

# Neural Networks

David Rosenberg

New York University

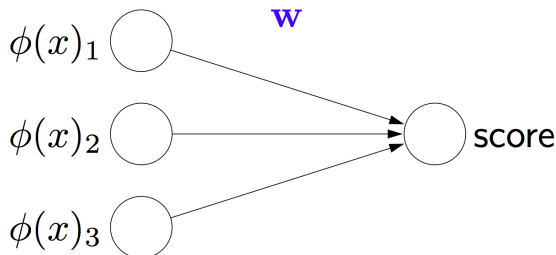
October 29, 2016

# Objectives

- What are neural networks?
- How do they fit into our toolbox?
- When should we consider using them?

# Linear Prediction Functions

- Linear prediction functions: SVM, ridge regression, Lasso
- Generate the feature vector  $\phi(x)$  by hand.
- Learn weight vector  $w$  from data. ()



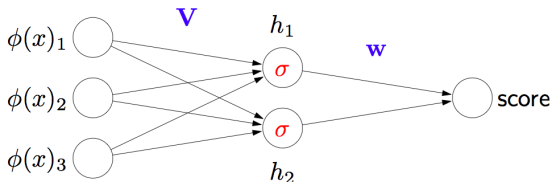
- So

$$\text{score} = w^T \phi(x)$$

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

# Neural Network

- Add an extra layer with a nonlinear transformation:

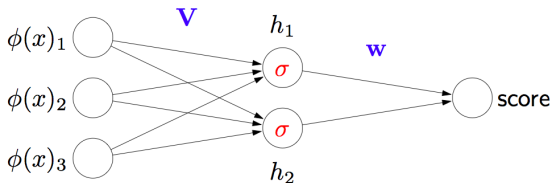


- We've introduced **hidden nodes**  $h_1$  and  $h_2$ .

$$h_i = \sigma(v_i^T \phi(x)),$$

where  $\sigma$  is a nonlinear **activation function**. (We'll come back to this.)

# Neural Network



- Score is just

$$\begin{aligned}\text{score} &= w_1 h_1 + w_2 h_2 \\ &= w_1 \sigma(v_1^T \phi(x)) + w_2 \sigma(v_2^T \phi(x))\end{aligned}$$

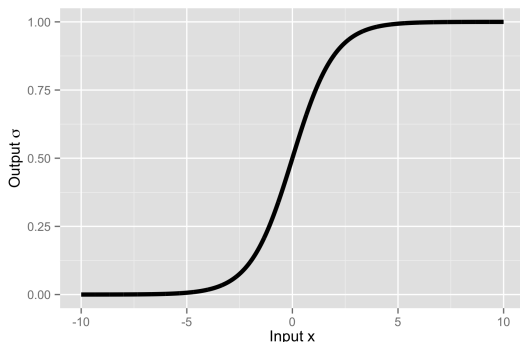
- This is the basic recipe.
  - We can add more hidden nodes.
  - We can add more hidden layers.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

# Activation Functions

- The **nonlinearity** of the activation function is a key ingredient.
- The **logistic sigmoid** function is one of the more commonly used:

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

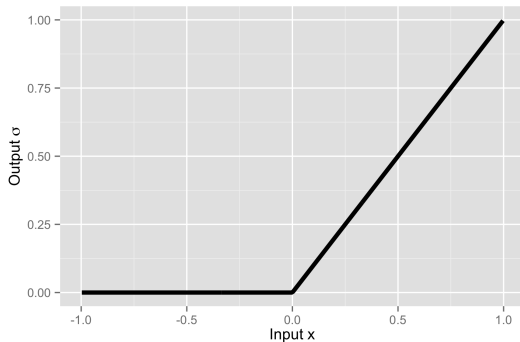


# Activation Functions

- More recently, the **rectified linear** function has been very popular:

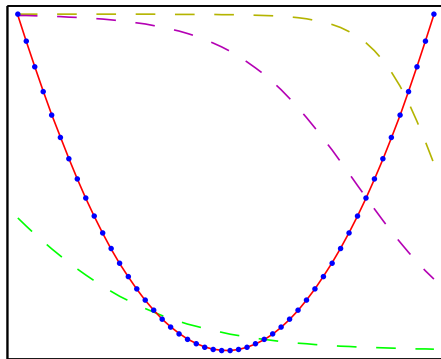
$$\sigma(x) = \max(0, x).$$

- Much faster to calculate, and to calculate its derivatives.



## Approximation Ability: $f(x) = x^2$

- 3 hidden units; logistic activation functions
- Blue dots are training points; Dashed lines are hidden unit outputs; Final output in Red.

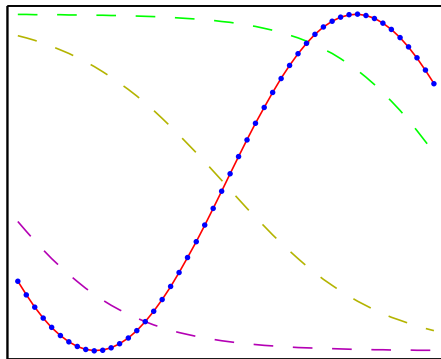


From Bishop's *Pattern Recognition and Machine Learning*, Fig 5.3



## Approximation Ability: $f(x) = \sin(x)$

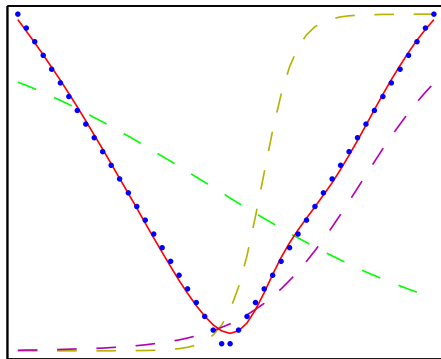
- 3 hidden units; logistic activation function
- Blue dots are training points; Dashed lines are hidden unit outputs; Final output in Red.



From Bishop's *Pattern Recognition and Machine Learning*, Fig 5.3

# Approximation Ability: $f(x) = |x|$

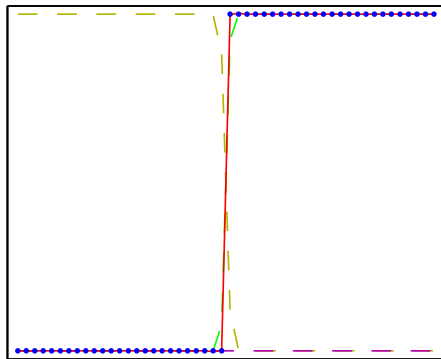
- 3 hidden units; logistic activation functions
- Blue dots are training points; Dashed lines are hidden unit outputs; Final output in Red.



From Bishop's *Pattern Recognition and Machine Learning*, Fig 5.3

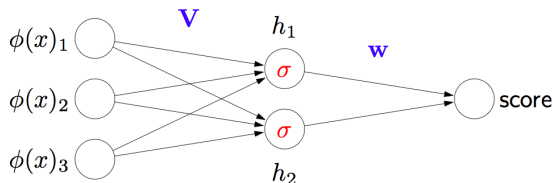
# Approximation Ability: $f(x) = 1(x > 0)$

- 3 hidden units; logistic activation function
- Blue dots are training points; Dashed lines are hidden unit outputs; Final output in Red.



From Bishop's *Pattern Recognition and Machine Learning*, Fig 5.3

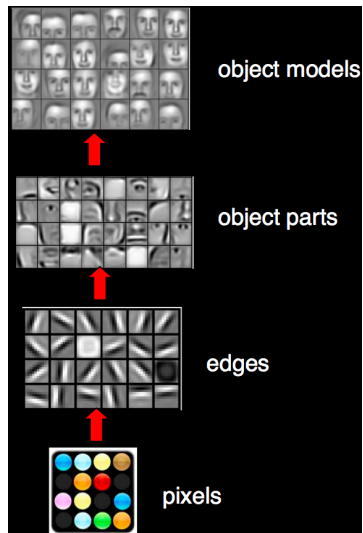
# Neural Network: Hidden Nodes as Learned Features



- Can interpret  $h_1$  and  $h_2$  as nonlinear features learned from data.

From Percy Liang's "Lecture 3" slides from Stanford's CS221, Autumn 2014.

# Facial Recognition: Learned Features



From Andrew Ng's CS229 Deep Learning slides

# Neural Network: The Hypothesis Space

- What hyperparameters describe a neural network?
  - Number of layers
  - Number of nodes in each hidden layer
  - Activation function (but so many to choose from)
- Example neural network hypothesis space:

$$\mathcal{F} = \{f : \mathbf{R}^d \rightarrow \mathbf{R} \mid f \text{ is a NN with 2 hidden layers, 500 nodes in each}\}$$

- Functions in  $\mathcal{F}$  **parameterized by the weights between nodes.**

# Neural Network: Loss Functions and Learning

- Neural networks give a **new hypothesis space**.
- But we can use all the **same loss functions** we've used before.
- Optimization method of choice: **stochastic gradient descent**.

# Neural Network: Objective Function

- In our simple network, the output score is given by

$$f(x) = w_1 \sigma(v_1^T \phi(x)) + w_2 \sigma(v_2^T \phi(x))$$

- Objective with square loss is then

$$J(w, v) = \sum_{i=1}^n (y_i - f_{w,v}(x_i))^2$$

- Note:  $J(w, v)$  is **not convex**.
  - makes optimization much more difficult
  - accounts for many of the “tricks of the trade”



# Learning with Back-Propagation

- Back-propagation is an **algorithm** for computing the SGD gradient
- Mathematically, it's not necessary.
- With lots of chain rule, you can work out the gradient by hand.
- Back-propagation is
  - a clean way to organize the computation of the gradient
  - an efficient way to compute the gradient
- Nice introduction to this perspective:
  - Stanford CS221 Lecture 3, Slides 63-96
  - <http://web.stanford.edu/class/cs221/lectures/learning2.pdf>

# Neural Network Regularization

- Neural networks are very expressive.
- Correspond to big hypothesis spaces.
- Many approaches are used for regularization.

# Tikhonov Regularization? Sure.

- Can add an  $\ell_2$  and/or  $\ell_1$  regularization terms to our objective:

$$J(w, v) = \sum_{i=1}^n (y_i - f_{w,v}(x_i))^2 + \lambda_1 \|w\|^2 + \lambda_2 \|v\|^2$$

- In neural network literature, this is often called **weight decay**.

# Regularization by Early Stopping

- As we train, check performance on validation set every once in a while.
- Don't stop immediately after validation error goes back up.
- The “**patience**” parameter: the number training rounds to continue after finding a minimum of validation error.
  - Start with  $\text{patience} = 10000$ .
  - Whenever we find a minimum at iteration  $T$ ,
    - Set  $\text{patience} \leftarrow \text{patience} + cT$ , for some constant  $c$ .
    - Then run at least  $\text{patience}$  extra iterations before stopping.

---

See <http://arxiv.org/pdf/1206.5533v2.pdf> for details.

# Max-Norm Regularization

- **Max-norm regularization:** Enforce max norm of incoming weight vector at every hidden node to be bounded:

$$\|w\|_2 \leq c.$$

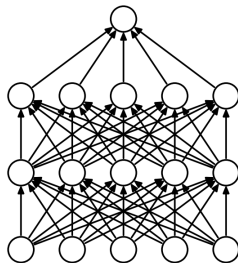
- Project any  $w$  that's too large onto ball of radius  $c$ .
- It's like  $\ell_2$ -complexity control, but locally at each node.
- Why?
  - There are heuristic justifications, but proof is in the performance.
  - We'll see below.

---

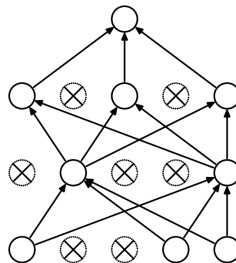
See <http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf> for details.

# Dropout for Regularization

- A recent trick for improving generalization performance is **dropout**.
- A fixed probability  $p$  is chosen.
- Before every stochastic gradient step,
  - each node is selected for “dropout” with probability  $p$
  - a dropout node is removed, along with its links
  - after the stochastic gradient step, all nodes are restored.



(a) Standard Neural Net

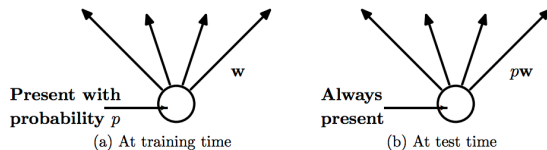


(b) After applying dropout.

Figure from <http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf>.

# Dropout for Regularization

- At prediction time
  - all nodes are present
  - outgoing weights are multiplied by  $p$ .



- Dropout probability set using a validation set, or just set at 0.5.
  - Closer to 0.8 usually works better for input units.

Figure from <http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf>.

# Dropout: Why might this help?

- Since any node may randomly disappear,
  - forced to “spread the knowledge” across the nodes.
- Each hidden only gets a randomly chosen sample of its inputs,
  - so won't become too reliant on any single input.
  - More robust.



# Dropout: Does it help?

| Method   | Unit Type | Architecture              | Error % |
|--|-----------|---------------------------|---------|
| Standard Neural Net (Simard et al., 2003)                  | Logistic  | 2 layers, 800 units       | 1.60    |
| SVM Gaussian kernel  | NA        | NA                        | 1.40    |
| Dropout NN   | Logistic  | 3 layers, 1024 units      | 1.35    |
| Dropout NN   | ReLU      | 3 layers, 1024 units      | 1.25    |
| Dropout NN + max-norm constraint                           | ReLU      | 3 layers, 1024 units      | 1.06    |
| Dropout NN + max-norm constraint                           | ReLU      | 3 layers, 2048 units      | 1.04    |
| Dropout NN + max-norm constraint                           | ReLU      | 2 layers, 4096 units      | 1.01    |
| Dropout NN + max-norm constraint                           | ReLU      | 2 layers, 8192 units      | 0.95    |
| Dropout NN + max-norm constraint (Goodfellow et al., 2013) | Maxout    | 2 layers, (5 × 240) units | 0.94    |

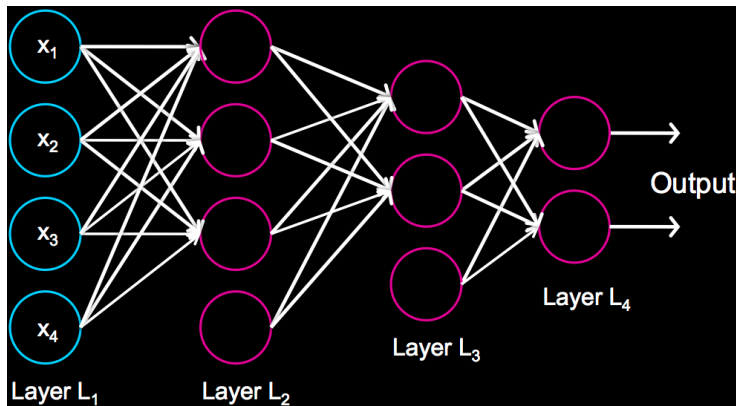
Figure from <http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf>.

# How big a network?

- How many hidden units?
- With proper regularization, too many doesn't hurt.
  - Except in computation time.

# Multiple Output Neural Networks

- Very easy to add extra outputs to neural network structure.



From Andrew Ng's CS229 Deep Learning slides  
(<http://cs229.stanford.edu/materials/CS229-DeepLearning.pdf>)

# Multitask Learning

- Suppose  $\mathcal{X} = \{\text{Natural Images}\}$ .
- We have two tasks:
  - Does the image have a cat?
  - Does the image have a dog?
- Can have one output for each task.
- Seems plausible that basic pixel features would be shared by tasks.
- Learn them on the same neural network – benefit both tasks.

# Single Task with “Extra Tasks”

- Only one task we're interested in.
- Gather data from related tasks.
- Train them along with the task you're interested in.
- No related tasks? Another trick:
  - Choose any input feature.
  - Change it's value to zero.
  - Make the prediction problem to predict the value of that feature.
  - Can help make model more robust (not depending too heavily on any single input).

# Multiclass Classification

- Could make each class a separate task / output.
- Suppose we have  $K$  classes.
- Use a one-hot encoding of each  $y_i \in \{1, \dots, K\}$ :

$$y_i = (y_{i1}, \dots, y_{ik}) \text{ with } y_{ik} = 1(y_i = k).$$

- $K$  output scores:  $f_1(x), \dots, f_K(x)$ . Each  $f_k$  is trained to predict 1 if class is  $k$ , 0 otherwise.
- Predict with  $f^*(x) = \arg \max_k [f_k(x)]$ .
- Old days: train each output separately, e.g. with square loss.

# Multiclass Classification: Cross-Entropy Loss

- Network can do better if it “knows” that classes are mutually exclusive.
- Need to introduce a joint loss across the outputs.
- Joint loss function (cross-entropy/deviance):

$$\ell(w, v) = - \sum_{i=1}^n \sum_{k=1}^K y_{ik} \log f_k(x_i),$$

where  $y_{ik} = 1(y_i = k)$ .

- We'll discuss this further when we talk about exponential families and generalized linear models.

# OverFeat: Features

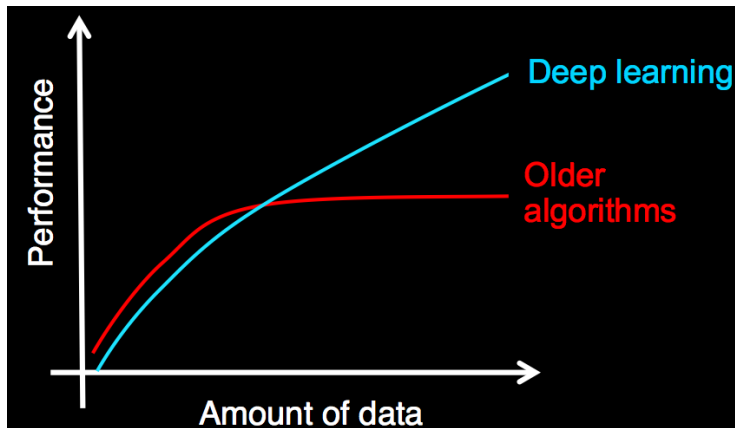
- OverFeat is a neural network for image classification
  - Trained on the huge ImageNet dataset
  - Lots of computing resources into training the network.
- All those hidden layers of the network are very valuable **features**.
  - Paper: “*CNN Features off-the-shelf: an Astounding Baseline for Recognition*”
  - Showed that using features from OverFeat makes it easy to achieve state-of-the-art performance on new vision tasks.

---

OverFeat code is at <https://github.com/sermanet/OverFeat>



# Neural Networks Benefit from Big Data



From Andrew Ng's CS229 Deep Learning slides  
(<http://cs229.stanford.edu/materials/CS229-DeepLearning.pdf>)

# Big Data Requires Big Resources

- Best results always involve GPU processing.
- Typically on huge networks.

## Google Brain



From Andrew Ng's CS229 Deep Learning slides  
(<http://cs229.stanford.edu/materials/CS229-DeepLearning.pdf>)

# Neural Networks: When to Use?

- Computer vision problems
  - All state of the art methods use neural networks
- Speech recognition
  - All state of the art methods use neural networks
- Natural Language problems?
  - Maybe. Check out “word2vec”  
<https://code.google.com/p/word2vec/>.
  - Represents words and phrases using real-valued vectors.
    - Potentially much better than bag of words.