Deep Learning - COSC2779/2972

Deep Feed Forward Networks

Dr. Ruwan Tennakoon

RMIT UNIVERSITY

Semester 2, 2022

Reference: *Chapter 6: Ian Goodfellow et. al., "Deep Learning", MIT Press, 2016.*

Outline

Part 1: Deep Feed Forward Networks

The **Task** can be expressed an unknown target function:

$$\mathbf{y} = f\left(\mathbf{x}\right)$$

ML finds a Hypothesis (model), $h\left(\cdot\right)$, from hypothesis space $\mathcal{H}$, which approximates the unknown target function.

$$\hat{\mathbf{y}} = h^{*}\left(\mathbf{x}\right) \approx f\left(\mathbf{x}\right)$$

The **Experience** is typically a data set, $\mathcal{D}$, of values

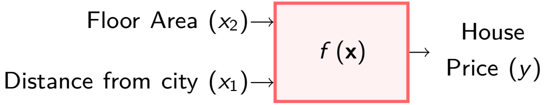
$$\mathcal{D} = \left\{\left(\mathbf{x}^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right)\right\}_{i=1}^{N}$$

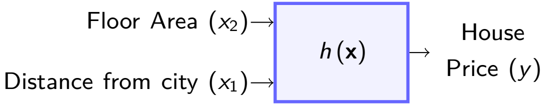
$*$Assume supervised learning for now

The **Performance** is typically numerical measure that determines how well the hypothesis matches the experience.

How is the Hypothesis (model), $h\left(\cdot\right)$ represented?

Example: Linear Regression





Hypothesis (Model):

$$\hat{y}^{(i)} = h\left(\mathbf{x}^{(i)}\right) = w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)}$$

Hypothesis space: $\mathcal{H} \in$
All possible combinations of (w_0, w_1, w_2)

What are the other methods we can use to represent $h(\mathbf{x})$?

- Tree (regression/classification)
- Rules
- Neural networks
- $\cdots$

Objectives for this Lecture

- Explore the elements used in representing the hypothesis space of a feed-forward neural network.
- Understand the applicable techniques so that we can identify the "best hypothesis space for a problem" in a way that is better than random search of all the possible combinations (not feasible).
- Gain the ability to justify your model to others.

Outline

Perceptron

Neural Networks are inspired by the structure of the human brain. The basic building block of the brain is a neuron.

A Neuron is formed of:

- A series of incoming synapses
- An activation cell
- A single outgoing synapse that connects to other Neurons.

A Neuron is modelled as a Perceptron (Rosenblatt 1962)

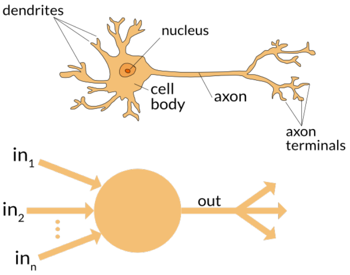


Image: https://appliedgo.net/perceptron/

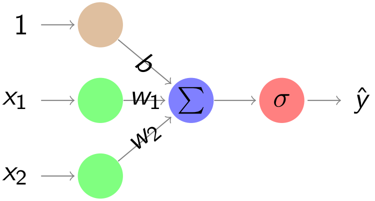
Reading: The nature of code - chapter 10

Neural Networks are inspired by the structure of the human brain. The basic building block of the brain is a neuron.

A Neuron is formed of:

- A series of incoming synapses
- An activation cell
- A single outgoing synapse that connects to other Neurons.

A Neuron is modelled as a Perceptron (Rosenblatt 1962)

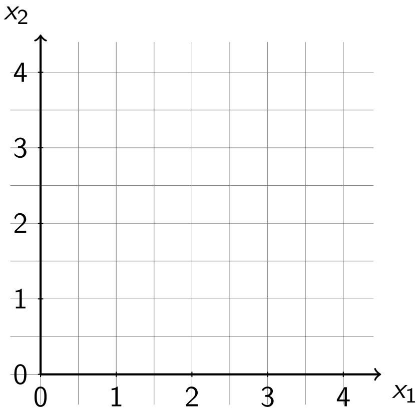


$$\hat{y} = \sigma\left(\mathbf{w}^\top \mathbf{x} + b\right)$$

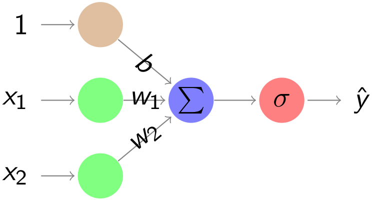
$$\sigma\left(\mathbf{w}^\top \mathbf{x} + b\right) = \frac{1}{1 + \exp\left(-\mathbf{w}^\top \mathbf{x} + b\right)}$$

With Sigmoid activation the basic perceptron is similar to logistics regression.

How to find the weights w, b?



$$\hat{y} = \sigma\left(w_1 x_1 + w_2 x_2 + b\right)$$
$$w_1 x_1 + w_2 x_2 + b = 0$$



$$\hat{y} = \sigma\left(\mathbf{w}^\top \mathbf{x} + b\right)$$

$$\sigma\left(\mathbf{w}^\top \mathbf{x} + b\right) = \frac{1}{1 + \exp\left(-\mathbf{w}^\top \mathbf{x} + b\right)}$$

With Sigmoid activation the basic perceptron is similar to logistics regression.

How to find the weights w, b?

Outline

Aside: Maximum Likelihood Estimation

$\mathcal{D} = \left\{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(n)}\right\}$: Set of data drawn independently from the unknown data-generating distribution p_{data}.

$p_{model}(\mathbf{x}; \theta)$: Family of distributions parameterize by θ.

We want to find θ that best matches with the observations (data $\mathcal{D}$). Or we want to find θ that maximize $p(\theta \mid \mathcal{D})$.

Likelihood Prior

$$p(\theta \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \theta) p(\theta)}{p(\mathcal{D})}$$

Posterior

Maximum Likelihood Estimation (MLE) is a method to fit a distribution to data.

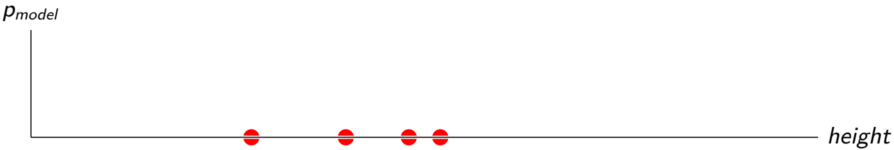
$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\ p(\mathcal{D} \mid \theta)$$

For independent data:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{N} p_{model}\left(\mathbf{x}^{(i)}; \theta\right)$$

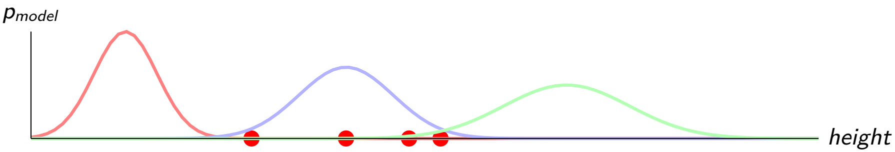
The logarithm of the likelihood does not change its argmax but makes the math convenient.

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{N} \log p_{model}\left(\mathbf{x}^{(i)}; \theta\right)$$



$$\theta_{ch} = [\mu = 0.3m, \sigma = 0.1m] \quad \theta_s = [1.0m, 0.15m] \quad \theta_c = [1.7m, 0.2m]$$

Aside: Maximum Likelihood Estimation



$$\theta_{ch} = [0.3m, 0.1m] \quad \theta_s = [1.0m, 0.15m] \quad \theta_c = [1.7m, 0.2m]$$

- $p_{model}\left(\mathbf{x}; \theta_{ch}\right) \Rightarrow \approx 0 \times 0 \times 0 \times 0$
- $p_{model}\left(\mathbf{x}; \theta_{s}\right) \Rightarrow \approx 0.02 \times 0.3 \times 0.15 \times 0.01$:
- $p_{model}\left(\mathbf{x}; \theta_{c}\right) \Rightarrow \approx 0 \times 0 \times 0.001 \times 0.01$:

Example only values are not accurate.

Aside: Maximum Likelihood Estimation

$\mathcal{D} = \left\{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(n)}\right\}$: Set of data drawn independently from the unknown data-generating distribution p_{data}.

$p_{model}\left(\mathbf{x}; \theta\right)$: Family of distributions parameterize by θ.

We want to find θ that best matches with the observations (data $\mathcal{D}$). Or we want to find θ that maximize $p\left(\theta \mid \mathcal{D}\right)$.

Likelihood

Prior

$$p(\theta \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \theta)\, p(\theta)}{p(\mathcal{D})}$$

Posterior

In supervised learning we have a conditional model.

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{N} \log p_{model}\left(\mathbf{x}^{(i)}; \theta\right)$$

Conditional Log-Likelihood:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{N} \log p\left(y^{(i)} \mid \mathbf{x}^{(i)}; \theta\right)$$

Take few minutes and go through the following videos to get a good understanding of MLE:

StatQuest: Maximum Likelihood, clearly explained

StatQuest: Maximum Likelihood For the Normal Distribution, step-by-step!

For the Sigmoid model we can write (y is Bernoulli RV):

$$p\left(y^{(i)} \mid \mathbf{x}^{(i)}; \mathbf{w}\right) = \begin{cases} \sigma\left(\mathbf{w}^\top \mathbf{x} + b\right) & y^{(i)} = 1 \\ 1 - \sigma\left(\mathbf{w}^\top \mathbf{x} + b\right) & y^{(i)} = 0 \end{cases}$$

$$p\left(y^{(i)} \mid \mathbf{x}^{(i)}; \mathbf{w}\right) = \left(\sigma\left(\mathbf{w}^\top \mathbf{x} + b\right)\right)^{y^{(i)}} \left(1 - \sigma\left(\mathbf{w}^\top \mathbf{x} + b\right)\right)^{\left(1 - y^{(i)}\right)}$$

- Assume you have a biased coin with $p\left(O = h\right) = 0.7$
- Then $p\left(O = t\right) = 1 - p\left(O = h\right) = 0.3$
- Assume you observed a sequence h, h, t, h, t, h. What is the likelihood that happening:
- Given coin tosses are independent: $0.7 \times 0.7 \times 0.3 \times 0.7 \times 0.3 \times 0.7$

RMIT UNIVERSITY

For the Sigmoid model we can write (y is Bernoulli RV):

$$p\left(y^{(i)} \mid \mathbf{x}^{(i)}; \mathbf{w}\right) = \begin{cases} \sigma\left(\mathbf{w}^\top \mathbf{x} + b\right) & y^{(i)} = 1 \\ 1 - \sigma\left(\mathbf{w}^\top \mathbf{x} + b\right) & y^{(i)} = 0 \end{cases}$$

$$p\left(y^{(i)} \mid \mathbf{x}^{(i)}; \mathbf{w}\right) = \left(\sigma\left(\mathbf{w}^\top \mathbf{x} + b\right)\right)^{y^{(i)}} \left(1 - \sigma\left(\mathbf{w}^\top \mathbf{x} + b\right)\right)^{\left(1 - y^{(i)}\right)}$$

The log likelihood:

$$\log p\left(y^{(i)} \mid \mathbf{x}^{(i)}; \mathbf{w}\right) = y^{(i)} \log\left(\sigma\left(\mathbf{w}^\top \mathbf{x} + b\right)\right) + \left(1 - y^{(i)}\right) \log\left(1 - \sigma\left(\mathbf{w}^\top \mathbf{x} + b\right)\right)$$

$$\log p\left(\mathbf{Y} \mid \mathbf{X}; \mathbf{w}\right) = \sum_{i=1}^{N} \left(y^{(i)} \log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)\right)$$

Loss (cost) function:

$$\mathcal{L}(\mathbf{w}) = -\log p(\mathbf{Y} \mid \mathbf{X}; \mathbf{w}) = -\sum_{i=1}^{N} \left(y^{(i)} \log \left(\hat{y}^{(i)} \right) + \left(1 - y^{(i)} \right) \log \left(1 - \hat{y}^{(i)} \right) \right)$$

$$\mathbf{w}^* = \mathrm{argmin}_{\mathbf{w}} \mathcal{L}(\mathbf{w})$$

- No closed form solution for the Maximum Likelihood for this model.
- However the error is convex.
- Gradient Descent/ascent.

Gradient Update:

$$w^{(t+1)} = w^{(t)} - \alpha_t \frac{\partial}{\partial w} \mathcal{L}(\mathbf{w})$$

1. What is the purpose of the non-linear function in Perceptron?
2. Can the Perceptron be used for regression? How?
3. Is Sigmoid-Perceptron appropriate to classify the data shown below:

4. Calculate the partial derivatives for sigmoid Perceptron $\frac{\partial}{\partial \mathbf{w}}\mathcal{L}(\mathbf{w})$.

Outline

Increasing Model Capacity

Given data with attributes $\mathbf{x} = [x_1, x_2]$, how can we increase the capacity of a linear regression model?

$$y = b + \sum_{j=1}^{d} w_j x_j$$

Increasing Model Capacity

Given data with attributes $\mathbf{x} = [x_1, x_2]$, how can we increase the capacity of a linear regression model?

$$y = b + \sum_{j=1}^{d} w_j x_j$$

- Do polynomial transformation (non-linear) on $\mathbf{x}$: $\mathbf{x} \to \phi(\mathbf{x})$.
- Fit a linear model on $\phi(\mathbf{x})$.

Are there better choices for $\phi(\cdot)$?

Increasing Model Capacity

Given data with attributes $\mathbf{x} = [x_1, x_2]$, how can we increase the capacity of a linear regression model?

$$y = b + \sum_{j=1}^{d} w_j x_j$$

- Do polynomial transformation (non-linear) on $\mathbf{x}$: $\mathbf{x} \to \phi(\mathbf{x})$.
- Fit a linear model on $\phi(\mathbf{x})$.

Are there better choices for $\phi(\cdot)$?

How to choose $\phi(\cdot)$?

- Use **generic transformations**: Radial Basis Functions (e.g. As used in SVM).
- **Hand crafted** (engineered): SIFT, HoG features in computer vision.
- **Learn from data**: Combines good points of first two approaches. $\phi(\cdot)$ can be highly generic and the engineering effort can go into architecture.

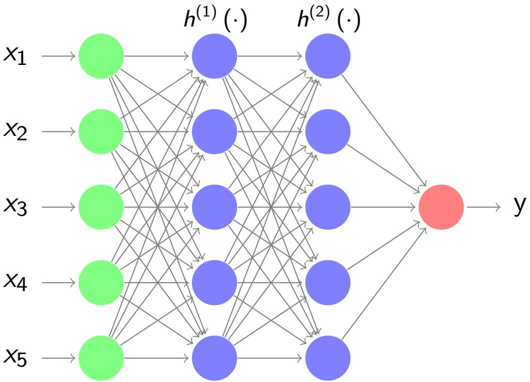
Learned transformation:

$$\phi(\mathbf{x}) := h^{(i)}\left(\mathbf{x}; \mathbf{w}^{(i)}\right)$$

The model compose of many such transformations organized in a sequence (hierarchical).

$$h(\mathbf{x}) = h^{(3)}\left(h^{(2)}\left(h^{(1)}(\mathbf{x})\right)\right)$$

Information flow in function evaluation begins at input, flows through intermediate computations, to produce the output (y).



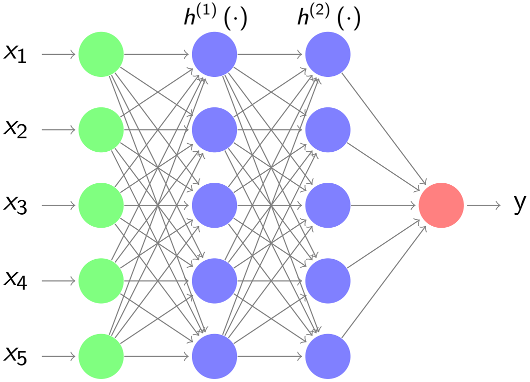
Note: All neurons have biases. For convenience they are not represented in the diagram.

$$h(\mathbf{x}) = h^{(3)}\left(h^{(2)}\left(h^{(1)}(\mathbf{x})\right)\right)$$

No feedback connections (Until we get to Recurrent Networks!)

Function composition can be described by a directed acyclic graph.

Gives up convexity.



Note: All neurons have biases. For convenience they are not represented in the diagram.

It is important for $h^{(i)}(\cdot)$ to have some non linearity. Stacking linear layers will still be linear.

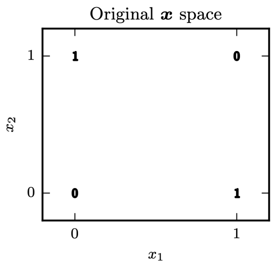
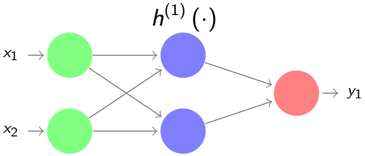


Image: Deep learning, Goodfellow.



Model:

$$y = \left(w^{(2)}\right)^{\top} \max\left\{0, \left(w^{(1)}\right)^{\top} \mathbf{x}\right\}$$

Weights:

$$w^{(1)} = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix} \quad w^{(2)} = \begin{bmatrix} 0 \\ 1 \\ -2 \end{bmatrix}$$

Assume the weights are given.

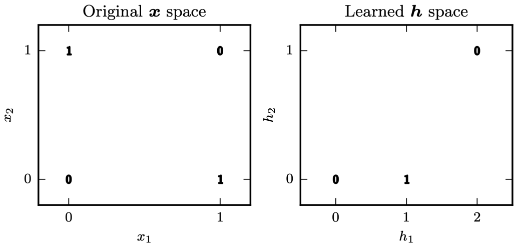
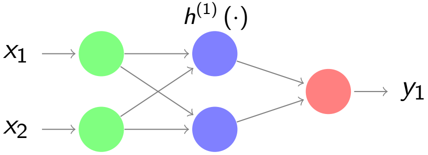


Image: Deep learning, Goodfellow.

Model: $y = \left(w^{(2)}\right)^\top \max\left\{0, \left(w^{(1)}\right)^\top \mathbf{x}\right\}$

Weights:

$$w^{(1)} = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix} \quad w^{(2)} = \begin{bmatrix} 0 \\ 1 \\ -2 \end{bmatrix}$$



Increase Model Capacity

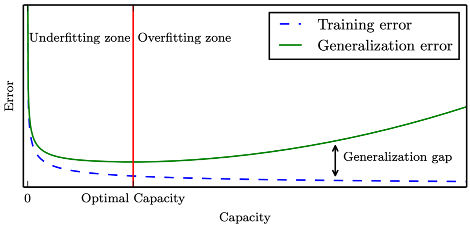
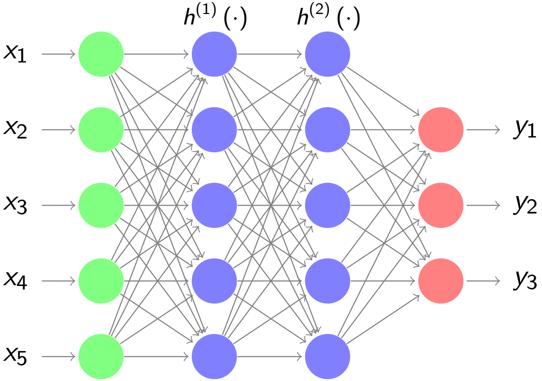


Image: Goodfellow, 2016.

Simpler functions are more likely to generalize, but a sufficiently complex hypothesis is needed to achieve low training error.

Multi-Class Classification: Add output unit equal to the number of classes.

Regression: Output units with linear activation.



Note: All neurons have biases. For convenience they are not represented in the diagram.

Does the cross entropy loss function work?

Outline

The Hidden layers of a Feed forward NN consists of an affine transformation and a activation:

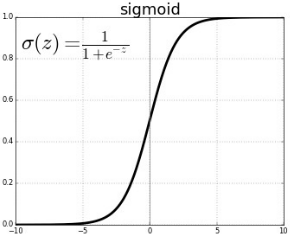
$$\mathbf{z}^{(i)} = \mathbf{W}^{(l)}\mathbf{x}^{(i)} + \mathbf{b}^{(l)} \qquad \triangleright \text{Affine transform}$$

$$h^{(i)} = g\left(\mathbf{z}^{(i)}\right) \qquad \triangleright \text{Activation}$$

The activation is applied (usually) element-wise.

What functions can be used for activation?

Design of Hidden units is an active area of research.



$$g\left(z^{(i)}\right) = \frac{1}{1 + exp\left(-z^{(i)}\right)}$$

- Squashing type non-linearity: pushes outputs to range $[0, 1]$.
- Saturate across most of their domain, strongly sensitive only when z is closer to zero.
- Saturation makes gradient based learning difficult.
- tanh function is similar to sigmoid but pushes outputs to range $[-1, 1]$.

$$g\left(z^{(i)}\right) = \max\left(0, z^{(i)}\right)$$

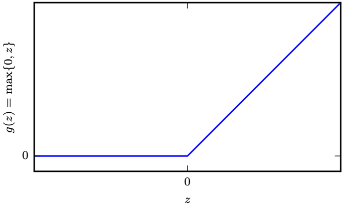


Image: Deep learning, Goodfellow.

- Gives large and consistent gradients (does not saturate) when active.
- Efficient to optimize, converges much faster than sigmoid.
- Not everywhere differentiable: In practice not a problem. Return one sided derivatives at $z = 0$. Stochastic Gradient based optimization is subject to numerical error.
- Units when inactive will never update.
- Good Practice: Initialize all elements of b to a small positive value, such as 0.1.

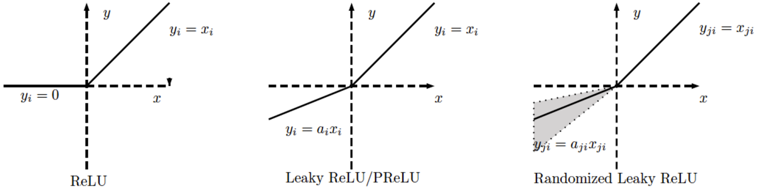


Image: Xu, B."Empirical evaluation of rectified activations in convolutional network".

$$g\left(z^{(i)}\right) = \max\left(0, z^{(i)}\right) + a_i \min\left(0, z^{(i)}\right)$$

Get a non-zero slope when $z^{(i)} < 0$.

- **Leaky ReLU** (Maas et al., 2013): Fix a_i to a small value (e.g 0.001)
- **Randomized ReLU** (Xu et al., 2015): Sample a_i from a fixed range during training, fix during testing.
- **Parametric ReLU** (He et al., 2015): Learn a_i

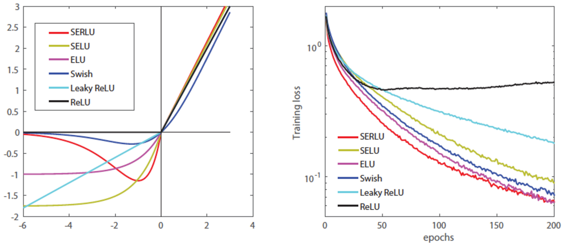


Image: G. Zhang, "Effectiveness of Scaled Exponentially-Regularized Linear Units (SERLUs)".

$$g(z^{(i)}) = \begin{cases} z^{(i)} & \text{if } z^{(i)} > 0 \\ \alpha \left(\exp\left(z^{(i)}\right) - 1\right) & \text{if } z^{(i)} \leq 0 \end{cases}$$

Get a non-zero slope when $z^{(i)} < 0$. Calculating exponent expensive.
Paper: Fast and Accurate Deep Network Learning by Exponential Linear Units

MaxOut Units

$$g\left(z^{(i)}\right)_j = \max_{k \in \mathbb{G}^{(j)}} z_k^{(i)}$$

- Fundamentally different to other units we discussed so far - Not elementwise.
- Generalize rectified linear units further.
- maxout units divide z into groups of k values. Each maxout unit then outputs the maximum element of one of these groups.
- A maxout unit can learn a piece-wise linear, convex function with up to k pieces.
- With large enough k, a maxout unit can learn to approximate any convex function with arbitrary fidelity (e.g. ReLU).

Goodfellow I, et. al. "Maxout networks". In International conference on machine learning 2013.

Revision Questions

1. Why don't we just use identity transform as a activation unit?
2. What is an issue with sigmoid activation?
3. What is an issue with Relu activation?

Similar to the Perceptron define p_{model} and use the principle of maximum likelihood.

$$\mathcal{L}(\mathbf{w}) = -\frac{1}{N}\sum_{i=0}^{N} \log p\left(y^{(i)} \mid \mathbf{x}^{(i)}; \mathbf{w}\right)$$

If $p_{model} = \mathcal{N}(y; h(\mathbf{x}; \mathbf{w}), I)$ then:

$$\mathcal{L}(\mathbf{w}) = -\frac{1}{N}\sum_{i=0}^{N} \|y - h(\mathbf{x}; \mathbf{w})\|^2$$

Choice of output units is very important for choice of cost function

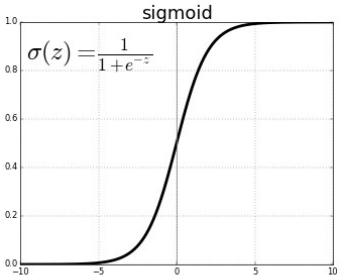
Task: Predict a binary variable $y \in \{0, 1\}$

Use a sigmoid unit. If the output of the penultimate layer is $\mathbf{g}$.

$$\hat{y}^{(i)} = p\left(y^{(i)} = 1 \mid x^{(i)}\right) = \sigma\left(\mathbf{W}^{\top}\mathbf{g}^{(i)} + b\right)$$

y is a Bernoulli random variable:

$$p\left(y^{(i)} \mid \mathbf{x}^{(i)}; \mathbf{w}\right) = \left(\hat{y}^{(i)}\right)^{y^{(i)}} \left(1 - \hat{y}^{(i)}\right)^{\left(1-y^{(i)}\right)}$$



$$\mathcal{L}(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} \log\left(\hat{y}^{(i)}\right) + \left(1 - y^{(i)}\right)\log\left(1 - \hat{y}^{(i)}\right)\right)$$

$*$This is not the only option.

Saturation thus occurs only when the model already has the right answer.

Other loss functions, such as mean squared error, can saturate anytime $\sigma(z)$ saturates.

Other Loss Functions

Loss	Usage	Comments
Hinge-Loss $\max\left[1 - h\left(x^{(i)}; w\right) y^{(i)}\right]^{p}$	SVM	When used for Standard SVM, the loss function denotes the size of the margin between linear separator and its closest points in either class.
Log-Loss $\log\left(1 + e^{-h\left(x^{(i)};w\right)y^{(i)}}\right)$	logistic regrssion	One of the most popular loss functions in Machine Learning, since its outputs are well-calibrated probabilities.
Exponential-Loss $e^{-h\left(x^{(i)};w\right)y^{(i)}}$	Ada-Boost	This function is very aggressive. The loss of a misprediction increases exponentially with the value
Zero-One-Loss $\delta\left(\mathrm{Sign}\left(h\left(x^{(i)}; w\right)\right) \neq y^{(i)}\right)$	Actual Classification loss	Non-continuous and thus impractical to optimize.

The Target is represented as: $y \in \{-1, +1\}$

Task: Predict a categorical variable $y \in \{0, 1\}^c$

Use linear layer followed by SoftMax (assume the output of the final linear layer is $\mathbf{z}$):

$$\hat{y}_j^{(i)} = \mathrm{SoftMax}(\mathbf{z})_j = \frac{\exp(z_j)}{\sum_j \exp(z_j)}$$

SoftMax across all units will add to one. As y is multinomial random variable, we can write:

$$p\left(y^{(i)} \mid \mathbf{x}^{(i)}; \mathbf{w}\right) = \prod_{j=1}^{c} \left(\mathrm{SoftMax}\left(\mathbf{z}^{(i)}\right)_j\right)^{y_j^{(i)}}$$

$$\mathcal{L}(\mathbf{w}) = -\sum_{i=1}^{N} \sum_{j=1}^{c} y_j^{(i)} \log\left(\hat{y}_j^{(i)}\right)$$

example:

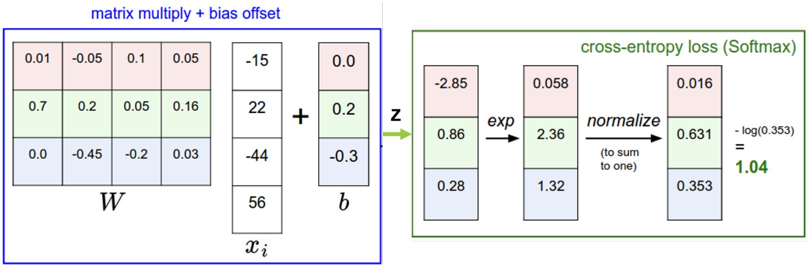
$$\hat{y}^{(i)} = \begin{bmatrix} 0.1 & 0.2 & 0.6 & 0.1 \end{bmatrix}$$

$$y^{(i)} = \begin{bmatrix} 0.0 & 0.0 & 1.0 & 0.0 \end{bmatrix}$$

$\mathcal{L}(\mathbf{w})^{(i)} =$
$0 \times 2.3 + 0 \times 1.6 + 1 \times 0.5 + 0 \times 2.3$

Maximizing log-likelihood encourages $z_j^{(i)}$ to be pushed up, while softmax encouraging all the other z to be pushed down (competition).

Multi-Class Classification

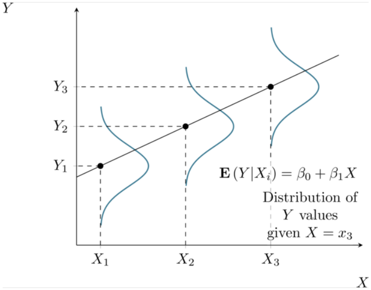


Affine

Activation

Task: Predict a real valued variable $y \in \mathbb{R}$

Use linear activation: $p_{model} = \mathcal{N}\left(y; h\left(\mathbf{x}; \mathbf{w}\right), I\right)$ then:



$$\mathcal{N}\left(y; h\left(\mathbf{x}; \mathbf{w}\right), I\right) = \frac{1}{2\pi\sigma^2} \exp\left\{ - \frac{\left\| y^{(i)} - h\left(x; w\right) \right\|^2}{2\sigma^2} \right\}$$

RMIT UNIVERSITY

Task: Predict a real valued variable $y \in \mathbb{R}$

Use linear activation: $p_{model} = \mathcal{N}\left(y; h\left(\mathbf{x}; \mathbf{w}\right), I\right)$ then:

$$\mathcal{L}\left(\mathbf{w}\right) = -\frac{1}{N}\sum_{i=0}^{N} \left\|y - h\left(\mathbf{x}; \mathbf{w}\right)\right\|^2$$

Because linear units do not saturate, they pose little difficulty for gradient-based optimization algorithms and may be used with a wide variety of optimization algorithms.

Other Loss Functions

Loss	Comments
Squared Loss $(h(\mathbf{x}_i) - y_i)^2$	ADVANTAGE: Differentiable everywhere DISADVANTAGE: Somewhat sensitive to outliers/noise Also known as Ordinary Least Squares (OLS)
Absolute Loss $\|h(\mathbf{x}_i) - y_i\|$	ADVANTAGE: Less sensitive to noise DISADVANTAGE: Not differentiable at 0
Huber Loss $\frac{1}{2}(h(\mathbf{x}_i) - y_i)^2$ if $\|h(\mathbf{x}_i) - y_i\| < \delta$, otherwise $\delta(\|h(\mathbf{x}_i) - y_i\| - \frac{\delta}{2})$	ADVANTAGE: "Best of Both Worlds" of Squared and Absolute Loss Once-differentiable Takes on behavior of Squared-Loss when loss is small, and Absolute Loss when loss is large.
Log-Cosh Loss $log(cosh(h(\mathbf{x}_i) - y_i))$	ADVANTAGE: Similar to Huber Loss, but twice differentiable everywhere

1. what happens if sigmoid output units (one for each class) are used when the task is multi-class classification?
2. Can you use softmax activation for binary classification?

Model Architecture

The word architecture refers to the overall structure of the network: how many units it should have and how these units should be connected to each other.

Commonly neural networks are organized into groups of units called layers. Most neural network architectures arrange these layers in a chain structure.

$$h(\mathbf{x}; \mathbf{W}) = h^{(3)}\left(h^{(2)}\left(h^{(1)}\left(\mathbf{x}; \mathbf{W}^{(1)}\right); \mathbf{W}^{(2)}\right); \mathbf{W}^{(3)}\right)$$

In these chain-based architectures, the main architectural considerations are choosing the **depth** of the network and the **width** of each layer.

Universal Approximation Theorem (Horniket al., 1989; Cybenko, 1989):
"*A feed-forward network with a linear output layer and at least one hidden layer with any activation function (such as the logistic sigmoid activation function) can approximate any Borel measurable function from one finite-dimensional space to another with any desired non zero amount of error, provided that the network is given enough hidden units.*"

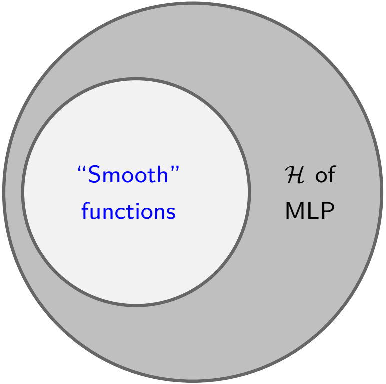
In simple terms — you can always come up with a neural network that will approximate any complex relation between input and output. Given that it has at least one hidden layer with non-linear activation and "enough" neurons.

Seems like this is a silver bullet. Are we done?

Universal Approximation Theorem say that a large MLP will be able to *represent* any complex function (with some assumption).

It does not guarantee that we would be able to *learn* this function. learning can fail:

- Optimization procedure may not find appropriate weights (e.g. may find local minimum).
- Might choose wrong weights due to over-fitting.



Network Depth

- The one hidden layer NN in theory might have infeasibly large number of neurons and may fail to learn and generalize correctly.
- In practice, using deeper models can reduce the number of neurons and can reduce the amount of generalization error.
- **Intuition:** When we choose a specific machine learning algorithm, we are implicitly stating some set of prior beliefs we have about what kind of function the algorithm should learn.
- Choosing a deep model encodes a very general belief that the function we want to learn should involve composition of several simpler functions.

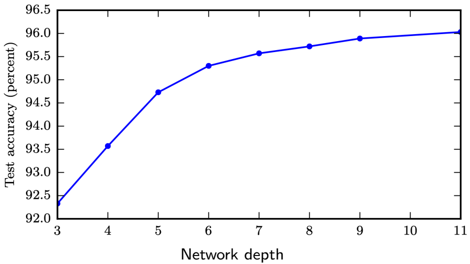


Image: Deep learning, Goodfellow.

Empirically, greater depth does seem to result in better generalization.

- Need many further practical tricks. e.g. residual blocks, convolutions, etc. We will cover these later.

Is this because high depth results in high number of parameters?

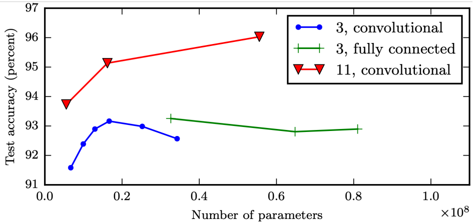


Image: Deep learning, Goodfellow.

Deeper models perform better not merely because the model is larger.

Above results from Goodfellow et al. shows that increasing the number of parameters in layers of convolutional networks with out increasing their depth is not nearly as effective at increasing test set performance.

Summary

1. Feed-forward Neural networks based on preceptrons are a very general (flexible) type of function approximater.
2. Many ways to customize the models.
3. Nice theoretical results that shows that neural networks models can be used to model “any” complex function.

Lab: We will see how NN can be implemented in TensorFlow.

Next week:

1. How to optimize feed forward NN.
2. Regularization.