11- learning based vision

the turing test = machine's ability to exhibit intelligent behavior indistinguishable from a human

nearest neighbor classifier → terrible performance for images (does not consider shift, illumination, ..)

↳ training = just remember / keep all the training data $O(n)$ → expensive

↳ predicting = smallest sum of absolute (→ or square) value differences between pixels → return (→ or take k ones) its label

* CNN's have expensive training, cheap test evaluation

* do not use test set to determine hyperparameters → use cross validation in train data

challenges in visual recognition = camera pose, illumination, deformation, occlusion, background clutter (dağınık), intraclass variations

$$\underset{\text{weights}}{W} \; . \; \underset{\text{input}}{x} \; = \; y \rightarrow \underset{\substack{\text{class scores} \\ 10\times1}}{f(x,w)} + \underset{10\times1}{b}$$

10×3072 $\quad$ $32\times32\times3 = 3072\times1$

loss functions

↳ multiclass SVM loss = $L_i = \sum_{j\neq y_i} \max(0, s_j - s_{y_i} + 1)$ (→ hinge loss (tries to find max margin)) $\quad s = f(x_i, w) \longrightarrow L = \frac{1}{N}\sum_i L_i$

loss of one image = $\sum_{j\neq y_i} \max(0, \text{other\_class\_scores} - \text{actual\_class\_score} + 1)$

ex: $\begin{array}{l} \\ \begin{bmatrix} 3.2 \\ 5.1 \\ -1.7 \end{bmatrix} \end{array} \begin{array}{l} \text{actual class} \\ \\ \\ \end{array} = \underbrace{\max(0, 5.1 - 3.2 + 1)}_{2.9} + \underbrace{\max(0, -1.7 - 3.2 + 1)}_{0} = 2.9$ (take the average loss for all images)

weight regularization = $L = \frac{1}{N}\sum_i^N L_i + \lambda . R(W) \rightarrow L_1, L_2, L_1 + L_2$ (elastic net), max norm, dropout

↳ lambda = regularization strength (hyperparameter)

↳ softmax classifier = normalize log probabilities of classes $\quad \frac{e^{s_k}}{\sum_j e^{s_j}} \xrightarrow{\text{to maximize}} -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) = L_i$ minimize this

ex: $\begin{bmatrix} 3.2 \\ 5.1 \\ -1.7 \end{bmatrix}$ actual class $\xrightarrow[\text{normalize}]{\text{exp+}}$ probabilities $[0,1]$ $\begin{bmatrix} 0.13 \\ 0.87 \\ 0 \end{bmatrix} \rightarrow -\log(0.13) = 0.89$

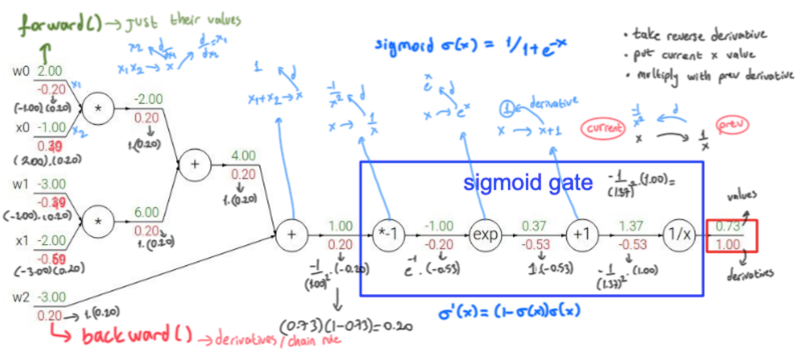
optimization

↳ numerical gradient = computing the gradients for each parameter and observing the change approximation, slow, easy to write

↳ analytic gradient = exact, fast, but error-prune (because of derivation with math)

↳ gradient check = ensuring the gradients computed by the back propagation (analytic gradient) are accurate by numerical gradients

↳ jacobian matrix = matrix of partial derivatives



activation functions

↳ sigmoid = not centered, vanishes for high and low values, expensive to compute, don't use

↳ tanh = centered at 0 ✓, vanishes, expensive

↳ relu = not centered, does not vanishes for positive values, converges x6 faster, use it ✓

weight initialization = W = random.randn(input_size, output_size) / np.sqrt (input_size)

batch normalization = reduces the dependency on initialization $\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$ (batch mean) $y_i \leftarrow \gamma \hat{x}_i + \beta$ (scale & shift)

↳ after fully connected layer and before non-linearity

↳ during the test time avg means of batches and variances are used

convolutional neural networks

- 32x32x3 image * 5x5x3 filter ⟶ 28x28x1 activation map
- 32x32x3 image * 6 5x5x3 filters ⟶ 28x28x6 activation map

these must be same

↳ 10 5x5x6 (filter) ⟶ 24x24x10 (map)

output size = (N-F) / stride + 1

↳ with padding = (2P + N - F) / stride + 1

example = input volume = 32x32x3 10 5x5 filters with stride 1, padding 2

↳ output size = (32+2.2-5)/1 +1 = 32 ⟶ 32x32x10

↳ number of parameters = 10 * (5x5x3 + 1) = 10 (75+1) = 760

↳ 75 dim dot product in each step

☆ 1x1 convolution can merge or extend the dimensions (can be usefull)

☆ recent trend towards smaller filters, deeper architectures, getting rid of pooling and fully connected layers (just convolution)

upsampling = used to increase size in the next step and in semantic segmentation