Deep Learning I: Feed-forward Networks

CSci 5525: Machine Learning

Instructor: Nicholas Johnson

October 13, 2020

Announcements

- HW2 is due in 2 days (due Oct 15 at 11:59 PM CDT)
- Exam 1 will be posted on Oct 20 (due 48 hours later)
- Course feedback form
- Project proposal grades/feedback posted

Kernels and Neural Networks

- Kernel functions implicitly lift raw feature vectors $\mathbf{x}$ to expanded feature vectors $\Phi(\mathbf{x})$
- Kernel trick allows us to learn non-linear boundaries when predicting with $\mathbf{w}^\top \Phi(\mathbf{x})$
- Mapping $\Phi(\cdot)$ fixed once hyperparameters are chosen
- Here we consider neural networks (i.e., multi-layered perceptrons)

Neural Networks

- Neural networks explicitly learn feature mapping $\Phi(\mathbf{x})$
- Mappings are learned via composition of linear functions
- Predictions are of the form $h(\mathbf{w}^\top \Phi(\mathbf{x}))$
- $h(\cdot)$ is an activation function

Composition of Linear Functions

- Let $x \in \mathbb{R}^p$
- First linear transformation: $x \to W_1 x + b_1$
- Second linear transformation: $x \to W_2(W_1 x + b_1) + b_2$
- Lth linear transformation: $x \to W_L(\ldots(W_1 x + b_1)\ldots) + b_L$
- What do we gain with this composition?

Composition of Linear Functions

- What do we gain with this composition? Nothing!
- Observe

$$W_L(\dots(W_1 x + b_1)\dots) + b_L = Wx + b$$

where $W = W_L \dots W_1$ and $b = b_L + W_L b_{L-1} + \dots W_2 b_1$

Non-linear Activations

- Need to introduce non-linearity between linear functions
- Recall in logistic regression we have

$\in \mathbb{R}$

$$\sigma(\overset{1}{z}) = \frac{1}{1+e^{-z}}$$

$$\sigma(\vec{v}) = \begin{array}{c} \begin{bmatrix} \\ \frac{1}{1+e^{-v_i}} \\ \\ \end{bmatrix} \end{array} \begin{array}{l} \leftarrow 1 \\ \leftarrow i \\ \leftarrow p \end{array}$$

$$P(y|x) = \frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x})} = \sigma(\mathbf{w}^\top \mathbf{x})$$

- Now consider vector-valued function that applies sigmoid coordinate-wise

$\vec{v} \in \mathbb{R}^p$

$$\mathbb{R}^p \ni f_i(z) = \sigma(W_i z + b_i)$$

- Composition of such functions gives basic feed-forward neural network

$$x \to (f_L \circ \overset{f_2}{\cdots} \circ f_1)(x) \qquad \text{where} \quad f_i(z) = \sigma(W_i z + b_i)$$

- Weights: $W_i \; \forall i$
- Biases: $b_i \; \forall i$

Basic Neural Network Form

- Given activation functions $\{h_i\}_{i=1}^{L}$, weights, and biases:

data

$$\longrightarrow \quad F(x, \theta) = h_L(W_L(\ldots W_2 \underbrace{h_1(W_1 x + b_1)}_{z} + b_2 \ldots) + b_L)$$

where θ denotes set of weights and biases

$$x \to z \longrightarrow z'$$

raw features

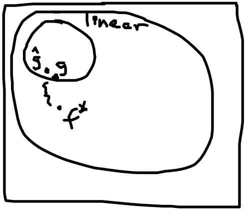
new features

new new features

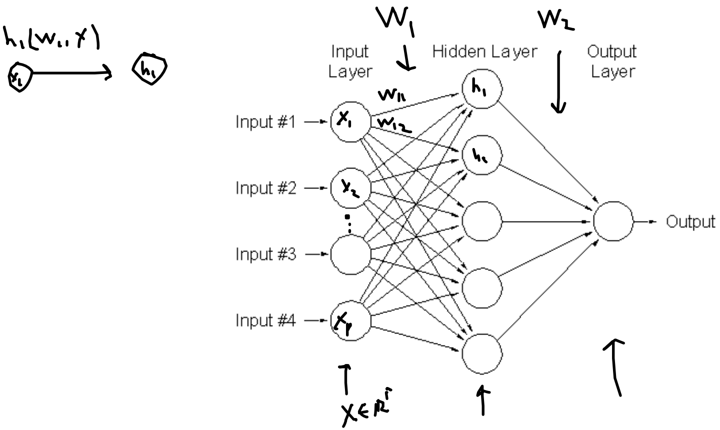
$$h_2(W_2 z + b)$$

Layered Linear Classifier

- Motivation
  - Linear classifiers have “limited” representation power
  - Most real problems have non-linear boundaries
- Feed-forward networks are layered linear classifiers
  - Use multiple layers of linear classifiers
  - Non-linear connections between layers
- Significantly higher representation power
  - Less “bias” compared to linear classifiers
- Training is challenging
  - Less bias usually implies more training data
  - Needs proper regularization
- Other types of deep networks
  - Recurrent networks, Deep Boltzman machines, Auto-encoders



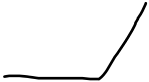
Example Feed-forward Network



- Each hidden layer has a non-linear activation function
  - tanh, sigmoid, ReLU, etc.
- Output layer has a non-linear activation function
  - softmax

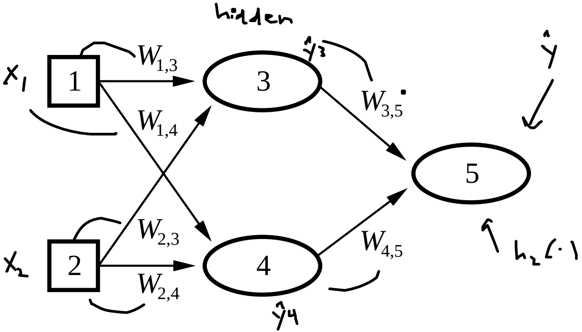
Activation functions

- Hidden layer output $\mathbf{h}(\mathbf{a})$, where $\mathbf{a} = W\mathbf{x} + b$
  - Typically applied elementwise: $h^i(a_i)$
- Hyperbolic tangent: $h(a) = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$



- Sigmoid: $h(a) = \sigma(a) = \frac{1}{1+e^{-a}}$
- Rectifier or rectified linear unit (ReLU): $h(a) = \max(0, a)$
- Softmax: $h(a) = \sigma(a) = \left[\frac{e^{a_i}}{\sum_j e^{a_j}} \right]$ (mostly for output)
- Others, e.g., leaky ReLU, softplus, hard tanh, etc.

Feed-forward Network



A parameterized family of nonlinear functions:

pred.

$$
\begin{aligned}
\hat{y}_5 &= h_2(\overline{w_{3,5} \cdot \hat{y}_3 + w_{4,5} \cdot \hat{y}_4}) \\
&= h_2(w_{3,5} \cdot \underline{h_1(w_{1,3} \cdot x_1 + w_{2,3} \cdot x_2)} \\
&\quad + w_{4,5} \cdot h_1\underline{(w_{1,4} \cdot x_1 + w_{2,4} \cdot x_2)})
\end{aligned}
$$

Adjusting weights changes the function

Expressiveness

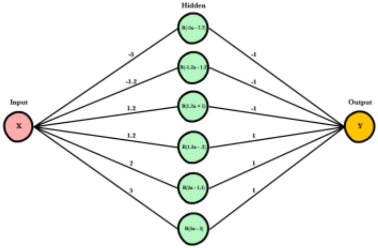
- Just one non-linear hidden layer provides more representation power than linear function

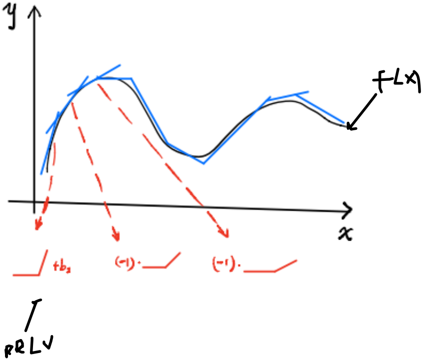
  Universal Approximation Theorem. Let $f : \mathbb{R}^p \to \mathbb{R}$ be any continuous function. For any approximation error $\epsilon > 0$, there exists a set of parameters $\theta = (W_1, b_1, W_2, b_2)$ such that for any $x \in [0, 1]^p$

$$|f(x) - (W_2 h(W_1 x + b_1) + b_2)| \leq \epsilon$$

  where $h(\cdot)$ is a nonconstant, bounded, and continuous function (e.g., ReLU and logistic).

- In other words, a single hidden layer neural network can approximate any continuous function to any degree of precision


Hidden
Input
Output
X
Y
-5
-1.2
1.2
1.2
2
5
-1
-1
-1
1
1
1

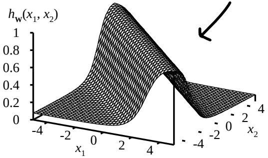

y
x
f(x)
(-1)·
(-1)·
ReLU

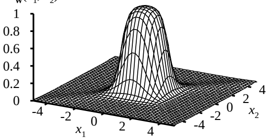
Expressiveness

- Analogous universal approximation theorem with deep neural networks with bounded widths
- Such neural networks can be very wide/deep and may be hard to find

Expressiveness

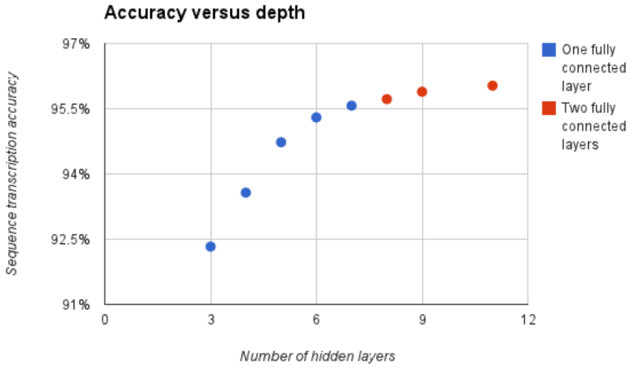
- Multi-Layer Perceptrons can express
  - All continuous functions with 2 layers
  - All functions with 3 layers





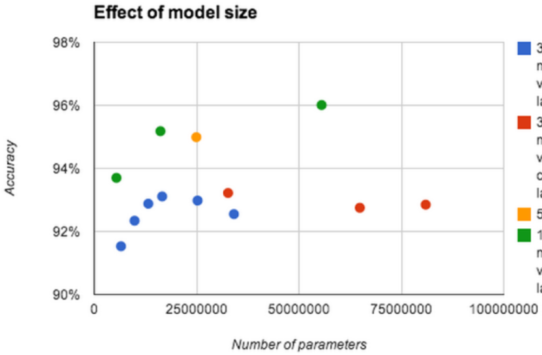
- Combine two opposite-facing functions to make a ridge
- Combine two perpendicular ridges to make a bump
- Add bumps of various sizes and locations to fit any surface
- Proof requires exponentially many hidden units

Deep networks: Effect of depth



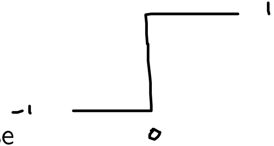
Generalization (test set performance) seems to improve with depth

Deep networks: Effect of size



Generalization (test set performance) has no clear improvement with size, e.g., width

Perceptrons Revisited



- Linear classifiers with weight vector $\mathbf{w}$
- Labels are encoded as $y_i \in \{-1, 1\}$
- Predict $y = 1$ if $\mathbf{w}^\top \mathbf{x} > 0$ and -1 otherwise
- Prediction on $\mathbf{x}_i$ is incorrect if $y_i \mathbf{w}^\top \mathbf{x}_i < 0$
- Let $\mathcal{M}(\mathbf{w})$ be the set of points on which prediction is incorrect
- The objective function to be minimized

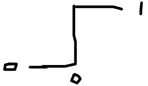
$$E(\mathbf{w}) = -\sum_{i \in \mathcal{M}(\mathbf{w})} y_i \mathbf{w}^\top \mathbf{x}_i$$

$$\min_{w} E(w)$$

- For any point $i \in \mathcal{M}(\mathbf{w})$, gradient $\nabla E_i(\mathbf{w}) = -y_i \mathbf{x}_i$
- The gradient based update

$$\mathbf{w}_{(new)} = \mathbf{w}_{(old)} - \eta \nabla E_i(\mathbf{w}) = \mathbf{w}_{(old)} + \eta y_i \mathbf{x}_i$$

Perceptrons Revisited



- Define labels as $y \in \{0, 1\}$
- Consider $h(\cdot)$ as the hard threshold activation function
- Predict $y = h(\mathbf{w}^\top \mathbf{x})$ where $h(\mathbf{w}^\top \mathbf{x}) = 1$ if $\mathbf{w}^\top \mathbf{x} > 0$ and $\overset{0}{-1}$ o/w
- Consider squared error for an example $(\mathbf{x}, y)$ is

$$\min_w E \longrightarrow \quad E = \frac{1}{2}\left(y - h\left(\overbrace{\sum_j w_j x_j}^{w^T x}\right)\right)^2 = \frac{1}{2}(y - \hat{y})^2 \, ,$$

- Perform optimization by gradient descent:

$$\begin{aligned} \frac{\partial E}{\partial w_j} &= -(y - \hat{y}) \times \frac{\partial}{\partial w_j}\left(h\left(\sum_j w_j x_j\right)\right) \\ &= -(y - \hat{y}) \times h'(a) \times x_j \end{aligned}$$

- Simple weight update rule:

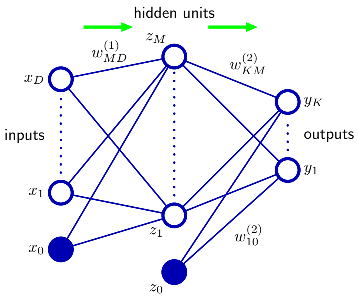
$$w_j \leftarrow w_j + \alpha \times \overbrace{(y - \hat{y}) \times h'(a) \times x_j}^{\nabla E}$$

Logistic Regression Revisited

- One problem is $h(\cdot)$ is not differentiable (gradient is zero almost everywhere)
- Fix by softening the threshold function $h(\cdot)$
- Let $h(\cdot) = \sigma(\cdot)$ the logistic function
    - Sometimes called sigmoid perceptron
- We can view this as logistic regression with squared loss
- Compute optimal $\mathbf{w}$ via gradient descent to get update:

$$w_j \leftarrow w_j + \alpha \times (y - \hat{y}) \times h'(a) \times x_j$$

The Picture: General



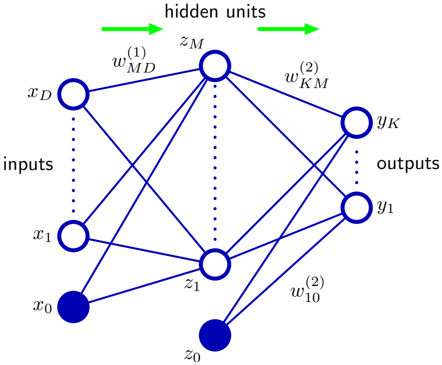
- Between input and hidden layers

$$w^{(1)}_{ji} \qquad a_j = \sum_{i=0}^{D} w_{ji} x_i \qquad z_j = h(a_j)$$

- Between hidden and output layers

$$w^{(2)}_{kj} \qquad a_k = \sum_{j=0}^{M} w_{kj} z_j \qquad \hat{y}_k = h(a_k)$$

The Picture: General



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