Machine Learning (CS 181): 9. Neural Networks 2

David Parkes and Sasha Rush

Contents

Contents

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

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

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

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

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

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

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

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

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

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

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

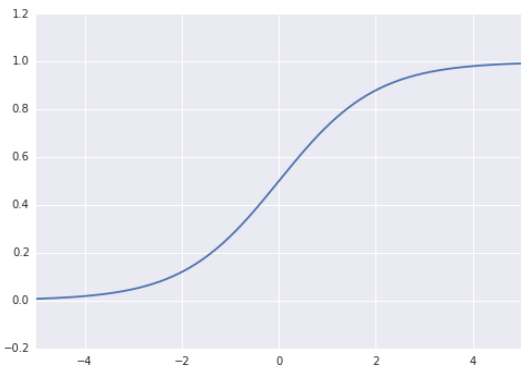
$$p(y = 0|\mathbf{x}; \mathbf{w}) = \frac{1}{1 + \exp(h(\mathbf{x}; \mathbf{w}))}$$

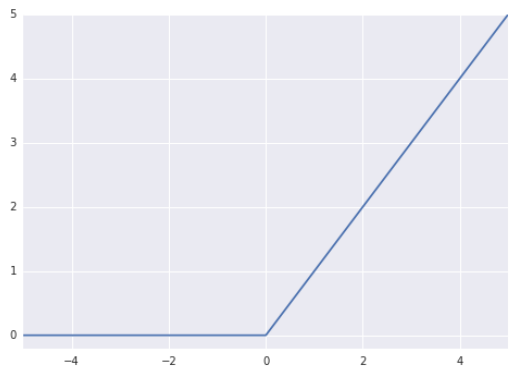
Finally, call (1) the logistic sigmoid activation:

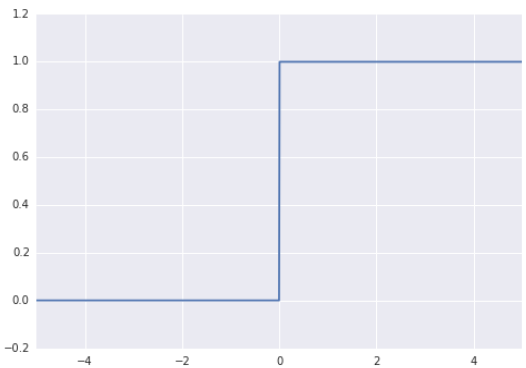
$$\sigma(h) = (1 + \exp -h)^{-1}$$

Activation Functions

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



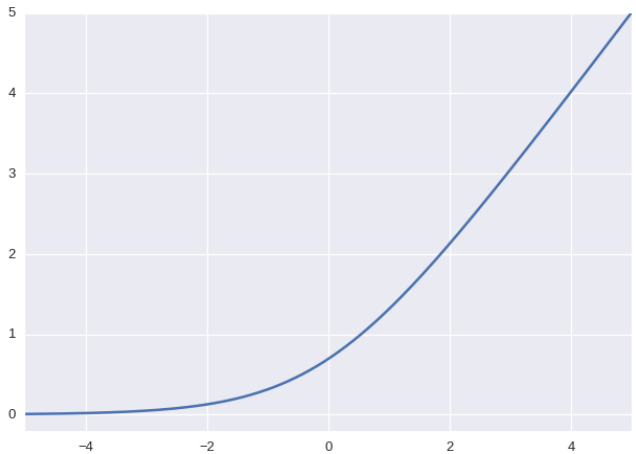


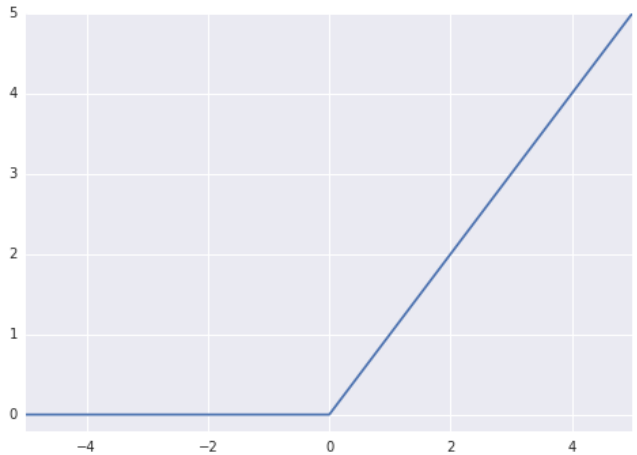


Sigmoid versus ReLU versus 0/1. x-axis input e.g $h(\mathbf{x}; \mathbf{w})$, y-axis output.

Algorithms come from Loss Function

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





Logistic regression loss $\mathcal{L}_\sigma$ 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,

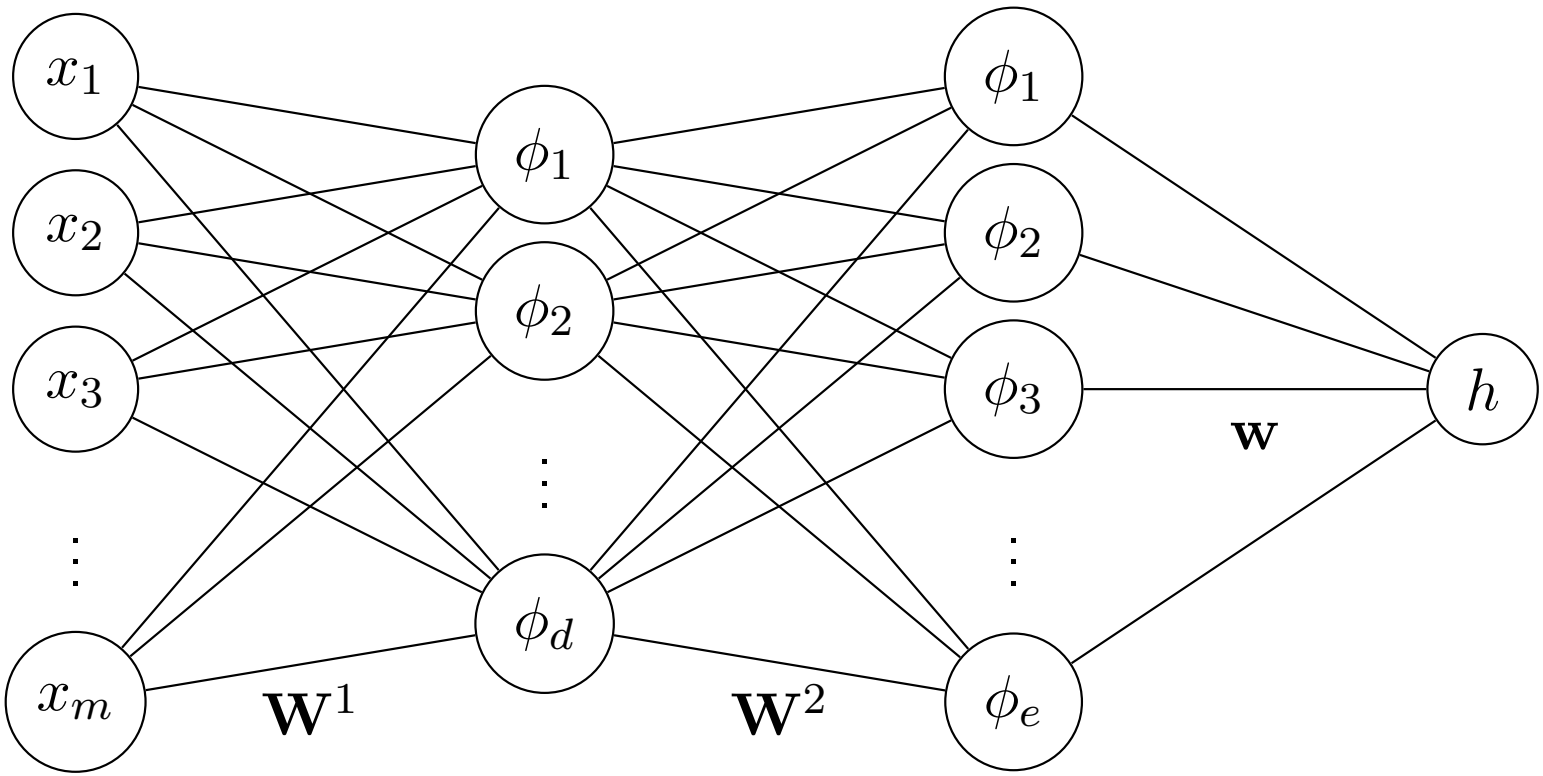
$$\boldsymbol{\phi}(\mathbf{x}; \mathbf{W}^1, \mathbf{w}_0^1) = \boldsymbol{\sigma}(\mathbf{W}^1\mathbf{x} + \mathbf{w}_0^1)$$

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

Full classification problem:

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

Classical Neural Network Diagram



Contents

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 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.

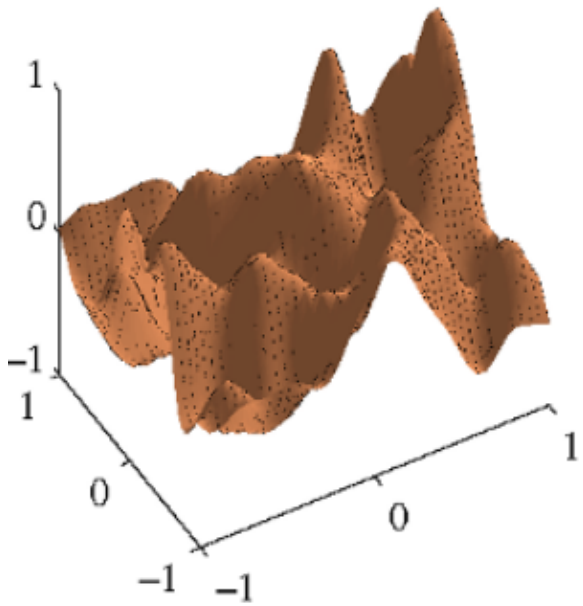
$$\mathcal{L}(\mathbf{w}, \mathbf{W}^1) = \frac{1}{2}\sum_{i=1}^{n}(y_i - h(\mathbf{x}_i; \mathbf{w}, \mathbf{W}^1))^2$$

Training With Gradient Descent

- Need gradient for $\mathbf{w}$ (as before)
- Also need gradients for $\mathbf{W}^1$ (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

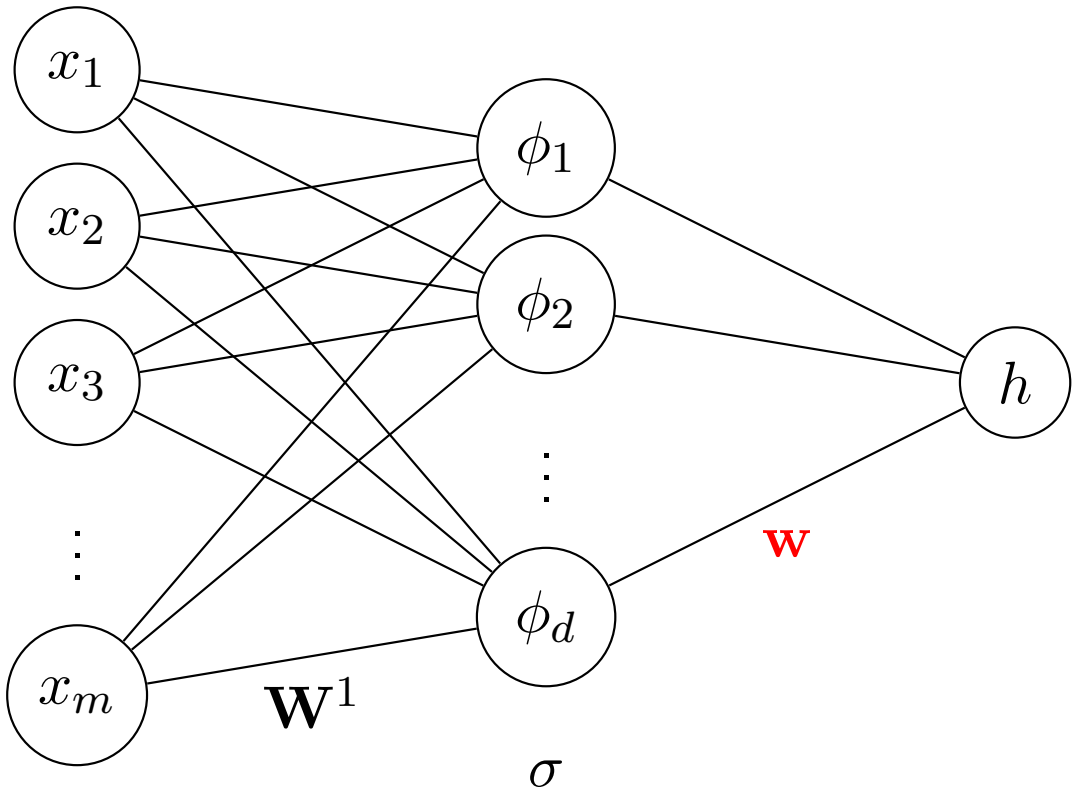
$$\mathbf{w} \in \mathbb{R}^m \rightarrow \mathbf{g}(\mathbf{w}) \in \mathbb{R}^n \rightarrow f(\mathbf{g}(\mathbf{w})) \in \mathbb{R}$$

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

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 $\mathbf{w}$

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

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

$$\begin{array}{ccc} \boldsymbol{\phi}_i & \Leftarrow & -(y_i - h_i) \\ \frac{\partial h_i}{\partial \mathbf{w}} & & \frac{\partial \mathcal{L}}{\partial h_i} \end{array}$$

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

Gradient for $\mathbf{w}$

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

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

$$\begin{array}{ccc} \boldsymbol{\phi}_i & \Leftarrow & -(y_i - h_i) \\ \frac{\partial h_i}{\partial \mathbf{w}} & & \frac{\partial \mathcal{L}}{\partial h_i} \end{array}$$

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

Gradient for $\mathbf{w}$

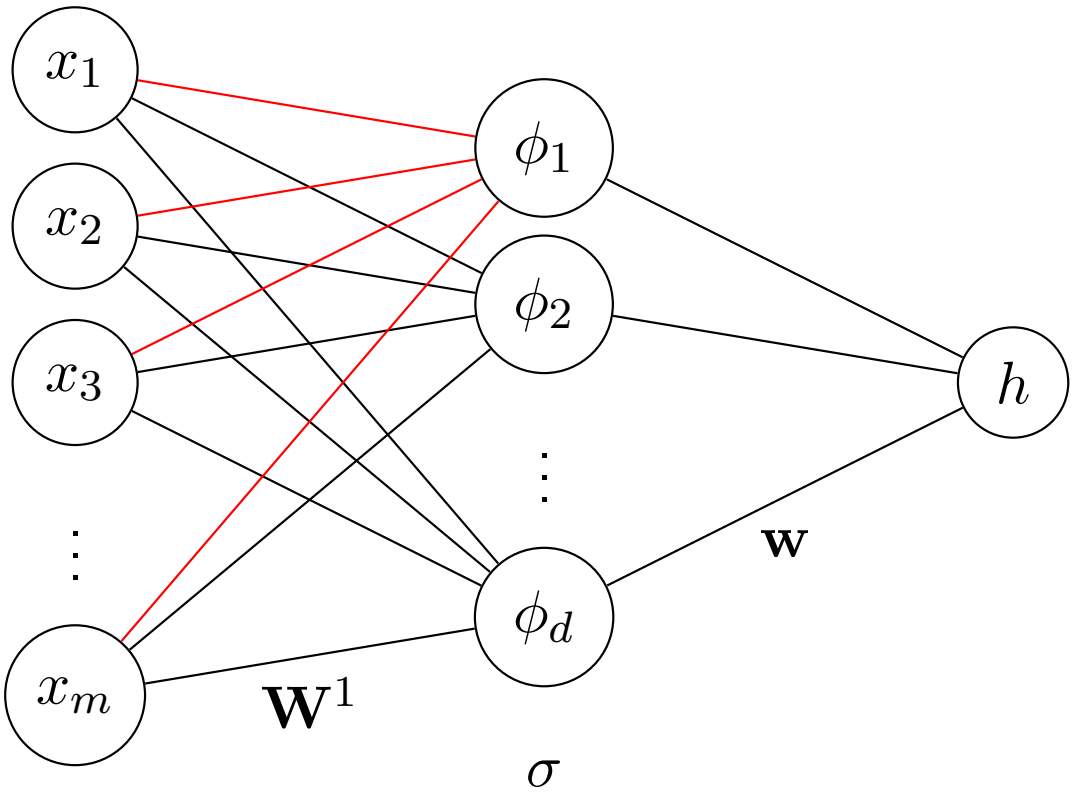
$$\mathcal{L} = \frac{1}{2}\sum_{i=1}^{n}(y_i - (\mathbf{w}^\top \boldsymbol{\phi}_i + w_0))^2$$

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

$$\begin{array}{ccc} \boldsymbol{\phi}_i & \Leftarrow & -(y_i - h_i) \\ \frac{\partial h_i}{\partial \mathbf{w}} & & \frac{\partial \mathcal{L}}{\partial h_i} \end{array}$$

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

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)

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

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

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

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)

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

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

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

Contents

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

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

  Save intermediary values for all layers l.

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

- Backward Step (b-prop):

  $$\dot{\boldsymbol{\phi}}^l = \frac{\partial \mathcal{L}}{\partial \boldsymbol{\phi}^l(\dots\boldsymbol{\phi}_1(\mathbf{x}))} = \sum_{j=1}^{d} \frac{\partial \phi_j^{(l+1)}}{\partial \phi_j^l} \frac{\partial \mathcal{L}}{\partial \phi_j^{(l+1)}}$$

- Compute parameter gradients

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

Backpropagation (Deep Network)

- Forward Step (f-prop):

  Compute

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

  Save intermediary values for all layers l.

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

- Backward Step (b-prop):

  $$\dot{\boldsymbol{\phi}}^l = \frac{\partial \mathcal{L}}{\partial \boldsymbol{\phi}^l(\dots\boldsymbol{\phi}_1(\mathbf{x}))} = \sum_{j=1}^{d} \frac{\partial \phi_j^{(l+1)}}{\partial \phi_j^l} \frac{\partial \mathcal{L}}{\partial \phi_j^{(l+1)}}$$

- Compute parameter gradients

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

Backpropagation (Deep Network)

- Forward Step (f-prop):

  Compute

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

  Save intermediary values for all layers l.

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

- Backward Step (b-prop):

  $$\dot{\boldsymbol{\phi}}^l = \frac{\partial\mathcal{L}}{\partial\boldsymbol{\phi}^l(\dots\boldsymbol{\phi}_1(\mathbf{x}))} = \sum_{j=1}^{d}\frac{\partial\phi_j^{(l+1)}}{\partial\phi_j^l}\frac{\partial\mathcal{L}}{\partial\phi_j^{(l+1)}}$$

- Compute parameter gradients

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

Backpropagation (Deep Network)

- Forward Step (f-prop):

  Compute

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

  Save intermediary values for all layers l.

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

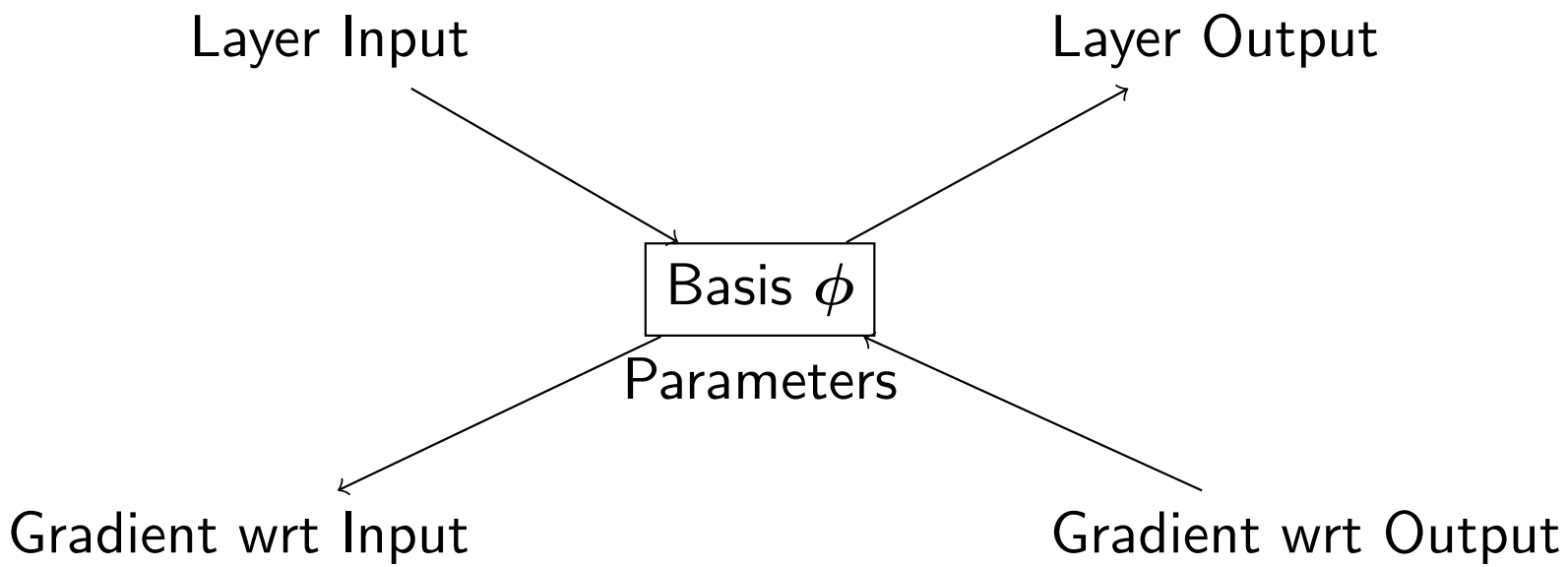
- Backward Step (b-prop):

$$\dot{\boldsymbol{\phi}}^l = \frac{\partial \mathcal{L}}{\partial \boldsymbol{\phi}^l(\dots\boldsymbol{\phi}_1(\mathbf{x}))} = \sum_{j=1}^{d} \frac{\partial \phi_j^{(l+1)}}{\partial \phi_j^l} \frac{\partial \mathcal{L}}{\partial \phi_j^{(l+1)}}$$

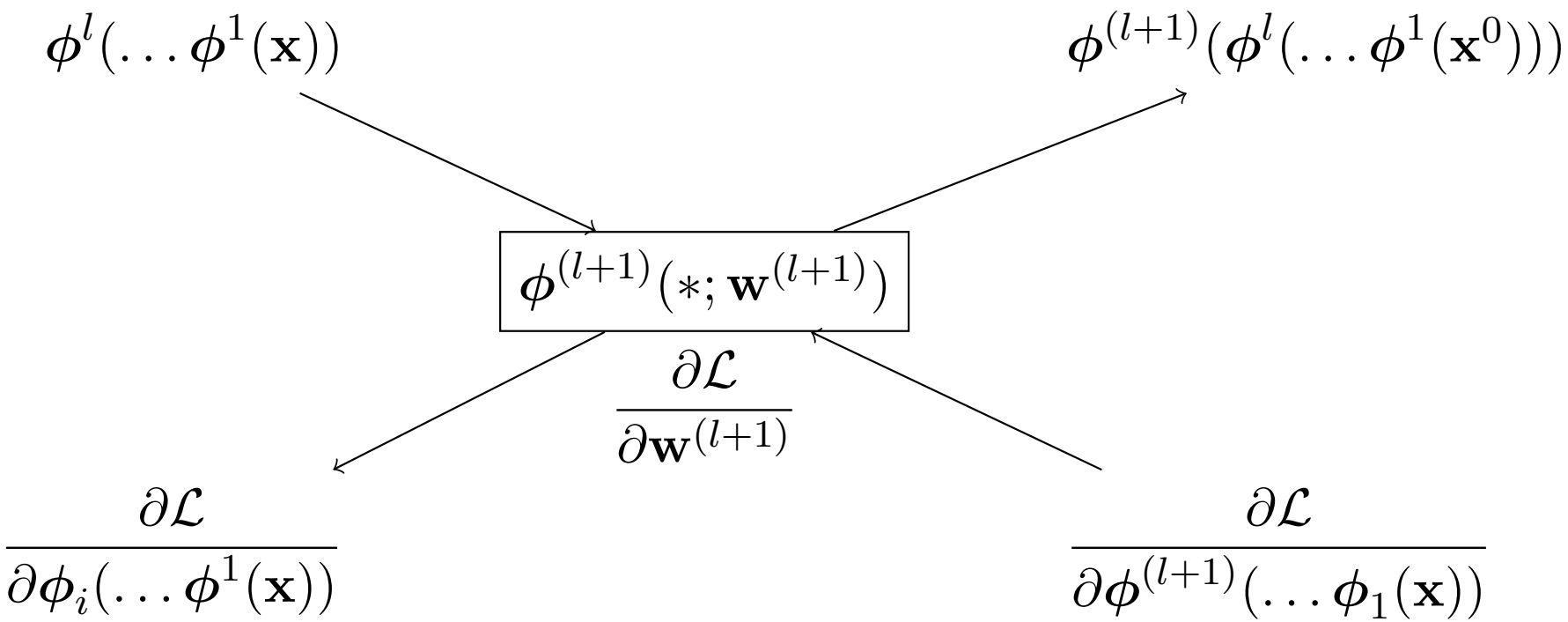
- Compute parameter gradients

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

Backpropagation: High Level



Backpropagation: Data flow



Computation Graph

Unit-based specification of NN

- Every function is a represented as a unit.

- Responsibilities:
  1. Compute $\boldsymbol{\phi}^{(l+1)}(\mathbf{x}; \mathbf{W}^{(l+1)})$ (forward)
  2. Compute chain-rule given $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\phi}^{(l+1)}(...\boldsymbol{\phi}^{1}(\mathbf{x}))}$ and $\boldsymbol{\phi}^{l}$ (backward)
  3. Compute parameter gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(l+1)}}$ (update)

Example

Contents

Neural Network Basis

So far, adaptive basis functions are mini logistic regressions:

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

But backpropagation can work with arbitrary differentiable functions.

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

Warning: Not much theory here.

Neural Network Basis

So far, adaptive basis functions are mini logistic regressions:

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

But backpropagation can work with arbitrary differentiable functions.

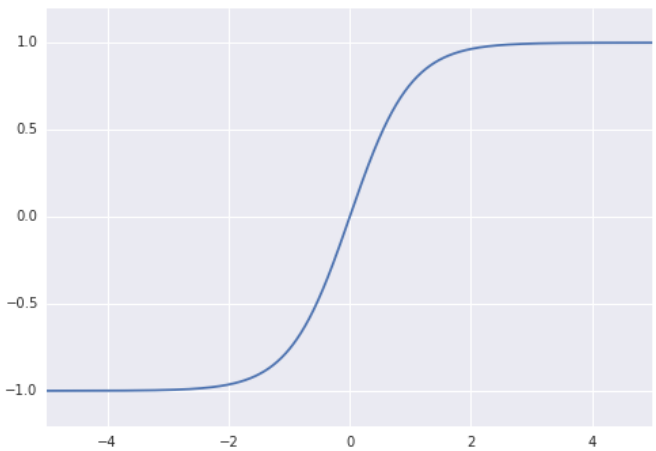
- What about other activation functions?
- What about other structures?

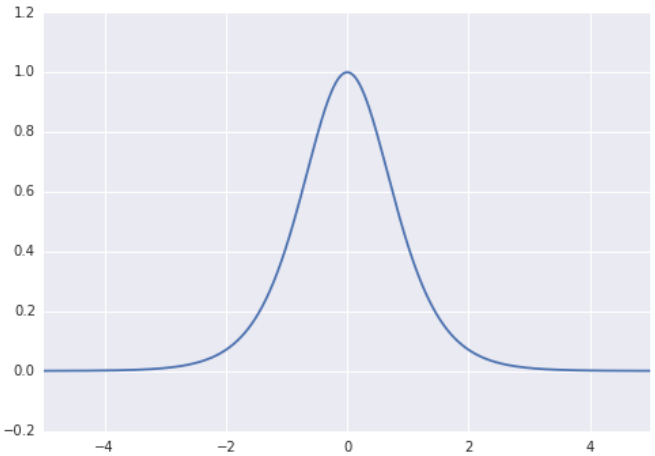
Warning: Not much theory here.

Activations: Tanh

Hyperbolic Tangeant:

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



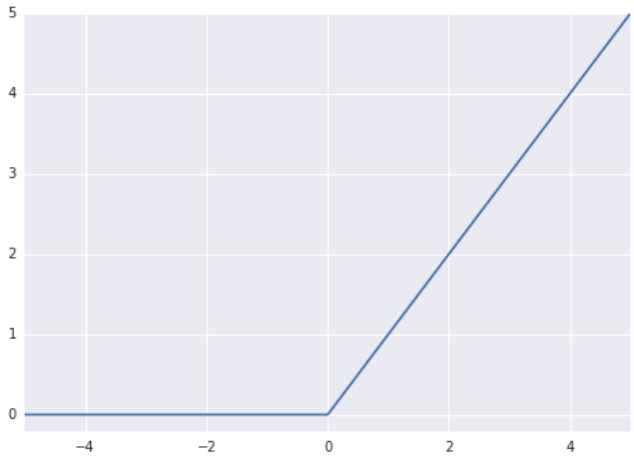


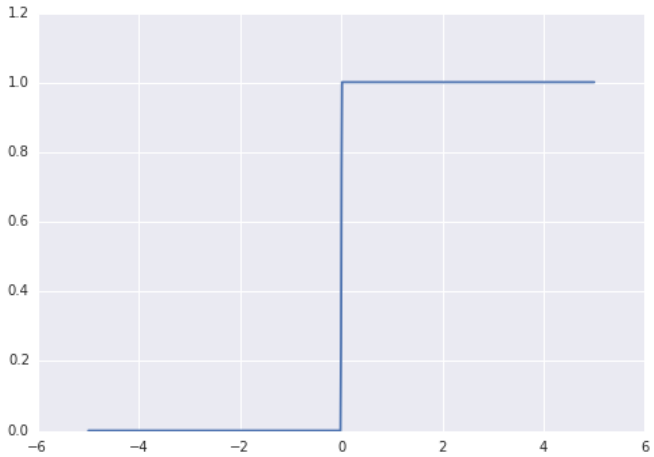
- Intuition: Similar to logistic sigmoid, but range between -1 and 1.
- More widely used when no probabilistic interpretation.

Activations: ReLU

Rectified Linear Unit:

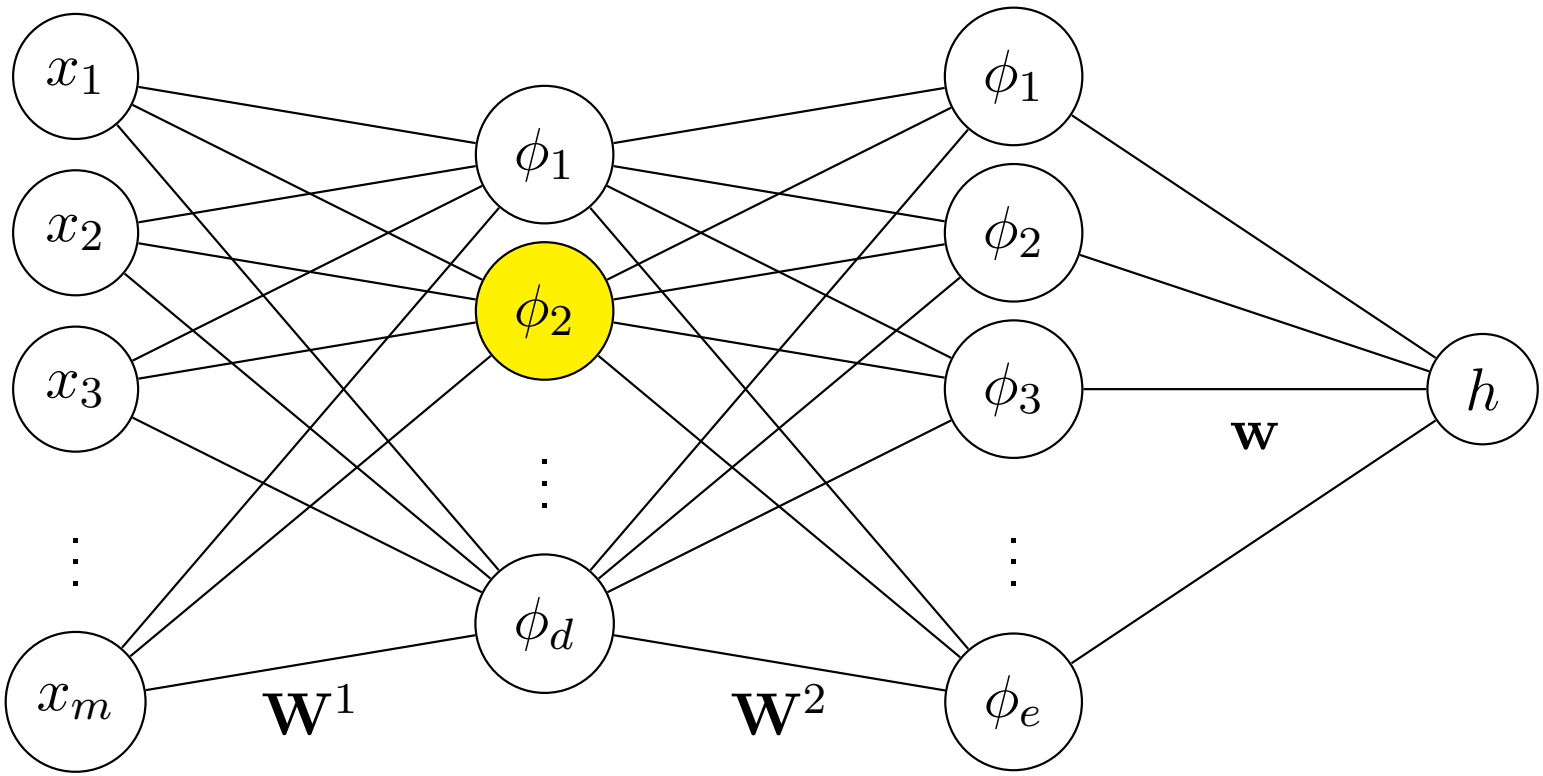
$$\text{ReLU}(h) = \max\{0, h\}$$





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

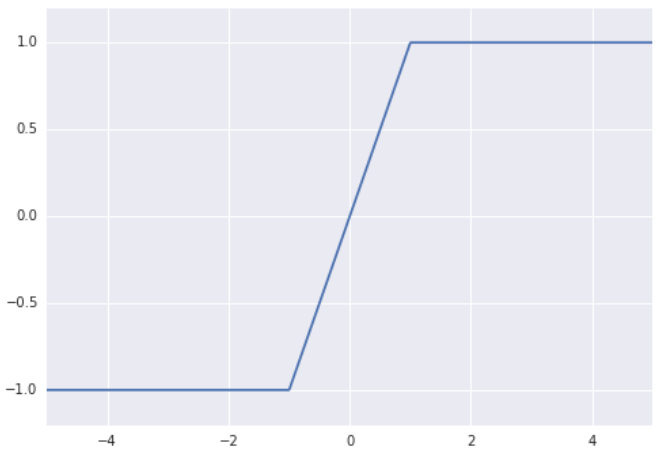
Issues: Saturation

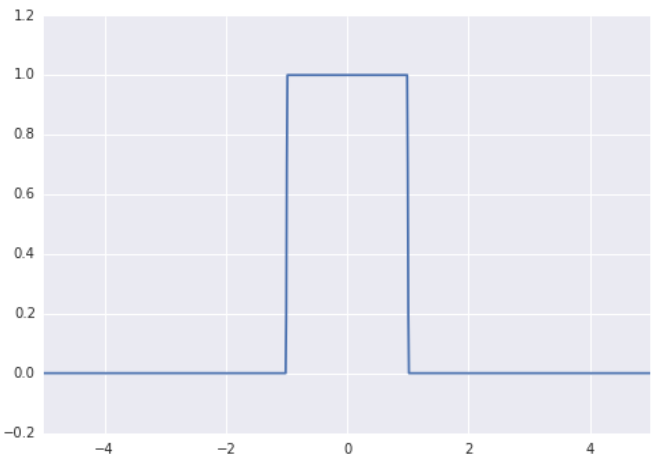


Activations: Hard Tanh

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

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

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

On-Diagonal Weight Sharing (2)

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

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

- 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

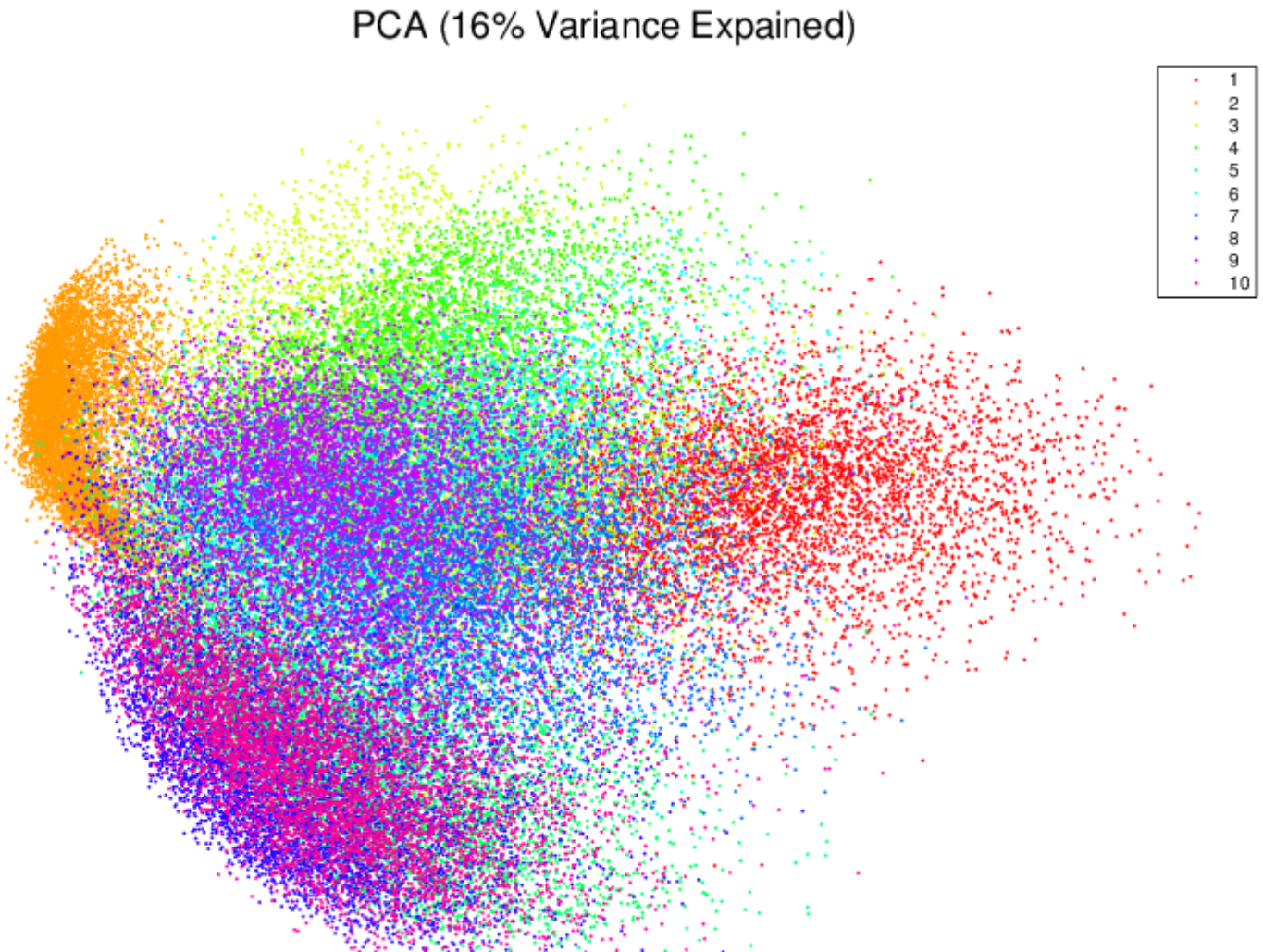
Contents

Example: Digit Classification

- Data: Handwritten US zip codes

Digit Recognition

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



Multiclass Logistic Regression

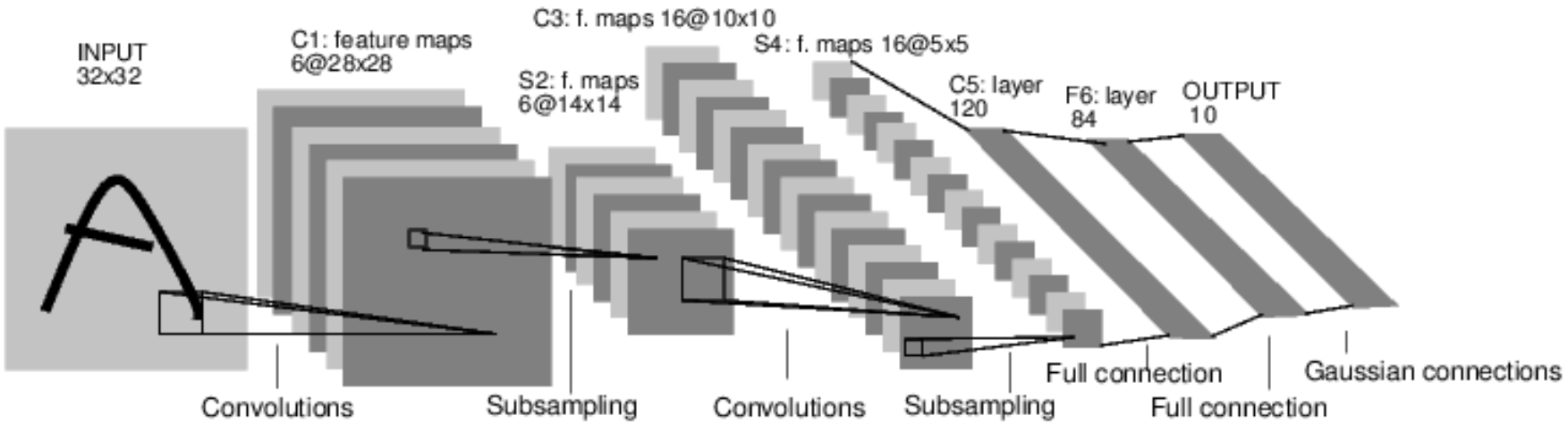
- Multiclass logistic regression uses a $\mathbf{w}_\ell$ for each class.
- Generalization of sigmoid is softmax function.

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

(Derivation and exploration on homework.)

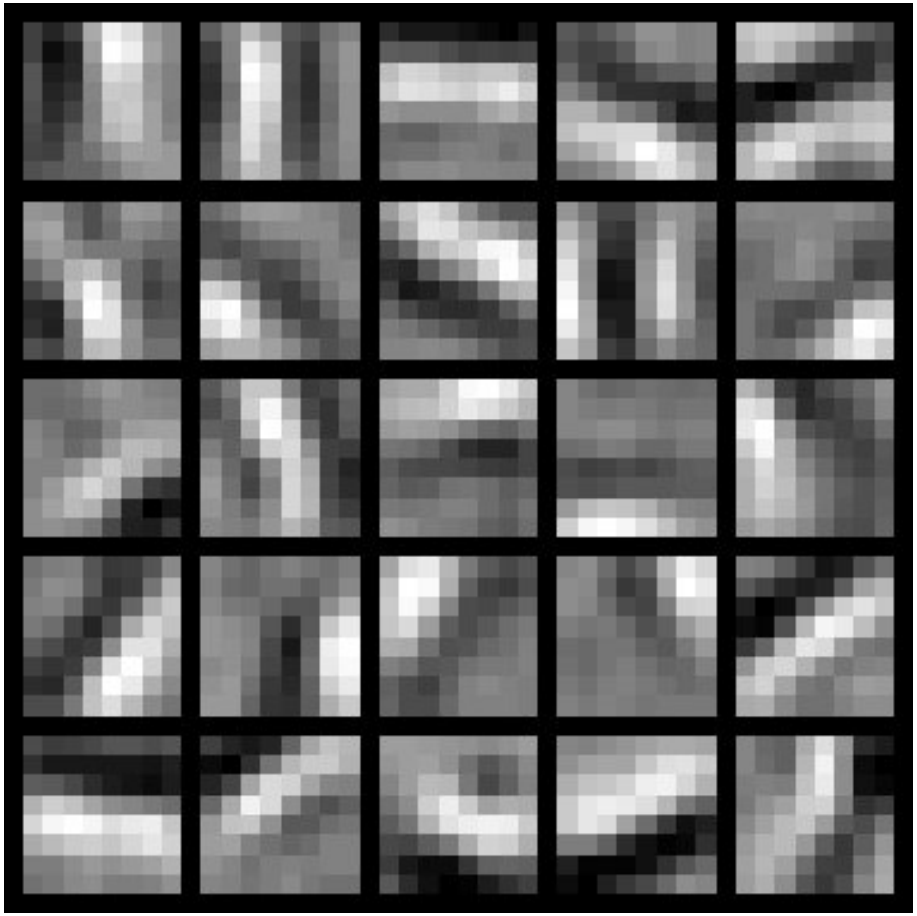
Convolutional Network

- LeCun (1989), convolutional image classification

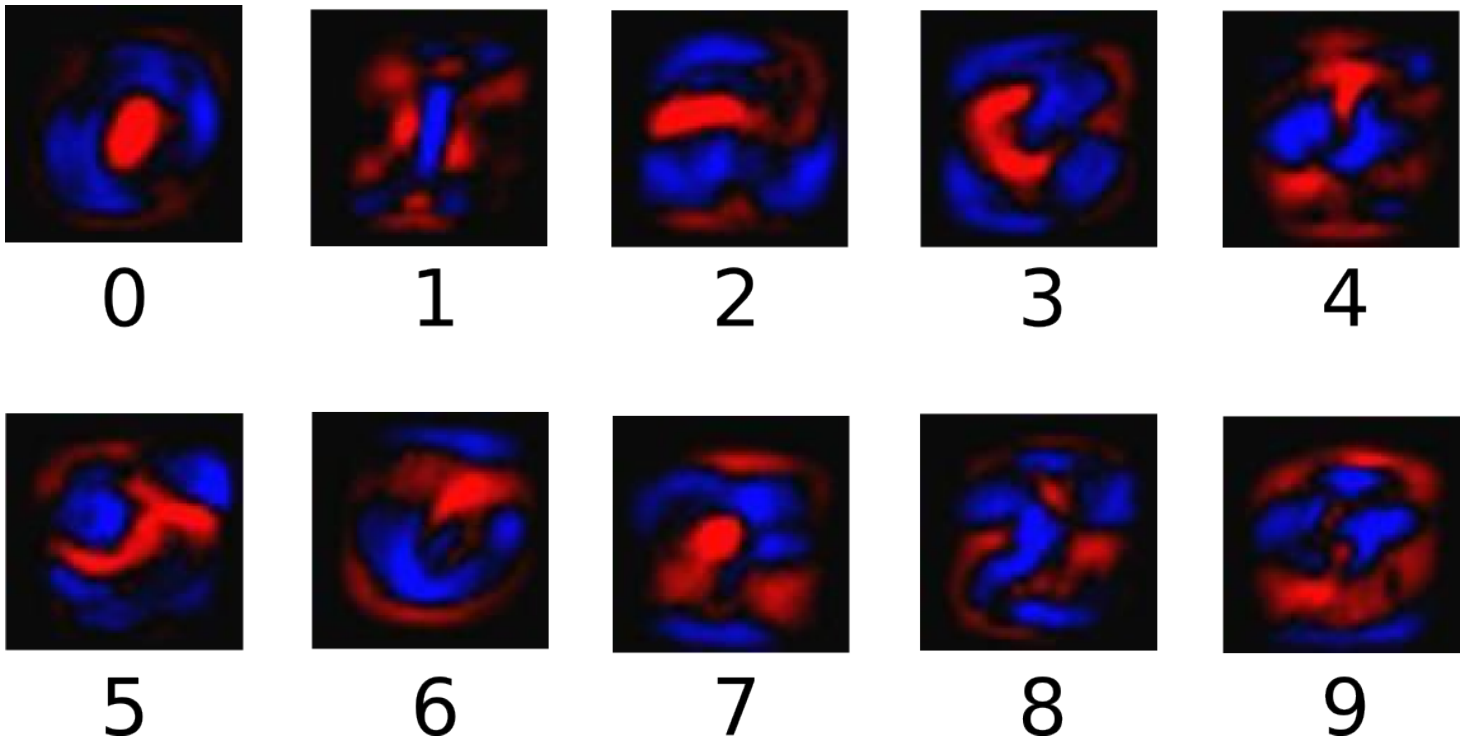


- Layers use convolutional weight sharing (as above).
- Each $k \times k$ filter uses shared weights.
- Many different filters learned per layer.

First Layer Filters

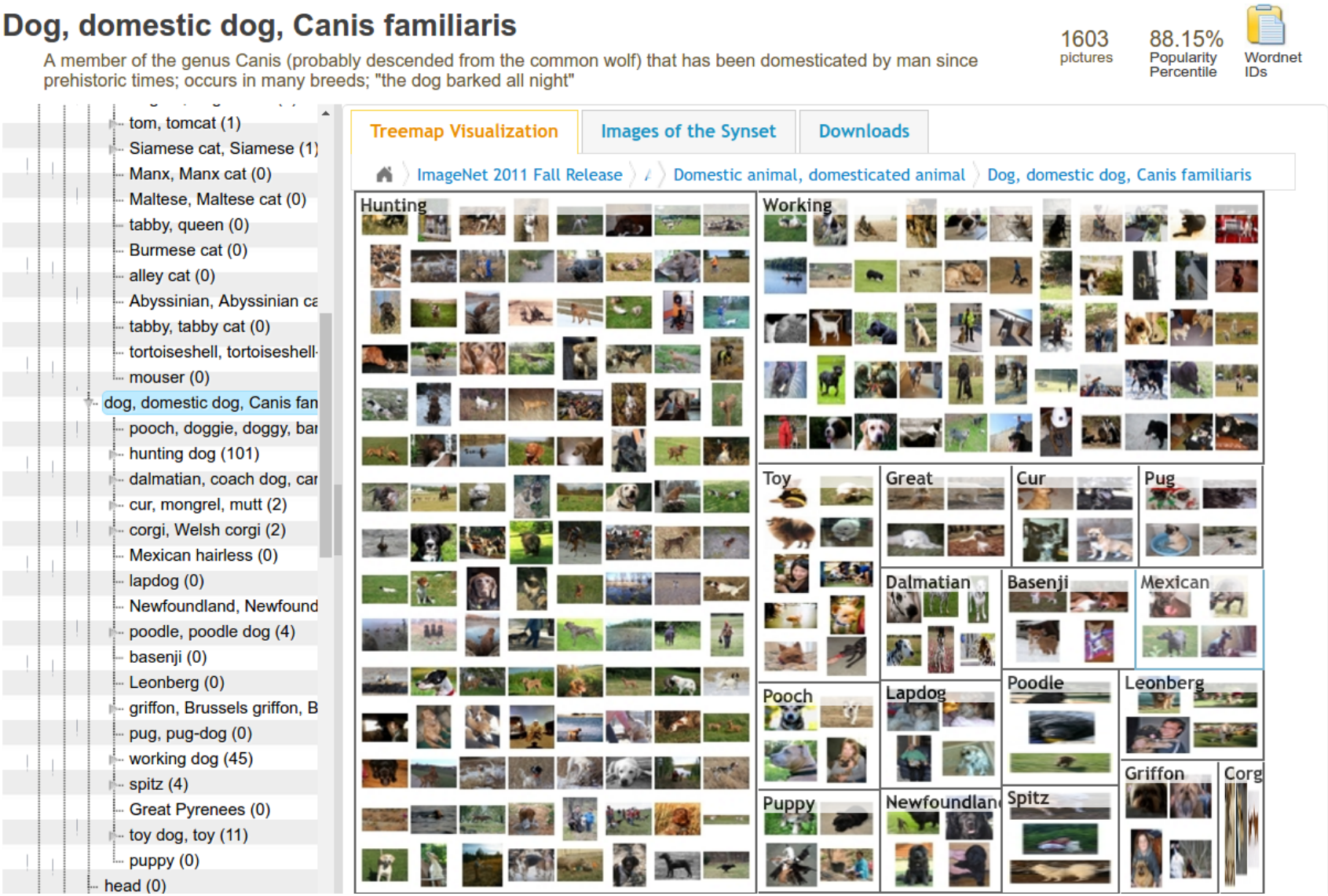


Final Transformations



ImageNet

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



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