Radial Basis Function Network (RBF Net)

- Linearly combined Gaussian bumps.
- \( F(X, W, U) = \sum_i u_i \exp(-k_i(X - W_i)^2) \)
- The centers of the bumps can be initialized with the K-means algorithm (see below), and subsequently adjusted with gradient descent.
- This is a good architecture for regression and function approximation.
classification (y is scalar and discrete). Let’s denote $E(y, X, W) = E_y(X, W)$

MAP/MLE Loss Function:

$$L(W) = \frac{1}{P} \sum_{i=1}^{P} [E_{y^i}(X^i, W) + \frac{1}{\beta} \log \sum_k \exp(-\beta E_k(X^i, W))]$$

This loss can be written as

$$L(W) = \frac{1}{P} \sum_{i=1}^{P} \left[ -\frac{1}{\beta} \log \frac{\exp(-\beta E_{y^i}(X^i, W))}{\sum_k \exp(-\beta E_k(X^i, W))} \right]$$
Cross-Entropy and KL-Divergence

- let’s denote \( P(j \mid X^i, W) = \frac{\exp(-\beta E_j(X^i,W))}{\sum_k \exp(-\beta E_k(X^i,W))} \), then

\[
L(W) = \frac{1}{P} \sum_{i=1}^{P} \frac{1}{\beta} \log \frac{1}{P(y^i \mid X^i, W)}
\]

\[
L(W) = \frac{1}{P} \sum_{i=1}^{P} \frac{1}{\beta} \sum_k D_k(y^i) \log \frac{D_k(y^i)}{P(k \mid X^i, W)}
\]

with \( D_k(y^i) = 1 \) iff \( k = y^i \), and 0 otherwise.

- example 1: \( D = (0, 0, 1, 0) \) and \( P(. \mid X^i, W) = (0.1, 0.1, 0.7, 0.1) \). with \( \beta = 1 \), \( L^i(W) = \log(1/0.7) = 0.3567 \)

- example 2: \( D = (0, 0, 1, 0) \) and \( P(. \mid X^i, W) = (0, 0, 1, 0) \). with \( \beta = 1 \), \( L^i(W) = \log(1/1) = 0 \)
Cross-Entropy and KL-Divergence

\[
L(W) = \frac{1}{P} \sum_{i=1}^{P} \frac{1}{\beta} \sum_{k} D_k(y^i) \log \frac{D_k(y^i)}{P(k|X^i, W)}
\]

- \(L(W)\) is proportional to the *cross-entropy* between the conditional distribution of \(y\) given by the machine \(P(k|X^i, W)\) and the *desired* distribution over classes for sample \(i\), \(D_k(y^i)\) (equal to 1 for the desired class, and 0 for the other classes).

- The cross-entropy also called *Kullback-Leibler divergence* between two distributions \(Q(k)\) and \(P(k)\) is defined as:

\[
\sum_{k} Q(k) \log \frac{Q(k)}{P(k)}
\]

- It measures a sort of dissimilarity between two distributions.

- the KL-divergence is not a distance, because it is not symmetric, and it does not satisfy the triangular inequality.
Multiclass Classification and KL-Divergence

- Assume that our discriminant module $F(X, W)$ produces a vector of energies, with one energy $E_k(X, W)$ for each class.
- A switch module selects the smallest $E_k$ to perform the classification.
- As shown above, the MAP/MLE loss below be seen as a KL-divergence between the desired distribution for $y$, and the distribution produced by the machine.

$$L(W) = \frac{1}{P} \sum_{i=1}^{P} [E_{y^i}(X^i, W) + \frac{1}{\beta} \log \sum_k \exp(-\beta E_k(X^i, W))]$$
Multiclass Classification and Softmax

- The previous machine: discriminant function with one output per class + switch, with MAP/MLE loss
- It is equivalent to the following machine: discriminant function with one output per class + softmax + switch + log loss

\[ L(W) = \frac{1}{P} \sum_{i=1}^{P} \frac{1}{\beta} - \log P(y^i | X, W) \]

with \( P(j | X^i, W) = \frac{\exp(-\beta E_j(X^i, W))}{\sum_k \exp(-\beta E_k(X^i, W))} \) (softmax of the \(-E_j's\)).

- Machines can be transformed into various equivalent forms to factorize the computation in advantageous ways.
Multiclass Classification with a Junk Category

- Sometimes, one of the categories is “none of the above”, how can we handle that?
- We add an extra energy wire $E_0$ for the “junk” category which does not depend on the input. $E_0$ can be a hand-chosen constant or can be equal to a trainable parameter (let’s call it $w_0$).
- everything else is the same.
NN-RBF Hybrids

- Sigmoid units are generally more appropriate for low-level feature extraction.
- Euclidean/RBF units are generally more appropriate for final classifications, particularly if there are many classes.
- Hybrid architecture for multiclass classification: sigmoids below, RBFs on top + softmax + log loss.
Reparameterizing the function by transforming the space

\[ E(Y, X, W) \rightarrow E(Y, X, G(U)) \]

- gradient descent in \( U \) space:
  \[ U \leftarrow U - \eta \frac{\partial G}{\partial U} \frac{\partial E(Y, X, W)}{\partial W} \]
- equivalent to the following algorithm in \( W \) space:
  \[ W \leftarrow W - \eta \frac{\partial G}{\partial U} \frac{\partial G}{\partial U} \frac{\partial E(Y, X, W)}{\partial W} \]
- dimensions: \( [N_w \times N_u][N_u \times N_w][N_w] \)
Parameter-Space Transforms: Weight Sharing

- A single parameter is replicated multiple times in a machine
- \( E(Y, X, w_1, \ldots, w_i, \ldots, w_j, \ldots) \rightarrow E(Y, X, w_1, \ldots, u_k, \ldots, u_k, \ldots) \)
- gradient: \( \frac{\partial E()}{\partial u_k} = \frac{\partial E()}{\partial w_i} + \frac{\partial E()}{\partial w_j} \)
- \( w_i \) and \( w_j \) are tied, or equivalently, \( u_k \) is shared between two locations.
Parameter Sharing between Replicas

- We have seen this before: a parameter controls several replicas of a machine.

\[ E(Y_1, Y_2, X, W) = E_1(Y_1, X, W) + E_1(Y_2, X, W) \]

- gradient:
  \[ \frac{\partial E(Y_1, Y_2, X, W)}{\partial W} = \frac{\partial E_1(Y_1, X, W)}{\partial W} + \frac{\partial E_1(Y_2, X, W)}{\partial W} \]

- \( W \) is shared between two (or more) instances of the machine: just sum up the gradient contributions from each instance.
Path Summation (Path Integral)

One variable influences the output through several others

- \( E(Y, X, W) = E(Y, F_1(X, W), F_2(X, W), F_3(X, W), V) \)

- gradient: \( \frac{\partial E(Y, X, W)}{\partial X} = \sum_i \frac{\partial E_i(Y, S_i, V)}{\partial S_i} \frac{\partial F_i(X, W)}{\partial X} \)

- gradient: \( \frac{\partial E(Y, X, W)}{\partial W} = \sum_i \frac{\partial E_i(Y, S_i, V)}{\partial S_i} \frac{\partial F_i(X, W)}{\partial W} \)

- there is no need to implement these rules explicitly. They come out naturally of the object-oriented implementation.
Mixtures of Experts

Sometimes, the function to be learned is consistent in restricted domains of the input space, but globally inconsistent. **Example: piecewise linearly separable function.**

- Solution: a machine composed of several “experts” that are specialized on subdomains of the input space.
- The output is a weighted combination of the outputs of each expert. The weights are produced by a “gater” network that identifies which subdomain the input vector is in.

\[
F(X, W) = \sum_k u_k F_k^k(X, W^k) \text{ with } u_k = \frac{\exp(-\beta G_k(X, W^0))}{\sum_k \exp(-\beta G_k(X, W^0))}
\]

- the expert weights \( u_k \) are obtained by softmax-ing the outputs of the gater.
- example: the two experts are linear regressors, the gater is a logistic regressor.
The input is a sequence of vectors $X_t$.

- **Simple idea:** the machine takes a time window as input
- $R = F(X_t, X_{t-1}, X_{t-2}, W)$
- **Examples of use:**
  - predict the next sample in a time series (e.g. stock market, water consumption)
  - predict the next character or word in a text
  - classify an intron/exon transition in a DNA sequence
Sequence Processing: Time-Delay Networks

One layer produces a sequence for the next layer: stacked time-delayed layers.

- layer1 \( X_t^1 = F^1(X_t, X_{t-1}, X_{t-2}, W^1) \)
- layer2 \( X_t^2 = F^1(X_t^1, X_{t-1}^1, X_{t-2}^1, W^2) \)
- cost \( E_t = C(X_t^1, Y_t) \)

Examples:
- predict the next sample in a time series with long-term memory (e.g. stock market, water consumption)
- recognize spoken words
- recognize gestures and handwritten characters on a pen computer.

How do we train?
Training a TDNN

Idea: isolate the minimal network that influences the energy at one particular time step $t$.

- in our example, this is influenced by 5 time steps on the input.
- train this network in isolation, taking those 5 time steps as the input.
- **Surprise**: we have three identical replicas of the first layer units that share the same weights.
- We know how to deal with that.
- do the regular backprop, and add up the contributions to the gradient from the 3 replicas
Convolutional Module

If the first layer is a set of linear units with sigmoids, we can view it as performing a sort of *multiple discrete convolutions* of the input sequence.

- 1D convolution operation:
  \[ S_t^1 = \sum_{j=1}^{T} W_{j}^1' X_{t-j}. \]
  
- \( w_{jk} \) \( j \in [1, T] \) is a convolution kernel

- sigmoid \( X_t^1 = \tanh(S_t^1) \)

- derivative: \( \frac{\partial E}{\partial w_{jk}^1} = \sum_{t=1}^{3} \frac{\partial E}{\partial S_t^1} X_{t-j} \)
Simple Recurrent Machines

The output of a machine is fed back to some of its inputs $Z$. $Z_{t+1} = F(X_t, Z_t, W)$, where $t$ is a time index. The input $X$ is not just a vector but a sequence of vectors $X_t$.

- This machine is a *dynamical system* with an internal state $Z_t$.
- Hidden Markov Models are a special case of recurrent machines where $F$ is linear.
To train a recurrent net: “unfold” it in time and turn it into a feed-forward net with as many layers as there are time steps in the input sequence.

An unfolded recurrent net is a very “deep” machine where all the layers are identical and share the same weights.

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E}{\partial Z_t} \frac{\partial F(X_t,Z_t,W)}{\partial W}$$

This method is called \textit{back-propagation through time}.

Examples of use: process control (steel mill, chemical plant, pollution control....), robot control, dynamical system modelling...