# AI Dictionary

(a work in progress)



## A

**Activation function –** Each neural unit on summing weighted inputs generally performs an activation function on this sum for output. Example include the **sigmoid function, rectified linear function** and the **softmax function.**

**AdaGrad Algorithm –** individually adapts the learning rates of all model parameters by scaling them inversely proportional to the square root of the cumulative sum of all of their squared gradients.

**Adam Optimization – (Adaptive moment estimation optimization)** Uses history updated first and second moments and uses the ratio of the first over the square root of the second to update the weights.

**Adversarial Training –** Constructing examples (adversarial examples) that can cause the network to misclassify that humans would correctly identify and add those to the training set.

**Artificial Neural Network (ANN) –** Generic name for a class of learning models roughly based on biological neural networks.

**Asynchronous Stochastic Gradient Decent (ASGD) –** Uses common parameters in either shared memory or using a parameter server to compute gradients. Resulting gradients are then vector summed, then applied.

**Automatic Speech Recognition (ASR) –** Using computers to take spoken speech and convert it to words. Current technology handles continuous speech from multiple speakers.

**Attention mechanism –** Generally a weighted average of a feature vector that may be used later as “memory” for a later time step. Used in ML

**Autoencoder –** Used to lower dimensionality. It trains on its input as output to train fewer hidden units to be reduced dimensionally. Such arrangements with fewer hidden units than inputs are called **undercomplete autocoders** and forces the hidden layer to capture the most salient features of the training data.

# B

**Back-Propagation (BP) –** Backward propagation of the propagation's output activations through the neural network using the training pattern target in order to generate the deltas (the difference between the targeted and actual output values) of all output and hidden neurons.
(From Wikipedia)
The following is a **stochastic gradient descent** algorithm for back propagation training a three-layer network (only one hidden layer):

```
Initialize network weights (often small random values)
do
  forEach training example named eg
    prediction = neural-net-output(network, eg)  // forward pass
    actual = teacher-output(ex)
    compute error (prediction - actual) at the output
    compute Δw_h for all weights from hidden layer to output layer  // backward pass
    compute Δw_i for all weights from input layer to hidden layer   // backward pass continued
    update network weights  // input layer not modified by error estimate
until all examples classified correctly or another stopping criterion satisfied
return the network.
```

**Back-Propagation Through Time (BPTT) –** Uses back propagation noting that hidden units $h(t)$ has descendants output $o(t)$ and hidden units $h(t+1)$.

**Bag of words –** A sparse vector where each entry indicates the presence or absence of a word from a vocabulary. Entries, other than 0 or 1, can indicate the number of instances of the the word in a document.

**Bagging – Bootstrap Aggregating:** A technique for reducing generalization error by combining several models.

**Batch Normalization –** Scale hidden layer output by subtracting the mean and dividing by std deviation from previous hidden layer states. This leads to stable hidden to hidden dynamics.

**Bayesian Statistics –** Calculation of data likelihood from prior data likelihoods using Bayes' rule:
$\mathbf{p}(Q|x^{(1)},...,x^{(m)})=\mathbf{p}(x^{(1)},...,x^{(m)}|Q)\mathbf{p}(Q)/\mathbf{p}(x^{(1)},...,x^{m)})$ where $x^{(i)}$ denoted sample i.

**Bias Importance Sampling –** Where the importance weights are normalized to sum to 1.

**Bias input –** A fixed input to a neural unit often 0 or 1.

**Binary activation function:** $f(x)=0,1$ for $x<0,x\geq 0$; $f'(x)=0,\infty$ for $x\neq 0,x=0$;

# C

**Cascade of Classifiers –** A set of classifier that can reject common non-matches early.

**Clustering –** There are several type of clustering algorithms including **Connectivity** based on connection distance, Centroid-based using **K-means algorithm** which is similar to **K-NC**, Distribution based using **Expectation-maximization algorithm,** and many others**. Canopy clustering** can be used on very large data sets to pre-cluster large, high dimensional data sets into more manageable subsets of lower dimensionality.

**Collaborative filtering –** Recommender systems based on similar product preferences of user extended to tastes for like products. Currently **ensemble models** are used. Usually using **ensemble models** including **RBM**s, bilinear prediction, and/or **SVD**.

**Computational Neuroscience –** An effort to understand how the brain works at an algorithmic level.

**Conjugate gradient Method –** Conjugate gradient $\beta_t = (g_t - g_{t-1})^T g_t / ((g_{t-1})^T g_{t-1})$ used to compute search direction $\rho_t = -g_t + \beta_t \rho_{t-1}$ to be applied to weights $\theta_{t+1} = \theta_t + \varepsilon^* \rho_t$ .
Note: $\varepsilon^*$ is the result of a search.

**Contractive autoencoder –** An **autoencoder** that is trained to resist perturbations of its inputs. Contracts the input neighborhood to a smaller output neighborhood.

**Contrast Normalization –** creating a new set of pixels by subtracting the pixel average from each pixel over a given area and normalizing by image contrast. See **Global contrast normalization** for rest of code.

```
// contrast normalization C-code snippet for picture pix(i,j,3)
// given # rows i, # colums j
int i, j, k;
double px_bar, psq, pmx, contrast;
px_bar= 0.0;
for(i=0;i<rows;i++) {
    for(j=0;j<cols;j++) {
        for(k=0;k<3;k++) { // colors
            px_bar+= px[i][j][k];
}}}
px_bar=px_bar/(3.0*rows*cols); // mean image intensity
psq= 0.0;
for(i=0;i<rows;i++) {
    for(j=0;j<cols;j++) {
        for(k=0;k<3;k++) {
            pmx= px[i][j][k]-px_bar;
            psq+= pmx*pmx;
}}}
contrast=sqrt(psq/(3.0*rows*cols)); // contrast of image
```

**Convolutional Neural networks - Output S(t)=(X*W)(t)** where **X** is the input and **W** is a probably density function. W is zero for all negative $x_i$. In two dimensions (for images) this can be in the form of a cross-correlation:
$S(i,j) = \sum_M \sum_N X(i+m, j+n)K(m, n)$ where $X$ is the Image and $K$ is the 2-d kernel.
Note that the range of S's indices can be less than $X$'s by M,N. If the same dimensionality is to be maintained then the values outside the perimeter of $X$ can be filled with zeros or values reflected from inside $X$'s perimeter.

The following C code demonstrates a convolutional network with output of the same dimensions as input by using zero fill outside the perimeter:

```
// C-code 2-D sharpen filter on a 2-D monochrome image
// stride=1, zero padding -- code snippet
#define NH 150  // # horizontal pixels
#define NV 100  // # vertical pixels
#define NK 3    // # kernel NKxNK NK always odd
int offset=NK/2
double newimage[NH][NV]; // sharpened image NHxNV
double image[NH][NV];    // normalized input image NHxNV
double sharpen[NK][NK]={-0.0313,-0.0625,-0.0323,-0.0625,0.9688,
-0.0625,-0.0313,-0.0625,-0.0313}; // called weights, kernel or filter
double pixsum;
int I.j.k.l.ik,jl;
for(i=0;i<NH;i++) {
    for(j=0;j<NV;j++) {
        pixsum=0.0;
        for(k=0;k<NK;k++) {
            for(l=0;l<NK;l++) [
                ik=i+k-offset; jl=j+l-offset;
                if(ik>=0 && jl>=0 && ik<NH && jl<NV) {
                    pixsum+=sharpen[k][l]*image[ik][jl];
                }
        }}
        newimage[i][j]=pixsum;
}}
```

**Cost function –** Typically the average squared loss over the training set or the data generating distribution. An example would be the sum or average of loss functions.

**Cross Entropy – The** average number of bits needed to identify an event drawn from the set, if a coding scheme is used that is optimized for an "unnatural" probability distribution q, rather than the "true" distribution p. The cross entropy between two discrete probability distributions p and q over the same set of events is: $H(p,q) = -\sum_X p(x)\log(q(x))$

**CUDA –** NVIDIA's C-like programming language for their GPUs. Another is **OpenCL** that is also C-like and works on most GPUs.
**Curriculum Learning (Shaping) –** Based on idea of beginning with simple concepts and progressing to more complex concepts.

# D

**Data parallelism –** Each input example run on different machines also known as **distributed inference**.

**Dataset augmentation –** This can be done by adding duplicated images to the training set with introduced noise or cropped in various ways.

**Decision Tree –** A tree-like graph that can be used to model decisions and their consequences or for classification based on data. Each node is a test and each branch is an outcome.

**DeepCL –** As elementary **deep learning** system using **OpenCL** using C++ or Python.

**Deep Learning –** Refers to learning, built on a hierarchy of concepts forming a many layer network

**Deep neural networks (DNN) –** Any of a variety of neural networks with many layers.

**Denoising Autoencoders (DAE) –** A denoising **autoencoder** is an autoencoder that on receiving corrupted data as input and is trained to predict the original uncorrupted data.

**Distributed Representation –** A representation of an object that has multiple parameters such a red 9-passenger SUV vehicle with California license plates.

**Dropout –** Dropping non-performing hidden units from an ensemble. Dropout is sometimes a useful **regularization** strategy.

**Dynamic Structure –** Sometimes called **conditional computation** refers to reconfigurable neural net structure that can determine from its input what configuration to use.

# E

**Early Stopping –** Techniques to stop training iterations early in order to avoid **overfitting**.

**Eigenvalues –** Roots to a system of linear equations and equivalent to solving for the roots $\lambda$ of the determinate $|M-\lambda I|=0$ where $M$ is a square matrix and $I$ is the identity matrix.

**Eigenvalue decomposition –** Let **A** be an NxN square matrix of linearly independent eigenvectors $q_i$ $(i=1,...,N)$ then **A** can be factorized as **A=Q Q$^{-1}$** where the $i^{th}$ column of **Q** is $q_i$ and is a diagonal matrix of the corresponding eigenvalues. Note: **A=Q Q$^{-1}$** is **AQ=Q** .

**Eigenvector –** A vector $v$ satisfying $Mv = \lambda v$ where $M$ is a square matrix and $\lambda$ is an eigenvalue of $M$.

**Echo State Network (ESN) –** In RCNs the recurrent weights are difficult to learn, but since they already capture the history well, just only learn the output weights.

**Ensemble model –** Composed of several different analytical models then synthesizing the results into a more accurate prediction.

# F

**Forget Gate –** Allows forgetting of an old state in a cell with memory that is judged no longer relevant.

# G

**Gabor filters –** Used for image processing to extract text when pictures are also present in documents. Also used for character recognition, finger print recognition or as used by our biological vision as a first level feature extractor as in the human V1 layer of the visual cortex.

$S(I)= \sum_{x\in\mathbb{X}}\sum_{y\in\mathbb{Y}}[w(x,y)I(x,y)]$ where the kernel W(x,y) has weights that are described by the Gabor function in each sample space $\mathbb{X}$ and $\mathbb{Y}$:

$w(x, y, \alpha, \beta_x, \beta_y, \lambda, \phi, x_0, y_0, \tau) = \alpha \exp(-\beta_x x'^2 - \beta_y y'^2)\cos(\lambda x' + \phi)$,

where: $x' = (x-x_0)\cos(\tau) + (y-y_0)\sin(\tau)$

$y' = -(x-x_0)\sin(\tau) + (y-y_0)\cos(\tau)$

$x_0, y_0, \tau$ define the coord system, $\alpha \exp(-\beta_x x'^2 - \beta_y y'^2)$ is a Gaussian factor where $\alpha$ is for scaling and $\beta_x$ and $\beta_y$ control drop-off from origin.

The cosine factor $\cos(\lambda x' + \phi)$ controls brightness with parameters $\lambda$ and $\phi$ controlling frequency and phase offset respectively.

```
// C-code for Gabor function w
double w(double x, double y, double alpha, double betax, double
betay, double lambda, double phi, double x0, double y0, double tau) {
     double xp=  (x-x0)*cos(tau)+(y-y0)*sin(tau);
     double yp= -(x-x0)*sin(tau)+(y-y0)*cos(tau);
     return alpha*exp(-betax*xp*xp-betay*yp*yp)*cos(lambda*xp+phi);
}
```

**Gater –** A selector of which expert network or classifier to use based on a given example. If a single expert is selected for each input then that is called a **hard mixture of experts**.

**Gated RNNs –** A gated **RNN** has connection weights that may change with each time step. For example, if an end of sequence is detected, past information may not be relevant.

**Gated Recurrent Unit –** This unit has two gates: a **reset gate** that determines how much new input to combine with previous memory and an **update gate** that determines how much previous memory to keep around. Now widely used in **LSTM**s it combines a forget and input gate into a single update gate and merged the cell state with the hidden state. [a,b] denoted vector concatenation. Input vector is $x_t$, output value is $h_t$. $z_t$ is the update value and $r_t$ is the reset value, both between 0 and 1.

$z_t = \sigma(W_z \bullet [h_{t-1}, x_t])$ $r_t = \sigma(W_r \bullet [h_{t-1}, x_t])$ $s_t = \tanh(W \bullet [r_t * h_{t-1}, x_t])$

$h_t = (1-z_t) * h_{t-1} + z_t * s_t$

```
// c-code snippet
// assume X is width N so Wz, Wr and W width are N+1
// assume global h initialized to 0 at beginning of run
```

```
double z,r,s,tempz,tempr,temps,temph;
tempz=Wz[n]*h;
tempr=Wr[n]*h;
for (i=0;i<n;i++) {
    tempz+=Wz[i]*x[i];
    tempr+=Wr[i]*x[i];
}
z=act_sig(tempz);
r=act_sig(tempr);
temps=W[n]*r*h;
for (I=0;i<n;i++) {
    temps+=W[i]*x[i];
}
s=act_tanh(temps);
h=(1.0-z)*h+z*s; // now have output and h for next use feedback.
```

**Gausian distribution –** See **Normal distribution**

**Gaussian mixture model (GMM) –** A Probabilistic model expressing the presence of subpopulations within a larger population. They are used to make statistical inferences about a subpopulation given the properties of the overall population. **GMM-HMM** systems were used in early **ASR** systems.

**Global Contrast Normalization (GCN) –** See **contrast normalization** for preceding code.

```
// C-code snippet:
contrast=(contrast>0.1)?contrast:0.1; // to avoid divide by zero
for(i=0;i<rows;i++) {
    for(j=0;j<cols;j++) {
        for(k=0;k<3;k++) { //colors
            px_new[i][j][k]= px[i][j][k]/contrast);
}}}
```

Similar code can be used for local contrast normalization for part of the image that may bring out features like edges and corners. Factor

**General Artificial Intelligence (AGI) –** Refers to Human level machine intelligence.

**Generalization error –** The error between training and test performance on new data sets. This can be due to many factors including insufficient training data, overtraining, or needing hyperparameter changes or a different model. A brute force way to decrease generalization error is to increase model capacity and training set size.

**Generative Models –** Networks that generate sound, images or actions.

**Gradient clipping –** Bounding gradients to keep steep cliffs from throwing off calculations. A reasonable way to do this is: **if ||g||>v** where **g** is the gradient and **v** is the bound of the new gradient **g' = gv / ||g||**

```
// c-code snippet:
```

```
double v=10.0; // some positive reasonable value
g = (fabs(g) <= v)? g : g*v/fabs(g);
```

**Gradient Descent –** Refers to the process of computing a gradient that is used to alter weights (coefficients) to find a minima of a function in a many dimensional space. Consider a multi-variable function $f(x)$ taking the partial derivative in all dimensions we have the gradient $\nabla f(x)$ using the negative gradient we can find the local minimum iteratively by getting successive values of $x_0, x_1, x_2, \ldots$ such that $x_{n+1} = x_n - \varepsilon_n \nabla f(x_n)$ and $\varepsilon_n$ is the step size which can also change on each step. A useful formula for calculating the step size is the following if $f(x)$ is convex: $\varepsilon_n = \frac{(X_n - X_{n-1})^T \cdot |\nabla(f(X_n)) - \nabla(f(X_{n-1}))|}{\|\nabla(f(X_n)) - \nabla(f(X_{n-1}))\|^2}$

**Greedy Algorithms –** Break a problem into many components and solve for the optimal version for each component in isolation.Usually applied to lower layers of a neural network as layer-wise unsupervised pretraining. Generally proceeds one layer at a time holding previous layers fixed.

**Greedy Supervised Pretraining (GSP) –** Pretraining algorithms that break supervised learning problems into simpler supervised learning problems.

# H

**Hadamard product –** Given tensors (from vectors to n-dimentional matrices of the same size) **A** and **B** a Hadamard product is denoted **A∘B** and means each element of **A** is multiplied by each element of **B** resulting in each respective element of **C** of the same dimensions. eg. $\mathbf{C}_{ij}=\mathbf{A}_{ij}x\mathbf{B}_{ij}$

```
// C-code Hadamard 2-D snippet
int nx=100, ny=50;
double A[100][50], B[100][50];
for(i=0;i<nx;I++) {
    for(j=0;j<ny;j++) {
        C[i][j]=A[i][j]*B[i][j];
    }
}
```

**Hard tanh activation function:** f (x)=max(-1,min(1,x)); f '(x)= 0,1,0 for x<-1, -1≤x≤1, x>1 respectively

```
// C-code functions:
double act_tramp(x) { return max(-1,min(1.0,x); }
double dact_tramp(x) {
     if (x<-1.0 || X>1.0) { return 0.0; }
     return 1.0;
}
```

**Helmholtz machine –** A generative model (developed by Hinton et.al.) which has inspired **variational autoencoder** and **Generative stochastic networks**.

**Hessian Matrix –** $H(f)(x)_{i,j}=$ second derivatives of function $f$ with respect to $x_i$ and $x_j$ which forms a symmetric matrix. Determines the curvature of the gradient function.

**Hidden Units –** Units that are neither input units nor output units but are neural units between them.

**Hidden Markov Models (HMM) –** Have been used for speech, gesture and handwriting recognition. Dynamic programming (Viterbi) algorithms are used to develop the probabilities of the transition and emission matrices. More recently being used in conjunction with artificial neural networks which in some cases replace them entirely.

**Hyper-parameters –** Learning rates, number of hidden units, Convolution kernel width, implicit padding, weight decay coefficient, dropout rate are all considered hyperparameters.

# I

**Independent Component Analysis (ICA) –** Separating observed data into independent linear factors that can be scaled and added together to recreate the observed data.

**Importance sampling –** General technique for estimating properties of a particular distribution having samples from another.

**Input Unit –** An input directly from a sensor or of a unit that conditions that input for use in a following neural network. It's important that inputs to a given neural network be of comparable magnitude range so an input unit might perform an operation such as **ax+b** where **x** is the input, **a** establishes the span of the range and **b** shifts it to the appropriate range for the following neural network. A common example is a neural network that takes an input range of [-1,1] but your sensor has a range of [0,1], **a**=2.0 and **b**=-1will give the desired result.

# J

**Jacobian –** Both the Jacobian matrix (if square) and its determinant are referred to as a Jacobian.

**Jacobian matrix –** Is the matrix of all first-order partial derivatives of a vector valued function.

# K

**K-means Clustering –** Aim is to cluster n observation vectors $(x_1,..,x_n)$ into k clusters ($k \leq n$) $S=(S_1,...,S_k)$ to minimize $\sum_i \sum_{x \varepsilon Si} ||x-\mu_i||^2$ where $\mu_i$ is the mean of points in $S_i$. The **k-means algorithm** (also called Lloyd's algorithm)

Step 1: initialization. Randomly assign k data points to k sets $S_i$ and call them means $\mu_i$.

Step 2: Assign each point $X_p$ to a cluster $S_i^{(t)}$ on the basis $||X_p-\mu_i^{(t)}||^2 \leq ||X_p-\mu_j^{(t)}||^2$ for for j =1,...,k.

Step 3: Compute new means $\mu_i^{(t+1)} = (\sum_{Xj \varepsilon Si} X_j)/|S_i^{(t)}|$ for i=1,...,k.

Step 4: Go to step 2 if appreciable change in minimizing $\sum_i \sum_{x \varepsilon Si} ||x-\mu_i||^2$ else done.

**K-neighbor classifier (K-NC)** also called **K-nearest neighbor algorithm (K-NN) -** Suppose a set of pairs (X,Y) with vector X in $R^d$ and Y a class label. A new X can then be classified by the majority label within the closest distance to point X. It can be majority rules for classification. The distance measure can be other than Euclidean distance. People have used Hamming distance for text data and correlation coefficients for gene expression data.

# L

**Laplace distribution – Laplace(x,u,b)=(1/2b)exp(-|x-u|/b)**

```
// C-code function:
double laplace(x,u,b) { return (1.0/2.0*b)*exp(-fabs(x-u)/b); }
```

**Layer Normalization –** See **batch normalization**

**Leaky Units –** Units with recurrent connections with gradients that diminish exponentially with T/d, instead of just T, where T is the time interval and d is the delay.

**Learning Rate –** A hyperparameter, often using the symbol λ or ε, used as a multiplier (0,1] of the gradient to update weight parameters.

**Linear regression –** This is a linear model using linear regression, normally using least-squares fitting. $y_i=\beta_0 + \beta_1 X_i + \varepsilon_i$, for i=1,...,n data points. This can easily be extended for higher powers of x. Model has numerical input and output. If multiple independent variables are involved, then it is called **multiple-regression.** The simple linear regression solution for $\beta_0$ and $\beta_1$ is: $\beta_1 = \sum_i ((x_i-\bar{x})(y_i-\bar{y}))/\sum_i (x_i-\bar{x})^2 \quad \beta_0 = \bar{y}-\beta_1 \bar{x} \quad \sigma^2=\sum_i(\varepsilon_i{}^2)/n$

```
// c-code
// assuming the following defined and set: n=number of samples,
// samples x[i],y[i] i=0,...,n-1
double sumx=0.0;
double sumy=0.0;
double x_bar,y_bar;
for (i=0; i<n; i++) {
    sumx+= x[i];
    sumy+= y[i];
}
x_bar=sumx/(double)n;
y_bar=sumy/(double)n;
double sumxy=0.0;
double sumxx=0.0;
for (i=0; i<n; i++) {
    sumxy+= (x[i]-x_bar)*(y[i]-y_bar);
    sumxx+= (x[i]-x_bar)*(x[i]-x_bar);
}
double beta1= sumxy/sumxx;     // beta coeffients
double beta0= y_bar-b1*x_bar;
Double rho2=0.0;
for (I=0;i<n;i++) {
    rho2+=pow(y[i]-beta0-beta1*x[i],2);
}
rho2=rho2/(n-1);               // common variance
```

**Logistic Regression –** Similar to linear regression but target variable is categorical.

**Logistic Sigmoid activation function:**

$$\sigma(x)=1/(1+\exp(-x)); \quad \sigma'(x)=\sigma(x)(1-\sigma(x))$$

```
// C-code functions:
double act_sig(x) { return 1.0/(1.0+exp(-x); }
double dact_sig(x) { return act_sig(x)*(1.0-act_sig(x)); }
```

**Loss function –** A function of goal versus actual data points. Typically the sum of squared errors or log of same.

**Long Short-Term Memory (LSTM) –** This unit has a memory cell and contains three gates, an **input gate**, a **forget gate** and an **output gate**. The input gate defines how much of the newly computed state for the current input you want to let through. The forget gate defines how much of the previous state you want to let through. Finally, The output gate defines how much of the internal state you want to expose to the external network.

# M

**Machine Learning –** Describes an AI system that can acquire its own knowledge.

**Maxout Unit –** The activation function is the max of the inputs.

**Max Pooling –** A layer where each unit picks the maximum input for output. Usually used for dimension reduction following a convolution layer.

```
// C-code 2-D MaxPool monochrome image code snippet
#define NH 150
// horizontal pixels
#define NV 100
// vertical pixels
#define NK 3
// maxpool window, note stride=2 in i,j indices
double newimage[NH/2][NV/2];
// incoming image NHxNV
double image[NH][NV];
double pmax;
int i,j,k,l,ik,jl;
for(i=0;i<NH-NK+1;i=i+2) {
  for(j=0;j<NV-NK+1;j=j+2) {
      pmax=0.0;
      for(k=0;k<NK;k++) {
        ik+i+k;
        for(l=0;l<NK;l++) {
            jl=j+l;
            pmax=(pmax>image[ik][jl])?pmax:image[ik][jl];
      }}
      newimage[i/2][j/2]=pmax;
}}
```

**Minibatch Algorithms –** Uses averaged gradients from small subsets of input training examples used for training. Often makes training faster.

**Mel-Frequency Cepstrum (MFC) –** Short term power spectrum of sound using the mel scale of frequency. Recently **DNN**s have been used successfully to replace the Speech front-end processing obviating the need for Fourier transforms, logs and discrete cosine transforms necessary to compute the **MFCC**s.

**Mel Frequency Cepstral Coefficients (MFCC) –** The Coefficients that make up the **MFC**.

**Maximum Likelihood Estimation (MLE) –** $\mathbf{L}(\theta_{1,}...,\theta_m) = \prod_{i=1:n} \mathbf{f}(x_i;\theta_1,...,\theta_m)$ is the likelihood function of vector of unknown parameters $\theta$ expressed as the joint likelihood

(probability) of each $\mathbf{f}(x_i,\theta_1,\ldots,\theta_m)$. Given **probability density function f**, one needs to find the value of $\theta$ that maximizes $\mathbf{L}(\theta_1,\ldots,\theta_m)$ knowing the random samples $x_i$.

**Multi-Layer Perceptron (MLP) –** A vague term that originally referred to artificial binary classification neural nets with one hidden layer using a Heaviside step activation function (**binary activation function**) trained by back propagation. Often, the term is applied to more general neural nets with a variety of activation functions.

**Model Identifiability –** A model is said to be identifiable if a sufficiently large training set can rule out all but one setting of the model's parameters.

**Model parallelism –** Each machine/core works on a different part of the same training example.

**Momentum –** In physics mass times velocity; here assume mass=$\alpha$ then the **velocity** is used to update the weights. The velocity $V_{t-1}$ (the prior weight change) is multiplied by $\alpha$ a hyperparameter <1 with the learning rate $r$ times the gradient of the cost or error function $J(\theta)$ subtracted computes the new velocity which is the new change in weights $\theta$.

In vector form $V_t = \alpha V_{t-1} - r\nabla_\theta J(\theta)$ then $\theta_t = \theta_{t-1} - V_t$

Element-wise we have: $v_k(t) = \alpha v_k(t-1) - r\partial J(\theta)/\partial w_k$ where $\alpha$ and $r$ are momentum and learning rate hyperparameters respectively.

# N

**N-gram –** an n-gram is a continuous series of items or tokens. These items can be phonemes, syllables, letters, words, etc. Unigram (1 token), bigram (2 tokens), trigram (3 tokens), ...

**Naive Bayes classifier (NB) –** Simple probabilistic classifiers based on Bayes' theorem with strong (naive) independence assumptions between features. Three probability distributions widely used include Gaussian (GNB), Multinomial (MNB) and Bernoulli (BNB). The decision rule follows:

$c = \max_k [(p(C_k))\prod_{i=1:n} p(x_i|C_k)]$ where $c$ is the label of the selected maximum category of $C_c$ probability times the product or the conditional probabilities over the feature space **X** for category $C_c$.

**Nesterov momentum –** also called Nesterov accelerated gradient descent

$v_{t+1} = \mu v_t - r\nabla f(\theta_t + \mu v_t)$ $\theta_{t+1} = \theta_t + v_{t+1}$ where $\mu$ in $[0,1]$ is the momentum coefficient, $v_t$ is momentum at step or time t, $\theta_t$ weigh at t, and $r>0$ is the learning rate.

$\nabla f(\theta_t)$ is the standard gradient for classical momentum whereas $\nabla f(\theta_t + \mu v_t)$ is the accelerated **gradient**.

**Neural Unit –** Any element containing multiple inputs times weighting factors, summed and passed through an **activation function** creating an output.

**Newton's Method –** Compute the **gradient** and the **Hessian** of the Loss function then increment the weights by minus the Hessian inverse times the gradient.

**Natural Language Processing (NLP) –** Includes machine translation from one language to another.

**Neural Net (NN) –** Any network of neural units (units with a weighted sum and an **activation function**.

**Normal Distribution –** Also known as **Gaussian distribution** (and sometimes as the bell curve) is a very common probability distribution described by:

$N(x; u,\sigma^2) = (1/\text{sqrt}(2\pi\sigma^2))\exp(-(x-u)^2/2\sigma^2)$ where $u$ is the mean and $\sigma$ is the standard deviation.

```
// C-code function
pi=3.14159265359
double norm(x,mu,rho) {
    double r2=rho*rho;
    double xu=x-u;
    return (1.0/sqrt(2.0*pi*r2)*exp(-xu*xu/(2.0*r2))
}
```

**NP-complete (NP-C) –** Nondeterministic Polynomial time Complete: Refers to a class of problems that have no known algorithmic solution in polynomial time and are reducible to a know NP-C problem like the Traveling Salesman problem..

# O

**Objective function –** Any function that you optimize during training. Although it could be a loss or cost function there are other cases such as MLE

**One-hot Code Vector –** A vector where only one element is 1 and the rest of the vector is 0. The vector is usually represented as a ordered pair (1el#,1) or (el#,0)

**OpenCL –** An open-source programming language for GPUs

**Optimization Techniques –** Includes **Adam, SGD, AdaGrad, RMSProp, Newton's Method**

**Output unit –** The output unit can be a standard neural cell but the output may have to be conditioned. For example, you may want a 0 or 1 output to turn on or off a device. This can be done buy seeing if the output is over a given threshold for 1. In training you generally need to take the preconditioned result for feedback. If only one output can be valid at a time then the highest output might be taken as 1 or true output and the others are taken as 0 or false.

**Overfitting –** A model that overfits the test data does not generalize well. When the gap between the training error and test error is too small.

# P

**Principle Components Analysis (PCA) –** A means of learning an orthogonal , linear transformation of the original input so individual elements are mutually uncorrelated. It can be done by **eigenvalue decomposition** of a data covariance (correlation) matrix or by **singular value decomposition** of a data matrix.

**Probability Density Function (PDF) –** corresponds to a specific probability density of an interval of real numbers. Common examples include uniform distributions and Gaussian distributions.

**Perplexity –** Perplexity is a measure of how well a probability distribution predicts a sample. A low perplexity indicated the distribution is a good predictor of the sample.

**Polyak Averaging –** Take the average of several points on the weight trajectory $\theta^{(1)}, \ldots, \theta^{(t)}$ works on convex problems. Use an exponentially decaying running average on non-convex problems

**Pooling –** Usually used by convolutional neural networks to reduce the network dimensions to the next layer. The pooling layer has no hyperparameters and often select the value to be output using the max function. See **Max Pooling**..

**Preprocessing –** processing input variables so that they are in the same range, This may also include global and/or local **contrast normalization (GCN & LCN**). Usually image pixels

are in the range [0,1] or [-1,1]. All input should be scaled to the same range

**Predictive Sparse Decomposition (PSD) –** A model incorporating **Sparse encoding** and **parametric autoencoders.**

**Pylearn2** – A Python machine learning library that specifies its algorithms in terms of calls to **Theano** and **cuda-convnet**

## Q

# R

**Random Forests –** Also called random decision forests is an **ensemble model** for regression or classiification.

**Ramp function –** See: **rectified linear activation function.**

**Ranking Loss:** $L=sum_i(max(0,1-a_y+a_i))$

**Restricted Boltzman Machines (RBM) –** A generative stochastic artificial neural network that can learn a probability distribution over its set of inputs.

**Recommender System:** A system to make recommendations to potential users or Customers. Includes two types, online advertising and item recommendations. The methodology used is generally **collaborative filtering**.

**Rectified Linear activation function (ReLU) –** Also known as the **ramp function** $f(x)=(x<0)?0:x;$ and the derivative $f'(x)=(x<0)?0:1$ also known as the **step function.** A zero dervative for X<0 can cause some ReLUs to become deactivated and permanently dead during backpropagation training.

```
// C-code functions:
double act_lin(x) { return (x<0.0)?0.0:x;}
double dact_lin(x) { return (x<0.0)?0.0:1; }
```

**Regularizing –** Getting NNs to perform better by constraining input elements, activations and/or other parameters. Often a regularization penalty is added to the objective function $J(\theta;X,y)$: $\underline{J}(\theta;X,y) = J(\theta;X,y) + \lambda\Omega(\theta)$ Where $\Omega(\theta)$ Is the regularization function on parameters $\theta$. An example would be $\Omega(\theta) = ||w||_1 = \sum_i|w_i|$ where **w** is the weights.

**Regularizing an autoencoder –** One way to regularize and autoencoder is to add a training penalty **Ω** as in **sparse encoders**. This one is geared to capture information about the training distribution:

$L(x,g(f(x))) + \Omega(h)$ where $\Omega(h)=\lambda \sum_i||\nabla_x h_i||^2$ and $\lambda$ is a **hyperparameter**.

**Reinforcement Learning –** Learning a task by trial and error with reward and/or punishment.

**Recurrent Neural Network (RNN) –** Each unit contain memory that accumulates the prior time steps. The output reflects not only the current stimulus but a portion of recent history. Each state vector $S_t = f(US_t + WS_{t-1})$ is an activation function $f$ of weighted $U$ sum of the inputs $X$ at time t plus the weighted $W$ sum of the states $S$ at time t-1.

**Recursive networks –** Recursive Units are like recurrent units but are arranged in a

hierarchical manner over data instead of being presented with a data set on each instance of time where time order is relevant.

**Residual networks –** In a residual network a  layers fit a residual mapping H(x) by having the next layer fit H(x) + x layer

**RMSProp –** Modifies AdaGrad to exponentially decay by the cumulative sum of the gradients squared. $E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$ and $\theta_{t+1} = \eta g_t / sqrt(E[g^2]_t + \varepsilon)$ where $\eta$ is the learning rate (usually 0.001) and $\varepsilon$ is a smoothing term that avoids a divide by zero.

# S

**Slow Feature Analysis (SFA) –** Is a linear factor model that uses information from time varying data to extract invariant features.

**Softplus activation function** $\zeta(x)=\ln(1+\exp(x));\ \zeta'(x)=\sigma(x)$ (the sigmoid function)

```
// C-code functions:
double act_lnexp(x) { return log(1.0+exp(x)); }
double dact_lnexp(x) { return 1.0/(1.0+exp(-x)); }
```

**Softmax activation function –** or normalized exponential function which spreads the results as a set of probabilities totaling 1: $P(Y=j|x) = \exp(x^T w_j)/\sum_k \exp(x^T w_k)$ the index of the highest inputXweight probability is the category. Often used as a last stage of a neural net for category output.

**Speech Recognition –** Machine recognition of acoustic signal into a string of spoken words. Typically the input data is in 20ms spectral samples often using the MEL-frequency Ceptrum Coefficients (**MFCC**s).

**Step activation function –** See **binary activation function.** In mathematics known as the Heaviside step function. It is discontinuous having the Dirac delta function at zero as a derivative.

**Stochastic Gradient Ascent (SGA) –** A variant of **SGD** used to maximize a criterion.

**Stochastic Gradient Descent (SGD) –** Behind most deep learning is SGD. A gradient is estimated for a minibatch of examples, multiplied by the learning rate and applied to model parameters (weights).

**Sigmoid activation function –** See **logistic sigmoid activation function.**

**Sparse autoencoders –** An autoencoder that that involves a sparsity penalty **Ω(h)** while training added to the loss function:

**L(x,g(f(x))) +** **Ω(h)** where **Ω(h)=**$\lambda\sum_i|h_i|$ and $\lambda$ is a hyperparameter.

**Sparse coding –** Sparse coding provides a class of algorithms for finding a succinct but unlike **PCA** obtains an **over-complete** set of basis vectors. Given only unlabeled input data, it discovers basis functions that capture higher-level features in the data. It accomplishes this by minimizing a sparse coding cost function using a sparse penalty. We desire that an input vector **X** be a linear combination of k basis vectors Ø where k>n (n=complete set size):

$\mathbf{x}=\sum_{i=1..k} a_i \phi_i$ so need minimum over $a_i^{(j)}, \phi_i$ for $\sum_j ||x^{(j)} - \sum_i a_i^{(j)} \phi_i|| + \lambda \sum_i S(a_i^{(j)})$ where $S(a_i)$ is frequently $|a_i|$ or $\log(1+a_i^2)$. i=1..k dims, j=1..m data sets, $\lambda$ scaling **hyperparameter**.

**Sparse connectivity – (a.k.a. sparce weights)** When an output from a given neuron in one layer (or input unit) goes to a subset of neurons in the next higher layer. Usually done in **convolutional networks**.

**Spectral radius –** The spectral radius of **Jacobian J(t)** is defined to be the maximum of the absolute values of its **eigenvalues.**

**Strong AI –** Refers to general artificial intelligence or a machine's intellectual capability functionally equivalent to a human's.

**Student-t distribution –** Used to estimate the mean of a normally distributed population when the sample size is small and the standard deviation is unknown.

$$tdist(t,v) = \frac{\Gamma((v+1)/2)}{\sqrt{(\pi v)}\,\Gamma(v/2)} (1+t^2/v)^{-(v+1)/2}$$

It is easier to compute the natural log of gamma(x) than gamma(x) so tdist(t,v) becomes:

```
// C code for tdist(t,v)
double tdist(t,v) {
    double part1,part2;
    double pi=3.141592653589793;
    part1=exp(gammln((v+1)/2.0)-(log(sqrt(v*pi))+ gammln(v/2.0)));
    part2=exp(-(v+1)/2*log(1+t*t/v));
    return part1*part2;
}
// C-code for gammln(x) from "Numerical Recipes in C"
double gammln(double x){ // returns natural log of gamma of x
    double xx,y,tmp.ser;
    static double cof[6]={76.18009172947146,-86.50532032941677,
        24.01409824083091,-1.231739572450155,
        0.1208650973866179e-2,-0.5395239384953e-5};
    int j;
    y=xx=x;
    tmp-= (xx+.05)*log(tmp);
    ser=1.000000000190015;
    for (j=0;j<=5;j++) { ser+=cof[j]/++y; }
    return -tmp+log(2.5066282746310005*ser/xx;
}
```

**Supervised Learning –** Each example in a dataset has a label or target so we create a model to that can learn to satisfy the input/output requirements.

**Singular Value Decomposition (SVD)** – **M** (mxn) is factorization of the form **M=U∑V*** where **U** (mxm) and **V** (nxn) unitary matrices (* superscript indicates conjugate) . ∑ (mxn) is a diagonal positive definite matrix. Diagonal entries of ∑ are the square roots of the non-zero **eigenvalues** of **M*M** and **MM***.

**Support Vector Machine (SVM)** – Supervised learning using linear regression on features that have been modified by a kernel function.

# T

**Tanh activation function:** $f(x)=\tanh(x)$; $f'(x)=1-\tanh(x)^2$

```
// C-code functions:
double act_tanh(x) { return tanh(x); }
double dact_tanh(x) {
     double tanhx=tanh(x);
     return 1.0-tanhx*tanhx;
}
```

**Time Delay Neural Network (TDNN) –** A neural net architecture designed to work on sequential data. An example: continuous speech converted to a phoneme stream.

**Teacher Forcing** – Usually used in **RNN**s Where the training output $y(t)$ is used as input to $h(t+1)$. At test time the model output $\sigma(t)$ is used as input to $h(t+1)$.

**Tensor –** Geometric objects that describe linear relations between geometric vectors and scalers expressed as an array of one or more dimensions. The term is often applied to arrays of three or more dimensions. A 0th-order tensor is a **scaler**, 1st-order a **vector**, 2nd-order a **matrix** and 3rd and higher order a **tensor**.

**TensorFlow –** (tensorflow.org) Open Source Software library (Python) for machine intelligence.

**Theano – Pylearn2** specifies machine learning as calls to **Theano** which also can be accessed by C++

**Threads –** Streams of code that can be executed in parallel given the appropriate hardware.

**Tied weights –** Same set of weights used in all neurons in a layer. Usually used in convolution networks.

**TIMIT –** Speech database

**Tokens –** A Token can be a word, a character or a byte or some other discrete entity.

**Torch –** A scientific computing framework with a C interface

**t-distributed Stochastic Neighbor Embedding (t-SNE) –** It is a non-linear dimension reduction methodology to embed higher dimensional data in 2 or 3 dimensional points such that similar objects become neighbors and dissimilar objects are far apart.

## U

**Underfitting –** When unable to get a sufficiently low error on training set.

**Unsupervised Learning –** gains experience from a dataset by learning useful properties enabling clustering, determining probability distribution, synthesis or denoising.

## V

**Velocity - A** term use for an enhanced gradient for momentum learning. See **momentum**.

## W

**Warp –** A collection of **threads**.

**Weight randomization –** In most cases neural net weights are randomly set at initialization. In some cases ,as in **convolutional neural nets,** they may be set to specific values and replicated for each neuron in a specific layer. Random settings are usually in the [0.0, 0.2] range if inputs are in the range of [0.0, 1.0] or [-0.2 to 0.2] if inputs are in the range of [-1.0, 1.0]. Often a neural net will not learn with a given set of starting weights so some experimentation may be necessary.