Deep Network 2

Softmax $z \to \left(\frac{e^{z_1}}{\sum_j e^{z_j}}, \dots, \frac{e^{z_K}}{\sum_j e^{z_j}} \right)$

converts input into probability error.

$$Pr[Y=y \mid X=x]$$

If some coordinate j of z dominates others, then softmax close to e_j

Cross entropy loss $p, q \in \Delta_d := \{v \in R^K : v \geq 0, \sum_i v_i = 1\}$

$$H(q,p) := -\sum_{i=1}^{K} p_i \ln q_i$$

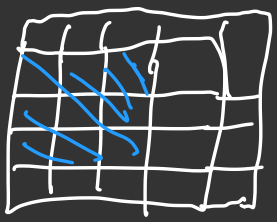
In pytorch: $l_{ce}(\hat{y}, y) = H(\text{softmax}(\hat{y}), e_y)$

$$= -\sum_{i=1}^{K} (e_y)_i \ln(\text{softmax}(\hat{y})) = \ln\left(\frac{\sum_i \exp(\hat{y}_i)}{\exp(\hat{y}_y)}\right) = -\hat{y}_y + \ln\left(\sum_i \exp(\hat{y}_i)\right)$$

$[0 0 \cdots \underset{\uparrow}{1} \cdots 0 0]$

yth

Max Pooling



Each filter takes the maximum.

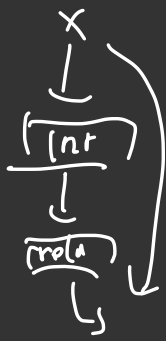
Batch Normalization

Standard output

$$x \to \frac{x - E(x)}{\text{stddev}(x)} \cdot \gamma + \beta$$

Skip connections

$$z \to z + f_i(z)$$



Gradient Descent and backpropagation

$$\nabla(L(yx^T;w)) = \nabla L(P(w)) = \frac{\partial L(P(w))}{\partial P(w)} \cdot \frac{\partial P(w)}{\partial w}$$

$$P_1 := \sigma_1(W_1 x)$$

$$P_{i+1} := \sigma_{i+1}(W_{i+1} P_i)$$

$$P_L := L(y, \sigma_L(W_L P_{L-1}))$$

$$\nabla_W(L(y, f(x;W))) = \frac{\partial P_L}{\partial P_i} \cdot \frac{\partial P_i}{\partial W_i}, \text{ where } \frac{\partial P_L}{\partial P_i} = \frac{\partial P_L}{\partial P_{i+1}} \cdot \frac{\partial P_{i+1}}{\partial P_i}$$

Forward pass: compute P_L by P_1 and more.

Backward pass: compute W_L down to W_1 via inductive chain rule

vanishing / exploding gradients

$$\nabla_{W_i}(L(y, f(x;W))) = \frac{\partial P_L}{\partial P_i} \cdot \frac{\partial P_i}{\partial W_i} = \prod_{j=i}^{L-1} \frac{\partial P_{j+1}}{\partial P_j}$$

With many-layered networks, this computation easily explodes and vanishes.

Initialization

$W_i \in \mathbb{R}^{d_i \times d_{i-1}} \sim \mathcal{N}(0, 1/d_{i-1})$ per coordinate.

Learning rates

$\eta \in \{0.001, 0.01, 0.1\}$

Data Augmentation

Besides original data, generate new data with some computation