Lecture Notes on CS231N

Yiwei Chen

Zhejiang University

Starting from March 26, 2025

Last updated: April 22, 2025

Contents

Lecture 1: History of CV and Introduction to CNNs

- **ImageNet:** Annual competition for image classification, started in 2010.
- **Convolutional Neural Networks (CNNs):** Introduced by Yann LeCun in 1998, CNNs are a type of neural network designed for processing structured grid data, such as images. CNNs show great performance in image classification tasks.

Lecture 2: Image Classification Pipeline

2.1 Goal and Attempts

1. **Goal:**

   The task in Image Classification is to predict a single label (or a distribution over labels as shown here to indicate our confidence) for a given image. Images are 3-dimensional arrays of integers from 0 to 255, of size Width x Height x 3. The 3 represents the three color channels Red, Green, Blue.

2. **Attempts:**

   - Find edges, then corners: does not work well.
   - Use large datasets with labels.

2.2 Classifiers:

1. **K-Nearest Neighbors (KNN):**

   - *Description:* When $K = 1$, Find the closest image in the dataset to the input image (Nearest Neighbors (NN)).
   - *Distance metric:*

     (a) L1(Mahanttan) Distance: a squared distance metric.

     $$d(x, y) = \sum_i |x_i - y_i| \tag{2.1}$$

     (b) L2(Euclidean) Distance: a squared distance metric.

     $$d(x, y) = \sqrt{\sum_i (x_i - y_i)^2} \tag{2.2}$$

     Rotating the coordinate system changes the L1 distance but not the L2 distance.

   - *Performance:*

     Training time: $O(1)$, as there is nothing to do.

     Prediction time: $O(N)$, which is inefficient.

   - K-Nearest Neighbors (KNN):

     Description:

     When $K = 1$, the classifier is too sensitive to noise.

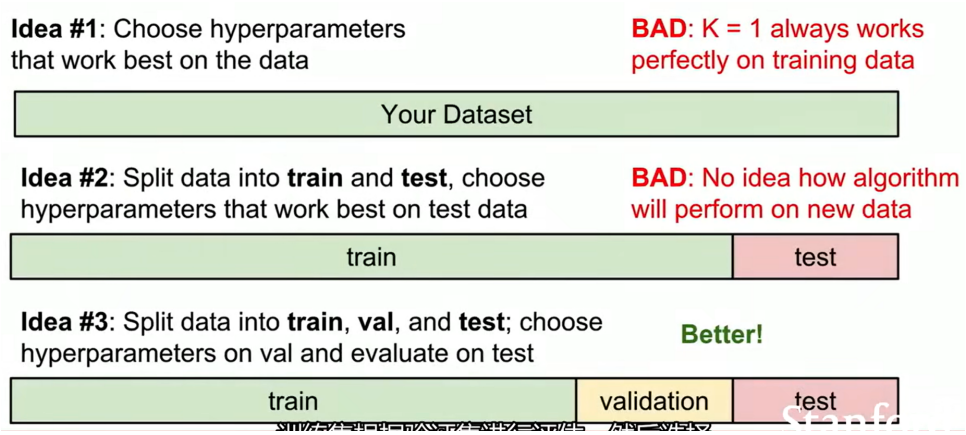
     Instead of copying the label of the closest image, take the majority vote of the K closest images.

   - *hyperparameters*(超参数):

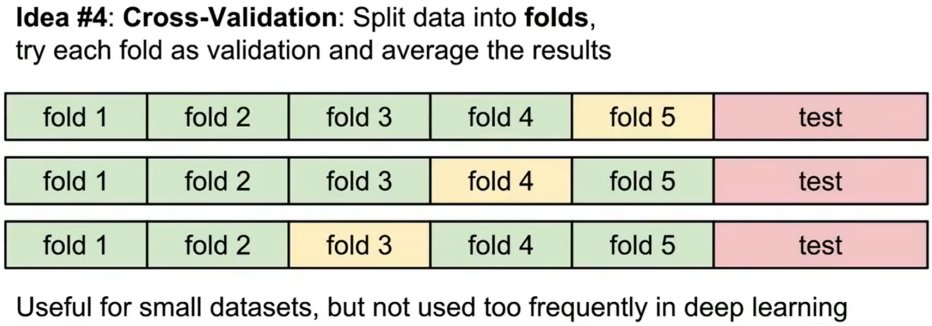
     Choices about the model that are not learned from the data, e.g., K in KNN.

     To set hyperparameters:

     - Never use the test set to set hyperparameters.
     - Splitting data into train and test is not enough.
     - **The better idea:** Splitting the training set into training set, validation set, and test set.



– **The common idea:** Cross-validation(交叉验证).



- *Pros and Cons:*

  Actually, KNN on image is never used:

  - Very slow at test time.

  - Distance-metrics on pixels are not informative.

  - Curse of dimensionality: as the number of dimensions increases, the distance between points becomes less meaningful.

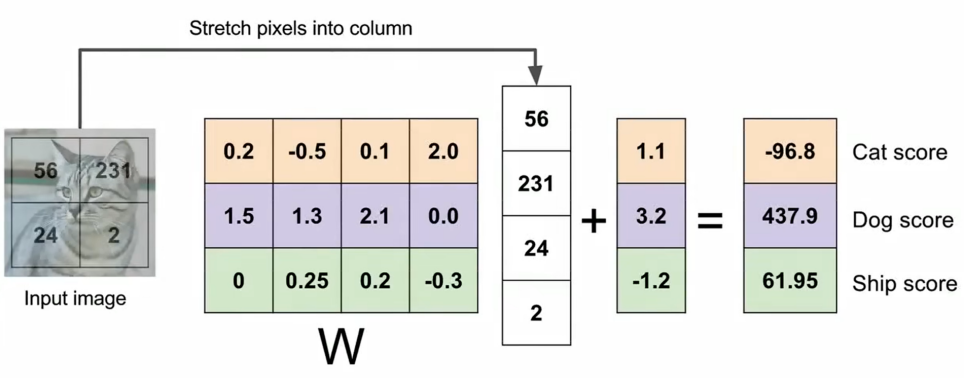
2. **Linear Classifier:**

- *Description:* A linear classifier makes its predictions based on a linear predictor function combining a set of weights with the feature vector.

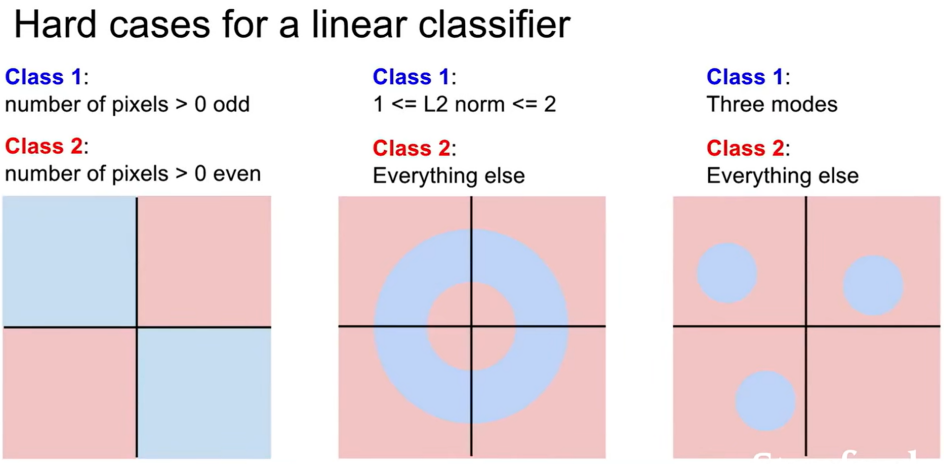
$$f(x, W) = Wx + b \tag{2.3}$$

where x is the input image and w is the weight vector, b is the bias term.

And it is important to center the data before applying the linear classifier, namely image data preprocessing.



- *Hard cases:*


Hard cases for a linear classifier
Class 1:
number of pixels > 0 odd
Class 2:
number of pixels > 0 even
Class 1:
1 <= L2 norm <= 2
Class 2:
Everything else
Class 1:
Three modes
Class 2:
Everything else

Lecture 3: Loss Functions and Optimization

3.1 Loss Functions

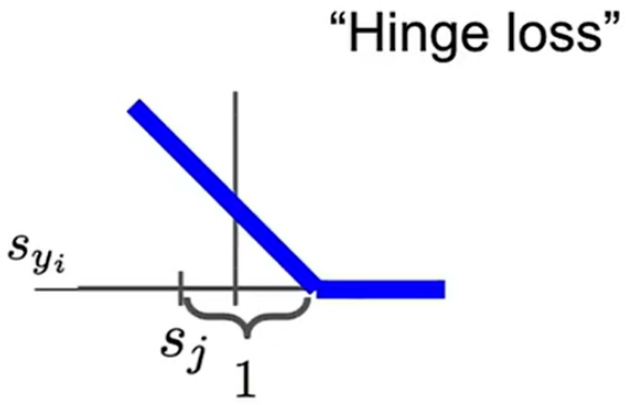
1. **Multiple SVM loss:**

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_{y_i} - s_j + 1 & \text{otherwise} \end{cases}$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) = \sum_{j \neq y_i} \max(0, w_j^T x_j - w_{y_i}^T x_{y_i} + 1) \qquad \text{(SVM-Loss)}$$

where y_i is the label for class i, $s = f(x_i, W)$ is the score for class i, where wj is the j-th row of W **reshaped as a column**.

And this kind of loss function is called **Hinge Loss**.



$$L = \frac{1}{N} \sum_{i=1}^{N} L_i \tag{3.1}$$

For the first training, the W is randomly initialized, so the loss is close to C - 1, where C is the number of classes, which can be used to check the correctness of the implementation.

2. **regularization:**

Furthermore, we can add a regularization term to the loss function to prevent overfitting:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \lambda R(W) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i, W)_j - f(x_i, W)_{y_i} + \Delta) + \lambda R(W) \tag{3.2}$$

Here Δ actually does not matter. It turns out that this hyperparameter can safely be set to $\Delta = 1.0$ in all cases.

The hyperparameters Δ and λ seem like two different hyperparameters, but in fact they both control the same tradeoff: The tradeoff between the data loss and the regularization loss in the objective.

Therefore, the exact value of the margin between the scores is in some sense meaningless because the weights can shrink or stretch the differences arbitrarily. Hence, the only real tradeoff is how large we allow the weights to grow (through the regularization strength λ).

In common use:

- **L2 regularization:** $R(W) = ||W||^2$
- L1 regularization: $R(W) = ||W||_1$
- Elastic net: $R(W) = \beta||W||^2 + ||W||_1$
- Max norm regularization: might see later
- Dropout: might see later
- Fansier: Batch normalization, stochastic depth, etc.

3. **Softmax Classifier(Multinomial Logistic Regression):**

   scores = unormalized log probabilities of the classes.

$$P(y_i|x_i) = \frac{e^{s_{y_i}}}{\sum_{j=1}^{C} e^{s_j}} \quad \text{where } C \text{ is the number of classes} \tag{3.3}$$

$$L_i = -\log P(y_i|x_i) = -\log \frac{e^{s_{y_i}}}{\sum_{j=1}^{C} e^{s_j}} = -s_{y_i} + \log \sum_{j=1}^{C} e^{s_j} \tag{3.4}$$

   For Softmax, even though the right score is much higher than all the other scores, the loss is still not zero, which is different from SVM.

   Here, it's actually a cross-entropy loss function, which is defined as follows:

$$H(p, q) = -\sum_{x} p(x) \log q(x) \qquad \text{(Cross-entropy)}$$

   where p is the "true" distribution and q is the predicted distribution.

   For Softmax, $q = e^{s_j} / \sum_{k=1}^{C} e^{s_k}$, and the "true" distribution is a one-hot vector(p = [0,...,1,...,0] contains a single 1 at the y_i-th position).

3.2 Optimization:

Strategy: follow the slope

So first we need to compute the gradient of the loss function with respect to the weights W.

The method of finite differences($(f(W+h)-f(W))/h$) is bad because of large amount of computation, dont't use it to train but to **check the correctness** of the implementation.

step size/learning rate: a hyperparameter that controls how much to change the model in response to the estimated gradient.

1. **Derivative of the loss function:**

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W) \tag{3.5}$$

   The scores of each class are influenced by the rows of W corresponding to the classes.

Hence, the gradient of the loss function with respect to the weights W should focus on the rows.(assuming W is (C * D))

- **For Hinge Loss:**

$$L_i = \sum_{j \neq y_j} [\max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta)]$$

where w_j is the j-th row of W reshaped as a column.

Taking the gradient with respect w_{y_i}

$$\nabla_{w_{y_i}} Li = -(\sum_{j \neq y_j} 1(w_j^T x_j - w_{y_i}^T x_{y_i} + \Delta > 0))x_i \tag{3.6}$$

where 1 is the indicator function that is one if the condition inside is true or zero otherwise.
For the other rows where $j \neq y_i$ (for every j) the gradient is:

$$\nabla_{w_j} Li = 1(w_j^T x_j - w_{y_i}^T x_{y_i} + \Delta > 0)x_i \tag{3.7}$$

- **For Softmax Loss:**

$$L_i = -\log P(y_i|x_i) = -\log \frac{e^{s_{y_i}}}{\sum_{j=1}^{C} e^{s_j}} = -s_{y_i} + \log \sum_{j=1}^{C} e^{s_j}$$

Taking the gradient with respect w_{y_i}

$$\nabla_{w_{y_i}} Li = -x_i + \frac{e^{s_{y_i}}}{\sum_{j=1}^{C} e^{s_j}} x_i = -x_i + P(y_i|x_i)x_i \tag{3.8}$$

For the other rows where $j \neq y_i$ (for every j) the gradient is:

$$\nabla_{w_j} Li = \frac{e^{s_j}}{\sum_{k=1}^{C} e^{s_k}} x_i = P(j|x_i)x_i \tag{3.9}$$

2. **Stochastic Gradient Descent(on-line gradient descent):**

   Full sum computation is too expensive, so we can use mini-batch(32/64/128) gradient descent.

   Every iteration, we randomly sample a mini-batch of N training examples from the training set, and compute the gradient of the loss function with respect to the weights W using only this mini-batch.

   SGD technically refers to using a single example at a time to evaluate the gradient.

   However, you will hear people use the term SGD even when referring to mini-batch gradient descent (i.e. mentions of MGD for "Minibatch Gradient Descent", or BGD for "Batch gradient descent" are rare to see), where it is usually assumed that mini-batches are used.

   MGD: using a small batch of examples to evaluate the gradient.

   BGD: using the entire training set to evaluate the gradient.

3.3 Image features:

Some times, only using the raw pixel values is not enough, we need to extract some features from the image.

For example:

- color histogram(直方图): a representation of the distribution of colors in an image.
- HOG(Histogram of Oriented Gradients): a feature descriptor focusing on the edges and contours of objects in an image.
- Bag of Words: a representation of an image as a collection of local features, where each feature is represented by a visual word.

Lecture 4: Backpropagation and Neural Networks

4.1 Backpropagation

1. **Computation Graph:**

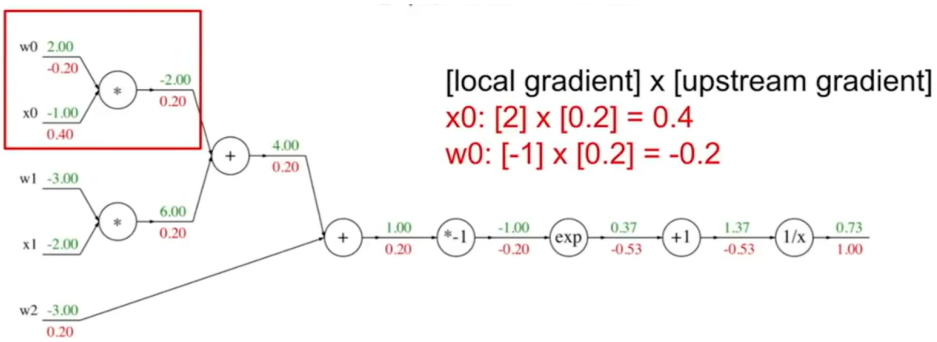


Figure 1: Computation Graph Example: $f(x) = \frac{1}{1+e^{-(\omega_0 x_0 + \omega_1 x_1 + \omega_2)}}$

Along the edges of the graph, we do **forward** computations to get the scores of the classes, and get the output label.

And **backward** computations to get the gradients of the loss function with respect to the weights W and the input x.

2. **Backpropagation**

Sigmoid funtion:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{4.1}$$

It is a common activation function used in neural networks, especially in the output layer for binary classification tasks.

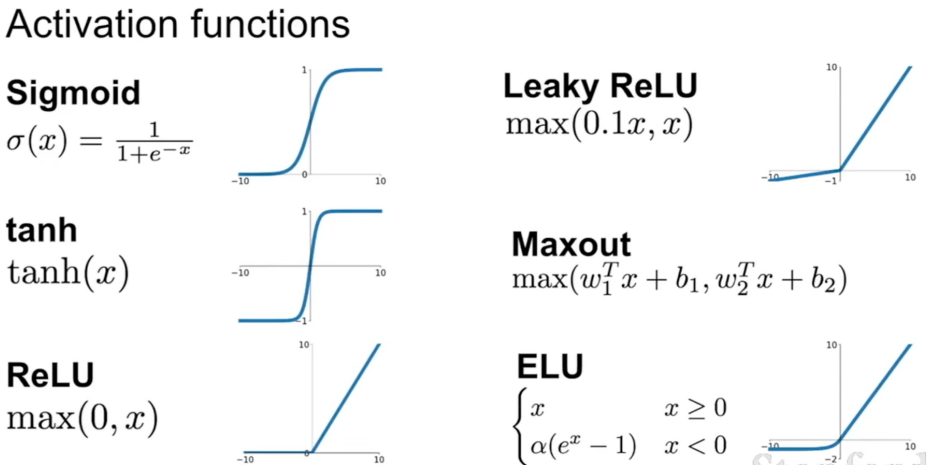


Figure 2: Activation Functions

- **Add gate:** gradient distributor
- **Max gate:** gradient router
  only the maximum value will pass through the gate, and the others will be zero.
- **Mul gate:** gradient switcher

Additionally, if the element is a vector, the gradient will be Jacobian matrix with the same shape as the input.

For f(x, W) = Wx + b(x is a batch), the gradient is:

$$\nabla_x = dout * W^T, \quad \nabla_W = x^T * dout, \quad \nabla_b = \sum dout * 1 \tag{4.2}$$

4.2 Neural Networks:

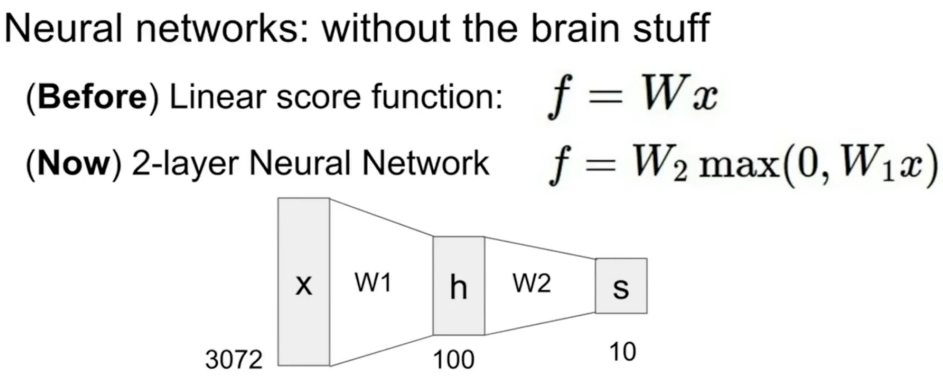


Figure 3: Neural Network Example: $f(x) = W_2 \max(0, W_1 x)$

Here, W_1 contains the weights for the first layer, which can be thought of as computing the scores of the templates.

And W_2 contains the weights for the second layer, which can be thought of as computing the final scores for the classes by taking a weighted sum of the scores from the first layer.

Below are figures showing the structure of a neural network.

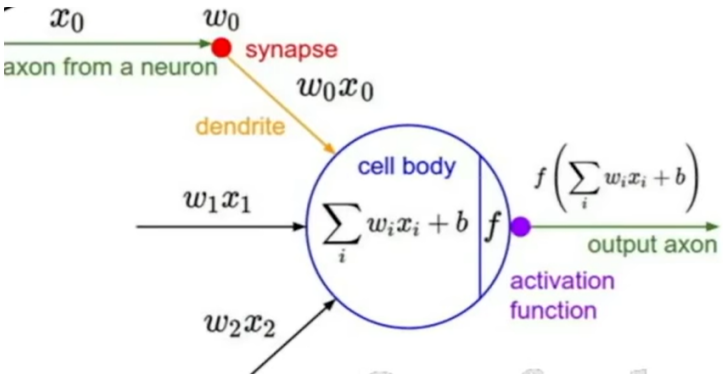


Figure 4: Neuron-like structure

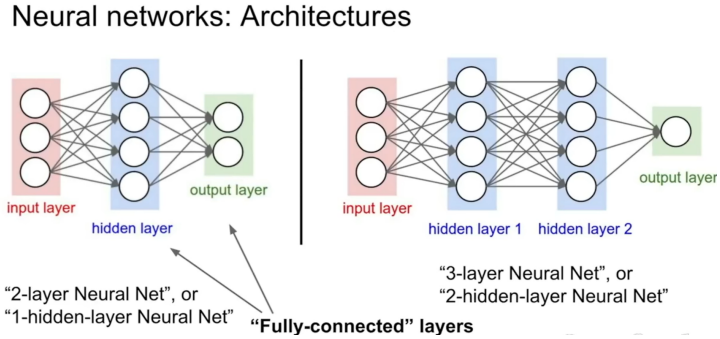


Figure 5: Neural Network Architecture

Lecture 5: Convolutional Neural Networks

5.1 Convolution Layer

Instead of stretching the input image into a long vector, now preserve the spatial structure.

- **Filter:** Filters always extend the full **depth** of the input volume.

  Convolve(slide) over all spatial locations. Each filter is kind of looking for a specific feature in the image, so usually we have many filters in a convolutional layer.

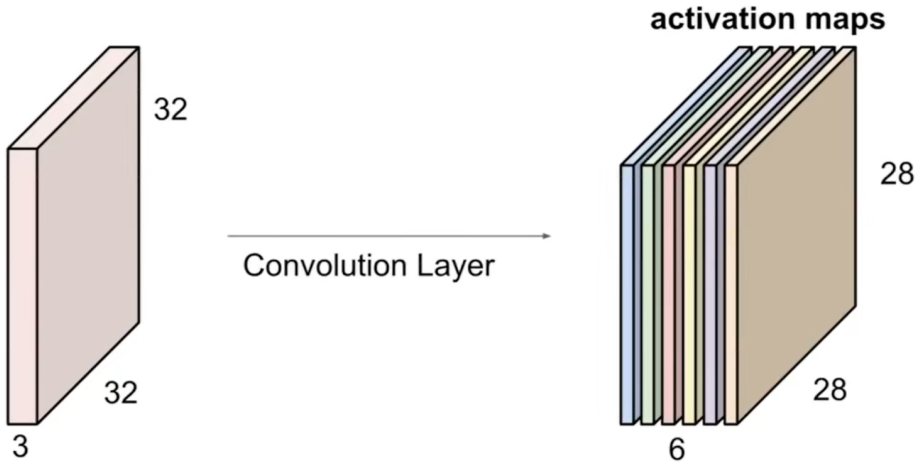


Figure 6: Convolution Operation

- **Convolution:**

  In mathematical terms, the convolution operation is defined as:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n1,n2] * g[x-n1, y-n2] \tag{5.1}$$

  which means that the filter is flipped both horizontally and vertically before being applied to the image.

  What we commonly do is a cross-correlation(互相关) operation, without flipping the filter, but we call it convolution.