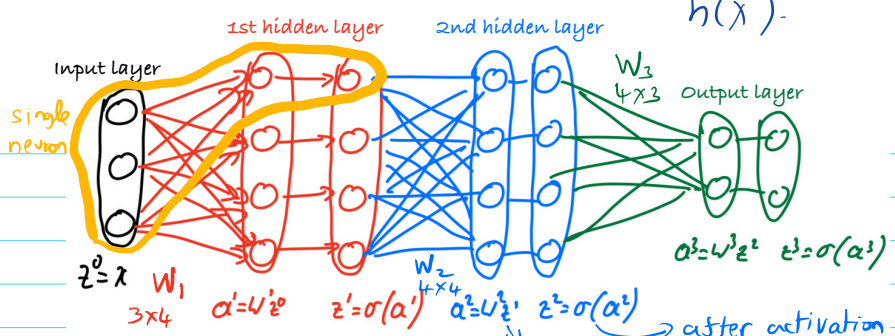
Neural Networks (Deep Learning)



Forward Pass (prediction):

$z_0 = x$ (Input Layer)

FOR $i = 1$ to L

$a_i = W_i z_{i-1} + \vec{b}_i$

$z_i = \sigma_i(a_i)$

END

Return z_L

Typically σ_L is different from others

As usual bias can be aborted in each layer by inserting 1 to each layer as extra feature

To learn the parameters of the NN, we use gradient descent!

How easy is it to compute gradients?

Warmup with 1 dimensional example without bias:

$x \in \mathbb{R}$ $\quad W_1, W_2, W_3, b_1, b_2, b_3 \in \mathbb{R}$

$\ell(h(x), y) = \frac{1}{2}(h(x) - y)^2$

$h(x) = W_3 \sigma(W_2 \sigma(W_1 \underbrace{x}_{z_0} + b_1) + b_2) + b_3$ (z_2 = argument of $W_3\sigma(\cdot)$, a_2 = argument of $W_2\sigma(\cdot)$, z_1 = argument of $W_1 x$, $z_3 = h(x)$)

ℓ' derivative of Loss

σ' derivative of σ

gradients w.r.t. W_3, b_3:

$$\frac{\partial \ell(h(x), y)}{\partial W_3} = \ell'(h(x), y) \times \frac{\partial z_3}{\partial W_3} = \ell'(h(x), y) \times z_2$$

$$\frac{\partial \ell(h(x), y)}{\partial b_3} = \ell'(h(x), y) \times \frac{\partial z_3}{\partial b_3} = \ell'(h(x), y)$$

Already computed

W_2, b_2:

$$\frac{\partial \ell(h(x), y)}{\partial W_2} = \ell'(h(x), y) \times \frac{\partial z_3}{\partial W_2} = \ell'(h(x), y)\, \sigma'(a_2) \frac{\partial a_2}{\partial W_2} = \ell'(h(x), y)\, \sigma'(a_2)\, z_1$$

$$\frac{\partial \ell(h(x), y)}{\partial b_2} = \ell'(h(x), y)\, \sigma'(a_2)$$

$\vec{a} \odot \vec{b} = \begin{bmatrix} a_1 b_1 \\ a_2 b_2 \\ a_3 b_3 \\ \vdots \end{bmatrix}$

elementwise product

Backward pass (gradient step)

$$\vec{\delta}_L = \nabla \ell(z_L, y) \odot \sigma_L'(a_L)$$

For $j = L-1$ to 1

$$W_j = W_j - \eta \vec{\delta}_j z_{j-1}^T$$

$$b_j = b_j - \eta \vec{\delta}_j$$

$$\vec{\delta}_{j-1} = \sigma'(a_{j-1}) \odot W_j^T \delta_j$$

End

1. Compute a's and z's from forward pass (store them)
2. Run backward pass using a's and z's from forward pass (computes gradients)

To minimize training loss using gradient descent:

$$\text{Training loss} = \frac{1}{n} \sum_{i=1}^{n} \ell(h(x_i), y_i)$$

each one using backprop

$$\nabla \text{Training loss} = \frac{1}{n} \sum_{i=1}^{n} \nabla \ell(h(x_i), y_i)$$

this can be expensive when n is large

Stochastic Gradient Descent:

Approximate ∇ Training loss using 1 or m $\ll n$ samples

$\nabla \approx \nabla \ell(h(x_i), y_i)$ $\quad (x_i, y_i) \overset{\text{randomly}}{\sim} D$

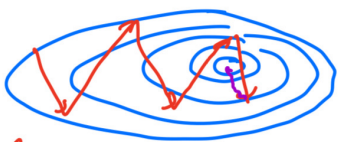
For every layer k, $W_k \leftarrow W_k - \eta \nabla_{W_k} \ell(h(x_i), y_i)$

One update using one or m samples randomly drawn from D

Important optimization tricks:

- Use Momentum
- Reduce stepsize during optimization
- Use mini-batch (SGD with $m \simeq 64$ inputs)
- scale features to be within $[0,1]$
- de-correlate features (PCA)

$$G \leftarrow \beta G + \frac{\partial \mathcal{L}}{\partial w}$$
$$w \leftarrow w - \alpha G$$



large step-size
small step-size

Batch-Norm: Re-normalize activations based on current mini-batch.

$$a_i \leftarrow \gamma_i \frac{a_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} + \beta_i$$

γ_i, β_i: parameters that are learned

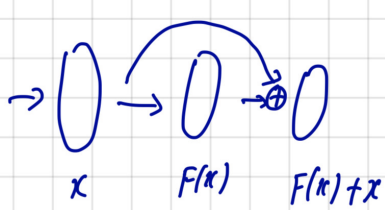
μ_i = mean activation in mini-batch

σ_i^2 = variance in mini-batch

Why? 1. reduces "internal covariate shift"

2. Smooths out loss

Residual Connection:

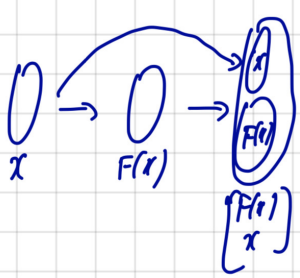


Add input to output activations after each layer*.

Why? 1. vanishing gradients problem

2. $\phi(x) = x$ is hard to learn

Dense Connections:



Concatenate input to output activations after each layer*.