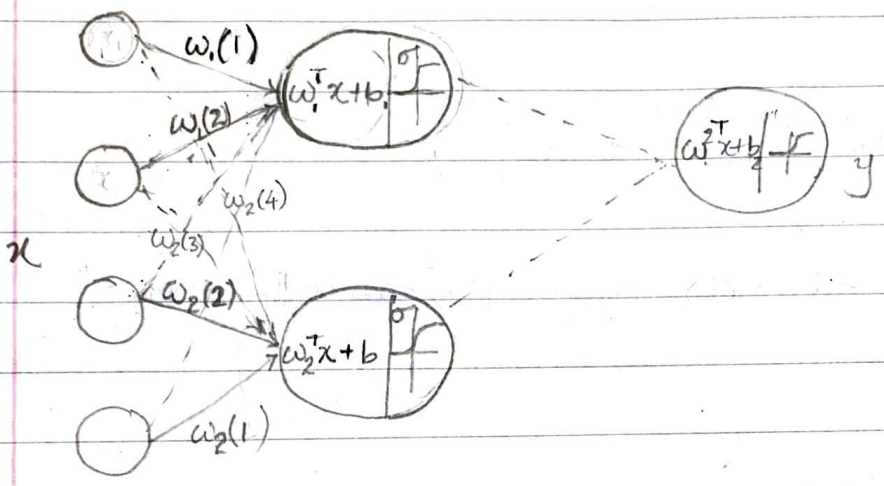
Neural Networks

A neural network takes a logistic regression or perceptron to the 'next level'



← Sometimes called Multi-Layer Perceptron

Here, our input data is transformed into an abstract feature space by multiple perception units or neurons. and this can be stacked layer after layer. The result is that for sufficiently large number of parameters the network becomes a universal approximator between x and y. characterized by its weights,

$$y = f(x, \theta)$$

where θ represents the set of all weights, vectors w^2s used in the network. The weights between layers: for example in the above diagram

$$W^1 = \begin{bmatrix} | & | \\ \omega_1 & \omega_2 \\ | & | \end{bmatrix}$$ is the weight matrix between the input and hidden layer

Hidden Layers

Each node in an hidden layer is a single perceptron connected to all nodes in the previous layer. Its value is the projection of the previous layer onto its weight plus a bias, all passed through an activation function

$$h_i^L = \sigma\left(\left(W_i^L\right)^T h^{L-1} + \omega_{0,i}^L\right)$$

where W_i is the ith column of W^L and $h^{L-1} = \left[h_1^{L-1}, \dots h_{N_{L-1}}^{L-1}\right]^T$

and thus

$$h^L = \sigma\left(W^{L\,T} h^{L-1} + \omega_0^L\right)$$

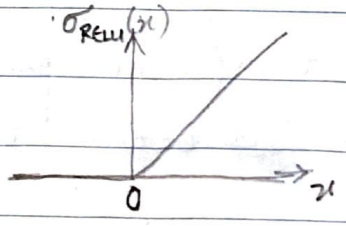
where $\omega_0^L = \left[\omega_{0,1}^L, \dots, \omega_{0,N_L}^L\right]^T$

and $\sigma(x)$ is applied element-wise

Activation Functions

Several activation functions can be used in addition to the sigmoid function. One of the most popular is the ReLU (Rectified Linear Unit)

$$\sigma_{RELU}(x) = \max(0, x)$$



ReLU is more commonly used in training NN, when gradient stability and sparse activation is needed

Output layer

$$\hat{y} = g\left((W^{L_y})^T h^{L_y - 1} + w_0^{L_y}\right)$$

g is the output (activation) function that depends on the task

- Binary classification
- Multi-class classification
- Regression

For binary classification we can use the sigmoid function
In general, it is common to use a softmax function for multiclass classification

$$\hat{y}_k = \frac{\exp\left[(W_k^{L_y})^T h^{L_y - 1} + w_{0,k}^{L_y}\right]}{\sum_{k=1}^{C} \exp\left[(W_k^{L_y})^T h^{L_y - 1} + w_{0,k}^{L_y}\right]}$$

Neural Network Training

Given training data $D = \{x_i, y_i\}_{i=1}^{N}$, the goal in training a NN is to learn the parameters θ that minimizes a loss function

$$\ell(D, \theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(x_i, y_i, \theta) + \alpha \underbrace{R(\theta)}_{\text{Regularizer (constrains learning space)}}$$

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, \ell(D, \theta)$$

$$\theta = \{W^1, \omega_0^1, W_2, \omega_0^2, \ldots, W^{Ly}, \omega_0^{Ly}\}$$

Loss Functions

Squared loss: $\ell(y_i, \hat{y}_i) = \frac{1}{2}(y_i - \hat{y}_i)^T (y_i - \hat{y}_i)$

(2-norm of the error)

This loss is applicable to both regression and classification tasks.

Cross-entropy loss: $l(y_i, \hat{y}_i) = -y_i^T \log \hat{y}_i$

$$= \sum_{k=1}^{C} y_i(k) \log \hat{y}_i(k)$$

This is typically used for classification-tasks

Training Steps

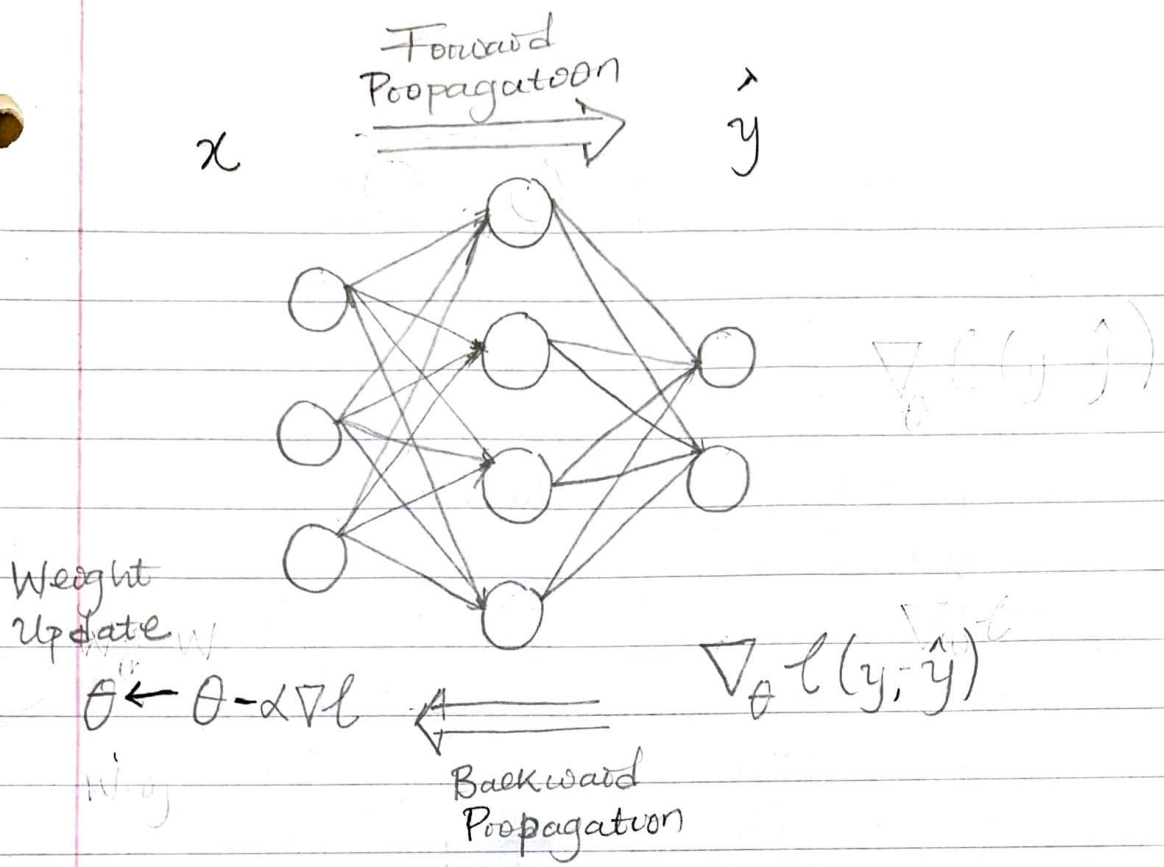
① Forward pass: compute values for the hidden layers, output layer $\hat{y}$ & loss $l(y, \hat{y})$ based on current parameters θ

② Backward pass: Starting at the output layer, compute the derivatives of the loss function with respect to the parameters of θ, $\nabla_\theta l(y, \hat{y}; \theta)$.
Pass the output gradient backward through the network, layer by layer, by recursively computing the weight gradients of each layer:

③ Weight update:

$$W^{Ly} \leftarrow W^{Ly} - \eta \nabla_{W^{Ly}} l \; ; \; \omega_0^{Ly} \leftarrow \omega_0^{Ly} - \eta \nabla_{\omega_0^{Ly}} l$$

$$W^1 \leftarrow W^1 - \eta \nabla_{W^1} l \qquad \omega_0^1 \leftarrow \omega_0^1 - \eta \nabla_{\omega_0^1} l$$



Backpropagation Gradients

Output Gradient:

(1) Square loss function: $\ell(y_i, \hat{y}_i) = \frac{1}{2}(y_i - \hat{y}_i)^T(y_i - \hat{y}_i)$

$$\frac{\partial \ell_i}{\partial \hat{y}_i} = \nabla_{\hat{y}_i} \ell_i = (\hat{y}_i - y_i)^T$$

(2) Cross-entropy loss function: $\ell(y_K, \hat{y}_K) = -y_K^T \log \hat{y}_K$

$$\frac{\partial \ell_i}{\partial \hat{y}_i} = \nabla_{\hat{y}_i} \ell_i = -y_i^T [\text{diag}(\hat{y}_i)]^{-1} = -\begin{bmatrix} \frac{y_i(1)}{\hat{y}_i(1)} \\ \vdots \\ \frac{y_i(C)}{\hat{y}_i(C)} \end{bmatrix}$$

An Aside on Matrix Calculus: Identities

Let $y \in \mathbb{R}^m$ and $x \in \mathbb{R}^n$.

① Numerator layout (Jacobian formulation)

$$\frac{dy}{dx} \in \mathbb{R}^{m \times n} = \begin{bmatrix} \frac{dy_1}{dx_1} & \frac{dy_1}{dx_2} & \cdots & \frac{dy_1}{dx_n} \\ \frac{dy_2}{dx_1} & & & \\ \vdots & & & \\ \frac{dy_m}{dx_1} & & & \frac{dy_m}{dx_n} \end{bmatrix} = J$$

e.g. $\frac{\partial u^T v}{\partial x} = u^T \frac{\partial v}{\partial x} + v^T \frac{\partial u}{\partial x}$, $\frac{\partial f(g(u))}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial u} \frac{\partial u}{\partial x}$

② Denominator layout (Hessian formulation)

$$\frac{dy}{dx} \in \mathbb{R}^{n \times m} = J^T$$

Ex.

$$\frac{\partial Ax}{\partial x} = A^T \; ; \; \frac{\partial u^T v}{\partial x} = \frac{\partial u}{\partial x} v + \frac{\partial v}{\partial x} u$$

$$\frac{dy}{dx} \text{ (for } y \in \mathbb{R}) = \begin{bmatrix} \frac{dy}{dx_1} \\ \vdots \\ \frac{dy}{dx_n} \end{bmatrix}$$

$$\frac{\partial f(g(u))}{dx} = \frac{\partial u}{\partial x} \frac{\partial g}{\partial u} \frac{\partial f}{\partial g}$$

Output layer $\hat{y}_i = g\left[(W^{Ly})^T h^{Ly-1} + \omega_0^{Ly}\right]$

$$\nabla_{W^{Ly}} L_i = \frac{\partial L_i}{\partial \hat{y}_i} \frac{\partial \hat{y}}{\partial W^{Ly}}$$

$$\nabla_{\omega_0^{Ly}} L_i = \frac{\partial L_i}{\partial \hat{y}_i} \frac{\partial \hat{y}}{\partial \omega_0^{Ly}}$$

$$\nabla_{h_k^{Ly-1}} L_i = \frac{\partial L_i}{\partial \hat{y}_i} \frac{\partial \hat{y}}{\partial h_k^{Ly-1}}$$

Let $z_i = (W^{Ly})^T h_i^{Ly-1} + \omega_0^{Ly}$

$$\hat{y}_i = g(z_i) = \sigma(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{C} z_i(k)}$$

$$\frac{\partial \hat{y}_i}{\partial z_i} = \text{diag}(\hat{y}_i) - \frac{e^{z_i}(e^{z_i})^T}{\left[\sum_{k=1}^{C} z_i(k)\right]^2} = \text{diag}(\hat{y}_i) - \hat{y}_i \hat{y}_i^T$$

Or we can write

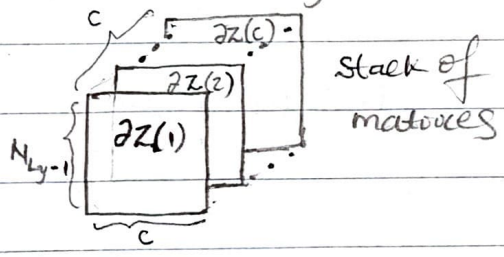
$$\frac{\partial \hat{y}_i(k)}{\partial z_i(j)} = \begin{cases} \hat{y}_i(k)\left(1 - \hat{y}_i(k)\right) & \text{if } k = j \\ -\hat{y}_i(k)\,\hat{y}_i(j) & \text{else} \end{cases}$$

Meanwhile, $\frac{\partial z_i}{\partial W^{Ly}}$ is given by

$$\frac{\partial z_i}{\partial W^{Ly}} = \begin{matrix} \frac{\partial z_i(c)}{\partial W^{Ly}} = \left[\frac{\partial z_i(c)}{\partial W_1^{Ly}} \quad \frac{\partial z_i(c)}{\partial W_2^{Ly}} \cdots \frac{\partial z_i(c)}{\partial W_c^{Ly}}\right] \\ \vdots \\ \frac{\partial z_i(2)}{\partial W^{Ly}} = \left[\frac{\partial z_i(2)}{\partial W_1^{Ly}} \quad \frac{\partial z_i(2)}{\partial W_2^{Ly}} \cdots \frac{\partial z_i(2)}{\partial W_c^{Ly}}\right] \\ \frac{\partial z_i(1)}{\partial W^{Ly}} = \underbrace{\left[\frac{\partial z_i(1)}{\partial W_1^{Ly}} \quad \frac{\partial z_i(1)}{\partial W_2^{Ly}} \cdots \frac{\partial z_i(1)}{\partial W_c^{Ly}}\right]}_{C} \end{matrix} \Big\} N_{Ly-1}$$

where W_k^{Ly} is the k^{th} column of the matrix W^{Ly}.

The above is a third-order tensor (of size $C \times N_{Ly-1} \times C$)



$$\nabla_{W^{Ly}} l_i = \frac{\partial l_i}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial z_i} \frac{\partial z_i}{\partial W^{Ly}}$$

$$= \underbrace{\nabla_{z_i} l_i}_{1 \times C} \underbrace{\frac{\partial z_i}{\partial W^{Ly}}}_{C \times N_y \times C}$$

To see how we implement this vector-tensor multiplication, think about vector-matrix multiplication:

$$\underbrace{v}_{1 \times C} \underbrace{M}_{C \times m} = [v_1 \; v_2 \cdots v_c] \begin{bmatrix} — \vec{m}_1 — \\ — \vec{m}_2 — \\ \vdots \\ — \vec{m}_c — \end{bmatrix} = \sum_{i=1}^{c} v_i \vec{m}_i$$

Similarly, we can now write

$$\nabla_{W^{Ly}} l_i = \sum_{k=1}^{C} \nabla_{z_i} l_i(k) \left[\frac{\partial z_i(k)}{\partial W_1^{Ly}}, \frac{\partial z_i(k)}{\partial W_2^{Ly}} \cdots \frac{\partial z_i(k)}{\partial W_C^{Ly}} \right]$$

$$= \sum_{k=1}^{C} \nabla_{z_i} l_i(k) \left[\delta_{k,1} h_i^{Ly-1}, \delta_{k,2} h_i^{Ly-1} \cdots \delta_{k,C} h_i^{Ly-1} \right]$$

where $\delta_{k,j}$ is the indicator function.

It is left as an exercise to show that $\nabla_{W^{Ly}} l_i$ may be compactly expressed as

$$\nabla_{W^{Ly}} l_i = h_i^{Ly-1} \nabla_{z_i} l_i$$

We know that $\frac{\partial z_i}{\partial h_i^{Ly-1}} = (W^{Ly})^T$ and $\frac{\partial z_i}{\partial w_0^{Ly}} = I$

$$\therefore \nabla_{h_i^{Ly-1}} l_i = \nabla_{z_i} l_i (W^{Ly})^T, \quad \nabla_{w_0^{Ly}} l_i = \nabla_{z_i} l_i$$

Hidden layer

For each hidden layer h_i^L, let h^{L-1} be its input, and W^L and w_0^L be its parameters

$$h_i^L = \sigma_L\left((W^L)^T h_i^{L-1} + w_0^L\right)$$

Let $z_i^L = (W^L)^T h_i^{L-1} + w_0^L$

$$h_i^L = \sigma_L(z_i^L)$$

For a sigmoid function: $h_i^L = \sigma_L(z_i^L) = \frac{1}{1+e^{-z_i^L}}$

$$\frac{\partial h_i^L}{\partial z_i^L} = \text{diag}\left(\sigma_L(z_i^L) * (1-\sigma_L(z_i^L))\right)$$ ← can you show this?

where '*' denotes element-wise multiplication.

For a ReLU function: $h_i^L = \sigma_L(z_i^L) = \max(0, z_i^L)$

$$\frac{\partial h_i^L}{\partial z_i^L} = \text{diag}\left(\sigma_L(z_i^L)^{\bullet} / \sigma_L(z_i^L)\right)$$

where '$\bullet$/' denotes element-wise division.

As with the gradient derivation for the output layer,

$\frac{\partial z_i^L}{\partial W^L}$ is a tensor. Now we know that

$$\nabla_{W^L} l_i = \nabla_{h_i^L} l_i \; \frac{\partial h_i^L}{\partial z_i^L} \; \frac{\partial z_i^L}{\partial W^L}$$

(already known)

$$= \nabla_{z_i^L} l_i \; \frac{\partial z_i^L}{\partial W^L}$$

($\nabla_{z_i^L} l_i$: vector $(1 \times N_L)$; $\frac{\partial z_i^L}{\partial W^L}$: tensor $(N_L \times N_{L-1} \times N_L)$)

Thus, we can write (following from gradient expressions for the hidden layer)

$$\nabla_{W^L} l_i = \sum_{k=1}^{N_L} \nabla_{z_i^L} l_i \left[\frac{\partial z_i^L(k)}{\partial W_1^L} \quad \frac{\partial z_i^L(k)}{\partial W_2^L} \quad \cdots \quad \frac{\partial z_i^L(k)}{\partial W_{N_L}^L} \right]$$

$$\left(= h_i^{L-1} \nabla_{z_i^L} l_i \right) \leftarrow \text{(can you show this?)}$$

$$\nabla_{h_i^{L-1}} l_i = \nabla_{z_i^L} l_i \, (W^L)^T \; ; \quad \nabla_{w_0^L} l_i = \nabla_{z_i} l_i$$

Recall that the total loss $\mathcal{L}(D, \theta)$ is the average over all individual losses $\mathcal{L}_i$ for each data point. Hence, the total gradient with respect to each parameter is the average gradient over all data points.

Hence in our update

$$\theta^{new} \leftarrow \theta^{old} - \eta \cdot \nabla_\theta \mathcal{L}$$

can be rewritten

$$\theta^{new} \leftarrow \theta^{old} - \eta \cdot \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \mathcal{L}_i$$

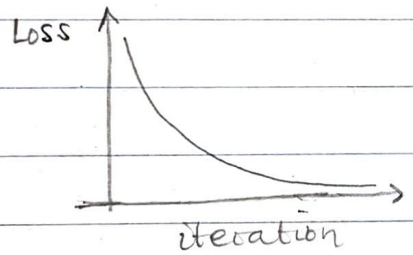
It is left as an exercise to show that the summation of individual $\nabla_\theta \mathcal{L}_i$ can be circumvented by efficiently expressing the total gradient as matrix (tensor) multiplications following the preceeding expressions we have derived.

Stochastic Gradient Descent

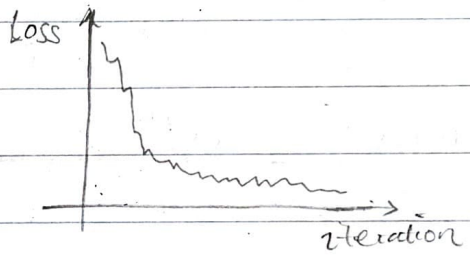
We can update the parameters of θ at each iteration of our gradient using the total gradient over our entire training set, that is, over N data pairs. But as you may suspect, if the training data is large, each (total) gradient evaluation will be

quite expensive. To alleviate this, at each gradient descent step, we can just randomly pick a mini-batch from the entire data set (say N_B data points) and carry out the update

While each gradient step may no longer be in the direction of steepest descent with respect to the entire training data set, over many iterations (or epochs), the loss function value trends downward, as desired.



(Batch) Gradient Descent



Stochastic (Mini-batch) Gradient Descent

Note: The term epoch denotes the number of iterations involved in going through all the training data.

Regularization

It is typical to use L1, L2 or Frobenius norm of our NN weights for regularization.

E.g.

Frobenius norm: $R(\theta_i = W^L) = \frac{1}{2}\|W^L\|_F^2$

$$= \sum_{i=1}^{N_L} \sum_{j=1}^{N_{L-1}} |w_{i,j}|^2$$

$$\frac{\partial R(W^L)}{\partial W^L} = W^L$$

The update, including regularization, is now

$$W^L \leftarrow W^L - \eta\left(\nabla_{W^L} \ell + \alpha \frac{\partial R(W^L)}{\partial W^L}\right)$$

Output Encoding

For a binary classification probably, it suffices to have a single output layer.

In a multi-class problem, it is typical to have an output layer whose dimension matches the number of classes. For example, if we have 3 categories: A, B, C; our output layer can have dimension = 3, and each category can be encoded as follows

$$\text{'A'} \rightarrow \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \text{'B'} \rightarrow \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \text{'C'} \rightarrow \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

One-hot encoding