Lecture 1.2: Deep Feedforward Networks

Haizhao Yang
Department of Mathematics
University of Maryland College Park

2022 Summer Mini Course
Tianyuan Mathematical Center in Central China

- Overview of Deep Feedforward Networks

- Architecture Design

- Back-Propagation

Goal of Deep Feedforward Networks

Typical tasks of machine learning

- Learn a map from an input x to an output y;

Goal of Deep Feedforward Networks

Typical tasks of machine learning

- Learn a map from an input x to an output y;
- Denote the map as $y = f(x)$;

Goal of Deep Feedforward Networks

Typical tasks of machine learning

- Learn a map from an input x to an output y;
- Denote the map as $y = f(x)$;
- Construct a finite family of maps $\{f(x;\theta)\}_\theta$;

Goal of Deep Feedforward Networks

Typical tasks of machine learning

- Learn a map from an input x to an output y;
- Denote the map as $y = f(x)$;
- Construct a finite family of maps $\{f(x;\theta)\}_\theta$;
- Create some criteria to quantify how good a map is;

Goal of Deep Feedforward Networks

Typical tasks of machine learning

- Learn a map from an input x to an output y;
- Denote the map as $y = f(x)$;
- Construct a finite family of maps $\{f(x; \theta)\}_\theta$;
- Create some criteria to quantify how good a map is;
- Use optimization to find the best $f(x; \theta)$

What's Deep Feedforward Networks?

Ideas of deep learning

- Instead of constructing the function approximation $f(x;\theta)$ directly, we use function composition to construct the approximation:

$$f(x;\theta) = f_L^{\theta_L} \circ f_{L-1}^{\theta_{L-1}} \circ f_1^{\theta_1}(x) = f_L(f_{L-1}(\ldots f_1(x;\theta_1)\ldots;\theta_{L-1});\theta_L),$$

where $\theta = (\theta_L, \ldots, \theta_1)$.

What's Deep Feedforward Networks?

Ideas of deep learning

- Instead of constructing the function approximation $f(x;\theta)$ directly, we use function composition to construct the approximation:
$$f(x;\theta) = f_L^{\theta_L} \circ f_{L-1}^{\theta_{L-1}} \circ f_1^{\theta_1}(x) = f_L(f_{L-1}(\dots f_1(x;\theta_1)\dots;\theta_{L-1});\theta_L),$$
where $\theta = (\theta_L, \dots, \theta_1)$.
- $f_1^{\theta_1}$ is called the first layer;

What's Deep Feedforward Networks?

Ideas of deep learning

- Instead of constructing the function approximation $f(x;\theta)$ directly, we use function composition to construct the approximation:
$$f(x;\theta) = f_L^{\theta_L} \circ f_{L-1}^{\theta_{L-1}} \circ f_1^{\theta_1}(x) = f_L(f_{L-1}(\dots f_1(x;\theta_1)\dots;\theta_{L-1});\theta_L),$$
where $\theta = (\theta_L, \dots, \theta_1)$.
- $f_1^{\theta_1}$ is called the first layer;
- $f_L^{\theta_L}$ is called the last layer;

What's Deep Feedforward Networks?

Ideas of deep learning

- Instead of constructing the function approximation $f(x;\theta)$ directly, we use function composition to construct the approximation:
$$f(x;\theta) = f_L^{\theta_L} \circ f_{L-1}^{\theta_{L-1}} \circ f_1^{\theta_1}(x) = f_L(f_{L-1}(\dots f_1(x;\theta_1)\dots;\theta_{L-1});\theta_L),$$
where $\theta = (\theta_L, \dots, \theta_1)$.
- $f_1^{\theta_1}$ is called the first layer;
- $f_L^{\theta_L}$ is called the last layer;
- Other functions are called hidden layers and L is the depth of the model.

Questions for Deep Feedforward Networks

Question

- How to construct each layer $f(x, \theta_i)$?

Questions for Deep Feedforward Networks

Question

- How to construct each layer $f(x, \theta_i)$?
- What's the capacity of

$$f(x;\theta) = f_L^{\theta_L} \circ f_{L-1}^{\theta_{L-1}} \circ f_1^{\theta_1}(x)?$$

Questions for Deep Feedforward Networks

Question

- How to construct each layer $f(x, \theta_i)$?
- What's the capacity of

$$f(x;\theta) = f_L^{\theta_L} \circ f_{L-1}^{\theta_{L-1}} \circ f_1^{\theta_1}(x)?$$

- What's the efficiency of

$$f(x;\theta) = f_L^{\theta_L} \circ f_{L-1}^{\theta_{L-1}} \circ f_1^{\theta_1}(x)?$$

Models for each layer

- Linear models are nice and simple: $y = f(x; w) = w^T x$

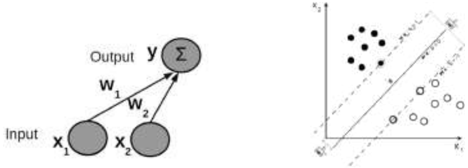


Figure: SVM for classifying two set of points.

Models for each layer

- Linear models are nice and simple: $y = f(x; w) = w^T x$

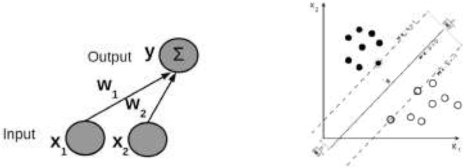


Figure: SVM for classifying two set of points.

- However, nonlinear problems are more common

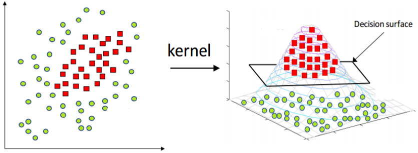


Figure: Need to introduce a nonlinear kernel mapping input x to its feature $\phi(x)$ such that we have a linear problem in the feature space. This is the kernel SVM $y = f(x; w, \phi) = w^T \phi(x)$.

Models for each layer

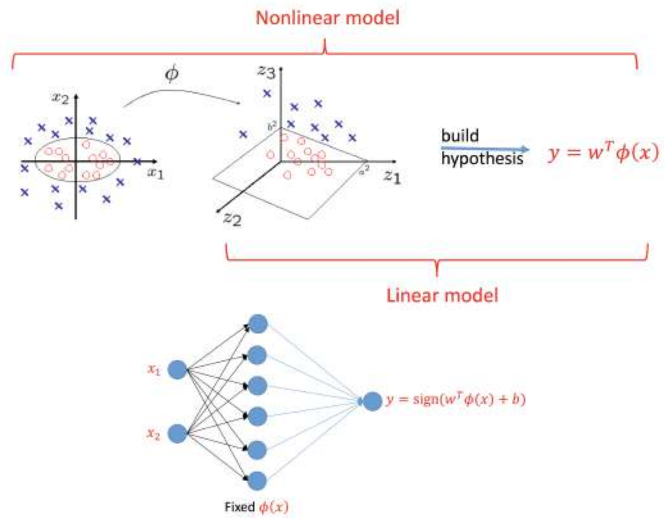


Figure: Kernel SVM for classifying two set of points.

Models for each layer

- Need a linear transform to shift points;
- Need a nonlinear transform to create linear separation;
- Need a linear transform for classification.

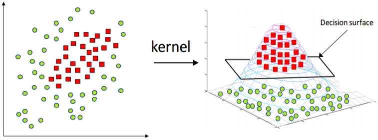


Figure: Kernel SVM for classifying two set of points.

Models for each layer

- Need a linear transform to shift points;
- Need a nonlinear transform to create linear separation;
- Need a linear transform for classification.

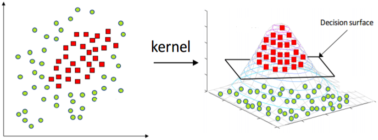


Figure: Kernel SVM for classifying two set of points.

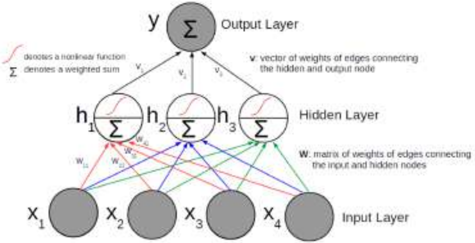


Figure: In sum, we have motivated the simple 3 layer neuron network: $y = V^T\sigma(W^Tx + b)$.

Models for each layer

- Below: FNN with 4 inputs, one hiddel layer with 3 nodes, and 1 output

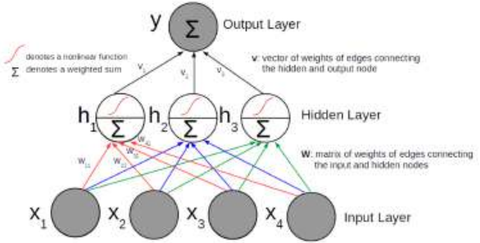


Figure: $y = V^T \sigma(W^T x + b)$.

Models for each layer

- Below: FNN with 4 inputs, one hiddel layer with 3 nodes, and 1 output

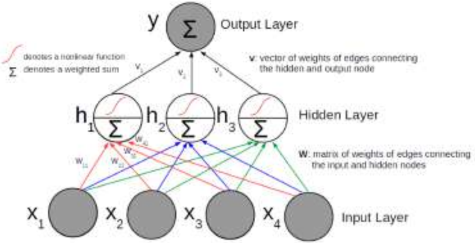


Figure: $y = V^T \sigma(W^T x + b)$.

- Each hidden node computes a nonlinear transformation of its incoming inputs
  - Weighted linear combination followed by a nonlinear activation function σ

Models for each layer

- Below: FNN with 4 inputs, one hiddel layer with 3 nodes, and 1 output

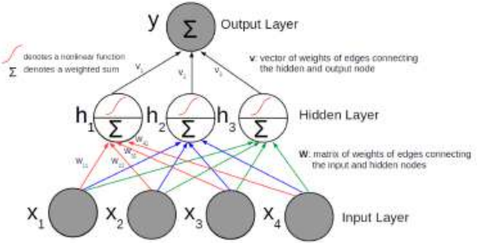


Figure: $y = V^T \sigma(W^T x + b)$.

- Each hidden node computes a nonlinear transformation of its incoming inputs
  - Weighted linear combination followed by a nonlinear activation function σ
  - Nonlinearity required by the nonlinear problem.

Models for each layer

- Below: FNN with 4 inputs, one hiddel layer with 3 nodes, and 1 output

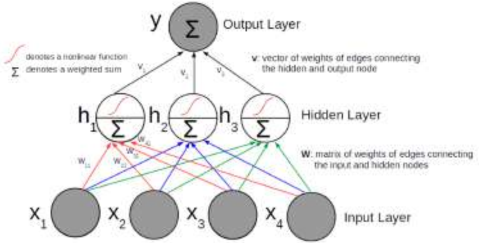
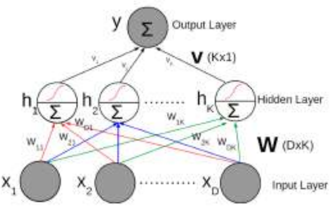


Figure: $y = V^T \sigma(W^T x + b)$.

- Each hidden node computes a nonlinear transformation of its incoming inputs
  - Weighted linear combination followed by a nonlinear activation function σ
  - Nonlinearity required by the nonlinear problem.
  - Output y is a linear transformation of the hidden nodes.

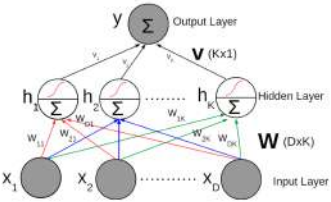
Models for each layer

- Below: A general 2 layer FNN



Models for each layer

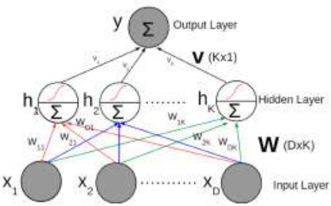
- Below: A general 2 layer FNN



- For an FNN with D inputs $\mathbf{x} = [x_1, \ldots, x_D] \in \mathbb{R}^D$

Models for each layer

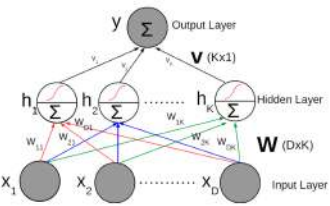
- Below: A general 2 layer FNN



- For an FNN with D inputs $\mathbf{x} = [x_1, \ldots, x_D] \in \mathbb{R}^D$, a single layer with K hidden nodes $\mathbf{h} = [h_1, \ldots, h_K]$

Models for each layer

- Below: A general 2 layer FNN

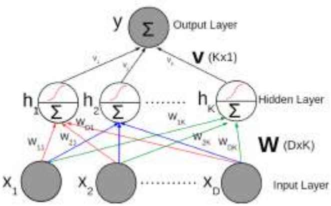


- For an FNN with D inputs $\mathbf{x} = [x_1, \ldots, x_D] \in \mathbb{R}^D$, a single layer with K hidden nodes $\mathbf{h} = [h_1, \ldots, h_K]$, and a scalar-valued output node

$$y = \mathbf{v}^T \mathbf{h}$$

Models for each layer

- Below: A general 2 layer FNN



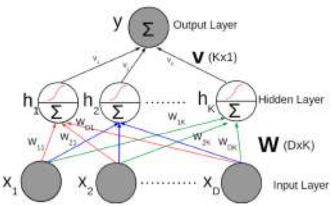
- For an FNN with D inputs $\mathbf{x} = [x_1, \ldots, x_D] \in \mathbb{R}^D$, a single layer with K hidden nodes $\mathbf{h} = [h_1, \ldots, h_K]$, and a scalar-valued output node

$$y = \mathbf{v}^T\mathbf{h} = \mathbf{v}^T\sigma(\mathbf{W}^T\mathbf{x} + b),$$

where $\mathbf{v} = [v_1, \ldots, v_K] \in \mathbb{R}^K$,

Models for each layer

- Below: A general 2 layer FNN



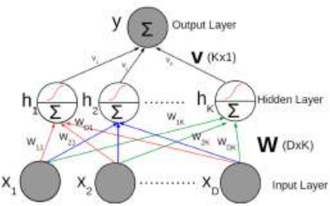
- For an FNN with D inputs $\mathbf{x} = [x_1, \ldots, x_D] \in \mathbb{R}^D$, a single layer with K hidden nodes $\mathbf{h} = [h_1, \ldots, h_K]$, and a scalar-valued output node

$$y = \mathbf{v}^T \mathbf{h} = \mathbf{v}^T \sigma(\mathbf{W}^T \mathbf{x} + b),$$

where $\mathbf{v} = [v_1, \ldots, v_K] \in \mathbb{R}^K$, $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_K] \in \mathbb{R}^{D \times K}$,

Models for each layer

- Below: A general 2 layer FNN



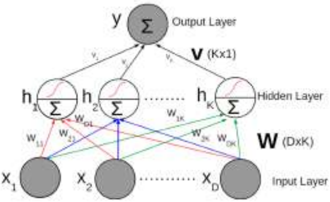
- For an FNN with D inputs $\mathbf{x} = [x_1, \ldots, x_D] \in \mathbb{R}^D$, a single layer with K hidden nodes $\mathbf{h} = [h_1, \ldots, h_K]$, and a scalar-valued output node

$$y = \mathbf{v}^T \mathbf{h} = \mathbf{v}^T \sigma(\mathbf{W}^T \mathbf{x} + b),$$

where $\mathbf{v} = [v_1, \ldots, v_K] \in \mathbb{R}^K$, $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_K] \in \mathbb{R}^{D \times K}$, σ is the nonlinear activation function.

Models for each layer

- Below: A general 2 layer FNN



- For an FNN with D inputs $\mathbf{x} = [x_1, \ldots, x_D] \in \mathbb{R}^D$, a single layer with K hidden nodes $\mathbf{h} = [h_1, \ldots, h_K]$, and a scalar-valued output node

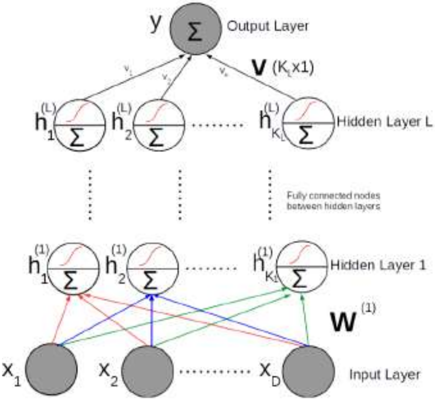
$$y = \mathbf{v}^T \mathbf{h} = \mathbf{v}^T \sigma(\mathbf{W}^T \mathbf{x} + b),$$

where $\mathbf{v} = [v_1, \ldots, v_K] \in \mathbb{R}^K$, $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_K] \in \mathbb{R}^{D \times K}$, σ is the nonlinear activation function.

- Each hidden node has a value $h_k = \sigma(w_k^T x + b_k) = \sigma(\sum_{d=1}^{D} w_{dk} x_d + b_k)$.

Deep Feedforward Neural Network

- Feed forward neural net with L hidden layers $\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}, \ldots, \boldsymbol{h}^{(L)}$, where $\boldsymbol{h}^{(1)} = \sigma(\boldsymbol{W}^{(1)^T}\boldsymbol{x} + \boldsymbol{b}^{(1)})$ and $\boldsymbol{h}^{(\ell)} = \sigma(\boldsymbol{W}^{(\ell)^T}\boldsymbol{h}^{(\ell-1)} + \boldsymbol{b}^{(\ell)})$ for $\ell \geq 2$.



- Note: The ℓ-th hidden layer contains K_ℓ hidden nodes, $\boldsymbol{W}^{(1)}$ is of size $D \times K_1$, $\boldsymbol{W}^{(\ell)}$ is of size $K_\ell \times K_{\ell+1}$, $\boldsymbol{v}$ is of size $K_L \times 1$.

Why deep?

- Recall the kernel SVM

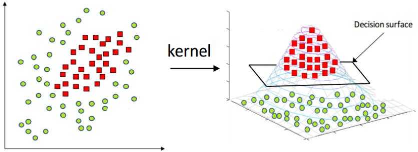


Figure: $y = f(x; w_1, w_2, b_1, b_2, \phi) = w_2^T \phi(w_1^T x + b_1) + b_2$.

Why deep?

- Recall the kernel SVM

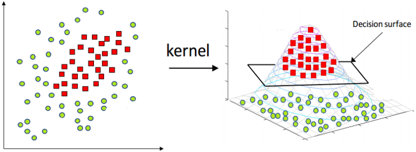


Figure: $y = f(x; w_1, w_2, b_1, b_2, \phi) = w_2^T \phi(w_1^T x + b_1) + b_2$.

- Question: how to find the kernel $\phi(x)$ automatically?

Why deep?

- Recall the kernel SVM

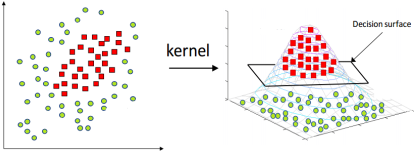
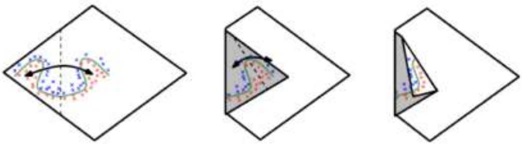


Figure: $y = f(x; w_1, w_2, b_1, b_2, \phi) = w_2^T \phi(w_1^T x + b_1) + b_2$.

- Question: how to find the kernel $\phi(x)$ automatically?
- Use the composition of simple NN's with a simple nonlinearity $|x|$



Why deep?

- Recall the kernel SVM

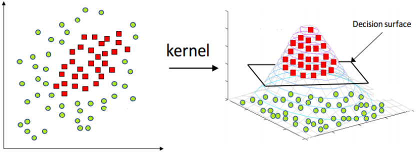
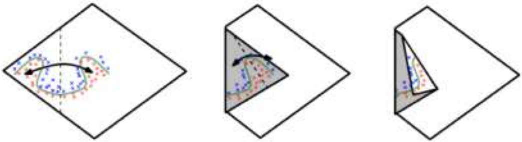


Figure: $y = f(x; w_1, w_2, b_1, b_2, \phi) = w_2^T \phi(w_1^T x + b_1) + b_2$.

- Question: how to find the kernel $\phi(x)$ automatically?
- Use the composition of simple NN's with a simple nonlinearity $|x|$



- Identifying the best $\phi(x)$ is equivalent to optimizing the coefficients of deep NN (DNN).

Why deep? To be discussed later.

Open Problems for Research

- Approximation theory;
- Optimization;
- Generalization error.

- Overview of Deep Feedforward Networks
- Architecture Design
- Back-Propagation

Gradient-Based Learning

Recall typical tasks of machine learning

- Learn a map from an input x to an output y;
- Denote the map as $y = f(x)$;
- Construct a family of DNN's $\{f(x;\theta)\}_\theta$;
- Create some criteria to quantify how good a map is;
- Use optimization to find the best $f(x;\theta)$.

Gradient-Based Learning

Recall typical tasks of machine learning

- Learn a map from an input x to an output y;
- Denote the map as $y = f(x)$;
- Construct a family of DNN's $\{f(x;\theta)\}_\theta$;
- Create some criteria to quantify how good a map is;
- Use optimization to find the best $f(x;\theta)$.

Gradient-Based Learning

Recall typical tasks of machine learning

- Learn a map from an input x to an output y;
- Denote the map as $y = f(x)$;
- Construct a family of DNN's $\{f(x;\theta)\}_\theta$;
- Create some criteria to quantify how good a map is;
- Use optimization to find the best $f(x;\theta)$.

Principles

- The computation of gradient is fast;

Gradient-Based Learning

Recall typical tasks of machine learning

- Learn a map from an input x to an output y;
- Denote the map as $y = f(x)$;
- Construct a family of DNN's $\{f(x;\theta)\}_\theta$;
- Create some criteria to quantify how good a map is;
- Use optimization to find the best $f(x;\theta)$.

Principles

- The computation of gradient is fast;
- Gradient descent can quickly decrease the energy;

Gradient-Based Learning

Recall typical tasks of machine learning

- Learn a map from an input x to an output y;
- Denote the map as $y = f(x)$;
- Construct a family of DNN's $\{f(x;\theta)\}_\theta$;
- Create some criteria to quantify how good a map is;
- Use optimization to find the best $f(x;\theta)$.

Principles

- The computation of gradient is fast;
- Gradient descent can quickly decrease the energy;
- Local minimizer is reasonably good;

Typical cost functions

Learning conditional distributions

- We often maximize the likelyhood function to obtain a distribution that best matches given data:

$$J(\theta) = -\mathbb{E}_{\mathrm{x},\mathrm{y}\sim\hat{p}_{data}} \log p^{\theta}_{model}(\mathbf{y}|\mathbf{x}).$$

Typical cost functions

Learning conditional distributions

- We often maximize the likelyhood function to obtain a distribution that best matches given data:

$$J(\theta) = -\mathbb{E}_{\mathrm{x},\mathrm{y}\sim\hat{p}_{data}} \log p^{\theta}_{model}(\mathbf{y}|\mathbf{x}).$$

- Specifying a model $p^{\theta}(\mathbf{y}|\mathbf{x})$ automatically determines a cost function.

Typical cost functions

Learning conditional distributions

- We often maximize the likelyhood function to obtain a distribution that best matches given data:

$$J(\theta) = -\mathbb{E}_{\mathrm{x},\mathrm{y}\sim\hat{p}_{data}} \log p^{\theta}_{model}(\mathbf{y}|\mathbf{x}).$$

- Specifying a model $p^{\theta}(\mathbf{y}|\mathbf{x})$ automatically determines a cost function.
- When we use an DNN to specify the disribution function, say

$$p^{\theta}(\mathbf{y}|\mathbf{x}) = f(\mathbf{y}; \mathbf{x}, \theta),$$

we need to choose the output and hidden units carefully so that the objective function is easy to optimize.

Typical cost functions

Example 1: Mean squared error

- Assumption: $y = f(\mathbf{x};\theta) + \omega$, where ω is a Gaussian random noise.
- Model: $p^{\theta}_{model}(\mathbf{y}|\mathbf{x}) = exp(-\frac{\|\mathbf{y}-f(\mathbf{x};\theta)\|_2^2}{2\sigma^2})$.
- Loss function in the MLE framework:

$$\begin{aligned} J(\theta) &= -\mathbb{E}_{\mathrm{x},\mathrm{y}\sim\hat{p}_{data}} \log p^{\theta}_{model}(\mathbf{y}|\mathbf{x}) \\ &= \mathbb{E}_{\mathrm{x},\mathrm{y}\sim\hat{p}_{data}} \frac{\|\mathbf{y}-f(\mathbf{x};\theta)\|_2^2}{2\sigma^2} \\ &= \frac{1}{N}\sum_{i=1}^{N} \frac{\|\mathbf{y}_i-f(\mathbf{x}_i;\theta)\|_2^2}{2\sigma^2} \end{aligned}$$

- Optimization problem:

$$\min_{\theta} \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{y}_i - f(\mathbf{x}_i;\theta)\|_2^2$$

Typical cost functions

Example 2: Mean absolute error

- Assumption: $y = f(\mathbf{x};\theta) + \omega$, where ω is a Laplace random noise.
- Model: $p^{\theta}_{model}(\mathbf{y}|\mathbf{x}) = \frac{1}{2b} exp(-\frac{\|\mathbf{y}-f(\mathbf{x};\theta)\|_1}{b})$.
- Loss function in the MLE framework:

$$\begin{aligned} J(\theta) &= -\mathbb{E}_{\mathrm{x},\mathrm{y}\sim\hat{p}_{data}} \log p^{\theta}_{model}(\mathbf{y}|\mathbf{x}) \\ &= \mathbb{E}_{\mathrm{x},\mathrm{y}\sim\hat{p}_{data}} \frac{\|\mathbf{y}-f(\mathbf{x};\theta)\|_1}{b} + \log(2b) \\ &= \frac{1}{N}\sum_{i=1}^{N} \frac{\|\mathbf{y}_i - f(\mathbf{x}_i;\theta)\|_1}{b} + \log(2b) \end{aligned}$$

- Optimization problem:

$$\min_{\theta} \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{y}_i - f(\mathbf{x}_i;\theta)\|_1$$

Typical cost functions

Example 3: Mean squared log error

- Assumption: $\log y = \log f(\mathbf{x};\theta) + \omega$, where ω is a Gaussian random noise.
- Model: $p^{\theta}_{model}(\log \mathbf{y}|\mathbf{x}) = exp(-\frac{\|\log \mathbf{y} - \log f(\mathbf{x};\theta)\|_2^2}{2\sigma^2})$.
- Loss function in the MLE framework:

$$
\begin{aligned}
J(\theta) &= -\mathbb{E}_{\mathrm{x},\log \mathrm{y}\sim \hat{p}_{data}} \log p^{\theta}_{model}(\log \mathbf{y}|\mathbf{x}) \\
&= \mathbb{E}_{\mathrm{x},\log \mathrm{y}\sim \hat{p}_{data}} \frac{\|\log \mathbf{y} - \log f(\mathbf{x};\theta)\|_2^2}{2\sigma^2} \\
&= \frac{1}{N}\sum_{i=1}^{N} \frac{\|\log \mathbf{y}_i - \log f(\mathbf{x}_i;\theta)\|_2^2}{2\sigma^2}
\end{aligned}
$$

- Optimization problem:

$$
\min_{\theta} \frac{1}{N}\sum_{i=1}^{N} \|\log \mathbf{y}_i - \log f(\mathbf{x}_i;\theta)\|_2^2
$$

Typical cost functions

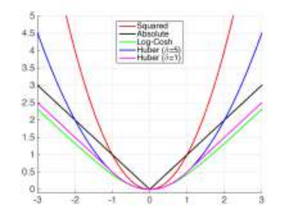
MSE vs. MSLE

We use MSLE instead of MSE when

- when you don't want to penalize huge differences in the predicted and the actual values when both predicted and true values are huge numbers.
- when you want to penalize under estimates more than over estimates. For example: Pi as predicted value, Ai as actual value,
  - Pi = 600, Ai = 1000, then
    RMSE = 400, RMSLE = 0.5108
  - Pi = 1400, Ai = 1000, then
    RMSE = 400, RMSLE = 0.3365

Typical cost functions

Other loss functions



- Cross entropy (classification);
- KL divergence (distribution);
- EMD (distribution);
- Huber (regression);
- (Squared) Hinge (classification).

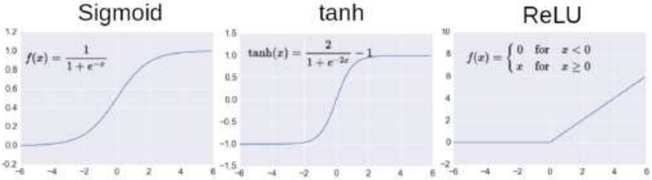
Reference: On Loss Functions for Deep Neural Networks in Classification `https://arxiv.org/abs/1702.05659`.

Choices of Units

- The choice of cost function is tightly coupled with the choice of output and hidden units

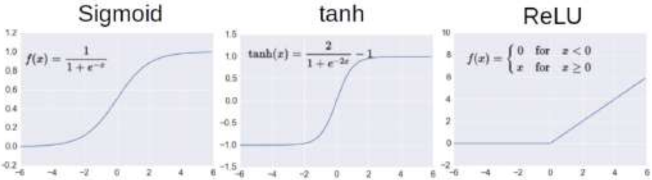
Choices of Units

- The choice of cost function is tightly coupled with the choice of output and hidden units
- Nonlinear activation functions:
  - Sigmoid: $f(x) = \sigma(x) = \frac{1}{1+\exp(-x)}$ (range between 0-1)
  - tanh: $f(x) = 2\sigma(2x) - 1$ (range between -1 and +1)
  - Rectified Linear Unit (ReLU): $f(x) = \max(0, x)$



Choices of Units

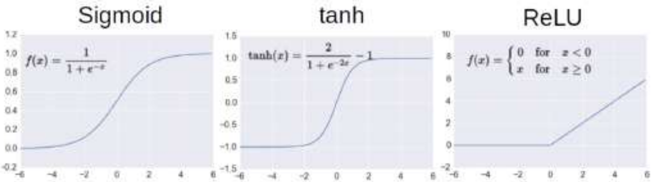
- The choice of cost function is tightly coupled with the choice of output and hidden units
- Nonlinear activation functions:
  - Sigmoid: $f(x) = \sigma(x) = \frac{1}{1+\exp(-x)}$ (range between 0-1)
  - tanh: $f(x) = 2\sigma(2x) - 1$ (range between -1 and +1)
  - Rectified Linear Unit (ReLU): $f(x) = \max(0, x)$



- Signoid saturates and can kill gradients

Choices of Units

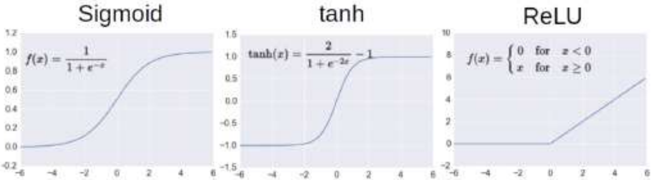
- The choice of cost function is tightly coupled with the choice of output and hidden units
- Nonlinear activation functions:
    - Sigmoid: $f(x) = \sigma(x) = \frac{1}{1+\exp(-x)}$ (range between 0-1)
    - tanh: $f(x) = 2\sigma(2x) - 1$ (range between -1 and +1)
    - Rectified Linear Unit (ReLU): $f(x) = \max(0, x)$



- Signoid saturates and can kill gradients
- tanh also saturates but is steeper around the center (thus preferred over sigmoid)

Choices of Units

- The choice of cost function is tightly coupled with the choice of output and hidden units
- Nonlinear activation functions:
  - Sigmoid: $f(x) = \sigma(x) = \frac{1}{1+\exp(-x)}$ (range between 0-1)
  - tanh: $f(x) = 2\sigma(2x) - 1$ (range between -1 and +1)
  - Rectified Linear Unit (ReLU): $f(x) = \max(0, x)$

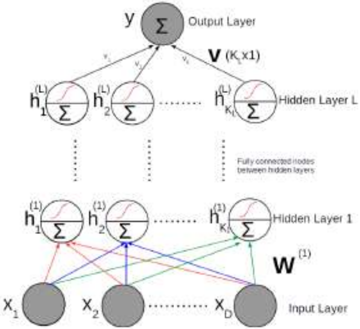


- Signoid saturates and can kill gradients
- tanh also saturates but is steeper around the center (thus preferred over sigmoid)
- ReLU is currently the most popular (also cheap to compute), leading to piecewise linear functions.

- Overview of Deep Feedforward Networks
- Architecture Design
- Back-Propagation

Back-Propagation

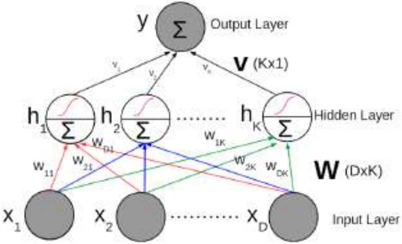
- Want to learn the parameters by minimizing some loss function



- Backpropagation (gradient descent + chain rule for derivatives) is commonly used to do this efficiently

Back-Propagation

- Consider the feedforward neural net with one hidden layer



- Recall that $\boldsymbol{h} = [h_1 \ h_2 \ \dots \ h_K] = f(\mathbf{W}^\top \boldsymbol{x})$
- Assuming a regression problem, the optimization problem would be

$$\min_{\mathbf{W},\boldsymbol{v}} \frac{1}{2} \sum_{n=1}^{N} \left(y_n - \boldsymbol{v}^\top f(\mathbf{W}^\top \boldsymbol{x}_n)\right)^2 = \min_{\mathbf{W},\boldsymbol{v}} \frac{1}{2} \sum_{n=1}^{N} \left(y_n - \sum_{k=1}^{K} v_k f(\boldsymbol{w}_k^\top \boldsymbol{x}_n)\right)^2$$

where $\boldsymbol{w}_k$ is the k-th column of the $D \times K$ matrix $\mathbf{W}$

Back-Propagation

- We can learn the parameters by doing gradient descent (or stochastic gradient descent) on the objective function

$$\mathcal{L} = \frac{1}{2}\sum_{n=1}^{N}\left(y_n - \sum_{k=1}^{K} v_k f(\boldsymbol{w}_k^\top \boldsymbol{x}_n)\right)^2 = \frac{1}{2}\sum_{n=1}^{N}\left(y_n - \boldsymbol{v}^\top \boldsymbol{h}_n\right)^2$$

- Gradient w.r.t. $\boldsymbol{v} = [v_1\ v_2\ \ldots\ v_K]$ is straightforward

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}} = -\sum_{n=1}^{N}\left(y_n - \sum_{k=1}^{K} v_k f(\boldsymbol{w}_k^\top \boldsymbol{x}_n)\right)\boldsymbol{h}_n = -\sum_{n=1}^{N} \boldsymbol{e}_n \boldsymbol{h}_n$$

- Gradient w.r.t. the weights $\mathbf{W} = [\boldsymbol{w}_1\ \boldsymbol{w}_2\ \ldots\ \boldsymbol{w}_K]$ is a bit more involved due to the presence of f but can be computed using chain rule

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_k} = \frac{\partial \mathcal{L}}{\partial f_k}\frac{\partial f_k}{\partial \boldsymbol{w}_k} \qquad \text{(note: } f_k = f(\boldsymbol{w}_k^\top \boldsymbol{x})\text{)}$$

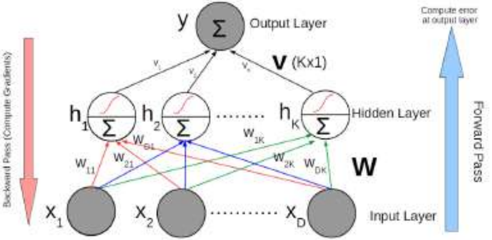
- We have: $\frac{\partial \mathcal{L}}{\partial f_k} = -\sum_{n=1}^{N}(y_n - \sum_{k=1}^{K} v_k f(\boldsymbol{w}_k^\top \boldsymbol{x}_n))v_k = -\sum_{n=1}^{N} \boldsymbol{e}_n v_k$

- We have: $\frac{\partial f_k}{\partial \boldsymbol{w}_k} = \sum_{n=1}^{N} f'(\boldsymbol{w}_k^\top \boldsymbol{x}_n)\boldsymbol{x}_n$, where $f'(\boldsymbol{w}_k^\top \boldsymbol{x}_n)$ is f's derivative at $\boldsymbol{w}_k^\top \boldsymbol{x}_n$

- These calculations can be done efficiently using backpropagation

Back-Propagation

- Basically consists of a forward pass and a backward pass



- Forward pass computes the errors $\boldsymbol{e}_n$ using the current parameters
- Backward pass computes the gradients and updates the parameters, starting from the parameters at the top layer and then moving backwards
- Also good at reusing previous computations (updates of parameters at any layer depends on parameters at the layer above)