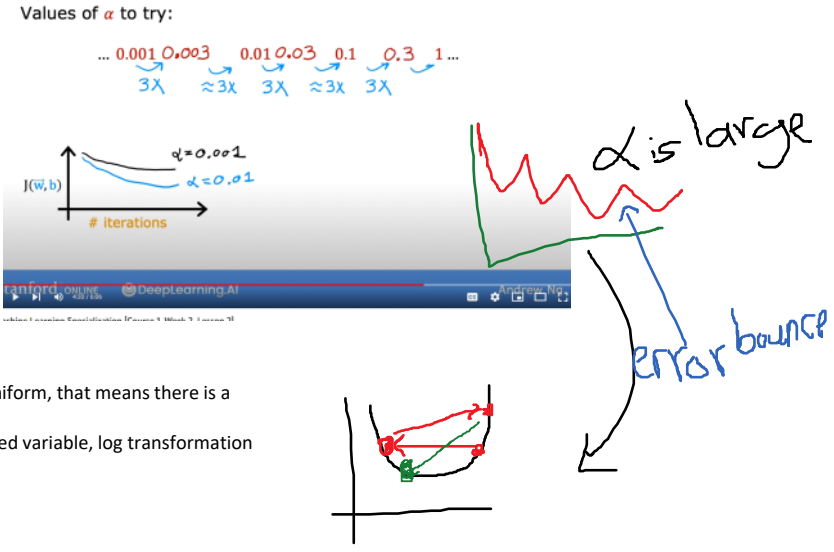
Udemy Data Scientist

Thursday, May 25, 2023 3:52 PM



1. Theory of data analysis
2. Probability
    a. A good review about the probability distribution, formulas
3. Statistics
    a. Central Limit Theorem
4. Supervised learning
    a. Linear Regression
        i. OLS
        ii. R^2
        iii. F-Test
    b. Multiple Linear Regression
        i. No Endogeneity
            1) When regressor is correlated with error term
        ii. Normality and Homoscedasticity
            1) Usually solved with central limit theorem
            2) Homoscedasticity is when the variance of error is not uniform, that means there is a pattern to it
                a) Can be solved by removing outliers, include omitted variable, log transformation
        iii. Feature Selection
            1) F-regression
        iv. Categorical variables need N-1 dummies
        v. Scaling variables
    c. Logistic Regression
        i. Used to classify labeled data and then predict new data
5. Unsupervised Learning
    a. Clustering
        i. KMeans
            1) WCSS Coeff to decide on the number of clusters
            - number of k --> use elbow method
            - seeding points --> use k-means++ init
            - sensitive to outliers --> remove outliers
            - spherical solutions because of using Euclidean method
            - Standardization bcz non-standard values affect the clustering, so you have to decide if you are going to do it or not
                - Even standardizing variables, may result in a case where both axis become a square shape and clustering won't be able to decide

        ii. Naming clusters is the best way for you to decide if the cluster makes any sense.
        iii. HeatMaps
        iv. Dendrogram
6. Linear Algebra
7. Deep Learning
    a. deep learning can be supervised, unsupervised, semi-supervised, self-supervised, or reinforcement, and it depends mostly on how the neural network is used.
    b. Objective functions
        i. Loss function: used in supervised learning. Tries to minimize the error
        ii. Reward function: used in reinforcement learning. Tries to maximize the reward
        iii. L2-norm loss function is similar to OLS function of regression
        iv. In classification model, cross entropy loss is used
    c. Optimization Function
        i. Gradient Descent Function
            1) Calculates the learning rate of the model
            2) Find global minimum by adding momentum
            3) Decide on learning rate Eta using some algorithms
            - batch-size = n --> Gradient Descent or Batch GD
                - that means we have one batch only
            - batch-size = 1 --> Stochastic Gradient Descent
                - that means we have n batches
            - batch-size = 1> and <n --> mini-batch Gradient Descent also referred to as SGD
        ii. Stochastic Gradient Descent SGD
        iii. Adaptive optimizer Adam
            1) Currently is the best optimizer function to use
    d. TensorFlow
        i. Simplifies Deep learning implementations
        ii. Keras acts as a window to the tensorflow
        iii. Tf.keras.sequential() builds a model
        iv. Tf.keras.Dense() adds layers to the network (dot products the input with weights)
        v. Epoch is one iteration over the WHOLE dataset
            1) Epoch can consist on multiple iterations according to the batch-size
            2) After passing each batch the weights are updated

3) Epochs should continue iterating until some max or when difference between new and old weight is less than 0.001

vi. Each layer must have non-linear transformation after the linear one also called activation function

vii. Each activation function performs a specific task
   1) Sigmoid produces a binary output
   2) Softmax produces a probability output using all of the input variables

viii. Backpropagation is used to update the weights properly

e. Overfitting and Underfitting
   i. To avoid overfitting, validation data is needed
   ii. when validation loss starts increasing, that means we are overfitting the model on the training data but it is doing poorly on the validation data.
   iii. In case of small data, you can have n-fold cross validation

f. Initializer is very important in neural network model
   i. using Xavier (glorot) initializer

8. Data encoding
   1. One-hot encoding
   2. Binary encoding

Hidden layers	Epoch	Time	Validation accuracy	Activation function	Batch-size	Test accuracy
50x50x50	Epoch 10	23.3 sec	0.9815	relu	100: 540 per epoch	
100x100x100	=	23.4	0.9880	=	100	
50x50x50x50	17	43	0.9898	=	100	
200x100x50x50	10	25	0.9897	=	=	
200x100x50x25	6	15.	0.9878	=	=	
200x100x50x25x50	10	20	0.9910	=	=	
=	18	47	0.9962	1: relu Remaining: tanh	=	
=	19	34.5	0.9992	=	1000: 54 per epoch	
=	32	66	1	=	1000: 54 per epoch	0.9811

Takeaway

2023年6月7日　　9:36

- Understand your data before building any model
- Pre-processing your data is one of the most important parts of data analysis
    - That includes creating dummies, standardizing, dealing with NA's and so on
    - If interpretability is more important then standardization is not needed
        - In this case play with the weights to reduce the overwhelming effect of very large numbers
        - Is standardization needed before fitting logistic regression? - Cross Validated (stackexchange.com)
    - If accuracy is more important then standardization is needed
- If data is linearly separable, you don't need NN
    - You can use a perceptron model
    - You can use a linear regression (or logistic regression)
- The majority of cases, ONE hidden layer should solve the problem, so focus on optimizing the model
    - Choosing number of Hidden Layers and number of hidden neurons in Neural Networks (linkedin.com)
    - model selection - How to choose the number of hidden layers and nodes in a feedforward neural network? - Cross Validated (stackexchange.com)
- You will rarely need multiple hidden layer (usually need it with huge number of inputs like images)
- The optimum number of nodes in a hidden layer can be sqrt(input_size * output_size)
- *tf.keras.Sequential* model does not need an explicit Input layer, the input shape will be inferred when you pass real data to it or call model.build.
    - python - What is the difference between tf.keras.layers.Input() and tf.keras.layers.Flatten() - Stack Overflow
- Kernel_initializer is important to set for each node
- Drop rate and kernel_constraint is important when having large number of inputs
- Other hyperparameters are important
    - Epochs
    - Batch size
    - Learning rate

My self-study notes on Machine Learning

2023年6月8日 15:08

Tips and Tricks

Monitoring Validation Loss vs. Training Loss

If you're somewhat new to Machine Learning or Neural Networks it can take a bit of expertise to get good models. The most important quantity to keep track of is the difference between your training loss (printed during training) and the validation loss (printed once in a while when the RNN is run on the validation data (by default every 1000 iterations)). In particular:

- If your training loss is much lower than validation loss then this means the network might be **overfitting**. Solutions to this are to decrease your network size, or to increase dropout. For example you could try dropout of 0.5 and so on.
- If your training/validation loss are about equal then your model is **underfitting**. Increase the size of your model (either number of layers or the raw number of neurons per layer)

Approximate number of parameters

The two most important parameters that control the model are rnn_size and num_layers. I would advise that you always use num_layers of either 2/3. The rnn_size can be adjusted based on how much data you have. The two important quantities to keep track of here are:

- The number of parameters in your model. This is printed when you start training.
- The size of your dataset. 1MB file is approximately 1 million characters.

These two should be about the same order of magnitude. It's a little tricky to tell. Here are some examples:

- I have a 100MB dataset and I'm using the default parameter settings (which currently print 150K parameters). My data size is significantly larger (100 mil >> 0.15 mil), so I expect to heavily underfit. I am thinking I can comfortably afford to make rnn_size larger.
- I have a 10MB dataset and running a 10 million parameter model. I'm slightly nervous and I'm carefully monitoring my validation loss. If it's larger than my training loss then I may want to try to increase dropout a bit and see if that heps the validation loss.

Best models strategy

The winning strategy to obtaining very good models (if you have the compute time) is to always err on making the network larger (as large as you're willing to wait for it to compute) and then try different dropout values (between 0,1). Whatever model has the best validation performance (the loss, written in the checkpoint filename, low is good) is the one you should use in the end.

It is very common in deep learning to run many different models with many different hyperparameter settings, and in the end take whatever checkpoint gave the best

validation performance.

By the way, the size of your training and validation splits are also parameters. Make sure you have a decent amount of data in your validation set or otherwise the validation performance will be noisy and not very informative.

From <https://github.com/karpathy/char-rnn>

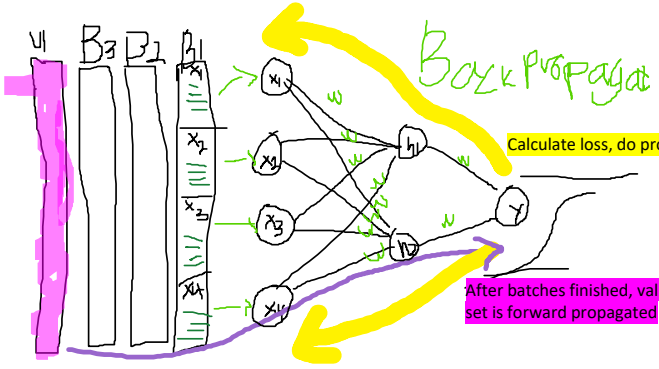
What is Epoch, Batch, Validation set

2023年6月8日 15:07

1. Epoch 1, takes the batch1 altogether all samples and runs it through the network and gives some weights to the features and produces an output
2. the output is compared to actual output and backpropagation is used to tune the weights such that it will produce these outputs.
3. batch2 comes to the network and previous weights are used as starting point to fit new data
4. similarly the weights are updated and next batch is used until all batches have finished which means one epoch has finished
5. Epoch 2 will start that repeats everything of epoch1 but starts with good weights generated during previous epoch.
6. the accuracy is improved after each epoch
7. validation set is used to after each epoch to evaluate the model on unseen data and compare it to the training accuracy to prevent overfit
8. you can tune the model hyperparameters to improve the validation accuracy
9. that means, even though the model has not seen validation data to train on, you have manually tweaked the model to match it.
10. in the end of all epochs and tunings when you are satisfied with the validation accuracy, you do the FINAL check using the test data.
11. after this step you should not tweak your model in any way because that means you are trying to match the test data and then the result won't be regarded as unseen data anymore.

Backpropagation, Epoch, Validation

2023年6月8日 15:09

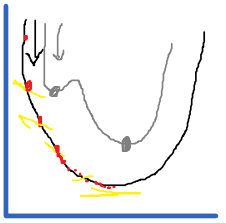


Calculate loss, do propagation → Report overall (loss, training accuracy)

After batches finished, validation set is forward propagated → Report validation (loss and accuracy)

1EPOCH

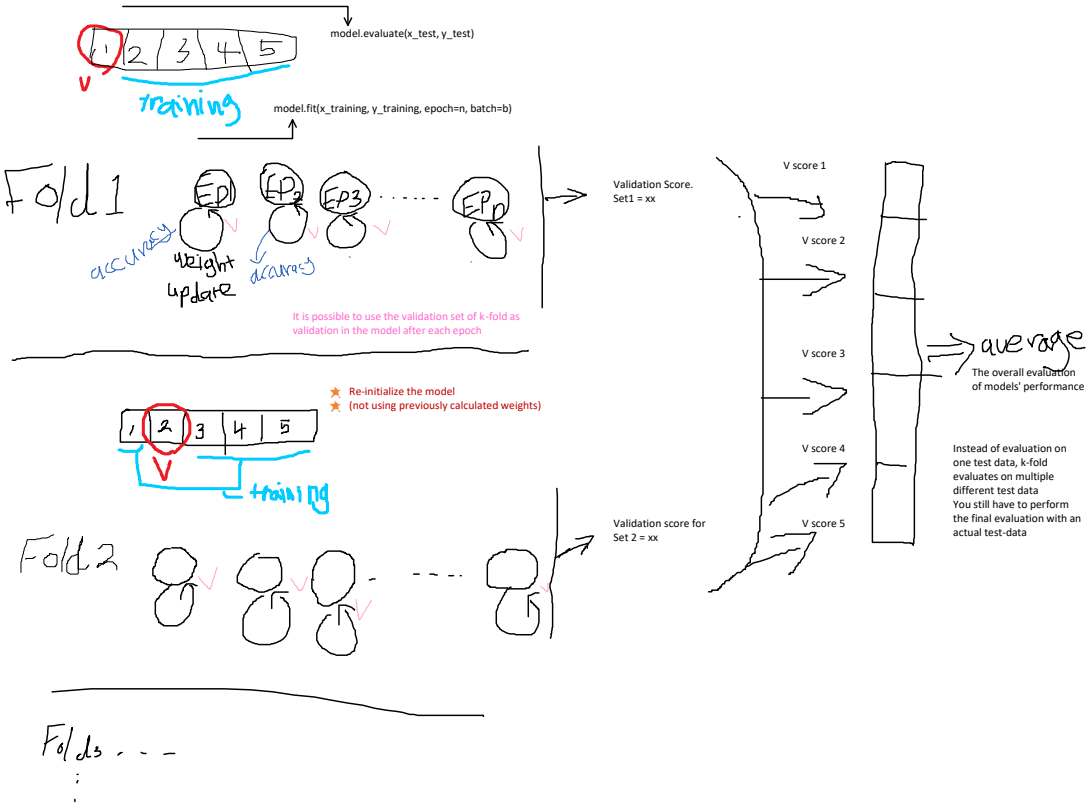
Use validation for accuracy monitoring and early stopping



Loss function SSE^2
The loss function should be minimized, that means the global minimum must be found where the slope of a tangent line is 0.
Optimization functions such as SGD, RMSP, and Adam provide different algorithms to find the optimum minimum value as fast as possible

What is K-Fold Cross Validation

2023年6月9日 11:45

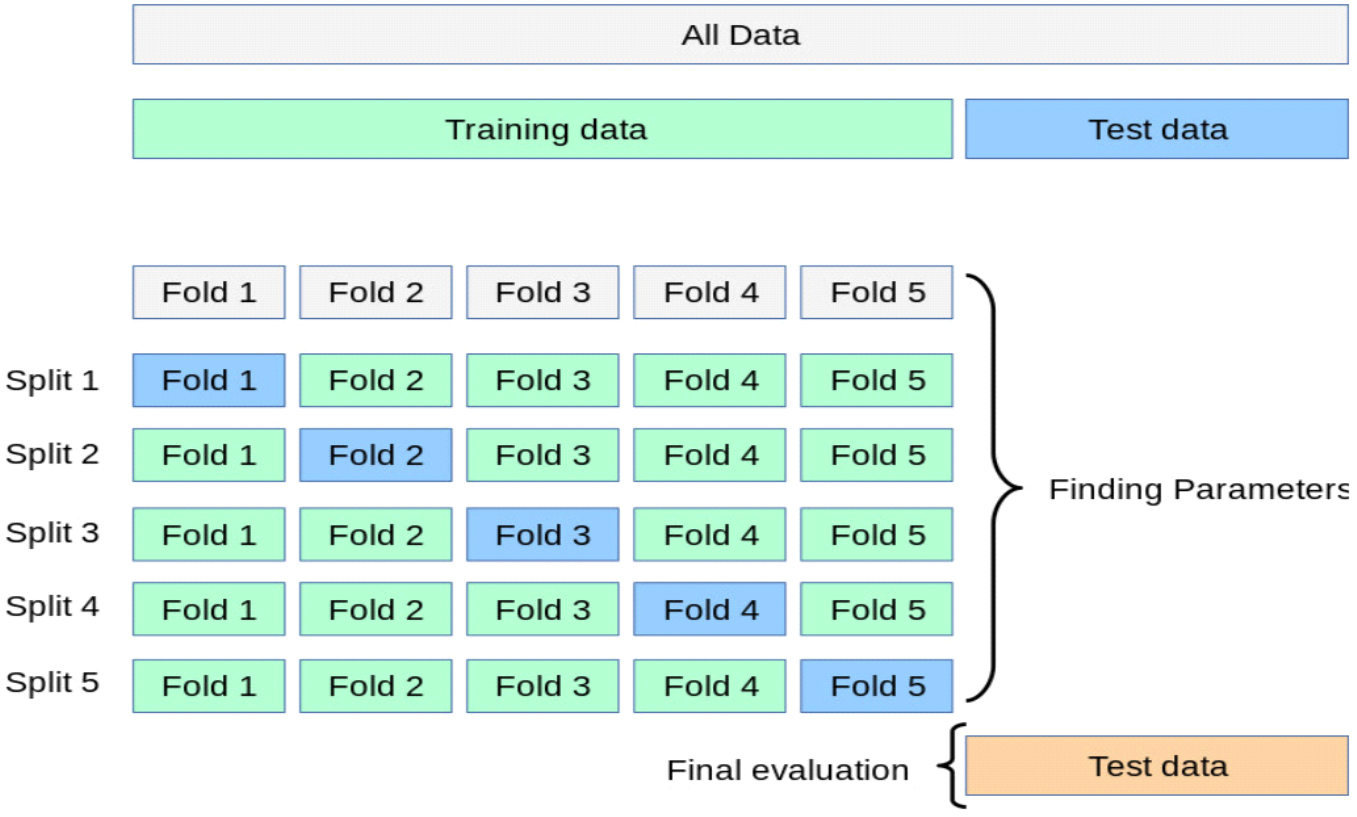


- If you're satisfied with the performance of your model, you can finalize it. There are two options for doing so:
  1. Save the best performing model instance (check "How to save and load a model with Keras?" - do note that this requires retraining because you haven't saved models with the code above), and use it for generating predictions.
     however, with cross_val_score function, we cannot store anything because this function is only used to get an idea of the model performance.
  2. Retrain the model, but this time with all the data - i.e., without making the train/test split. Save that model, and use it for generating predictions. I do suggest to continue using a validation set, as you want to know when the model is overfitting.
- Both sides have advantages and disadvantages.
  - The advantages of the first are that you don't have to retrain, as you can simply use the best-performing fold which was saved during the training procedure. As retraining may be expensive, this could be an option, especially when your model is large.
  - However, the disadvantage is that you simply miss out a percentage of your data - which may bring your training sample closer to the actual patterns in the population rather than your sample. If that's the case, then the second option is better.
- https://github.com/christianversloot/machine-learning-articles/blob/main/how-to-use-k-fold-cross-validation-with-keras.md
- How to Finalize the model
  - How to Train a Final Machine Learning Model - MachineLearningMastery.com
- KFold and test-split are two different types of cross-validation
  - Understanding 8 types of Cross-Validation | by Satyam Kumar | Towards Data Science
- Visualize the kfold split behavior
  - Visualizing cross-validation behavior in scikit-learn — scikit-learn 1.2.2 documentation

Practical code

2023年6月9日　　17:52

- Sklearns' KFold() object defined the number of folds
- Kfold.split(X=x_train, y=y_train) splits the index of the given X and y and returns a generator that you can loop through
- Suring each loop you can access train_index and test_index of the current fold
- Use the index to access the actual data in the dataframe like x_train[train_index] and x_test[test_index]

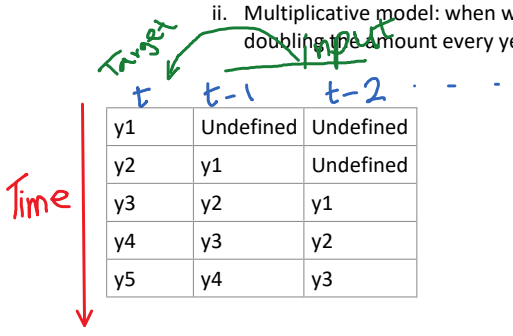


Time Series Data Analysis

2023年6月14日 8:14

1. We can resample the frequency of the time series data for example from daily to monthly and use an aggregation function on the grouping data for example take the mean of the month
2. Rolling is a method that can be used to smooth the time series data, also used in moving averages and other statistical analysis
    a. Rolling creates a window object of a specific size and applies some aggregation function.
3. Time series data has three components:
    a. General trend: upward, downward, horizontal trends
        i. The slow-moving changes in a time series general direction of a time series.
    b. Seasonal trend: repeating pattern
        i. The cycle that occur over a fixed period known frequency.
    c. Cyclical trend: no repeating pattern
        i. Fluctuations with no known frequency Business cycle.
        ii. This sequence of changes is recurrent but not periodic.

4. Hodrick-prescott filter can decompose the time series data into its' general trend and cyclic components.
    a. it has a minimization function with parameter lambda
    b. lambda = 1600 for quarterly data, 6.25 ($1600/4^4$) for annually data, and 129600 ($1600*3^4$) for monthly data
5. Error-Trend-Seasonal (ETS) decomposition, decomposes the data into general trend, seasonal, and residual which is the noise that was not explainable by trend and seasonal.
    a. It has two models
        i. Additive: when it seems that the trend is more linear and the seasonality and trend components seem to be constant over time (e.g. every year we add 10,000 passengers)
        ii. Multiplicative model: when we are increasing (or decreasing) at a non-linear rate (eg. doubling the amount every year)



y1	Undefined	Undefined
y2	y1	Undefined
y3	y2	y1
y4	y3	y2
y5	y4	y3

- Data has one feature through time for X
- Multiple other features needs to be generated for X using historic data
- Going back each step to past, we get one undefined data
- In the analysis, these undefined rows can be excluded or we can use values of t-1 in it and so on
- The model (or let's say the linear regression equation in case of AR) would be something like this:
    - $Y_t = W_{t-1}X_{t-1} + W_{t-2}X_{t-2}$
- The model will recreate the target during the training process and evaluate itself to minimize the error
- Then row by row data is used to forecast into the future
    - $Y_6 = W_5Y_5 + W_4Y_4$
    - $Y_7 = W_6Y_6 + W_5Y_5$
    - $Y_8 = W_7Y_7 + W_6Y_6$

Machine Learning (ML) gets a lot of hype, but its classical predecessors are still immensely powerful, especially in the time-series space. Error, Trend, Seasonality Forecast (ETS), Autoregressive Integrated Moving Average (ARIMA) and Holt-Winters are three Classical methods that are not only incredibly popular but are also excellent time-series predictors.

Moving Average

2023年6月14日 9:36

Moving average is used to:

- moving averages are important as they allow analysts to predict the potential direction of a stock before it happens.
- It tries to eliminate the noise to create a somewhat clearer representation of where a stock is moving.
- By calculating the moving average, the impacts of random, short-term fluctuations on the price of a stock over a specified time frame are mitigated.
- It is useful for identifying the overall direction of the data, detecting changes in trends, and reducing noise in the time series.
- 50-day and 200-day moving average figures are widely followed by investors and traders and are considered to be important trading signals.

Types of Moving Average:

1. Simple Moving Average SMA:
    - SMA is straightforward to calculate and interpret, making it a popular tool for technical analysis, forecasting, and trend analysis.
    - Use it when the trend looks linear
2. Exponential Weighted Moving Average
    - Puts more weight on the recent data compared to past data

Holt-Winters Forecasting Method

2023年6月14日 10:29

- Holt-Winters is a forecasting method that incorporates trend and seasonality components into time series data to make accurate predictions.
- Unlike moving average methods, which focus on data smoothing and trend identification, Holt-Winters explicitly models and forecasts trend and seasonality components in time series data, providing more comprehensive and accurate predictions.
- In the late 1950s, Charles Holt recognized the issue with the simple EWMA model with time series with trend. He modified the simple exponential smoothing model to account for a linear trend.
    - It consists of two EWMAs: one for the smoothed values of xt, and another for its slope. The terms level and trend are also used.
- Holt-Winter consist of Triple exponential smoothing that accounts for level, trend and seasonality.
- It can be used for forecasting and anomaly detection
- To determine whether your time series data is a good candidate for Holt-Winters or not, you need to make sure that your data:
    - Isn't stochastic. If it is random, then it's actually a good candidate for Single Exponential Smoothing.
    - Has a trend.
    - Has seasonality. In other words, your data has patterns at regular intervals. For example, if you were monitoring traffic data, you would see a spike in the middle of the day and a decrease in activity during the night. In this case, your seasonal period might be one day.
        - From <https://thenewstack.io/when-holt-winters-is-better-than-machine-learning/>
- Hold-Winters method can be used for anomaly detection
    - https://medium.com/@tle3006/single-seasonal-time-series-anomaly-detection-with-brutlags-algorithm-and-holt-winter-ets-d8aea1fd1bfc

AR/ARMA/ARIMA/SARIMA

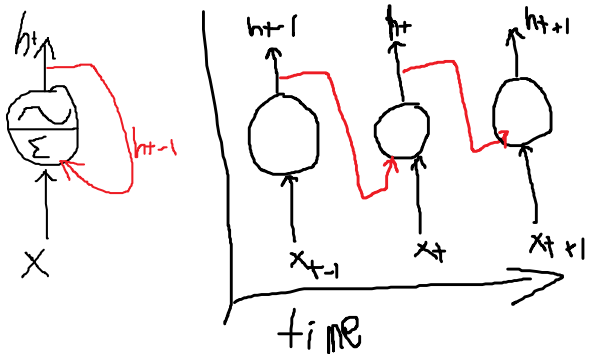
2023年6月15日　　17:42

- Just like Holt-Winters model, ARIMA models can be used to predict time series data
- It is said that Holt-Winters model is a special case of ARIMA
- ARIMA needs a non-stationary data
- AR considers the trend
- It considers the stationarity
- MA smooths out the noise
- AR is used when just having a data with trends but no seasonality
    - AR says that the output depends linearly on its own previous values and some error
- Autoarima can help in determining the parameters

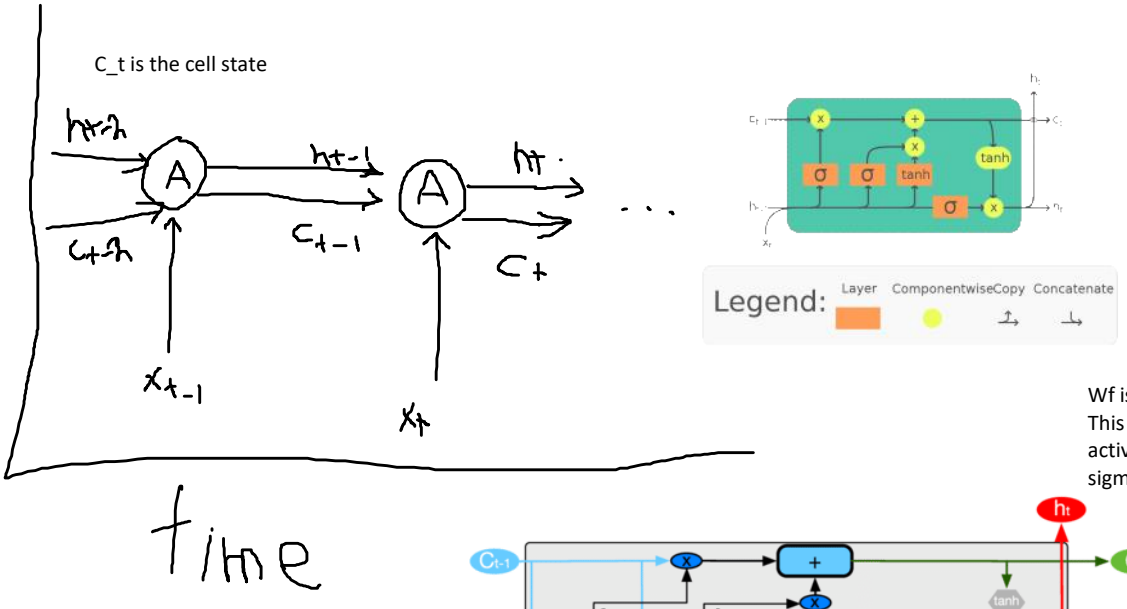
LSTM

2023年6月16日 17:20

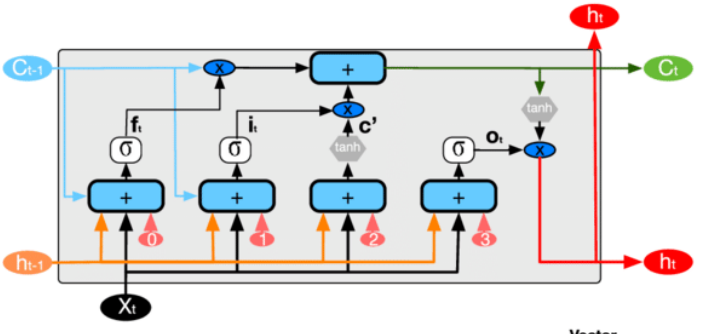
- Normal neuron in FNN (forward neural network) takes input, performs a linear operation and a non-linear activation function then produces output
- Recurrent Neuron in RNN after producing the output, it sends the output back to itself.



- RNN tends to forget the early weights or inputs after some time
- LSTM solves this issue by creating a balance between long term and short term memory in the model



Wf is weight matrix while b is bias
This linear function is given to activation function for example sigmoid function during forget layer



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$c'_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

$$c_t = f_t * c_{t-1} + i_t * c'_t$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * tanh(c_t)$$

Learn gate C'_t
i_t ignore factor

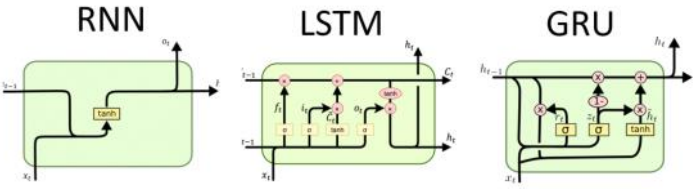
Forget gate is about long term memory
Forget factor f_t

Remember gate C_t combines data from

Forget gate is about long term memory
Forget factor f_t

Remember gate C_t combines data from short term and long term memories
Output from forget gate and from learn gate and add them together

Output gate:



Standardization or Normalization

2023年6月16日 18:53

- Feature scaling is about transforming the value of features in the similar range like others for machine learning algorithms to behave better resulting in optimal models.
- Feature scaling is not required for algorithms such as random forest or decision tree
- Standardization and normalization are two most common techniques for feature scaling.
- **Normalization** is about **transforming** the feature values to fall within the **bounded intervals (min and max)**
- **Standardization** is about **transforming** the feature values to fall around **mean as 0 with standard deviation as 1**
- Standardization maintains useful information about outliers and makes the algorithm less sensitive to them in contrast to min-max scaling

From <https://vitalflux.com/minmaxscaler-standardscaler-python-examples/#Normalization_vs_Standardization>

When to use MinMaxScaler or StandardScaler?

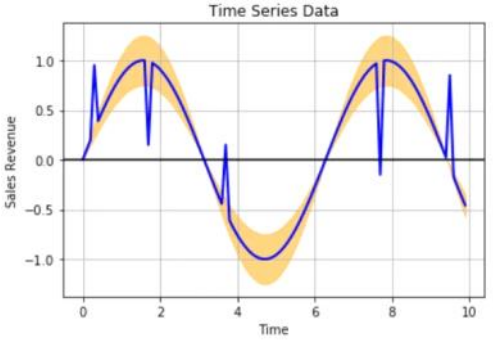
MinMaxScaler is useful when the data has a bounded range or when the distribution is not Gaussian. For example, in image processing, pixel values are typically in the range of 0-255. Scaling these values using MinMaxScaler ensures that the values are within a fixed range and contributes equally to the analysis. Similarly, non-Gaussian distributions such as a power-law, MinMaxScaler can be used to ensure that the range of values is scaled between 0 and 1.
StandardScaler is useful when the data has a Gaussian distribution or when the algorithm requires standardized features. For example, in linear regression, the features need to be standardized to ensure that they contribute equally to the analysis. Similarly, with clustering algorithms such as KMeans, StandardScaler can be used to ensure that the features are standardized and contribute equally to the analysis.
For most cases, StandardScaler is the scaler of choice. If you know that you have some outliers, go for the RobustScaler.

From <https://vitalflux.com/minmaxscaler-standardscaler-python-examples/#Normalization_vs_Standardization>

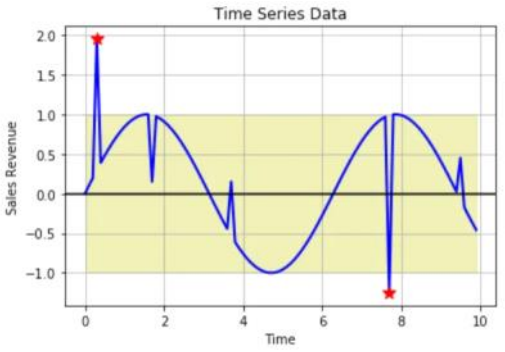
Anomaly Detection in Time Series Data

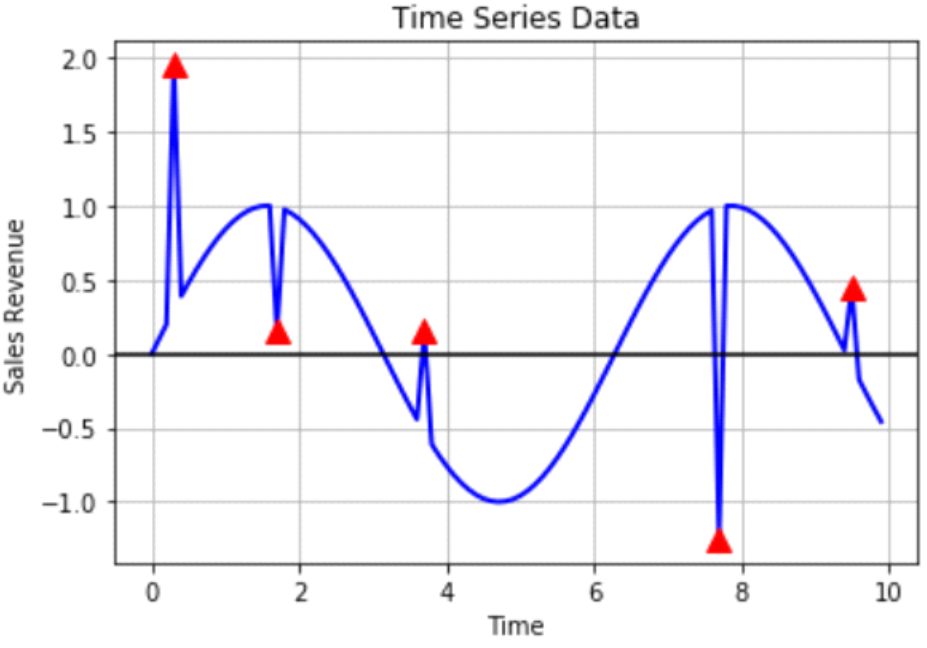
2023年6月22日 12:53

Three approaches:

1. By Predictive Confidence Level Approach
    - One way of doing anomaly detection with time series data is by building a predictive model using the historical data to estimate and get a sense of the overall common trend, seasonal or cyclic pattern of the time series data.
    - Pros:
        - The main advantage of this approach is finding local outlier
    - Cons:
        - but the main disadvantage is, this approach highly depends on the efficiency of the predictive model. Any loop hole in the predictive model can give false positives and false negatives.
    - 

2. Statistical Profiling Approach:
    - Generating a statistical model or profile of the given data can be the fastest and the most useful approach, as this method can provide a more controlled and explainable outcome.
    - This can be done by calculating statistical values like mean or median moving average of the historical data and using a standard deviation to come up with a band of statistical values which can define the uppermost bound and the lower most bound and anything falling beyond these ranges can be an anomaly
    - Example mnist_pytorch.selfcheck.ipynb:
      [10 12 23 23 16 23 21 16 35]
      Mean: 19.88888888888889
      Std: 7.062332703142535
      any sample between the following boundaries can be considered as anomaly (mean+-1.5std)
      Upper bound: 30.482387943602692
      Lower bound: 9.295389834175086
      any sample between the following boundaries can be considered as anomaly (mean+-1std)
      26.951221592031423
      12.826556185746355
    - Pros:
        - This is very effective for highly volatile time series as well, as most of the time series predictive model algorithms fail when the data is highly volatile.
    - Cons:
        - But the main drawback of this approach is detecting the local outliers.
        - A traditional window based statistical profiling can even solve this problem.

- 

3. Clustering Based Unsupervised Approach:
    - Does not require any labelled data, mentioning that a particular data point is an anomaly. So, clustering algorithms can be very handy for time series anomaly detection.
    - One common pitfall or bottleneck for clustering algorithms for anomaly detection is defining the number of clusters.
    - DBSCAN does not require any predefined number of clusters and has only two parameters (minimum number of points in a cluster and epsilon, distance between clusters).
    - Pros:
        - It does not group all data points to a cluster like conventional hard clustering techniques like K-Means.
        - DBSCAN does not group anomaly or outlier data point to any cluster and thus it becomes very easy to apply.
        - DBSCAN helps to map the “new normal” which most of the other approaches may fail.
        - HDBSCAN - Hierarchical Density-Based Spatial Clustering of Applications with Noise. Performs DBSCAN over varying epsilon values and integrates the result to find a clustering that gives the best stability over epsilon.
    - Cons:
        - Some anomaly data points if it repeats many times in a sparse interval, these might not be mapped as an anomaly according to DBSCAN.
        - in such a case, going with a rolling window based DBSCAN helps to map these local anomalies more effectively.
    - 

Udacity

2023年6月27日 16:36

Prevent overfitting

- Early stopping
- Regularization
  - L1 or L2 that used to penalize the weights
- Dropout

Local Minima

- Use random restart
  Another way to solve the local minimum problem is with **momentum**. Momentum is a constant β between 0 and 1.
  We use β to get a sort of weighted average of the previous steps:
  $\text{step}(n)+\beta\text{step}(n-1)+\beta^2\text{step}(n-2)+\beta^3\text{step}(n-3)+\ldots$

Vanishing and Exploding gradient

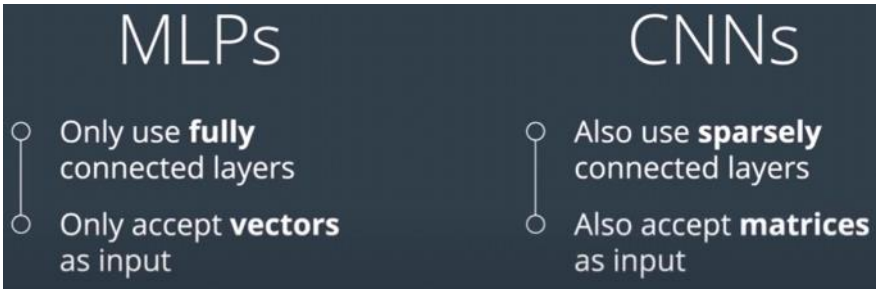
Vanishing Gradients	Exploding Gradients
Neuron weight drops	Large neuron weight
High, stalled training loss	Unstable loss

- Consider changing your activation function
- For exploding gradients, consider gradient clipping or weight regularization

CNN

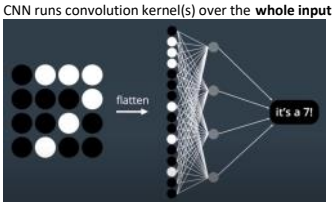
2023年7月5日　　11:16

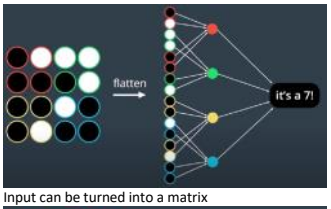
- Nowadays transformers outperform CNN based models
    - However, CNN models are still popular due to small data and low processing power requirements
- NOTE: In PyTorch it is customary to have the network output class scores (or logits) instead of probabilities like in other frameworks. Then, the PyTorch loss function itself will first normalize the scores with a Softmax function, and then compute the CCE loss.
- You can make two distinct forward method and two distinct loss function, one to be used during the training and the other to be used during the test
- This way, you can use softmax activation during test but changing the loss function to NLLLost



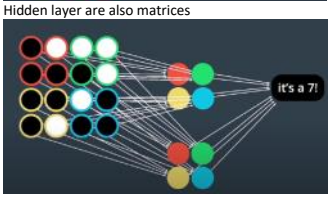
Structure of CNN

2023年7月7日　14:06

- CNNs are locally connected layers instead of a dense (fully connected) layer
- Spatial information (shape and color) of the image is preserved in CNN
- MPL neural network looks at individual inputs
- CNN can look at image as a whole or in patches
- CNN runs convolution kernel(s) over the **whole input image** and extracts multiple features
- 

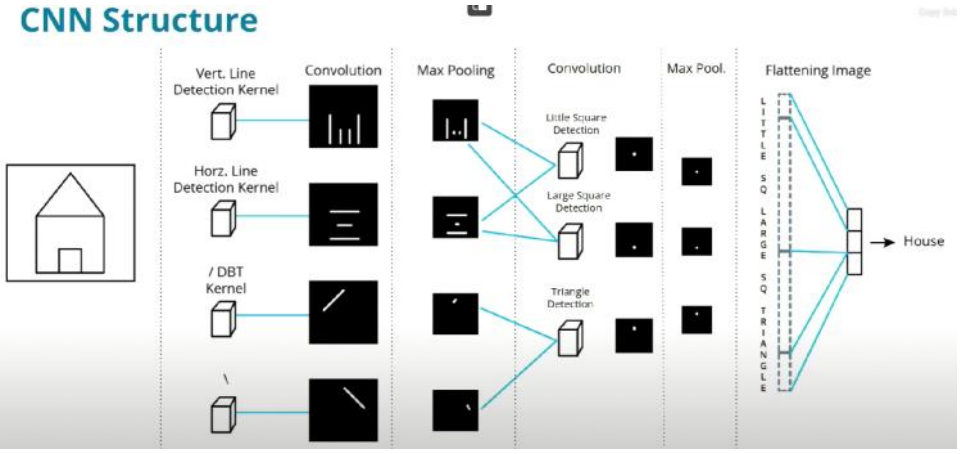
- 

- Input can be turned into a matrix
- 

- Hidden layer are also matrices
- 

-

We can see here that the convolution part is used to extract features in the form of activated pixels in the flattened array, and then the MLP decides the class based on those features.
https://adamharley.com/nn_vis/



https://www.youtube.com/watch?v=U_v_7MR0WIQ&t=181s

Image Frequency

2023年7月7日 14:23

- Frequency in images is a rate of change
- High frequency image is one where the intensity changes a lot And the level of brightness changes quickly from one pixel to the next
- Low frequency image one that is relatively uniform in brightness or changes very slowly.
- Images have high and low frequency parameters
- High-frequency components also correspond to the edges of objects in images, which can help us classify those objects.
- Edges are areas in an image where the intensity changes very quickly

- A block consisting of a convolutional layer followed by a max pooling layer (and an activation function) is the typical building block of a CNN.
- Maxpooling reduces the filtered image by taking the maximum pixel in a group of pixels and discard the others
- Receptive Field: Unlike in fully connected networks, where the value of each unit depends on the entire input to the network, a unit in convolutional networks only depends on a region of the input. This region in the input is the receptive field for that unit.
- The receptive field size of a unit can be increased in a number of ways.
    - One option is to stack more layers to make the network deeper, which increases the receptive field size linearly by theory, as each extra layer increases the receptive field size by the kernel size.
    - Sub-sampling on the other hand increases the receptive field size multiplicatively.
    - Modern deep CNN architectures like the VGG networks [18] and Residual Networks [8, 6] use a combination of these techniques.
- Not all pixels in a receptive field contribute equally to an output unit's response
    - The border pixels has less impact while center pixel has highest impact
    - Central pixels can propagate information to the output through many different paths, while the pixels in the outer area of the receptive field have very few paths to propagate its impact.
- Feature map: The result of applying a filter on an image or another feature map. One kernel/filter generates one feature map.
- Pooling: The operation of sliding a window over the input image or a feature map and applying a function to the numbers present in that window (for example, taking the maximum).

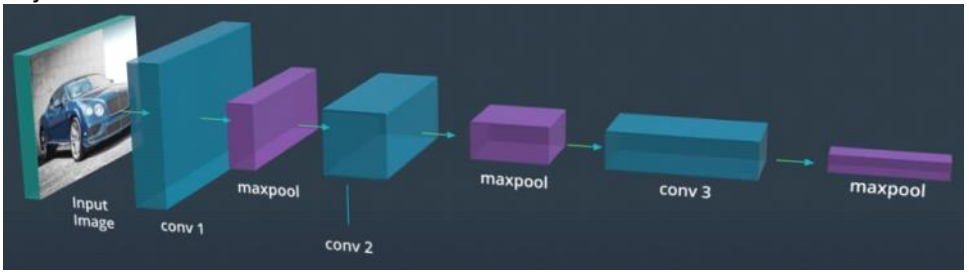
Kernels (filters/channels):

- Each filter creates a specific feature map for example a vertical edge detection or horizontal or …
- Filters can be stacked together to extract a pattern
- Another layer can use these results to extract patterns within patterns and so on
- We don't specify what pattern to extract, we let the model learn by itself using the loss function
    - That means, when we want to classify an image of a dog, we don't explicitly specify the patterns for dog, we let the model learn by changing the weights according to the weight functions
- a convolutional layer that takes feature maps with a depth of 64 and outputs 128 feature maps is said to have 64 channels as input and 128 as outputs.
- $n_p = n_k \left(c\, k^2 + 1 \right)$

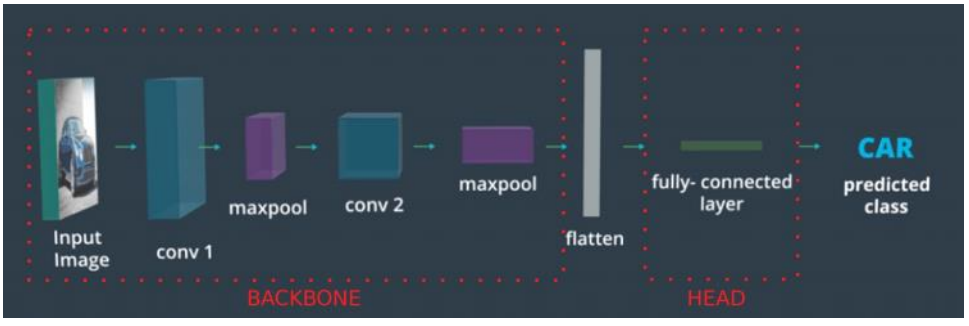
Hyperparameters:

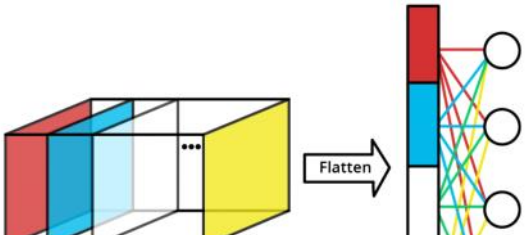
- Number of filters: Similar to increasing the number of neurons in Dense layer
- Size of filter: Size of the detected patterns
- Stride: how many pixels at a time the sliding window should move
    - Stride=1: will output the image the same size as input image (roughly, depends on what you do at the edges)
    - Stride=2: will output image the half size of input image

$$o = \left[\frac{i + 2p - k}{s}\right] + 1$$

- any image, has larger width and height compared to depth
  - RGB has width of 3
- Through the CNN layers, we increase the depth of the image to pick up more and more patterns
- We reduce the width and height to reduce dimensionality using Max-pooling.
- the initial layers focus on the constituents of the objects (edges, textures, and so on),
- the deeper layers represent and recognize more abstract concepts such as shapes and entire objects.
- 

- Increase the depth gradually in the order of power of 2
- 

- A classical CNN is made of two distinct parts, sometimes called the backbone and the head, illustrated below.
- 

Feature Vector Process:
- 

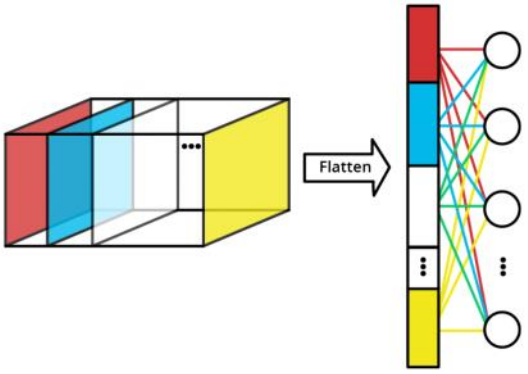
- 

Image augmentation is a very common method to:

1. Increase the robustness of the network
2. Avoid overfitting
3. Introduce rotational, translational and scale invariance as well as insensitiveness to color changes
4. Avoid **shortcut learning**
5. **AutoAugment** implements augmentation policies that have been optimized in a data-driven way, by performing large-scale experiments on datasets such as ImageNet and testing many different recipes, to find the augmentation policy giving the best result. It is then proven that these policies provide good performances also on datasets different from what they were designed for.

Batch Normalization (BatchNorm):

- Normalizes the output of feature maps between each layer and the next

Most Important Hyperparameters

2023年7月12日 17:35

The following are the most important hyperparameters to consider:

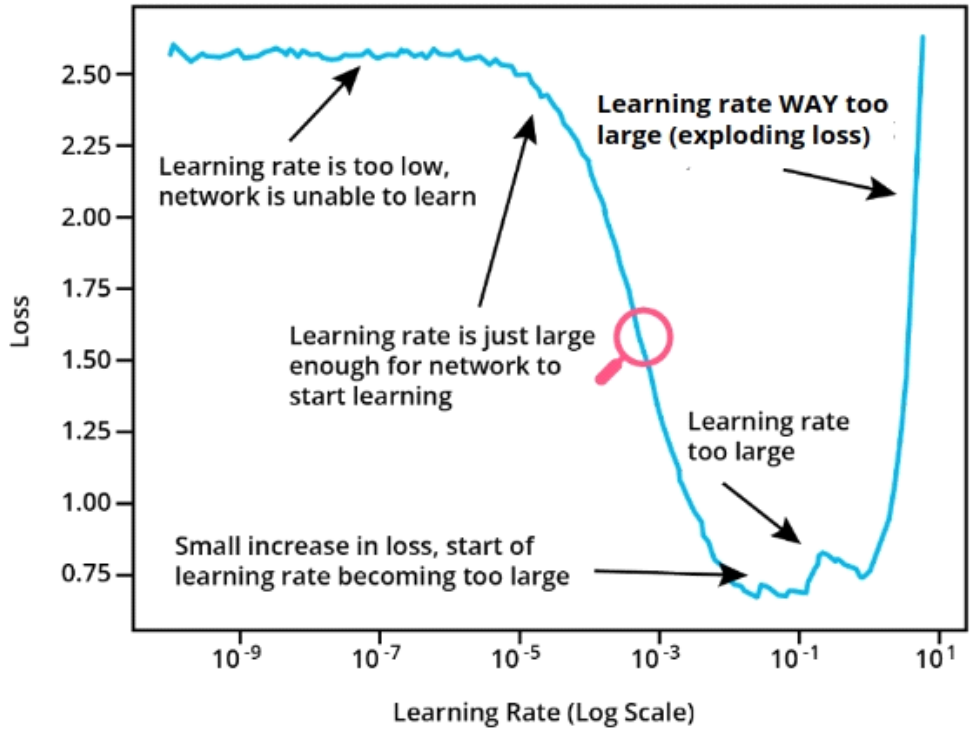
1. **Design parameters:** When you are designing an architecture from scratch, the number of hidden layers, as well as the layers parameters (number of filters, width and so on) are going to be important.
2. **Learning rate:** Once the architecture is fixed, this is typically the most important parameter to optimize. The next video will focus on this.
3. **Batch size:** This is typically the most influential hyperparameter after the learning rate. A good starting point, especially if you are using BatchNorm, is to use the maximum batch size that fits in the GPU you are using. Then you vary that value and see if that improves the performances.
4. **Regularization:** Once you optimized the learning rate and batch size, you can focus on the regularization, especially if you are seeing signs of overfitting or underfitting.
5. **Optimizers:** Finally, you can also fiddle with the other parameters of the optimizers. Depending on the optimizers, these vary. Refer to the documentation and the relevant papers linked there to discover what these parameters are.

Best way is to create Experiments and introduce list of hyperparameters using some algorithm like gridsearch and others then record the output of each experiment using
https://www.mlflow.org/docs/latest/tracking.html

```
import mlflow
 with mlflow.start_run():
         ... train and validate ...
     # Track values for hyperparameters
    mlflow.log_param("learning_rate", learning_rate)
    mlflow.log_param("batch_size", batch_size)
     # Track results obtained with those values
    mlflow.log_metric("val_loss", val_loss)
    mlflow.log_metric("val_accuracy", val_accuracy)
     # Track artifacts (i.e. files produced by our experiment)
    # For example, we can save the weights for the epoch with the
    # lowest validation loss
    mlflow.log_artifact("best_valid.pt")
```

Learning rate:

- One useful tool to discover a good starting point for the learning rate is the so-called "learning rate finder.
- It scans different values of the learning rate, and computes the loss obtained by doing a forward pass of a mini-batch using that learning rate. Then, we can plot the loss vs. the learning rate and obtain a plot similar to this:

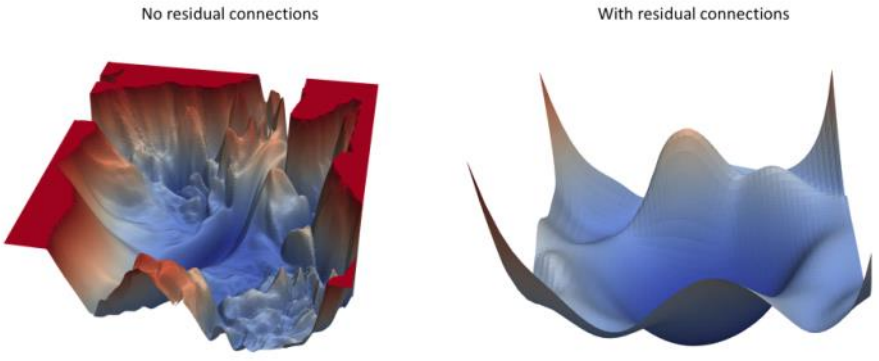


Learning rate scheduler:

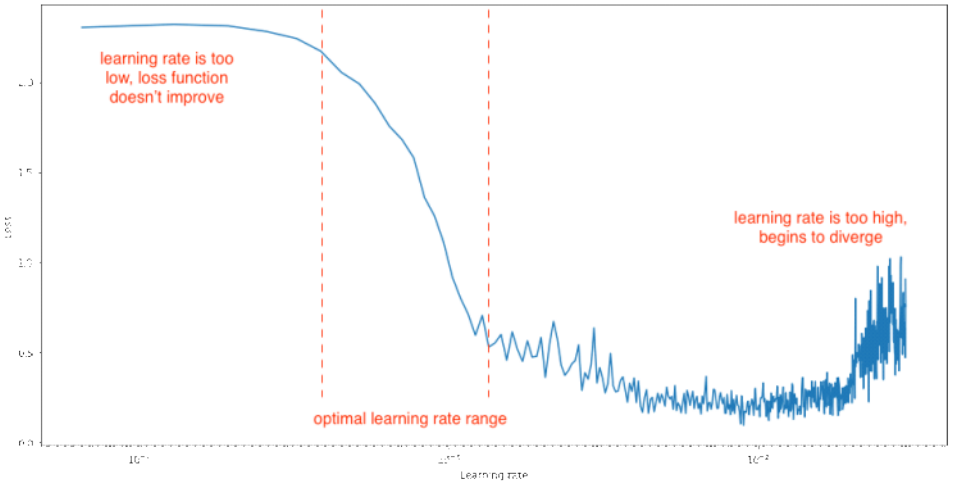
- Learning rate scheduler can be used to automatically decay the learning rate after each step:

Optimizing Learning Rate

2023年7月21日　　9:14

- https://www.jeremyjordan.me/nn-learning-rate/
- The optimal learning rate will be dependent on the topology of your loss landscape, which is in turn dependent on both your model architecture and your dataset. While using a default learning rate (ie. the defaults set by your deep learning library) may provide decent results, you can often improve the performance or speed up training by searching for an optimal learning rate.
- 

  Same general network architecture

- For learning rates which are too low, the loss may decrease, but at a very shallow rate. When entering the optimal learning rate zone, you'll observe a quick drop in the loss function. Increasing the learning rate further will cause an increase in the loss as the parameter updates cause the loss to "bounce around" and even diverge from the minima. Remember, the best learning rate is associated with the steepest drop in loss, so we're mainly interested in analyzing the slope of the plot.
- 

- Setting a schedule to adjust your learning rate during training

Transfer Learning

2023年7月21日　　17:26

What is Transfer Learning:

- Transfer learning is a technique that allows you take a neural network that has been already trained of one of these very large datasets, and tweak it slightly to adapt it to a new dataset.
- Using a pre-designed and pre-trained architecture instead of designing your own gives you most of the time the best results, and also saves a lot of time and experimentation.

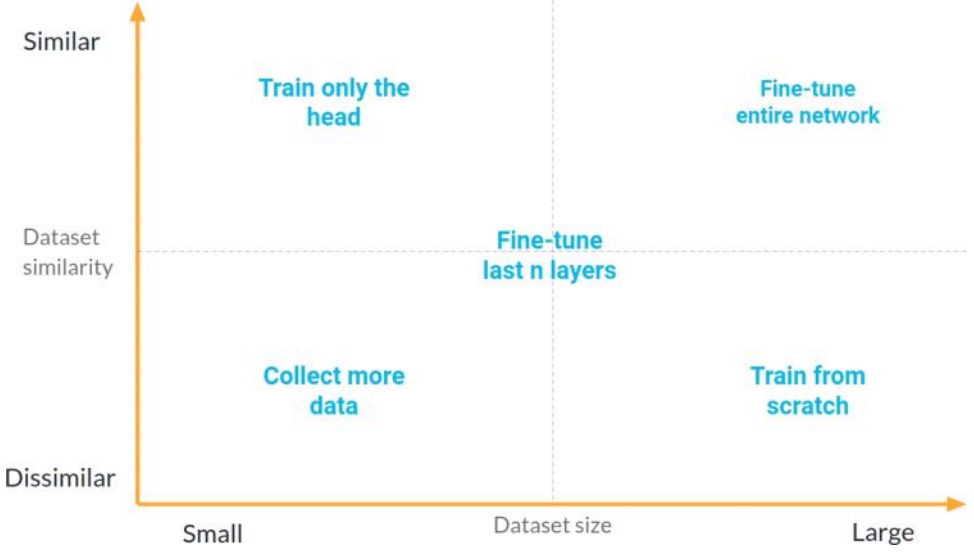
How to stop a layer in the pre-trained network:

- Last layer of the network can be frozen as follows:

```
for param in model.fc.parameters():
  param.requires_grad = False
```

-

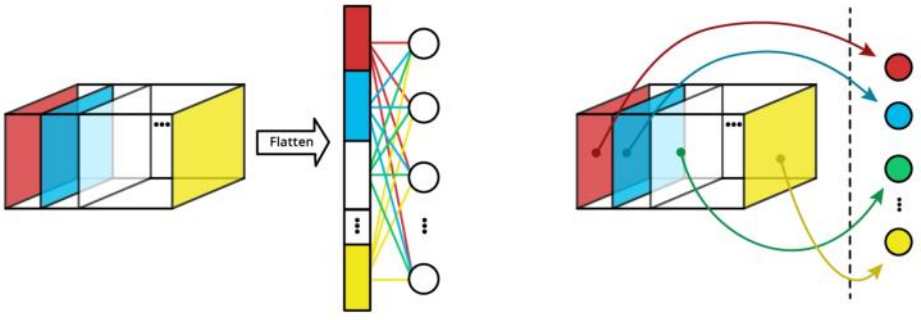
Fine Tuning:

- Transfer learning changes on the type and size of the new data
- 

-
- **Timm library** can be used to use pre-trained models and easily change the head to match your needs

General Average Pooling (GAP)

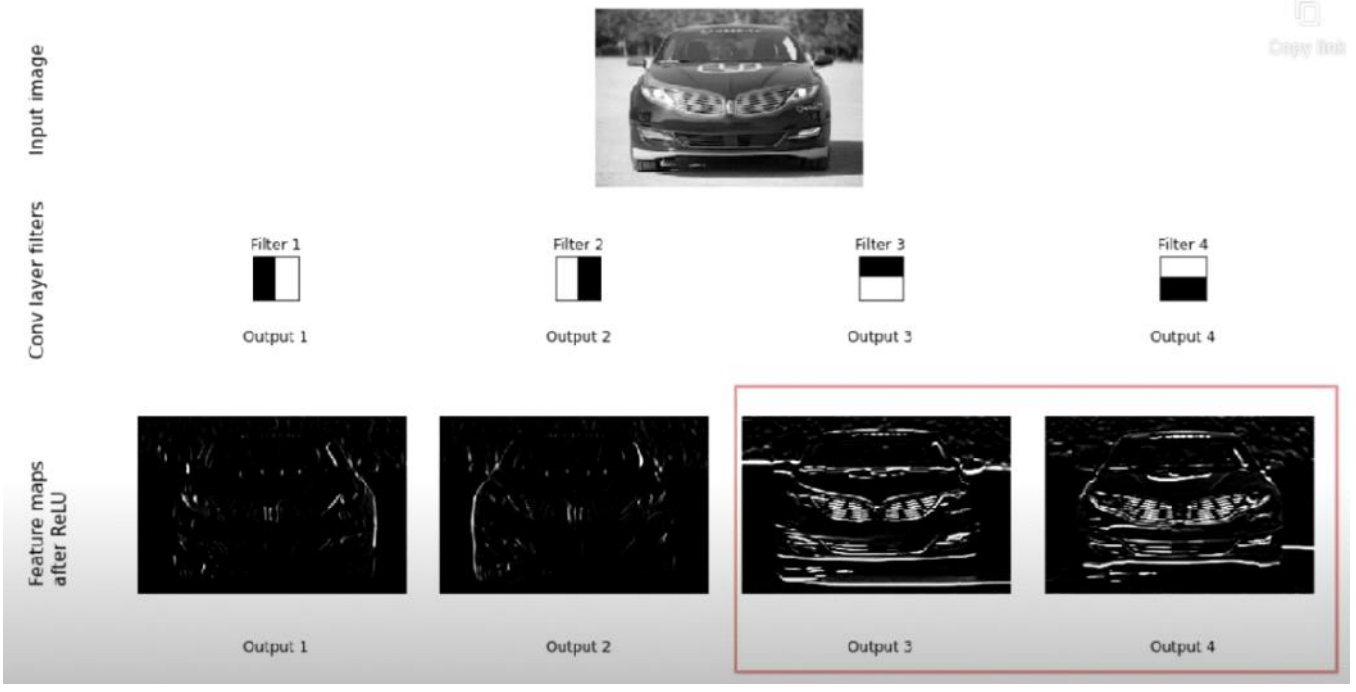
2023年7月21日 10:18

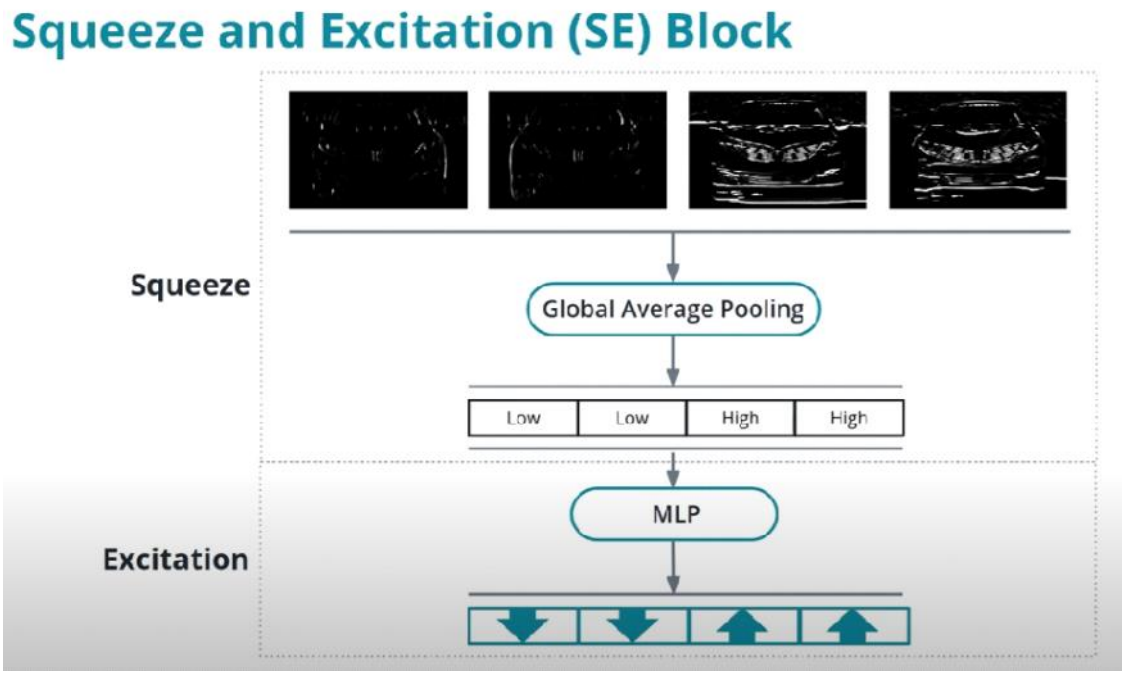
- ResNet architecture allows for a very deep neural network while avoiding the vanishing gradient
- ResNet skips connections so that very deep layers stay connected to upper layers
- Global Average Pooling GAP
    - Uses pooling of windows size same as feature map size
    - 

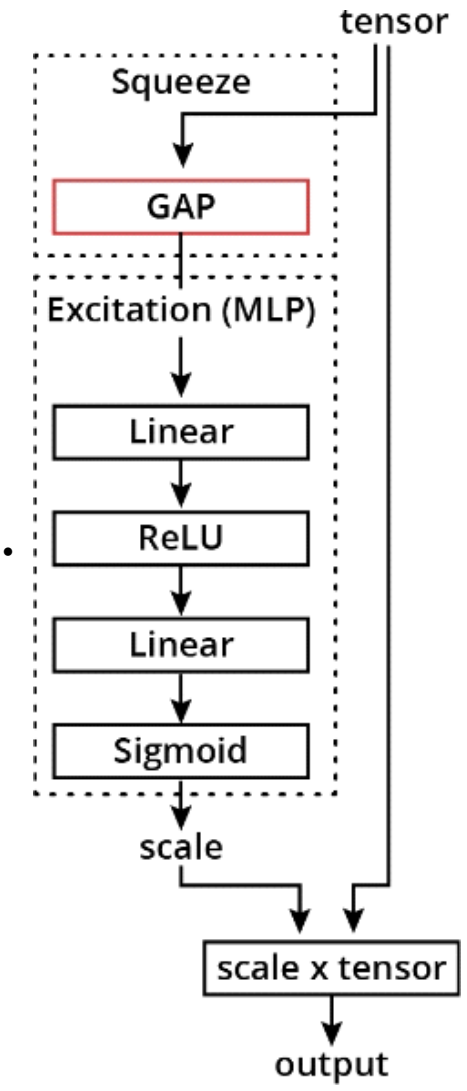
    - With GAP, the input image can have any size because this will not influence the number of feature maps, but only their height and width.

Channel Attention: Squeeze and Excitation

2023年7月21日　　11:09

- Not all feature maps have the same importance
- We need to teach that to the model
- 

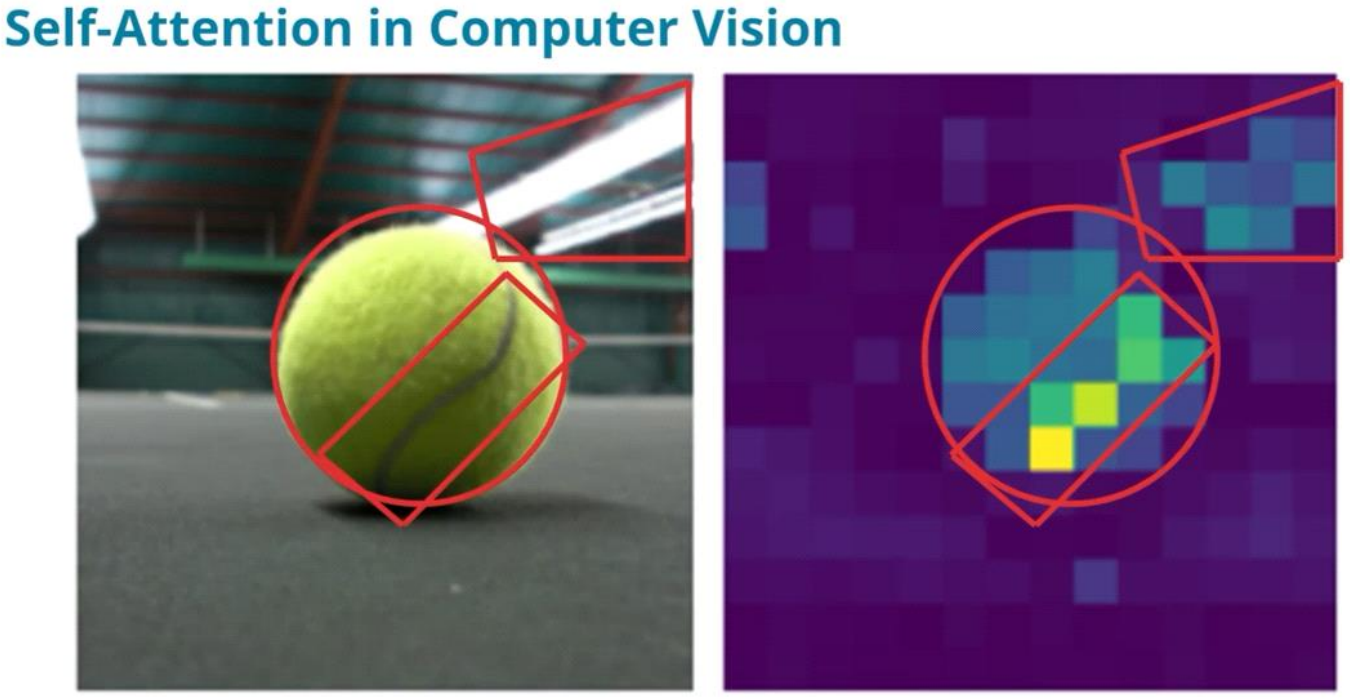
- 


tensor
Squeeze
GAP
Excitation (MLP)
Linear
ReLU
Linear
Sigmoid
scale
scale x tensor
output

Transformers

2023年7月21日　　11:24

- Coming from NLP, transformations can focus the attention on spatial information
- Self-attention is one common method used to narrow the focus on relevant information
- For example to understand what is the ball and is it outdoor or indoor environment, the self-attention creates this spatial information where the relevancy is computed

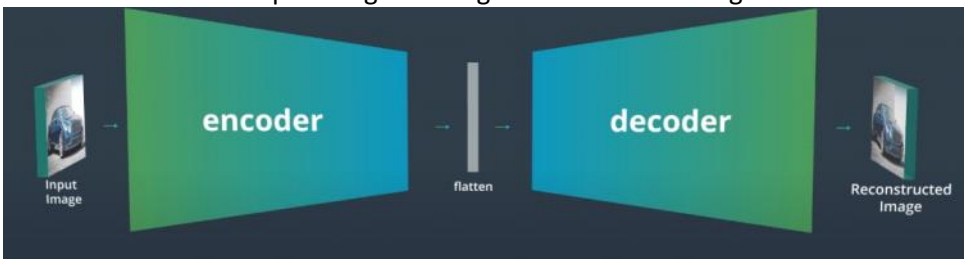


- Summarizing, there are currently 3 categories of computer vision models:

- Pure CNN architectures - still widely used for the majority of real-world applications. Examples: EfficientNet V2, ConvNeXt
- Pure Vision Transformers - currently widely used in academic environments and in large-scale real-world applications. Examples: ViT, Swin V2
- Hybrid architectures that mix elements of CNNs with elements of Transformers. Example: CoatNet

Auto-Encoder

2023年7月25日 10:17

- Autoencoder is unsupervised learning model
- Autoencoders have a similar backbone to CNN (called encoder in this context)
- It produces a feature vector (called embedding in this context).
- However, they substitute the fully-connected layers (the head) with a decoder stage whose scope is to reconstruct the input image starting from the embeddings
- 

- S

We can use autoencoders to:

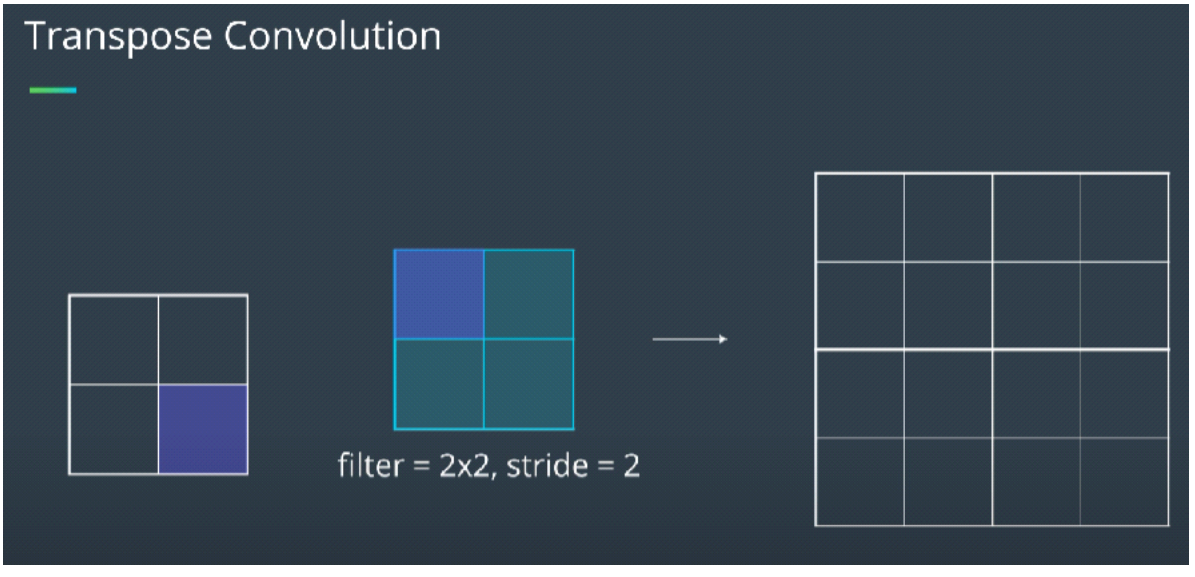
- Compress data
- Denoise data
- Find outliers (do anomaly detection) in a dataset
- Do inpainting (i.e., reconstruct missing areas of an image or a vector)
- With some modifications, we can use autoencoders as generative models - models capable of generating new images

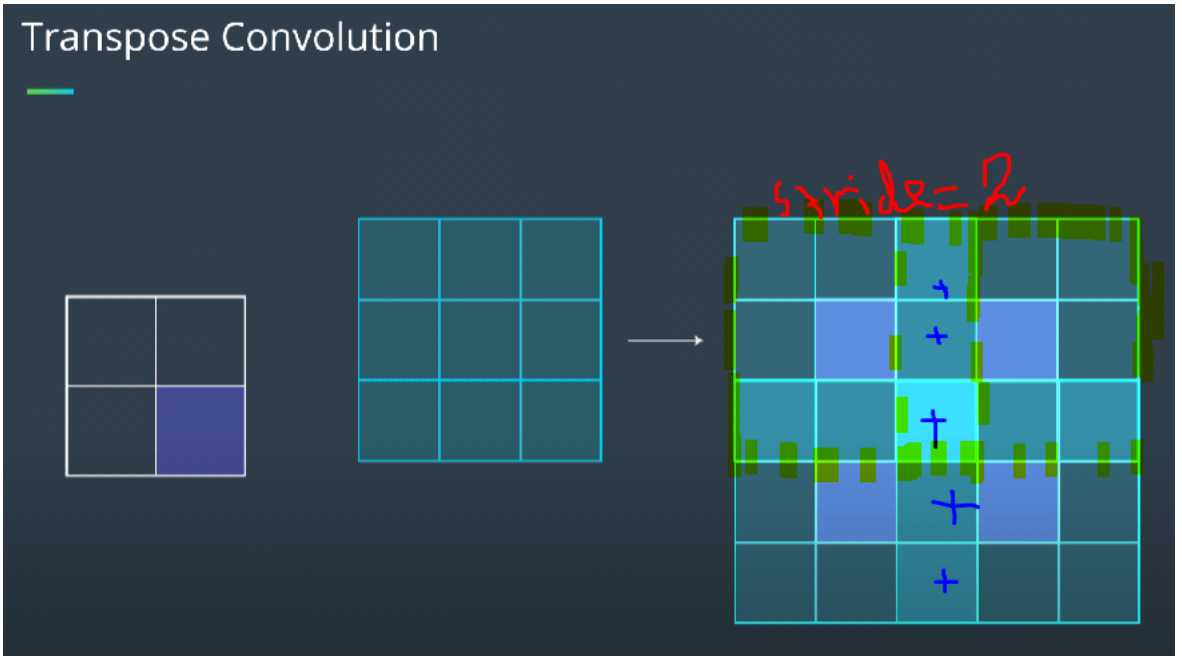
Dense Autoencoder

- Consists of some fully connected layers and reverse of it during decoder

CNN Autoencoder

- Use CNN for the encoder to downsample the image and finally flatten it
- Use upsampling to increase the size of the data
  - Upsampling is done with learnable parameters instead of designated formulas
  - Upsampling is called transpose convolution
- Convolutional Autoencoder can be extended to generate new samples.




Transpose Convolution
stride=2

Object Detection and Segmentation

2023年7月25日　　16:32

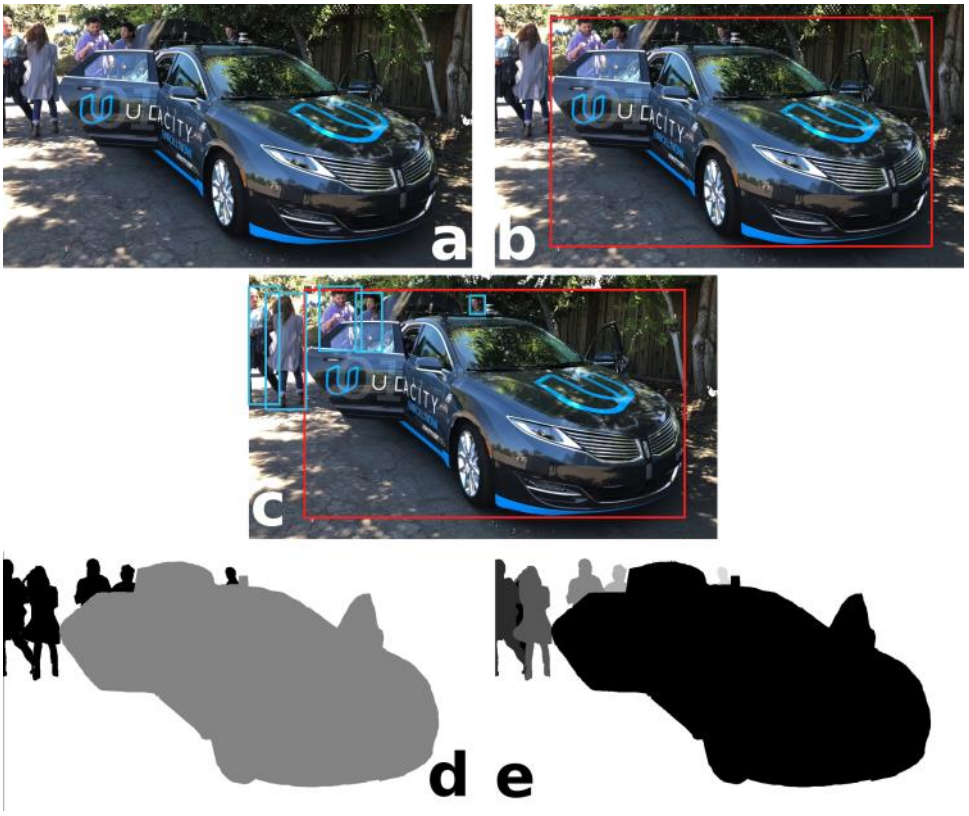
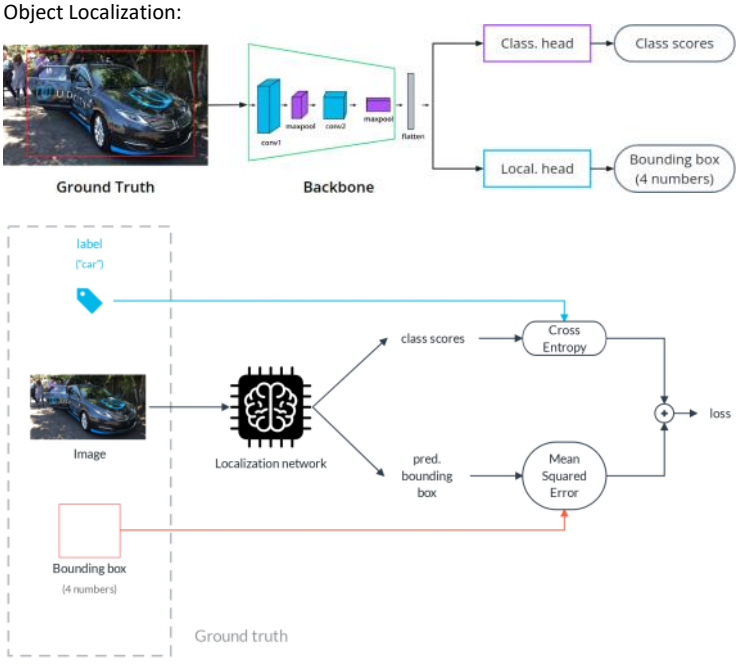


Image	Task	Objective
A	Image classification	Assign one or more labels to an image
B	Object localization	Assign a label to most prominent object, define a box around that object
C	Object detection	Assign a label and define a box for all objects in an image
D	Semantic segmentation	Determine the class of each pixel in the image.
E	Instance segmentation	Determine the class of each pixel in the image distinguishing different instances of the same class **here**.

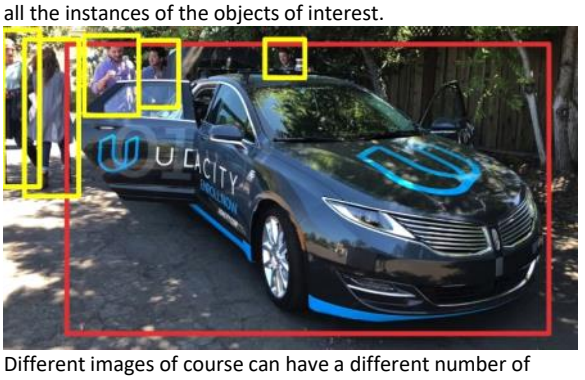
Architecture of Object Localization Networks

2023年7月26日 13:12

Object Localization:



Object Detection

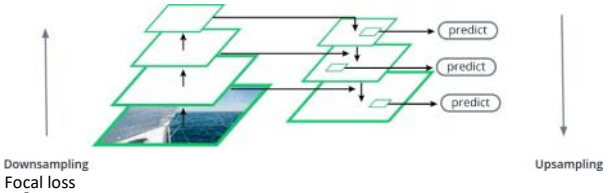
- The task of object detection consists of detecting and localizing all the instances of the objects of interest.
- 
- Different images of course can have a different number of objects of interest.
- In each one of these cases, we need the network to output one vector of class scores plus 4 numbers to define the bounding box for each object.
- How do we build a network with a variable number of outputs?
    - Variable number of outputs just means that normally we feed one image to the network to get the class and bounding box for one object in the image, but for object detection, we may have multiple objects, thus the network needs to output class and bounding box for each of the objects in that one image.
- One way would be to slide a window over the image,
- for each location of the image we run a normal object localization network.
- This sliding window approach works to a certain extent, but is not optimal
    - because different objects can have different sizes and aspect ratios. Thus, a window with a fixed size won't fit well all objects of all sizes.
- Nowadays there are two approaches to solving the problem of handling a variable number of objects, and of their different aspect ratios and scales:
    1. One-stage object detection:
        - We consider a fixed number of windows with different scales and aspect ratios, centered at fixed locations (anchors).
        - The output of the network then has a fixed size.
        - The localization head will output a vector with a size of 4 times the number of anchors,
        - The classification head will output a vector with a size equal to the number of anchors multiplied by the number of classes.
    2. Two-stage object detection
        - In the first stage, an algorithm or a neural network proposes a fixed number of windows in the image.
        - These are the places in the image with the highest likelihood of containing objects.
        - Then, the second stage considers those and applies object localization, returning for each place the class scores and the bounding box.

RetinaNet

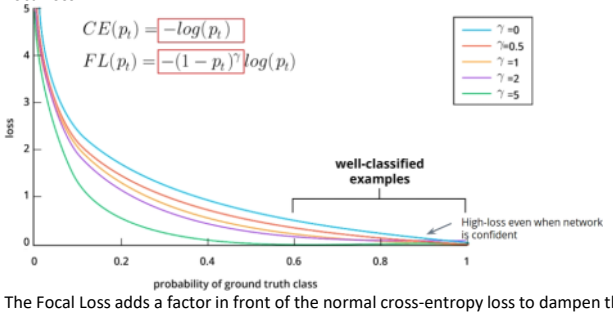
2023年7月31日 13:01

RetinaNet is characterized by three key features:

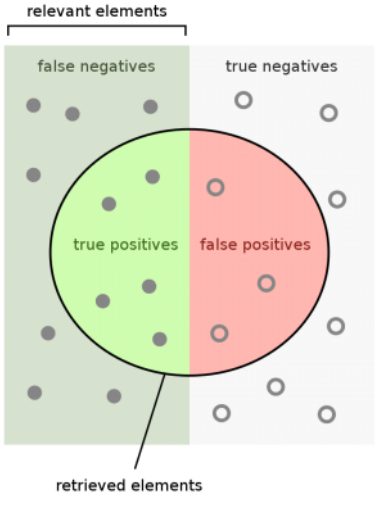
1. Anchors
2. Feature Pyramid Networks

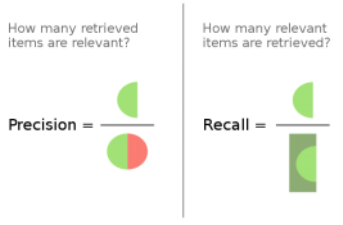


3. Focal loss

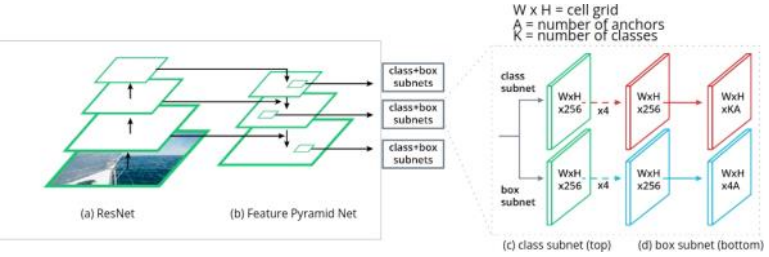


The Focal Loss adds a factor in front of the normal cross-entropy loss to dampen the loss due to examples that are already well-classified so that they do not dominate. This factor introduces a hyperparameter γ: the larger γ, the more the loss of well-classified examples is suppressed.





The RetinaNet Architecture



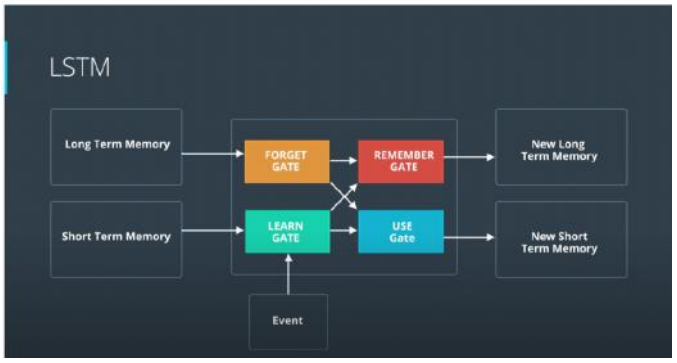
RNNs & Transformers

2023年8月29日 11:35

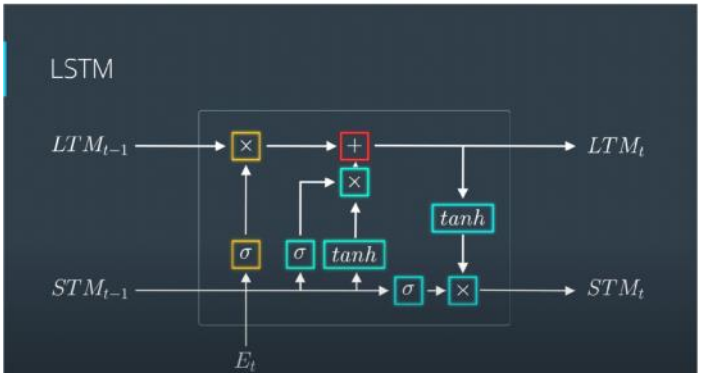
LSTM

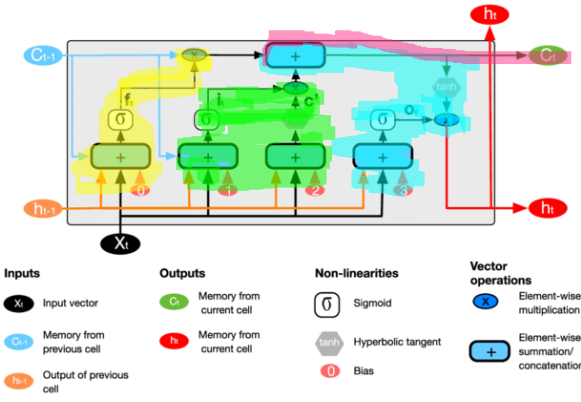
2023年8月29日　11:36

Putting it All Together



Putting it All Together





$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i)$$

$$c'_t = tanh(W_C.[h_{t-1}, x_t] + b_C)$$

$$c_t = f_t * c_{t-1} + i_t * c'_t$$

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * tanh(c_t)$$

A few points regarding LSTM:

- LSTM network is fully connected
- The hidden layer size is like the number of features for each input (character, number, word, ...).
- The input sequence means the lagged time to look back by the model.
  - For example, assume we have temperature daily data for a year and we want to lagged time by a week
  - Having only one batch, we should feed the model a data of 7 days window and let the model to predict 8th day
  - We can also have 2 batches. In this case, divide the data by two and feed the first 7 days of both batches to the model.
  - So we iterate over the batches using a specified window size unlike NN where batch size refers to the number of rows extracted and fed to the model at each iteration
- In this case, each batch has the same sequence of data because the same window size is used.
  - We better say each ROW not each BATCH, because we iterate through this whole data that is divided into a specific number of rows, and we take a sequence of elements in each row according to the window size.
    - The collected data is a batch of data, of size "number of rows or batch size" with window size "sequence length"
- So the mechanism of batching is quite different between time series data and non-time series data.

Batching in word by word text data

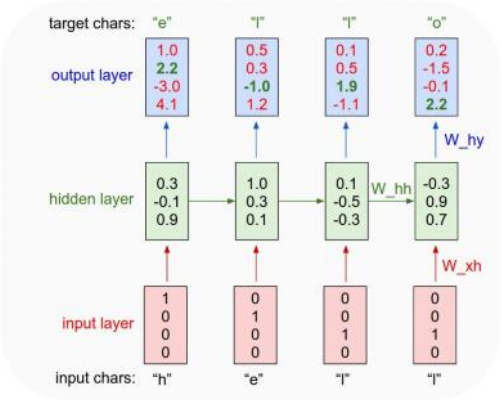
- Processing a writing is different than a numerical time series data or character by character time series data in the way that the window size will be different for each batch.
- You can still use the fixed window size but it may not be appropriate depending on the objective of the model.
  - For example, in machine translation, the model needs to comprehend the whole sentence to be able to predict sequence of words in the target language.
  - If you use fixed window size, you may get a correct literal translation for each word, but the result will be wrong grammatically.
    - Assume having a window size of 1 word, "I like watermelon" will become "私は　好き　スイカ"
- The window size in word by word data in a Seq2Seq RNN model is determined by the length of the sentence.
- We read a bunch of sentences at each iteration according to the batch size.
- We pad the shorter sentences according to the longest sentence in the current batch.

Two ways to create batches in time series data:

1. From batch size to sequence:
   a. Divide the number of rows in the data by the batch size to get the number of batches.
   b. Shape the data into [batch_size, -1] so that you get a dataset where you have correct number of rows with respect to batch size
   c. Then, loop through the data using sequence windows as steps for the loop, reading a window_size elements.
      i. Or you may loop through the data in one step at a time reading a window_size elements
      ii. This actually depends on what you want to do like to overlap the batches or not
         1) https://stackoverflow.com/questions/58276337/proper-way-to-feed-time-series-data-to-stateful-lstm
2. From Sequence to batch size
   a. Divide your data into sequences. That means create sequence number of rows.
   b. Now according to your batch size you can read the number of rows and feed it to the model.
   c. The number of rows will represent the batch size.
   d. Again, you can overlap the rows by moving one row at a time or, moving a bunch of rows at a time.

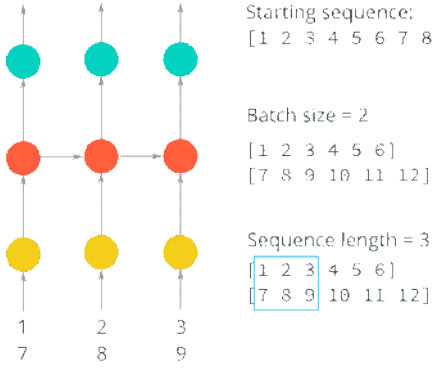
Character Level LSTM

2023年9月11日　　14:00



- Insert a sequence of characters to predict the next character
- Data is one-hot encoded
- Data is transformed into integers to make computation easy

Making training mini-batches:

- To train the network, we want to create mini-batches
- Batches need to be a multiple sequences of some desired number of sequence steps.
  - Sequence steps is the time lag we want the model to look back.
- 

1. The first thing we need to do is discard some of the text so we only have completely full mini-batches.

Each batch contains $N \times M$ characters, where N is the batch size (the number of sequences in a batch) and M is the seq_length or number of time steps in a sequence. Then, to get the total number of batches, K, that we can make from the array arr, you divide the length of arr by the number of characters per batch. Once you know the number of batches, you can get the total number of characters to keep from arr, $N * M * K$.

2. After that, we need to split arr into N batches.

You can do this using arr.reshape(size) where size is a tuple containing the dimensions sizes of the reshaped array. We know we want N sequences in a batch, so let's make that the size of the first dimension. For the second dimension, you can use -1 as a placeholder in the size, it'll fill up the array with the appropriate data for you. After this, you should have an array that is $N \times (M * K)$.

3. Now that we have this array, we can iterate through it to get our mini-batches.

The idea is each batch is a $N \times M$ window on the $N \times (M * K)$ array. For each subsequent batch, the window moves over by seq_length. We also want to create both the input and target arrays. Remember that the targets are just the inputs shifted over by one character. The way I like to do this window is use range to take steps of size n_steps from 00 to arr.shape[1], the total number of tokens in each sequence. That way, the integers you get from range always point to the start of a batch, and each window is seq_length wide.

From <https://oclux0019.itec.mazda.co.jp/jupyter/lab/tree/udacity/RNN/char-rnn/Character_Level_RNN_Exercise.ipynb>

For example:

- Assume having a total of 1024 characters
- Let batch size (N) = 50
    - --> that is each batch contains 50 sequences.
- Let sequence step length (M) = 5
    - --> that is we want to look back 5 steps.
- Now, each batch contains 50 sequences of length 5 --> 50 x 5
- Total number of batches K = total length/(NxM)
    - 1024/(50*5) = 4.096 ~ approx. 4
    - K = 4

Hyperparameters

2023年9月12日 13:42

Optimizer (Training) Hyperparameters

- Learning rate
- Minibatch size
- Training epochs

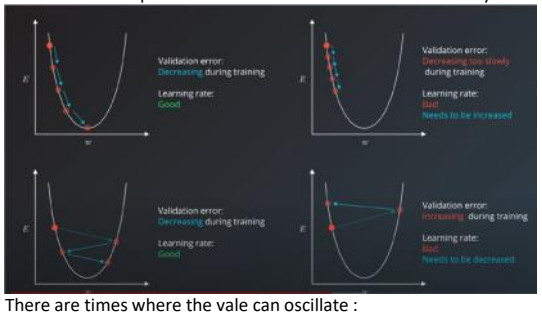
Model Hyperparameters

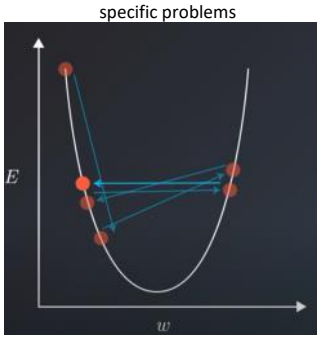
- Number of layers and hidden units
- Model specific parameters

Must read:
http://karpathy.github.io/2019/04/25/recipe/

From <https://learn.udacity.com/nanodegrees/nd101/parts/cd1822/lessons/343b4aef-0c21-4dd8-b2a4-d72fa777ca41/concepts/57e4086c-bf6a-41ad-a625-9e7430558fea>

Learning rate:

- Most important hyperparameter
- If data is normalized then 0.01 would be a good start
  - Other values to consider include 0.1, 0.001, 0.0001, 0.000001
- Below is an example of one dimensional model which has only one neuron
- 

- There are times where the vale can oscillate :
- Still the value is better than error at first but still not good
- To overcome this problem we should use **Learning rate decay**
  - In this method the model will decrease the learning rate (linearly, exponentially) when the validation error not improved for a certain number of epochs
- New approaches have **adaptive learning rate**
  - In this approach, the learning rate decrease or increase according to the data and problem.
  - **Adam Optimizer**
    - You may need to tune hyperparameters like the learning rate (lr) and decay rates (beta1, beta2) for specific problems
- 

Minibatch size:

- Minibatch size can speed-up the calculation but may decrease accuracy.
- Intuition: Large minibatch sizes can increase the speed of training. However, they often get caught at local minima during the training process.
- Common sizes are:
  - 1, 2, 4, 8, …, 256

Training Epochs:

- SessionRunHook in tensorflow can be used to do the early stopping

RNN Hyperparameters

2023年9月12日 16:48

- Choosing cell type
  - Vanilla RNN
  - LSTM
  - GRU
- How deep:
  - Number of stacks
  - 2 stacks are enough
  - Speech recognition
    - 5 to 7
- Word embedding size also matters

Example RNN Architectures

Application	Cell	Layers	Size	Vocabulary	Embedding Size	Learning Rate	
Speech Recognition (large vocabulary)	LSTM	5, 7	600, 1000	82K, 500K	--	--	paper
Speech Recognition	LSTM	1, 3, 5	250	--	--	0.001	paper
Machine Translation (seq2seq)	LSTM	4	1000	Source: 160K, Target: 80K	1,000	--	paper
Image Captioning	LSTM	--	512	--	512	(fixed)	paper
Image Generation	LSTM	--	256, 400, 800	--	--	--	paper
Question Answering	LSTM	2	500	--	300	--	pdf
Text Summarization	GRU		200	Source: 119K, Target: 68K	100	0.001	pdf

From <https://learn.udacity.com/nanodegrees/nd101/parts/cd1822/lessons/343b4aef-0c21-4dd8-b2a4-d72fa777ca41/concepts/572e4447-a642-4abd-840a-bedac9ab02c6>

Seq2Seq

2023年9月12日 17:30

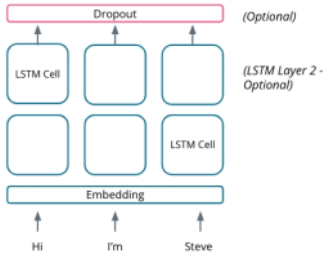
- Seq2Seq allows revolutionized machine translation by google
- It uses LSTM and encoder/decoder
- The model receives a sequence of input and produces a sequence of output.
  - It can be used to translate sequence of input to another language
  - It can be used to produce a reply to the input like chatbot
  - Speech recognition
  - Solving differential equations
  - Text summarization
  - Image captioning

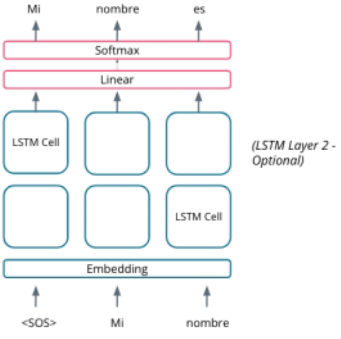
Building a vocabulary:

- SOS token EOS token: start/end of sequence
- Tokenizer to
  - uniquely identify words
  - Reduce variants of the word to get to its root

Seq2Seq Architecture

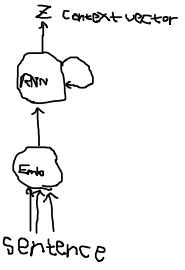
- Encoder: Input -> Embedding -> LSTM
  - Some inlcude a pretrained embedding to the input
- Decoder: Embedding -> LSTM -> output

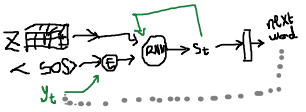




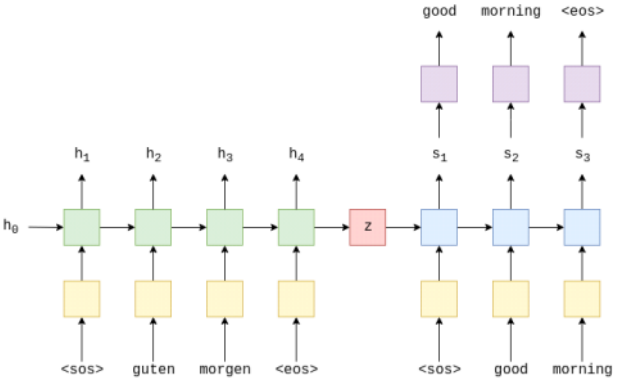
Visualizing Seq2Seq
http://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/

- The basic idea of encoder-decoder is to compress information and decompress it.
- Here, we compress the whole sentence into a context vector Z
- Then we use the Z to decompress information and produce proper output.





The y_t is either the generated *next word* or it is the ground truth target word (also called teacher forcing).

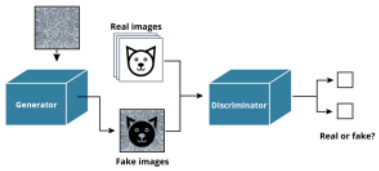


- Once the final word, x^T, has been passed into the RNN via the embedding layer, we use the final hidden state, h^T, as the context vector, i.e. $h^T = x^T$.
- This is a vector representation of the entire source sentence.
  - Because it is the result of RNN which accumulates previous information to produce this final output.
- Z is used as input to the decoder to get the output/target sentence.
- With seq2seq, we want the model to learn how to produce the output based on the feature map of input.
- Meaning, we teach the model which word translates to which word in what context even when the words have similar meaning.
- All thanks to the embedding, after enough training, the model can pickup the rules by itself as to identify the use of correct words in correct context.
  - That is why transformers get rid of RNN layer and use pure embedding.
- In the decoder, we need to go from the hidden state to an actual word, therefore at each time-step we use s_t to predict (by passing it through a Linear layer, shown in purple) what we think is the next word in the sequence, $yhat_t$.

GAN

2023年10月10日 10:40

https://poloclub.github.io/ganlab/



- Loss function is BCEWithLogitsLoss
  - It combines the sigmoid and binary cross entropy
  - So we don't need sigmoid at the output layer of discriminator
- LeakyReLU is used as activation function
- Generators' output layer uses Tanh

Equilibria and GANs

Most ML models are based on optimization and follow the general pattern of

1. Determine model parameters
2. Have a cost function of these parameters
3. Minimize the cost

GANs are different because there are two players, the generator and the discriminator, and each player has its own cost. The "game" is therefore defined by a value function.

- The **generator wants to minimize** the value function.
- The **discriminator wants to maximize** the value function.
- The **saddle point** is when equilibrium is reached, a point in the parameters of both players that is simultaneously a local minimum for each player's costs with respect to that player's parameters.
  A key learning problem for GANs is finding the equilibrium of a game involving cost functions that are:
- High dimensional
- Continuous
- Non-convex
- The equilibrium point is where non of the players can improve outcome by changing their strategy while the other player doesn't change their strategy
- So usually, we can't find the equilibrium but we may get close to it

A common error

- A common error is that people forget to use a numerically stable version of cross-entropy, where the loss is computed using the logits.
  - Logits – the values produced by the discriminator right before the sigmoid.

Label Smoothing

- For the real images, we want D(real_images) = 1. That is, we want the discriminator to classify the the real images with a label = 1, indicating that these are real.
- To help the discriminator generalize better, the labels are reduced a bit from 1.0 to 0.9. For this, we'll use the parameter smooth; if True, then we should smooth our labels by a factor of 0.9 .
- From original paper: "We propose a mechanism for encouraging the model to be less confident. While this may not be desired if the goal is to maximize the log-likelihood of training labels, it does regularize the model and makes it more adaptable"

GAN variants:

GANs	Architecture		Loss Function	Normalization	Optimizer	Weight Init	Label smoothing	G:D Update Frequency	lr
Vanilla GAN	fully connected D and G		1, 5	BN	Adam	None	Yes	1:1	
DCGAN	deep convolution GAN, no fc, no pooling		1-5	BN	Adam	Mean=0, Std=0.02	Yes	1:1	
WGAN	Same as DCGAN		3	BN	RSMprop	Mean=0, Std=0.02	No	1:c c=5	0.00005 Batchsize=64 Weight clipping=0.01
WGAN-GP	Same as WGAN		4	Instance Normalization	Adam				
ProGAN	deep convolution GAN, no fc, no pooling, resolution increased progressively		4, 2	PixelWise	Adam				
CycleGAN	DCGAN encoder decoder style (domain translation)								
Pix2PixGAN	Domain translation								
StarGAN	Domain translation								

Loss Functions for GANs: actually these loss functions work on a variety of architectures

#	Loss Function	Optimization	G:D Update Frequency	New GAN name
1	BCEWithLogits	Adam	1:1	
2	LSE (or MSE)	Adam	1:1	LSGAN
3	Wasserstein Distance with weight clipping	RMSProp	1:c	WGAN
4	Wasserstein Distance with gradient penalty	Adam	1:c, lambda=10	WGAN-GP
5	Wasserstein Distance with noise gradient penalty	Adam	1:c	DRAGAN (Deep Regret Analytic GAN)
6				

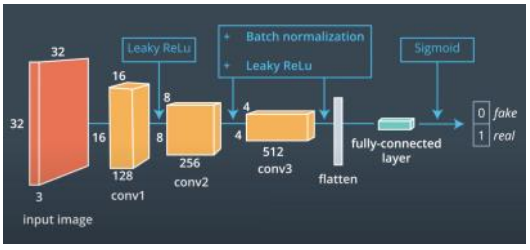
Survey paper about different GANs until 2021
https://arxiv.org/pdf/2203.11242.pdf

For some reason tensorboard real only has 16 images
I was trying to implement the original WGAN

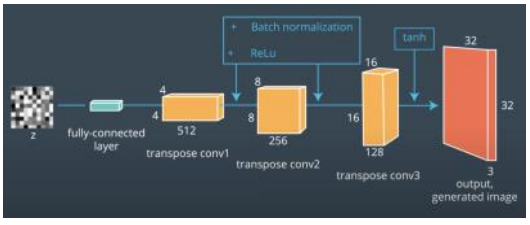
Deep Convolutional GANs (DCGANs)

2023年10月11日 17:23

Discriminator:



Generator:

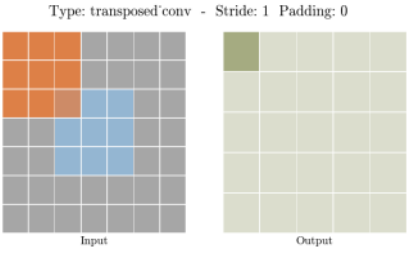


- GAN is usually symmetrical (about convolutional layers) to keep generator and discriminator balanced in skill
https://datascience.stackexchange.com/questions/45039/why-do-most-gan-generative-adversarial-network-implementations-have-symmetric
- Original paper doesn't use maxpool to downsample, rather, it uses strides
- Convolution visualizer:
  - https://ezyang.github.io/convolution-visualizer/

Transpose convolution:

- https://towardsdatascience.com/what-is-transposed-convolutional-layer-40e5e6e31c11
- The idea behind transposed convolution is to carry out trainable upsampling
- Transposed convolutions are standard convolutions but with a modified input feature map.
- The stride and padding do not correspond to
  - the number of zeros added around the image
  - the amount of shift in the kernel when sliding it across the input,

  as they would in a standard convolution operation.



GAN Optimization

2023年10月17日 16:56

Learning Rate and Optimization

- GAN is very sensitive to learning rate and optimization.
- Have a slightly higher learning rate for the discriminator.

Average weights of discriminator and generator:

- Averaging the weights of discriminator and generator during the training is useful
  - Use exponential moving average

Two Times Update Rule [TTUR]:

- This approach consists in running more update steps for the discriminator than for the generator.
  - For example, for each update of the generator, we run 3 updates of the discriminator.

Discriminator Overconfidence:

- Use regularizations on the Discriminator helps the training (to reach a good equilibrium)
  - Dropouts
  - One-sided label smoothing (only on real images)
    - https://github.com/soumith/ganhacks/issues/10
  - Noisy Label:
    - Randomly Flip the labels for real and fake images in the discriminator
- From Chatgpt:
  Label Smoothing: In the context of GANs, label smoothing involves assigning a range of values to the target labels instead of using binary values (0 or 1). Instead of assigning a target label of 1 to real samples and 0 to fake samples, label smoothing assigns values slightly less than 1 to real samples and slightly more than 0 to fake samples. For example, instead of using 1 for real samples, you might use a value like 0.9. This helps prevent the discriminator from becoming too confident and overfitting to the training data.

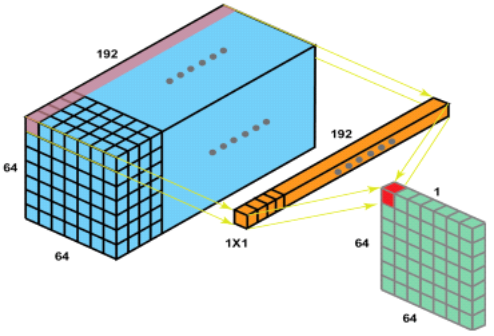
  Noisy Labels: Noisy labels involve intentionally introducing errors or noise into the target labels during training. This means flipping the labels for real and fake samples during the discriminator update step. For example, instead of using 1 for real samples and 0 for fake samples, you might randomly flip the labels, using 0 for real samples and 1 for fake samples. This can help prevent the discriminator from becoming too confident and improve the overall training stability of the GAN.

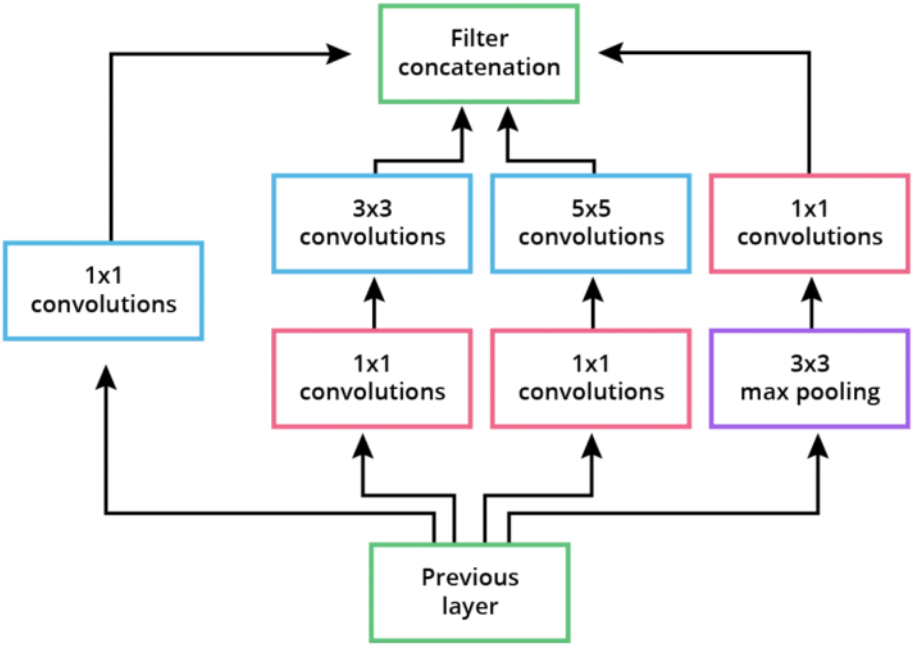
  Both label smoothing and noisy labels are regularization techniques that can be used to prevent the discriminator from becoming too powerful and dominating the training process. They introduce a level of uncertainty and prevent the discriminator from easily distinguishing between real and fake samples. This encourages the

generator to improve its performance and generate more realistic samples.

Inception Model Architecture

2023年10月20日　　10:38

- The Inception Model is a concatenation of the outputs of layers of different filter sizes that allows deeper networks.
  - https://youtu.be/VxhSouuSZDY
- Inception V2, V3, and important notes to scale up convolution network
  - Rethinking the Inception Architecture for Computer Vision
    https://arxiv.org/abs/1512.00567v3
  - 

  - 

GAN Evaluation

2023年10月19日 10:10

- The Inception Score and the Frechet Inception use the Inception Model for their calculations.
  -

The Inception Score:

- The Inception Score leverages the KL divergence and the inception model to evaluate generated samples against the dataset used to train the inception model.
- To calculate the inception score, build two probability distributions:
  1. **Conditional Distribution** – feed a generated sample through the inception model pre-trained on the ImageNet dataset. The inception model will output a probability distribution with the last soft-max layer.
  2. **Marginal Distribution** – the mean of all the p(y|x) over all of the x values.

- Use KL Divergence to measure the distance between the two distributions.
- Detailed easy to understand article about inception score:
  - https://medium.com/octavian-ai/a-simple-explanation-of-the-inception-score-372dff6a8c7a

Kullback Leibler (KL) Divergence

- The KL divergence is a measure of distance between two probability distributions.
  - **Low KL divergence** means that the distributions are similar
  - **High KL divergence** means that they are different

-

$$D_{KL}(P||Q) = \sum_{x \in X} P(x) \log(\frac{P(x)}{Q(x)})$$

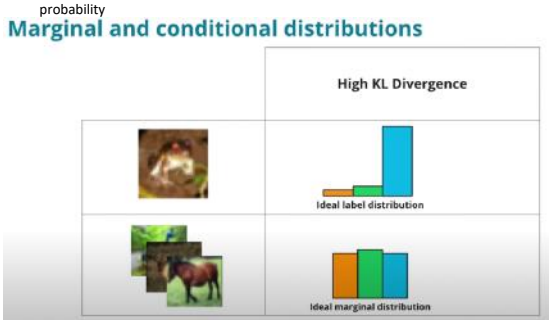
Inception Score Limitations

- The inception score is a great tool to calculate a GAN performance but has some limitations:
  - It relies on a model pre-trained on the ImageNet dataset.
  - It does not take into account the real dataset, which can be limiting in some situatior

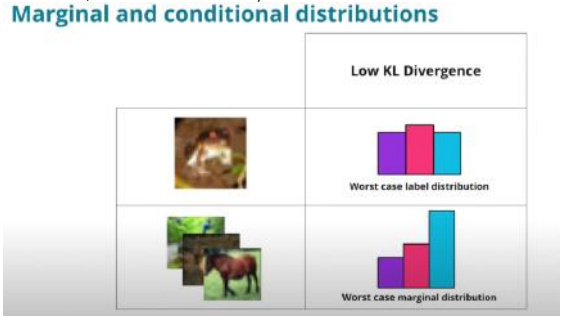
Ideal and Worst cases of KL divergence

- The ideal case for the KL divergence:
  - High label distribution means the inception model mixes the generated samples with it is trained on with high confidence
  - Uniform marginal distribution means the GAN model generates from diverse labels v probability

-



-
- Not confident, and does not show diversity
-



Frechet Inception Distance (FID):

- Pass both the real images and the generated images through the Inception model to obtain their respective feature representations.
- Calculate the mean and covariance of the feature representations for both the real and generated images.
- Use these mean and covariance values to calculate the FID score.
-

FID vs IS

Frechet Inception Distance (FID)	Inception Score (IS)
Uses the inception model to get a latent representation of images	Uses the inception model to get a probability distribution over the labels

FID vs IS

Frechet Inception Distance (FID)	Inception Score (IS)
Uses the inception model to get a latent representation of images	Uses the inception model to get a probability distribution over the labels
Requires to calculate statistics over the real dataseT	Does not require the real dataset
Calculates a distance using mean and covariance	Calculates a score using the KL divergence

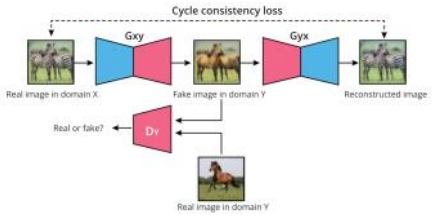
Breaking Models with GAN

2023年10月24日　　14:44

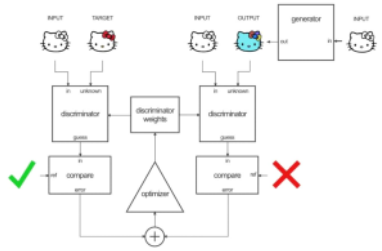
http://karpathy.github.io/2015/03/30/breaking-convnets/

CycleGAN

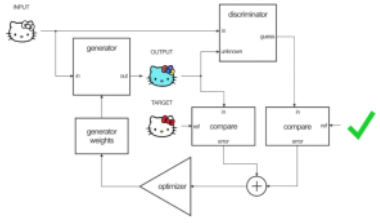
2023年10月24日　17:10



Pix2Pix
Interactive: https://affinelayer.com/pixsrv/
Article: https://neurohive.io/en/popular-networks/pix2pix-image-to-image-translation/



And from generator perspective:



Pix2Pix problem:

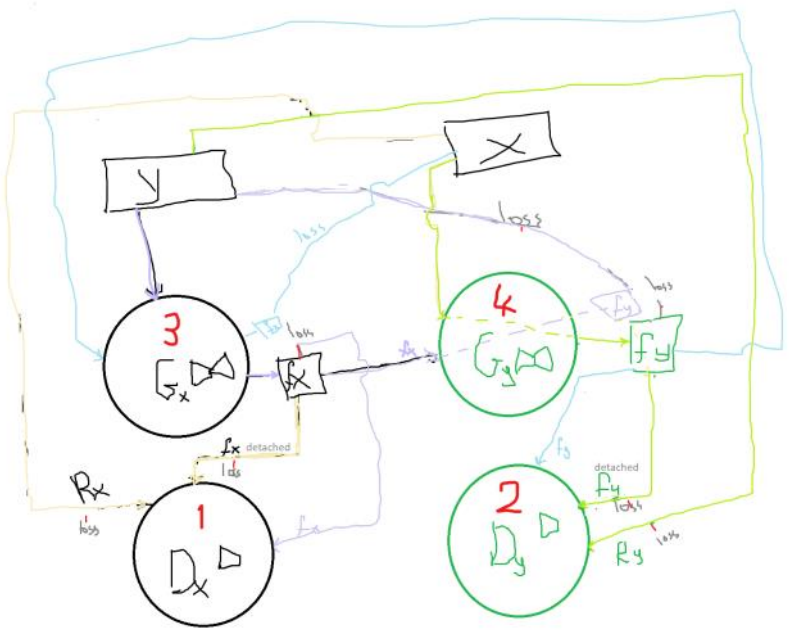
- It needs a pair of input-output image.
- The output (ground truth) is necessary, otherwise, any realistic output image will be considered as real despite being so different from the input image
- CycleGAN solves the problem by creating a cycle in the generator.
- The generator needs to take the input image and generate a realistic output, then use that output to generate the an image similar to the input image.
    - In this way, we force the GAN to produce images within the same domain.

- To transfer a photo into the style of a **particular painting**, you can use a model called Neural Style Transfer.
- To transfer a photo into the style of a domain (not a specific painting), you can use a model called CycleGAN.

Loss Function: Least Squares GANs

- We've seen that regular GANs treat the discriminator as a classifier with the sigmoid cross entropy loss function.
    - However, this loss function may lead to the vanishing gradients problem during the learning process.
- To overcome such a problem, we'll use a least squares loss function for the discriminator.
- This structure is also referred to as a least squares GAN or LSGAN, and you can read the original paper on LSGANs, here.
- The authors show that LSGANs are able to generate higher quality images than regular GANs and that this loss type is a bit more stable during training!
-

How training performed in CycleGAN:



WGAN, WGAN-GP, and DRAGAN: Loss functions

2023年10月31日 14:45

https://github.com/LynnHo/DCGAN-LSGAN-WGAN-GP-DRAGAN-Pytorch

Wasserstein GAN architecture and Wasserstein GAN architecture with gradient penalty

To prevent mode collapse and vanishing gradient there is another loss function to train GANs:

- **The Earth Mover Distance or Wasserstein Metric** also referred to as **Wasserstein Loss** and **Wasserstein Distance**
  The Wasserstein Loss is mathematically represented as follows: $E[C(x)]-E[C(G(z))]$
- Similar to the BCE Loss, note that the logs have disappeared. Indeed the Wasserstein distance gets rid of the log function and only considers the probabilities.
- With the Wasserstein distance the discriminator is called Critic.
- Critic measures distance between the two distributions.
- Critic tries to maximize the expression E[C(x)]−E[C(G(z))] while the generator minimizes E[C(G(z))]

Wasserstein GAN needs to enforce 1-Lipschitz continuity over the critic

1. **Weight Clipping (normal WGAN):**
   - The weights of Critic must be bounded by a box [-0.01, 0.01] using weight clipping.
   - However, the authors say that weight clipping is not an efficient way to enforce 1-Lipschitz because:
     - If clipping is large, it takes a long time for weights to reach their limit --> hard to train
     - If clipping is small, it can easily lead to vanishing gradients when number of layers are big or bn is not used such as in RNNs
2. **Gradient Penalty (GP) (WGAN-GP): All experiments in this paper use λ = 10,**
   - We add a penalty to the original equation.

$$\underbrace{E[C(X)] - E[C(G(z))]}_{\text{Wasserstein Loss}} + \underbrace{\lambda E[(\|\nabla_{\hat{x}} C(\hat{x})\|_2 - 1)^2]}_{\text{Gradient penalty}}$$

λ: penalty coefficient

$\hat{x}$: uniformly sampled data point between the real and fake distributions

   - X_hat is a generated sample that falls between the current fake and real data points.
     - Randomly sample a coefficient α of between 0 and 1
     - Calculate the interpolation as:

$$\alpha x + (1-\alpha)G(z)$$

   - 

   - X is sample of real distribution. G(z) is sample from fake distribution
   - When alpha is close to 0, result is more noisy.
   - When alpha is close to 1, result is more real
3. **DRAGAN gradient penalty (DRAGAN)**
   - The DRAGAN paper offered a different approach to calculate the gradient penalty and enforce the 1-Lipschitz constraint on the critic.
   - $\lambda \cdot \mathbb{E}_{x \sim P_{real}, \delta \sim N_d(0,cI)} \left[\|\nabla_{\mathbf{x}} D_\theta(x+\delta)\| - k \right]^2$
   - Instead of fake image, the DRAGAN calculates a noisy image as $Xp=X+\delta$ as follows:
     - $Xp=X+0.5*\sigma(X)*N$
   - where σ is the standard deviation and N a noise term sampled from the uniform distribution.
   - Then sample a random point (X_hat) between real distribution X and Xp
   - After that just like GP, runs the sample through critic and calculate the gradient
   - then, measures the L2Norm of the gradient
   - Finally, subtracts k (usually 1) from it, square it, and take its mean.
   - **Interestingly, using this gradient penalty lifts some of the constraint on the BCE Loss and the authors use the above gradient penalty with the vanilla GAN losses (BCE Loss).**

There is also LSGAN where we use least square

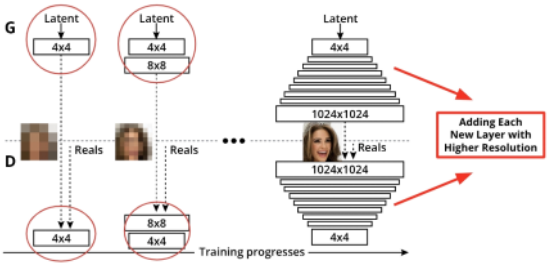
Important Notes:

- BN disrupts the effect of GP in critic model and weakens it.
- Quote from original WGAN-GP paper:
  - **No critic batch normalization** Most prior GAN implementations [22, 23, 2] use batch normalization in both the generator and the discriminator to help stabilize training,
  - but batch normalization changes the form of the discriminator's problem from mapping a single input to a single output to mapping from an entire batch of inputs to a batch of outputs [23].
  - Our penalized training objective is no longer valid in this setting, since we penalize the norm of the critic's gradient with respect to each input independently, and not the entire batch.
  - To resolve this, we simply omit batch normalization in the critic in our models, finding that they perform well without it.
  - Our method works with normalization schemes which don't introduce correlations between examples.
  - In particular, we recommend layer normalization [3] as a drop-in replacement for batch normalization.

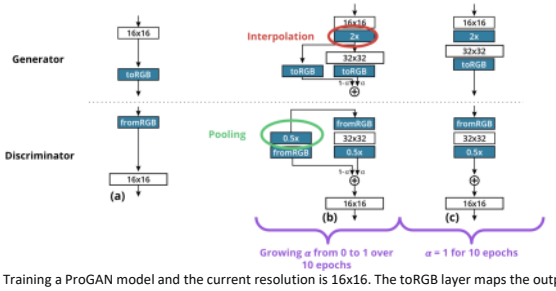
- Solutions:
  - Remove BN from the critic/discriminator
  - Use other types of normalization that does not create correlations between examples in the batch
    - Such as: pixelwise normalization and minibatch standard deviation

ProGAN

2023年11月1日 16:47

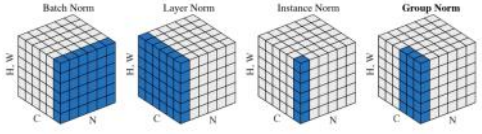
- The aim of this architecture is to train GAN progressively.
- First, we train GAN on a low resolution image say 4x4
- After a few epochs, we increase the resolution by adding another layer to the models making it 8x8
- We continue adding layers until we reach desired resolution say 1024x1024.
- 

- But, we don't just add the new layer, rather, we need to fade it in.
  - That means we add an interpolation of it then slightly make it the real higher resolution image.



1. Training a ProGAN model and the current resolution is 16x16. The toRGB layer maps the output of the last convolution to an RGB image and the from RGB layer takes a RGB image as input and feeds it into the next convolution layer.
2. To increase the resolution to 32x32 use **layer fading**. Add a new layer to the generator and the discriminator by doubling the resolution of the 16x16 layer using an **interpolation** method such as nearest neighbor.
   - In the generator, fuse the output of the 32x32 layer with the interpolated output.
   - In the discriminator, fuse the output of the 32x32 layer with a downsampled image and use a pooling method such as average pooling for downsampling.
3. For both cases, perform a weighted sum of the learned output of the new layer with the non parametric output of the previous layer. Slowly increase the weight of the output of the new layer over 10 epochs to reach a stage where a fade is not needed in that layer.
4. Train the network at the 32x32 resolution for another 10 epochs

ProGAN Tricks

- **Progressive Growing** – Progressively train layers and increase resolution
- **Minibatch Discrimination** – Enforce fake and real batches to have similar statistics
  - **Minibatch standard deviation layer:** A parameter-free alternative to minibatch discrimination layer.
    - appends the std-dev of the minibatch to the input as an additional constant feature map in the discriminator
- **Equalized Learning Rates** – Scale the weights of each layer by a different constant to make sure the layers are learning at the same speed
- **Pixelwise Normalization** – Normalize each pixel of a feature map along the channel axis
- 

With pixel normalization, we normalize each pixel of the input volume as follow:

y = x.pow(2.0).mean(dim=1, keepdim=True).add(alpha).sqrt()
x = x / y
where **x** is the input volume of dimensions (NCHW). We square the input volume, calculate the mean over the channel dimension, add a very small factor alpha and calculate the square root.

From <https://learn.udacity.com/nanodegrees/nd101/parts/cd1823/lessons/a1c7e7b0-aa17-4fc4-886d-e3cd167dd6f2/concepts/065de438-4c23-42e7-9614-bebdfb459b4b>

```
def minibatch_stddev_layer(x, group_size=4):
    with tf.variable_scope('MinibatchStddev'):
        group_size = tf.minimum(group_size, tf.shape(x)[0])     # Minibatch must be divisible by (or smaller than) group_size.
        s = x.shape                                             # [NCHW]  Input shape.
        y = tf.reshape(x, [group_size, -1, s[1], s[2], s[3]])   # [GMCHW] Split minibatch into M groups of size G.
        y = tf.cast(y, tf.float32)                              # [GMCHW] Cast to FP32.
        y -= tf.reduce_mean(y, axis=0, keepdims=True)           # [GMCHW] Subtract mean over group.
        y = tf.reduce_mean(tf.square(y), axis=0)                # [MCHW]  Calc variance over group.
        y = tf.sqrt(y + 1e-8)                                   # [MCHW]  Calc stddev over group.
        y = tf.reduce_mean(y, axis=[1,2,3], keepdims=True)      # [M111]  Take average over fmaps and pixels.
        y = tf.cast(y, x.dtype)                                 # [M111]  Cast back to original data type.
        y = tf.tile(y, [group_size, 1, s[2], s[3]])             # [N1HW]  Replicate over group and pixels.
        return tf.concat([x, y], axis=1)
```

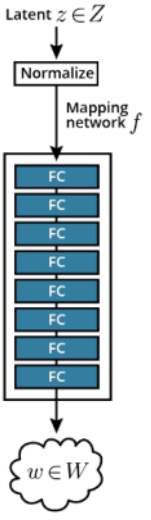
StyleGAN

2023年11月7日　　17:16

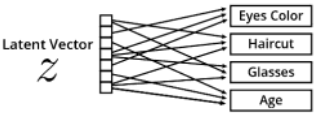
Controllable Generation

Conditional Generation indicates that the training set must be labeled and conditioning is limited to examples from the training set.

- **Conditional Generation** – each image is conditioned with a label (e.g. MNIST dataset).
  For example, fake 11s can not be generated with a conditional GAN trained on the MNIST dataset because the data set only includes digits in the range of 0 to 9.
  Conditional Generative Adversarial Nets [1] introduced the **conditional GAN**. An example of conditional implementation in PyTorch here. Conditional GANs have an extra input from the discriminator and generator networks.
  For **controllable generation,** instead of inputting a label to condition the output, the latent vector z is modified to control the aspect of the output. This makes the assumption that the components of z each control a different aspect of the output image.
- **Controllable Generation** – does not require labels.
  The Mapping Network
  The **mapping network** is a new component of the StyleGAN generator. A mapping network:
- Takes the latent vector z as input
- Outputs a new latent vector w
- Helps to disentangle the latent vector z for controllable generation.

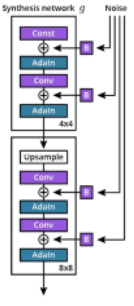


- The Entanglement Problem
- When modifying some components, we impact more than one feature. This is the **entanglement problem**.
- For example in trying to generate faces, features could include:
  - Haircut
  - Eye color
  - Glasses
  - Age



- **Noise Injection**
- Another new component of StyleGAN is the injection of noise at different locations in the generator.
  This **noise injection** will:
- Help with stochastic variation! Injecting noise in the network will help create more diverse features.
- Happen at different locations in the network and impacts the variability of the images at different levels.
  To add noise:

1. A random feature map is sampled from a gaussian distribution
2. The map is multiplied by a learned scaling factor
3. This noise is applied to the output of the convolutional layers



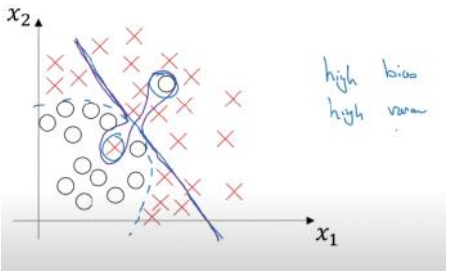
Course 2 of the Deep Learning Specialization

2023年7月4日　　9:28

Improving Deep Neural Networks: Hyperparameter Tuning, Regularization and Optimization

Bias and Variance

2023年7月4日 9:28

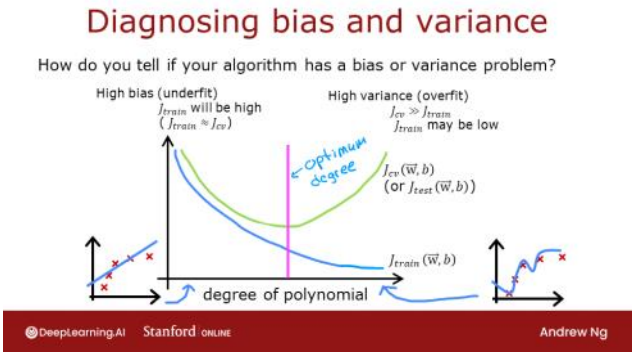


High bias:: Underfitting
- Performs poor on training data and similarly on validation data

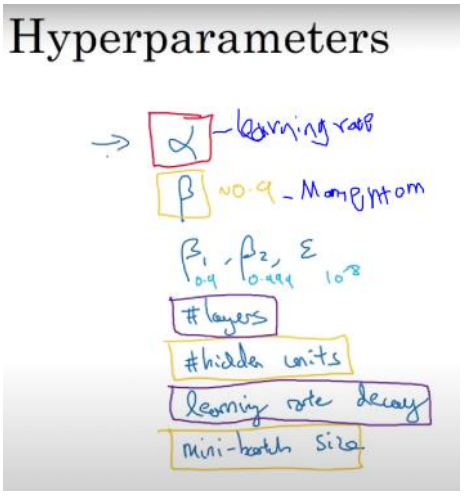
High variance: overfitting
- Performs very well on training data and pretty poor on validation data

High bias and high variance: underfit most and overfit some outliers
- Performs poor on training data and much poorer on validation datax

Problem	Solutions		
High bias	Bigger Network	Train longer	Change NN architecture
High Variance	More data	Regularization	Change NN architecture
Vanishing/Exploding gradients (happens when network is too deep)	Random weight init		

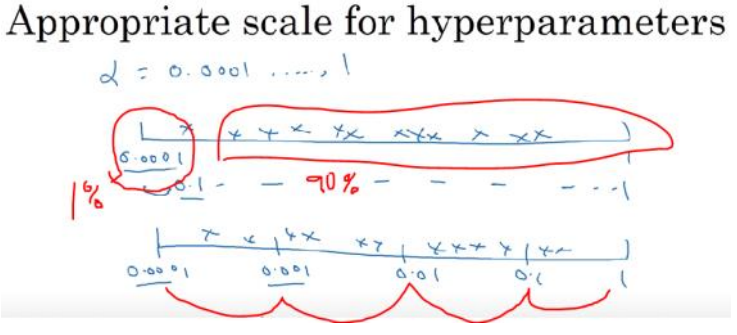


Regularization:
- L2 Norm
- Dropout: inverted dropout
- Data Augmentation:
  - Flip, crop, random distortion
- Early Stopping



Normalization will speedup the model training by allowing the cost function to find the optimal solution faster

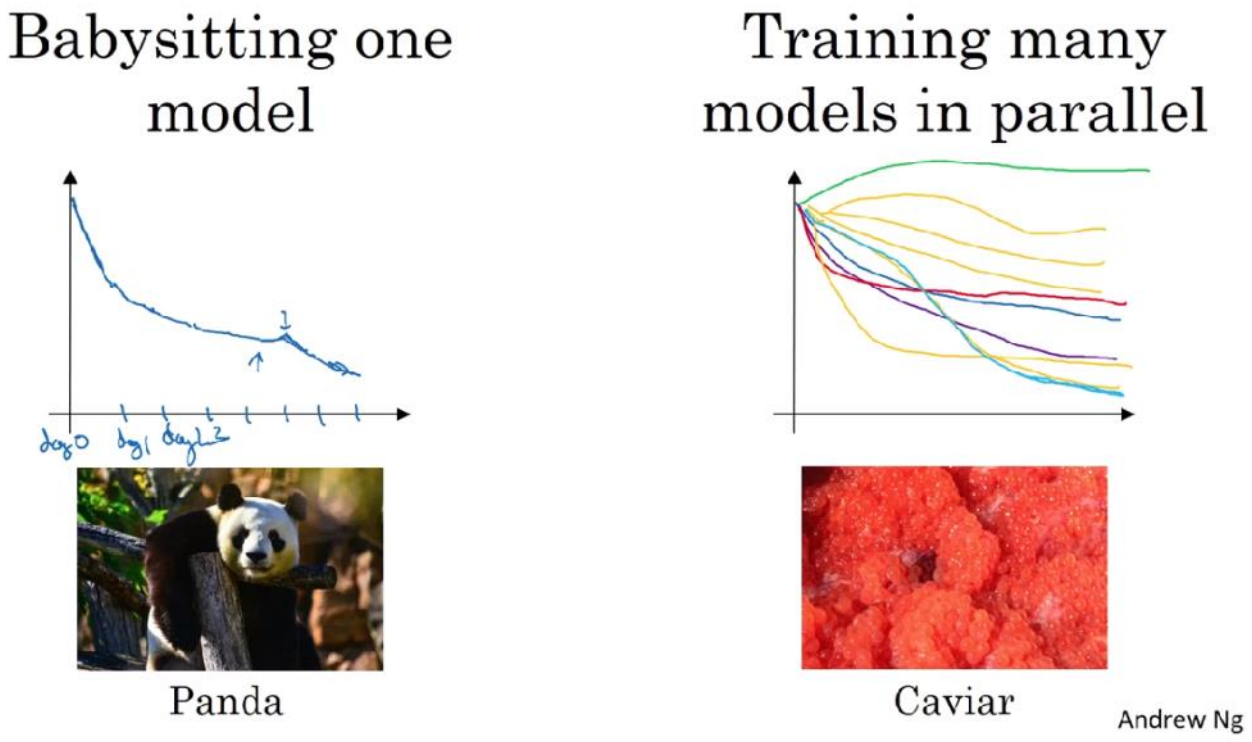
Gradient Checking
- To make sure that back propagation works correctly
- Only used to debug not for training that means don't do it in every iteration
- Change all weight matrix into one giant vector theta
- Use some equation of derivatives to get a value
- The value of 10^{-7} means good
- The value of 10^{-5} means you may need some checks
- The value of 10^{-2} means there is a high chance that there is something wrong with your network
  - Look at the components to identify the bug
- Doesn't work with dropouts



- Searching hyperparameters at random is one good way to detect a good hyperparameter
- Sometimes the search needs to be logged then try to uniformly pick values
- Use coarse to fine search

Hyperparameter Tuning

2023年7月4日 11:06



Babysitting one model

- Create one model and improve it by changing the hyperparameters everyday or so and watch how it learns
- When one day the learning rate worsened, you go back to previous days' parameters and start there again

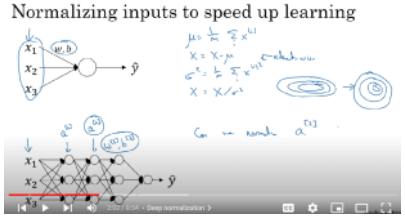
Training many models in parallel

- Create many models with different parameters in parallel
- Let all of them run and watch the learning rate
- This method is resource hungry

Batch normalization

2023年7月4日　11:10

- Batch Normalization
  It's called "batch" normalization because, during training, we normalize each layer's inputs by using the mean and standard deviation (or variance) of the values in the current batch. These are sometimes called the batch statistics.
- The questions is, can we normalize the values in the hidden layers?
- Makes hyperparameter search much more easier.
- Trains much more efficiently
- It usually normalizes the value of Z instead of a
  - Z is the value before activation function
  - A is the value after activation function
- It also results in a slight regularization
  - Increasing the batchsize will decrease this effect
  - Consider this as unintended side-effect
  - Use actual regularization methods to do regularization
- During the test time batch norm causes incorrectness in the result because the test is not batches rather it is a unit sample which leads to incorrect mu and variance
  - We test one sample by one sample
  - The mean and variance for the test can be approximated using moving average on the training mini batch data
    - In this way, you have a quite good estimate on the actual mean and variance for the test data
    - Tensorflow already considers this situation and automatically calculates the means and variances during training to be used during test
    - I'm not sure about pytorch



Benefits of Batch Normalization

From <https://learn.udacity.com/nanodegrees/nd101/parts/cd1823/lessons/d6494283-62b5-4fba-a7eb-e669965af2a7/concepts/a6257510-0e7b-4767-b5e5-978734a2328d>

- You can make a powerpoint for AI benkyo about BatchNorm fusion.
- Start by describing batch norm and its benefits then move to the fusion technique.

why BatchNorm needs to set bias=False

- BatchNorm will center the data and including a bias in previous layer will be cancelled here. So first few iterations will go on to cancel this bias and slow down the training.
- The above statement is only true if we do Linear/Conv -> BatchNorm -> Activation function.
- It is best practice to structure your network with Linear (or Conv) layer -> BatchNorm -> ActivationFunction. The trainable offset and scaling in the BatchNorm the data can hit the non-linearity however it is best to do so. Since there is no operation in between the linear layer and the BatchNorm, and BatchNorm automatically subtracts the mean of the input and re-offsets the data with its own bias vector, the bias term in the Linear layer is useless. It will just get undone by BatchNorm (whether the BatchNorm has an affine transformation or not). But it will slow down training as these two "gears in the machine" grind against each other.
- If you are doing Linear (or Conv) layer -> ActivationFunction -> BatchNorm (not recommended), the bias vector in the linear layer will be doing something because it will change how the γ vector hits the non-linearity, and it is important to still include the bias vector in this situation.
- However way you're structuring the bulk of the network, if your network ends with a Linear (or Conv) layer, or Linear into Softmax, you need to make sure that final Linear that has a bias term

Batch Norm Folding: An easy way to improve your network speed

- https://pytorch.org/tutorials/intermediate/fx_conv_bn_fuser.html
- BatchNorm fusion during training:
  - https://pytorch.org/tutorials/intermediate/custom_function_conv_bn_tutorial.html
- Fusing adjacent convolution and batch norm layers together is typically an inference-time optimization to improve run-time.

From <https://scortex.io/batch-norm-folding-an-easy-way-to-improve-your-network-speed/#:%7E:text=Folding%20in%20Convolution%20layer,one%20convolution%20with%20different%20weights.>

From ChatGPT:

"Fusing adjacent convolution and batch norm layers together is typically an inference-time optimization to improve run-time. It is usually achieved by eliminating the batch norm layer entirely and updating the weight and bias of the preceding convolution [0]. However, this technique is not applicable for training models."

Here's an explanation of the text:

1. **Fusing Convolution and Batch Norm Layers:** In many neural network architectures, convolutional layers are often followed by batch normalization layers. The purpose of batch normalization is to normalize the activations of the convolutional layer, which can help with training and improve the generalization of the model. However, during inference (when you use the model to make predictions), this additional batch normalization step can introduce some computational overhead.
2. **Inference-Time Optimization:** The text suggests that fusing convolution and batch normalization layers is an optimization technique applied during inference. The goal is to make the inference process more efficient and faster.
3. **Eliminating Batch Norm for Inference:** To achieve this optimization, the batch normalization layer is removed entirely during inference. Instead of using batch normalization, the text proposes that the weights and biases of the preceding convolutional layer are updated or adjusted to compensate for the absence of batch normalization.
4. **Not Applicable for Training Models:** The last sentence makes it clear that this fusion technique is not suitable or applicable during the training of models. During training, batch normalization is typically essential for stabilizing and accelerating the training process. It helps with gradient flow and makes it easier for the model to learn the underlying patterns in the data. Therefore, this fusion approach is reserved for optimizing the model's performance during inference while maintaining the batch normalization benefits during the training phase.

Learning Progress

2023年6月30日 17:28

So far

- Statistical methods
  - Linear regression
    - Polynomial regression can fit a curve
    - You can change linear to polynomial manually by squaring the feature
      - $w_1 x_1 + w_2 x^2_1$
  - Logistic regression
    - Same as in linear regression, by adding extra features you can fit an ellipse of other different shapes to classify the data
  - Time series analysis
    - ARIMA based models (prediction)
    - Facebook Prophet Library
- Machine learning methods
  - Classification
    - Logistic regression
    - SVM Support vector machine for classification
  - Clustering
    - Kmeans
    - Random Forest (a bit)
  - Prediction
    - Linear regression
  - Deep Learning
    - NN
      - Linear + activation functions
      - Classification (sigmoid/softmax)
        - Image data and numeric data
    - RNN
      - LSTM
        - Time series analysis (prediction)
    - CNN