Deep Learning

3.2 Multi-layer Perceptron (MLP)

Dr. Konda Reddy Mopuri
kmopuri@iittp.ac.in
Dept. of CSE, IIT Tirupati

Recap: Linear classifier

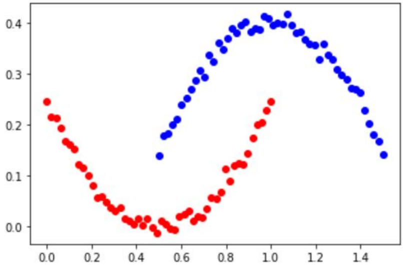
1. $f(x) = \sigma(\mathbf{w}^T\mathbf{x} + b)$

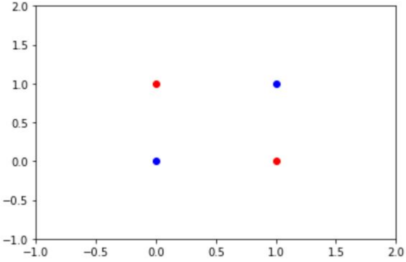
Recap: Linear classifier

1. $f(x) = \sigma(\mathbf{w}^T\mathbf{x} + b)$
2. Seen a couple of simple examples: MP neuron and Perceptron

Linear Classifiers: Shortcomings

- Lower capacity: data has to be linearly separable
- Some times no hyper-plane can separate the data (e.g. XOR data)





Pre-processing

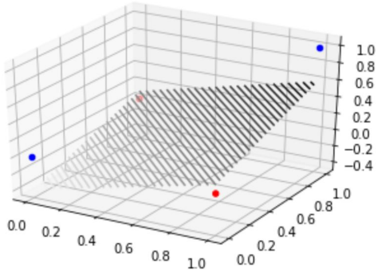
1. Sometimes, data specific pre-processing makes the data linearly separable

Pre-processing

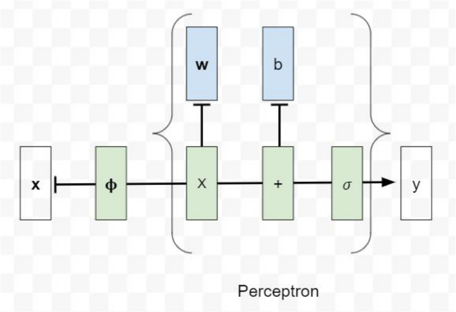
1. Sometimes, data specific pre-processing makes the data linearly separable
2. Consider the xor case
   $\phi(\mathbf{x}) = \phi(x_u, x_v) = (x_u, x_v, x_u x_v)$

Pre-processing

1. Sometimes, data specific pre-processing makes the data linearly separable
2. Consider the xor case
   $\phi(\mathbf{x}) = \phi(x_u, x_v) = (x_u, x_v, x_u x_v)$
3. Consider the perceptron in the new space $f(\mathbf{x}) = \sigma(\mathbf{w}^T \phi(\mathbf{x}) + b)$



Pre-processing



Pre-processing

1. Recap the polynomial regression, by increasing the degree D, we can increase the model capacity

Pre-processing

1. Recap the polynomial regression, by increasing the degree D, we can increase the model capacity
2. Also, remember the Bias-Variance decomposition: for reducing the bias error, we increased the model capacity

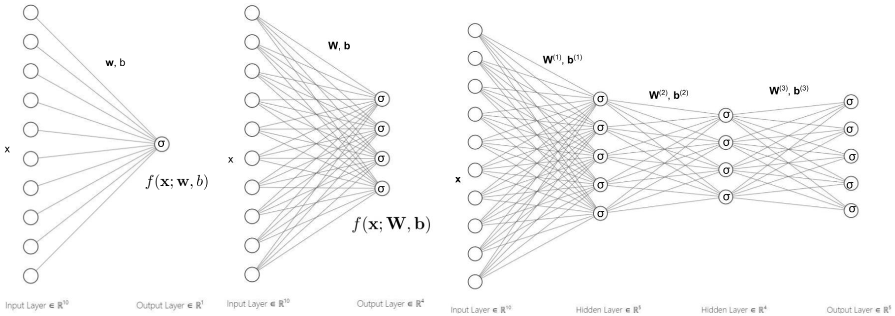
Pre-processing

1. Recap the polynomial regression, by increasing the degree D, we can increase the model capacity
2. Also, remember the Bias-Variance decomposition: for reducing the bias error, we increased the model capacity
3. Feature design (or pre-processing) may also be another way to reduce the capacity without affecting (or improving) the bias

Extending Linear Classifier

1. Linear classifier $f(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + b)$ from $\mathcal{R}^D \to \mathcal{R}$ where $\mathbf{w}$ and $\mathbf{x} \in \mathcal{R}^D$
   can be extended to multi-dimension output
   $f(\mathbf{x}) = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$ from $\mathcal{R}^D \to \mathcal{R}^C$ where
   $\mathbf{W} \in \mathcal{R}^{C \times D}$ and $\mathbf{b} \in \mathcal{R}^C$, and σ is applied element-wise

Single unit to a layer of Perceptrons



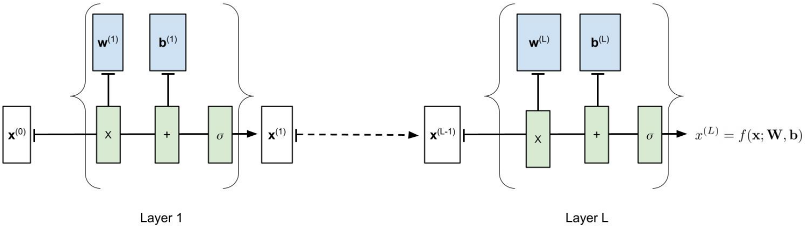
Formal Representation

1. Latter is known as an MLP: Multi-Layered Perceptron (i.e, Multi-Layered network of Perceptrons)

Formal Representation

1. Latter is known as an MLP: Multi-Layered Perceptron (i.e, Multi-Layered network of Perceptrons)
2. can be represented as:
   $\mathbf{x}^{(0)} = \mathbf{x}$,
   $\forall l = 1, \ldots, L, \quad \mathbf{x}^{(l)} = \sigma(\mathbf{W}^{(l)^T} \mathbf{x}^{(l-1)} + \mathbf{b}^{(l)})$, and

MLP



$\mathbf{w}^{(1)}$
$\mathbf{b}^{(1)}$
$\mathbf{x}^{(0)}$
X
+
σ
$\mathbf{x}^{(1)}$
$\mathbf{x}^{(L-1)}$
$\mathbf{w}^{(L)}$
$\mathbf{b}^{(L)}$
X
+
σ
$x^{(L)} = f(\mathbf{x}; \mathbf{W}, \mathbf{b})$
Layer 1
Layer L

Nonlinear Activation

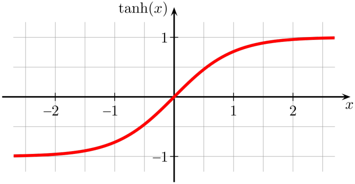
1. Note that σ is nonlinear

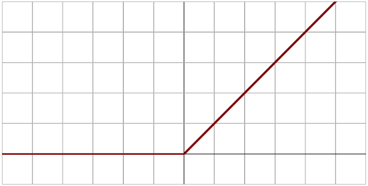
Nonlinear Activation

1. Note that σ is nonlinear
2. If it is an affine function, the full MLP becomes a complex affine transformation (composition of a series of affine mappings)

Nonlinear Activation

Familiar activation functions





Hyperbolic Tangent (Tanh) $x \to \frac{2}{1+e^{-2x}} - 1$ and Rectified Linear Unit (ReLU) $x \to \max(0, x)$ respectively

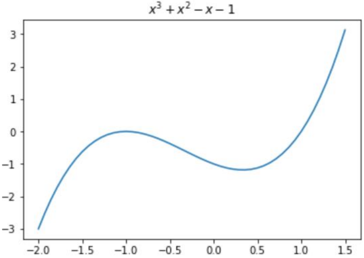
Universal Approximation using ReLU functions

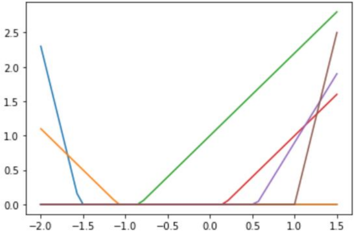
1. We can approximate any function f from $[a, b]$ to $\mathcal{R}$ with a linear combination of ReLU functions

Universal Approximation using ReLU functions

1. We can approximate any function f from $[a, b]$ to $\mathcal{R}$ with a linear combination of ReLU functions
2. Let's approximate the following function using a bunch of ReLUs:

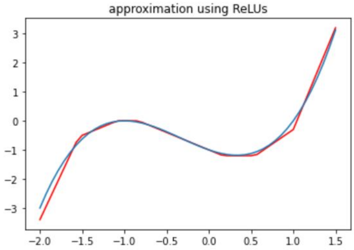
$n_1 = ReLU(-5x - 7.7), n_2 = ReLU(-1.2x - 1.3), n_3 = ReLU(1.2x + 1), n_4 = ReLU(1.2x - 0.2), n_5 = ReLU(2x - 1.1), n_6 = ReLU(5x - 5)$





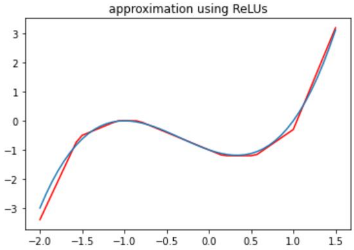
Universal Approximation using ReLU functions

1. Appropriate combination of these ReLUs:
$-n_1 - n_2 - n_3 + n_4 + n_5 + n_6$



Universal Approximation using ReLU functions

1. Appropriate combination of these ReLUs:
   $-n_1 - n_2 - n_3 + n_4 + n_5 + n_6$
2. Note that this also holds in case of other activation functions with mild assumptions.



Universal Approximation Theorem

1. We can approximate any continuous function $\psi : \mathcal{R}^D \rightarrow R$ with one hidden layer of perceptrons

Universal Approximation Theorem

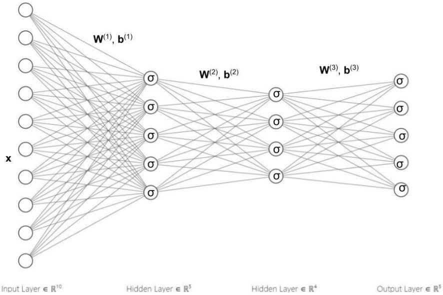
1. We can approximate any continuous function $\psi : \mathcal{R}^D \rightarrow R$ with one hidden layer of perceptrons
2. $\mathbf{x} \rightarrow \mathbf{w}^T \sigma(W\mathbf{x} + \mathbf{b})$
   $\mathbf{b} \in \mathcal{R}^C, W \in \mathcal{R}^{C \times D}, \mathbf{w} \in \mathcal{R}^C,$ and $\mathbf{x} \in \mathcal{R}^D$

Universal Approximation Theorem

1. We can approximate any continuous function $\psi : \mathcal{R}^D \rightarrow R$ with one hidden layer of perceptrons
2. $\mathbf{x} \rightarrow \mathbf{w}^T \sigma(W\mathbf{x} + \mathbf{b})$
   $\mathbf{b} \in \mathcal{R}^C, W \in \mathcal{R}^{C \times D}, \mathbf{w} \in \mathcal{R}^C,$ and $\mathbf{x} \in \mathcal{R}^D$
3. Better approximation requires larger hidden layer (C)
   - Theorem doesn't discuss their relation

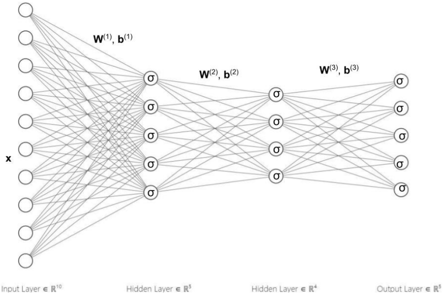
MLP for regression

1. Output is a continuous variable in $\mathcal{R}^D$
   - Output layer has that many perceptrons (When $D = 1$, regresses a scalar value)
   - Generally employs a squared error loss



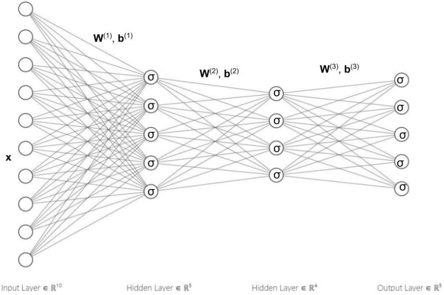
MLP for regression

1. Output is a continuous variable in $\mathcal{R}^D$
   - Output layer has that many perceptrons (When $D = 1$, regresses a scalar value)
   - Generally employs a squared error loss
2. Can have an arbitrary depth (number of layers)



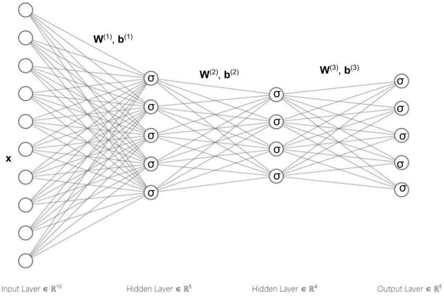
MLP for classification

1. Categorical output in $\mathcal{R}^C$ where C is the number of categories



MLP for classification

1. Categorical output in $\mathcal{R}^C$ where C is the number of categories
2. Predicts the scores/confidences/probabilities towards each category
   - Then converts into a pmf
   - Employs loss that compares the probability distributions (e.g. cross-entropy)



MLP for classification

1. Categorical output in $\mathcal{R}^C$ where C is the number of categories
2. Predicts the scores/confidences/probabilities towards each category
   - Then converts into a pmf
   - Employs loss that compares the probability distributions (e.g. cross-entropy)
3. Can have an arbitrary depth