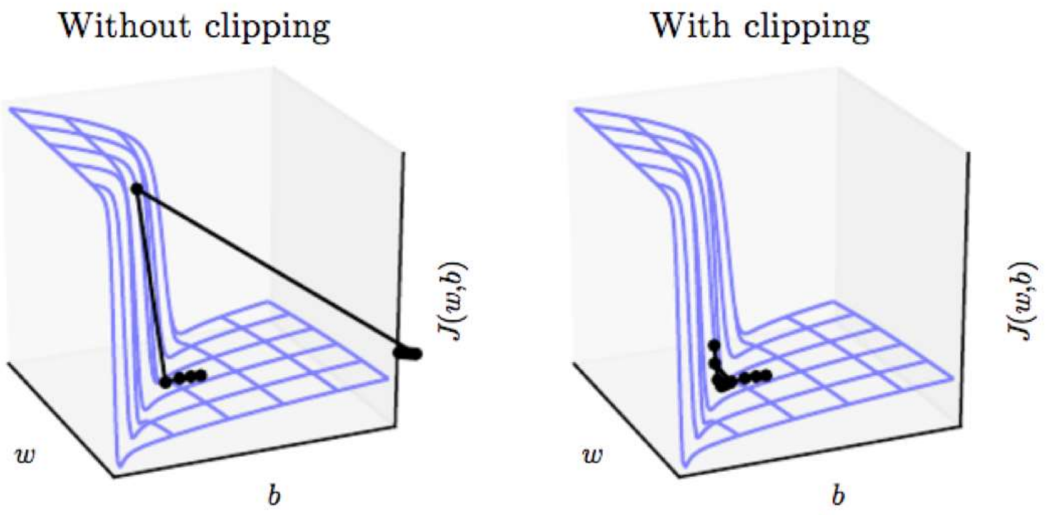
- **Early Stop**

Literally, just stop the training when you see the validation score decreases, where overfitting begins

- **Gradient Clipping**

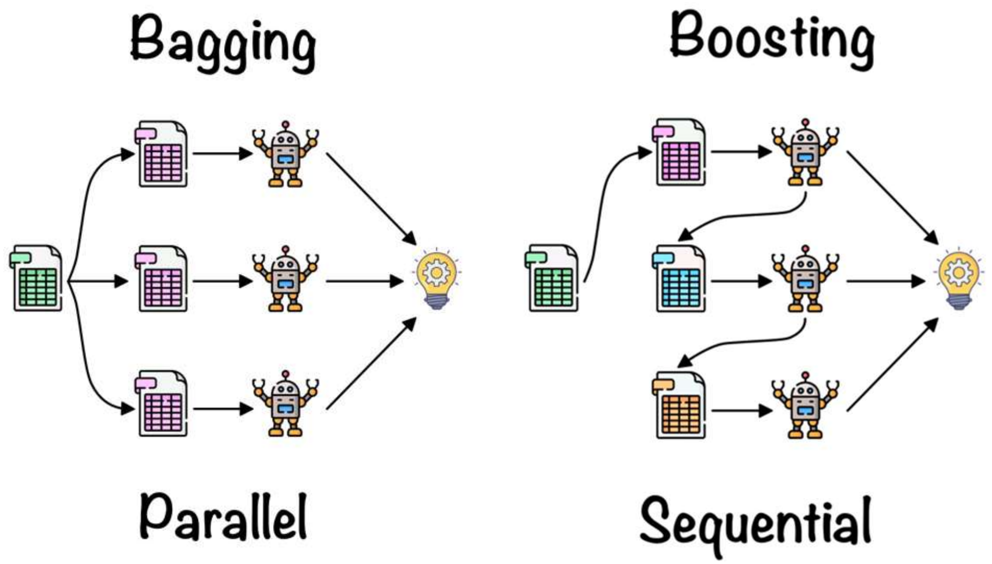
Once the gradient is over the threshold, clip and keep them to the threshold value. It is important for RNN.





Ensemble

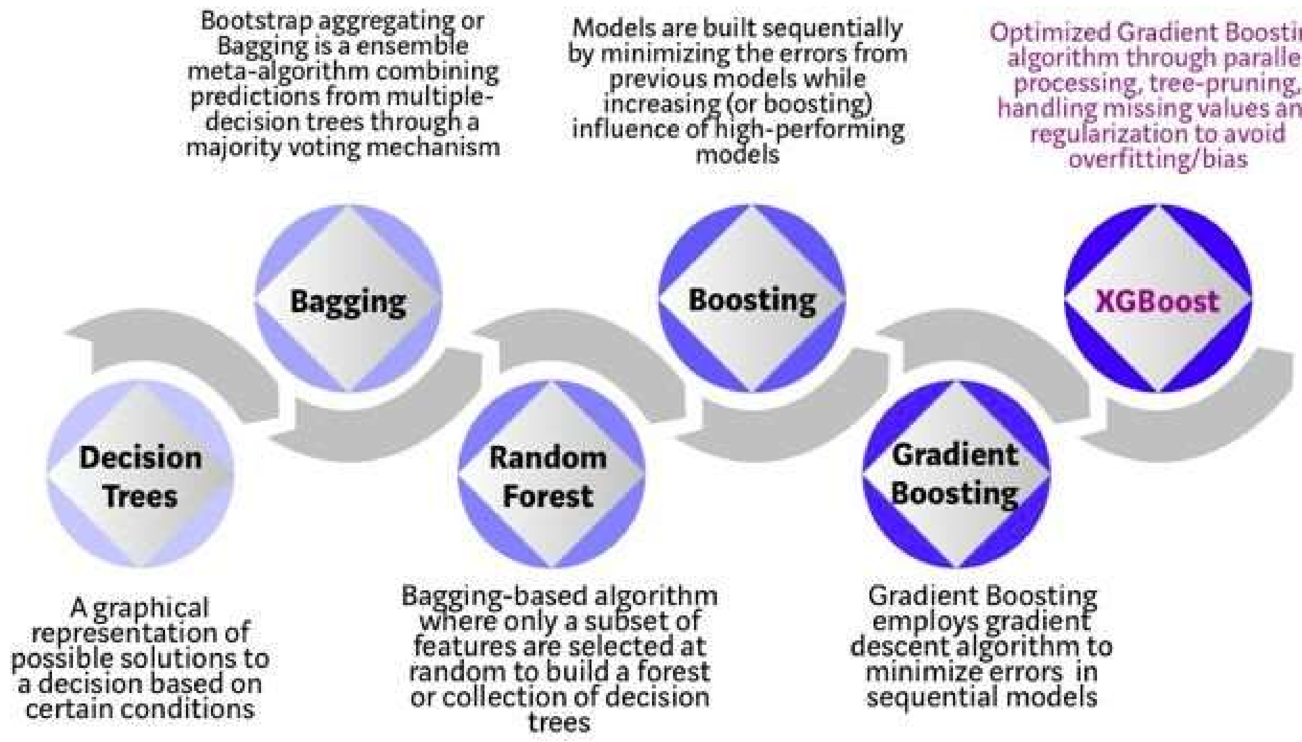
Ensemble learning is **a machine learning paradigm where multiple learners are trained to solve the same problem**. In contrast to ordinary machine learning approaches which try to learn one hypothesis from training data, ensemble methods try to construct a set of hypotheses and combine them to use.



Bagging: considers homogeneous weak learners, learns them independently from each other in parallel and combines them following some kind of deterministic averaging process. **Decrease variance.**

Boosting: considers homogeneous weak learners, learns them sequentially in a very adaptative way (a base model depends on the previous ones) and combines them following a deterministic strategy. **Decrease bias.**

Ensembles

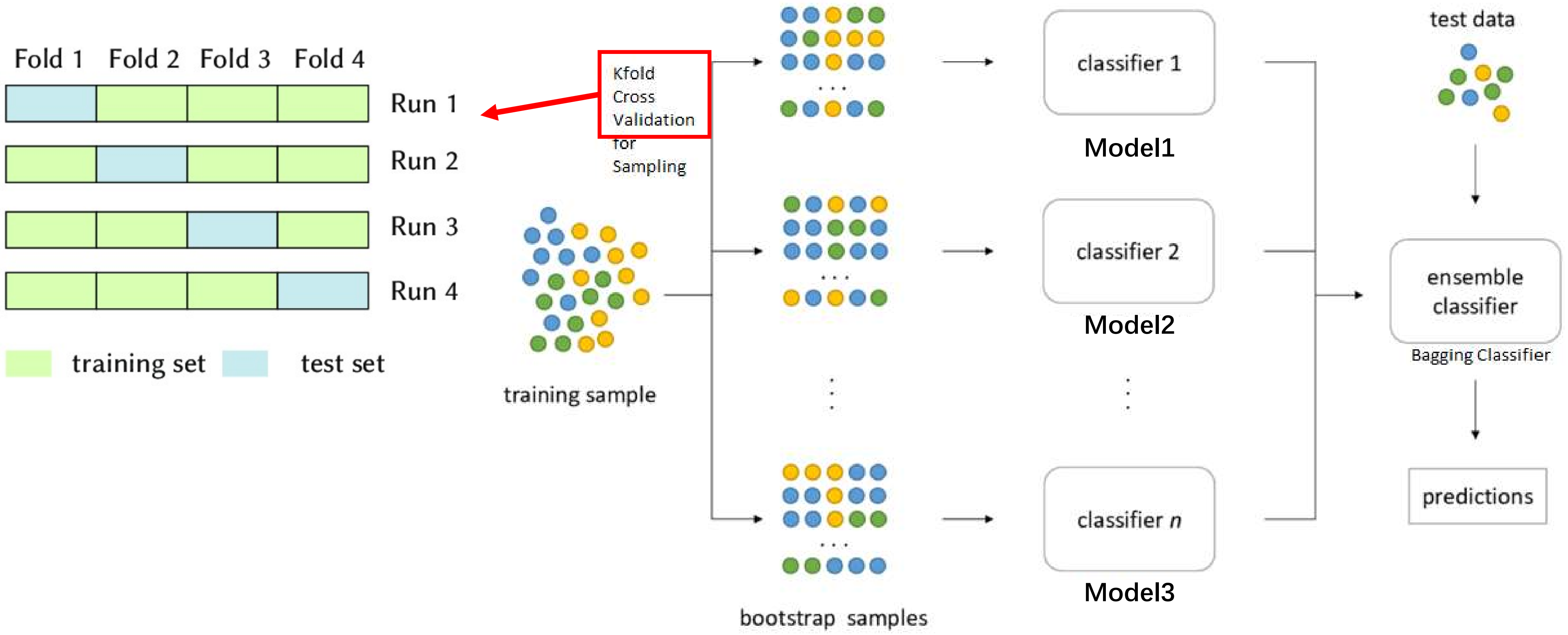

Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism
Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models
Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias
Bagging
Boosting
XGBoost
Decision Trees
Random Forest
Gradient Boosting
A graphical representation of possible solutions to a decision based on certain conditions
Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees
Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

Develop of ensembles

Refer to paper. Author

Ensembles

- **Bagging: Voting-based algorithm. Train different models in isolation and use average voting or weighted voting method to get result.**

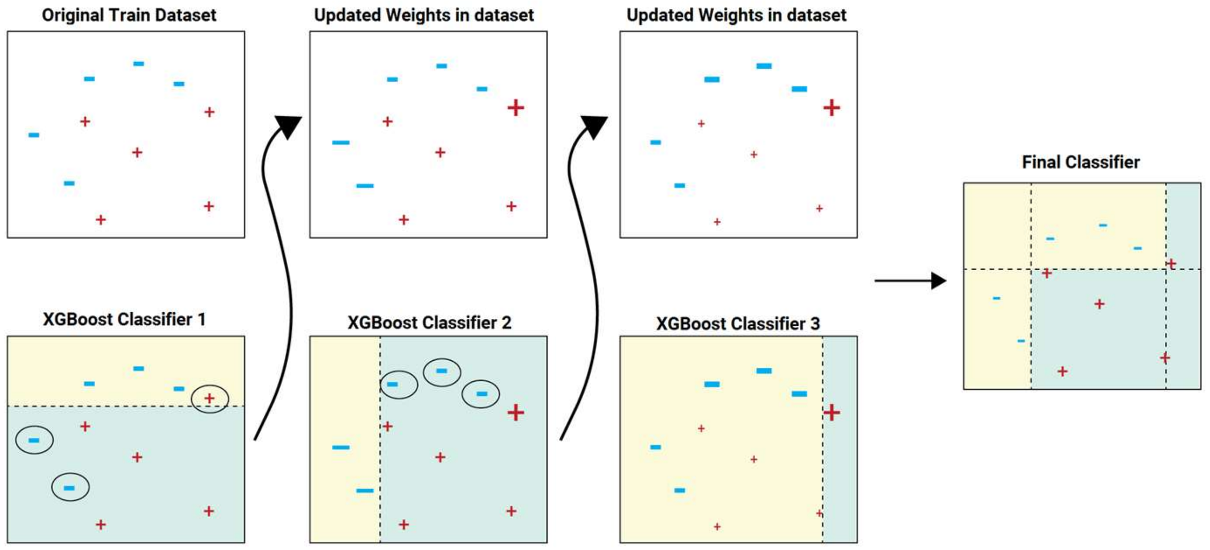


Bagging Classifier Process Flow

Ensembles

- **XGBoost: One of most powerful weapons in Kaggle. Many deep learner use it reached top in many competitions**

  XGBoost as a boosting method is a category of ensemble methods. Rather than training all of the models in isolation of one another, boosting trains models in succession, with **each new model being trained to correct the errors made by the previous ones.** Models are added sequentially until no further improvements can be made, that's why it is called additive model.



Refer to paper. Author Tianqi Chen joined CMU in 2020!

Ensembles

- **XGBoost: One of most powerful weapons in Kaggle. Many deep learner use it reached top in many competitions**

  Pseudocode:

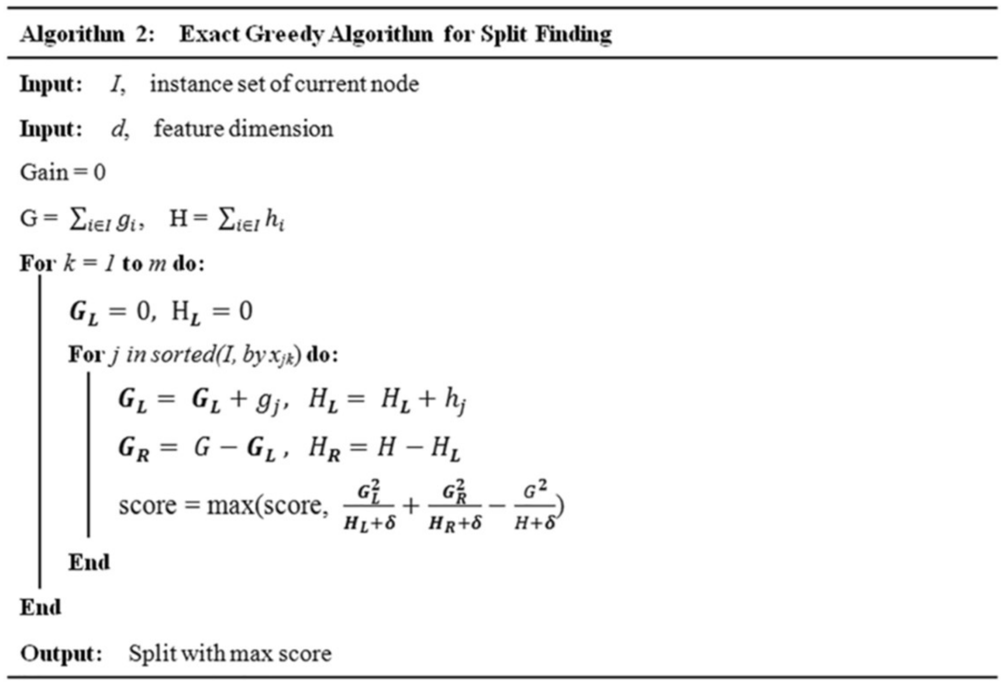
  ① Initialize $f_0(x)$;

  ② For m = 1 to M:

  a. Compute the 1st derivative and 2nd derivative of loss function over each sample;

  b. Recursively use "Greedy Algorithm for Split Finding" to generate the base learner;

  c. Add the base learner to models.



```
Algorithm 2:  Exact Greedy Algorithm for Split Finding
Input:  $I$,  instance set of current node
Input:  $d$,  feature dimension
Gain = 0
G = $\sum_{i \in I} g_i$,   H = $\sum_{i \in I} h_i$
For $k = 1$ to $m$ do:
    $G_L = 0$,  $H_L = 0$
    For $j$ in sorted($I$, by $x_{jk}$) do:
        $G_L = G_L + g_j$,  $H_L = H_L + h_j$
        $G_R = G - G_L$,  $H_R = H - H_L$
        score = max(score, $\frac{G_L^2}{H_L+\delta} + \frac{G_R^2}{H_R+\delta} - \frac{G^2}{H+\delta}$)
    End
End
Output:  Split with max score
```

Refer to paper. Author Tianqi Chen joined CMU in 2020!