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

David Parkes and Sasha Rush

Contents

1 Fitting Neural Networks

2 Backpropagation

3 Neural Network Architectures

4 Case Study: Image Recognition

Neural Networks: Matrix-view

Define entire adaptive basis layer as d problems,

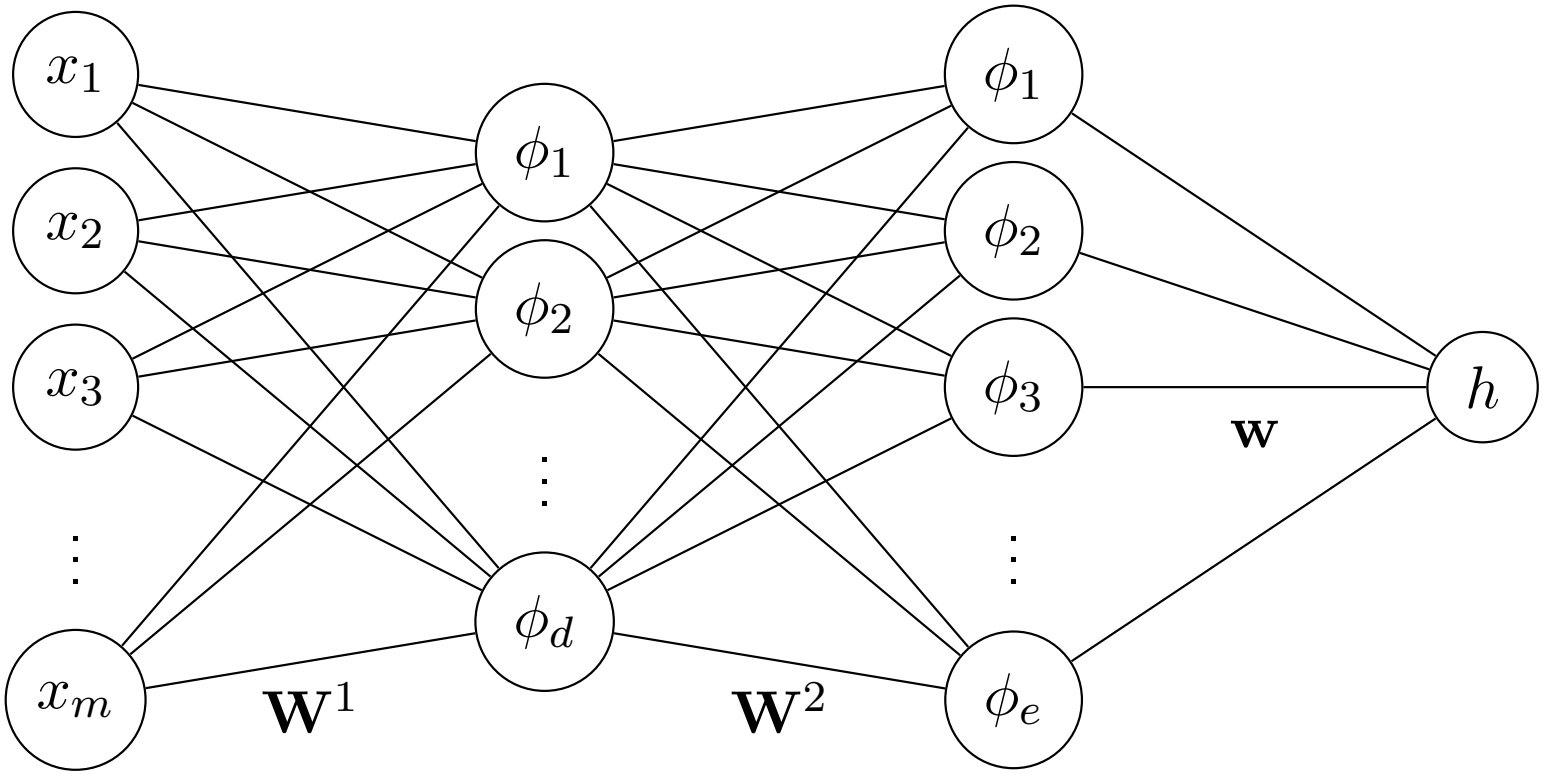
$$\boldsymbol{\phi}(\mathbf{x}; \mathbf{W}^1, \mathbf{w}_0^1) = \boldsymbol{\sigma}(\mathbf{W}^1\mathbf{x} + \mathbf{w}_0^1)$$

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

1 Fitting Neural Networks

2 Backpropagation

3 Neural Network Architectures

4 Case Study: Image Recognition

Loss Function

- Neural networks are an adaptive *basis*, don't specify the loss function.
- Could train using conditional likelihood...

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

- Can also train for standard regression...

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

- Only requires useful derivatives.

Loss Function

- Neural networks are an adaptive *basis*, don't specify the loss function.
- Could train using conditional likelihood...

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

- Can also train for standard regression...

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

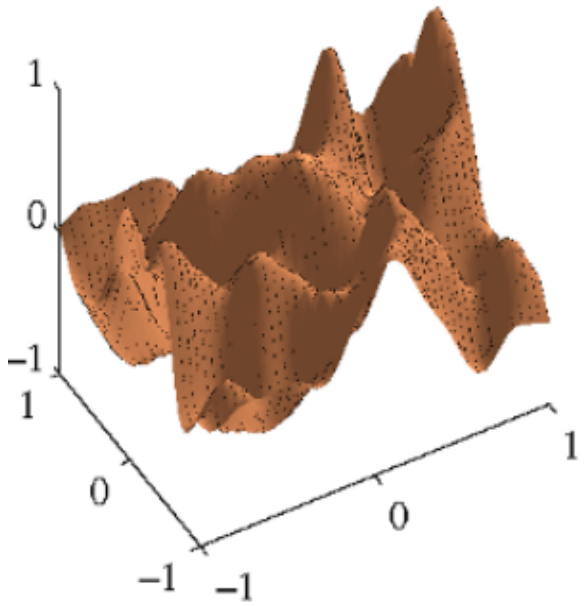
- Only requires useful derivatives.

Training With SGD

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

Non-Convexity

- Issue: neural network loss is non-convex!



Non-convex loss surface.

- SGD will not find global optimizer, only local minimum.
- Can be a real issue (many practitioners ignore).

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:

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

As before: need gradient to train $\mathbf{w}, \mathbf{W}^1$.

$$h(\mathbf{x}; \mathbf{w}, \mathbf{W}^1) = h_i = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}_i; \mathbf{W}^1) + w_0$$

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:

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

As before: need gradient to train $\mathbf{w}, \mathbf{W}^1$.

$$h(\mathbf{x}; \mathbf{w}, \mathbf{W}^1) = h_i = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}_i; \mathbf{W}^1) + w_0$$

Interlude: Vector-Valued Chain Rule

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

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

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

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

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

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

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

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

Chain rule, vector-valued functions:

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

Applying here.

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

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

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

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

Applying here.

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

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

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

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

Contents

1 Fitting Neural Networks

2 Backpropagation

3 Neural Network Architectures

4 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.

Backpropagation (Deep Network)

- Forward Step (f-prop):

  Compute

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

  Save intermediary values for all layers l.

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

- Backward Step (b-prop):

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

- 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

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

  Save intermediary values for all layers l.

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

- Backward Step (b-prop):

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

- 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

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

  Save intermediary values for all layers l.

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

- Backward Step (b-prop):

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

- 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

- Forward Step (f-prop):

  Compute

$$L(f_k(\dots f_1(\mathbf{x}^0)))$$

  Saving intermediary values

$$f_i(\dots f_1(\mathbf{x}^0)))$$

- Backward Step (b-prop):

$$\frac{\partial L}{\partial f_i(\dots f_1(\mathbf{x}^0))} = \sum_{j=1}^{n_i} \frac{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}{\partial f_i(\dots f_1(\mathbf{x}^0))} \frac{\partial L}{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_i} = \sum_{j=1}^{n_i} \frac{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}{\partial \boldsymbol{\theta}_i} \frac{\partial L}{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}$$

Backpropagation

- Forward Step (f-prop):

  Compute

$$L(f_k(\dots f_1(\mathbf{x}^0)))$$

  Saving intermediary values

$$f_i(\dots f_1(\mathbf{x}^0)))$$

- Backward Step (b-prop):

$$\frac{\partial L}{\partial f_i(\dots f_1(\mathbf{x}^0))} = \sum_{j=1}^{n_i} \frac{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}{\partial f_i(\dots f_1(\mathbf{x}^0))} \frac{\partial L}{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_i} = \sum_{j=1}^{n_i} \frac{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}{\partial \boldsymbol{\theta}_i} \frac{\partial L}{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}$$

Backpropagation

- Forward Step (f-prop):

  Compute

$$L(f_k(\dots f_1(\mathbf{x}^0)))$$

  Saving intermediary values

$$f_i(\dots f_1(\mathbf{x}^0)))$$

- Backward Step (b-prop):

$$\frac{\partial L}{\partial f_i(\dots f_1(\mathbf{x}^0))} = \sum_{j=1}^{n_i} \frac{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}{\partial f_i(\dots f_1(\mathbf{x}^0))} \frac{\partial L}{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_i} = \sum_{j=1}^{n_i} \frac{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}{\partial \boldsymbol{\theta}_i} \frac{\partial L}{\partial f_{i+1}(\dots f_1(\mathbf{x}^0))_j}$$

Backpropagation: Data flow

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

$$\boldsymbol{\phi}^{(l+1)}(\boldsymbol{\phi}^{l}(\ldots\boldsymbol{\phi}^{1}(\mathbf{x}^{0})))$$

$$\boxed{\boldsymbol{\phi}^{(l+1)}(*;\mathbf{w}^{(l+1)})}$$

$$\frac{\partial\mathcal{L}}{\partial\mathbf{w}^{(l+1)}}$$

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{\phi}_{i}(\ldots\boldsymbol{\phi}^{1}(\mathbf{x}))}$$

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{\phi}^{(l+1)}(\ldots\boldsymbol{\phi}_{1}(\mathbf{x}))}$$

Computation Graph

Unit-based specification of NN

- Every function is a represented as a unit.

- Responsibilities:
  1. Expose parameters $\mathbf{W}^{l+1}$
  2. Compute $\phi^{(l+1)}(\mathbf{x}; \mathbf{W}^{l+1})$ (forward)
  3. Compute any necessary state needed for backward.
  4. Compute chain-rule given $\frac{\partial \mathcal{L}}{\partial \phi^{l+1}(...\phi^1(\mathbf{x}))}$ and ϕ^l
  5. Compute parameter gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{l+1}}$

- Contract: forward will always be called before backward.

Contents

1 Fitting Neural Networks

2 Backpropagation

3 Neural Network Architectures

4 Case Study: Image Recognition

Neural Network Basis

So far, adaptive basis functions are mini logistic regressions:

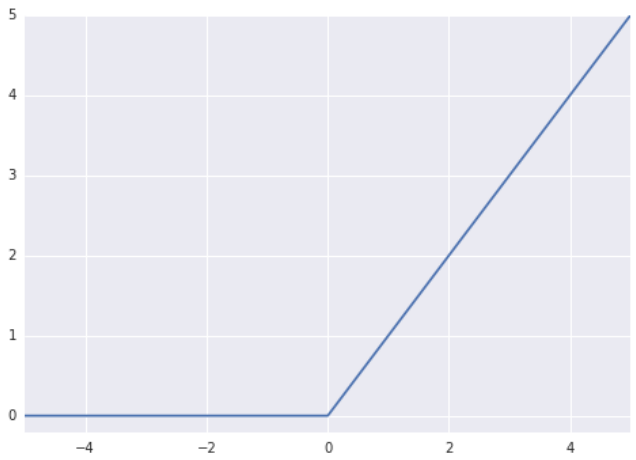
$$\phi(\mathbf{x}; \mathbf{W}^1, \mathbf{w}_0^1) = \boldsymbol{\sigma}(\mathbf{W}^1\mathbf{x} + \mathbf{w}_0^1)$$

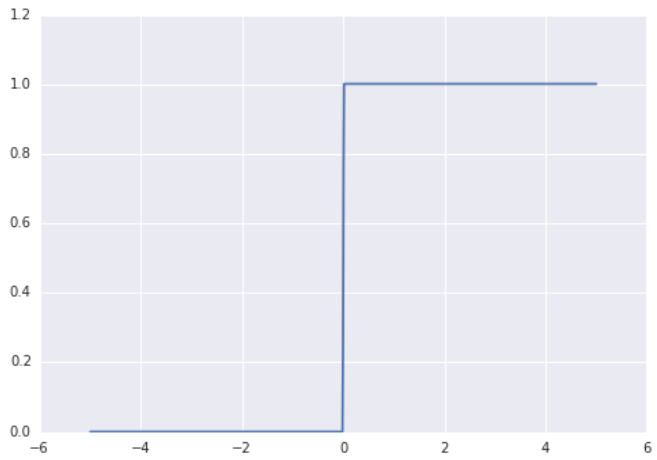
- Backpropagation can work with broad array of differentiable functions.
- What about other activation functions?
- What about other structures?

Non-Linearities: ReLU

Rectified Linear Unit:

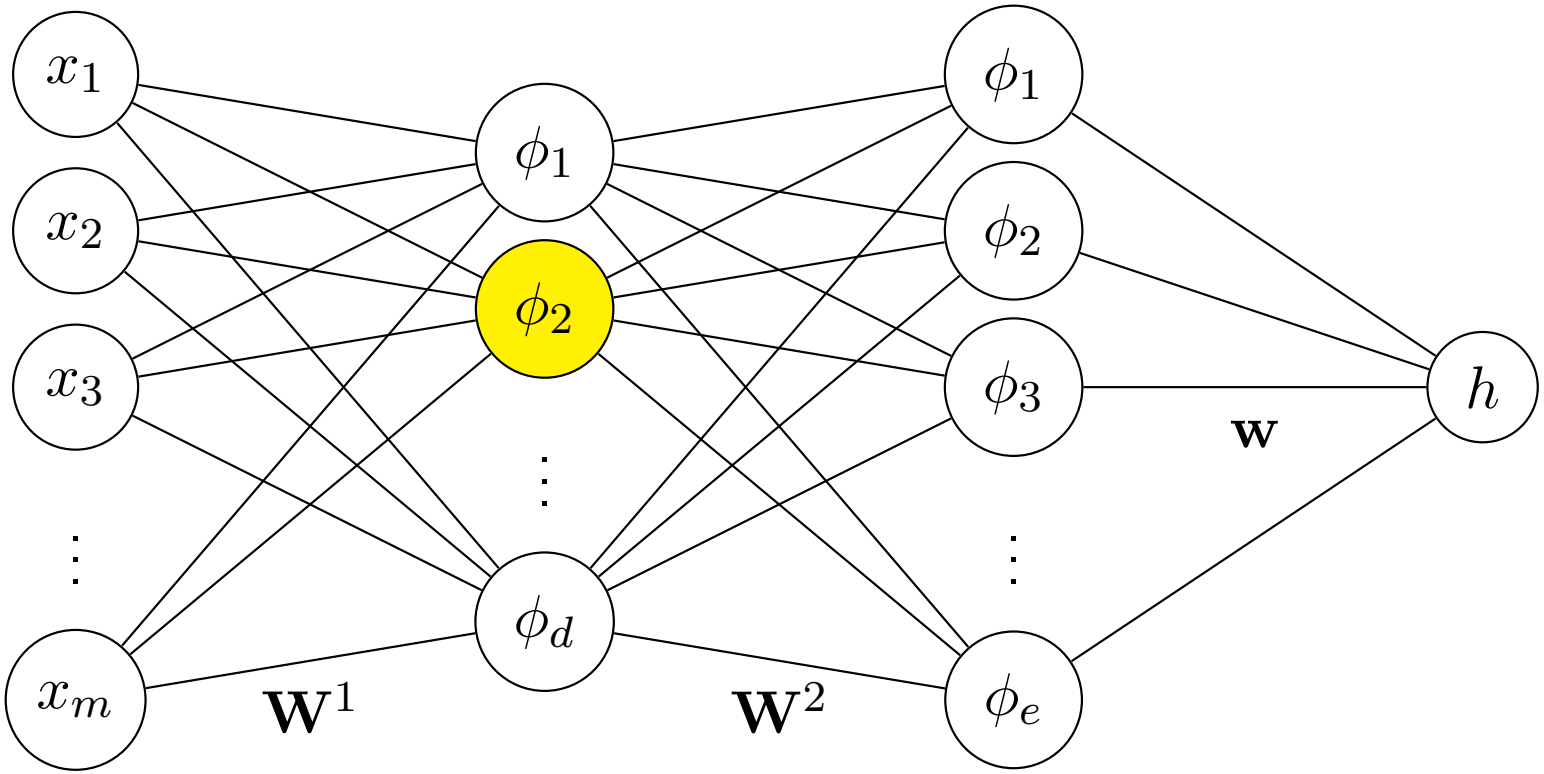
$$\mathrm{ReLU}(h) = \max\{0, h\}$$





- Intuition: Each hidden-unit gives activation margin
- No gradient (saturation) when below 0.

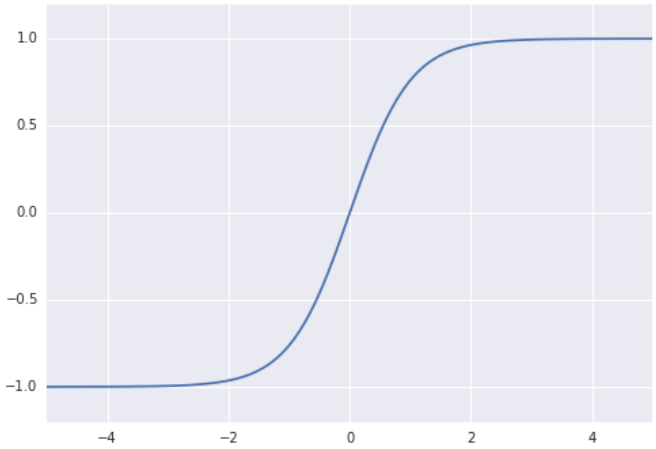
Issues: Saturation

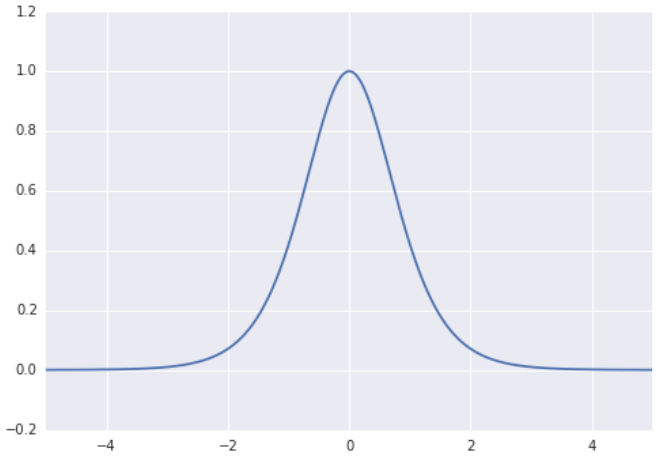


Other Non-Linearities: Tanh

Hyperbolic Tangeant:

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



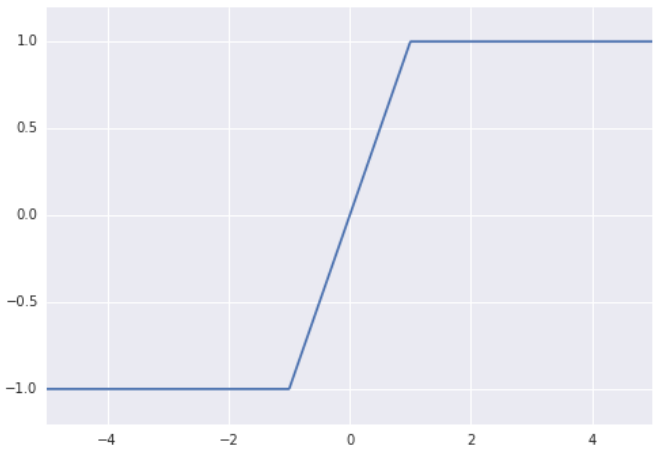


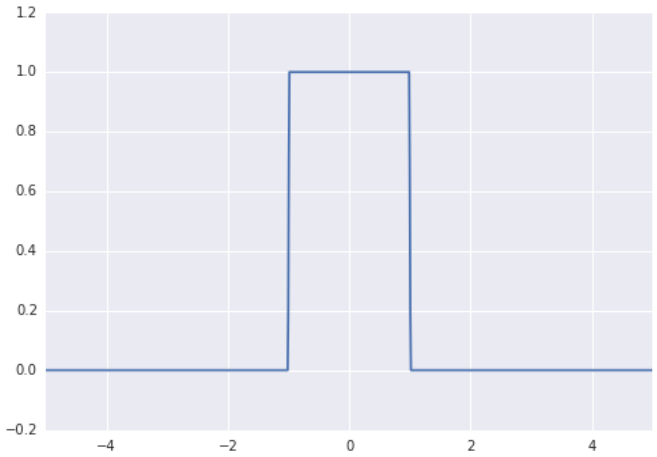
- Intuition: Similar to sigmoid, but range between 1 and -1.

Other Non-Linearities: Hard Tanh

Piecewise linear:

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





- Intuition: properties of both ReLU and tanh.

Shared Weights

A multilayer neural network will have significant 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.

Simple Weight Sharing

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

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

Simple Weight Sharing (2)

- If $\mathbf{x}$ has local structure, each ϕ_j is a "filter".
- For instance, if $\mathbf{x}$ is a time series, w_1, w_2, w_3 can pick up on patterns.

$$\mathbf{x} = [-1; -1; 1; 1; -1; 1; \ldots; 1]$$

$$w_0 = -2, w_1 = 1, w_2 = 1, w_3 = -1$$

$$\boldsymbol{\phi} = [0; 0; 1; 0 \ldots]$$

- This type of sharing is known as a convolutional network

Contents

1 Fitting Neural Networks

2 Backpropagation

3 Neural Network Architectures

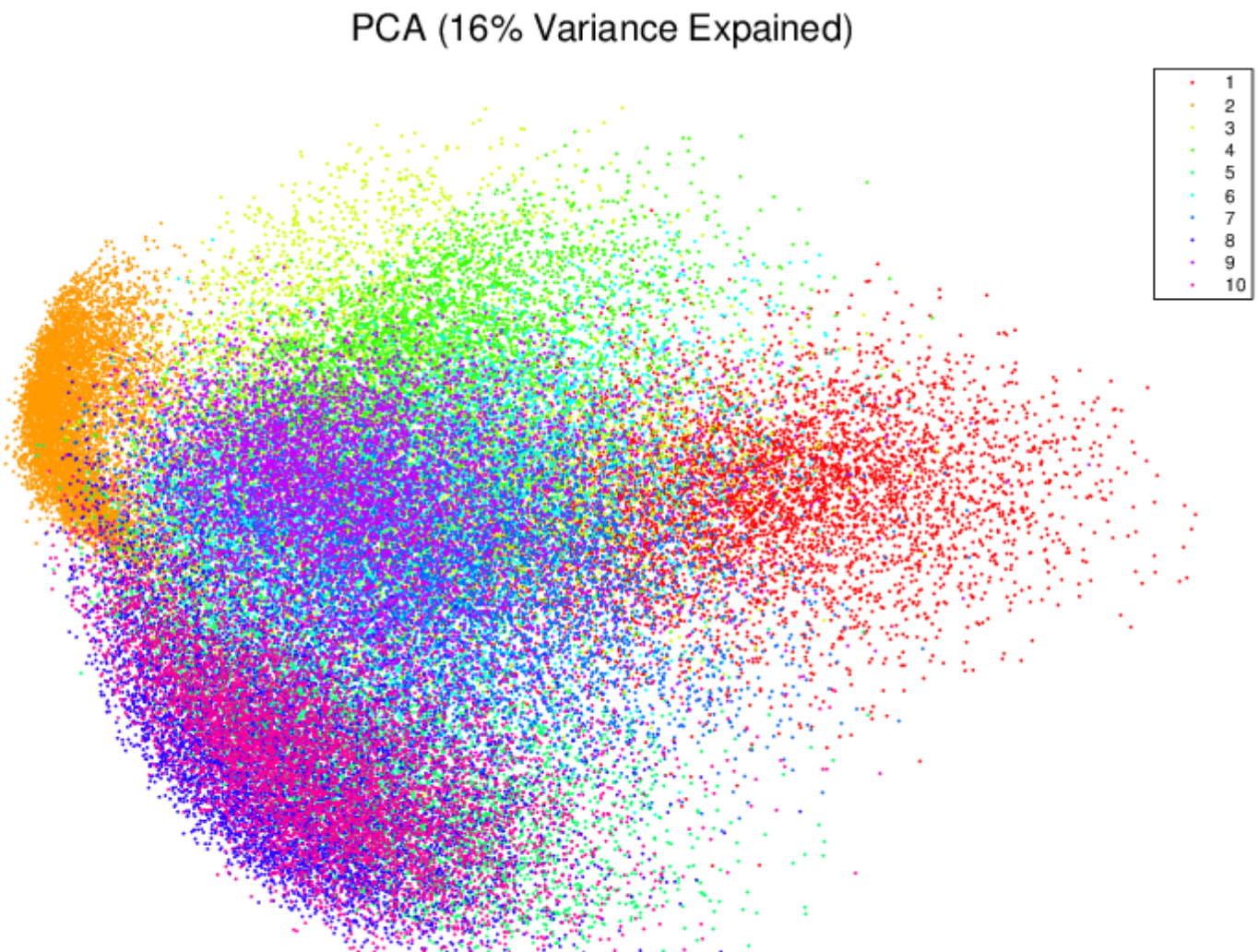
4 Case Study: Image Recognition

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

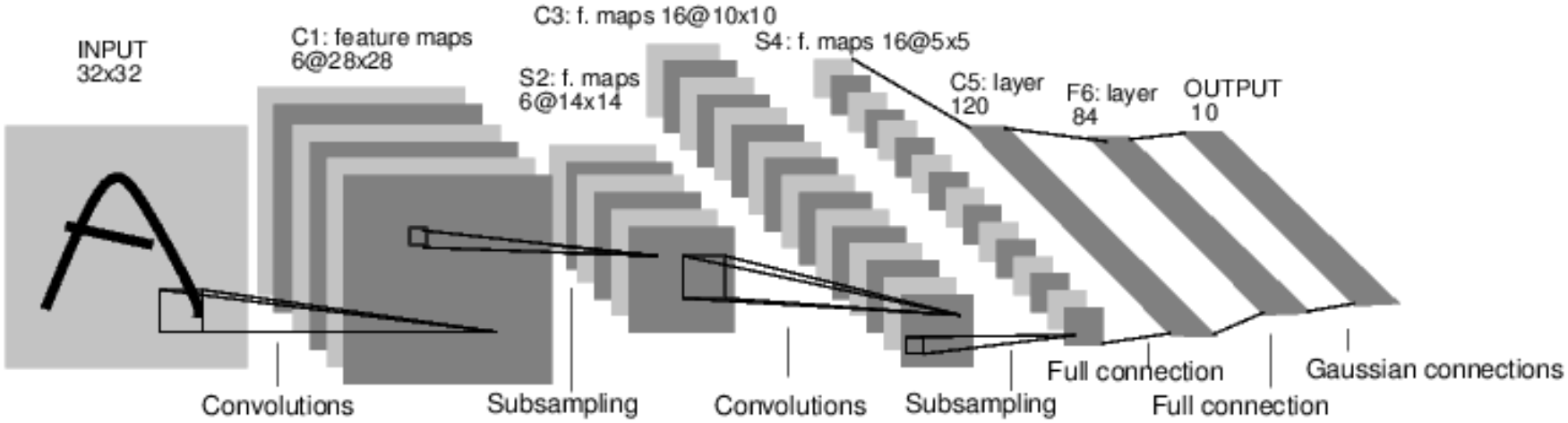
- Multiclass logistic regression uses a $\mathbf{w}_\ell$ for each class.
- Generalization of sigmoid is softmax function.

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

(Derivation and exploration on homework.)

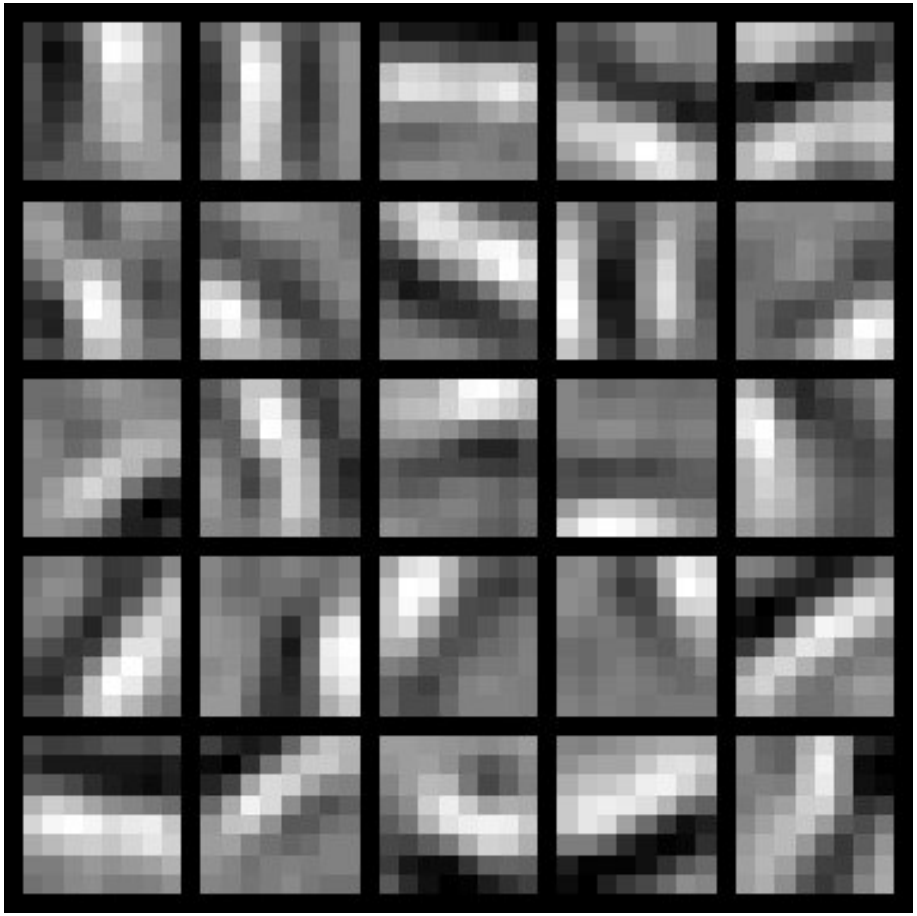
Convolutional Network

- LeCun (1989), convolutional image classification

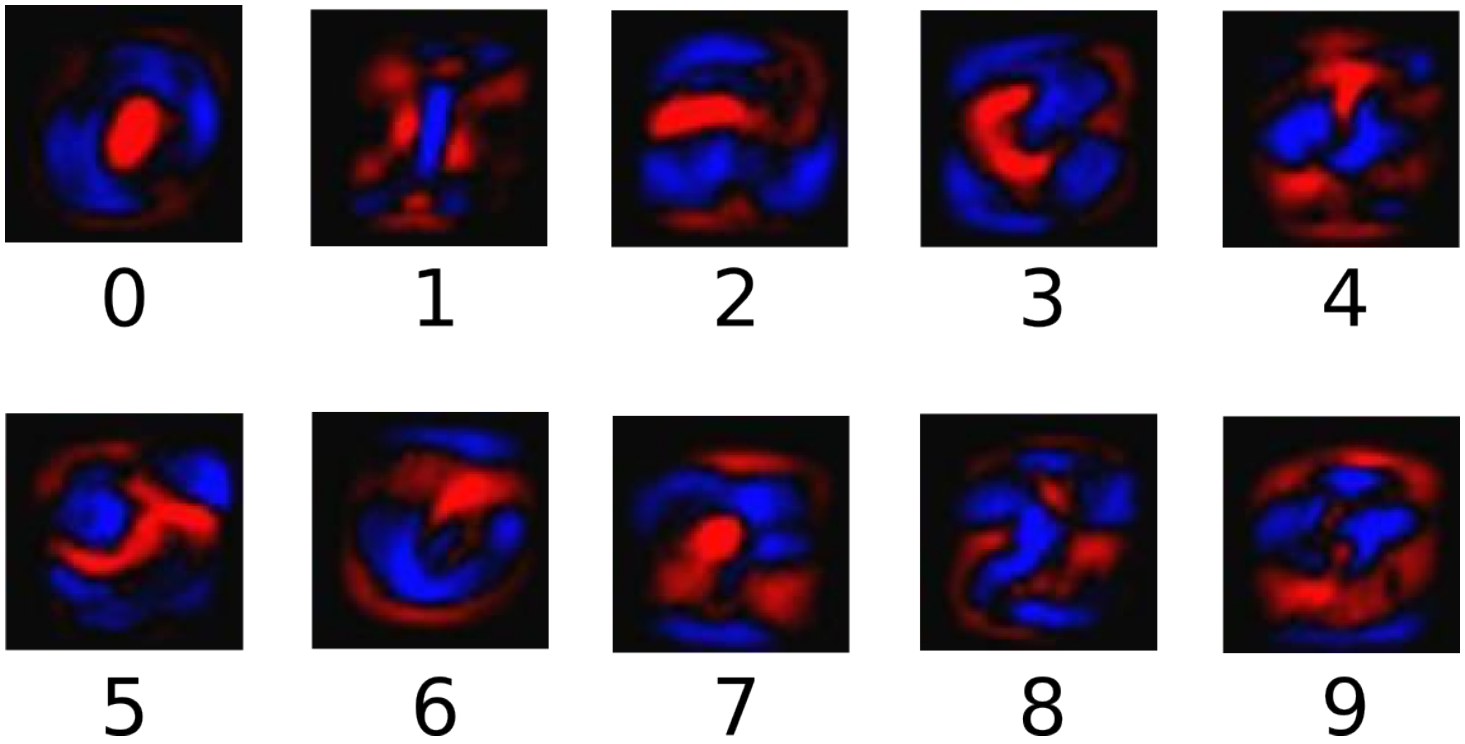


- Layers use convolutional weight sharing.
- Each $k \times k$ patch uses shared weights.
- Basis: Did this filter match patch in input?

First Layer Filters

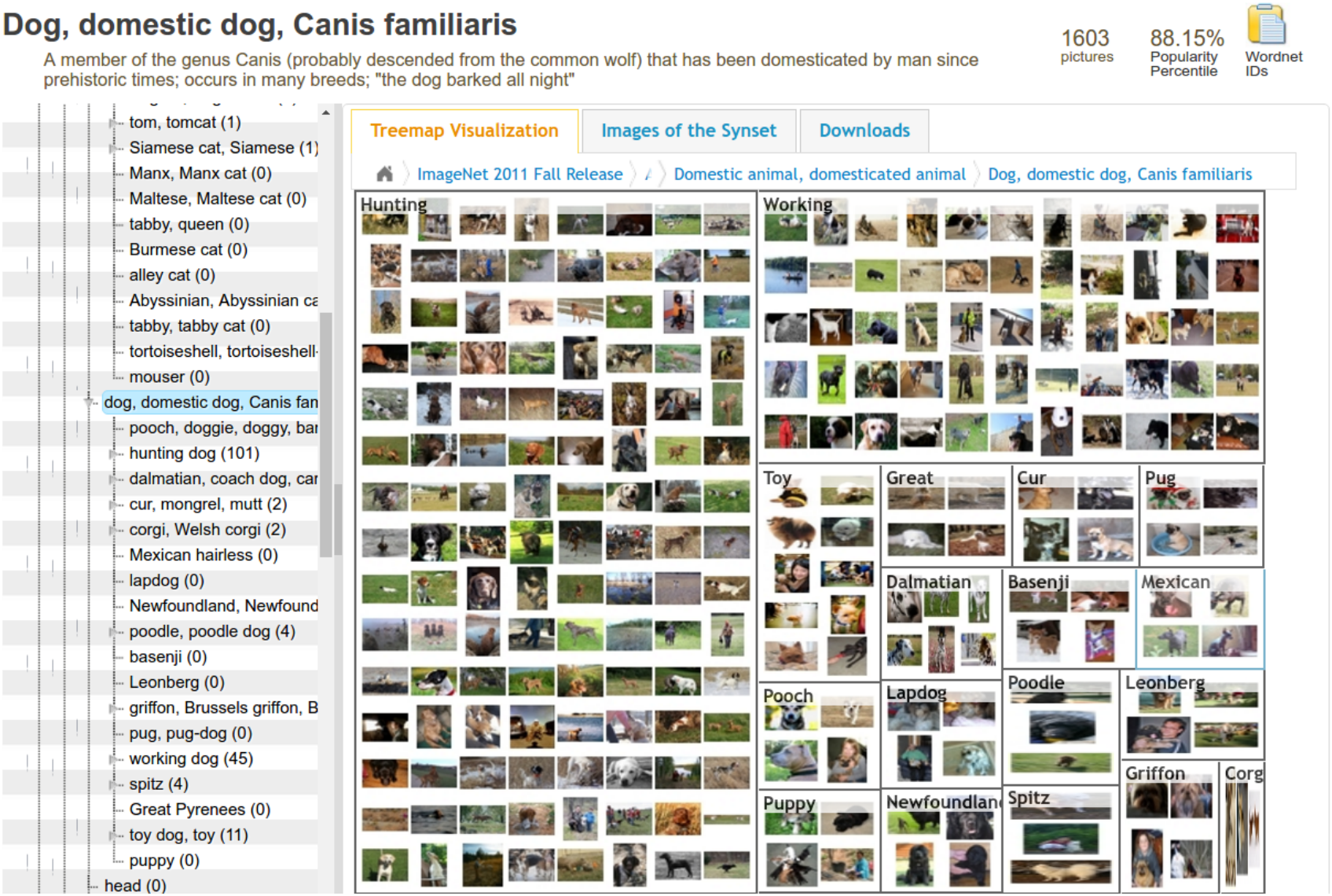


Final Transformations



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