Machine Learning (CSE 446): Neural Networks

Sham M Kakade

University of Washington
cse446-staff@cs.washington.edu

Classifiers We've Covered So Far

	decision boundary?	difficult part of learning?
decision trees	piecewise-axis-aligned	greedy split decisions
K-nearest neighbors	possibly very complex	indexing training data
perceptron	linear	iterative optimization method required
logistic regression	linear	iterative optimization method required
naïve Bayes	linear (see A4)	none

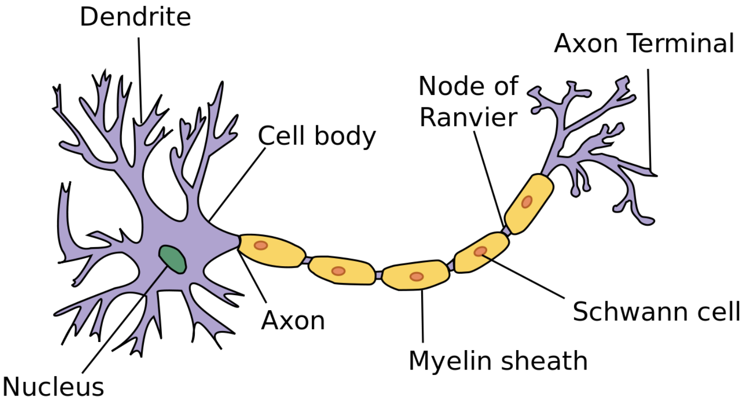
Classifiers We've Covered So Far

	decision boundary?	difficult part of learning?
decision trees	piecewise-axis-aligned	greedy split decisions
K-nearest neighbors	possibly very complex	indexing training data
perceptron	linear	iterative optimization method required
logistic regression	linear	iterative optimization method required
naïve Bayes	linear (see A4)	none

The next methods we'll cover permit **nonlinear** decision boundaries.

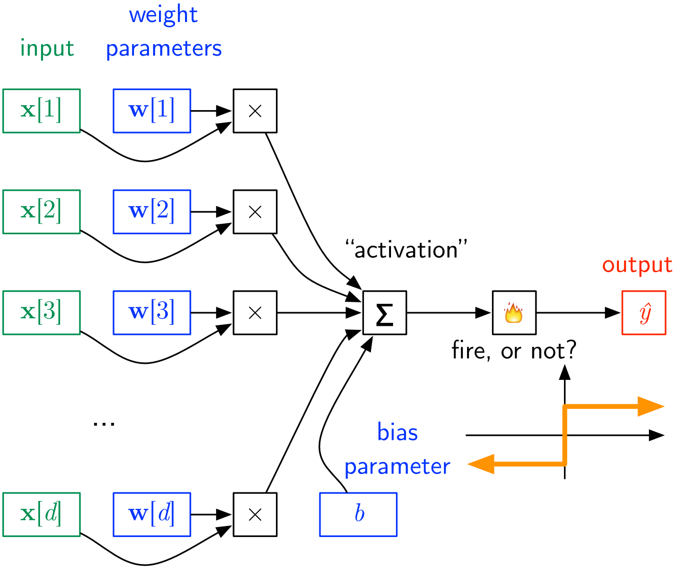
Inspiration from Neurons

Image from Wikimedia Commons.

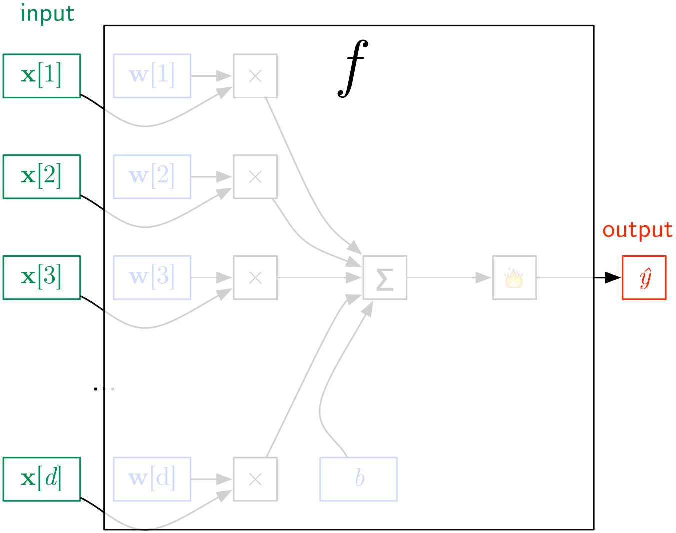


Input signals come in through dendrites, output signal passes out through the axon.

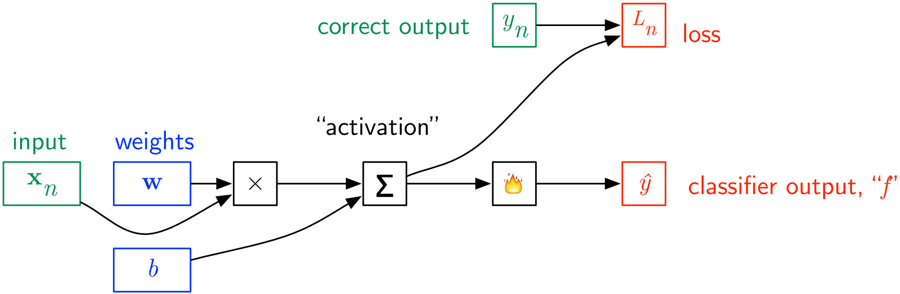
Neuron-Inspired Classifiers



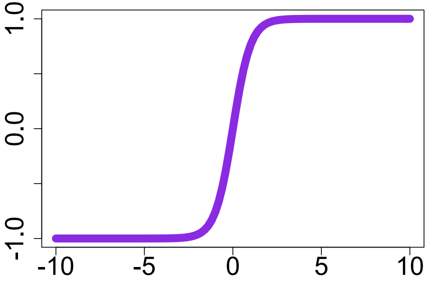
Neuron-Inspired Classifiers



Neuron-Inspired Classifiers



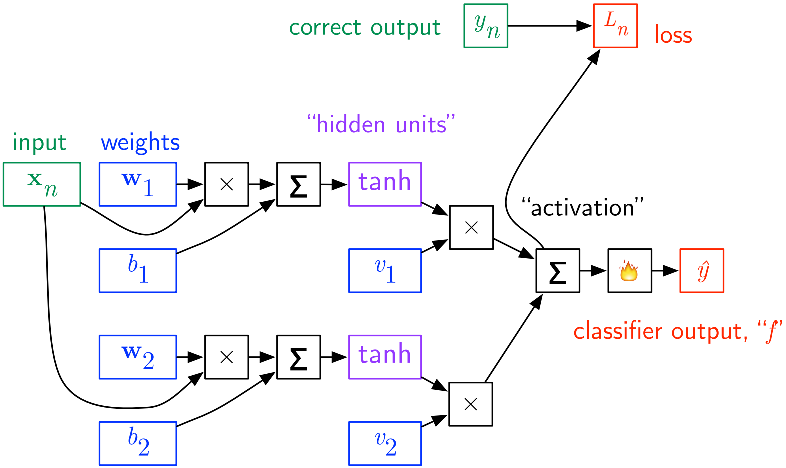
Neuron-Inspired Classifiers



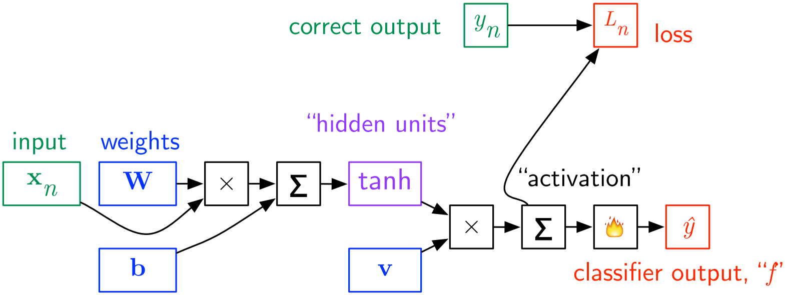
Hyperbolic tangent function, $\tanh(z) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}}$.

Generalization: apply elementwise to a vector, so that $\tanh : \mathbb{R}^k \to (-1, 1)^k$.

Neuron-Inspired Classifiers



Neuron-Inspired Classifiers



Two-Layer Neural Network

$$
\begin{aligned}
f(\mathbf{x}) &= \mathsf{sign}\left(\sum_{h=1}^{H} v_h \cdot \tanh\left(\mathbf{w}_h \cdot \mathbf{x} + b_h\right)\right) \\
&= \mathsf{sign}\left(\mathbf{v} \cdot \tanh\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right)\right)
\end{aligned}
$$

Two-Layer Neural Network

$$
\begin{aligned}
f(\mathbf{x}) &= \mathsf{sign}\left(\sum_{h=1}^{H} v_h \cdot \tanh\left(\mathbf{w}_h \cdot \mathbf{x} + b_h\right)\right) \\
&= \mathsf{sign}\left(\mathbf{v} \cdot \tanh\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right)\right)
\end{aligned}
$$

- Two-layer networks allow decision boundaries that are nonlinear.

Two-Layer Neural Network

$$
\begin{aligned}
f(\mathbf{x}) &= \mathsf{sign}\left(\sum_{h=1}^{H} v_h \cdot \tanh\left(\mathbf{w}_h \cdot \mathbf{x} + b_h\right)\right) \\
&= \mathsf{sign}\left(\mathbf{v} \cdot \tanh\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right)\right)
\end{aligned}
$$

- Two-layer networks allow decision boundaries that are nonlinear.
- It's fairly easy to show that "XOR" can be simulated (recall *conjunction* features from the "practical issues" lecture on 10/18).

Two-Layer Neural Network

$$
\begin{aligned}
f(\mathbf{x}) &= \mathsf{sign}\left(\sum_{h=1}^{H} v_h \cdot \tanh\left(\mathbf{w}_h \cdot \mathbf{x} + b_h\right)\right) \\
&= \mathsf{sign}\left(\mathbf{v} \cdot \tanh\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right)\right)
\end{aligned}
$$

- Two-layer networks allow decision boundaries that are nonlinear.
- It's fairly easy to show that "XOR" can be simulated (recall *conjunction* features from the "practical issues" lecture on 10/18).
- Theoretical result: any continuous function on a bounded region in $\mathbf{R}^d$ can be approximated arbitrarily well, with a finite number of hidden units.

Two-Layer Neural Network

$$
\begin{aligned}
f(\mathbf{x}) &= \mathsf{sign}\left(\sum_{h=1}^{H} v_h \cdot \tanh\left(\mathbf{w}_h \cdot \mathbf{x} + b_h\right)\right) \\
&= \mathsf{sign}\left(\mathbf{v} \cdot \tanh\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right)\right)
\end{aligned}
$$

- Two-layer networks allow decision boundaries that are nonlinear.
- It's fairly easy to show that "XOR" can be simulated (recall *conjunction* features from the "practical issues" lecture on 10/18).
- Theoretical result: any continuous function on a bounded region in $\mathbf{R}^d$ can be approximated arbitrarily well, with a finite number of hidden units.
- The number of hidden units affects how complicated your decision boundary can be and how easily you will overfit.

Learning with a Two-Layer Network

Parameters: $\mathbf{W} \in \mathbb{R}^{H \times d}$, $\mathbf{b} \in \mathbb{R}^{H}$, and $\mathbf{v} \in \mathbb{R}^{H}$

Learning with a Two-Layer Network

Parameters: $\mathbf{W} \in \mathbb{R}^{H \times d}$, $\mathbf{b} \in \mathbb{R}^{H}$, and $\mathbf{v} \in \mathbb{R}^{H}$

- If we choose a differentiable loss, then the the whole function will be differentiable with respect to all parameters.

Learning with a Two-Layer Network

Parameters: $\mathbf{W} \in \mathbb{R}^{H \times d}$, $\mathbf{b} \in \mathbb{R}^{H}$, and $\mathbf{v} \in \mathbb{R}^{H}$

- If we choose a differentiable loss, then the the whole function will be differentiable with respect to all parameters.
- Because of the squashing function, which is not convex, the overall learning problem is not convex.

Learning with a Two-Layer Network

Parameters: $\mathbf{W} \in \mathbb{R}^{H \times d}$, $\mathbf{b} \in \mathbb{R}^{H}$, and $\mathbf{v} \in \mathbb{R}^{H}$

- If we choose a differentiable loss, then the the whole function will be differentiable with respect to all parameters.
- Because of the squashing function, which is not convex, the overall learning problem is not convex.
- What does (stochastic) (sub)gradient descent do with non-convex functions?

Learning with a Two-Layer Network

Parameters: $\mathbf{W} \in \mathbb{R}^{H \times d}$, $\mathbf{b} \in \mathbb{R}^{H}$, and $\mathbf{v} \in \mathbb{R}^{H}$

- If we choose a differentiable loss, then the the whole function will be differentiable with respect to all parameters.
- Because of the squashing function, which is not convex, the overall learning problem is not convex.
- What does (stochastic) (sub)gradient descent do with non-convex functions? It finds a *local* minimum.
- To calculate gradients, we need to use the chain rule from calculus.

Learning with a Two-Layer Network

Parameters: $\mathbf{W} \in \mathbb{R}^{H \times d}$, $\mathbf{b} \in \mathbb{R}^{H}$, and $\mathbf{v} \in \mathbb{R}^{H}$

- If we choose a differentiable loss, then the the whole function will be differentiable with respect to all parameters.
- Because of the squashing function, which is not convex, the overall learning problem is not convex.
- What does (stochastic) (sub)gradient descent do with non-convex functions? It finds a *local* minimum.
- To calculate gradients, we need to use the chain rule from calculus.
- Special name for (S)GD with chain rule invocations: **backpropagation**.