

Machine Learning (CS 181):

9. Neural Networks 2

David Parkes and Sasha Rush

Contents

1 Logistic Regression / Neural Networks

2 Fitting Neural Networks

3 Backpropagation

4 Neural Network Architectures

5 Case Study: Image Recognition

Contents

1 Logistic Regression / Neural Networks

2 Fitting Neural Networks

3 Backpropagation

4 Neural Network Architectures

5 Case Study: Image Recognition

Linear Discriminative Model

We will require that the difference between classes in log space is linear:

$$h(\mathbf{x}; \mathbf{w}) = \ln p(y = 1|\mathbf{x}; \mathbf{w}) - \ln p(y = 0|\mathbf{x}; \mathbf{w})$$

Equivalently, $\ln(p(y = 1|\mathbf{x}; \mathbf{w})/(1 - p(y = 1|\mathbf{x}; \mathbf{w}))) = h(\mathbf{x}; \mathbf{w})$, and:

$$\frac{p(y = 1|\mathbf{x}; \mathbf{w})}{1 - p(y = 1|\mathbf{x}; \mathbf{w})} = \exp(h(\mathbf{x}; \mathbf{w}))$$

Solving for $p(y = 1|\mathbf{x}; \mathbf{w})$, we have

$$p(y = 1|\mathbf{x}; \mathbf{w}) = \frac{\exp(h(\mathbf{x}; \mathbf{w}))}{1 + \exp(h(\mathbf{x}; \mathbf{w}))}, \quad (1)$$

$$p(y = 0|\mathbf{x}; \mathbf{w}) = \frac{1}{1 + \exp(h(\mathbf{x}; \mathbf{w}))}$$

Finally, call (1) the logistic sigmoid activation:

$$\sigma(h) = (1 + \exp -h)^{-1}$$

Linear Discriminative Model

We will require that the difference between classes in log space is linear:

$$h(\mathbf{x}; \mathbf{w}) = \ln p(y = 1|\mathbf{x}; \mathbf{w}) - \ln p(y = 0|\mathbf{x}; \mathbf{w})$$

Equivalently, $\ln(p(y = 1|\mathbf{x}; \mathbf{w})/(1 - p(y = 1|\mathbf{x}; \mathbf{w}))) = h(\mathbf{x}; \mathbf{w})$, and:

$$\frac{p(y = 1|\mathbf{x}; \mathbf{w})}{1 - p(y = 1|\mathbf{x}; \mathbf{w})} = \exp(h(\mathbf{x}; \mathbf{w}))$$

Solving for $p(y = 1|\mathbf{x}; \mathbf{w})$, we have

$$p(y = 1|\mathbf{x}; \mathbf{w}) = \frac{\exp(h(\mathbf{x}; \mathbf{w}))}{1 + \exp(h(\mathbf{x}; \mathbf{w}))}, \quad (1)$$

$$p(y = 0|\mathbf{x}; \mathbf{w}) = \frac{1}{1 + \exp(h(\mathbf{x}; \mathbf{w}))}$$

Finally, call (1) the logistic sigmoid activation:

$$\sigma(h) = (1 + \exp -h)^{-1}$$

Linear Discriminative Model

We will require that the difference between classes in log space is linear:

$$h(\mathbf{x}; \mathbf{w}) = \ln p(y = 1 | \mathbf{x}; \mathbf{w}) - \ln p(y = 0 | \mathbf{x}; \mathbf{w})$$

Equivalently, $\ln(p(y = 1 | \mathbf{x}; \mathbf{w}) / (1 - p(y = 1 | \mathbf{x}; \mathbf{w}))) = h(\mathbf{x}; \mathbf{w})$, and:

$$\frac{p(y = 1 | \mathbf{x}; \mathbf{w})}{1 - p(y = 1 | \mathbf{x}; \mathbf{w})} = \exp(h(\mathbf{x}; \mathbf{w}))$$

Solving for $p(y = 1 | \mathbf{x}; \mathbf{w})$, we have

$$p(y = 1 | \mathbf{x}; \mathbf{w}) = \frac{\exp(h(\mathbf{x}; \mathbf{w}))}{1 + \exp(h(\mathbf{x}; \mathbf{w}))}, \quad (1)$$

$$p(y = 0 | \mathbf{x}; \mathbf{w}) = \frac{1}{1 + \exp(h(\mathbf{x}; \mathbf{w}))}$$

Finally, call (1) the logistic sigmoid activation:

$$\sigma(h) = (1 + \exp -h)^{-1}$$

Linear Discriminative Model

We will require that the difference between classes in log space is linear:

$$h(\mathbf{x}; \mathbf{w}) = \ln p(y = 1 | \mathbf{x}; \mathbf{w}) - \ln p(y = 0 | \mathbf{x}; \mathbf{w})$$

Equivalently, $\ln(p(y = 1 | \mathbf{x}; \mathbf{w}) / (1 - p(y = 1 | \mathbf{x}; \mathbf{w}))) = h(\mathbf{x}; \mathbf{w})$, and:

$$\frac{p(y = 1 | \mathbf{x}; \mathbf{w})}{1 - p(y = 1 | \mathbf{x}; \mathbf{w})} = \exp(h(\mathbf{x}; \mathbf{w}))$$

Solving for $p(y = 1 | \mathbf{x}; \mathbf{w})$, we have

$$p(y = 1 | \mathbf{x}; \mathbf{w}) = \frac{\exp(h(\mathbf{x}; \mathbf{w}))}{1 + \exp(h(\mathbf{x}; \mathbf{w}))}, \quad (1)$$

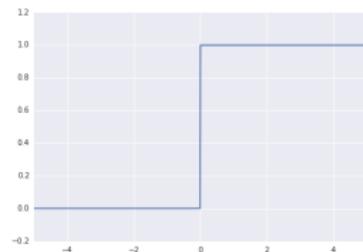
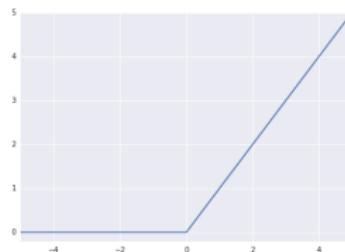
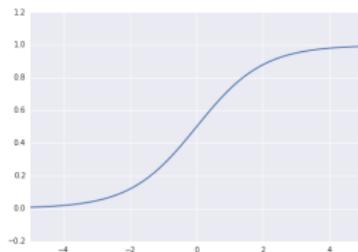
$$p(y = 0 | \mathbf{x}; \mathbf{w}) = \frac{1}{1 + \exp(h(\mathbf{x}; \mathbf{w}))}$$

Finally, call (1) the **logistic sigmoid** activation:

$$\sigma(h) = (1 + \exp -h)^{-1}$$

Activation Functions

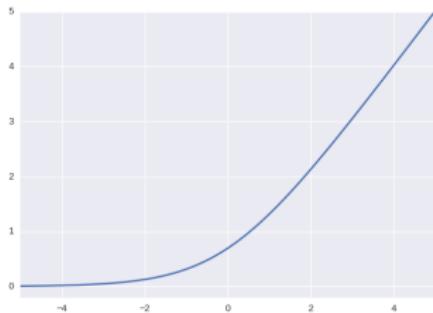
- Activation functions provide transformations of input.
- Good activations also provide useful derivatives. (0/1 does not.)



Sigmoid versus ReLU versus 0/1. x-axis input e.g $h(\mathbf{x}; \mathbf{w})$, y-axis output.

Algorithms come from Loss Function

Direct comparison of **negative ln σ** and ReLU in $\mathcal{Y} = \{0, 1\}$ notation



Logistic regression loss \mathcal{L}_σ versus Perceptron loss \mathcal{L}_{perc} . x -axis is misclassification error, y -axis is increased loss.

$$\mathcal{L}_\sigma(\mathbf{w}) = \sum_{i=1}^n -\ln \sigma(h_i)^{y_i} (1 - \sigma(h_i))^{1-y_i}$$

$$\mathcal{L}_{perc}(\mathbf{w}) = \sum_{i=1}^n f_{relu}(-h_i)^{y_i} f_{relu}(h_i)^{1-y_i}$$

Neural Networks: Matrix-view

Define entire adaptive basis layer as d problems,

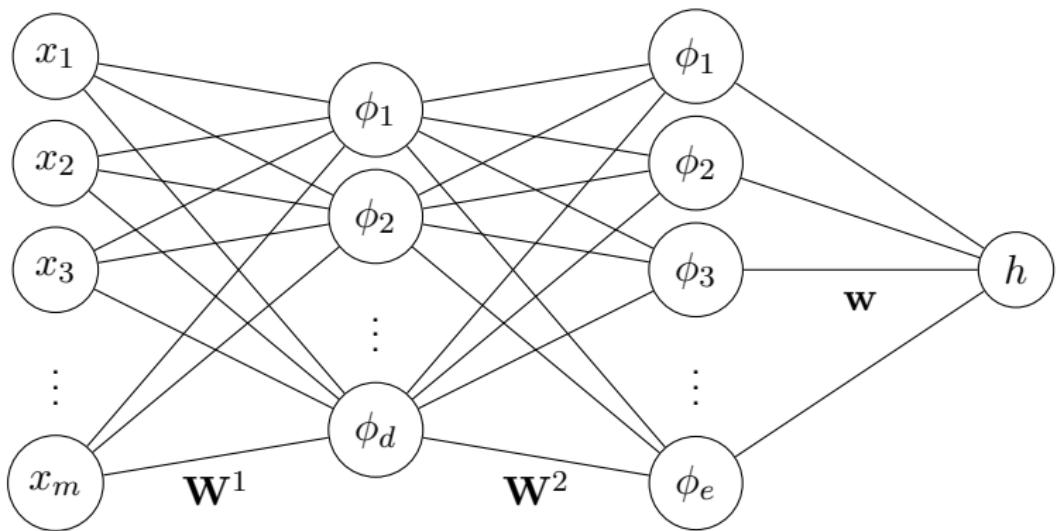
$$\phi(\mathbf{x}; \mathbf{W}^1, \mathbf{w}_0^1) = \sigma(\mathbf{W}^1 \mathbf{x} + \mathbf{w}_0^1)$$

Where σ is pointwise sigmoid and $\mathbf{W}^1 \in \mathbb{R}^{d \times m}$

Full classification problem:

$$\begin{aligned} h(\mathbf{x}; \mathbf{w}, w_0, \mathbf{W}^1, \mathbf{w}_0^1) &= \mathbf{w}^\top \phi(\mathbf{x}; \mathbf{W}^1, \mathbf{w}_0^1) + w_0 \\ &= \mathbf{w}^\top \sigma(\mathbf{W}^1 \mathbf{x} + \mathbf{w}_0^1) + w_0 \end{aligned}$$

Classical Neural Network Diagram



Contents

1 Logistic Regression / Neural Networks

2 Fitting Neural Networks

3 Backpropagation

4 Neural Network Architectures

5 Case Study: Image Recognition

Loss Function

- Neural networks are an adaptive *basis*.
- Can use apply to different problems.
 - (1) Classification, use final logistic sigmoid and MLE.

$$\mathcal{L}(\mathbf{w}, \mathbf{W}^1) = -\sum_{i=1}^n \ln p(y_i | \mathbf{x}_i; \mathbf{w}, \mathbf{W}^1)$$

- (2) Regression, use least squares loss on top of neural network.

$$\mathcal{L}(\mathbf{w}, \mathbf{W}^1) = \frac{1}{2} \sum_{i=1}^n (y_i - h(\mathbf{x}_i; \mathbf{w}, \mathbf{W}^1))^2$$

Loss Function

- Neural networks are an adaptive *basis*.
- Can use apply to different problems.
- (1) Classification, use final logistic sigmoid and MLE.

$$\mathcal{L}(\mathbf{w}, \mathbf{W}^1) = - \sum_{i=1}^n \ln p(y_i | \mathbf{x}_i; \mathbf{w}, \mathbf{W}^1)$$

- (2) Regression, use least squares loss on top of neural network.

$$\mathcal{L}(\mathbf{w}, \mathbf{W}^1) = \frac{1}{2} \sum_{i=1}^n (y_i - h(\mathbf{x}_i; \mathbf{w}, \mathbf{W}^1))^2$$

Loss Function

- Neural networks are an adaptive *basis*.
- Can be applied to different problems.
- (1) Classification, use final logistic sigmoid and MLE.

$$\mathcal{L}(\mathbf{w}, \mathbf{W}^1) = - \sum_{i=1}^n \ln p(y_i | \mathbf{x}_i; \mathbf{w}, \mathbf{W}^1)$$

- (2) Regression, use least squares loss on top of neural network.

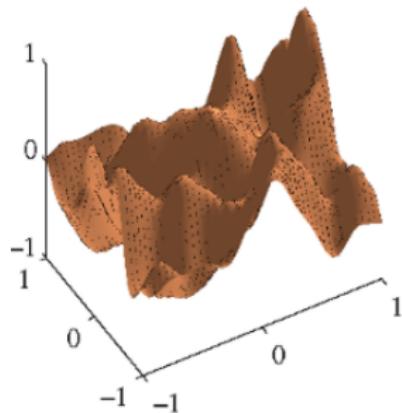
$$\mathcal{L}(\mathbf{w}, \mathbf{W}^1) = \frac{1}{2} \sum_{i=1}^n (y_i - h(\mathbf{x}_i; \mathbf{w}, \mathbf{W}^1))^2$$

Training With Gradient Descent

- Need gradient for w (as before)
- Also need gradients for W^1 (**new**)
- Gradients will depend on loss and structure of network.

Non-Convexity

- Important issue: neural network loss is non-convex!



Non-convex loss surface.

- Gradient descent will not find a global minimum.
- Can be a real issue (although often ignored in practice).

Regression Loss

$$\mathcal{L}(\mathbf{w}, \mathbf{W}) = \frac{1}{2} \sum_{i=1}^n (y_i - h(\mathbf{x}_i; \mathbf{w}, \mathbf{W}))^2 = \frac{1}{2} \sum_{i=1}^n (y_i - h_i)^2$$

Take gradients wrt h_i :

$$\frac{\partial}{\partial h_i} \mathcal{L} = -(y_i - h_i)$$

- Need gradient for \mathbf{w} and \mathbf{W}^1 , use chain rule.

Regression Loss

$$\mathcal{L}(\mathbf{w}, \mathbf{W}) = \frac{1}{2} \sum_{i=1}^n (y_i - h(\mathbf{x}_i; \mathbf{w}, \mathbf{W}))^2 = \frac{1}{2} \sum_{i=1}^n (y_i - h_i)^2$$

Take gradients wrt h_i :

$$\frac{\partial}{\partial h_i} \mathcal{L} = -(y_i - h_i)$$

- Need gradient for \mathbf{w} and \mathbf{W}^1 , use chain rule.

Interlude: Vector-Valued Chain Rule

$$\mathbf{w} \in \mathbb{R}^m \rightarrow \mathbf{g}(\mathbf{w}) \in \mathbb{R}^n \rightarrow f(\mathbf{g}(\mathbf{w})) \in \mathbb{R}$$

$$\frac{\partial f(\mathbf{g}(\mathbf{w}))}{\partial \mathbf{w}} \in \mathbb{R}^n \leftarrow \frac{\partial f(\mathbf{g}(\mathbf{w}))}{\partial \mathbf{g}(\mathbf{w})} \in \mathbb{R}^d \leftarrow f(\mathbf{g}(\mathbf{w})) \in \mathbb{R}$$

Chain rule, vector-valued functions:

$$\frac{\partial f(\mathbf{g}(\mathbf{w}))}{\partial \mathbf{w}} = \sum_{i=1}^n \frac{\partial g(\mathbf{w})_i}{\partial \mathbf{w}} \frac{\partial f(g(\mathbf{w}))}{\partial g(\mathbf{w})_i}$$

Interlude: Vector-Valued Chain Rule

$$\mathbf{w} \in \mathbb{R}^m \rightarrow \mathbf{g}(\mathbf{w}) \in \mathbb{R}^n \rightarrow f(\mathbf{g}(\mathbf{w})) \in \mathbb{R}$$

$$\frac{\partial f(\mathbf{g}(\mathbf{w}))}{\partial \mathbf{w}} \in \mathbb{R}^n \leftarrow \frac{\partial f(\mathbf{g}(\mathbf{w}))}{\partial \mathbf{g}(\mathbf{w})} \in \mathbb{R}^d \leftarrow f(\mathbf{g}(\mathbf{w})) \in \mathbb{R}$$

Chain rule, vector-valued functions:

$$\frac{\partial f(\mathbf{g}(\mathbf{w}))}{\partial \mathbf{w}} = \sum_{i=1}^n \frac{\partial g(\mathbf{w})_i}{\partial \mathbf{w}} \frac{\partial f(g(\mathbf{w}))}{\partial g(\mathbf{w})_i}$$

Interlude: Vector-Valued Chain Rule

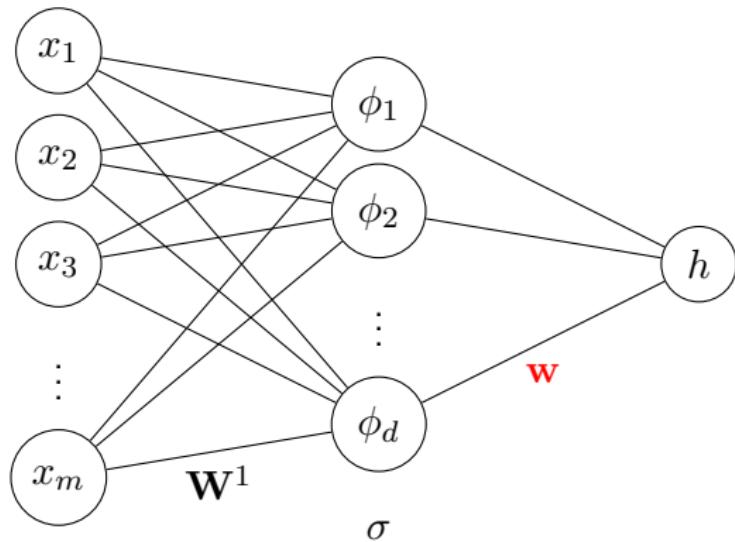
$$\mathbf{w} \in \mathbb{R}^m \rightarrow \mathbf{g}(\mathbf{w}) \in \mathbb{R}^n \rightarrow f(\mathbf{g}(\mathbf{w})) \in \mathbb{R}$$

$$\frac{\partial f(\mathbf{g}(\mathbf{w}))}{\partial \mathbf{w}} \in \mathbb{R}^n \leftarrow \frac{\partial f(\mathbf{g}(\mathbf{w}))}{\partial \mathbf{g}(\mathbf{w})} \in \mathbb{R}^d \leftarrow f(\mathbf{g}(\mathbf{w})) \in \mathbb{R}$$

Chain rule, vector-valued functions:

$$\frac{\partial f(\mathbf{g}(\mathbf{w}))}{\partial \mathbf{w}} = \sum_{i=1}^n \frac{\partial g(\mathbf{w})_i}{\partial \mathbf{w}} \frac{\partial f(g(\mathbf{w}))}{\partial g(\mathbf{w})_i}$$

Neural Network Diagram



Gradient for \mathbf{w}

$$\mathcal{L} = \frac{1}{2} \sum_{i=1}^n (y_i - (\mathbf{w}^\top \phi_i + w_0))^2$$

$$\begin{array}{ccc} h_i & & \mathcal{L} \\ \mathbf{w}^\top \phi_i + w_0 & \Rightarrow & \frac{1}{2}(y_i - h_i)^2 \end{array}$$

$$\begin{array}{ccc} \phi_i & \leftarrow & -(y_i - h_i) \\ \frac{\partial h_i}{\partial \mathbf{w}} & & \frac{\partial \mathcal{L}}{\partial h_i} \end{array}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \sum_{i=1}^n \frac{\partial h_i}{\partial \mathbf{w}} \frac{\partial \mathcal{L}}{\partial h_i} = \sum_{i=1}^n \phi_i \times -(y_i - h_i)$$

Gradient for \mathbf{w}

$$\mathcal{L} = \frac{1}{2} \sum_{i=1}^n (y_i - (\mathbf{w}^\top \phi_i + w_0))^2$$

$$\begin{array}{ll} h_i & \mathcal{L} \\ \mathbf{w}^\top \phi_i + w_0 & \Rightarrow \frac{1}{2}(y_i - h_i)^2 \end{array}$$

$$\begin{array}{ll} \phi_i & \Leftarrow -(y_i - h_i) \\ \frac{\partial h_i}{\partial \mathbf{w}} & \frac{\partial \mathcal{L}}{\partial h_i} \end{array}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \sum_{i=1}^n \frac{\partial h_i}{\partial \mathbf{w}} \frac{\partial \mathcal{L}}{\partial h_i} = \sum_{i=1}^n \phi_i \times -(y_i - h_i)$$

Gradient for \mathbf{w}

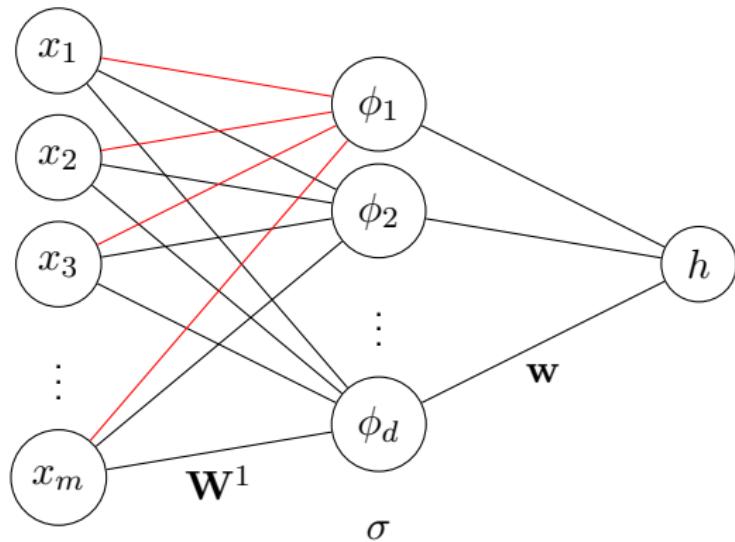
$$\mathcal{L} = \frac{1}{2} \sum_{i=1}^n (y_i - (\mathbf{w}^\top \phi_i + w_0))^2$$

$$\begin{array}{ll} h_i & \mathcal{L} \\ \mathbf{w}^\top \phi_i + w_0 & \Rightarrow \frac{1}{2}(y_i - h_i)^2 \end{array}$$

$$\begin{array}{ll} \phi_i & \Leftarrow -(y_i - h_i) \\ \frac{\partial h_i}{\partial \mathbf{w}} & \frac{\partial \mathcal{L}}{\partial h_i} \end{array}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \sum_{i=1}^n \frac{\partial h_i}{\partial \mathbf{w}} \frac{\partial \mathcal{L}}{\partial h_i} = \sum_{i=1}^n \phi_i \times -(y_i - h_i)$$

Neural Network Diagram



Gradient for \mathbf{W}^1

$$\mathcal{L} = \frac{1}{2} \sum_{i=1}^n (y_i - (\mathbf{w}^\top \boldsymbol{\sigma}(\mathbf{W}^1 \mathbf{x}_i + \mathbf{w}^1) + w_0))^2$$

How does \mathbf{w}_j^1 impact loss? (Adaptive parameter for ϕ_j)

$$\begin{array}{cccccc} g_{ij} & & \phi_{ij} & & h_i & & \mathcal{L} \\ \mathbf{w}_j^{1\top} \mathbf{x}_i + w_{j0}^1 & \Rightarrow & \sigma(g_{ij}) & \Rightarrow & \mathbf{w}^\top \phi_i + w_0 & \Rightarrow & \frac{1}{2}(y_i - h_i)^2 \end{array}$$

$$\begin{array}{cccccc} \mathbf{x}_i & & \Leftarrow & \sigma'(g_{ij}) & \Leftarrow & w_j & \Leftarrow & -(y_i - h_i) \\ \frac{\partial g_{ij}}{\partial \mathbf{w}_j^1} & & & \frac{\partial \phi_{ij}}{\partial g_{ij}} & & \frac{\partial h_i}{\partial \phi_{ij}} & & \frac{\partial \mathcal{L}}{\partial h_i} \end{array}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}_j^1} = \sum_{i=1}^n \mathbf{x}_i \times \sigma'(g_{ij}) \times w_j \times -(y_i - h_i)$$

Gradient for \mathbf{W}^1

$$\mathcal{L} = \frac{1}{2} \sum_{i=1}^n (y_i - (\mathbf{w}^\top \boldsymbol{\sigma}(\mathbf{W}^1 \mathbf{x}_i + \mathbf{w}^1) + w_0))^2$$

How does \mathbf{w}_j^1 impact loss? (Adaptive parameter for ϕ_j)

$$\begin{array}{cccccc} g_{ij} & \phi_{ij} & h_i & & \mathcal{L} \\ \mathbf{w}_j^{1\top} \mathbf{x}_i + w_{j0}^1 & \Rightarrow & \sigma(g_{ij}) & \Rightarrow & \mathbf{w}^\top \phi_i + w_0 & \Rightarrow & \frac{1}{2}(y_i - h_i)^2 \end{array}$$

$$\begin{array}{ccccc} \mathbf{x}_i & \Leftarrow & \sigma'(g_{ij}) & \Leftarrow & w_j \\ \frac{\partial g_{ij}}{\partial \mathbf{w}_j^1} & & \frac{\partial \phi_{ij}}{\partial g_{ij}} & & \frac{\partial h_i}{\partial \phi_{ij}} \\ & & & & \frac{\partial \mathcal{L}}{\partial h_i} \end{array}$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}_j^1} = \sum_{i=1}^n \mathbf{x}_i \times \sigma'(g_{ij}) \times w_j \times -(y_i - h_i)$$

Contents

1 Logistic Regression / Neural Networks

2 Fitting Neural Networks

3 Backpropagation

4 Neural Network Architectures

5 Case Study: Image Recognition

Chain Rule

- The chain rule is applied to compute gradient of loss wrt \mathbf{W} .

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}_j^1} = \sum_{i=1}^n \mathbf{x}_i \times \sigma'(g_{ij}) \times w_j \times -(y_i - h_i)$$

- Note that this chain rule step makes use of intermediate terms h , g , ϕ in process
- Computing these terms is called the forward pass.
- Applying the chain rule is called the backward pass.
- In general, methodology is known as backpropagation.

Chain Rule

- The chain rule is applied to compute gradient of loss wrt \mathbf{W} .

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}_j^1} = \sum_{i=1}^n \mathbf{x}_i \times \sigma'(g_{ij}) \times w_j \times -(y_i - h_i)$$

- Note that this chain rule step makes use of intermediate terms h , g , ϕ in process
- Computing these terms is called the **forward** pass.
- Applying the chain rule is called the **backward** pass.
- In general, methodology is known as **backpropagation**.

Backpropagation (Deep Network)

■ Forward Step (f-prop):

Compute

$$\mathcal{L}(\phi^L(\dots \phi^1(\mathbf{x})))$$

Save intermediary values for all layers l .

$$\phi^l = \phi^l(\dots \phi^1(\mathbf{x}))$$

■ Backward Step (b-prop):

$$\dot{\phi}^l = \frac{\partial \mathcal{L}}{\partial \phi^l(\dots \phi_1(\mathbf{x}))} = \sum_{j=1}^d \frac{\partial \phi_j^{(l+1)}}{\partial \phi_j^l} \frac{\partial \mathcal{L}}{\partial \phi_j^{(l+1)}}$$

■ Compute parameter gradients

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}^l} = \sum_{j=1}^d \frac{\partial \phi_j^l}{\partial \mathbf{w}^l} \dot{\phi}_j^l$$

Backpropagation (Deep Network)

- Forward Step (f-prop):

Compute

$$\mathcal{L}(\phi^L(\dots \phi^1(\mathbf{x})))$$

Save intermediary values for all layers l .

$$\phi^l = \phi^l(\dots \phi^1(\mathbf{x}))$$

- Backward Step (b-prop):

$$\dot{\phi}^l = \frac{\partial \mathcal{L}}{\partial \phi^l(\dots \phi_1(\mathbf{x}))} = \sum_{j=1}^d \frac{\partial \phi_j^{(l+1)}}{\partial \phi_j^l} \frac{\partial \mathcal{L}}{\partial \phi_j^{(l+1)}}$$

- Compute parameter gradients

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}^l} = \sum_{j=1}^d \frac{\partial \phi_j^l}{\partial \mathbf{w}^l} \dot{\phi}_j^l$$

Backpropagation (Deep Network)

■ Forward Step (f-prop):

Compute

$$\mathcal{L}(\phi^L(\dots \phi^1(\mathbf{x})))$$

Save intermediary values for all layers l .

$$\phi^l = \phi^l(\dots \phi^1(\mathbf{x}))$$

■ Backward Step (b-prop):

$$\dot{\phi}^l = \frac{\partial \mathcal{L}}{\partial \phi^l(\dots \phi_1(\mathbf{x}))} = \sum_{j=1}^d \frac{\partial \phi_j^{(l+1)}}{\partial \phi_j^l} \frac{\partial \mathcal{L}}{\partial \phi_j^{(l+1)}}$$

■ Compute parameter gradients

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}^l} = \sum_{j=1}^d \frac{\partial \phi_j^l}{\partial \mathbf{w}^l} \dot{\phi}_j^l$$

Backpropagation (Deep Network)

- Forward Step (f-prop):

Compute

$$\mathcal{L}(\phi^L(\dots \phi^1(\mathbf{x})))$$

Save intermediary values for all layers l .

$$\phi^l = \phi^l(\dots \phi^1(\mathbf{x}))$$

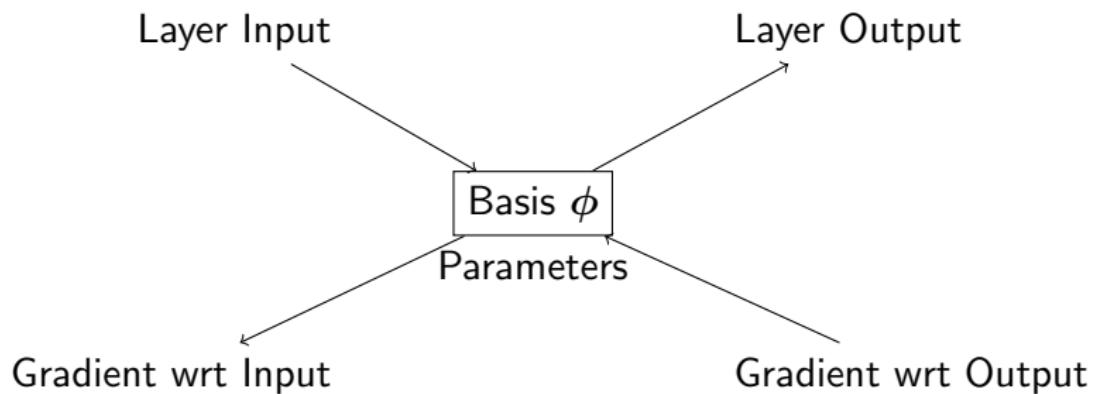
- Backward Step (b-prop):

$$\dot{\phi}^l = \frac{\partial \mathcal{L}}{\partial \phi^l(\dots \phi_1(\mathbf{x}))} = \sum_{j=1}^d \frac{\partial \phi_j^{(l+1)}}{\partial \phi_j^l} \frac{\partial \mathcal{L}}{\partial \phi_j^{(l+1)}}$$

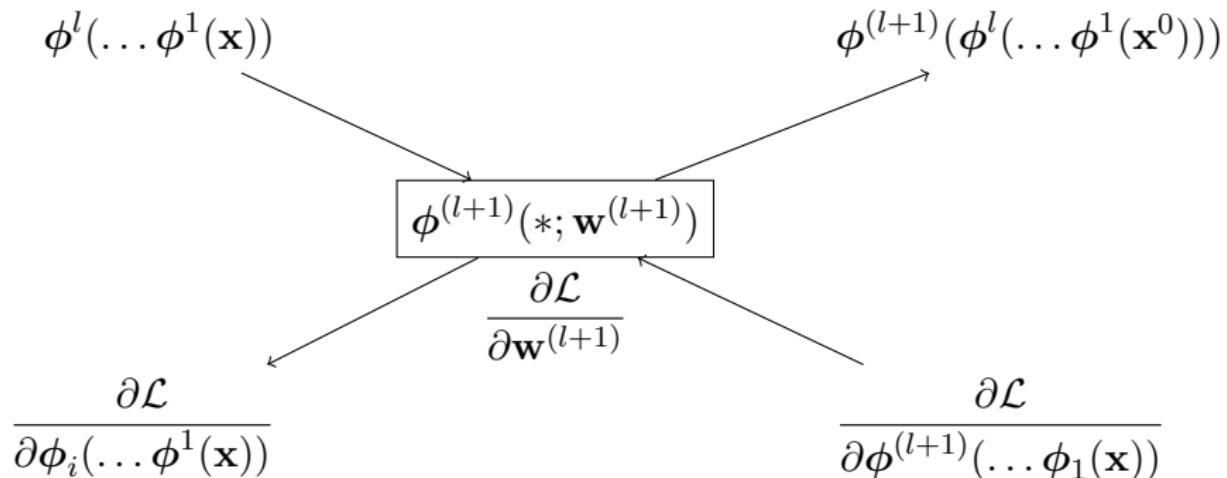
- Compute parameter gradients

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}^l} = \sum_{j=1}^d \frac{\partial \phi_j^l}{\partial \mathbf{w}^l} \dot{\phi}_j^l$$

Backpropagation: High Level



Backpropagation: Data flow



Computation Graph

Unit-based specification of NN

- Every function is represented as a unit.
- Responsibilities:
 1. Compute $\phi^{(l+1)}(\mathbf{x}; \mathbf{W}^{(l+1)})$ (forward)
 2. Compute chain-rule given $\frac{\partial \mathcal{L}}{\partial \phi^{(l+1)}(\dots \phi^1(\mathbf{x}))}$ and ϕ^l (backward)
 3. Compute parameter gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l+1)}}$ (update)

Example

Contents

1 Logistic Regression / Neural Networks

2 Fitting Neural Networks

3 Backpropagation

4 Neural Network Architectures

5 Case Study: Image Recognition

So far, adaptive basis functions are mini logistic regressions:

$$\phi(\mathbf{x}; \mathbf{W}^1, \mathbf{w}_0^1) = \sigma(\mathbf{W}^1 \mathbf{x} + \mathbf{w}_0^1)$$

But backpropagation can work with arbitrary differentiable functions.

- What about other activation functions?
- What about other structures?

Warning: Not much theory here.

So far, adaptive basis functions are mini logistic regressions:

$$\phi(\mathbf{x}; \mathbf{W}^1, \mathbf{w}_0^1) = \sigma(\mathbf{W}^1 \mathbf{x} + \mathbf{w}_0^1)$$

But backpropagation can work with arbitrary differentiable functions.

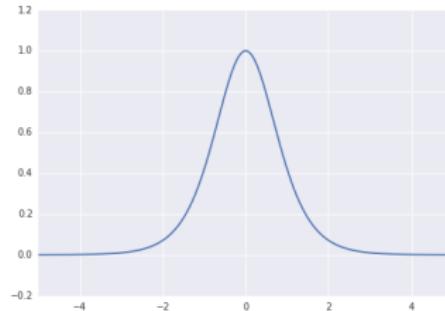
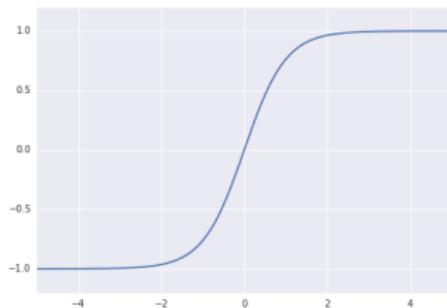
- What about other activation functions?
- What about other structures?

Warning: Not much theory here.

Activations: Tanh

Hyperbolic Tangent:

$$\tanh(h) = \frac{\exp(h) - \exp(-h)}{\exp(h) + \exp(-h)}$$

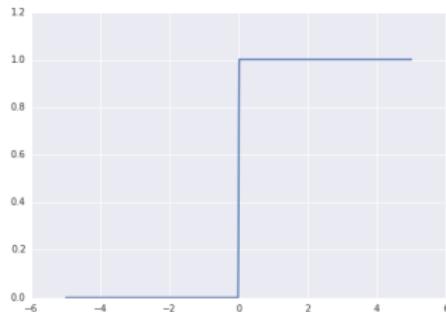
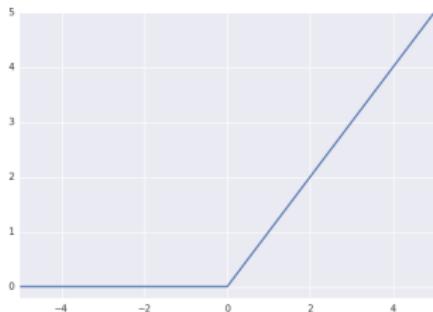


- Intuition: Similar to logistic sigmoid, but range between -1 and 1.
- More widely used when no probabilistic interpretation.

Activations: ReLU

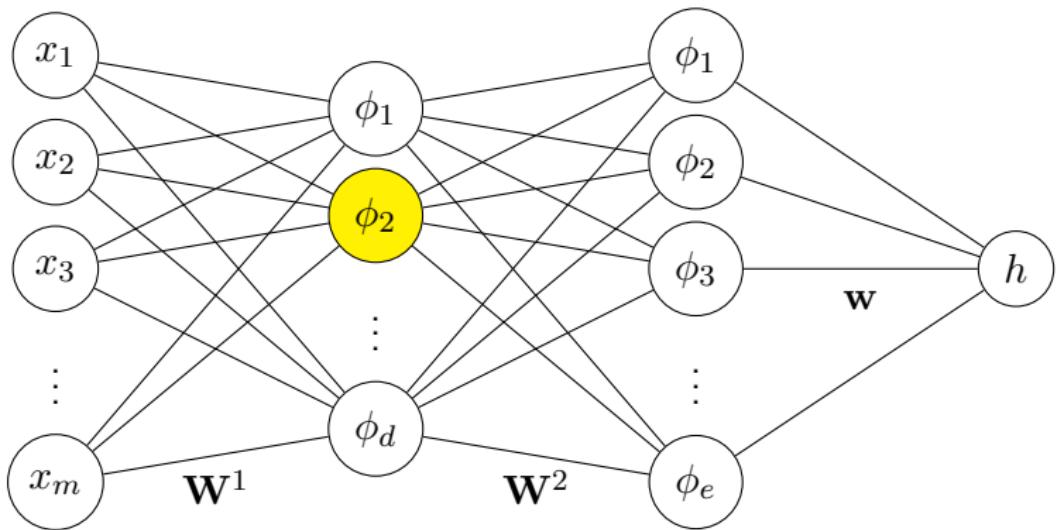
Rectified Linear Unit:

$$\text{ReLU}(h) = \max\{0, h\}$$



- Intuition: Each basis function is linear hinge.
- Popular in vision applications. Very fast to compute, no upper cutoff.
- No gradient (saturation) when below 0.

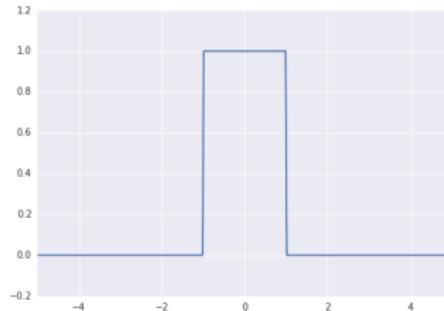
Issues: Saturation



Activations: Hard Tanh

Piecewise linear:

$$\text{hardtanh}(h) = \begin{cases} -1 & h < -1 \\ h & -1 \leq h \leq 1 \\ 1 & h > 1 \end{cases}$$



- Intuition: properties of both ReLU and tanh.

Architectures: Shared Weights

Neural networks quickly reach a large number of parameters

$$\mathbf{W}^l \in \mathbb{R}^{m \times d}$$

- Balance between introducing new learned features and overfitting.
- One solution is to have shared weights between ϕ_j 's.
- Obviously cannot share all weights, symmetric output.

Architectures: Shared Weights

Neural networks quickly reach a large number of parameters

$$\mathbf{W}^l \in \mathbb{R}^{m \times d}$$

- Balance between introducing new learned features and overfitting.
- One solution is to have **shared** weights between ϕ_j 's.
- Obviously cannot share all weights, symmetric output.

On-Diagonal Weight Sharing

$$\mathbf{W}^1 = \begin{bmatrix} w_1 & w_2 & w_3 & 0 & \dots & & 0 \\ 0 & w_1 & w_2 & w_3 & 0 & \dots & 0 \\ 0 & 0 & w_1 & w_2 & w_3 & 0 & \dots & 0 \\ \vdots & & & & & & & \\ \dots & & & 0 & w_1 & w_2 & w_3 & 0 \\ & \dots & & 0 & w_1 & w_2 & w_3 & 0 \end{bmatrix}$$

- Same w_1 , w_2 , w_3 at each location.
- Only three trainable parameters here.

On-Diagonal Weight Sharing (2)

$$\mathbf{W}^1 \mathbf{x}$$

$$\begin{bmatrix} w_1 & w_2 & w_3 & 0 & \dots & & 0 \\ 0 & w_1 & w_2 & w_3 & 0 & \dots & 0 \\ 0 & 0 & w_1 & w_2 & w_3 & 0 & \dots & 0 \\ \vdots & & & & & & & \vdots \\ \dots & & 0 & w_1 & w_2 & w_3 & 0 \\ & \dots & & 0 & w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_4 \\ x_5 \end{bmatrix}$$

- Each basis applies the same **filter** to a local region of x

$$\phi_j = \sigma(w_1 x_j + w_2 x_{j+1} + w_3 x_{j+2} + w_0)$$

- This type of operation is known as a **convolution**

Contents

1 Logistic Regression / Neural Networks

2 Fitting Neural Networks

3 Backpropagation

4 Neural Network Architectures

5 Case Study: Image Recognition

Example: Digit Classification

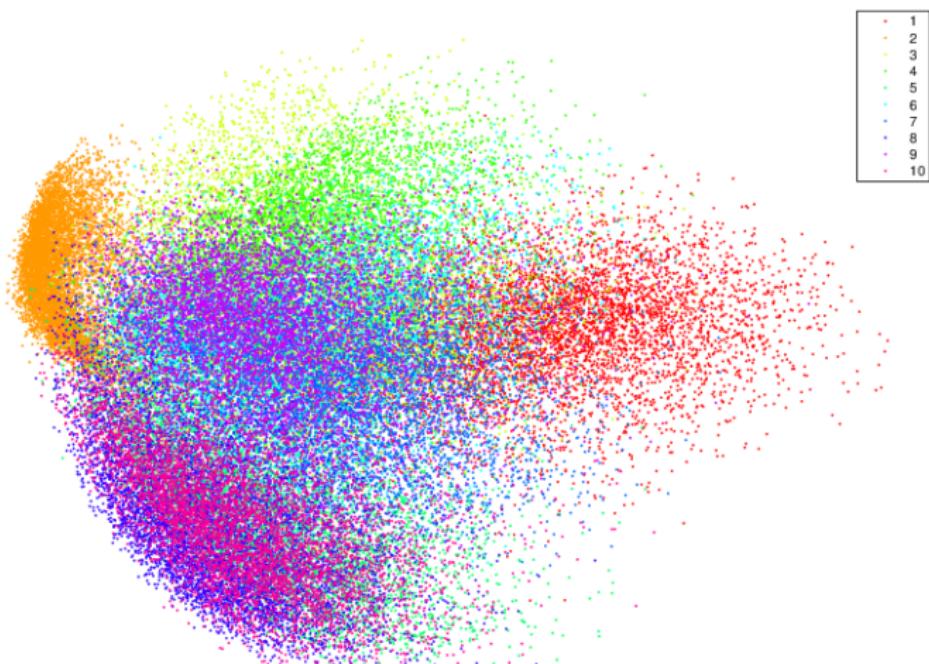
- Data: Handwritten US zip codes

3	6	8	1	7	9	6	6	9	1
6	7	5	7	8	6	3	4	8	5
2	1	7	9	7	1	2	8	4	5
4	8	1	9	0	1	8	8	9	4
7	6	1	8	6	4	1	5	6	0
7	5	9	2	6	5	8	1	9	7
1	2	2	2	2	3	4	4	8	0
0	2	3	8	0	7	3	8	5	7
0	1	4	6	4	6	0	2	4	3
7	1	2	8	7	6	9	8	6	1

Digit Recognition

- MNist: collection of handwritten digits.
- Problem: given B&W image, classify as digit.

PCA (16% Variance Explained)



Multiclass Logistic Regression

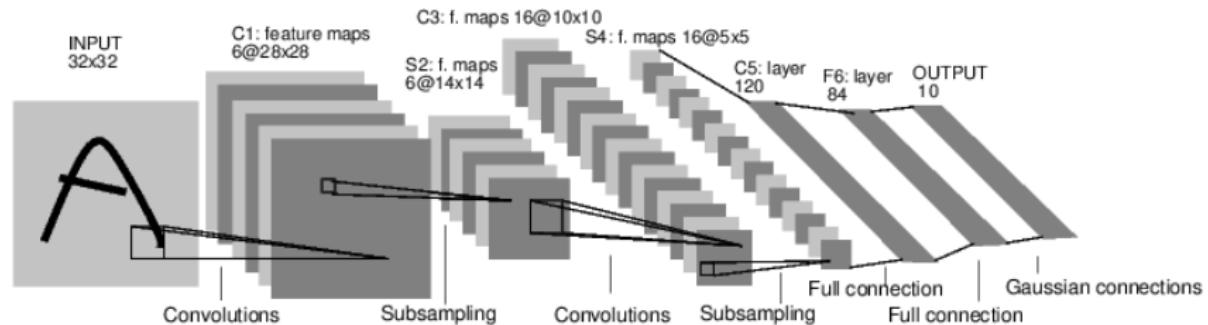
- Multiclass logistic regression uses a \mathbf{w}_ℓ for each class.
- Generalization of sigmoid is softmax function.

$$p(\mathbf{y} = C_k | \mathbf{x}; \{\mathbf{w}_\ell\}_{\ell=1}^c) = \frac{\exp(\mathbf{x}^\top \mathbf{w}_k)}{\sum_\ell \exp(\mathbf{x}^\top \mathbf{w}_\ell)}$$

(Derivation and exploration on homework.)

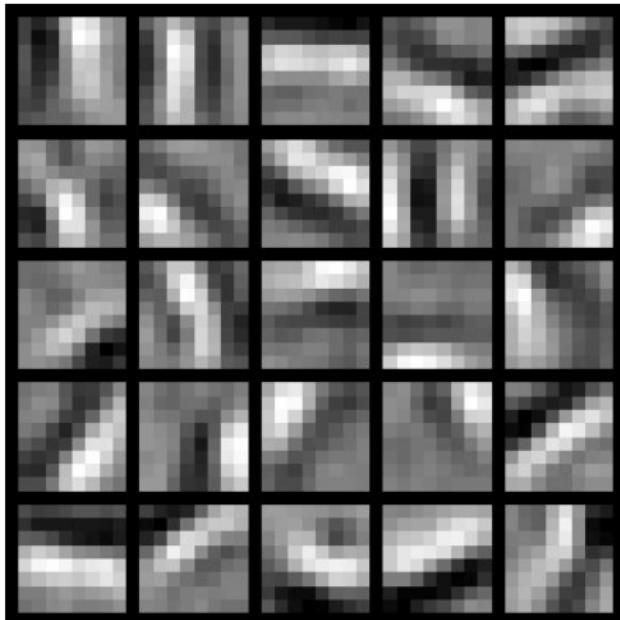
Convolutional Network

- LeCun (1989), convolutional image classification

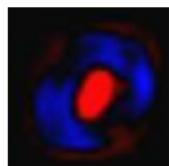


- Layers use **convolutional** weight sharing (as above).
- Each $k \times k$ filter uses shared weights.
- Many different filters learned per layer.

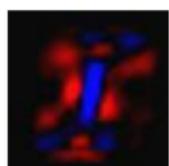
First Layer Filters



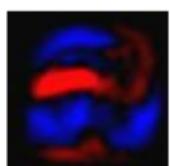
Final Transformations



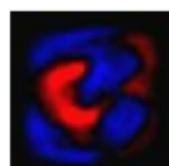
0



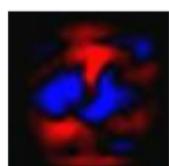
1



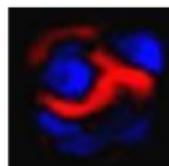
2



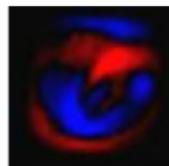
3



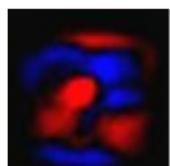
4



5



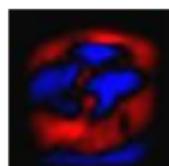
6



7



8



9

- ImageNet (2009) introduced a large labeled dataset of real images

Dog, domestic dog, *Canis familiaris*

A member of the genus *Canis* (probably descended from the common wolf) that has been domesticated by man since prehistoric times; occurs in many breeds; "the dog barked all night"

1603 pictures
88.15% Popularity Percentile
 Wordnet IDs

tom, tomcat (1)
Siamese cat, Siamese (1)
Manx, Manx cat (0)
Maltese, Maltese cat (0)
tabby, queen (0)
Burmese cat (0)
alley cat (0)
Abyssinian, Abyssinian cat (0)
tabby, tabby cat (0)
tortoiseshell, tortoiseshell cat (0)
mouser (0)

dog, domestic dog, *Canis familiaris* (1)
pooch, doggie, doggy, dog (1)
hunting dog (101)
dalmatian, coach dog, carriage dog (1)
cur, mongrel, mutt (2)
corgi, Welsh corgi (2)
Mexican hairless (0)
lapdog (0)
Newfoundland, Newfoundland (0)
poodle, poodle dog (4)
basenji (0)
Leonberg (0)
griffon, Brussels griffon, Belgian griffon (0)
pug, pug-dog (0)
working dog (45)
spitz (4)
Great Pyrenees (0)
toy dog, toy (11)
puppy (0)
head (0)

[Treemap Visualization](#) [Images of the Synset](#) [Downloads](#)

 ImageNet 2011 Fall Release > Domestic animal, domesticated animal > Dog, domestic dog, *Canis familiaris*

Hunting	Working	Toy	Great	Cur	Pug
					<img alt="Thumbnail grid for

Modern Convolutional Networks

- AlexNet: demonstrated that convolutional nets were remarkably effective on this task.

What was new?

- Many details to figure out.
- Very large models, with many layers (deep).
- Much faster compute power (GPUs)

Learned Representations

