Statistical Machine Learning

Christian Walder

College of Engineering and Computer Science
The Australian National University

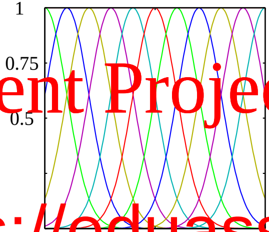
Canberra
Semester One, 2020.

(Many figures from C. M. Bishop, "Pattern Recognition and Machine Learning")

Outlines

Part VII

Basis Functions



- Play a crucial role in the algorithms explored so fa
- Previously (*e.g.* Linear Regression and Lin Classification): were fixed before learning sta
- Now for Neural Networks: number of basis functions fixed, parameters of the basis functions are adaptive
- Later in kernel methods: center basis functions on the data / have an infinite number of effective basis functions (*e.g.* Support Vector Machines).

Outline

- The functional form of the network model (including special parametrisation of the basis functions).
- Ho
  ma
  (So

- Error backpropagation : efficiently evaluate t
  of the log likelihood function with respect to the ne
  parameters.

- Various approaches to regularise neural net

Feed-forward Network Functions

- Same goal as before: *e.g.* for regression, decompose

$$t(\mathbf{x}) = y(\mathbf{x}, \mathbf{w}) + \epsilon$$

  where ϵ is the noise.

- (Ge

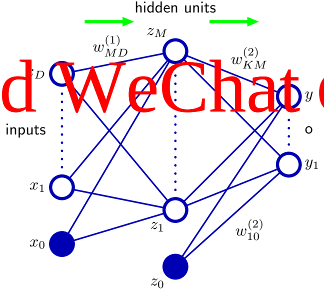
$$j=0$$

  where $\phi = (\phi_0, \ldots, \phi_M)^T$ is the fixed mode
  $\mathbf{w} = (w_0, \ldots, w_M)^T$ are the model paramete

- For regression: $f(\cdot)$ is the identity function.
- For classification: $f(\cdot)$ is a nonlinear activation function.
- Goal : Let $\phi_j(\mathbf{x})$ depend on parameters, and then adjust these parameters together with $\mathbf{w}$.

Feed-forward Network Functions

- Goal : Let $\phi_j(\mathbf{x})$ depend on parameters, and then adjust these parameters together with $\mathbf{w}$.
- Many ways to do this.
- Neural networks use basis functions which follow the same form as the (generalised) linear model.
- EA
  ada



Functional Transformations

- Construct M linear combinations of the input variables $x_1, \dots, x_D$ in the form

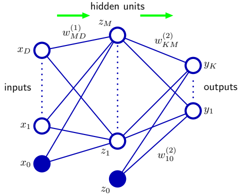
$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \qquad j = 1, \dots, M$$

activations   weights   bias

- Ap
get t

j j

- $h(\cdot)$ is typically sigmoid, $\tanh$, or more rec
ReLU$(x) = \max(x, 0)$

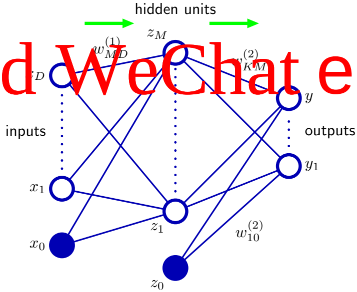


Functional Transformations

- Outputs of the hidden units are again linearly combined

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \qquad k = 1, \dots, K$$

- Ap
  $g(\cdot)$



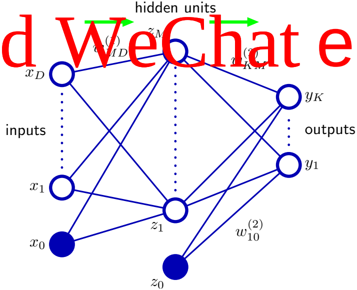
Functional Transformations

- The activation function $g(\cdot)$ is determined by the nature of the data and the distribution of the target variables.
- For standard regression: $g(\cdot)$ is the identity so $y_k = a_k$.
- For multiple binary classification, $g(\cdot)$ is a logistic sigmoid:

$$y_k = \sigma(a_k) = \frac{1}{[illegible]}$$

- Re [illegible]

$$\frac{[illegible]}{p(\mathbf{x}, \mathcal{C}_{k_2})}$$

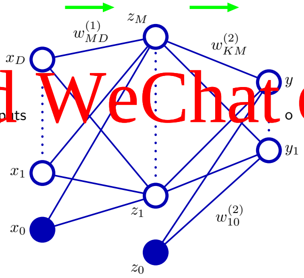


Functional Transformations

- Combine all transformations into one formula

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

wh

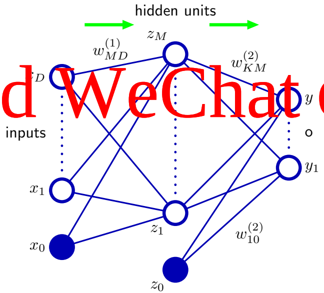


Functional Transformations

- As before, the biases can be absorbed into the weights by introducing an extra input $x_0 = 1$ and a hidden unit $z_0 = 1$.

$$y_k(\mathbf{x}, \mathbf{w}) = g\left(\sum_{j=0}^{M} w_{kj}^{(2)}\, h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right)$$

wh

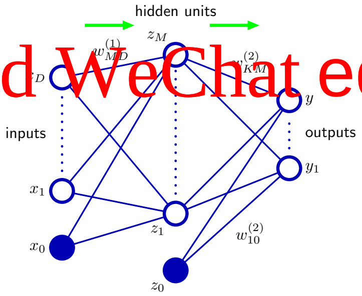


Comparison to Perceptron

- A neural network looks like a multilayer perceptron.
- But perceptron's nonlinear activation function was a step function — neither smooth nor differentiable.

$$f(a) = \begin{cases} +1, & a \geq 0 \end{cases}$$

- The
  are s



Linear Activation Functions

- If all activation functions are linear functions then there exists an equivalent network without hidden units. (Composition of linear functions is a linear function.)
- But if t
  tha
  line

- Dimensionality reduction.
- *c.f.* Principal Component Analysis (upcomi

- Generally, most neural networks use nonline functions as the goal is to approximate a nonline mapping from the input space to the outputs.

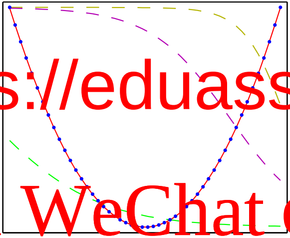
Neural Networks as Universal Function Approximators

- Feed-forward neural networks are universal app
- Exa
  can
  co
  enough hidden units.
- Holds for a wide range of hidden unit activation fu
- Remaining big question : Where do we get the ap settings for the weights from? With other words we learn the weights from training examples?

Aproximation Capabilities of Neural Networks

- Neural network approximating

$$f(x) = x^2$$



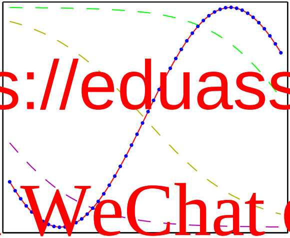
Two-layer network with 3 hidden units (
functions) and linear outputs trained on 50 data points sampled from the interval $(-1, 1)$. Red: resulting output. Dashed: Output of the hidden units.

Aproximation Capabilities of Neural Networks

Ong & Walder & Webers
Data61 | CSIRO
The Australian National University

- Neural network approximating

$$f(x) = \sin(x)$$

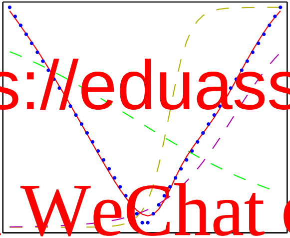


Two-layer network with 3 hidden units (
functions) and linear outputs trained on 50 data points sampled from the interval $(-1, 1)$. Red: resulting output. Dashed: Output of the hidden units.

Aproximation Capabilities of Neural Networks

- Neural network approximating

$$f(x) = |x|$$

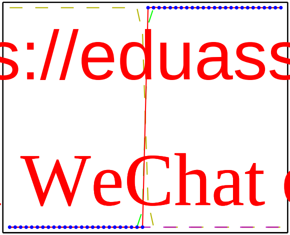


Two-layer network with 3 hidden units (functions) and linear outputs trained on 50 data points sampled from the interval $(-1, 1)$. Red: resulting output. Dashed: Output of the hidden units.

Aproximation Capabilities of Neural Networks

- Neural network approximating Heaviside function

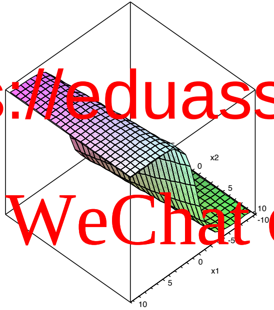
$$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$



Two-layer network with 3 hidden units ($\tanh$ activation functions) and linear outputs trained on 50 data points sampled from the interval $(-1, 1)$. Red: resulting output. Dashed: Output of the hidden units.

Variable Basis Functions in a Neural Networks

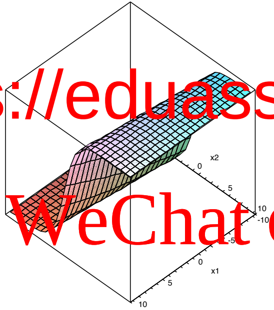
- Hidden layer nodes represent parametrised basis functions



$z = \sigma(w_0 + w_1 x_1 + w_2 x_2)$ for $(w_0, w_1, w_2) = (0.0, 1.0, 0.1)$

Variable Basis Functions in a Neural Networks

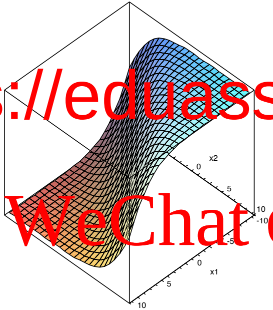
- Hidden layer nodes represent parametrised basis functions



$z = \sigma(w_0 + w_1 x_1 + w_2 x_2)$ for $(w_0, w_1, w_2) = (0.0, 0.1, 1.0)$

Variable Basis Functions in a Neural Networks

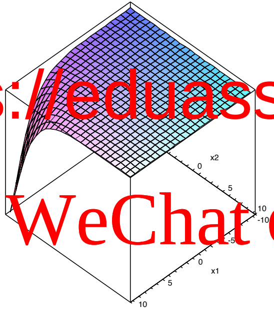
- Hidden layer nodes represent parametrised basis functions



$z = \sigma(w_0 + w_1 x_1 + w_2 x_2)$ for $(w_0, w_1, w_2) = (0.0, -0.5, 0.5)$

Variable Basis Functions in a Neural Networks

- Hidden layer nodes represent parametrised basis functions

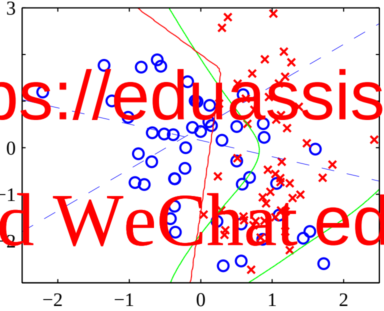


$z = \sigma(w_0 + w_1 x_1 + w_2 x_2)$ for $(w_0, w_1, w_2) = (10.0, -0.5, 0.5)$

Aproximation Capabilities of Neural Networks

Ong & Walder & Webers
Data61 | CSIRO
The Australian National University

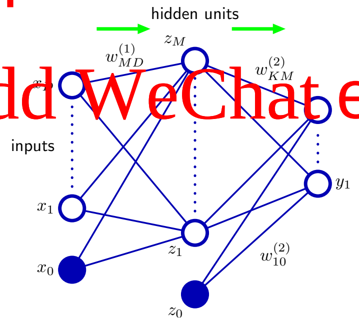
- Neural network for two-class classification.
- 2 inputs, 2 hidden units with $\tanh$ activation function, 1 output with logistic sigmoid activation function.



Red: $y = 0.5$ decision boundary. Dashed blue: $z = 0.5$ hidden unit contours. Green: Optimal decision boundary from the known data distribution.

Weight-space Symmetries

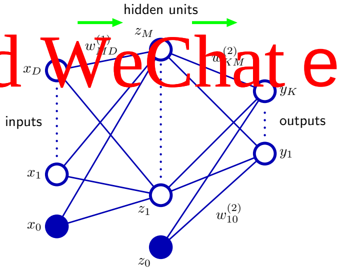
- Given a set of weights $\mathbf{w}$. This fixes a mapping from the input space to the output space.
- Does there exist another set of weights realising the same mapping?
- Assume $\tanh$ activation function for the hidden units. As $\tan$
- Ch
  of th



Weight-space Symmetries

- M hidden units, therefore 2^M equivalent weight vectors.
- Furthermore, exchange all of the weights going into and out of a hidden unit with the corresponding weights of another hidden unit. Mapping stays the same. $M!$ symmetries.
- Ov

M							



- Assume the error $E(\mathbf{w})$ is a smooth function of the weights.
- Sm

- This could be a minimum, maximum, or saddle point.
- Furthermore, because of symmetry in weight s are at least $M!\,2^M$ other critical points with for the error.

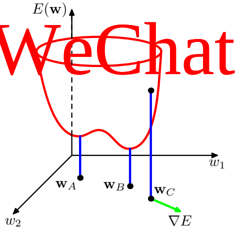
Definition (Global Minimum)

A point $\mathbf{w}^*$ for which the error $E(\mathbf{w}^*)$ is smaller than any other error $E(\mathbf{w})$.

Definitio

A point

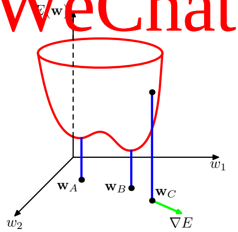
error $E($



Parameter Optimisation

- Finding the global minimum is difficult in general (would have to check everywhere) unless the error function comes from a special class (e.g. smooth convex functions have only one local minimum).
- Error functions for neural networks are not convex (sy
- But fi
- Us find

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(}$$



Local Quadratic Approximation

- Around a stationary point $\mathbf{w}^*$ we can approximate

$$E(\mathbf{w}) \simeq E(\mathbf{w}^*) + \frac{1}{2}(\mathbf{w} - \mathbf{w}^*)^T \mathbf{H}(\mathbf{w} - \mathbf{w}^*),$$

where the Hessian $\mathbf{H}$ is evaluated at $\mathbf{w}^*$ so that

- Usi $\{\ _i\}$

$$\mathbf{H}\mathbf{u}_i = \lambda_i \mathbf{u}_i,$$

to expand

$$\mathbf{w} - \mathbf{w}^* = \sum_i \alpha_i \mathbf{u}_i.$$

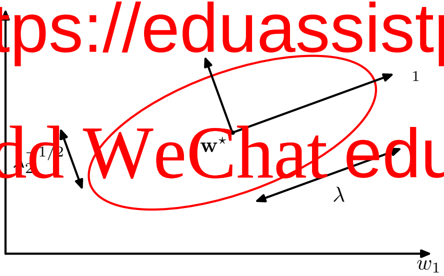
- We get

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2} \sum_i \lambda_i \alpha_i^2.$$

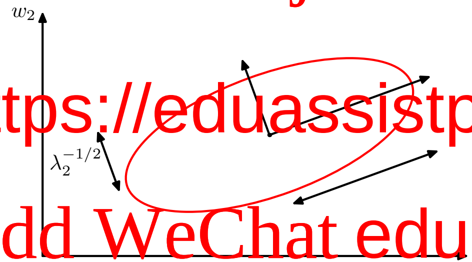
Local Quadratic Approximation

- Around a minimum $\mathbf{w}^*$ we can approximate

$$E(\mathbf{w}) = E(\mathbf{w}^*) + \frac{1}{2} \quad \lambda_i \alpha_i^2 .$$



- Around a minimum $\mathbf{w}^*$, the Hessian $\mathbf{H}$ must be positive definite if evaluated at $\mathbf{w}^*$.



- This explains why the Laplace approximation always yields a valid covariance matrix.

Gradient Information improves Performances

- Hessian is symmetric and contains $W(W+1)/2$ independent entries where W is the total number of weights in the network.

- If we use function evaluations only:
  - 2

- If we
  - Surprisingly the gradient ∇E also costs although it provides W pieces of informa
  - We now need only $O(W)$ steps, so the o complexity is reduced to $O(W^2)$.

FYI only: In general we have the "cheap gradient principle". See (Griewank, A., 2000. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, Section 5.1).

Gradient Descent Optimisation

- Batch processing : Update the weight vector with a small ste

  where η is the learning rate.
- After each step, re-evaluate the gradient
- Gradient Descent has problems in long valley

Gradient Descent Optimisation

- Gradient Descent has problems in 'long valleys'.

Example of zig-zag of Gradient Descent Algorithm.

Nonlinear Optimisation

- Use Conjugate Gradient Descent instead of Gradient Descent to avoid zig-zag behaviour.
- Us
  He
  usu
- Us
  calculates an estimate of the inverse Hessian w
  iterating.
- Even simpler are *momentum* based stra
- Run the algorithm from a set of starting points to fin
  smallest local minimum.

- Remaining big problem: Error function is defined over the whole training set. Therefore, need to process the whole training set for each calculation of the gradient $\nabla E(\mathbf{w}^{(\tau)})$.
- If the

$$n=1$$

we can use on-line gradient descent (also called sequential gradient descent or stochastic gradient descent) to update the weights by one data point

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)}).$$

FYI Only: NN Extensions

- Add more hidden layers (deep learning). To make it work we need many of the following tricks:
- Clever weight initialisation to ensure the gradient is flowing through the entire network.
- So
  sub
- Fav
  gra
- Clever regularisation methods such as dropo
- Specific architectures, not further considere
  - Parameters may be shared, notably as in conv neural networks for images.
  - A state space model with neural network transitions is a recurrent neural network.
  - Attention mechanisms learn to focus on specific parts of an input.