Day 6: Neural networks, backpropagation

Introduction to Machine Learning Summer School
June 18, 2018 - June 29, 2018, Chicago

Instructor: Suriya Gunasekar, TTI Chicago

25 June 2018

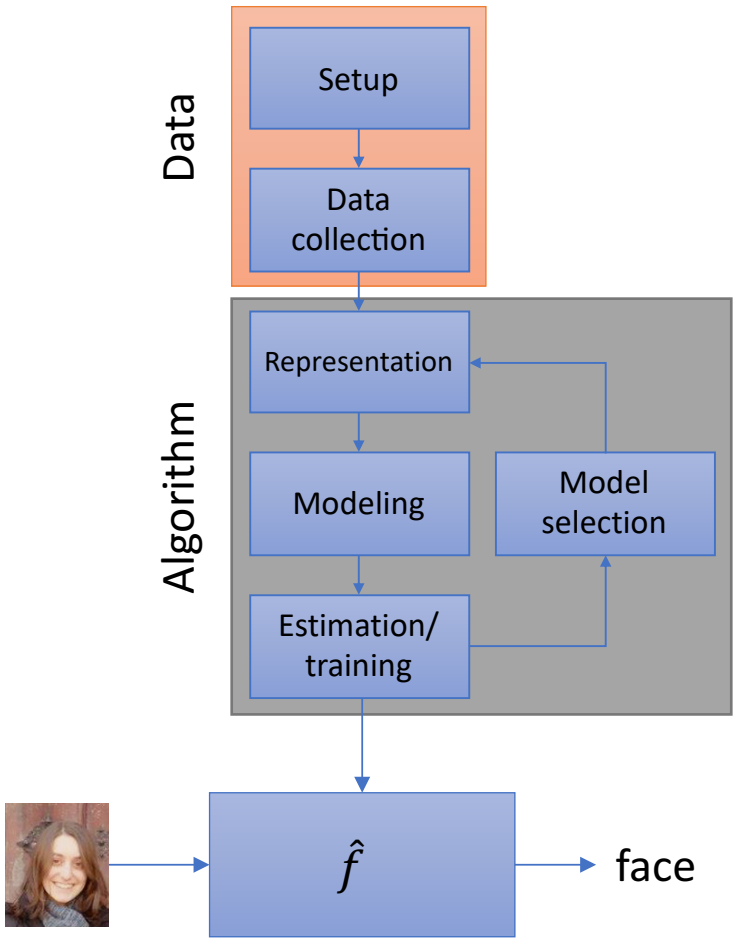
TOYOTA TECHNOLOGICAL INSTITUTE AT CHICAGO
THE UNIVERSITY OF CHICAGO

Schedule

- 9:00am-10:25am – Lecture 6.a: Review of week 1, introduction to neural networks
- 10:30am-11:30am – Invited Talk - Greg Durett (also the TTIC colloquium talk)
- 11:30am-12:30pm – Lunch
- 12:30pm-2:00pm – Lecture 6.b: Backpropagation
- 2:00pm-5:00pm – Programming

Review of week 1

Supervised learning – key questions



- Data: what kind of data can we get? how much data can we get?
- Model: what is the correct model for my data? – want to minimize the effort put into this question!
- Training: what resources - computation/memory - does the algorithm need to estimate the model $\hat{f}$?
- Testing: how well will $\hat{f}$ perform when deployed? what is the computational/memory requirement during deployment?

Linear regression

- Input $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^d$, output $y \in \mathbb{R}$, want to learn $f: \mathcal{X} \to \mathbb{R}$
- Training data $S = \{(\boldsymbol{x}^{(i)}, y^{(i)}): i = 1,2, \ldots, N\}$
- Parameterize candidate $f: \mathcal{X} \to \mathbb{R}$ by linear functions,

$$\mathcal{H} = \{\boldsymbol{x} \to \boldsymbol{w}.\boldsymbol{x}: w \in \mathbb{R}^d\}$$

- Estimate $\boldsymbol{w}$ by minimizing loss on training data

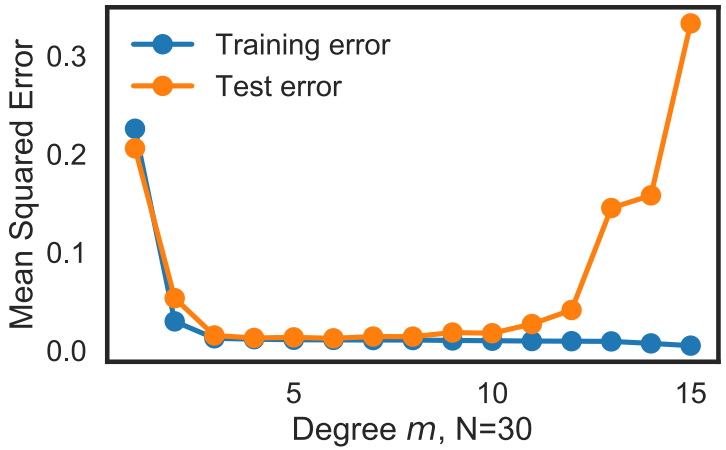
$$\widehat{\boldsymbol{w}} = \operatorname*{argmin}_{\boldsymbol{w}} J_S^{LS}(\boldsymbol{w}) := \sum_{i=1}^{N} \left(\boldsymbol{w}.\boldsymbol{x}^{(i)} - y^{(i)}\right)^2$$

  - $J_S^{LS}(\boldsymbol{w})$ is convex in $\boldsymbol{w}$ → minimize $J_S^{LS}(\boldsymbol{w})$ by setting gradient to 0
  - $\nabla_{\boldsymbol{w}} J_S^{LS}(\boldsymbol{w}) = \sum_{i=1}^{N} \left(\boldsymbol{w}.\boldsymbol{x}^{(i)} - y^{(i)}\right)\boldsymbol{x}^{(i)}$
  - Closed form solution $\widehat{\boldsymbol{w}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}\boldsymbol{y}$

- Can get non-linear functions by mapping $\boldsymbol{x} \to \phi(\boldsymbol{x})$ and doing linear regression on $\phi(\boldsymbol{x})$

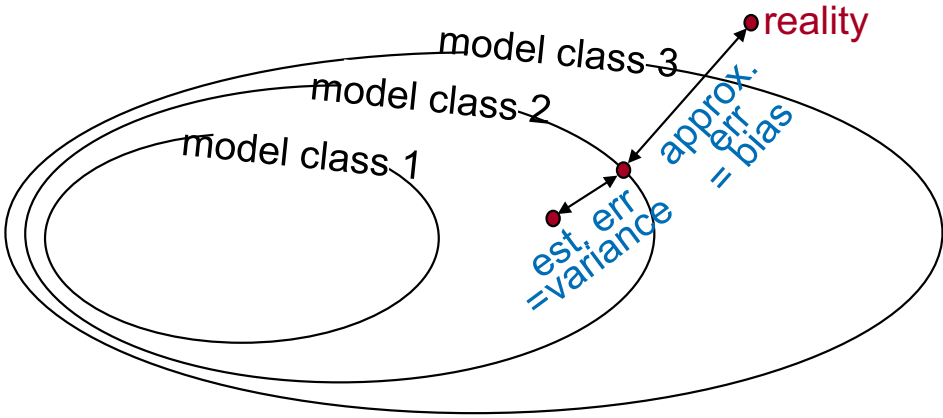
Overfitting

- For same amount of data, more complex models (e.g., higher degree polynomials) overfit more
- or need more data to fit more complex models
- complexity ≈ number of parameters



Model selection

- m model classes $\{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_m\}$
- $S = S_{train} \cup S_{val} \cup S_{test}$
- Train on S_{train} to pick best $\hat{f}_r \in \mathcal{H}_r$
- Pick $\hat{f}^*$ based on validation loss on S_{val}
- Evaluate test loss $L_{S_{test}}(\hat{f}^*)$



Regularization

- Complexity of model class can also be controlled by norm of parameters – smaller range of values allowed
- Regularization for linear regression

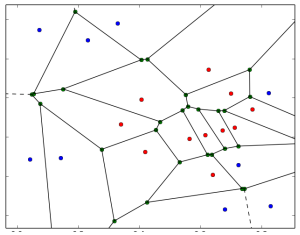
$$\underset{\boldsymbol{w}}{\operatorname{argmin}}\, J_S^{LS}(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_2^2$$

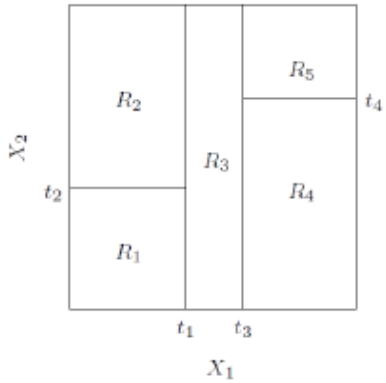
$$\underset{\boldsymbol{w}}{\operatorname{argmin}}\, J_S^{LS}(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_1$$

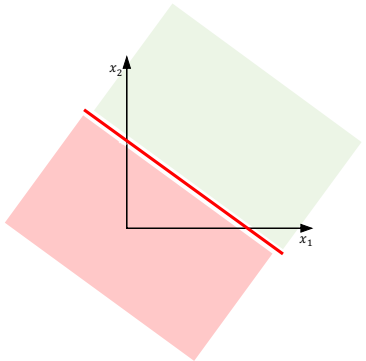
- Again do model selection to pick λ– using S_{val} or cross-validation

Classification

- Output $y \in \mathcal{Y}$ takes discrete set of values, e.g., $\mathcal{Y} = \{0,1\}$ or $\mathcal{Y} = \{-1,1\}$ or $\mathcal{Y} = \{spam, nospam\}$
  - Unlike regression, label-values do not have meaning
- Classifiers divide the space of input $\mathcal{X}$ (often $\mathbb{R}^d$) to "regions" where each region is assigned a label
- Non-parametric models
  - k-nearest neighbors – regions defined based on nearest neighbors
  - decision trees – structured rectangular regions
- Linear models – classifier regions are halfspaces



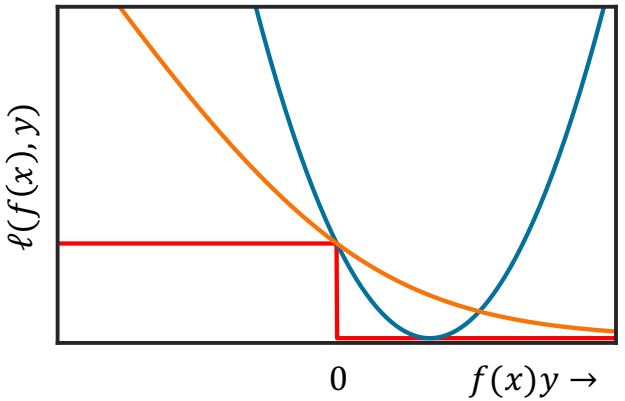




Classification – logistic regression

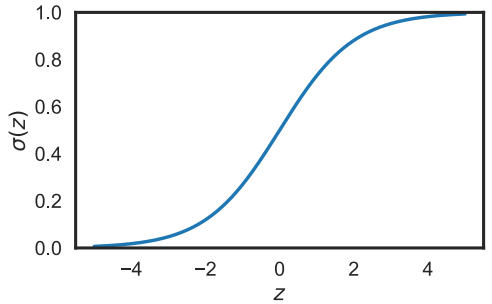
Logistic loss

$$\ell(f(\boldsymbol{x}), y) = \log\left(1 + \exp(-f(\boldsymbol{x})y)\right)$$



- $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \{-1,1\}, S = \{(\boldsymbol{x}^{(i)}, y^{(i)}): i = 1,2,\ldots,N\}$
- Linear model $f(\boldsymbol{x}) = f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}.\boldsymbol{x}$
- Output classifier $\hat{y}(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}.\boldsymbol{x})$
- Empirical risk minimization

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\text{argmin}} \sum_i \log\left(1 + \exp\left(-\boldsymbol{w}.\boldsymbol{x}^{(i)} y^{(i)}\right)\right)$$



- Probabilistic formulation: $\Pr(y = 1|\boldsymbol{x}) = \frac{1}{1+\exp(-\boldsymbol{w}.\boldsymbol{x})}$
- Multi-class generalization: $\mathcal{Y} = \{1,2,\ldots,m\}$
- $\Pr(y|\boldsymbol{x}) = \frac{\exp(-\boldsymbol{w}_y.\boldsymbol{x})}{\sum_{y'} \exp\left(-\boldsymbol{w}_{y'}.\boldsymbol{x}\right)}$
- Can again get non-linear decision boundaries by mapping $\boldsymbol{x} \rightarrow \phi(\boldsymbol{x})$

Classification – maximum margin classifier

Separable data

- Original formulation

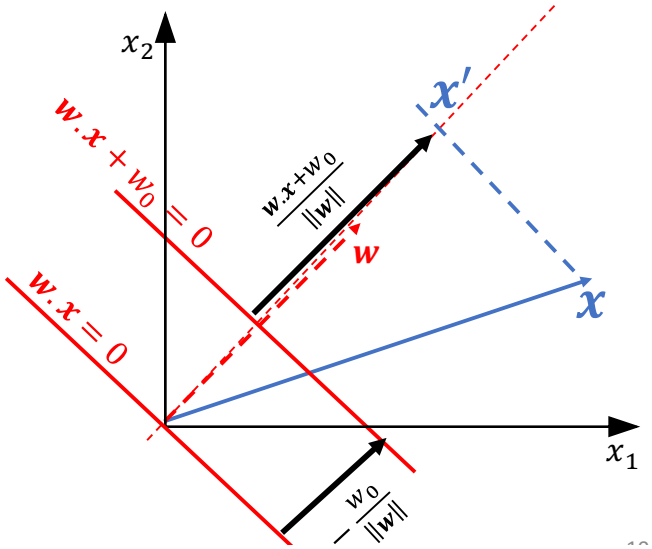
$$\widehat{\boldsymbol{w}} = \underset{\boldsymbol{w}\in\mathbb{R}^d}{\operatorname{argmax}} \quad \min_i \frac{y^{(i)}\boldsymbol{w}.\boldsymbol{x}^{(i)}}{\|\boldsymbol{w}\|}$$

- Fixing $\|\boldsymbol{w}\| = 1$

$$\widehat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmax}} \quad \min_i y^{(i)}\left(\boldsymbol{w}.\boldsymbol{x}^{(i)}\right) \quad \text{s.t.} \quad \|\boldsymbol{w}\| = 1$$

- Fixing $\min_i \; y^{(i)}\boldsymbol{w}.\boldsymbol{x}^{(i)} = 1$

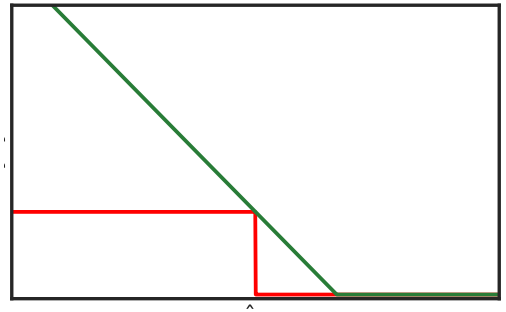
$$\widetilde{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \|\boldsymbol{w}\|^2 \quad \text{s.t.} \quad \forall i, y^{(i)}(\boldsymbol{w}.\boldsymbol{x}^{(i)}) \geq 1$$



Slack variables for non-separable data

$$\widehat{\boldsymbol{w}} = \underset{\boldsymbol{w},\{\xi_i\geq 0\}}{\operatorname{argmin}} \|\boldsymbol{w}\|^2 + \lambda \sum_i \xi_i \quad \text{s.t.} \quad \forall i, y^{(i)}\left(\boldsymbol{w}.\boldsymbol{x}^{(i)}\right) \geq 1 - \xi_i$$

$$= \underset{\boldsymbol{w},\{\xi_i\geq 0\}}{\operatorname{argmin}} \|\boldsymbol{w}\|^2 + \lambda \sum_i \max\left(0, 1 - y^{(i)}\left(\boldsymbol{w}.\boldsymbol{x}^{(i)}\right)\right)$$



Kernel trick

- Using representor theorem $\boldsymbol{w} = \sum_{i=1}^{N} \beta_i \boldsymbol{x}^{(i)}$

$$\min_{\boldsymbol{w}} \|\boldsymbol{w}\|^2 + \lambda \sum_i \max\left(0, 1 - y^{(i)}\, \boldsymbol{w}.\boldsymbol{x}^{(i)}\right)$$

$$\equiv \min_{\boldsymbol{\beta} \in \mathbb{R}^N} \boldsymbol{\beta}^\top \boldsymbol{G} \boldsymbol{\beta} + \lambda \sum_i \max\left(0, 1 - y^{(i)} (\boldsymbol{G}\boldsymbol{\beta})_i\right)$$

$\boldsymbol{G} \in \mathbb{R}^{N \times N}$ with $G_{ij} = \boldsymbol{x}^{(i)}.\boldsymbol{x}^{(j)}$ is called the gram matrix

- Optimization depends on $\boldsymbol{x}^{(i)}$ only through $G_{ij} = \boldsymbol{x}^{(i)}.\boldsymbol{x}^{(j)}$
- For prediction $\widehat{\boldsymbol{w}}.\boldsymbol{x} = \sum_i \beta_i\, \boldsymbol{x}^{(i)}.\boldsymbol{x}$, we again only need $\boldsymbol{x}^{(i)}.\boldsymbol{x}$
- Function $K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}.\boldsymbol{x}'$ is called the Kernel
- When learning non-linear classifiers using feature transformations $\boldsymbol{x} \to \phi(\boldsymbol{x})$ and $f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}.\phi(\boldsymbol{x})$
  - Classifier fully specified in terms of $K_\phi(\boldsymbol{x}, \boldsymbol{x}') = K(\phi(\boldsymbol{x}), \phi(\boldsymbol{x}'))$
  - $\phi(\boldsymbol{x})$ itself can be very very high dimensional (maybe even infinite dimensional)

Optimization

- ERM+regularization optimization problem

$$\widehat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} J_S^{\lambda}(\boldsymbol{w}) := \sum_{i=1}^{N} \ell(\boldsymbol{w}.\phi(\boldsymbol{x}^{(i)}), y^{(i)}) + \lambda \|\boldsymbol{w}\|$$

- If $J_S^{\lambda}(\boldsymbol{w})$ is convex in $\boldsymbol{w}$, then $\widehat{\boldsymbol{w}}$ is optimum if and only if gradient at $\widehat{\boldsymbol{w}}$ is 0, i.e., $\nabla J_S^{\lambda}(\widehat{\boldsymbol{w}}) = 0$
- Gradient descent: start with initialization $\boldsymbol{w^0}$ and iteratively update
  - $\boldsymbol{w^{t+1}} = \boldsymbol{w^t} - \eta^t \nabla J_S^{\lambda}(\boldsymbol{w^t})$
  - where $\nabla J_S^{\lambda}(\boldsymbol{w^t}) = \sum_{\boldsymbol{i}} \nabla \ell\left(\boldsymbol{w^t}.\phi\left(\boldsymbol{x^{(i)}}\right), y^{(i)}\right) + \lambda \nabla \|\boldsymbol{w^t}\|$
- Stochastic gradient descent
  - use gradients from only one example
  - $\boldsymbol{w^{t+1}} = \boldsymbol{w^t} - \eta^t \, \hat{\nabla}^{(i)} J_S^{\lambda}(\boldsymbol{w^t})$
  - where $\hat{\nabla}^{(i)} J_S^{\lambda}(\boldsymbol{w^t}) = \nabla \ell\left(\boldsymbol{w^t}.\phi\left(\boldsymbol{x^{(i)}}\right), y^{(i)}\right) + \lambda \nabla \|\boldsymbol{w^t}\|$ for a random sample $(\boldsymbol{x^{(i)}}, y^{(i)})$

Other classification models

- Optimal unrestricted predictor
  - Regression + squared loss→ $f^{**}(\boldsymbol{x}) = \mathbf{E}[y|\boldsymbol{x}]$
  - Classification + 0-1 loss → $\hat{y}^{**}(\boldsymbol{x}) = \text{argmax}_c \Pr(y = c|\boldsymbol{x})$
- Discriminative models: directly model $\Pr(y|\boldsymbol{x})$, e.g., logistic regression
- Generative models: model full joint distribution $\Pr(y, \boldsymbol{x}) = \Pr(\boldsymbol{x}|y) \Pr(y)$
- Why generative models?
  - One conditional might be simpler to model with prior knowledge, e.g., compare specifying $\Pr(\text{image}|\text{digit} = 1)$ vs $\Pr(\text{digit} = 1|\text{image})$
  - Naturally handles missing data
- Two examples of generative models
  - Naïve Bayes classifier
  - Hidden Markov model

Other classifiers

- Naïve Bayes classifier: with d features $x = [x_1, x_2, \dots, x_d]$ where each $x_1, x_2, \dots, x_d$ can take one of K values → $C\ K^d$ parameters
  - NB assumption: features are independent given class y → $C\ K\ d$ params.

$$\Pr(x_1, x_2, \dots, x_d|y) = \Pr(x_1|y)\Pr(x_2|y)\dots\Pr(x_d|y) = \prod_{k=1}^{d} \Pr(x_k|y)$$

  - Training amounts to averaging samples across classes
- Hidden Markov model: variable length input/observations $\{x_1, x_2, \dots, x_m\}$ (e.g., words) and variable length output/state $\{y_1, y_2, \dots, y_m\}$ (e.g., tags)
  - HMM assumption: a) current state conditioned on immediate previous state is conditionally independent of all other variables, and (b) current observation conditioned on current state is conditionally independent of all other variables.

$$\Pr(x_1, x_2, \dots, x_m, y_1, y_2, \dots, y_m) = \Pr(y_1)\Pr(x_1|y_1)\prod_{k=2}^{m}\Pr(y_k|y_{k-1})\Pr(y_k|x_k)$$

  - Parameters estimated using MLE dynamic programming

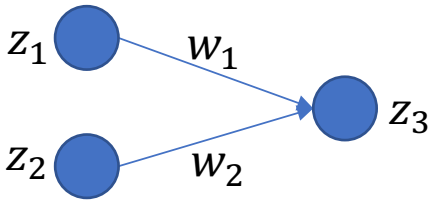
Today

Introduction to neural networks

Backpropagation

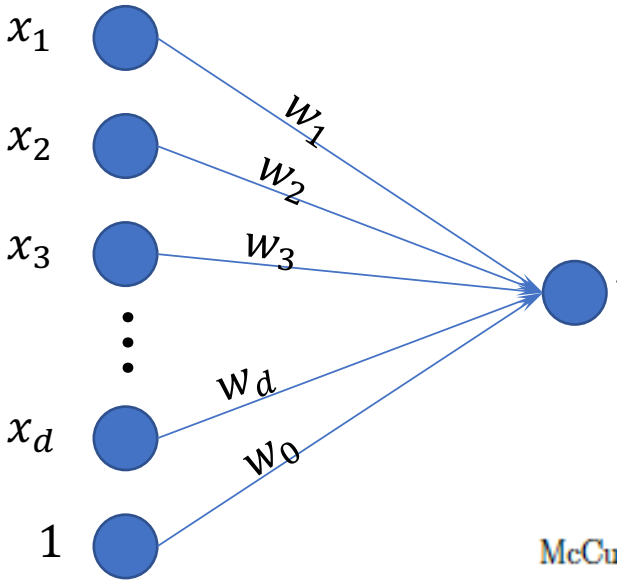
Graph notation

- General variables
  - can be input variables like $x_1, x_2, \ldots x_d$
  - prediction $\hat{y}$
  - or any intermediate computation (we will see examples soon)



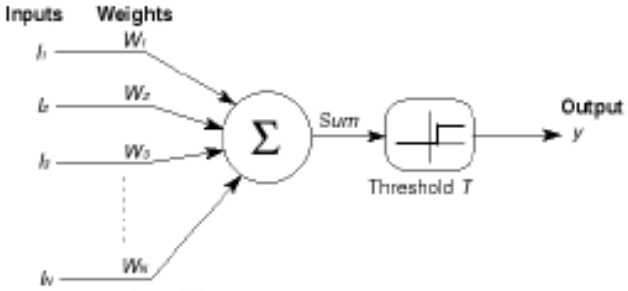
denotes computation $z_3 = \sigma(w_1 z_1 + w_2 z_2)$ for some "activation" function σ (specified apriori)

Linear classifier



$$f(\boldsymbol{x}) = \mathbf{1}(\boldsymbol{w}.\boldsymbol{x} + w_0 \geq 0)$$

McCullock and Pitts 1943 — introduced the linear threshold "neuron".



- Biological analogy: single neuron – stimuli reinforce synaptic connections

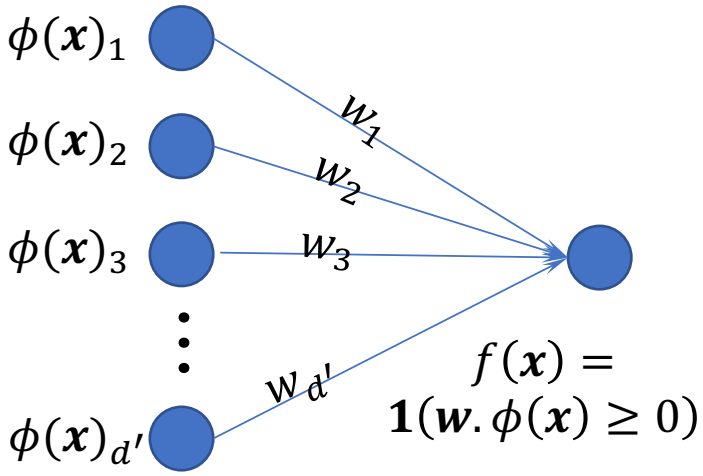
Slide credits: Nati Srebro, David McAllester

Shallow learning

- We already saw how to use linear models to get non-linear decision boundaries
- Feature transform: map $\boldsymbol{x} \in \mathbb{R}^d$ to $\boldsymbol{\phi}(\boldsymbol{x}) \in \mathbb{R}^{d'}$ and use

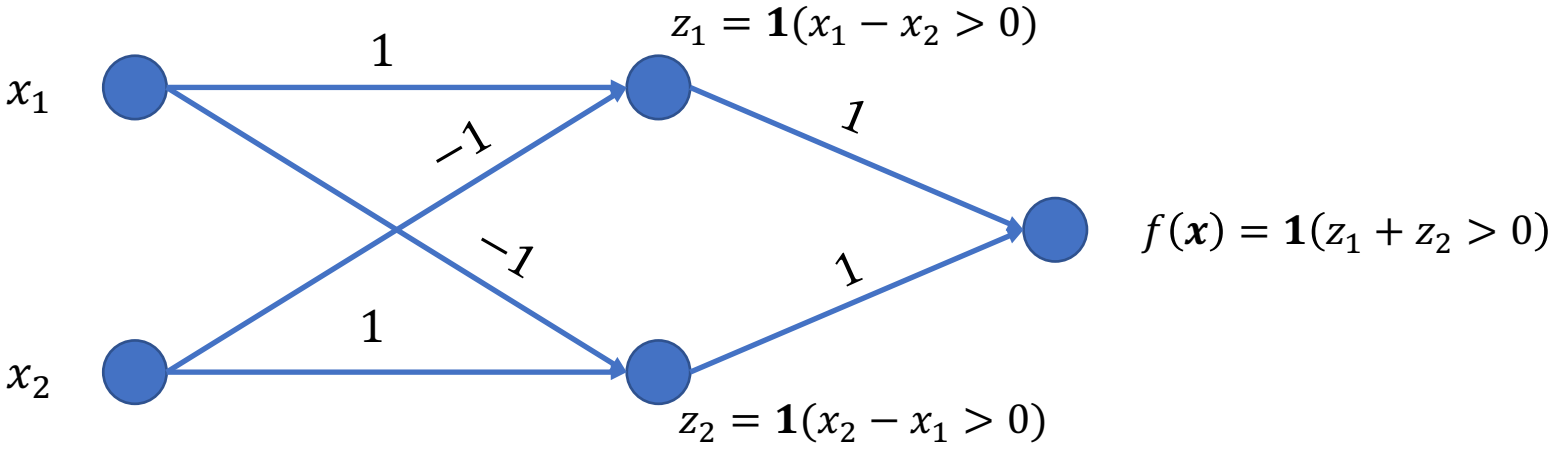
$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}.\boldsymbol{\phi}(\boldsymbol{x})$$

- Shallow learning: hand-crafted and non-hierarchical $\boldsymbol{\phi}$
  - Polynomial regression with squared or logistic loss, $\phi(x)_p = x^p$
  - Kernel SVM: $K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x}).\boldsymbol{\phi}(\boldsymbol{x}')$



Slide credit: Nati Srebro

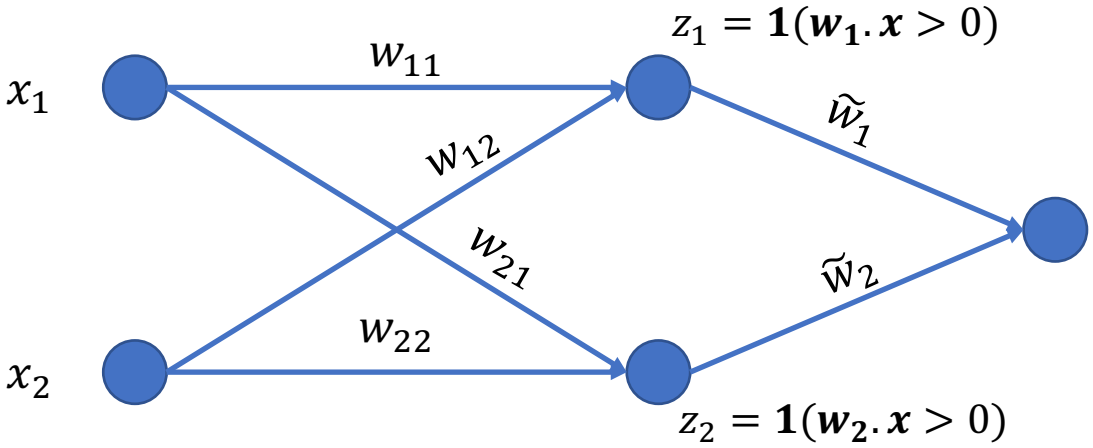
Combining Linear Units



- The network represents the function
  $f(x) = \left(x_1 \text{ and not}(x_2)\right) \text{ or } \left(x_2 \text{ and not}(x_1)\right)$
- Not a linear function of x

Slide credit: Nati Srebro

Combining Linear Units



$$f(\boldsymbol{x}) = \mathbf{1}(\tilde{w}_1 z_1 + \tilde{w}_2 z_2 \geq 0)$$

Feed-Forward Neural Networks

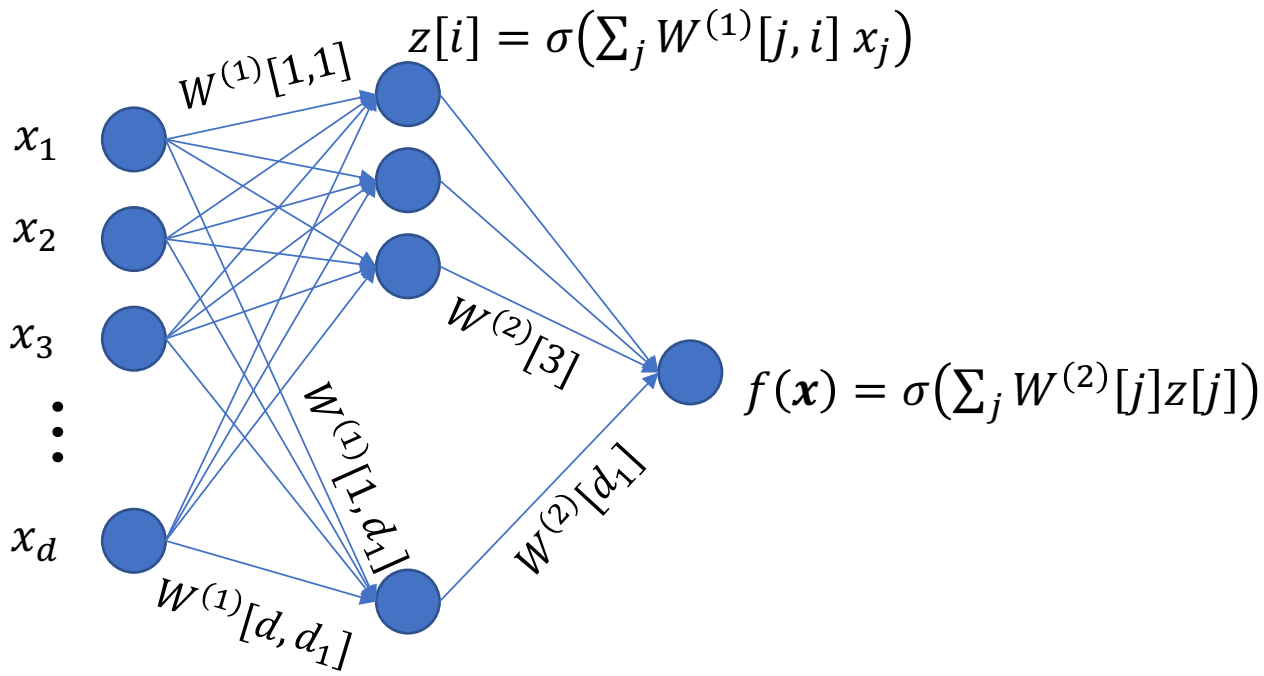
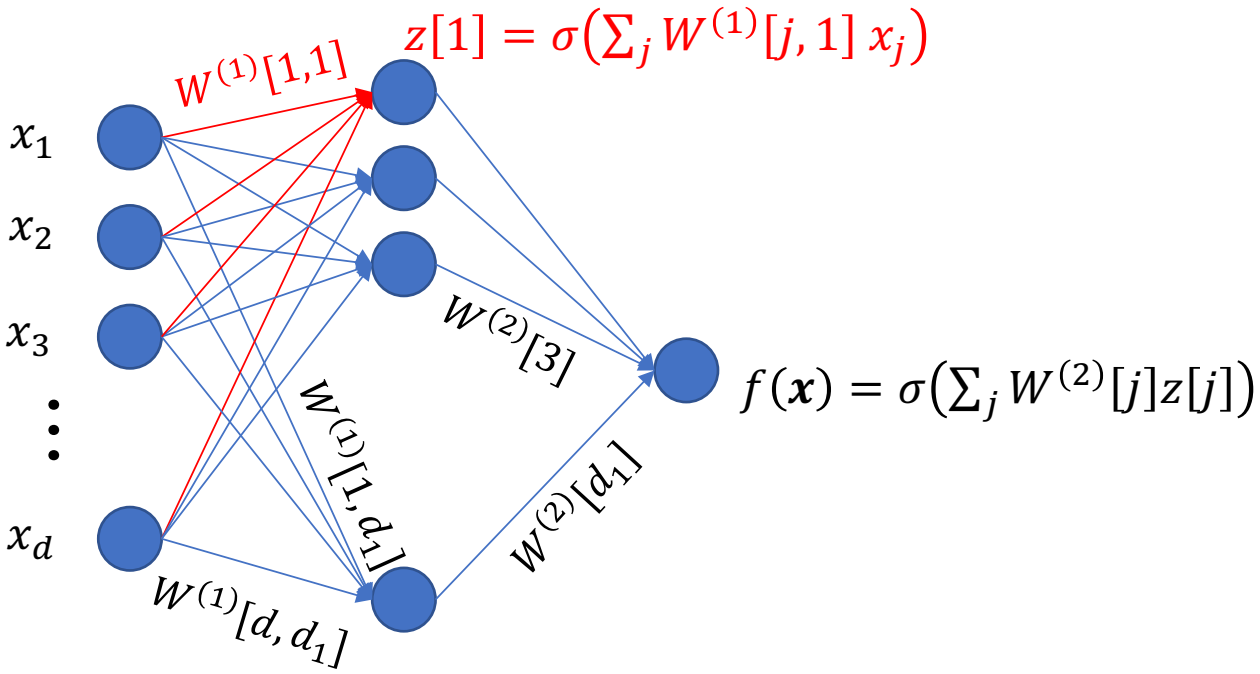
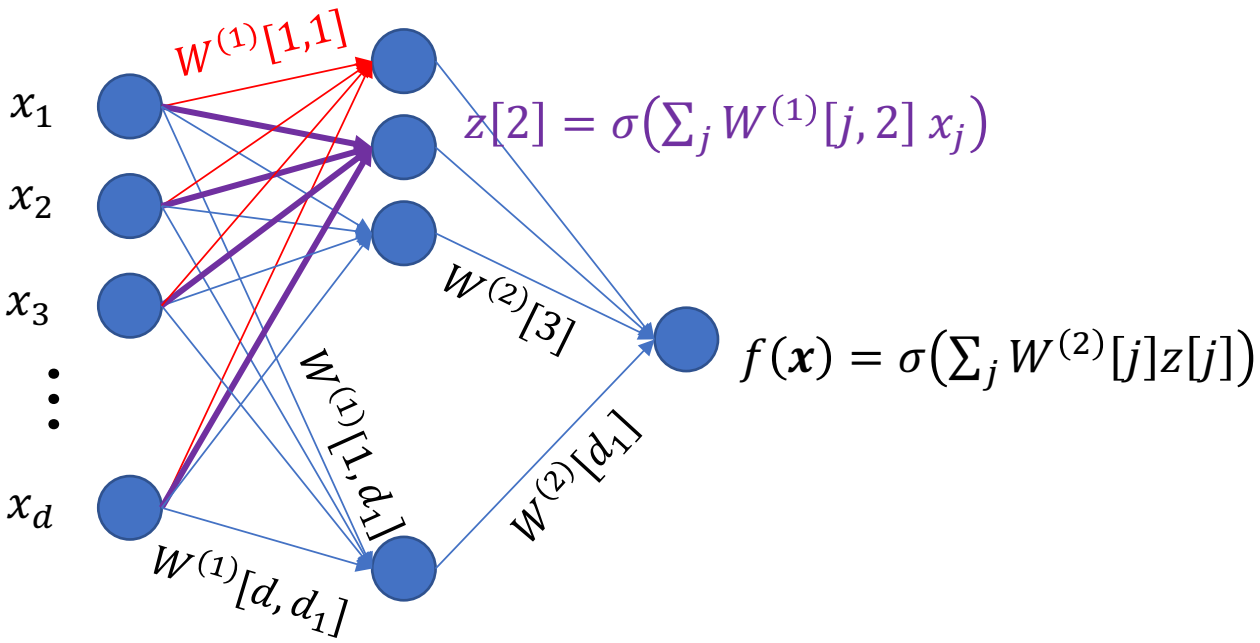


Figure credit: Nati Srebro

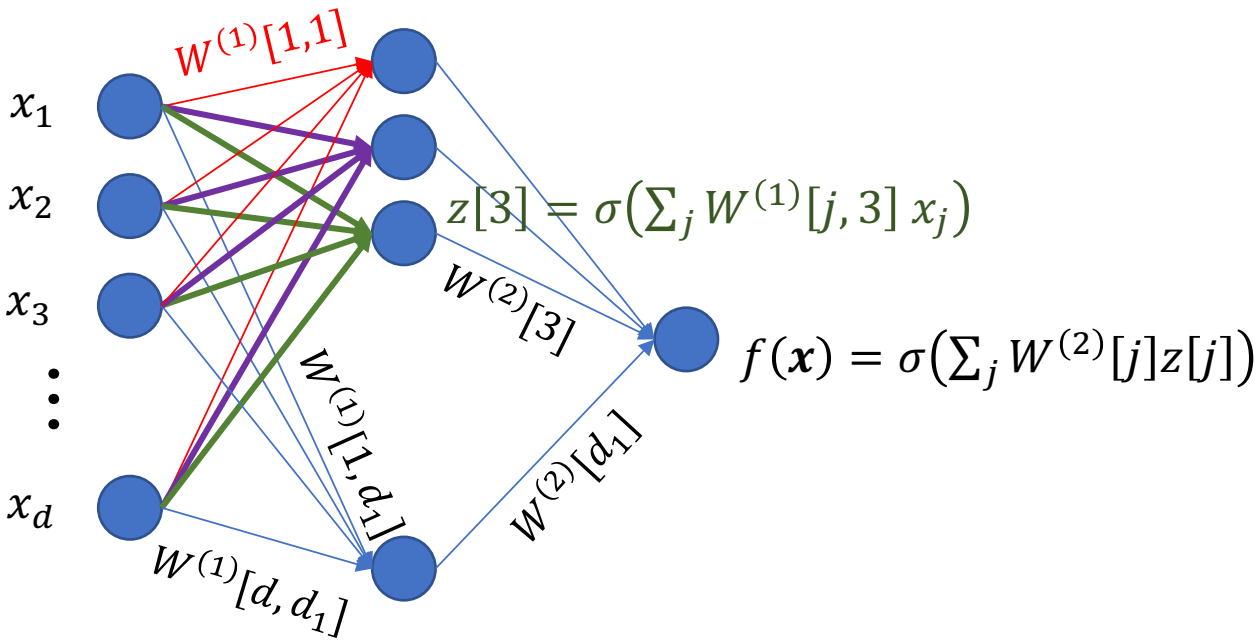
Feed-Forward Neural Networks



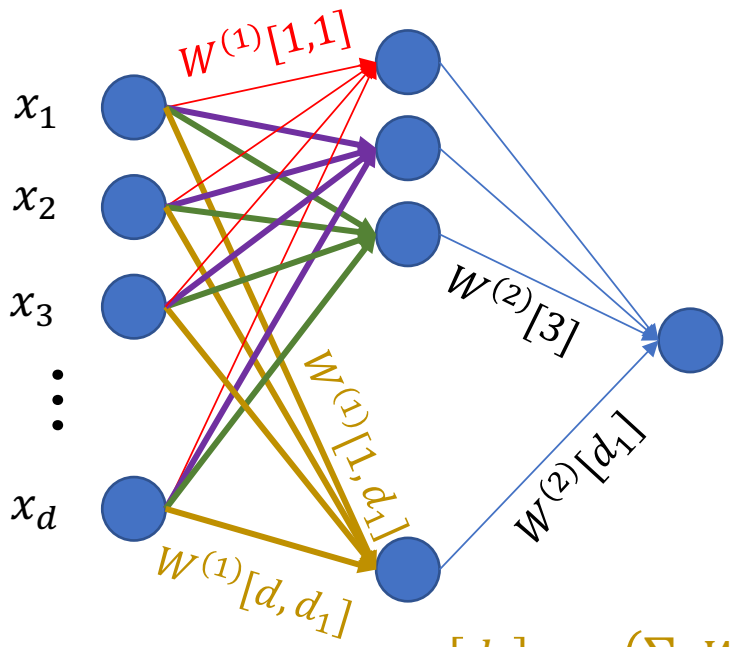
Feed-Forward Neural Networks



Feed-Forward Neural Networks



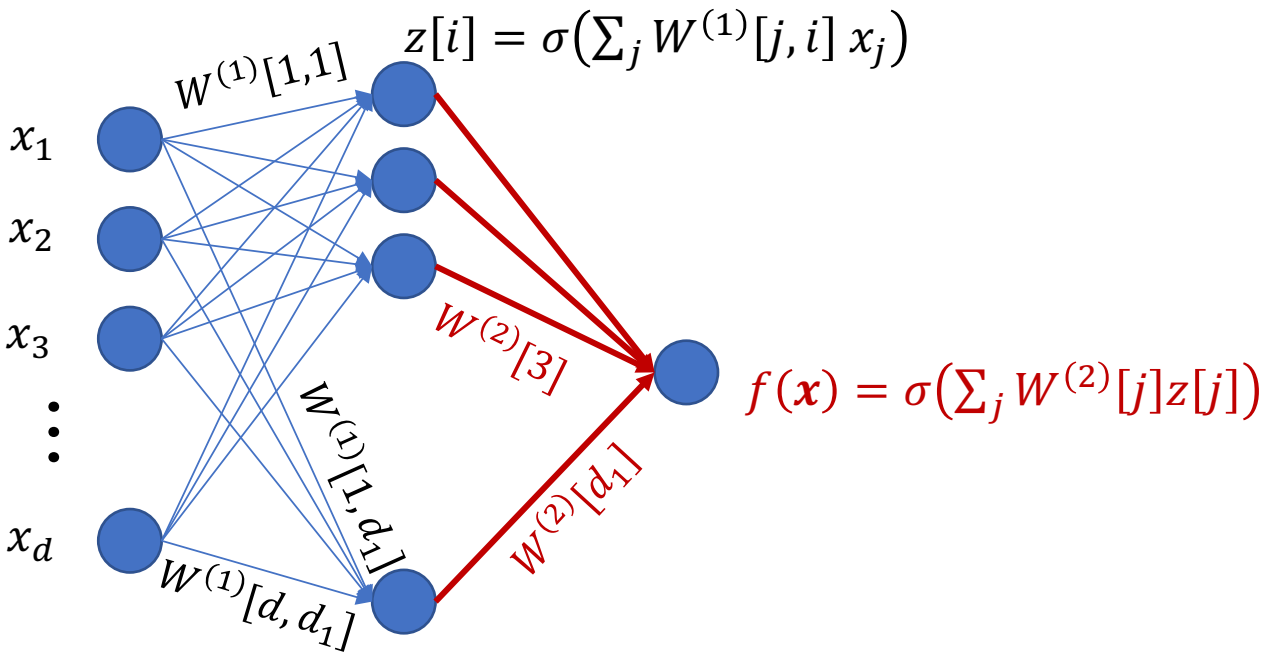
Feed-Forward Neural Networks



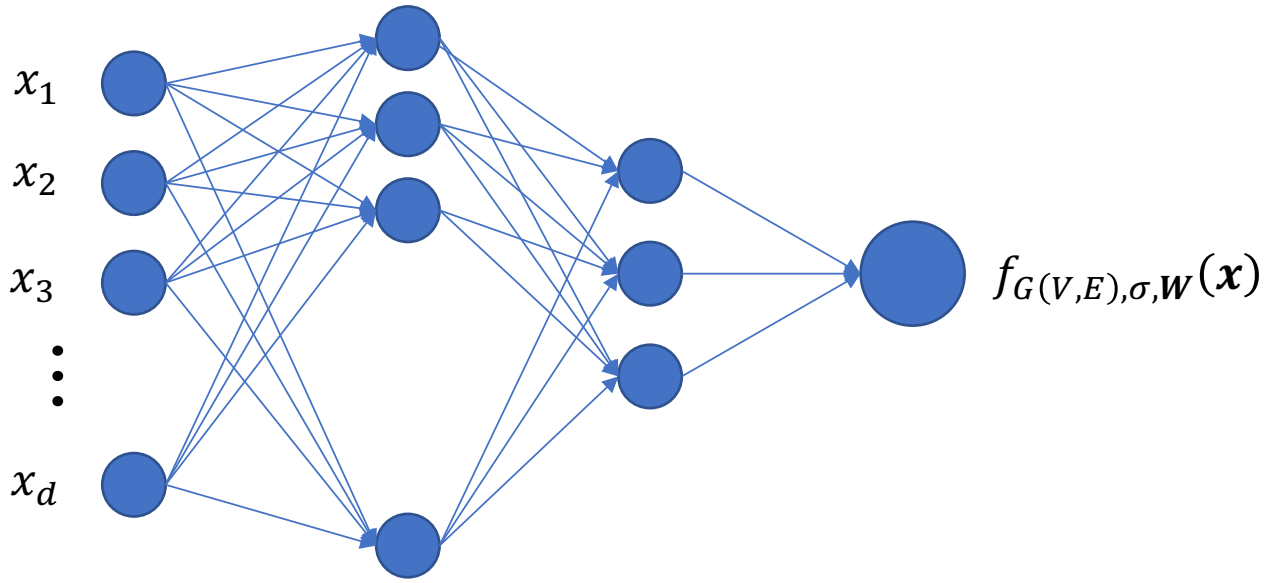
$$f(\boldsymbol{x}) = \sigma\left(\sum_j W^{(2)}[j] z[j]\right)$$

$$z[d_1] = \sigma\left(\sum_j W^{(1)}[j,d_1]\, x_j\right)$$

Feed-Forward Neural Networks



Feed-Forward Neural Networks

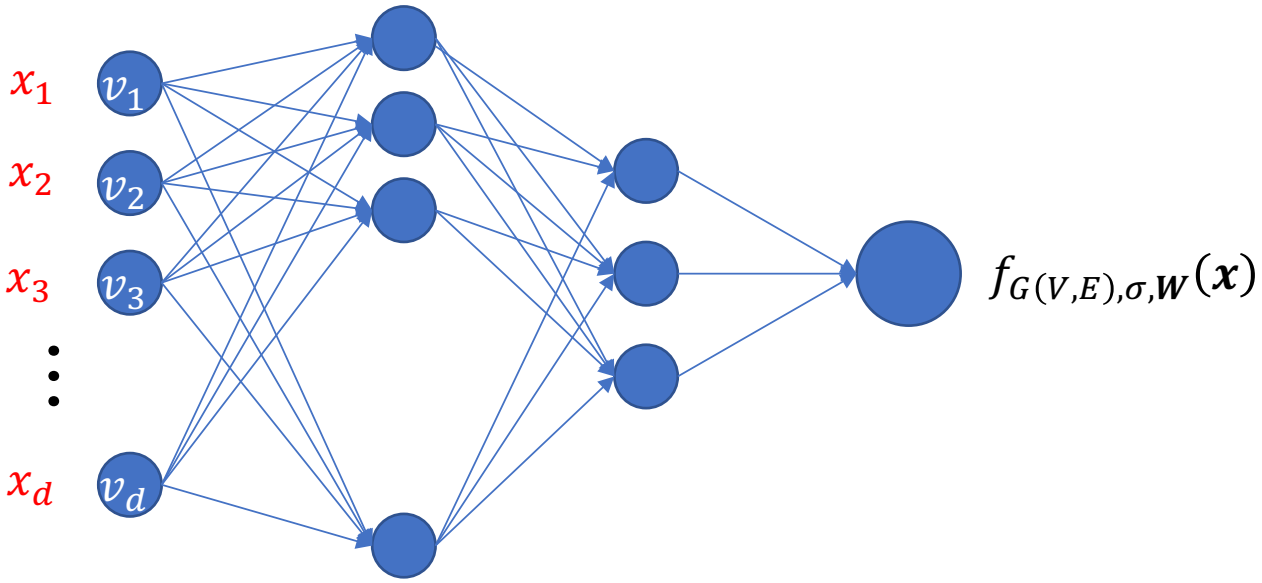


Architecture:

- Directed Acyclic Graph $G(V, E)$. Units (neurons) indexed by vertices in V.

Slide credit: Nati Srebro

Feed-Forward Neural Networks

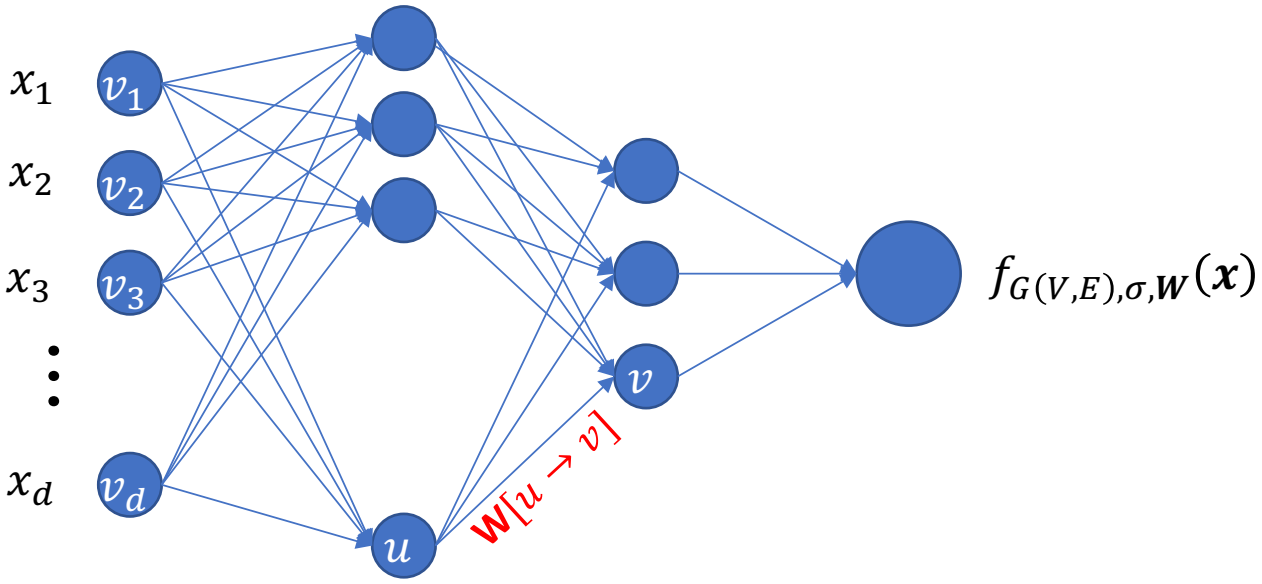


Architecture:

- Directed Acyclic Graph G(V,E). Units (neurons) indexed by vertices in V.
  - "Input Units" $v_1 \ldots v_d \in V$: no incoming edges have value $o[v_i] = x_i$

Slide credit: Nati Srebro

Feed-Forward Neural Networks

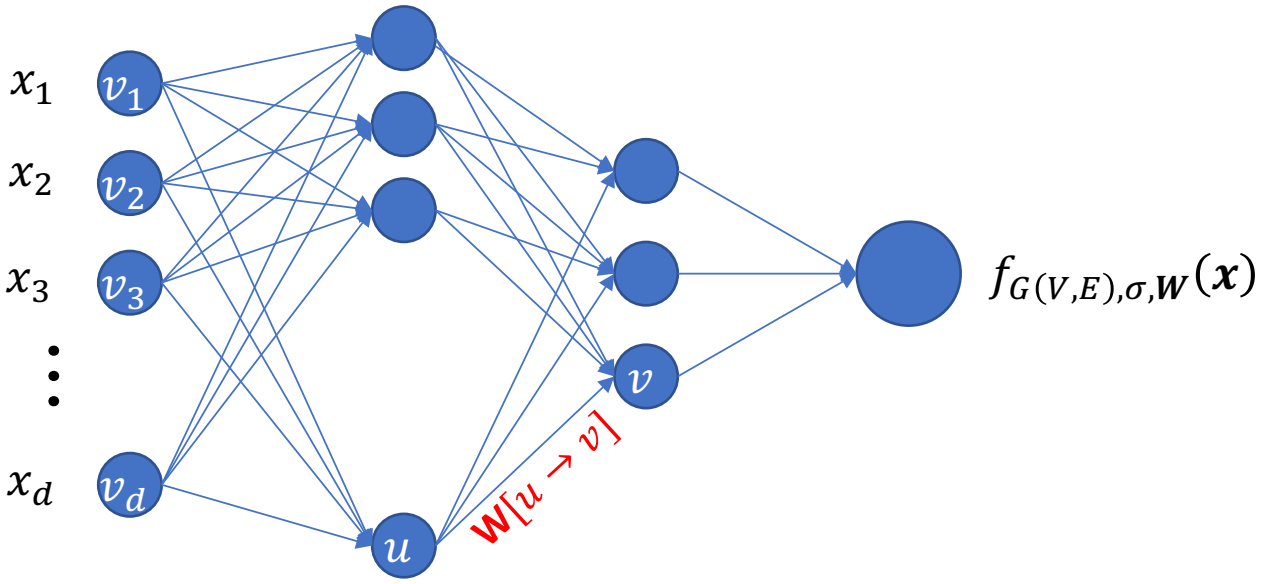


Architecture:

- Directed Acyclic Graph G(V,E). Units (neurons) indexed by vertices in V.
  - "Input Units" $v_1 \ldots v_d \in V$: no incoming edges have value $o[v_i] = x_i$
  - Each edge $u \to v$ has weight $\boldsymbol{W}[u \to v]$
    - Pre-activation $a[v] = \sum_{u \to v \in E} \boldsymbol{W}[u \to v]\, o[u]$

Slide credit: Nati Srebro

Feed-Forward Neural Networks

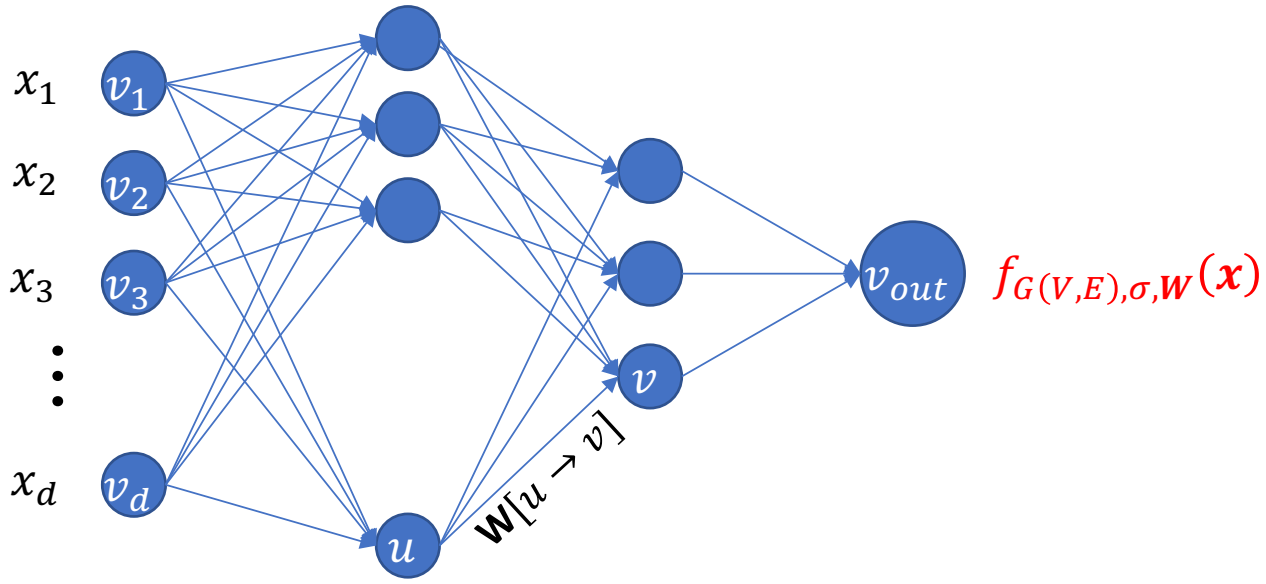


Architecture:

- Directed Acyclic Graph G(V,E). Units (neurons) indexed by vertices in V.
  - "Input Units" $v_1 \ldots v_d \in V$: no incoming edges have value $o[v_i] = x_i$
  - Each edge $u \to v$ has weight $\boldsymbol{W}[u \to v]$
    - Pre-activation $a[v] = \sum_{u \to v \in E} \boldsymbol{W}[u \to v]\, o[u]$
    - Output value $o[v] = \sigma(\, a[v]\,)$

Slide credit: Nati Srebro

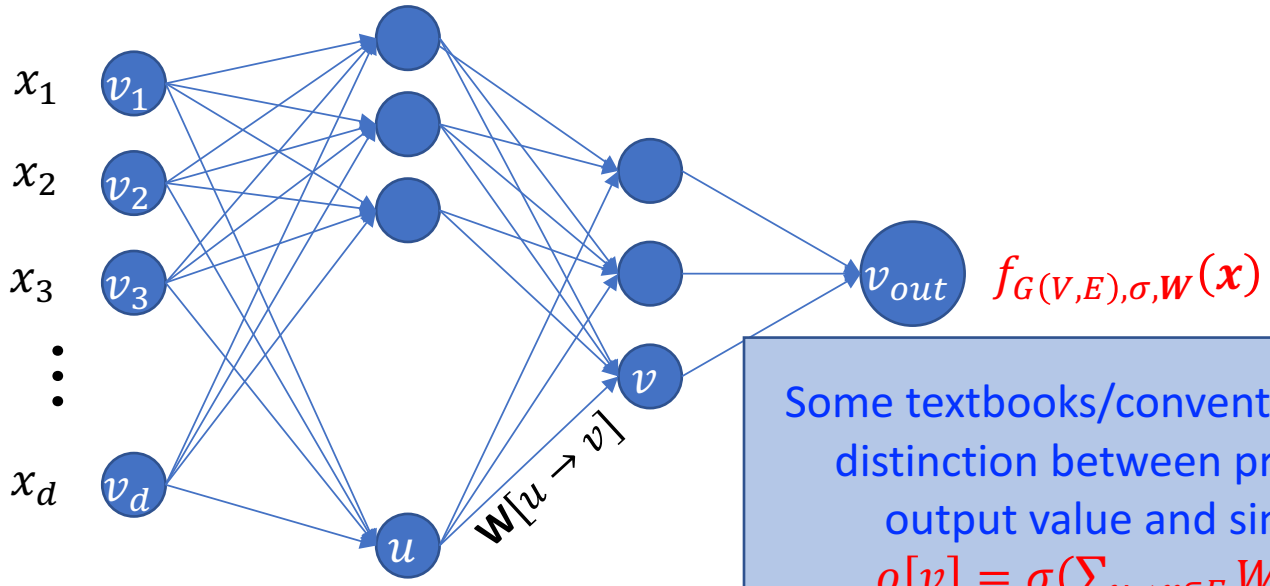
Feed-Forward Neural Networks



Architecture:

- Directed Acyclic Graph G(V,E). Units (neurons) indexed by vertices in V.
  - "Input Units" $v_1 \dots v_d \in V$: no incoming edges have value $o[v_i] = x_i$
  - Each edge $u \to v$ has weight $\boldsymbol{W}[u \to v]$
    - Pre-activation $a[v] = \sum_{u \to v \in E} \boldsymbol{W}[u \to v]\, o[u]$
    - Output value $o[v] = \sigma(\, a[v]\,)$
  - "Output Unit" $v_{out} \in V$, $f_W(\boldsymbol{x}) = a[v_{out}]$

Feed-Forward Neural Networks

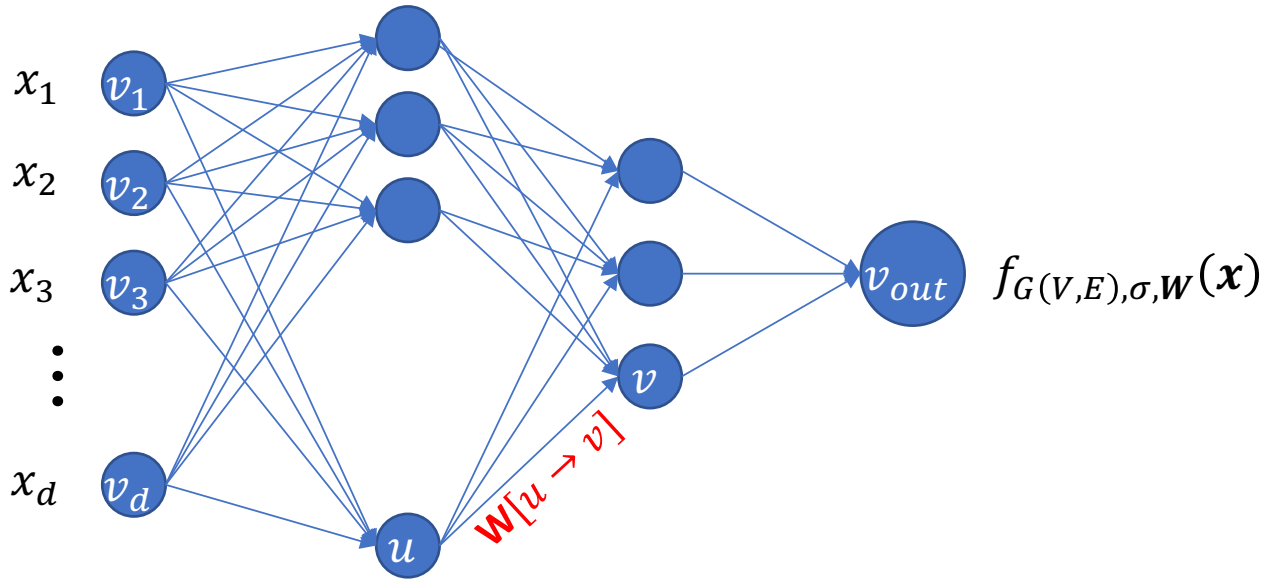


Some textbooks/convention don't make the distinction between pre-activation and output value and simply compute $o[v] = \sigma(\sum_{u \to v \in E} W[u \to v] o[u])$

Architecture:

- Directed Acyclic Graph G(V,E). Units (neurons) indexed by vertices in V.
  - "Input Units" $v_1 \dots v_d \in V$: no incoming edges have value $o[v_i] = x_i$
  - Each edge $u \to v$ has weight $\boldsymbol{W}[u \to v]$
    - Pre-activation $a[v] = \sum_{u \to v \in E} \boldsymbol{W}[u \to v]\, o[u]$
    - Output value $o[v] = \sigma(\, a[v]\,)$
  - "Output Unit" $v_{out} \in V$, $f_W(\boldsymbol{x}) = a[v_{out}]$

Feed-Forward Neural Networks



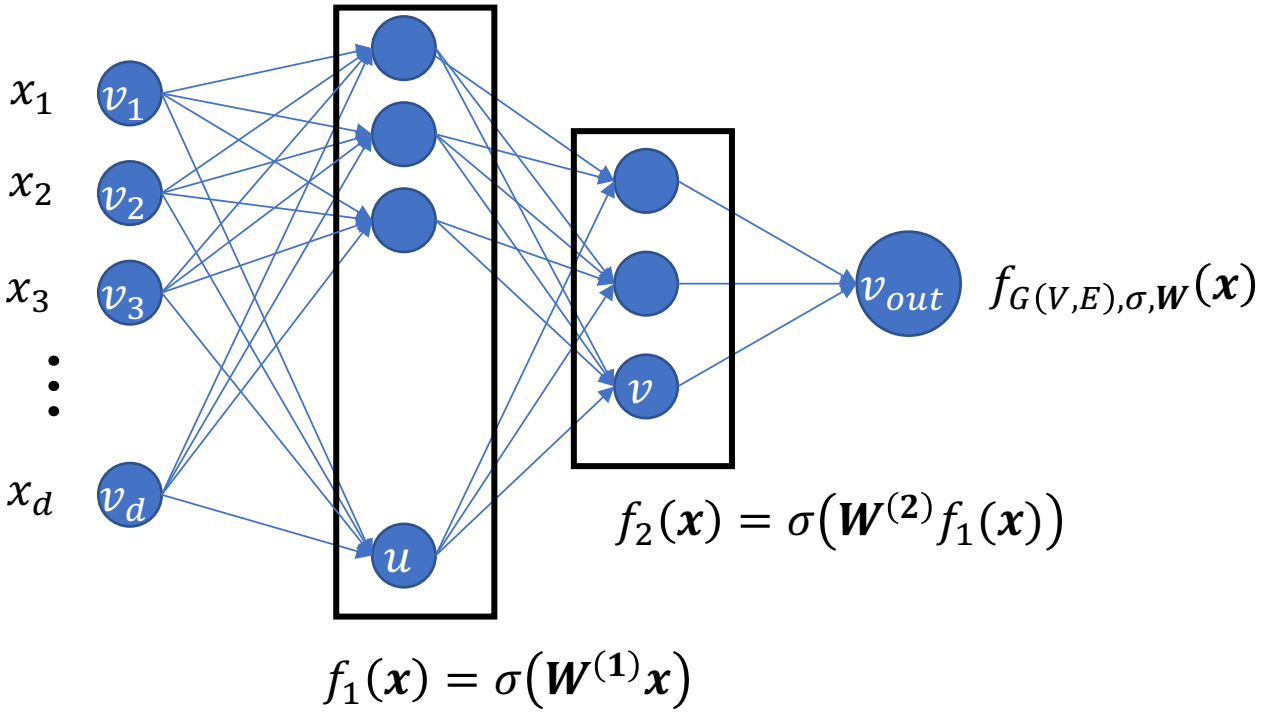
Parameters:

- Each edge $u \to v$ has weight $\boldsymbol{W}[u \to v]$

Activations:

- $\sigma: \mathbb{R} \to \mathbb{R}$, for example
  - $\sigma(z) = sign(z)$ or $\sigma(z) = \frac{1}{1+\exp(-z)}$
  - $\sigma(z) = \mathrm{ReLU}(z) = \max(0, z)$

Feed-Forward Neural Networks

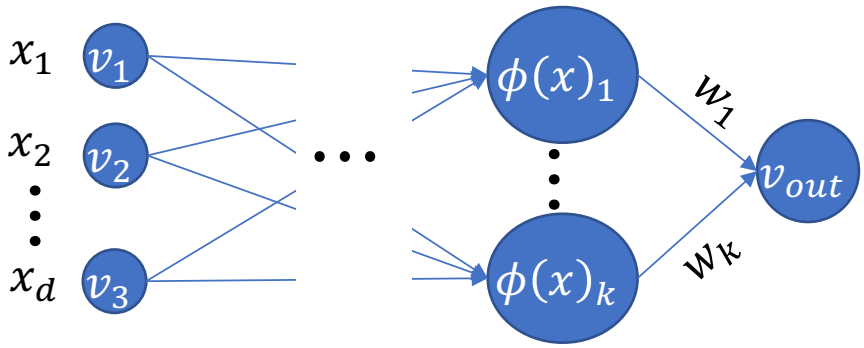


Deep learning

Generalize to hierarchy of transformations of the input, learned end-to-end jointly with the predictor.

$$f_{\boldsymbol{W}}(\boldsymbol{x}) = f_L\left(f_{L-1}\left(f_{L-2}(\dots f_1(\boldsymbol{x})\dots)\right)\right)$$

Neural Nets as Feature Learning



- Can think of hidden layer as "features" $\phi(x)$, then a linear predictor based on $\boldsymbol{w}.\phi(\boldsymbol{x})$
- "Feature Engineering" approach: design $\phi(\cdot)$ based on domain knowledge
- "Deep Learning" approach: learn features from data
- Multilayer networks with non-linear activations
  - more and more complex features

Slide credit: Nati Srebro

Multi-Layer Feature Learning

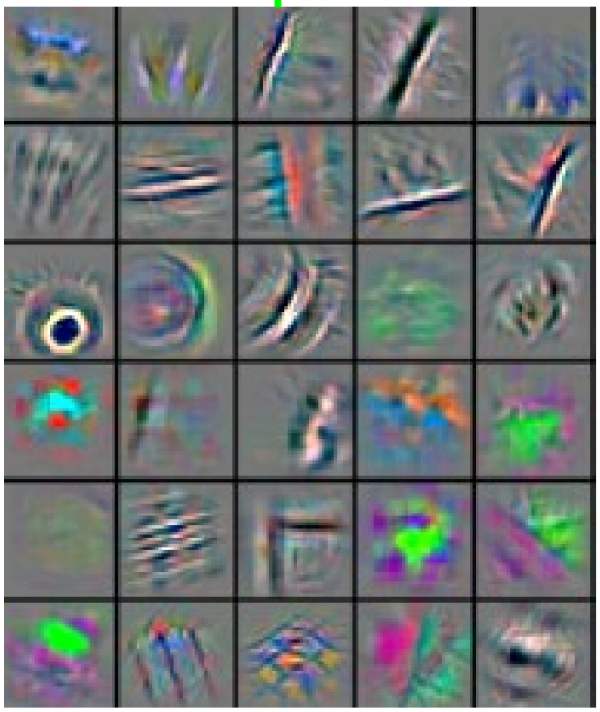
tion

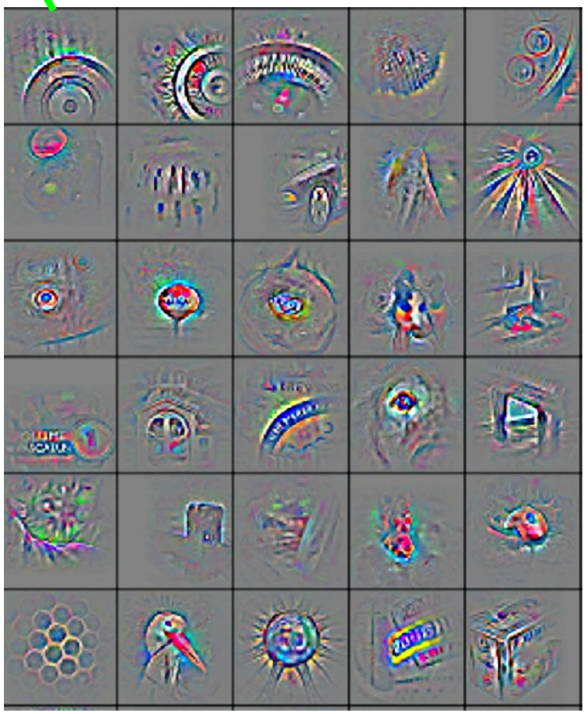
Low-Level Feature

Mid-Level Feature

High-Level Feature

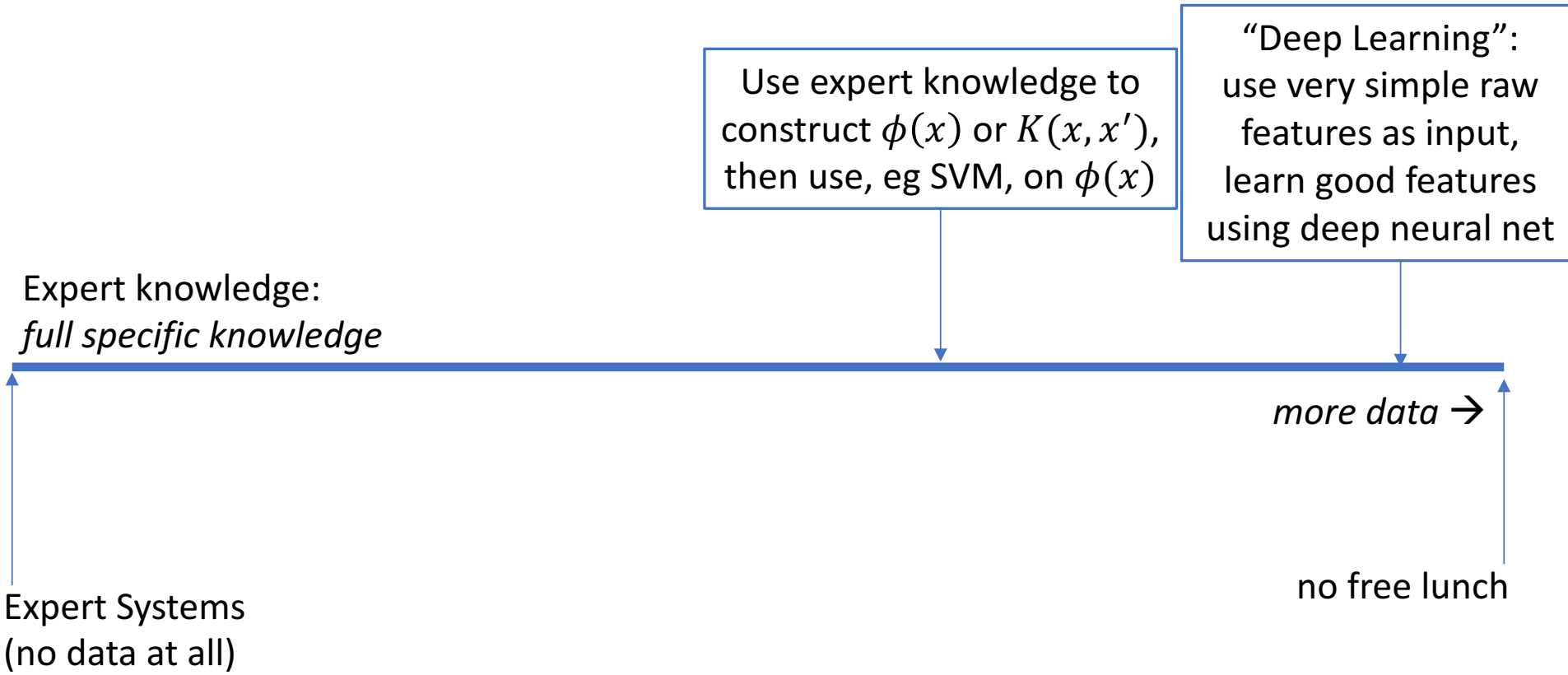
Trainable Classifier





Slide credit: Nati Srebro

More knowledge or more learning



Slide credit: Nati Srebro

Neural networks as hypothesis class

- Hypothesis class specified by:
  - Graph G(V,E)
  - Activation function σ

  } Based on architecture and fixed

  - Weights $\mathbf{W}$, with weight $\mathbf{W}[u \to v]$ for each edge $u \to v \in E$

$$\mathcal{H} = \left\{ f_{G(V,E),\sigma,\mathbf{W}} \mid \boldsymbol{W}: E \to \mathbb{R} \right\}$$

- Expressive power:

$$\{ f \mid f \text{ computable in time } T \} \subseteq \mathcal{H}_{G(V,E),sign} \quad \text{with } |E| = O(T^2)$$

- Computation: empirical risk minimization

$$\widehat{\mathbf{W}} = \arg\min_{W} \sum_{i=1}^{N} \ell\left(f_{G(V,E),\sigma,\boldsymbol{W}}\left(\boldsymbol{x}^{(i)}\right), y^{(i)}\right)$$

  - Highly non-convex problem, even if $loss\ \ell$ is convex
  - Hard to minimize over even tiny neural networks are hard

So how do we learn?

$$\widehat{\boldsymbol{W}} = \arg\min_{W} \sum_{i=1}^{N} \ell\left(f_{G(V,E),\sigma,\boldsymbol{W}}\left(\boldsymbol{x}^{(i)}\right), y^{(i)}\right)$$

- Stochastic gradient descent: for random $\left(\boldsymbol{x}^{(i)}, y^{(i)}\right) \in S$

  $$\boldsymbol{W}^{(t+1)} \leftarrow \boldsymbol{W}^{(t)} - \eta^{(t)} \nabla \ell\left(f_{G(V,E),\sigma,\boldsymbol{W}^{(t)}}\left(\boldsymbol{x}^{(i)}\right), y^{(i)}\right)$$

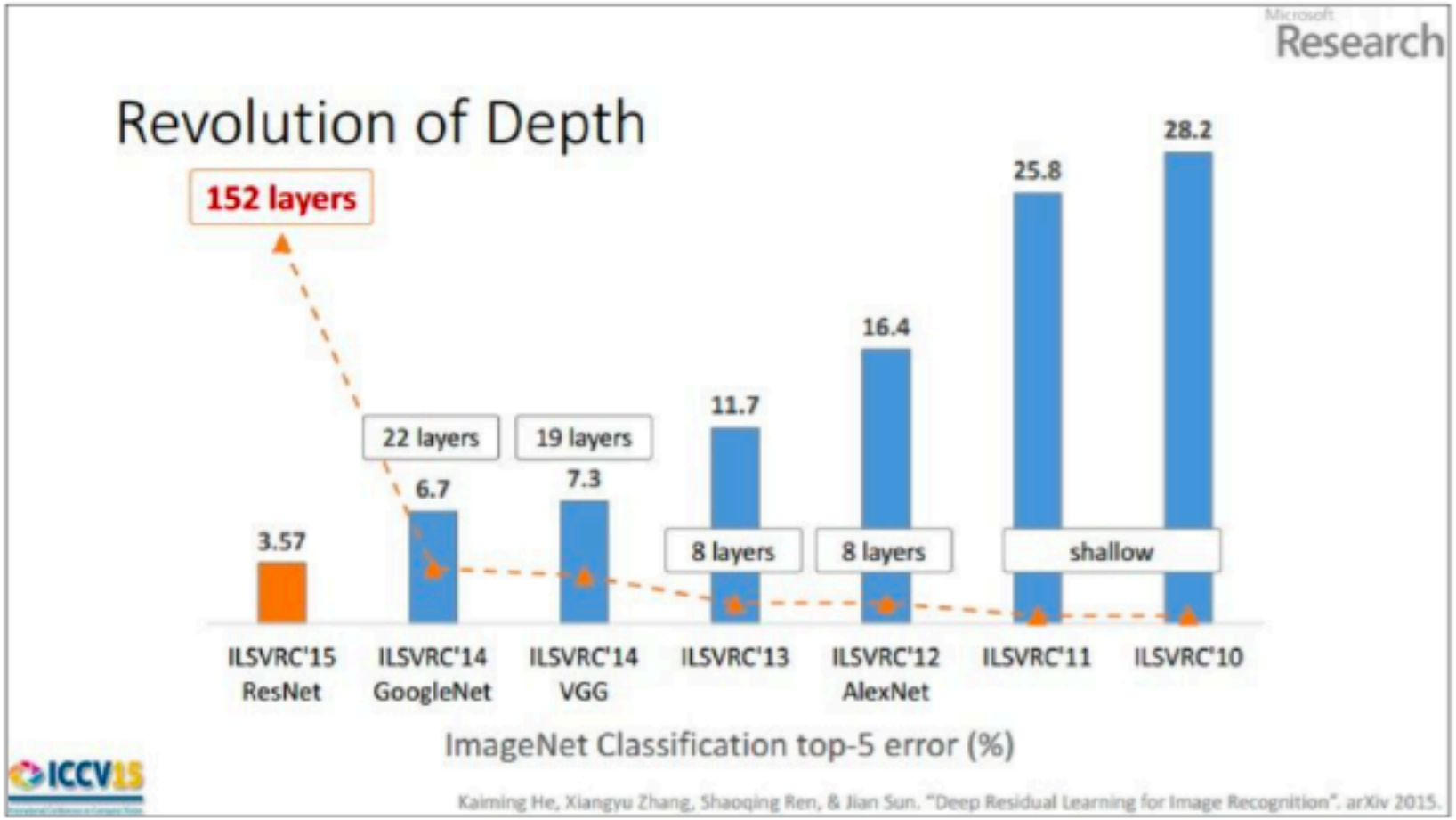
  (Even though its not convex)

- How do we efficiently calculate $\nabla \ell\left(f_{G(V,E),\sigma,\boldsymbol{W}^{(t)}}\left(\boldsymbol{x}^{(i)}\right), y^{(i)}\right)$?
  - Karl will tell you!

- Now a brief detour into history and resurrection of NNs

Imagenet challenge – object classification

1000 kinds of objects.



(slide from Kaiming He's recent presentation)

Object detection

PASCAL VOC Object Detection

	bicycle	bus	car	motorbike	person	20 class average
2007	36.9	23.2	34.6	27.6	21.3	17.1
2008	42.0	23.2	32.0	38.6	42.0	22.9
2009	46.8	43.8	37.2	42.0	41.5	27.9
2010	54.3	54.2	49.1	51.6	47.5	36.8
2011	58.1	57.6	54.4	58.3	51.6	40.9
2012	54.5	57.1	49.3	59.4	46.1	41.1
2013 DNN	56.3	51.4	48.7	59.8	44.4	43.2
2014 DNN						63.8
2015 ResNet	88.4	86.3	87.8	89.6	90.9	83.8
2016 ResNet						86

Slide credit: David McAllester

History of Neural Networks

- 1940s-70s:
  - Inspired by learning in the brain, and as a model for the brain (Pitts, Hebb, and others)
  - Various models, directed and undirected, different activation and learning rules
  - Perceptron Rule (Rosenblatt), Problem of XOR, Multilayer perceptron (Minksy and Papert)
  - Backpropagation (Werbos 1975)
- 1980s-early 1990s:
  - Practical Backprop (Rumelhart, Hinton et al 1986) and SGD (Bottou)
  - Relationship to distributed computing; “Connectionism”
  - Initial empirical success
- 1990s-2000s:
  - Lost favor to implicit linear methods: SVM, Boosting
- 2000-2010s:
  - revival of interest (CIFAR groups)
  - ca. 2005: layer-wise pretraining of deepish nets
  - progress in speech and vision with deep neural nets
- 2010s:
  - Computational advances allow training HUGE networks
  - …and also a few new tricks
  - Krizhevsky et al. win ImageNet
  - Empirical success and renewed interest

Deep learning - today

State of the art performance in several tasks and are actively deployed in real systems

- Computer vision
- Speech recognition
- Machine translation
- Dialog systems
- Computer games
- Information retrieval