# Classical Neural Networks

Sung-Yub Kim

Dept of IE, Seoul National University

March 4, 2017


Bishop, C. M. Pattern Recognition and Machine Learning *Information Science and Statistics*, Springer, 2006.


Kevin P. Murphy. Machine Learning - A Probabilistic Perspective *Adaptive Computation and Machine Learning*, MIT press, 2012.


Ian Goodfellow and Yoshua Bengio and Aaron Courville. Deep Learning *Computer Science and Intelligent Systems*, MIT Press, 2016.

- **Curse of Dimensionality**
  The origin of main fallacy in fixed basis model is so called *Curse of dimensionality*. Therefore, we need to apply models can adapt its basis function to the data.

- **Solution for Curse of Dimensionality**
  1. Define basis functions that are centered on the training data points, and then select a subset of these during training.(SVM)
  2. Fix the number of basis function in advance but allow them to adaptive, *i.e.* use parametric forms for the basis functions in which the parameter values are adapted during training.(NN)

- **Motivation**
  While the linear models are

$$y(\mathbf{x}, \mathbf{w}) = f\left\{\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right\} \tag{1}$$

  with *fixed* $\phi_j(\mathbf{x})$ s. Extend this model by making the basis functions $\phi_j(\mathbf{x})$ *depend on parameters* and then to allow these parameters to be adjusted, along with the coefficients $\{w_j\}$, during training.
- **How?**
  NN use basis functions that follow from the same form as 1, so that each basis function is itself a nonolinear function of a linear combination of the inputs.

## Structure of a Neuron

A neuron can be described as *synthesis of signals* and following *nonlinear transformation*.

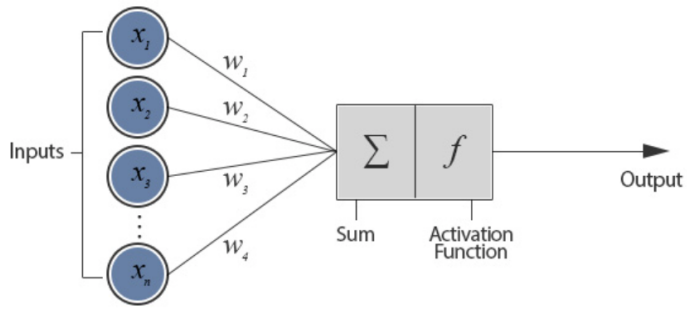


Figure: Single Neuron

1. *A synthesis of signals* can be represented by linear combination of inputs

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \tag{2}$$

where $j = 1, \ldots, M$. $a_j$ is known as *activations*.

2. *A nonlinear transformation* can be represented as

$$z_j = h(a_j) \tag{3}$$

and we call *activation function* $h(\cdot)$ and $z_j$ *hidden unit*.

- **Linear Model with adaptive basis functions**
  Following previous page, we can make linear combination of *hidden units*.

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \tag{4}$$

  where $k = 1, \ldots, K$ and K is the dimension of final ouput. If we take another activation function to this $a_k$ s, then we get $\mathbf{y}$, the final solution. Of course, the final activation can be selected easily.
  (Regression problem $\rightarrow$ identity function with Gaussian assumption, Classification problem $\rightarrow$ Softmax function.)

- **Entire Structure of FFNN**
  Combine our previous discussion, we get

$$y_k(\mathbf{x}, \mathbf{w}) = \left\{ \sum_{j=1}^{M} w_{k0}^{(2)} + w_{kj}^{(2)} h \left\{ \sum_{i=1}^{D} w_{j0}^{(1)} + w_{ji}^{(1)} x_i \right\} \right\} \tag{5}$$

  Since the process of evaluating 5 can be interpreted as a *forward propagation* of information through the network, we call this NN Feed-forward Neural Network(FFNN).

- **Terminology of NN**
  There is some confusion in literatures regarding the terminology for counting the number of layers in FFNN.
  1. If you focus on *how data is transformed*, you would like to argue that 5 is *3-layer network*, regarding that there are *input, output* and *one hidden layer* in network.
  2. If you focus on *what paramters you need to optimize*, you would like to argue that 5 is *2-layer network*, regarding that there are *two synthesize layers* $w^{(1)}, w^{(2)}$.
- **Restriction on FFNN**
  Although we can develop more general definition of network and its mathematical function, we don't discuss those topics in this presentation. Our main interest is FFNN and FFNN must have no closed directed cycles, to ensure that *the outputs are deterministic functions of the inputs*.
- **Approximation property**
  1. First, *NNs are Universal Approximator*. Two-layer network with linear outputs can uniformly approximate any continuous function on a compact input domain to arbitrary accuracy provided the network haas a sufficiently large number of hidden units.
  2. But second, we don't know *how to find the optimal point*. Although there exists so many technique to find the optimal parameters, this research area is still valid and important to Deep Learning Community.

- **W-space Symmetry**
  1. **Sign-flip Symmetry**
     Multiple distinct shoices for the weight vector $\mathbf{w}$ can all give rise to the same mapping function from inputs to outputs. For $M$ hidden units, there will be $M$ *'sign-flip'*, and thus any given weight vector will be one of a set $2^M$ equivalent weight vectors.
  2. **Interchange Symmetry**
     Similarly, imagine that we *interchange* the values of all of the weights leading both into and out of a particular hidden unit with the corresponding values of the weights associated with a different hidden unit. For $M$ hidden units, any given weight vector will belong to a set of $M!$ different orderings of the hidden units.

Therefore, a FFNN will have an overall W-space symmetry factor of $M!2^M$.

- **Regression Problem**

  We assume that target $t$ has a Gaussian distribution with an $\mathbf{x}$ dependent mean, which is given by the output of the neural network, so that

$$\mathcal{L}(t;\mathbf{w}) = \mathcal{N}(t|y(\mathbf{x},\mathbf{w}),\beta^{-1}) \tag{6}$$

  where $\beta$ is the precision of the Gaussian noise.
  And finding process of MLE is equivalent to minimize this function.

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n,\mathbf{w}) - t_n\}^2 \tag{7}$$

  Since $y(\mathbf{x}_n,\mathbf{w})$ is nonlinear function of $\mathbf{w}$, $E(\mathbf{w})$ is nonconvex function of $\mathbf{w}$. Therefore, we cannot find the Global Optimal point easily. When we get $\mathbf{w}_{ML}$, we can evaluate the variance by

$$\frac{1}{\beta_{ML}} = \frac{1}{NK}\sum_{n=1}^{N}\|\mathbf{y}(\mathbf{x}_n,\mathbf{w}) - \mathbf{t}_n\|^2 \tag{8}$$

  where $K$ is the number of target variables.
  By canonical link function, we get

$$\frac{\partial E}{\partial a_k} = y_k - t_k \tag{9}$$

If we want to solve classification problem, we need to assume that our output vector is a discrete probability distribution. In that case, our Likelihood function is

$$\mathcal{L}(t; \mathbf{w}) = y(\mathbf{x}, \mathbf{w})^t \{1 - y(\mathbf{x}, \mathbf{w})\}^{(1-t)}, \text{ (Binary Case)} \tag{10}$$

$$\mathcal{L}(\mathbf{t}; \mathbf{w}) = \prod_{k=1}^{K} y_k(\mathbf{x}, \mathbf{w})^{t_k}, \text{ (K-class Case)} \tag{11}$$

Therefore, we can say that our (cross entropy) loss function is

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln y_k(\mathbf{x}_n, \mathbf{w}) \tag{12}$$

By canonical function, our activation function is

$$y_k(\mathbf{x}, \mathbf{w}) = \frac{\exp(a_k(\mathbf{w}, \mathbf{w}))}{\sum_j \exp(a_j(\mathbf{w}, \mathbf{w}))} \tag{13}$$

Also we get

$$\frac{\partial E}{\partial a_k} = y_k - t_k \tag{14}$$

- **Motivation**
  Since our objective function is disfferentiable in almost everywhere even our activation function is RELU, we can adopt classical nonlinear optimization algorithm to find our local minima. Interesting point is that if we find one local minima, then we can find more than $M!2^M$ equivalent local minima.
- **Initialization**
  Since there exists esveral local minima, we need to compare each local to obtain better local minima.
- **General Scheme**
  In most nonlinear optimization algorithm, we iterate this to obtain local minima

$$w^{(\tau+1)} \leftarrow w^{(\tau)} + \Delta \mathbf{w}^{(\tau)} \tag{15}$$

  Different algorithms involve different choices for the weight vector update $\Delta \mathbf{w}^{(\tau)}$. Many algorithms so called *'First Order Method'* make use of gradient information to update. Deeper discussion can be found in Nesterov.

- **Local Quadratic Approximation**
  By Taylor expansion, we get

$$E(\mathbf{w}) \simeq E(\mathbf{w}_0) + \langle \nabla E(\mathbf{w}_0), (\mathbf{w} - \mathbf{w}_0) \rangle + \frac{1}{2} \langle \nabla^2 E(\mathbf{w}_0)(\mathbf{w} - \mathbf{w}_0), (\mathbf{w} - \mathbf{w}_0) \rangle \tag{16}$$

  Therefore, local approximation to the gradient is

$$\nabla E(w) \simeq \nabla E(\mathbf{w}_0) + \nabla^2 E(\mathbf{w}_0)(\mathbf{w} - \mathbf{w}_0) \tag{17}$$

  By simple calculus, we know that if $\mathbf{w}_0$ is (local) optimal, then $\nabla^2 E(\mathbf{w}_0)$ is PD matrix.

- **How to get Gradient Information?**
  We can get gradient information by means of the *backpropagation procedure*.
- **Why use of gradient information speed up?**
  If we use quadratic approximation, it depends on $O(W^2)$ parameteres, and we should not expect to be able to locate the minimum until we have gathered $O(W^2)$ independent pieces of information.
  If we don't make use of gradient information, we would expect to have to perform $O(W^2)$ function evaluations, each of which would require $O(W)$ steps. Thus, the computational effort needed to find the minimum using such an approach would be $O(W^3)$.
  If we use gradient information, we might hope to find the minimum of the function in $O(W)$ gradient evaluations. By using backprop, each evaluation takes only $O(W)$ steps and so the minimum can now be found in $O(W^2)$ steps.

- **General Gradient Descent**
  General gradient descent method means

$$\mathbf{w}^{(\tau+1)} \leftarrow \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)}) \tag{18}$$

  where the parameter $\eta > 0$ is known as *learning rate*. Techniques that use the whole data set at once are called *batch* methods.

- **Stochastic Gradient Descent**
  Stochastic gradient descent makes an update to the weight vector based on one data point at a time, so that

$$\mathbf{w}^{(\tau+1)} \leftarrow \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)}) \tag{19}$$

- **Property of SGD**
  1. SGD handles redundancy in the data much more efficiently.
  2. Possibility of escaping from local minima, since a stationary point with repect to the error function for the whole data set will generally not be a stationary point for each data point individually.

- **Training Algorithm Scheme**
  At each iterative step, we can distinguish between two distinct stages.
  1. The derivatives of the error function with repect to the weights must be evaluated. Bakcprop can be used for this stage to get derivatives.
  2. The derivatives are then used to compute the adjustments to be made to the weights.
- **What is Backpropagation?**
  Using a local message passing scheme in which information is sent alternately forwards and backwards through the network. In short, Backprop is just the way to get *gradient information efficiently*.

- **Forward Propagation**
  We have supplied the corresponding input vector to the network and calculated the activations of all of the hidden and output units in the network by successive application of *linear combination* and *nonlinear transformation*. We call this process *forward propagation*.
- **Motivation of Back Propagation**
  1. Evaluation of the derivative of $E_n$ wrt a output weight $w_{ji}$ is

$$\begin{aligned} \frac{\partial E_n}{\partial w_{ji}} &= \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} (\because z_j = h(a_j)) \\ &= \delta_j z_i \end{aligned} \tag{20}$$

  2. Evaluation fo the $\delta$'s for hidden units, we make use of the chain rule for partial derivatives

$$\begin{aligned} \delta_j = \frac{\partial E_n}{\partial a_j} &= \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} \\ &= \sum_k \delta_k h'(a_j) w_{kj} \\ &= h'(a_j)\sum_k \delta_k w_{kj} \end{aligned} \tag{21}$$

  which tells us that the value of $\delta$ for a particular hidden unit can be obtained by *propagating* the $\delta$'s backwards from units higher up in the network.

- **BP for Batch method**
  For batch method, the derivative of the batch error $E$ can be obtained by repeating the above steps for each pattern in training set and then summing over

$$\frac{\partial E}{\partial w_{ji}} = \sum_n \frac{\partial E_n}{\partial w_{ji}} \tag{22}$$

- **Evaluation of error function**
  Since the number of weights is typically much greater than the number of units, and so the bulk of computational effort in forward propagation is concerned with evaluating the sums in linear combinations. Therefore, a single evaluation of the error function would require $O(W)$ operations.
- **Finite differences method**
  Each forward propagation requires $O(W)$ steps, and there are $W$ weights in the network each of which must be perturbed individually, so overall scaling is $O(W^2)$.

- **Jacobian matirx**
  *Jacobian matrix* is defined as

$$J_{ki} = \frac{\partial y_k}{\partial x_i} \tag{23}$$

Since the Jacobian matrix provides a measure of the local sensitivity of the outputs to change in each of the input variables, it also allows any known errors $\delta x_i$ associated with the inputs to be propagated the trained network in order to estimate their contribution $\delta y_k$ to the errors ar the outputs, through the relation

$$\delta y_k \simeq \sum_i \frac{\partial y_k}{\partial x_i} \delta x_i \tag{24}$$

which is valid provided the $|\delta x_i|$ are small.

- **Evaluation Jacobian using BP**
  The Jacobian matrix can be evaluated using BP.

$$\begin{aligned} J_{ki} = \frac{\partial y_k}{\partial x_i} &= \sum_j \frac{\partial y_k}{\partial a_j}\frac{\partial a_j}{\partial x_i} = \sum_j \frac{\partial y_k}{\partial a_j} w_{ji} \\ &= \sum_j w_{ji} \sum_l \frac{\partial y_k}{\partial a_l}\frac{\partial a_l}{\partial a_j} = \sum_j w_{ji} h'(a_j) \sum_l w_{lj} \frac{\partial y_k}{\partial a_l} \end{aligned} \tag{25}$$

  This backpropagation starts at the output units for which the required derivatives can be found directly from the functional form of the output-unit activation function. We can calculate $\partial y_k / \partial a_l$ like

$$\frac{\partial y_k}{\partial a_l} = \delta_k(l)\sigma'(a_l)(\text{Binary case}) \tag{26}$$

$$\frac{\partial y_k}{\partial a_l} = \delta_k(l) y_k - y_k y_l(\text{K-class case}) \tag{27}$$

- **Role of Hessian matrix**
  1. Many nonlinear optimization algorithm used for training neural networks are based on considerations of the second-order properties of the error surfaces.
  2. Hessian forms the basis of a fast procedure for re-training a FFNN following a small change in the training data.
  3. The inverse of the Hessian has been used to identify the least significant weights in a network.
  4. Hessian plays a central role in Laplace approximation.

- **Diagonal approximation**
  Diagonal approximation: Replace the off-diagonal elements with zeros, because *its inverse is trivial to evaluate*.
  1. First, we need only diagonal matrix of hessian. Therefore, we need
  $$\frac{\partial^2 E}{\partial w_{ji}^2} = \frac{\partial^2 E}{\partial a_j^2} z_i^2 \tag{28}$$
  2. We can find this value exactly by recursively applying backprop
  $$\begin{aligned} \frac{\partial^2 E}{\partial a_j^2} &= \frac{\partial}{\partial a_j}\left(h'(a_j)\sum_k w_{kj}\frac{\partial E_n}{\partial a_k}\right) \\ &= h'(a_j)^2 \sum_k \sum_{k'} w_{kj} w_{k'j} \frac{\partial^2 E_n}{\partial a_k \partial a_{k'}} + h''(a_j)\sum_k w_{kj}\frac{\partial E_n}{\partial a_k} \end{aligned} \tag{29}$$
  3. Also, we can approximate this quantity by
  $$\frac{\partial^2 E}{\partial a_j^2} \simeq h'(a_j)^2 \sum_k w_{kj}^2 \frac{\partial^2 E_n}{\partial a_k^2} + h''(a_j)\sum_k w_{kj}\frac{\partial E_n}{\partial a_k} \tag{30}$$

  Note that this approximation require $O(W)$ complexity.

- **Outer product approximation**
  If we assume that error function is sum of square, we can write the Hessian in the form

$$\nabla^2 E = \sum_{n=1}^{N} \nabla y_n \nabla y_n^T + \sum_{n=1}^{N} (y_n - t_n) \nabla^2 y_n \tag{31}$$

  Since training process make $y_n$ similar to $t_n$, we can neglect the second term of equation. We call it this approximation *outer product approximation* or *Levenberg-Marquardt approximation*. Note that this approximation is only likely to be valid for a network that has been trained appropriately.

- **Inverse Hessian**

  We can write the outer-product approximation in matrix notation as

$$\nabla^2 E = \sum_{n=1}^{N} \nabla y_n \nabla y_n^T \tag{32}$$

  Suppose we have already obtained the inverse Hessian using the first $L$ data points. By separating off the contribution from data point $L+1$, we obtain

$$\nabla^2 E_{L+1} = \nabla^2 E_L + \nabla y_{L+1} \nabla y_{L+1}^T \tag{33}$$

  In order to evaluate the inverse of Hessian, we consider *Woodbury identity*

$$(M + vv^T)^{-1} = M^{-1} - \frac{(M^{-1}v)(v^T M^{-1})}{1 + v^T M^{-1} v} \tag{34}$$

  By applying this, we get

$$H_{L+1}^{-1} = H_L^{-1} - \frac{(H_L^{-1} \nabla y_{L+1})(\nabla y_{L+1}^T H_L^{-1})}{1 + \nabla y_{L+1}^T H_L^{-1} \nabla y_{L+1}} \tag{35}$$

- **Motivation of Consistency**
  If we train network using the original data and linear transformed data, consistency requires that we should obtain euqivalent network. Any regularizer should be consistent with this property, otherwise it arbitrarily favors one solution over another. Such a regularizer is given by

$$\frac{\lambda_1}{2} \sum_{w \in \mathcal{W}_1} w^2 + \frac{\lambda_2}{2} \sum_{w \in \mathcal{W}_2} w^2 \tag{36}$$

  But this prior is *improper* because the boas parameter are unconstrained.

- **Motivation of early stopping**
  During training process, generalization error shows a decrease at first, followed by an increase as the network starts to over-fit. The behavior of the network in this case is sometimes explained qualitatively in terms of the effective number of degrees of freedom in the network, in which starts out small and then grows during training process, corresponding to a steady increase in the effecive complexity of the model.
  In the case of a quadratic error function, we can verify this insight, and show that early stopping should exhibit similar behavior to regularization using a simple weight-decay term.

- **Needs of invariant**
  In may applications of pattern recognition, it is known that predicitons should be *invariant* under one or more transformations of the input variables.
- **Approaches to invariant**
  1. Augmented training set: Computationally cost.
  2. Regularization term is added: Tangent propagation
  3. Invariance is built into the pre-processing by extracting features that are invariant under the required transformations.
  4. Build the invariance properties into the structure of a neural network.:CNN
  5. Soft weight sharing: Reduce the effective complexity of a network with a large number of weights is to constrain weights within certain groups to be equal.

- **Mixture density network**
  Using a mixture model for $p(\mathbf{t}|\mathbf{x})$ in which both the mixing coefficients as well as the component densities are flexible functions of the input vector $\mathbf{x}$.
- **Output of Mixture density network**
  1. Mixing coefficients(Prior probabilities): $\pi_k(\mathbf{x}) \in R$
  2. Means: $\mu_k(\mathbf{x}) \in \mathbb{R}^L$
  3. Variances: $\sigma_k^2(\mathbf{x}) \in \mathbb{R}$

  Since there exist $k = 1, \ldots, K$ mixture densities, our output is $(L+2)K$.
- **Error function of MDN**
  For independent data, error function takes the form

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k(\mathbf{x}_n, \mathbf{w}) \mathcal{N}(\mathbf{t}_n | \mu_k(\mathbf{x}_n, \mathbf{w}), \sigma_k^2(\mathbf{x}_n, \mathbf{w})) \right\} \tag{37}$$

- **Bayesian treatment of NN**
  Bayesian treatment we need to marginalize over the distribution of parameters in order to make predictions. In the case of multilayered network, the hughly nonlinear dependence of the network function on the parameter values means that an exact Bayesian treatment can no longer be found.
- **Log-likelihood of model**
  If we set Gaussian prior for $\mathbf{w}$ and noise, then our log-posterior is

$$\ln p(\mathbf{w}|\mathcal{D}) = -\frac{\alpha}{2}\mathbf{w}^T\mathbf{w} - \frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 + const. \tag{38}$$

  Use nonlinear optimization algorithms to get MAP and we can use linear approximation to get linear-Gaussian model of likelihood function

$$p(t|\mathbf{x}, \mathbf{w}, \beta) \simeq \mathcal{N}(t|y(x, \mathbf{w}_{MAP}) + \nabla_w y(w - \mathbf{w}_{MAP}), \beta^{-1}) \tag{39}$$

  By convolution, we get Gaussian predictive distribution

$$p(t|\mathbf{x}, \mathcal{D}, \alpha, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}_{MAP}), \sigma^2(\mathbf{x})) \tag{40}$$

  where $\sigma^2(\mathbf{x}) = \beta^{-1} + \mathbf{g}^T A^{-1}\mathbf{g}$.

- **Marginal Likelihood**
  Marginal likelihood, or evidence, for the hyperparameters is obtained by integrating over the network weights

$$p(\mathcal{D}|\alpha,\beta) = \int p(\mathcal{D}|\mathbf{w},\beta)p(\mathbf{w}|\alpha)d\mathbf{w} \tag{41}$$

By using Laplace approximation

$$\ln p(\mathcal{D}|\alpha,\beta) \simeq -E(\mathbf{w}_{MAP}) - \frac{1}{2}\ln|A| + \frac{W}{2}\ln\alpha + \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) \tag{42}$$

where

$$E(\mathbf{w}_{MAP}) = \frac{\beta}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n,\mathbf{w}_{MAP}) - t_n\}^2 + \frac{\alpha}{2}\mathbf{w}_{MAP}^T\mathbf{w}_{MAP} \tag{43}$$

Then we get

$$\alpha = \frac{\gamma}{\mathbf{w}_{MAP}^T\mathbf{w}_{MAP}}, \frac{1}{\beta} = \frac{1}{N-\gamma}\sum_{n=1}^{N}\{y(\mathbf{x}_n,\mathbf{w}_{MAP}) - t_n\}^2 \tag{44}$$

- **Find MAP and optimal hyperparamter**
  The only difference is we use cross entropy loss function. Suppose we get $\mathbf{w}_{MAP}$ and optimal hyper parameter. Now we would like to get predictive distribution. The simples approximation is to assume that the posterior distribution is very narrow and hence make the approximation

$$p(t|\mathbf{x}, \mathcal{D}) \simeq p(t|\mathbf{x}, \mathbf{w}_{MAP}) \tag{45}$$

- **Take account of the variance of the posterior**
  Make a linear approximation for the output unit activation in the form

$$a(\mathbf{x}, \mathbf{w}) \simeq a_{MAP}(\mathbf{x}) + \nabla_w a^T(\mathbf{x} - \mathbf{w}_{MAP}) \tag{46}$$

  Posterior distribution of output unit activation value is

$$p(a|\mathbf{x}, \mathcal{D}) = \int \delta(a - a_{MAP}(x)(\mathbf{x} - \nabla_w a(x)(\mathbf{w} - \mathbf{w}_{MAP}))q(\mathbf{w}|\mathcal{D})d\mathbf{w} \tag{47}$$

  and this is Gaussian distribution. Finally we get predictive distribution

$$p(t = 1|\mathbf{x}, \mathcal{D}) = \int \sigma(a)p(a|\mathbf{x}, \mathcal{D})da \tag{48}$$