NNs

Static data processing

The goal of a **pattern recognition** system is to **estimate the correct class corresponding to a given features vector**. Consequently, the aim of learning algorithm is to determine the **decision boundaries** of the pattern recognition. In other words, pattern recognition can also be cast as a **function approximation problem**.
Consider a two-classis (binary) classification problem. If:

$$\text{if} \quad w_1x_1 + w_2x_2 + \cdots + w_Mx_M \geq threshold, \qquad y = +1, \qquad \text{else} \quad y = -1$$

We can consider the threshold as an additional (negative) weight:

($\mathbf{x_0 w_0}$: $\mathrm{w_0}$ is the threshold or bias; $\mathrm{x_0}$ is 1)

$$\sum_{\mathbf{i=1}}^{\mathbf{M}} \mathbf{w_i x_i + x_0 w_0 \geq 0 \;\; then \;\; y = +1}$$

$$\sum_{\mathbf{i=1}}^{\mathbf{M}} \mathbf{w_i x_i + x_0 w_0 < 0 \;\; then \;\; y = -1}$$

The **regression** is a method of estimating a **curve interpolating the data** with parametric model of the form $\mathbf{y = f(x_i, w) = w^T x_i}$.

The **classification** task, is like the regression problem except for the fact that:

1. the output takes discrete values, for example, **0 and 1 (or -1 and +1)**
2. The output of the classifier $\mathbf{y = \varphi(w^T x)}$, can be written as $y \in [0,1]$, where is defined in the continuous interval, i.e. such that, although it is not, can be considered as a probability.

Dynamic data processing

Time is important:
Countinuous Time (CT) [differential eq.] vs Discrete Time (DT) [difference of eq.]

Multi Layers Perceptron (MLP)

The most learning used algorithm for MLP, that is **a direct extension of the Widrow LMS**, is the so called **Back-Propagation (BP) learning algorithm**.

Forward

The output of a MLP network can be computed as:

$$\begin{cases} s_k^{(l)} = \sum_{j=0}^{N_{l-1}} w_{kj}^{(l)} x_j^{(l-1)}, \qquad x_0^{(l-1)} \equiv 1 \\ x_k^{(l)} = \varphi_j^{(l)}\left(s_k^{(l)}\right) \end{cases}, \qquad \text{for } l = 1, \dots, L \quad \text{for } k = 1, \dots, N_l$$

Compact notation:

$$s^{(l)} = \left[s_1^{(l)}\, s_2^{(l)} \dots s_{N_l}^{(l)}\right]^T$$

$$w_k^{(l)} = \left[w_{0k}^{(l)}\, w_{1k}^{(l)} \dots w_{N_l k}^{(l)}\right]^T$$

$$\Phi^{(l)} = \left[\varphi_1^{(l)}(\cdot) \dots \varphi_{N_l}^{(l)}(\cdot)\right]^T$$

$$x^{(l-1)} = \left[1\, x_1^{(l-1)} \dots s_{N_{l-1}}^{(l-1)}\right]^T$$

The neuron linear-combiner *l-th*:

$$s^{(l)} = \underset{(N_l \times N_{l-1})}{W^{(l)}} \cdot \underset{(N_{l-1} \times 1)}{x^{(l-1)}}$$

The output:

$$y \equiv x^{(L)} = \Phi^{(L)}\left(W^{(L)}\Phi^{(L-1)}\left(\dots \Phi^{(2)}\left(W^{(2)}\Phi^{(1)}\left(W^{(1)}x^{(0)}\right)\right)\right)\right)$$

Back-Propagation

Derive the LF with chain rule at the end we obtain the generalized delta rule for MLP:

$$\delta^{(l)} = W^{T(l+1)}\delta^{(l+1)} \odot \Phi'\left(s^{(l)}\right)$$

$$W_{new}^{(l)} = W_{old}^{(l)} + \mu\delta^{(l)}x^{T(l)}$$

Due to the way the formula is defined, with multiple layers we will have the derivative of the activation function multiplied several times. For an AF like sigmoid, the derivative has a maximum value of 0.25, which means that the generated gradient will tend to be 0: this problem is called "vanishing gradient". There are several solutions to this problem, the easiest being to choose an AF with a derivative that does not present this problem such as the ReLU.

Recurrent Neural Network (RNN)

Buffered RNNs

The network memory is made with an external MLP to manage the delay line.

State Space RNNs

The memory of the network is realized in its **internal state**. Introducing a *state variable* $\mathbf{h}_n$ we define a model that takes in our current input $\mathbf{x}_n$ also the state of the previous time-step.

Elmam

$$h_n = \Phi^{(1)}\left(W_1^{(1)} x_n + W_2^{(1)} h_{n-1}\right)$$
$$h_{n-1} = z^{-1}(h_n)$$
$$y_n = \Phi^{(2)}(h_n), \qquad \text{or } y_n \subseteq h_n$$

Jordan

$$h_n^{(1)} = \Phi^{(1)}\left(W_x^{(1)} x_n + W_y^{(1)} y_{n-1}\right)$$
$$h_n^{(2)} = \Phi^{(2)}\left(W^{(2)} h^{(1)}\right)$$
$$\vdots$$
$$y_n = h_n^{(k)} = \Phi^{(k)}\left(W^{(k)} h_n^{(k-1)}\right)$$

Deep RNN

$$h_n^{(i)} = \Phi^{(i)}\left(W_b^{(i)} h_n^{(i-1)} + W_a^{(i)} h_{n-1}^{(i)}\right), \qquad \text{for } i = 1,2,\dots,K$$
$$y_n = h_n^{(K)}$$
The network is a rather general model that includes, as a particular case, the Elman network

Gradient-based RNN: Backpropagation Through Time (BPTT)

State Space RNNs

BPTT is local in space but not in time. Assuming a training set consisting of N pairs, the BPTT has a CF defined as:

$$J(w) = E^{\text{total}}(1,N) = \sum_{n=1}^{N} \|d_n - y_n\|^2$$

The Real-Time Recurrent Learning

It is an Elman network and is local in time but not in space.

Convolutional Neural Networks

$$x^{(l)} = \Phi\left(W^{(l)} x^{(l-1)}\right), \qquad \text{became}, \qquad x^{(l)} = \Psi\left[\Phi\left(\widetilde{W}^{(l)} * x^{(l-1)}\right)\right]$$
$\widetilde{W}^{(l)}$ is the **convolution kernel**.
$\Psi(\cdot)$ is an operator, usually denoted as *detector* or *sub-sampling-stage*, that further modify the output of the layer.
Three stages:

1. Convolution;
2. Activation;
3. Pooling or subsampling.

Generative models

Autoencoder AE

The autoencoder is a network that takes a given input, encodes it through an encoder, obtaining a hidden rapresentation h of the data. At this point **h** is passed to a decoder with the objective to obtain an output as close as possible to the initial input.
Encoder, produces code or latent representation **h**:

$$h = \Phi(W_1 x + b) = f(x)$$

Decoder, produces reconstruction of the input $\hat{\mathbf{x}}$ through the latent variable **h:**

$$\hat{\mathbf{x}} = \Phi(W_2 h + b) = g(h)$$

So the AE's output can be written as $\hat{\mathbf{x}} = g\big(f(x)\big)$
The loss function is:

$$J(x) = \arg\min_{w} \|x - \hat{x}\|_2^2 = \arg\min_{w} \left\|x - \Phi\big(W_2 \Phi(W_1 x)\big)\right\|_2^2$$

Autoencoder AE

The encoder transforms the input samples **x** into two parameters in a latent space **z**, the mean value **μ**, and the variance **σ** (or its **log σ**).

$$z = \mu + \exp(\log\sigma) \odot \epsilon, \qquad \text{where } \epsilon \text{ is a random normal tensor, i.e. } \epsilon \sim \mathcal{N}(0,1)$$

The learning of the parameters μ and σ, is done by considering the combination of two loss functions:

1. a reconstruction loss that forces the decoded samples to match the initial inputs (just as in our previous autoencoders),
2. the KL divergence between the learned latent distribution and the previous distribution, which acts as a regularization term.

In other words, we consider a loss function $\mathcal{L}(\ldots)$:

$$\mathcal{L}(q) = \mathbb{E}_{z-q(z|x)}[\ln p_{\text{model}}(x|z)] - D_{KL}\big(q(z|x)||p_{\text{model}}(z)\big) \leq \ln p_{\text{model}}(x)$$

Generative Adversarial Networks

In this game there are two players: the **forger** (or **generator G**) and the **expert** (or discriminator D). The forger continues to generate new forgeries until the expert can no longer tell the true from the false. The game ends when the expert is no longer able to recognize the true from the fake. Equilibrium when **Pr(True) = Pr(false) = 0.5**.

Optimizer

An Optimization Problem (OP) is defined as the minimization or maximization of a real-valued scalar functional called **Cost Function (CF)** (or depending on the contexts, also denoted as Objective Function, Energy function, Loss Function, ...), which may be subject to **equality and inequality constraints which delimit the space of the feasible region of solutions.**

The **scalar** CF, referred to **as** $\mathbf{J(w)}$, $\mathbf{J} : \mathbb{R}^{\mathbf{M}} \to \mathbb{R}$, is defined over the vector of unknown variables $\mathbf{w} \in \mathbb{R}^{\mathbf{M}}$, containing the set of the decision variables, related to the given problem.

The vector **w***, called optimal solution, which minimizes the CF J(w).

Gradient descent algorithm or steepest-descent algorithm (SDA) can be formalized as:

$$\mathbf{w_{k+1} = w_k - \mu_k \nabla J(w_k)}$$

Second-order algorithms (Newton's method):

$$\mathbf{w_{k+1} = w_k - \mu_k \left[\nabla^2 J(w_k)\right]^{-1} \nabla J(w_k) \qquad \nabla^2 J(w_k) \neq 0}$$

Levernberg-Marquardt's Variant:

$$\mathbf{w_{k+1} = w_k - \mu_k \left[\nabla^2 J(w_k) + \delta I\right]^{-1} \nabla J(w_k) \qquad \nabla^2 J(w_k) \neq 0, \qquad \boldsymbol{\delta} > 0}$$

The constraints represent the a priori knowledge of the region of the research space in which the solution to the problem is found: the **feasible region**:

$$\underset{w \in \mathbb{R}^M}{\arg\min} J(w) \quad \text{s.t.} \quad \begin{cases} g_i(w) \leq 0, & i = 0,1,\dots,P-1 \\ h_i(w) = 0, & j = 0,1,\dots,Q-1 \end{cases}$$

The equality constraint **h** and the inequality constraint **g**.

Momentum BP

The idea is to accelerate or decelerate the progress depending on the "speed" of the gradient:
$w_n = w_{n-1} - v_n, \qquad v_n = \gamma v_{n-1} + \mu J(w)$

Momentum BP: the Nesterov accelerated gradient variant (NAG)

This method insert a predictive term to the calculation of the moment:
$w_n = w_{n-1} - v_n, \qquad v_n = \gamma v_{n-1} + \mu J(w - \gamma v_{n-1})$

Adaptive step-size based on gradient energy: The AdaGrad

$$w_n = w_{n-1} - \frac{\mu}{\sqrt{G_n} + \delta} \odot g_n$$

Where $G_n = g_n g_n^T$ and δ is a regularization term

Adaptive step-size based on gradient energy: The AdaDelta

We consider only the gradient energy:

$$\Delta w_n = -\frac{\mathrm{RMS}(\Delta w_{n-1})}{\mathrm{RMS}[g]} \cdot g_n$$

$$\Delta w_{n+1} = w_n + \Delta w_n$$

Where RMS stand for "root mean square":

$$\mathrm{RMS}(g_n) \triangleq \sqrt{E_{g_n^2} + \delta}, \qquad E_{g_n^2} = \gamma E_{g_{n-1}^2} + (1-\gamma) \cdot g_n^2, \qquad g_n^2 = g_n \odot g_n$$

$$\mathrm{RMS}(\Delta w_n) \triangleq \sqrt{E_{\Delta w_n^2} + \delta}, \qquad E_{\Delta w_n^2} = \gamma E_{\Delta w_{n-1}^2} + (1-\gamma) \cdot \Delta w_n^2, \qquad \Delta w_n^2 = \Delta w_n \odot \Delta w_n$$

ADAptive low-order Moments estimation (ADAM)

We consider the first moment (the mean) and the second moment (the variance) of the gradient:

$$m_n = \beta_1 m_{n-1} + (1-\beta_1) g_n$$

$$v_n = \beta_2 v_{n-1} + (1-\beta_2) g_n \odot g_n$$

The authors propose default values of $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

Finally weights update rule is:

$$w_n = w_{n-1} - \mu \frac{\widehat{m}_n}{\sqrt{\hat{v}_n} + \delta}, \qquad \widehat{m}_n = \frac{m_n}{1-\beta_1^n}, \qquad \hat{v}_n = \frac{v_n}{1-\beta_2^n}$$

Loss Function

Least Square (LS)

The term "least squares" is related to the sum of the squares of the error **e**.

$$\mathbf{y} = \mathbf{f}(\mathbf{X}, \mathbf{w}) + \mathbf{e} \;\longrightarrow\; \mathbf{e} = \mathbf{d} - \mathbf{f}(\mathbf{X}, \mathbf{w})$$

$$\mathbf{J}(\mathbf{w}) = \|\mathbf{e}\|_2^2 = \|\mathbf{d} - \mathbf{f}(\mathbf{X}, \mathbf{w})\|_2^2$$

Covariance

$$\boldsymbol{\Sigma} = \mathbf{E}\{(\mathbf{x} - \mathbf{E}\{\mathbf{x}\})(\mathbf{x} - \mathbf{E}\{\mathbf{x}\})^{\mathbf{T}}\}$$

Correlation matrix

$$\mathbf{R} = \mathbf{E}\{\mathbf{x}\mathbf{x}^{\mathbf{T}}\}$$

Batch

Mean value:

$$\mathbf{w} \triangleq \hat{\mathbf{E}}\{\mathbf{x}\} = \frac{\mathbf{1}}{\mathbf{N}} \sum_{\mathbf{n}=\mathbf{1}}^{\mathbf{N}} \mathbf{x}_{\mathbf{n}}$$

Variance:

$$\sigma_x^2 \triangleq \hat{E}\left\{\left[x - \hat{E}\{x\}\right]^2\right\} = \frac{\mathbf{1}}{\mathbf{P}}\sum_{\mathbf{n=1}}^{\mathbf{N}}(\mathbf{x_n} - \mathbf{w})^{\mathbf{2}}$$

$$\sigma_x^2 \triangleq \hat{E}\{x^2 - 2\{x\}^2 + \hat{E}\{x\}^2\} = \hat{E}\{x^2\} - 2\hat{E}\{x\}\hat{E}\{x\} + \hat{E}\{x\}^2 = \hat{E}\{x^2\} - \hat{E}\{x\}^2 = \frac{\mathbf{1}}{\mathbf{P}}\sum_{\mathbf{n=1}}^{\mathbf{N}}\mathbf{x_n^2} - \mathbf{w^2}$$

$$\boldsymbol{\sigma_x^2} = \frac{\sum_{\mathbf{n=1}}^{\mathbf{N}}\mathbf{x_n^2} - \frac{\left(\sum_{\mathbf{n=1}}^{\mathbf{N}}\mathbf{x_n}\right)^2}{\mathbf{N}}}{\mathbf{P}}$$

Correlation Matrix:

$$\mathbf{R_{xx}} = \frac{\mathbf{1}}{\mathbf{N}}\mathbf{X^TX}$$

Cross-Correlation Vector:

$$\mathbf{R_{xd}} = \frac{\mathbf{1}}{\mathbf{N}}\mathbf{X^Td}$$

Ordinary Linear Least Square Error (OLS):

Error is:

$\mathbf{e} = \mathbf{d} - \mathbf{Xw}$

In the OLS the loss function can be written as *residual sum of square error* (SSE or RSS):

$$\mathbf{J}(\mathbf{w}) = \sum_{i=1}^{N} e_i^2 = \sum_{i=1}^{N}\left(d_i - x_i^T w\right)^2 = e^T e = [d - Xw]^T[d - XW] = \mathbf{d^Td} - \mathbf{2w^TX^Td} + \mathbf{w^TX^TXw}$$

And the gradient of J(w) is:

$$\nabla\mathbf{J}(\mathbf{w}) \triangleq \frac{\partial\mathbf{J}(\mathbf{w})}{\partial\mathbf{w}} = -\mathbf{2X^Td} + \mathbf{2X^TXw}$$

The minimum of the loss function J(w) is when the gradient $\nabla J(w) \to 0$, i.e., for:

$\mathbf{X^TXw} = \mathbf{X^Td}$

Then solving for **w** we obtain LS solution:

$$\boxed{\mathbf{w_{LS}} = \left(\mathbf{X^TX}\right)^{-\mathbf{1}}\mathbf{X^Td} = \mathbf{X^T}\left(\mathbf{XX^T}\right)^{-\mathbf{1}}\mathbf{d} = \mathbf{R}_{xx}^{-1}\mathbf{R_{xd}}}$$

With **Yule Walker regularization**:

$$\boxed{\mathbf{w_{LS}} = (\mathbf{X^TX} + \boldsymbol{\delta}\mathbf{I})^{-\mathbf{1}}\mathbf{X^Td} = \mathbf{X^T}(\mathbf{XX^T} + \boldsymbol{\delta}\mathbf{I})^{-\mathbf{1}}\mathbf{d}}$$

Online

Mean value:

$$w_k = \frac{(k-1)\cdot w_{k-1} + x_k}{k} = \frac{1}{k}x_k + \frac{k-1}{k}w_{k-1}$$

$$w_k = w_{k-1} + \frac{x_k - w_{k-1}}{k}$$

$$\mathbf{w_{new}} = \mathbf{w_{old}} + \boldsymbol{\Delta}\mathbf{w}$$

Variance:

$$\delta_w = \frac{x_k - w_{k-1}}{k}$$
$$w_k = w_{k-1} + \delta_w$$
$$\boldsymbol{\sigma_k^2 = \frac{(k-1)\sigma_{k-1}^2}{k} + \delta_w(x_k - w_k)}$$

Correlation Matrix:

$$\mathbf{R_{xx,n} = \frac{n-1}{n}R_{xx,n-1} + \frac{1}{n}x_n x_n^T}$$

Least Mean Square Error (LMS)

Consider a simple online adaptation scheme:

$\mathbf{w_i = w_{i-1} + \Delta w_i}, \quad \mathbf{i = 0, 1, \ldots ,} \quad$ **with initial condition** $\mathbf{w_{-1}}$

From optimization theory, the term Δw_i, assume the form:

$\mathbf{\Delta w_i} \backsim \left(-\nabla \mathbf{J(w_i)}\right)$

$\mathbf{J(w) \triangleq e_i^2 = (d_i - w^T x_i)^2 = d_i^2 - w^T x_i d_i - x_i^T w d_i + w^T x_i i_i^T w}$

The adaptation rule can be rewritten as:

$\mathbf{w_i = w_{i-1} + \mu\left\{-\frac{1}{2}\nabla J(w_i)\right\}}, \quad$ **for** $\mathbf{i = 0, 1, \ldots}$

Incorporating all constant values in μ:

$\mathbf{w_i = w_{i-1} + \mu \nabla J(w_i)}, \quad$ **for** $\mathbf{i = 0, 1, \ldots}$

The gradient ad index *i* assume the form:

$\mathbf{\nabla J(w) \triangleq \frac{\partial J(\mathit{w})}{\partial \mathit{w}} = -2(\mathit{d_i} - \mathit{w^T x_i})\mathit{x_i}}$

$\mathbf{\nabla J(w) = -\mathit{e_i x_i}}$

Finally, the **LMS** adaptation step:

$$\boxed{\mathbf{w_i = w_{i-1} + \mu \mathit{e_i x_i}}, \quad \textbf{for } \mathbf{i = 0, 1, \ldots\ N.}}$$

Lagrangian loss function

$$J_L(w, \lambda) = J(w) + \lambda[h(w) - b]$$

The solution can be found solving:

$$\underset{w \in \mathbb{R}^M,\ \lambda \in \mathbb{R}^M}{\arg\min}\ J_L(w, \lambda) \quad \text{i.e.} \quad \begin{cases} \nabla_w J_L(w, \lambda) \to 0 \\ \nabla_\lambda J_L(w, \lambda) \to 0 \end{cases}$$

That is:

$$\nabla_w J_L(w, \lambda) = \nabla J(w) + \lambda \nabla h(w) = 0$$
$$\nabla_\lambda J_L(w, \lambda) = h(w) - b = 0$$

Bayesian approach

$\theta_i \triangleq$ is an "event" (we can enumerate the possible events)

$x \triangleq$ are the "data" associated with the "event"

$$\Pr(\theta_i|x) \triangleq \frac{\Pr(x|\theta_i)\Pr(\theta_i)}{\sum_{k=1}^{N_i}\Pr(x|\theta_k)\Pr(\theta_k)}$$

$\Pr(\theta_i) \triangleq$ is the **"a priori" probability**
$\Pr(x|\theta_i) \triangleq$ is the **likelihood**
$\Pr(\theta_i|x) \triangleq$ is the **"a posterior" probability**
$\Pr(x) \triangleq$ is denoted as **model evidence** or **marginal likelihood**
Mnemotic formula:

$$Posterior\ probability = \frac{Likelihood \times Prior}{Evidence}$$

Bayes' Rule in Pattern Recognition
Given a features *vector x* that describe an input pattern, consider a *C-classes* classification problem the formula become:

$$\Pr(C_i|x) \triangleq \frac{\Pr(x|C_i)\Pr(C_i)}{\sum_{k=1}^{N_i}\Pr(x|C_k)\Pr(C_k)}$$

Maximum Likelihood Estimation (MLE)

We consider $p(\theta) = 1$, and the Bayes' rule simplify as:

$$p(\theta|x) \triangleq \frac{p(x|\theta)}{p(\theta)}$$

the parameters $\boldsymbol{\theta}$, can be estimated be a simple optimization algorithm:

$$\theta_{MLE} = \underset{\theta\in\mathbb{R}^M}{\arg\max}\{p(x;\theta)\}$$

This function, indicated as $\mathcal{L}(\theta)$ is called **likelihood function.**
Its logarithm $L(\theta) = \ln p(x;\theta)$ **log-likelihood function**, and its average as $L(\theta) = \ln p(x_i;\theta)$

Maximum A Posteriori Estimation (MAP)

In case that $\boldsymbol{\theta}$ is a RV of unknown parameters, but with **known pdf** the MLE can be extended considering the *joint probability*. For the Bayes' Rule maximizing $\mathbf{p}(\mathbf{x},\boldsymbol{\theta})$ is equivalent to maximizing $\mathbf{p}(\mathbf{x}|\boldsymbol{\theta})\mathbf{p}(\boldsymbol{\theta})$. So, we can write:

$$\theta_{MAP} = \underset{\theta\in\mathbb{R}^M}{\arg\max}\{p(x,\theta)\} = \underset{\theta\in\mathbb{R}^M}{\arg\max}\{p(x|\theta)p(\theta)\}$$

Unified view of MLE and MAP

In case that θ is a RV for the Bayes' rule: $p(x,\theta) = p(\theta|x)p(x) = p(x,\theta)p(\theta)$ we can write:

$$\theta_* = \underset{\theta\in\mathbb{R}^M}{\arg\max}\{p(x,\theta)\} = \underset{\theta\in\mathbb{R}^M}{\arg\max}\{p(\theta|x)\} = \underset{\theta\in\mathbb{R}^M}{\arg\max}\left\{\frac{p(x|\theta)p(\theta)}{p(x)}\right\}$$

Naïve Bayes classifier

If we have a *strong independence assumption* on $\mathbf{x}$, the likelihood function can be factorized as product:

$$p(\theta_i|x) = \prod_{k=1}^{N} p(x_k|\theta_i), \qquad \text{then,} \qquad p(x|\theta_i) = \frac{\left(\prod_{k=1}^{N} p(x_k|\theta_i)\right)p(\theta_i)}{p(x)}$$

So, the classifier can be derived as:

$$\theta_i \mathrel{\therefore}= \underset{i=[i,C]}{\arg\max}\left\{p(\theta_i)\prod_{k=1}^{N} p(x_k|\theta_i)\right\}$$

Or to avoid multiplications:

$$\theta_i \mathrel{\therefore}= \underset{i=[i,C]}{\arg\max}\left\{\ln p(\theta_i) + \sum_{k=1}^{N} \ln p(x_k|\theta_i)\right\}$$

The solution can be found as a simple optimization problem: $\theta_* \mathrel{\therefore}= \nabla L(\theta) \rightarrow \theta$

From a statistical point of view, the **LS solution is a MLE solution**

The Kullback-Leibler divergence or relative entropy

Given two *discrete probability distributions* $p(x_n)$ and $q(x_n)$, the **Kullback-Leibler's divergence** also denoted **Relative Entropy** is defined as:

$$D_{K,p||q} = \langle p(x), \ln\frac{p(x)}{q(x)}\rangle$$

$D_{K,p||q}$ can be used as Loss Function

$$D_{K,p||q} \simeq \frac{1}{N}\sum_{n=1}^{N}[-\ln q(x_n|w) + \ln p(x_n)]$$

We note that the term $p(x_n)$ does not depends on of w. Thus we minimize the following *likelihood function*:

$$D_{K,p||q} \simeq -\frac{1}{N}\sum_{n=1}^{N}[\ln q(x_n|w)]$$

Minimizing this Kullback-Leibler divergence is equivalent to maximizing a likelihood function.

Conclusions: Loss and Loss Functions for Training

1. Regression Loss Functions
   1. Mean Squared Error Loss
   2. Mean Squared Logarithmic Error Loss
   3. Mean Absolute Error Loss
2. Binary Classification Loss Functions
   1. Binary Cross-Entropy
   2. Hinge Loss (developed for SVD)
   3. Squared Hinge Loss
3. Multi-Class Classification Loss Functions
   1. Multi-Class Cross-Entropy Loss

2. Sparce Multiclass Cross-Entropy Loss
3. KulBack Leibler Divergence Loss

Activation Function

Sigmoid

$$\varphi(s) = \frac{e^{\alpha s}}{1 + e^{\alpha s}} = \frac{1}{1 + e^{-\alpha s}}, \qquad \text{where } s = w^T x$$

The **derivative** can be expressed as the output of the **function itself**:

$$\frac{d\varphi(s)}{ds} = \frac{d}{ds}\left(\frac{1}{1 + e^{-s}}\right) = \frac{1}{1 + e^{-s}} \cdot \left(1 - \frac{1}{1 + e^{-s}}\right) = \varphi(s)[1 - \varphi(s)] = \mathbf{y}[\mathbf{1} - \mathbf{y}]$$

In some case we want the output $\mathbf{y} \in \{-\mathbf{1}, +\mathbf{1}\}$, so the AF become:

$$\varphi(s) = \frac{2}{1 + e^{-2s}} - 1$$

With derivative:

$$\varphi'(s) = 1 - \varphi^2(s) = 1 - y^2$$

The Generalized Delta Rule

Considering the adaptation rule of the form:

$$w_{new} = w_{old} + \frac{1}{2}\mu[-\nabla J(w)]$$

where, as in the LMS and the CF is $\mathbf{J}(\mathbf{w}) = \mathbf{e}^2$ and $\mathbf{e} = \mathbf{d} - \boldsymbol{\varphi}(\mathbf{s})$ the gradient is:

$$\nabla J(w) = \frac{\partial J(w)}{\partial e} \cdot \frac{\partial e}{\partial s} \cdot \frac{\partial s}{\partial w} = -2e \cdot \varphi'(s) \cdot x$$

If $\boldsymbol{\varphi}(\mathbf{s})$ is the sigmoid function the gradient can be written as:

$$\nabla J(w) = -2e \cdot y(1 - y) \cdot x$$

And the *generalized delta rule* is:

$$w_k = w_{k-1} + \mu \cdot e[k] \cdot y[k] \cdot (1 - y[k]) \cdot x_k = w_{k-1} + \mu \cdot e[k] \cdot \delta_k \cdot x_k$$

Where the quantity "delta" is defined as $\boldsymbol{\delta}_\mathbf{k} = \mathbf{e}[\mathbf{k}] \cdot \mathbf{y}[\mathbf{k}] \cdot (\mathbf{1} - \mathbf{y}[\mathbf{k}])$ from which the name generalized delta-rule

MLE with Bernulli distribution

Bernulli distribution

PDF:

$$\Pr(y = 1|x; w) = p(x|w)$$
$$\Pr(y = 0|x; w) = 1 - p(x|w)$$

In compact form:

$$\Pr(y|x; w) = \mathcal{L}(w) = p(x|w)^y \cdot [1 - p(x|w)]^{1-y}$$

Maximize the likelihood function:

$$\mathcal{L}(\mathrm{w}) = \prod_{i=1}^{N} y_i^{d_i} \cdot (1 - y_i)^{1-d_i}$$

The log-likelihood is:

$$L(\mathrm{w}) = \ln[\mathcal{L}(\mathrm{w})] = \sum_{i=1}^{N} [d_i \ln y_i + (1 - d_i)\ln(1 - y_i)] = d \odot \ln y + (1 - d) \odot \ln(1 - y)$$

Thus, the adaptation rule can be written as:

$$\mathrm{w}_k = \mathrm{w}_{k-1} + \mu \nabla L(\mathrm{w}), \qquad L(\mathrm{w}) = d_i \ln y_i + (1 - d_i)\ln(1 - y_i), \qquad y_i = \varphi(s)$$

The gradient is:

$$\frac{\partial L(\mathrm{w})}{\partial \mathrm{w}} = \big(d_i(1 - \varphi(s)) - (1 - d_i)\varphi(s)\big)x_i = \big(d_i - \varphi(s)\big)x_i = e_i x_i$$

Finally, the adaptation rule is:

$$\mathrm{w}_k = \mathrm{w}_{k-1} + \mu e_i x_i$$

It does not depend on activation function

Hyperbolic tangent

Takes values from -1 to 1.

$$\varphi(s) = \frac{1 - e^{-\alpha s}}{1 + e^{-\alpha s}} = \tanh\left(\frac{\alpha s}{2}\right)$$

The derivative is:

$$\frac{d\varphi(s)}{ds} = \frac{\alpha}{2}[1 - \varphi^2(s)]$$

Rectified linear activation function (ReLU), Leaky ReLU, Soft Plus and Softmax

ReLU:

$$\varphi(s) = \max(0, s) = \begin{cases} s, & \text{if } s \geq 0 \\ 0, & \text{if } s < 0 \end{cases}$$

Leaky ReLU:

$$\varphi(s) = \max(s, \alpha s) = \begin{cases} s, & \text{if } s \geq 0 \\ \alpha s, & \text{if } s < 0 \end{cases}, \qquad \text{where } \alpha \ll 1$$

Soft-plus:

$$\varphi(s_i) = \log(1 + e^s), \qquad \text{where } \varphi'(s) = \frac{1}{1 + e^{-s}}$$

Soft-max:

$$\varphi(s_i) = \frac{e^{s_i}}{\sum_{k=1}^{Nclass} e^{s_k}}$$

Flexible Activation Functions

The activation function's shape is flexible and depends on some free parameters $\varphi(s) \rightarrow \varphi(s, \mathbf{q})$. So, in gradient-based learning the LF is $J(\mathrm{w}, \mathbf{q})$ and the gradient descent learning rule can be written as:

$$\mathrm{w_{new}} = \mathrm{w_{old}} + \mu_\mathrm{w}\left(-\nabla_\mathrm{w}\mathrm{J}(\mathrm{w},\mathrm{q})\right)$$
$$\mathrm{q_{new}} = \mathrm{q_{old}} + \mu_\mathrm{q}\left(-\nabla_\mathrm{q}\mathrm{J}(\mathrm{w},\mathrm{q})\right)$$

Adaptive Sigmoid Activation Function

Sigmoid (unipolar or bipolar) whit adaptive slope and gain: $\mathrm{q} = [\alpha, \mathrm{G}]^\mathrm{T}$

$$\varphi(s) = \frac{\mathrm{G}}{1 + \mathrm{e}^{-\alpha \mathrm{s}}}$$

Adaptive Polynomial (AP) Neural Networks

Let the error as $\mathbf{e}[\boldsymbol{n}] = \boldsymbol{d}[\boldsymbol{n}] - \boldsymbol{\varphi}(\boldsymbol{s})$, at each step, the updating of both the synaptic weights **w** and the coefficients of activation function **a** is performed as:

$$\mathrm{w_{n+1}} = \mathrm{w_n} + \Delta\mathrm{w_n}, \qquad \mathrm{a_{n+1}} = \mathrm{a_n} + \Delta\mathrm{a_n}$$

The loss function:

$$\nabla\hat{\mathrm{J}}(\mathrm{w},\mathrm{a}) = \begin{bmatrix} \nabla_\mathrm{a}\mathrm{e}^2[\mathrm{n}] \\ \nabla_\mathrm{w}\mathrm{e}^2[\mathrm{n}] \end{bmatrix}$$

With Delta-Rule we have:

$$\Delta\mathrm{w_n} = \frac{1}{2}\mu_\mathrm{w}\left[-\nabla_\mathrm{w}\hat{\mathrm{J}}(\mathrm{w},\mathrm{a})\right], \qquad \Delta\mathrm{a_n} = \frac{1}{2}\mu_\mathrm{a}\left[-\nabla_\mathrm{a}\hat{\mathrm{J}}(\mathrm{w},\mathrm{a})\right]$$

Proceeding with generalized Delta-Rule, finally we obtain:

$$\Delta\mathrm{w_n} = \mu_\mathrm{w}\mathrm{e}[\mathrm{n}]\frac{\partial\varphi(\cdot)}{\partial \mathrm{s}}\mathrm{x}, \qquad \Delta\mathrm{w_n} = \mu_\mathrm{w}\mathrm{e}[\mathrm{n}][1 \;\; \mathrm{s} \;\; \mathrm{s}^2 \;\; \dots \;\; \mathrm{s}^\mathrm{m}]^\mathrm{T}$$

Statistic vs Deterministic