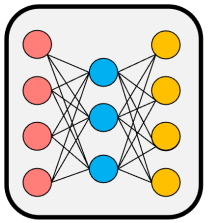
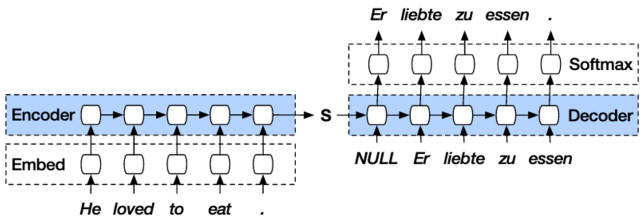
CS 446/ECE 449: Machine Learning

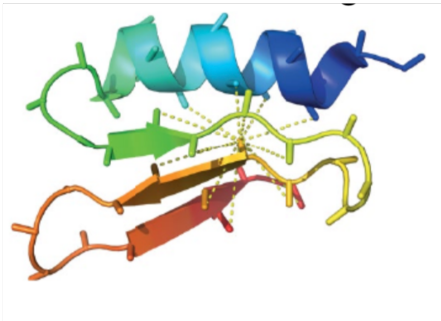
Shenlong Wang

University of Illinois at Urbana-Champaign, 2024

Deep Learning




Er liebte zu essen .
Softmax
Encoder
S
Decoder
Embed
NULL Er liebte zu essen
He loved to eat .




DEEP LEARNING
DEEP LEARNING
EVERYWHERE
memes.com

Overview of the next five lectures

- Single layer perceptron
- Multi-layer perceptron / Backpropagation
- Convolutional neural network
- Sequential modeling
- Transformers

Goals of this lecture

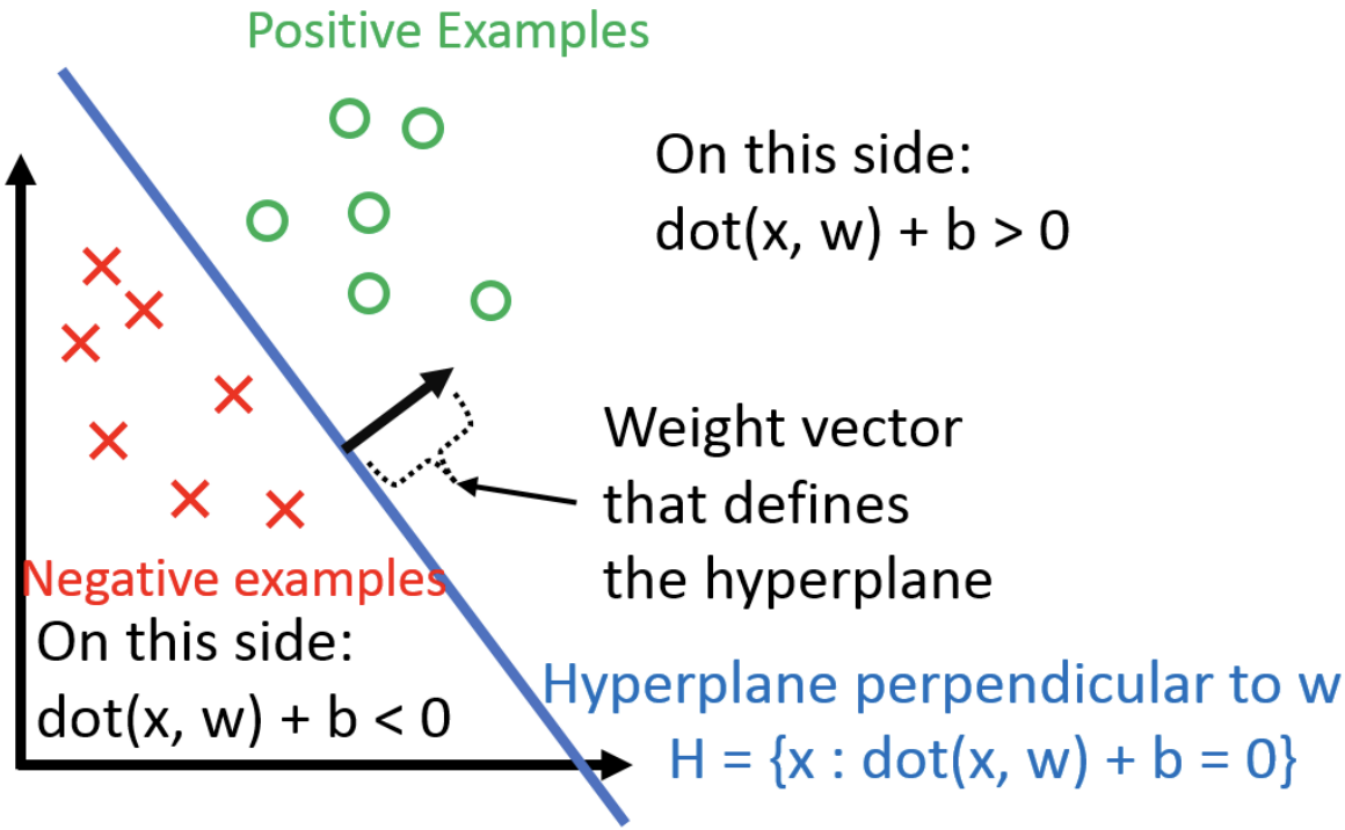
- Getting to know perceptron
- Getting to know deep nets
- Understanding forward pass
- Learning about deep net components

Reading material

- I. Goodfellow et al.; Deep Learning; Chapters 6-9

Perceptron

$$y = \text{sign}(\boldsymbol{w}^T \boldsymbol{x}) \text{ let's use } \boldsymbol{x} \leftarrow [\boldsymbol{x}, 1]$$



Note that

$$y_i(\boldsymbol{w}^T \boldsymbol{x}_i) > 0 \iff \boldsymbol{x}_i \text{ is classified correctly}$$

How can we learn?

Perceptron algorithm (Frank Rosenblatt, 1958):

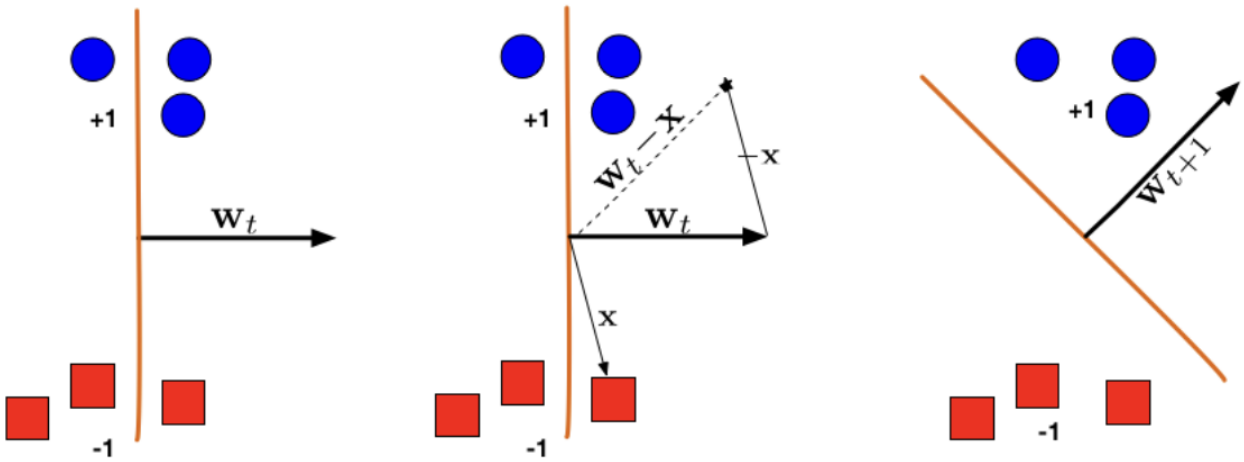
$$y = \text{sign}(\boldsymbol{w}^T \boldsymbol{x} + b)$$

Perceptron Algorithm

Initialize $\vec{w} = \vec{0}$ // Initialize $\vec{w}$. $\vec{w} = \vec{0}$ misclassifies everything.
while TRUE **do** // Keep looping
  $m = 0$ // Count the number of misclassifications, m
  for $(x_i, y_i) \in D$ **do** // Loop over each (data, label) pair in the dataset, D
    if $y_i(\vec{w}^T \cdot \vec{x_i}) \leq 0$ **then** // If the pair $(\vec{x_i}, y_i)$ is misclassified
      $\vec{w} \leftarrow \vec{w} + y\vec{x}$ // Update the weight vector $\vec{w}$
      $m \leftarrow m + 1$ // Counter the number of misclassification
    end if
  end for
  if $m = 0$ **then** // If the most recent $\vec{w}$ gave 0 misclassifications
    break // Break out of the while-loop
  end if
end while // Otherwise, keep looping!

What does a perceptron update do?

If $y(\boldsymbol{w}^T \boldsymbol{x}) \leq 0$, then $\boldsymbol{w}_{\text{new}} \leftarrow \boldsymbol{w} + y\boldsymbol{x}$

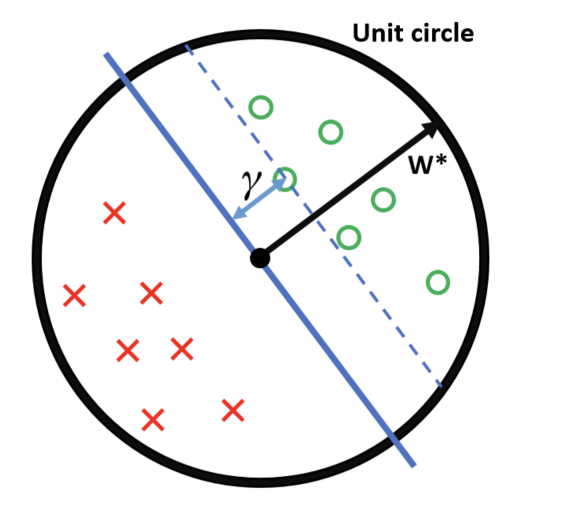


$$
\begin{aligned}
y(\boldsymbol{w}_{\text{new}}^T \boldsymbol{x}) &= y(\boldsymbol{w} + y\boldsymbol{x})^T \boldsymbol{x} \\
&= y(\boldsymbol{w}^T \boldsymbol{x}) + \|\boldsymbol{x}\|^2 \\
&> y(\boldsymbol{w}^T \boldsymbol{x})
\end{aligned}
$$

Quiz: Assume a data set consists only of a single data point $\{(\mathbf{x}, +1)\}$. How often can a perceptron misclassify this point $\mathbf{x}$ repeatedly?

$$y(\boldsymbol{w} + ky\boldsymbol{x})^T \boldsymbol{x} \leq 0 \rightarrow k \leq -y(\boldsymbol{w}^T \boldsymbol{x})/\|\boldsymbol{x}\|^2$$

Perceptron Convergence



Assumption:

- All inputs $\mathbf{x}_i$ live within the unit sphere. (after rescaling)
- There exists a separating hyperplane defined by $\mathbf{w}^*$, with $\|\mathbf{w}\|^* = 1$ (i.e. $\mathbf{w}^*$ lies exactly on the unit sphere.)
- γ is linear separation margin.

Theorem: If all of the above holds, the perceptron algorithm makes at most $1/\gamma^2$ updates.

Perceptron Convergence

Theorem: If all of the above holds, the perceptron algorithm makes at most $1/\gamma^2$ updates.

Proof: Consider the effect of an update:

If $y(\boldsymbol{w}^T\boldsymbol{x}) \leq 0$, then $\boldsymbol{w}_{\text{new}} \leftarrow \boldsymbol{w} + y\boldsymbol{x}$, we have:

- $y(\mathbf{x}^\top \mathbf{w}) \leq 0$: **x** is misclassified by **w** - otherwise wouldn't update.
- $y(\mathbf{x}^\top \mathbf{w}^*) \geq 0$: $\mathbf{w}^*$ is a separating hyper-plane

(i) Consider the effect of an update on $\mathbf{w}^\top \mathbf{w}^*$:

- $(\mathbf{w} + y\mathbf{x})^\top \mathbf{w}^* = \mathbf{w}^\top \mathbf{w}^* + y(\mathbf{x}^\top \mathbf{w}^*) \geq \mathbf{w}^\top \mathbf{w}^* + \gamma$
- This means: $\mathbf{w}^\top \mathbf{w}^*$ grows by at least γ.

(ii) Consider the effect of an update on $\mathbf{w}^\top \mathbf{w}$:

- $(\mathbf{w} + y\mathbf{x})^\top (\mathbf{w} + y\mathbf{x}) = \mathbf{w}^\top \mathbf{w} + \underbrace{2y(\mathbf{w}^\top \mathbf{x})}_{<0} + \underbrace{y^2(\mathbf{x}^\top \mathbf{x})}_{0 \leq \ \leq 1} \leq \mathbf{w}^\top \mathbf{w} + 1$
- This means: $\mathbf{w}^\top \mathbf{w}$ grows by at most 1.

Perceptron Convergence

Now recall that we initialize $\mathbf{w} = \mathbf{0}$. Hence, initially $\mathbf{w}^\top \mathbf{w} = 0$ and $\mathbf{w}^\top \mathbf{w}^* = 0$ and combining (i) and (ii), after M updates the following two inequalities must hold:

- $\mathbf{w}^\top \mathbf{w}^* \geq M\gamma$ (1)
- $\mathbf{w}^\top \mathbf{w} \leq M$. (2)

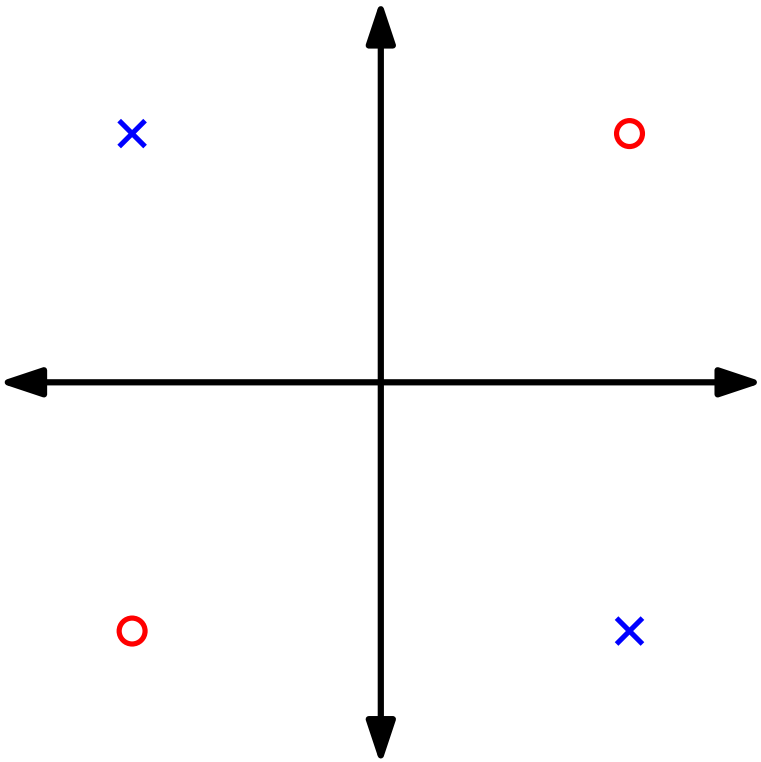
We can then complete the proof:

$$
\begin{aligned}
M\gamma &\leq \mathbf{w}^\top \mathbf{w}^* && \text{By (1)} \\
&= \|\mathbf{w}\| \cos(\theta) && \text{by definition of inner-product} \\
&\leq ||\mathbf{w}|| && \text{we must have } \cos(\theta) \leq 1. \\
&= \sqrt{\mathbf{w}^\top \mathbf{w}} && \text{by definition of } \|\mathbf{w}\| \\
&\leq \sqrt{M} && \text{By (2)}
\end{aligned}
$$

$$
\Rightarrow M\gamma \leq \sqrt{M} \Rightarrow M \leq \frac{1}{\gamma^2} \quad \text{Hence, } M \text{ is bounded from above.}
$$

A little bit history

- Initially, huge wave of excitement ("Digital brains") (See The New Yorker December 1958)
- Then, contributed to the first A.I. Winter. Famous example of a simple non-linearly separable data set, the XOR problem (Minsky 1969):



No linear separator classifies perfectly!

Neural networks via *features.*

To make a linear predictor nonlinear, we rely upon feature mapping ϕ:

$$\boldsymbol{x} \mapsto \boldsymbol{w}^\top \boldsymbol{x} \qquad \text{becomes} \qquad \boldsymbol{x} \mapsto \boldsymbol{w}^\top \phi(\boldsymbol{x}).$$

We are at the mercy of the quality of ϕ.

Why not *learn* ϕ? e.g.,

$$\arg\min_{\boldsymbol{w}} \frac{1}{n} \sum_{i=1}^{n} \ell\left(y^{(i)} \boldsymbol{w}^\top \mathbf{x}^{(i)}\right) \quad \text{becomes} \quad \arg\min_{\boldsymbol{w}, \phi \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \ell\left(y^{(i)} \boldsymbol{w}^\top \phi(\mathbf{x}^{(i)})\right)$$

where $\mathcal{F}$ is some class of functions **(why not every function?)**.

Neural networks as iterated linear prediction (part 1).

Natural choice: build feature maps out of linear predictors!

$$\boldsymbol{x} \mapsto \boldsymbol{w}^\top \boldsymbol{x} \qquad \text{becomes} \qquad \boldsymbol{x} \mapsto \boldsymbol{v}^\top \phi(\boldsymbol{x}) \quad \text{where } \phi(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x} + \mathbf{b}$$

with $\boldsymbol{w} \in \mathbb{R}^d, \boldsymbol{v} \in \mathbb{R}^m, \boldsymbol{A} \in \mathbb{R}^{m \times d}, \mathbf{b} \in \mathbb{R}^m$.

There is something wrong with this!
Gained nothing! $\boldsymbol{v}^\top(\boldsymbol{A}\boldsymbol{x} + \mathbf{b}) = (\boldsymbol{A}^\top \boldsymbol{v})^\top \boldsymbol{x} + \boldsymbol{v}^\top \mathbf{b}$.

Fix: introduce **nonlinearity/transfer/activation** $\sigma : \mathbb{R}^m \to \mathbb{R}^m$:

$$\phi(\boldsymbol{x}) := \sigma\left(\boldsymbol{A}\boldsymbol{x} + \mathbf{b}\right).$$

Neural networks as iterated linear prediction (part 2).

We will predict with

$$\boldsymbol{x} \mapsto \boldsymbol{w}^\top \phi(\boldsymbol{x}) \qquad \text{where } \phi(\boldsymbol{x}) = \sigma\left(\boldsymbol{A}\boldsymbol{x} + \mathbf{b}\right).$$

We will train with

$$\underset{\boldsymbol{w}\in\mathbb{R}^m, \boldsymbol{A}\in\mathbb{R}^{m\times d}, \mathbf{b}\in\mathbb{R}^m}{\arg\min} \frac{1}{n}\sum_{i=1}^{n} \ell\left(y^{(i)}\boldsymbol{w}^\top \sigma\left(\boldsymbol{A}\mathbf{x}^{(i)} + \mathbf{b}\right)\right).$$

(**Question:** which training procedure? Why does it work?)

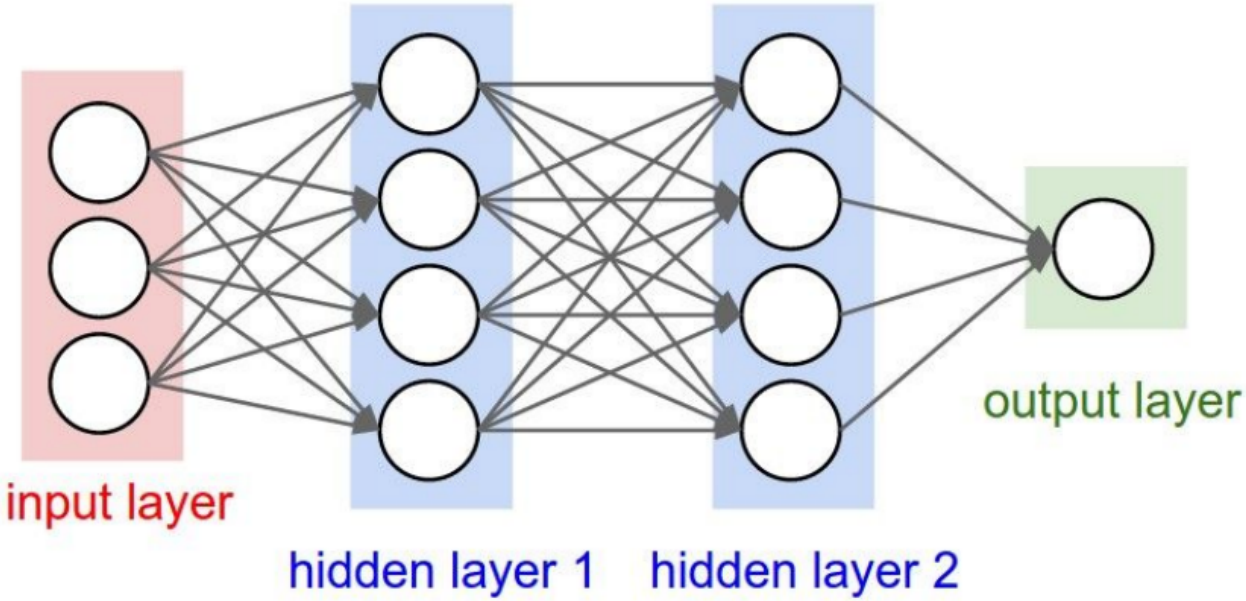
Why stop there? We can also do

$$\boldsymbol{x} \mapsto \boldsymbol{w}^\top \sigma_1\left(\boldsymbol{A}_1 \phi(\boldsymbol{x}) + \mathbf{b}_1\right) \qquad \text{where } \phi(\boldsymbol{x}) = \sigma_2\left(\boldsymbol{A}_2\boldsymbol{x} + \mathbf{b}_2\right),$$

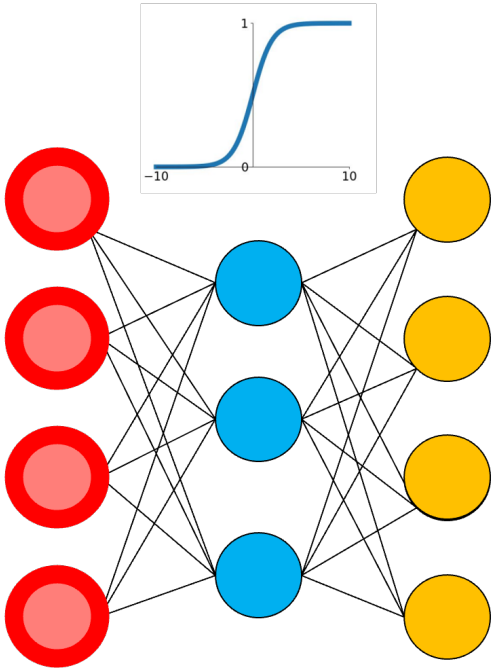
and iterate further. *This is a neural network.*

Multi-layer Perceptron

$$\boldsymbol{x} \mapsto \boldsymbol{w}^\top \sigma_1 \left(\boldsymbol{A}_1 \phi(\boldsymbol{x}) + \mathbf{b}_1 \right) \qquad \text{where } \phi(\boldsymbol{x}) = \sigma_2 \left(\boldsymbol{A}_2 \boldsymbol{x} + \mathbf{b}_2 \right),$$

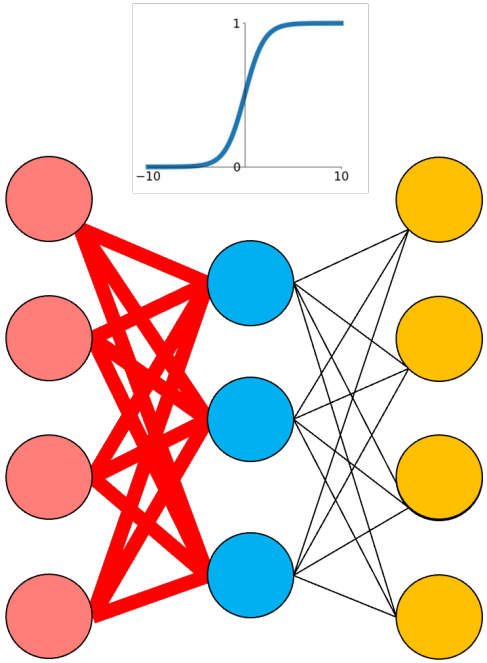


Forward pass



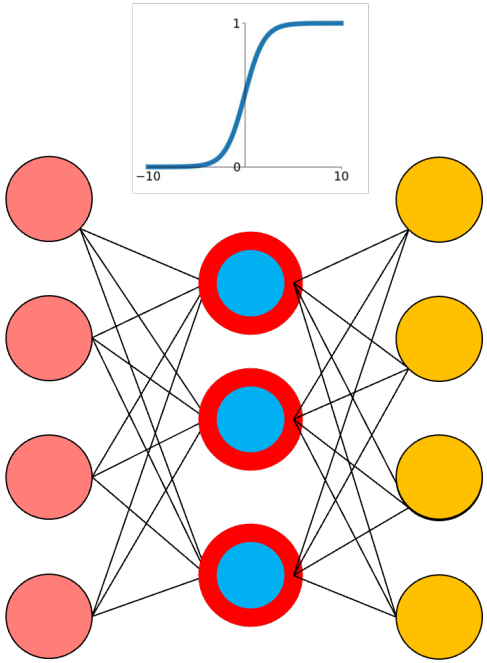
$$y = W_2\sigma(W_1x+b_1)+b_2$$

Forward pass



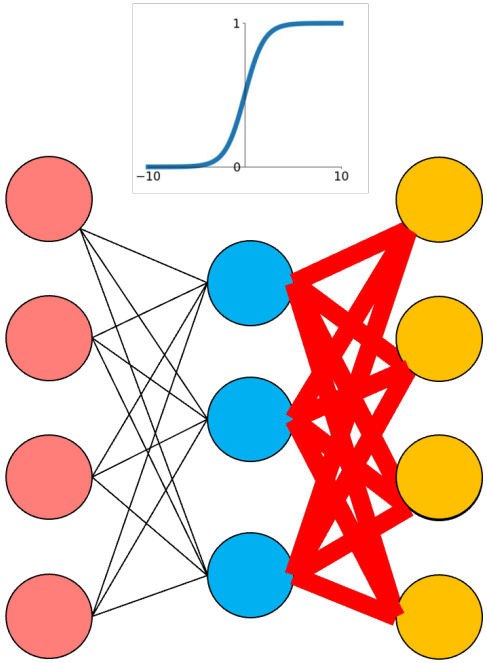
$$y = W_2\sigma(W_1x+b_1)+b_2$$

Forward pass



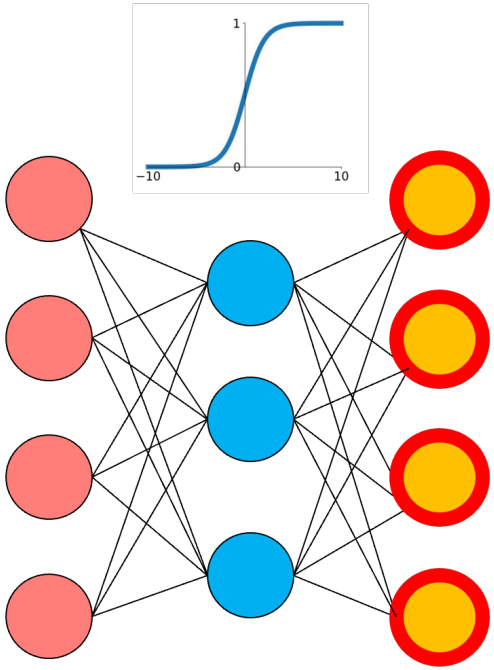
$$y = W_2\sigma(W_1x+b_1)+b_2$$

Forward pass



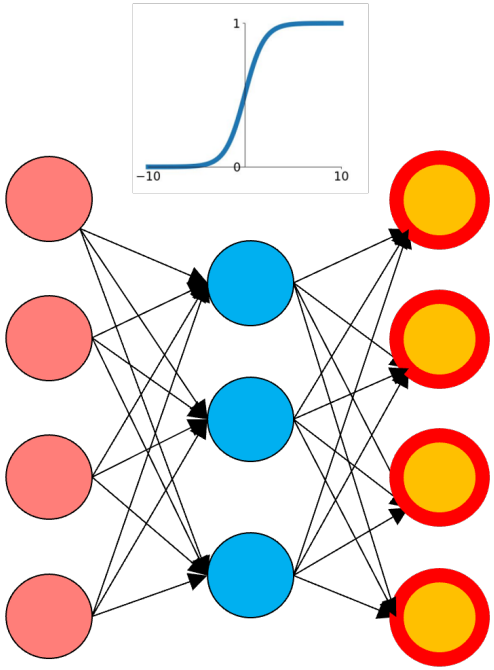
$$y = W_2\sigma(W_1x+b_1)+b_2$$

Forward pass



$$y = W_2\sigma(W_1x+b_1)+b_2$$

Forward pass



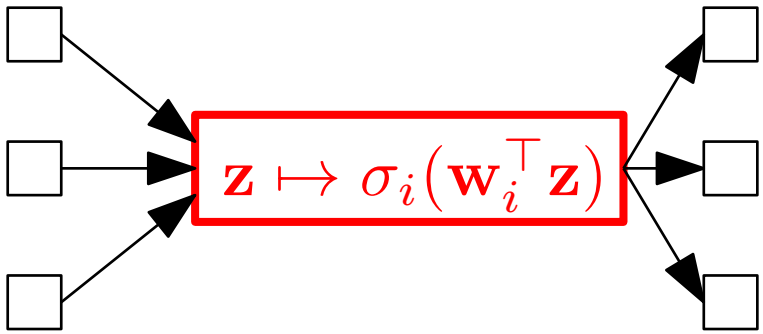
$$y = W_2\sigma(W_1x+b_1)+b_2$$

Classical formulation of neural networks as graphs.

("Classical" because tensorflow "computation graphs" differ slightly.)

Node j in this graph:

- Collects a vector $\boldsymbol{z}$ from its in-edges;
- Computes $\boldsymbol{z} \mapsto \sigma_j(\boldsymbol{w}_j^\top \boldsymbol{z} + b_j)$;
- Propagates this value along its out-edges.



Computation of whole network can be written this way.

Tensorflow computation graphs: everything needed to train is in the graph; e.g., parameters get nodes.

Neural networks as functions.

A linear predictor (**one layer network**) has the form

$$\boldsymbol{x} \mapsto \boldsymbol{w}^\top \boldsymbol{x}.$$

A **two layer network** has the form

$$\boldsymbol{x} \mapsto \boldsymbol{w}^\top \sigma_1 \left(\boldsymbol{A}_1 \boldsymbol{x} + \mathbf{b}_1 \right).$$

Iterating, a **multi-layer network** has the form

$$\boldsymbol{x} \mapsto \boldsymbol{w}^\top \sigma_1 \left(\boldsymbol{A}_1 \sigma_2 \left(\cdots \boldsymbol{A}_{L-2} \sigma_{L-1} \left(\boldsymbol{A}_{L-1} \boldsymbol{x} + \mathbf{b}_{L-1} \right) + \mathbf{b}_{L-2} \cdots \right) + \mathbf{b}_1 \right).$$

ERM now takes the form

$$\underset{\boldsymbol{w}, \boldsymbol{A}_1, \ldots, \boldsymbol{A}_{L-1}, \mathbf{b}_1, \ldots, \mathbf{b}_{L-1}}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} \ell \left(y^{(i)} \boldsymbol{w}^\top \sigma_1 \left(\cdots \sigma_{L-1} \left(\boldsymbol{A}_{L-1} \mathbf{x}^{(i)} + \mathbf{b}_{L-1} \right) \cdots \right) \right).$$

Neural network (univariate) activations.

We mentioned that **nodes** compute

$$\boldsymbol{z} \mapsto \sigma\left(\boldsymbol{v}^\top \boldsymbol{z}\right),$$

where *activation/transfer/nonlinearity* $\sigma : \mathbb{R} \to \mathbb{R}$ is:

- ReLU (Rectified Linear Unit) $z \mapsto \max\{0, z\}$;
- Sigmoid $z \mapsto \frac{1}{1+\exp(-z)}$;
- ....

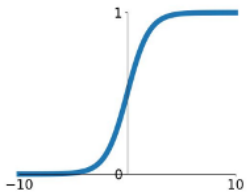
By $\boldsymbol{z} \mapsto \sigma(\boldsymbol{A}\boldsymbol{z} + \mathbf{b})$,
we meant "apply a univariate σ coordinate-wise".

Soon we will see multivariate nonlinearities,
sometimes with output dimension $\neq$ input dimension!

Nonlinearity.

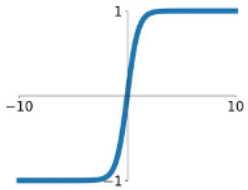
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



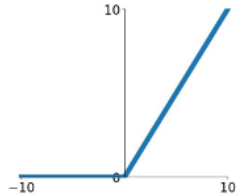
tanh

$$\tanh(x)$$



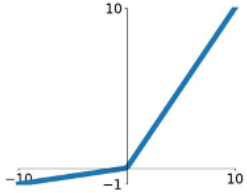
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

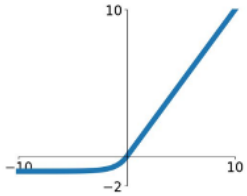


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Some modern usage.

Multiclass output.

Modern networks often end with **softmax** nonlinearity:

$$\boldsymbol{z} \mapsto \sum_{i=1}^{k} \frac{\exp(\boldsymbol{z}_i)\mathbf{e}_i}{\sum_{j=1}^{k} \exp(\boldsymbol{z}_j)}$$

(where $\mathbf{e}_i$ is i^{th} standard basis vector.)
Output is now a probability vector!

Alternate notation: output vector $\boldsymbol{v}_i \propto \exp(\boldsymbol{z}_i)$.

Cross-entropy loss.

Given *one hot* $\boldsymbol{y} \in \{\mathbf{e}_1, \ldots, \mathbf{e}_k\}$ and probability vector $\hat{\boldsymbol{y}} \in \mathbb{R}^k$,

$$\ell(y, \hat{y}) = -\sum_{i=1}^{k} \boldsymbol{y}_i \ln(\hat{\boldsymbol{y}}).$$

Combined with softmax $\hat{\boldsymbol{y}} \propto \exp(\boldsymbol{z})$:

$$-\sum_{i=1}^{k} \boldsymbol{y}_i \ln\left(\frac{\exp(\boldsymbol{z}_i)}{\sum_j \exp(\boldsymbol{z}_j)}\right) = -\sum_{i=1}^{k} \boldsymbol{y}_i \boldsymbol{z}_i + \ln\left(\sum_{i=1}^{k} \exp(\boldsymbol{z}_i)\right).$$

(For numerical stability, use $\ln \sum_i \exp v_i = c + \ln \sum_i \exp(v_i - c)$.)

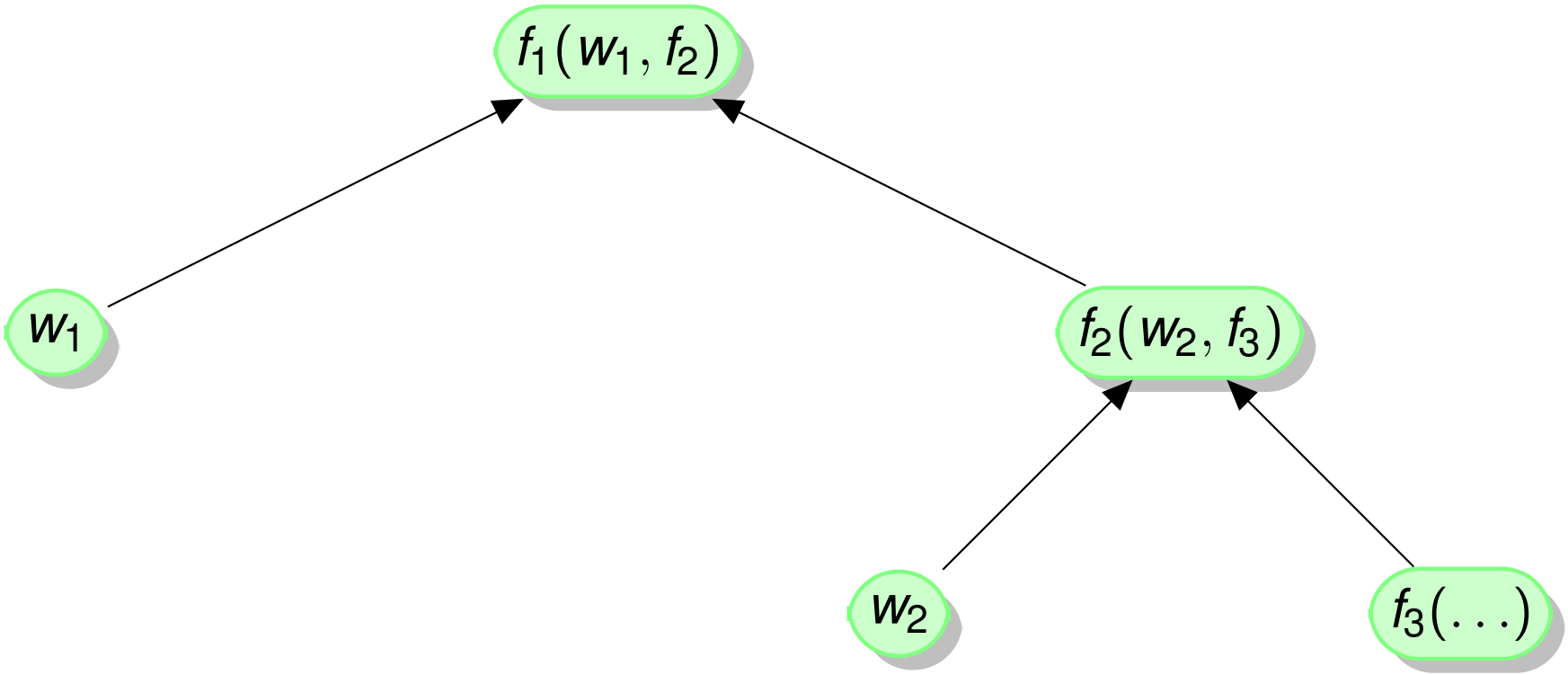
(In binary case $k = 2$: generalizes logistic loss.)

Question: since last expression is convex in $\boldsymbol{z}$,
is the corresponding ERM problem convex?

Example:

$$F(\boldsymbol{w}, x, y) = f_1(w_1, f_2(w_2, f_3(\ldots)))$$

Nodes are weights, data, and functions:



Internal representation used by deep net packages.

Example (PyTorch.py):

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(16, 120) % specify shape, init params
        self.fc2 = nn.Linear(120, 84) % specify shape, init params
        self.fc3 = nn.Linear(84, 10) % specify shape, init params

    def forward(self, x):
        x = F.relu(self.fc1(x)) % execute relu(w1*x + b1)
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

What are the input dimensions?

Quiz:

- What are deep nets?
- How do deep nets relate do SVMs and logistic regression
- What components of deep nets do you know?

Important topics of this lecture

- Perceptron
- Multi-layer perceptron
- Components of deep nets

Up next:

- Backpropagation