Feedforward Neural Networks and Backpropagation

Changyou Chen

Department of Computer Science and Engineering
Universitpy at Buffalo, SUNY
changyou@buffalo.edu

February 14, 2019

Project Computational Resources

- Students can use Google Cloud Platform for the projects.
- Each student has $50 coupon.
- You will need to use your buffalo.edu email address to redeem the coupon.
- Redeem no later than 5/29/2019, valid through 1/29/2020.
- Details for redeem will be emailed to you after the lecture.

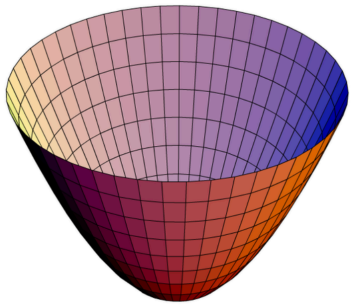
Objective Function in Gradient-Based Learning

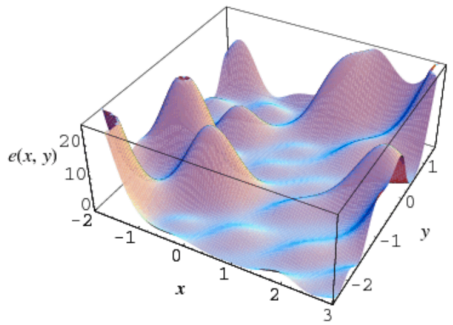
Standard ML Training vs. NN Training

1. Similar to standard ML training, in deep learning we need
   - cost function, *e.g.*, MLE.
   - models, *e.g.*, FNN.
   - optimization algorithm, *e.g.*, gradient descent.
2. Different from traditional ML: nonlinearity causes non-convex loss
   - use iterative gradient-based optimizers that merely drives cost to low value.
   - in standard ML, we have exact linear equation solvers used for linear regression, or convex optimization algorithms used for logistic regression or SVMs.

Convex vs Non-convex

1. Convex methods:
   - converge to global optima from any initial parameters.
   - Robust and theoretically sound.
2. Non-convex with SGD:
   - sensitive to initial parameters.
   - for FNN, important to initialize Weights to small values, Biases to zero or small positives.
   - SGD also adopted for training Linear Regression and SVM, especially with large training sets.
   - training neural net no different to other models, except computing gradient is more complex.
   - general convergence theory has not been established, though there are some pioneer work.

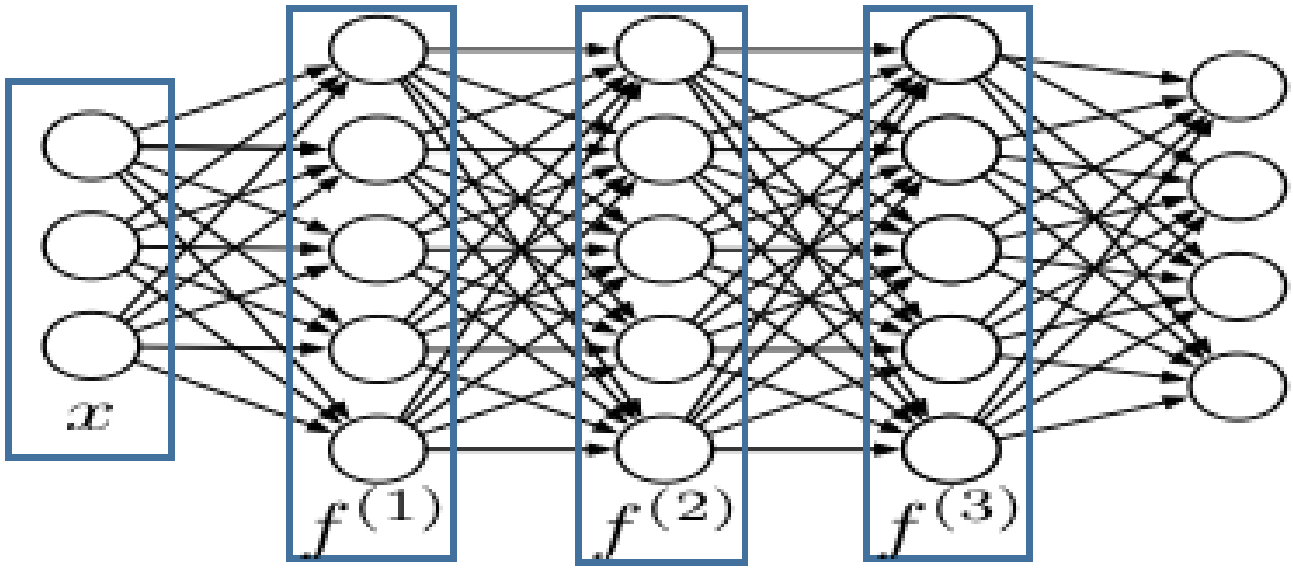




Cost Functions for Deep Learning

1. Important to make a deep neural network work:
   - similar to those for parametric models such as linear models
2. In FNN, the goal is to try to match the output to the true data (labels).
3. How to measure how well they match?

Cost function corresponds to what happens in the output layer.



Cross Entropy

Cross entropy for distributions p and q

$$H(p, q) = \mathbb{E}_p [-\log q] = -\int \log q(x) p(x) \mathrm{d}x$$

$$\xrightarrow{\text{discrete } p \text{ and } q} = -\sum_x p(x) \log q(x)$$

- To use this metric, must specify both target output data and the output from the neural network as distributions.
- Specifying the model $p(\mathbf{y} \mid \mathbf{x}; \theta)$ automatically determines a cost function:

$$J(\theta) = -\mathbb{E}_{\mathbf{x}, y \sim \hat{p}_{\text{data}}} \log p(\mathbf{y} \mid \mathbf{x}; \theta)$$

  - equivalently described as the cross-entropy between the training data and the model distribution

Output Units

1. Choice of cost function typically only depends on the choice of output unit, at least in FNN.
2. Types of output units:
   - Sigmoid units: for Bernoulli Output Distributions, *e.g.*, binary classification.
   - Softmax units: for Multinomial Output Distributions, *e.g.*, multi-class classification.
   - Linear units: no non-linearity, for Gaussian Output distributions, *e.g.*, regression.

Cost Functions for Binary Classification with Logistic Regression

1. Logistic regression: data $\{\mathbf{x}_n \in \mathbb{R}^L, y_n \in \{0,1\}\}$, likelihood

$$p(\mathbf{y} \mid \mathbf{x}; \theta) = \prod_{n=1}^{N} \sigma(\theta^T \mathbf{x}_n)^{y_n} (1 - \sigma(\theta^T \mathbf{x}_n))^{1-y_n}$$

2. Use the principle of maximum likelihood

$$J(\theta) = -\log p(\mathbf{y} \mid \mathbf{x}; \theta) = -\sum_{n=1}^{N} \Big(y_n \log \sigma(\theta^T \mathbf{x}_n) + (1 - y_n) \log(1 - \sigma(\theta^T \mathbf{x}_n)) \Big)$$

3. Let $p_n \triangleq [y_n, 1 - y_n]$, $q_n = [\sigma(\theta^T \mathbf{x}_n), 1 - \sigma(\theta^T \mathbf{x}_n)]$:
   - they form probability vectors, can be regarded as two discrete distributions of dimension two.
4. The negative log-likelihood coincides with cross-entropy of p_n and q_n.
5. Gradient is: $\nabla_\theta J(\theta) = \sum_{n=1}^{N} (\tilde{y}_n - y_n) \mathbf{x}_n$, where $\tilde{y}_n \triangleq \sigma(\theta^T \mathbf{x}_n)$.

Binary Classification with FNN

1. Only relates to the output layer of an FNN.
2. Given features $\mathbf{h}$ prior to the output layer, a layer of sigmoid output units produces scalar

$$\hat{y} = \sigma(\mathbf{w}^T \mathbf{h} + b) = \frac{1}{1 + \exp(-\mathbf{w}^T \mathbf{h} - b)}$$

3. Use to produce a conditional Bernoulli output $y \in 0, 1$

$$p(y = 1 | \mathbf{x}) = \hat{y}, \text{let } z = \mathbf{w}^T \mathbf{h} + b$$

$$\xrightarrow{\text{write it in another way}} p(y | \mathbf{x}) = \frac{\exp(yz)}{\sum_{y'=0}^{1} \exp(y'z)} = \sigma((2y - 1)z)$$

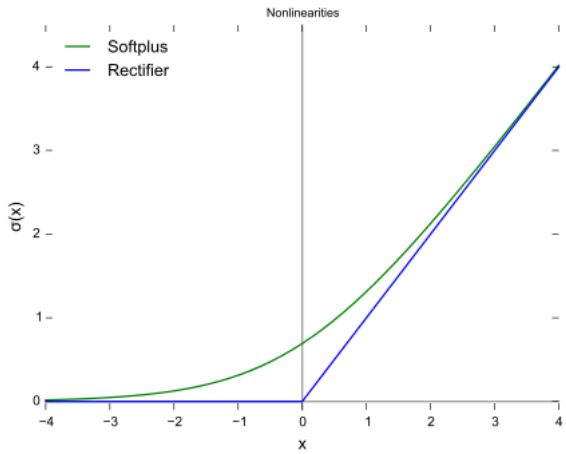
4. Loss function with negative log-likelihood:

$$J(\theta) = -\log p(y | \mathbf{x}) = -\log \sigma((2y - 1)z) = \varsigma((1 - 2y)z),$$

where $\varsigma(x) = \log(1 + \exp(x))$ is the softplus function.

Property of Loss Function for Bernoulli MLE

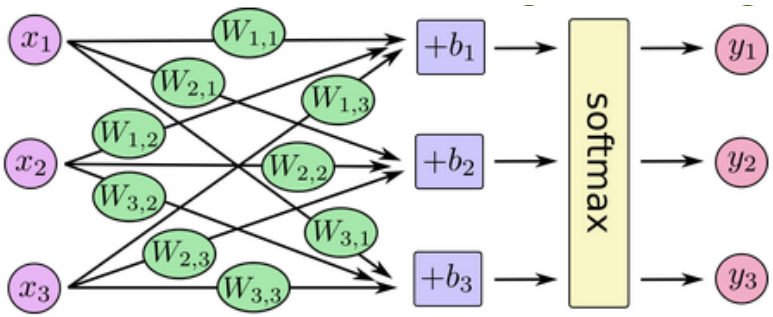
Softplus: $J(\theta) = \varsigma((1-2y)z)$



1. The loss function saturates only when $(1-2y)z << 0$.
2. Saturation occurs only when model already has the right answer:
   - *i.e.*, when $y = 1$ and $z >> 0$ or $y = 0$ and $z << 0$. This data then contributes little when applying gradient-based learning algorithms.
   - when z has the wrong sign, $(1-2y)z$ can be simplified to $|z|$, which does not shrink the gradient at all, a useful property because gradient-based learning can act quickly to correct a mistaken z.

Multi-class Classification with FNN

Generalization of Logistic Regression to multivalued output



Softmax definition

$$\boldsymbol{y} = \text{softmax}(\boldsymbol{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

Network Computes

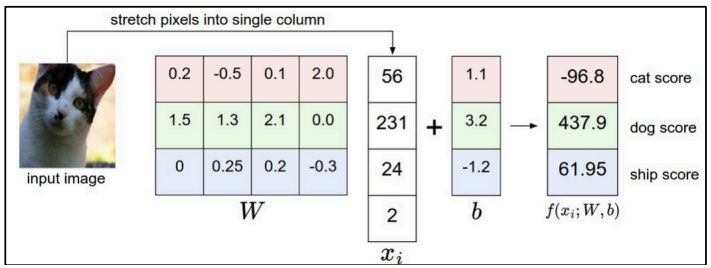
$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{bmatrix} \right)$$

In matrix multiplication notation

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \right)$$

$$\boldsymbol{z} = W^T \boldsymbol{x} + \boldsymbol{b}$$

An example



Softmax Output in FNN

1. The label is encoded as a n-dimensional probability vector y, with the y-th element being 1, while others being 0:
   - recognized as one-hot vectors.
2. The output of an FNN is Softmax, which forms a probability distribution over a discrete variable with n values.
3. Softmax produces a vector $\hat{y}$ with values $\hat{y} = P(y = i | \mathbf{x})$:
   - need elements of $\hat{y}$ lie in $[0, 1]$ and they sum to 1.
4. Similar to the sigmoid case, we normalize n exponentials:

$$\text{softmax}(\mathbf{z})_k = \frac{\exp(z_k)}{\sum_{k'} \exp(z_{k'})}, \quad \mathbf{z} = \mathbf{W}^T \mathbf{h} + \mathbf{b}$$

Saturation of Softmax

$$\text{softmax}(\mathbf{z})_k = \frac{\exp(z_k)}{\sum_{k'} \exp(z_{k'})}$$

1. Like sigmoid, softmax activation can saturate:
   - when the differences between input values become extreme.
2. This is a generalization of the way the sigmoid units saturate:
   - it can cause similar difficulties in learning if the loss function is not designed to compensate for it.
   - the cross-entropy loss function ease the issue, shown later.
3. In practice, we use a numerically stable variant of softmax

$$\text{softmax}(\mathbf{z}) = \text{softmax}(\mathbf{z} - \max_i z_i)$$

4. Reformulation allows us to evaluate softmax with high accuracy.

Likelihood with Log-Softmax

1. Using softmax as the likelihood function, we get

$$\log \text{softmax}(\mathbf{z})_i = z_i - \log \sum_j \exp(z_j) \approx z_i - max_j z_j$$

2. Easy to train with gradient descent because it behaves similarly to a linear function:
   - if the correct answer already has the largest input to softmax, then the two terms will roughly cancel. This data will then contribute little to overall training cost.

Multi-class Classification with FNN

$$\text{softmax}(\mathbf{z})_k = \frac{\exp(z_k)}{\sum_{k'} \exp(z_{k'})} \triangleq \tilde{y}_k, \quad \mathbf{z} = \mathbf{W}^T \mathbf{h} + \mathbf{b}$$

Cross entropy

$$J = -\sum_{n=1}^{N} \sum_{k=1}^{K} y_k^{(n)} \log \tilde{y}_k^{(n)}$$

Gradient (consider $N = 1$)

$$\nabla_{\mathbf{W}} J = -\sum_{k=1}^{K} y_k \nabla_{z_k} \log \tilde{y}_k \frac{\partial z_k}{\partial \mathbf{W}}$$

Derivative of softmax (assume $N = 1$)

$$J = -\sum_{k=1}^{K} y_k \log \tilde{y}_k, \quad \tilde{y}_k = \frac{\exp(z_k)}{\sum_{k'} \exp(z_{k'})}$$

Derive $\frac{\partial J}{\partial z_i}$

Linear units for Gaussian regression

1. Given features $\mathbf{h}$, a layer of linear output units produces a vector

$$\hat{y} = \mathbf{W}^T \mathbf{h} + \mathbf{b}$$

2. Use to produce mean $\hat{y}$ of a conditional Gaussian distribution

$$p(y|\mathbf{x}) = \mathcal{N}(y; \hat{y}, \mathbf{I}) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}\|y - \tilde{y}\|^2\right)$$

3. Cross entropy recovers the mean squared error cost ($\theta \triangleq \{\mathbf{W}, \mathbf{b}\}$)

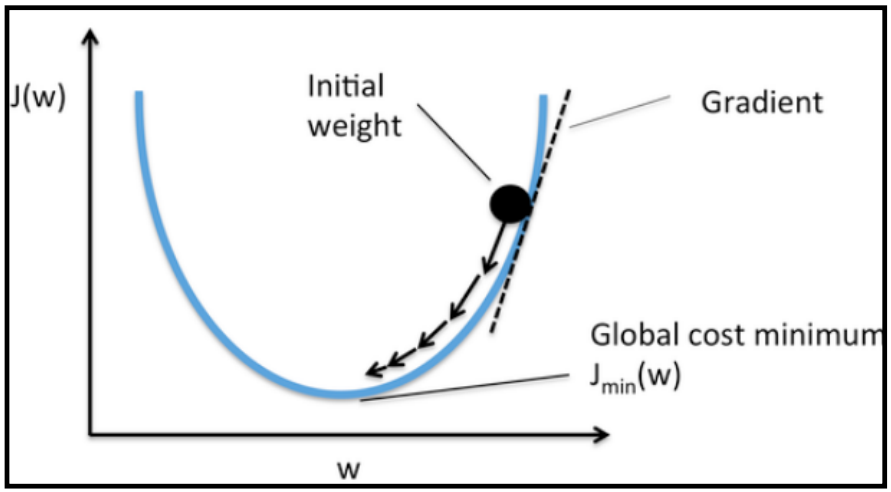
$$J(\theta) = -\frac{1}{2}\mathbb{E}_{\mathbf{x},y\sim\hat{p}_{\text{data}}} \|y - f(\mathbf{x};\theta)\|^2 + const$$

4. Gradient:

$$\nabla_\theta = \mathbb{E}_{\mathbf{x},y\sim\hat{p}_{\text{data}}} (y - \tilde{y}) \nabla_\theta \tilde{y}$$

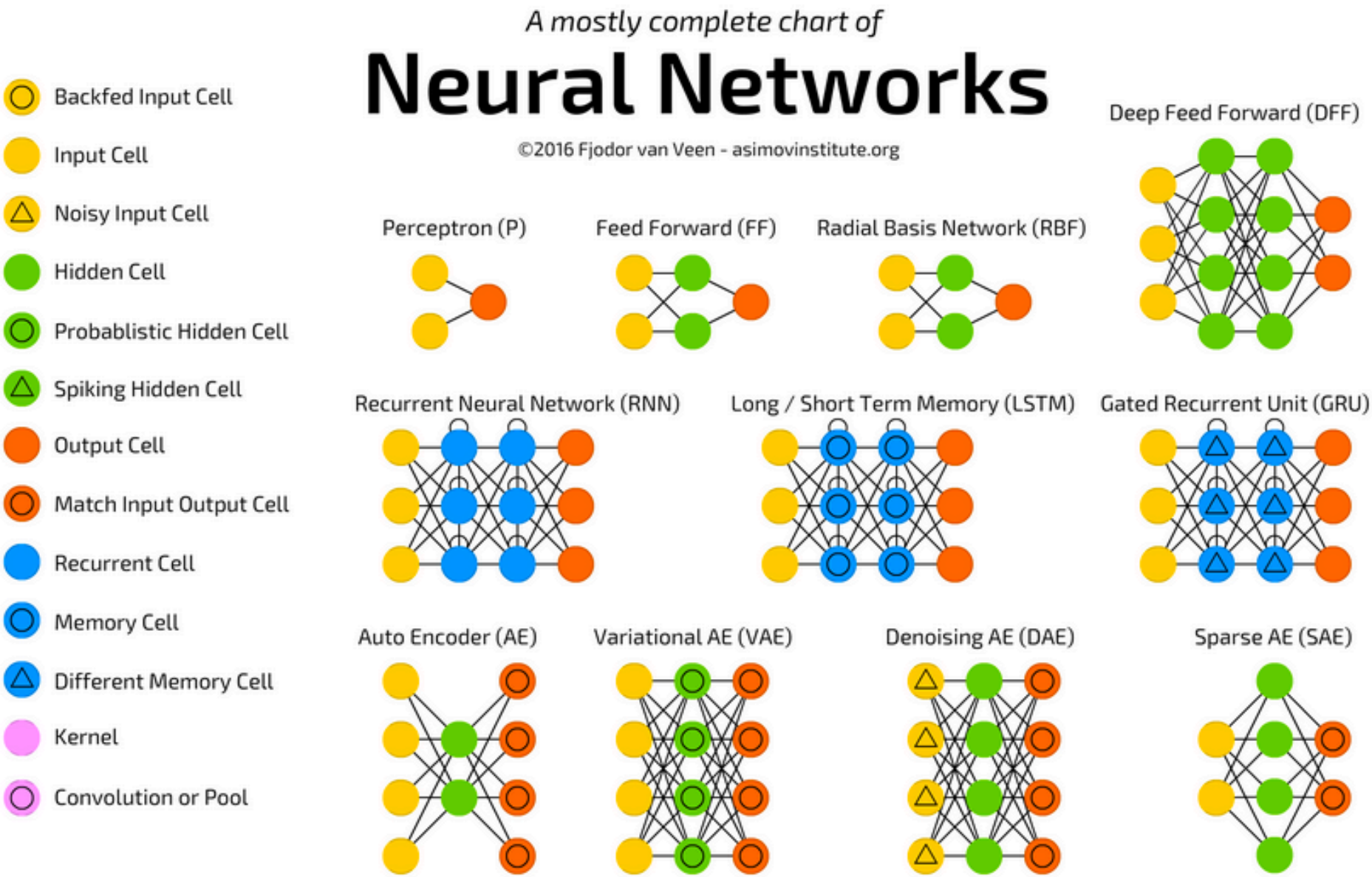
Desirable Property of Gradient

1. Gradients in neural networks should be large and predictable enough to make learning algorithms progressed.
2. Functions that saturate (become very flat) undermine this objective because the gradient becomes very small.
   - Happens when activation functions producing output of hidden/output units saturate.
3. Negative log-likelihood helps avoid saturation for many models
   - Log function in Negative log likelihood cost function undoes exp of some units.



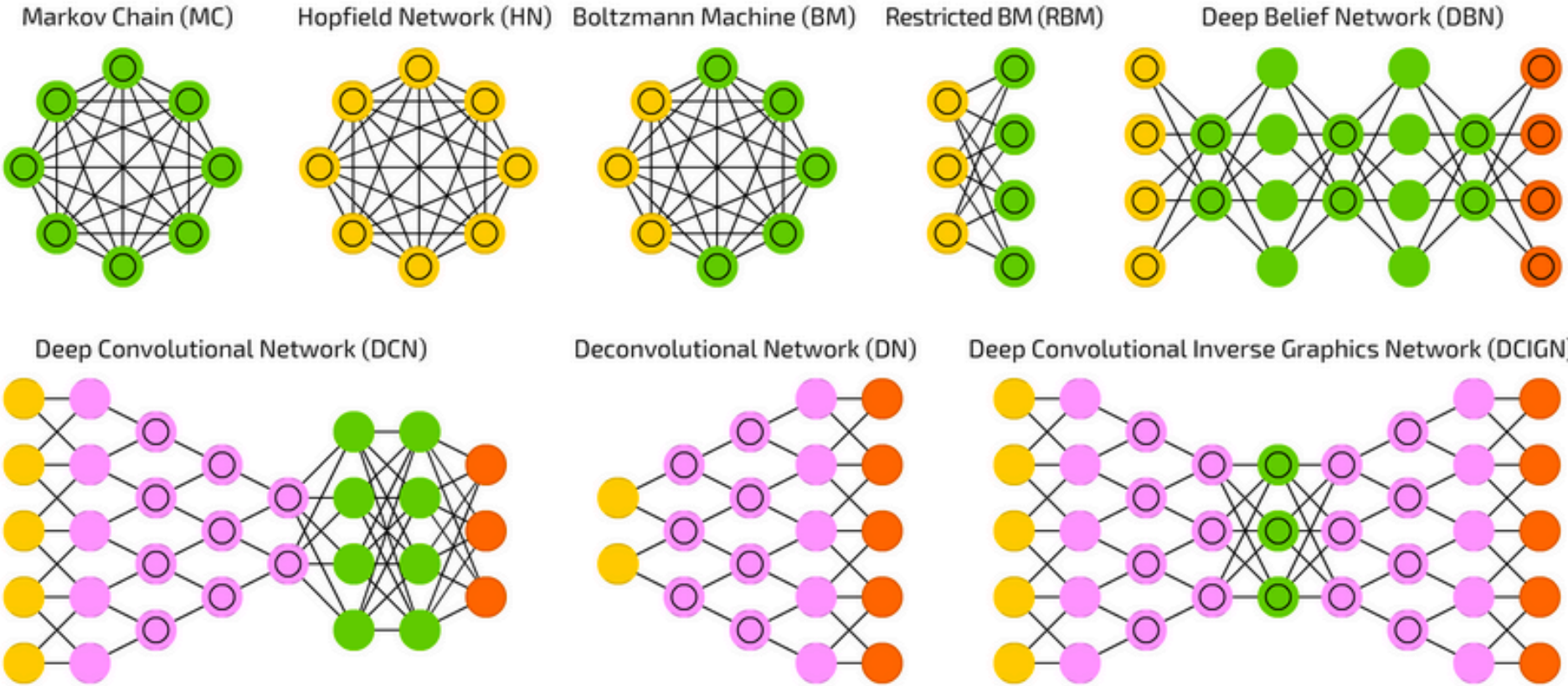
Architecture Design for Deep Learning

Generic Neural Architectures (1-11)



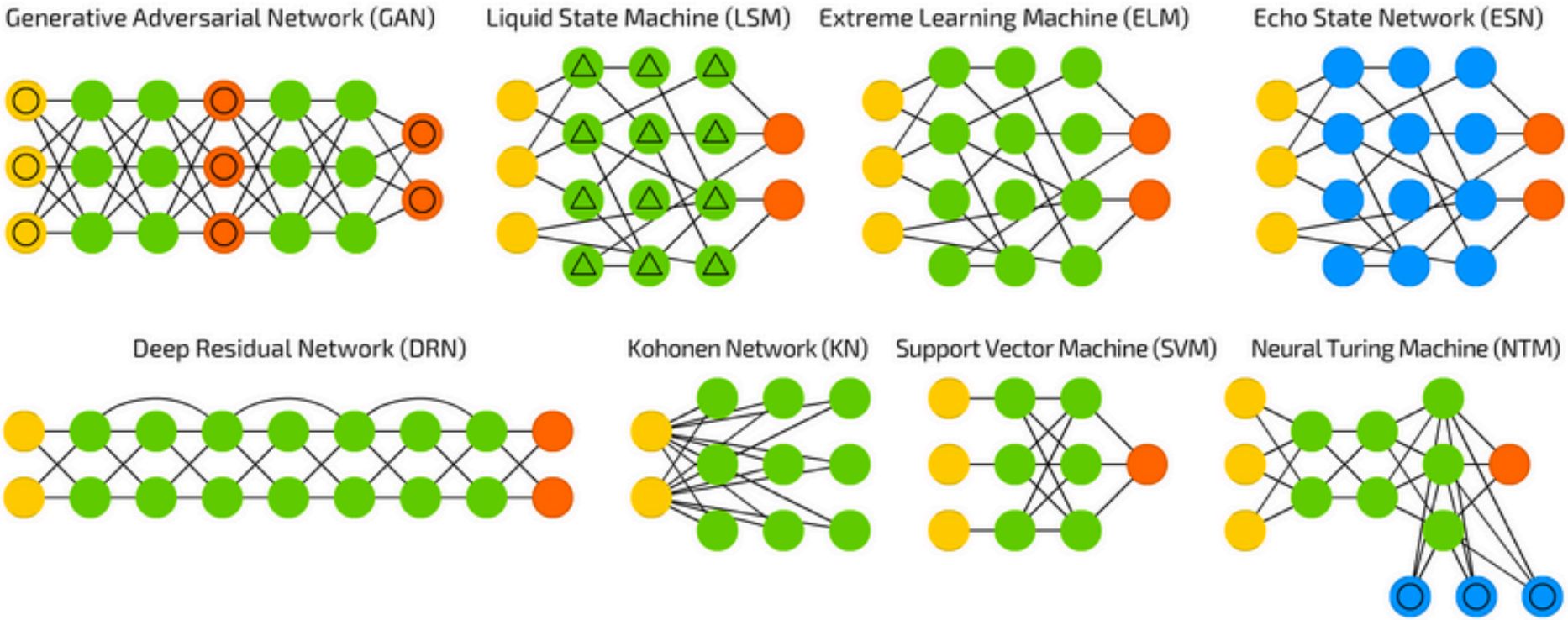
http://www.asimovinstitute.org/neural-network-zoo/

Generic Neural Architectures (12-19)



http://www.asimovinstitute.org/neural-network-zoo/

Generic Neural Architectures (20-27)



http://www.asimovinstitute.org/neural-network-zoo/

Architecture Terminology

1. The word architecture refers to the overall structure of the network.
   - How many units should it have?
   - How the units should be connected to each other?
2. Most neural networks are organized into groups of units called layers.
   - Most architectures arrange these layers in a chain structure.
   - Each layer is a function of the layer that preceded it, *e.g.*,
     - ★ first layer is given by $\mathbf{h}^{(1)} = g^{(1)}\left(\mathbf{W}^{(1)^T}\mathbf{x} + \mathbf{b}^{(1)}\right)$
     - ★ second layer is given by $\mathbf{h}^{(2)} = g^{(2)}\left(\mathbf{W}^{(2)^T}\mathbf{h}^{(1)} + \mathbf{b}^{(2)}\right)$

Advantage of Deeper Networks

1. Main architectural considerations: depth and width of a network.
2. Deeper networks have:
   - far fewer units in each layer.
   - far fewer parameters.
   - often generalize well to the test set.
   - but are often more difficult to optimize.
3. Ideal network architecture should be found via experimentation guided by validation set error.

Universal Approximation Theorem

Theorem

A feed-forward network with a single hidden layer containing a finite number of neurons can approximate any continuous functions on compact subsets of $\mathbb{R}^n$, under mild assumptions on the activation function.

1. Simple neural networks can represent a wide variety of interesting functions when given appropriate parameters.
2. However, it does not touch upon the algorithmic learnability of those parameters.
   - Optimizing algorithms may not find the parameters.
   - May choose wrong function due to overfitting.

FNN & No Free Lunch

FNN

Feed-forward networks provide a universal system for representing functions:

- Given a continuous function, there is a feed-forward network that approximates the function.

No Free Lunch

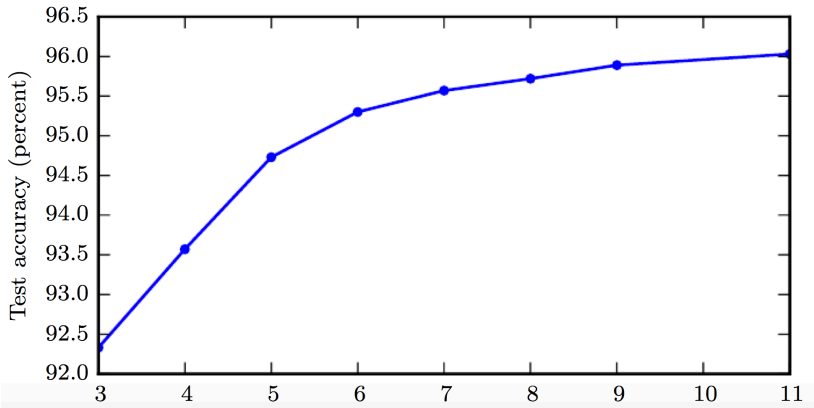
There is no universal procedure for examining a training set of specific examples and choosing a function that will generalize to points not in training set.

On Size of Network

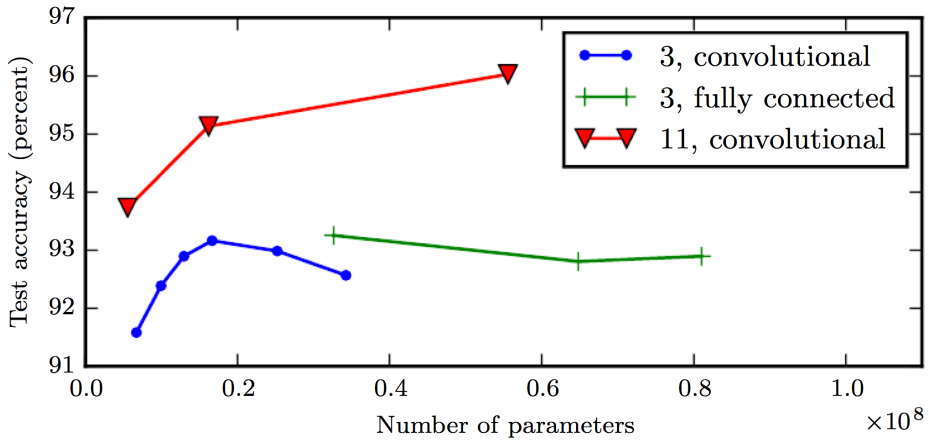
1. Universal Approximation Theorem implies there is a network large enough to achieve any degree of accuracy:
   - but does not say how large the network will be.
2. Some bounds on size of the single-layer network exist for a broad class of functions, but worst case is exponential number of hidden units w.r.t. data dimension.
3. Using deeper models can reduce number of units required and reduce generalization error:
   - Some families of functions can be represented efficiently if depth $> d$ but require much larger model if depth $< d$.
   - Still a challenging task for learning theory community.

Empirical Justification for Deep

- CNN for street-view image recognition.
- Deeper networks perform better.



Test accuracy consistently increases with depth



Increasing parameters without increasing depth is not as effective