27: Deep Learning II

Lisa Yan

November 22, 2019

Logistic Regression Model

$$\hat{Y} = \arg\max_{y=\{0,1\}} P(Y \mid \boldsymbol{X})$$

Predict the Y that is most likely given our observation $\boldsymbol{X}$

where $$P(Y = 1|\boldsymbol{X} = \boldsymbol{x}) = \sigma\left(\theta_0 + \sum_{j=1}^{m} \theta_j x_j\right)$$

models $P(Y \mid \boldsymbol{X})$ directly

sigmoid fn
$\sigma(z) = 1/(1 + e^{-z})$

Logistic Regression Model

Review

Training

Learn parameters $\theta = (\theta_0, \theta_1, \ldots, \theta_m)$

via gradient ascent + MLE: $\theta_j^{\text{new}} = \theta_j^{\text{old}} + \eta \cdot \sum_{i=1}^{n} \left[y^{(i)} - \sigma\left(\theta^{\text{old}^T} \boldsymbol{x}^{(\boldsymbol{i})} \right) \right] x_j^{(i)}$

Testing

For input feature vector $\boldsymbol{X} = \boldsymbol{x}$:

- Use parameters to compute
$$\hat{y} = P(Y = 1 | \boldsymbol{X} = \boldsymbol{x}) = \sigma(\theta^T \boldsymbol{x})$$
- If $\hat{y} > 0.5$, predict 1. Else, predict 0.

👉 ⚠️ Parameters θ_j are **not** updated during testing phase

Training: Logistic Regression via MLE

1. Logistic Regression model: $P(Y=1|\boldsymbol{X}=\boldsymbol{x}) = \sigma(\theta^T \boldsymbol{x})$

2. Compute log-likelihood of training data:
$$LL(\theta) = \sum_{i=1}^{n} y^{(i)} \log \sigma\left(\theta^T \boldsymbol{x}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - \sigma\left(\theta^T \boldsymbol{x}^{(i)}\right)\right)$$

3. Compute derivative of log-likelihood with respect to each $\theta_j, j = 0, 1, \dots, m$:
$$\frac{\partial LL(\theta)}{\partial \theta_j} = \sum_{i=1}^{n} \left[y^{(i)} - \sigma\left(\theta^T \boldsymbol{x}^{(i)}\right)\right] x_j^{(i)}$$

4. Optimize

Gradient ascent

How we computed gradient for Logistic Regression

Review

Log-likelihood: $$LL(\theta) = \sum_{i=1}^{n} \underbrace{y^{(i)} \log \hat{y}^{(i)} + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)}_{\text{Call this } LL(\theta)^{(i)}\text{, contribution from } i\text{-th datapoint}}$$

Let $\hat{y} = P(Y = 1 | \boldsymbol{X} = \boldsymbol{x}) = \sigma(\theta^T \boldsymbol{x})$

$$\frac{\partial \hat{y}}{\partial \theta_j} = \hat{y}[1 - \hat{y}]x_j \quad = \sigma(\theta^T \boldsymbol{x})[1 - \sigma(\theta^T \boldsymbol{x})]x_j$$

sigmoid derivative

$$\frac{\partial LL(\theta)}{\partial \theta_j} = \sum_{i}^{n} \frac{\partial LL(\theta)^{(i)}}{\partial \theta_j}$$

sum of derivatives

$$\frac{\partial LL(\theta)^{(i)}}{\partial \theta_j} = \frac{\partial}{\partial \hat{y}^{(i)}} \left[LL(\theta)^{(i)}\right] \cdot \frac{\partial \hat{y}^{(i)}}{\partial \theta_j}$$

chain rule

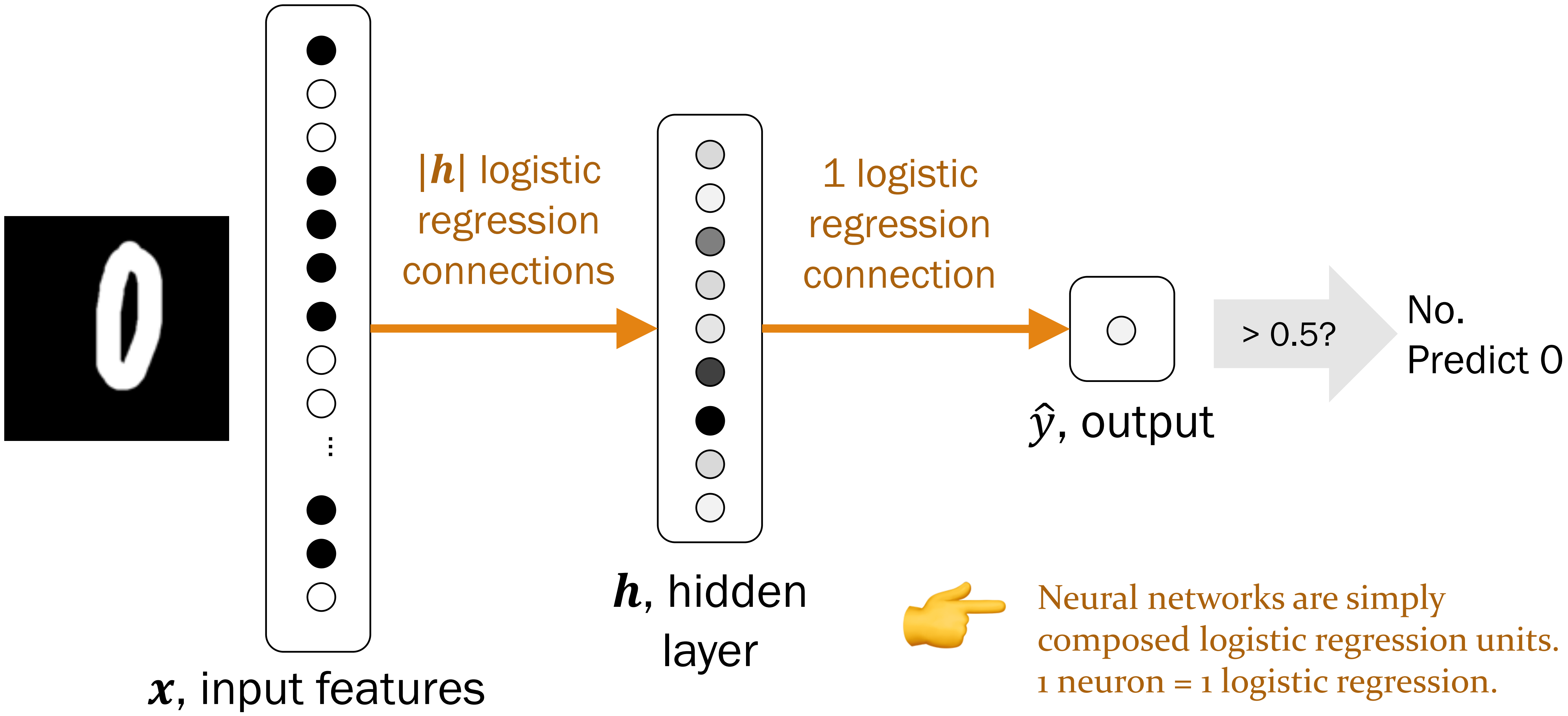
Today's plan

Intro to Deep Learning

- Parameters of a neural network
- Training neural networks
- Extra ideas

Neural networks

Review



👉 Neural networks are simply composed logistic regression units. 1 neuron = 1 logistic regression.

Neural networks

Review

A neural network (like logistic regression) gets intelligence from its parameters θ.

Training

- Learn parameters θ
- Find θ_{MLE} that maximizes likelihood of training data (MLE)

Testing/ Prediction

For input feature vector $\boldsymbol{X} = \boldsymbol{x}$:

- Use parameters to compute $\hat{y} = P(Y = 1|\boldsymbol{X} = \boldsymbol{x})$
- If $\hat{y} > 0.5$, predict 1. Else, predict 0.

Predict: **Forward Pass**

A neural network (like logistic regression) gets intelligence from its parameters θ.

Training

- Learn parameters θ
- Find θ_{MLE} that maximizes likelihood of training data (MLE)

Testing/ Prediction

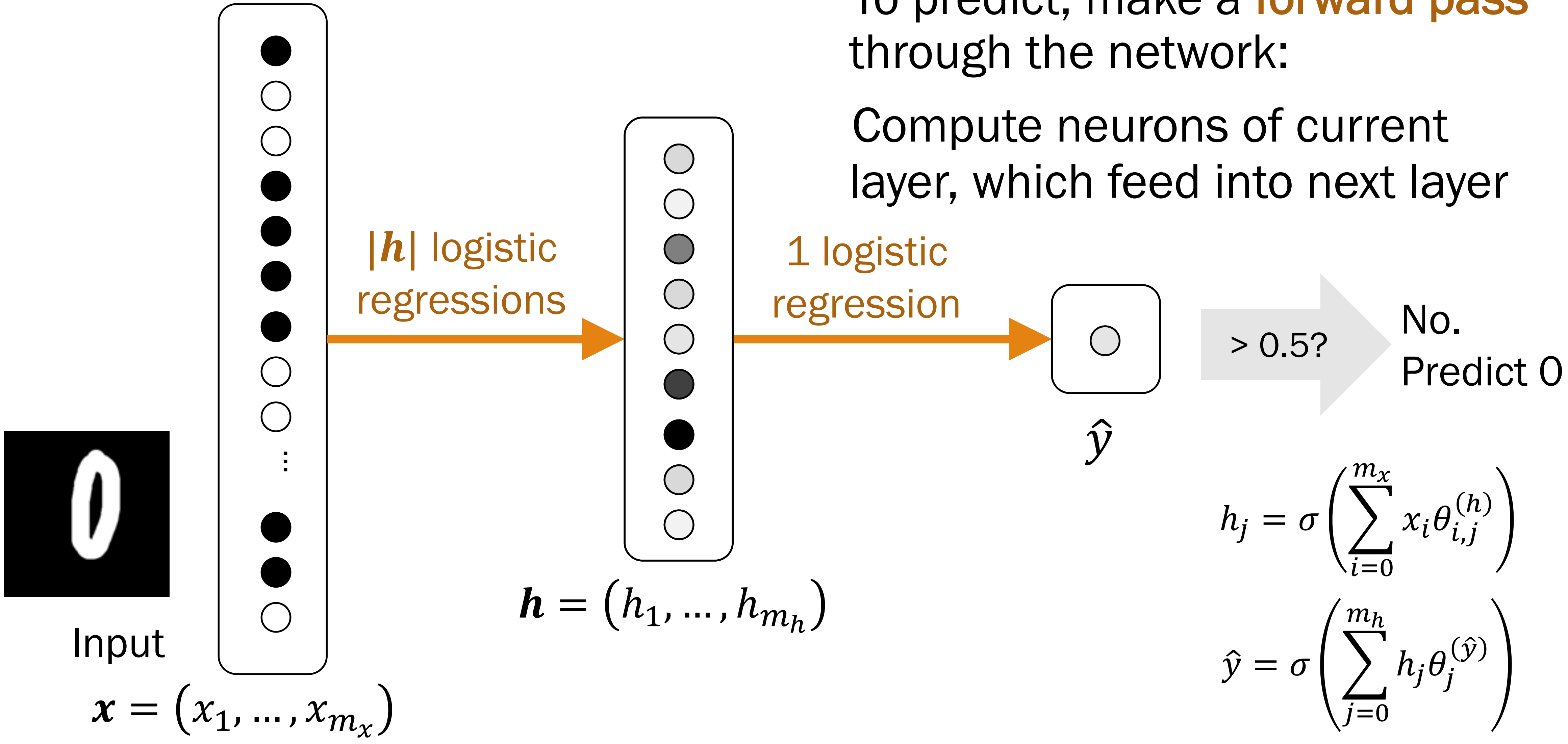
For input feature vector $\boldsymbol{X} = \boldsymbol{x}$:

- Use parameters to compute $\hat{y} = P(Y = 1|\boldsymbol{X} = \boldsymbol{x})$
- If $\hat{y} > 0.5$, predict 1. Else, predict 0.

Predict: **Forward Pass**

To predict, make a **forward pass** through the network:

Compute neurons of current layer, which feed into next layer

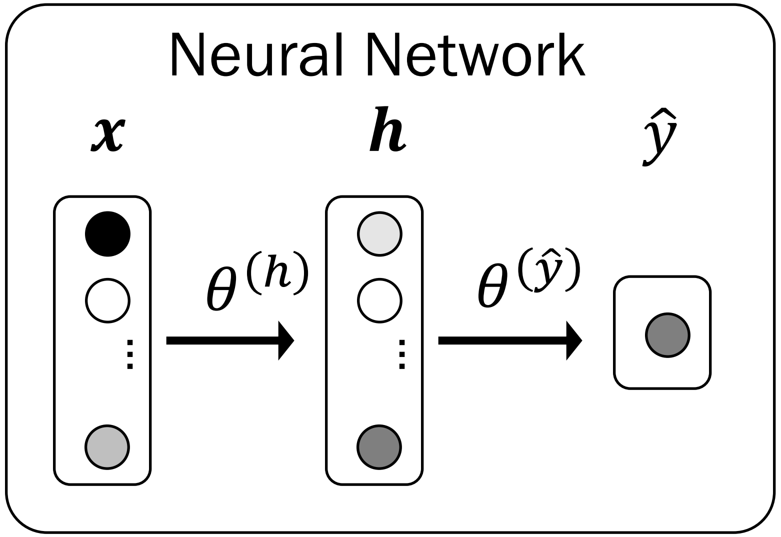


$$h_j = \sigma\left(\sum_{i=0}^{m_x} x_i \theta_{i,j}^{(h)}\right)$$

$$\hat{y} = \sigma\left(\sum_{j=0}^{m_h} h_j \theta_j^{(\hat{y})}\right)$$

Neural network model

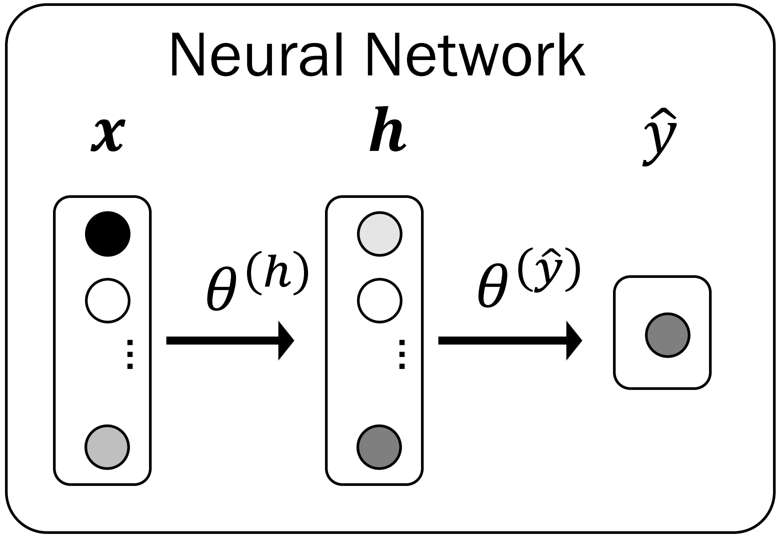
Review



$$h_j = \sigma\left(\sum_{i=0}^{m_x} x_i \theta_{i,j}^{(h)}\right) \qquad \hat{y} = \sigma\left(\sum_{j=0}^{m_h} h_j \theta_j^{(\hat{y})}\right)$$

$$\hat{y} = P(Y = 1 | \boldsymbol{X} = \boldsymbol{x})$$

Quick check



$$h_j = \sigma\left(\sum_{i=0}^{m_x} x_i \theta_{i,j}^{(h)}\right) \qquad \hat{y} = \sigma\left(\sum_{j=0}^{m_h} h_j \theta_j^{(\hat{y})}\right)$$

$$\hat{y} = P(Y = 1 | \boldsymbol{X} = \boldsymbol{x})$$

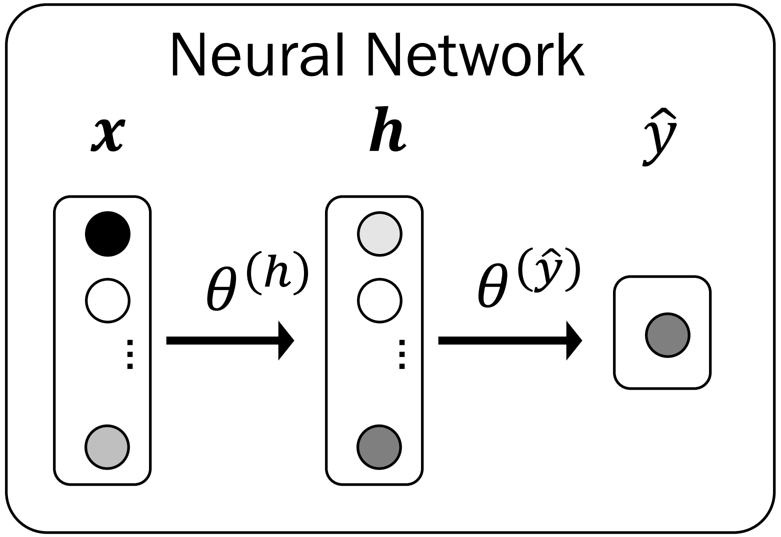
Let $|\boldsymbol{x}| = 40$ and $|\boldsymbol{h}| = 20$.

1. How many parameters are in $\theta^{(\hat{y})}$?
   A. 2
   B. 20
   C. 40
   D. 800

2. How many parameters are in $\theta^{(h)}$?
   A. 2
   B. 20
   C. 40
   D. 800

3. How many parameters in total?
   A. 800
   B. 20
   C. 820
   D. 16000

Quick check



$$h_j = \sigma\left(\sum_{i=0}^{m_x} x_i \theta_{i,j}^{(h)}\right) \qquad \hat{y} = \sigma\left(\sum_{j=0}^{m_h} h_j \theta_j^{(\hat{y})}\right)$$

$$\hat{y} = P(Y = 1 | \boldsymbol{X} = \boldsymbol{x})$$

Let $|\boldsymbol{x}| = 40$ and $|\boldsymbol{h}| = 20$.

1. How many parameters are in $\theta^{(\hat{y})}$?
   A. 2
   B. 20
   C. 40
   D. 800
2. How many parameters are in $\theta^{(h)}$?
   A. 2
   B. 20
   C. 40
   D. 800
3. How many parameters in total?
   A. 800
   B. 20
   C. 820
   D. 16000

Today's plan

Intro to Deep Learning

- Parameters of a neural network
- Training neural networks
- Extra ideas

Training: Neural networks

A neural network (like logistic regression) gets intelligence from its parameters θ.

Training

- Learn parameters $\theta = \left(\theta^{(h)}, \theta^{(\hat{y})}\right)$
- Find θ_{MLE} that maximizes likelihood of training data (MLE)

Testing/ Prediction

For input feature vector $\boldsymbol{X} = \boldsymbol{x}$:

- Use parameters to compute $\hat{y} = P(Y = 1 | \boldsymbol{X} = \boldsymbol{x})$
- If $\hat{y} > 0.5$, predict 1. Else, predict 0.

Training: Learning the parameters via MLE

1. Neural network model

$$h_j = \sigma\left(\sum_{i=0}^{m_x} x_i \theta_{i,j}^{(h)}\right) \qquad \hat{y} = \sigma\left(\sum_{j=0}^{m_h} h_j \theta_j^{(\hat{y})}\right)$$

$$P(Y = 1|\boldsymbol{X} = \boldsymbol{x}) = \hat{y}$$

2. Compute log-likelihood of training data

Quick check: Why?

3. Compute partial derivative of log-likelihood with respect to each parameter

Quick check: Why?

Training: Learning the parameters via MLE

1. Neural network model

$$h_j = \sigma\left(\sum_{i=0}^{m_x} x_i \theta_{i,j}^{(h)}\right) \qquad \hat{y} = \sigma\left(\sum_{j=0}^{m_h} h_j \theta_j^{(\hat{y})}\right)$$

$$P(Y = 1 | \boldsymbol{X} = \boldsymbol{x}) = \hat{y}$$

2. Compute log-likelihood of training data

Want to find θ that maximizes $LL(\theta)$ of training data

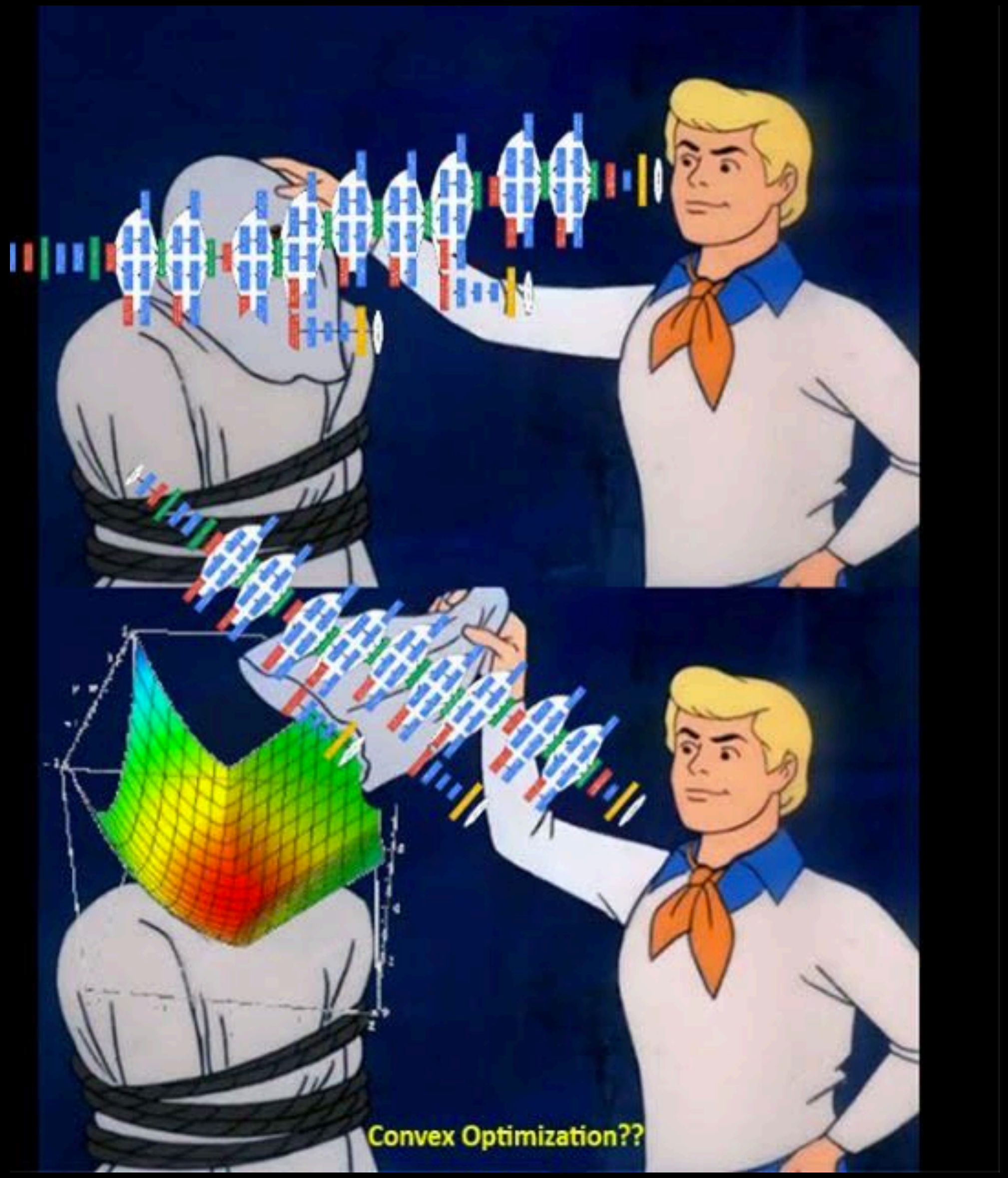
$$\theta_{MLE} = \arg\max_{\theta} LL(\theta)$$

3. Compute partial derivative of log-likelihood with respect to each parameter

Need gradient of $LL(\theta)$ w.r.t. all parameters to optimize (e.g., via gradient ascent)

🤔


Okay gang, let's see what deep learning really is.


Convex Optimization??

Training: Learning the parameters via MLE

1. Neural network model

$$h_j = \sigma\left(\sum_{i=0}^{m_x} x_i \theta_{i,j}^{(h)}\right) \qquad \hat{y} = \sigma\left(\sum_{j=0}^{m_h} h_j \theta_j^{(\hat{y})}\right)$$

$$P(Y = 1|\boldsymbol{X} = \boldsymbol{x}) = \hat{y}$$

2. Compute log-likelihood of training data

3. Compute partial derivative of log-likelihood with respect to each parameter

Same prediction, same log-likelihood

Neural network model:

$$h_j = \sigma\left(\sum_{i=0}^{m_x} x_i \theta_{i,j}^{(h)}\right) \qquad \hat{y} = \sigma\left(\sum_{j=0}^{m_h} h_j \theta_j^{(\hat{y})}\right)$$

$$P(Y=1|\boldsymbol{X}=\boldsymbol{x}) = \hat{y}$$
$$P(Y=0|\boldsymbol{X}=\boldsymbol{x}) = 1-\hat{y}$$
$$\Leftrightarrow \quad P(Y=y|\boldsymbol{X}=\boldsymbol{x}) = (\hat{y})^y(1-\hat{y})^{1-y}$$

(see Bernoulli PMF)

Likelihood of training data:

$$L(\theta) = \prod_{i=1}^{n} P\left(Y=y^{(i)}|\boldsymbol{X}=\boldsymbol{x}^{(i)},\theta\right) = \prod_{i=1}^{n}\left(\hat{y}^{(i)}\right)^{y^{(i)}}\left(1-\hat{y}^{(i)}\right)^{1-y^{(i)}}$$

Log-likelihood:

$$LL(\theta) = \sum_{i=1}^{n} y^{(i)} \log \hat{y}^{(i)} + \left(1-y^{(i)}\right)\log\left(1-\hat{y}^{(i)}\right)$$

👉 Neural networks maximize the same log-likelihood function as logistic regression.

Training: Learning the parameters via MLE

1. Neural network model

$$h_j = \sigma\left(\sum_{i=0}^{m_x} x_i \theta_{i,j}^{(h)}\right) \qquad \hat{y} = \sigma\left(\sum_{j=0}^{m_h} h_j \theta_j^{(\hat{y})}\right)$$

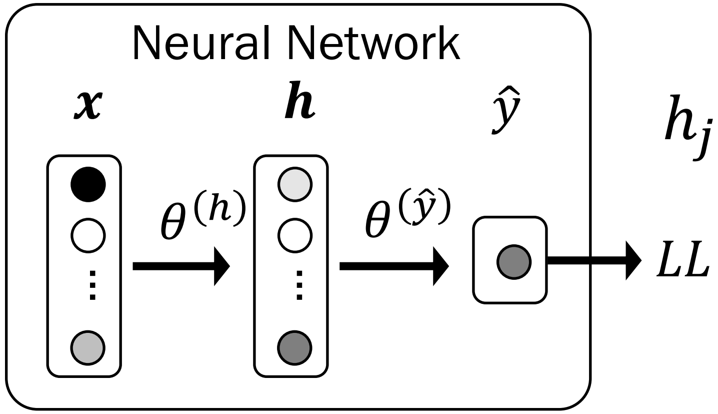
$$P(Y = 1 | \boldsymbol{X} = \boldsymbol{x}) = \hat{y}$$

2. Compute log-likelihood of training data

$$LL(\theta) = \sum_{i=1}^{n} y^{(i)} \log \hat{y}^{(i)} + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)$$

3. Compute partial derivative of log-likelihood with respect to each parameter

Computing gradient



$$h_j = \sigma\left(\sum_{i=0}^{m_x} x_i \theta_{i,j}^{(h)}\right), \hat{y} = \sigma\left(\sum_{j=0}^{m_h} h_j \theta_j^{(\hat{y})}\right)$$

$$\hat{y} = P(Y = 1 | \boldsymbol{X} = \boldsymbol{x})$$

Goals:

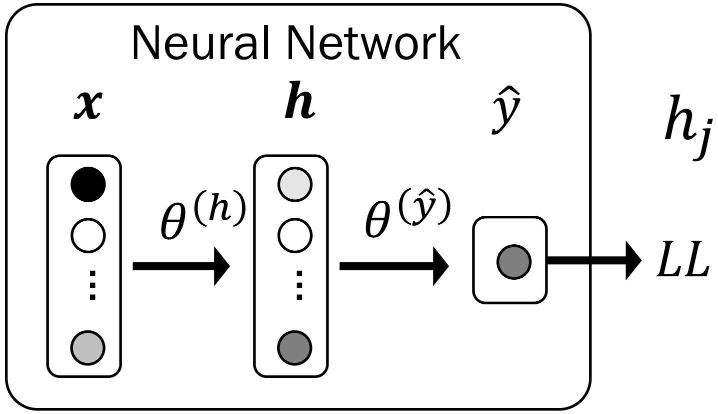
$$\frac{\partial LL(\theta)}{\partial \theta_j^{(\hat{y})}}$$

Gradient with respect to output layer parameters

$$\frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}}$$

Gradient with respect to hidden layer parameters

Bad choice



$$h_j = \sigma\left(\sum_{i=0}^{m_x} x_i \theta_{i,j}^{(h)}\right), \hat{y} = \sigma\left(\sum_{j=0}^{m_h} h_j \theta_j^{(\hat{y})}\right)$$

$$\hat{y} = P(Y = 1 | \boldsymbol{X} = \boldsymbol{x})$$

Goals:

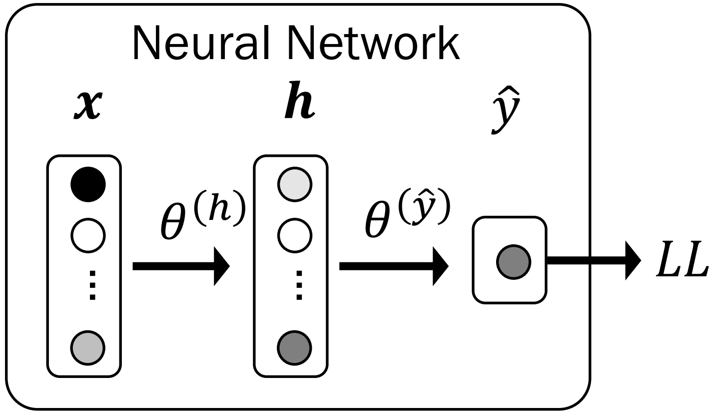
$$\frac{\partial LL(\theta)}{\partial \theta_j^{(\hat{y})}}, \frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}}$$

$$\hat{y} = \sigma\left(\sum_{j=0}^{m_h} h_j \theta_j^{(\hat{y})}\right) = \sigma\left(\sum_{j=0}^{m_h} \sigma\left(\sum_{i=0}^{m_x} x_i \theta_{i,j}^{(h)}\right) \theta_j^{(\hat{y})}\right)$$

$$LL(\theta) = \sum_{i=1}^{n} y^{(i)} \log \hat{y}^{(i)} + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)$$

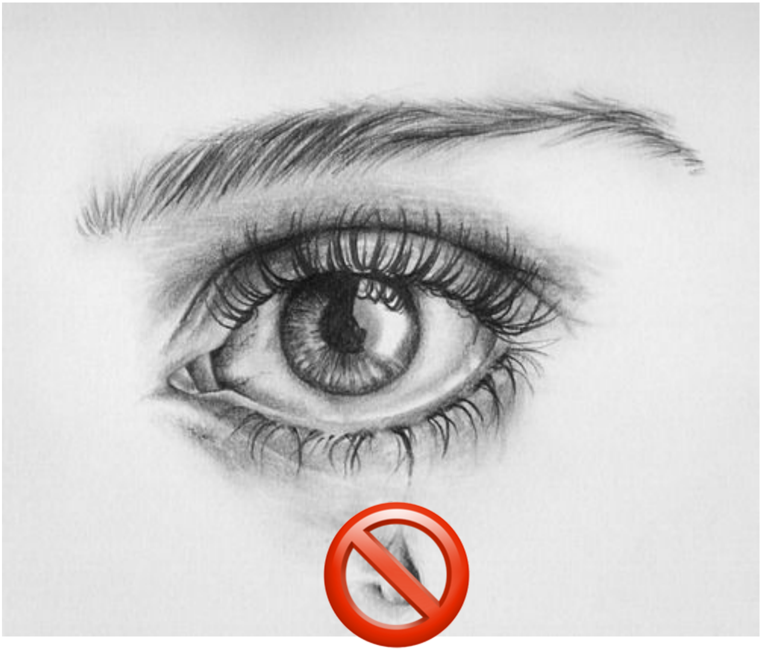
Math bugs galore

Derivatives without tears

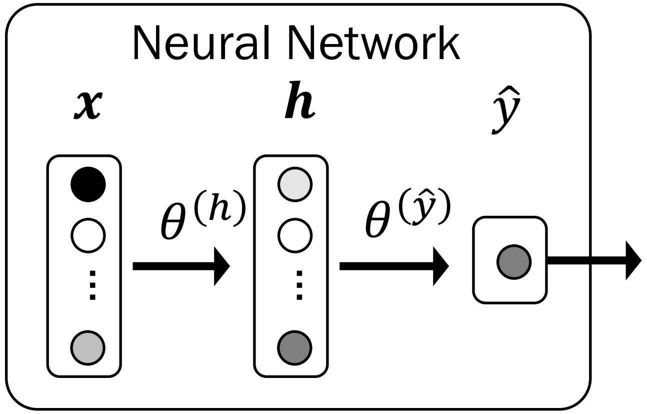


Goals:

$$\frac{\partial LL(\theta)}{\partial \theta_j^{(\hat{y})}}, \frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}}$$



Big idea #1: Derivative of sum



$$LL(\theta) = \sum_{i=1}^{n} y^{(i)} \log \hat{y}^{(i)} + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)$$

Goals:

$$\frac{\partial LL(\theta)}{\partial \theta_j^{(\hat{y})}}, \frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}}$$

$$\frac{\partial LL(\theta)}{\partial \theta_j} = \sum_{i=1}^{n} \frac{\partial}{\partial \theta_j}\left[y^{(i)} \log \hat{y}^{(i)} + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)\right]$$

sum of derivatives

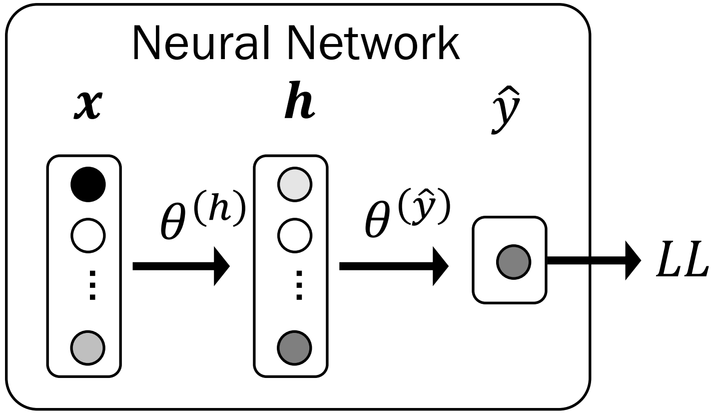
- We only need to calculate the gradients with respect to one training example!
- We can then sum up these gradients to get the final answer.

For the next few slides, pretend you only have one training example $(\boldsymbol{x}, y)$:

$$LL(\theta) = y \log \hat{y} + (1 - y) \log(1 - \hat{y})$$

Big idea #2: Chain rule

$$LL(\theta) = y \log \hat{y} + (1-y)\log(1-\hat{y})$$



Goals:

$$\frac{\partial LL(\theta)}{\partial \theta_j^{(\hat{y})}}, \frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}}$$

Chain Rule of Calculus:

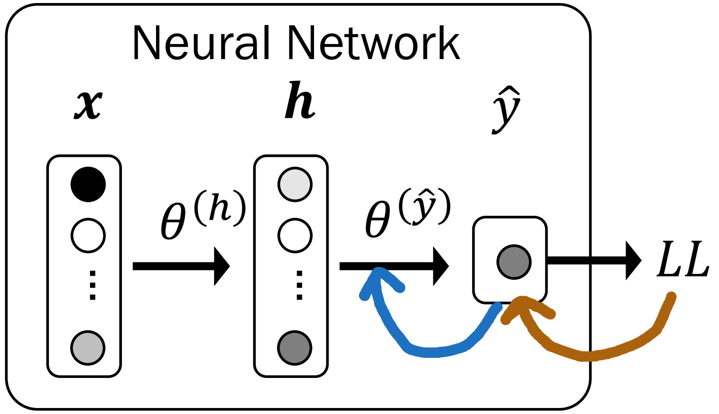
$$f(x) = f\big(z(x)\big)$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial z} \cdot \frac{\partial z}{\partial x}$$

👉 Decomposing the gradient via chain rule greatly simplifies our partial derivatives.

Applying chain rule (Example 1)

$LL(\theta) = y \log \hat{y} + (1 - y) \log(1 - \hat{y})$



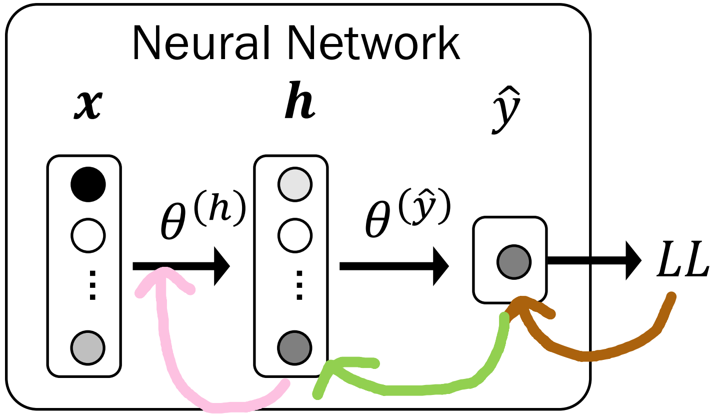
Goals:

$\boxed{\dfrac{\partial LL(\theta)}{\partial \theta_j^{(\hat{y})}}}, \dfrac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}}$

$$\frac{\partial LL(\theta)}{\partial \theta_j^{(\hat{y})}} = \frac{\partial LL}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \theta_j^{(\hat{y})}}$$

Applying chain rule (Example 2)

$$LL(\theta) = y \log \hat{y} + (1-y)\log(1-\hat{y})$$



Goals:

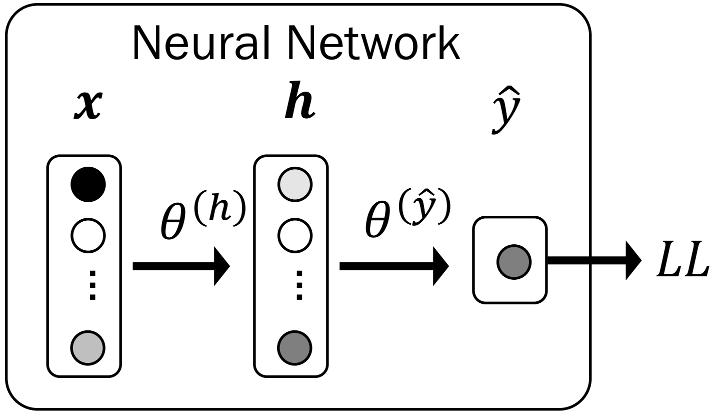
$$\frac{\partial LL(\theta)}{\partial \theta_j^{(\hat{y})}}, \boxed{\frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}}}$$

$$\frac{\partial LL(\theta)}{\partial \theta_j^{(\hat{y})}} = \frac{\partial LL}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \theta_j^{(\hat{y})}}$$

$$\frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}} = \frac{\partial LL}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h_j} \cdot \frac{\partial h_j}{\partial \theta_{i,j}^{(h)}}$$

Chain Rule results

$$LL(\theta) = y \log \hat{y} + (1 - y) \log(1 - \hat{y})$$



Goals:

$$\frac{\partial LL(\theta)}{\partial \theta_j^{(\hat{y})}}, \frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}}$$

$$\frac{\partial LL(\theta)}{\partial \theta_j^{(\hat{y})}} = \frac{\partial LL}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \theta_j^{(\hat{y})}}$$

$$\frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}} = \frac{\partial LL}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h_j} \cdot \frac{\partial h_j}{\partial \theta_{i,j}^{(h)}}$$

Gradients to calculate:

$$\frac{\partial LL}{\partial \hat{y}} \quad \frac{\partial \hat{y}}{\partial \theta_j^{(\hat{y})}}$$

$$\frac{\partial \hat{y}}{\partial h_j} \quad \frac{\partial h_j}{\partial \theta_{i,j}^{(h)}}$$

Big idea #3: Derivative of sigmoid

Sigmoid function:

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

Derivative:

$$\frac{d}{dz}\sigma(z) = \sigma(z)[1 - \sigma(z)]$$

Dear Past Lisa,
Write this on the board!
– Thanks, Future Lisa

👉 When taking derivatives of the sigmoid function, just look at this slide.

Break for Friday/ announcements

Announcements

<u>Office Hours</u>

During Thanksgiving break: None

<u>Week 10 schedule</u>

Monday: **Review lecture** (TA-run)
Optional project due, 11:59pm

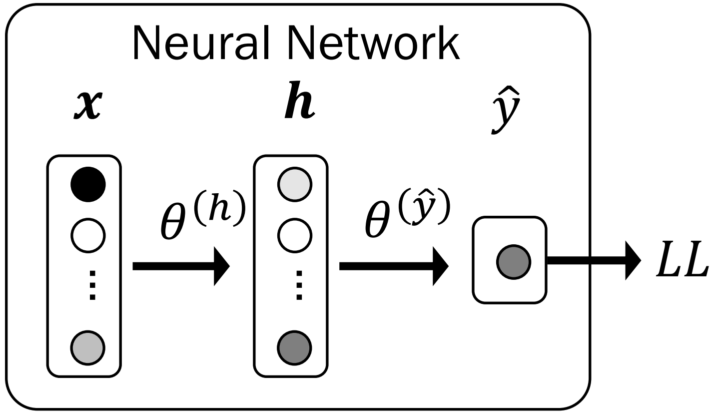
Tuesday: Last concept check (1pm)

Wednesday: **Beyond CS109 lecture**
Problem Set 6 due (1pm)

Friday: **No class (Dead day)**
Last day to turn in pset6 (late)

Chain Rule results

$$LL(\theta) = y \log \hat{y} + (1 - y) \log(1 - \hat{y})$$



Goals:

$$\frac{\partial LL(\theta)}{\partial \theta_j^{(\hat{y})}}, \frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}}$$

$$\frac{\partial LL(\theta)}{\partial \theta_j^{(\hat{y})}} = \frac{\partial LL}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \theta_j^{(\hat{y})}}$$

$$\frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}} = \frac{\partial LL}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h_j} \cdot \frac{\partial h_j}{\partial \theta_{i,j}^{(h)}}$$

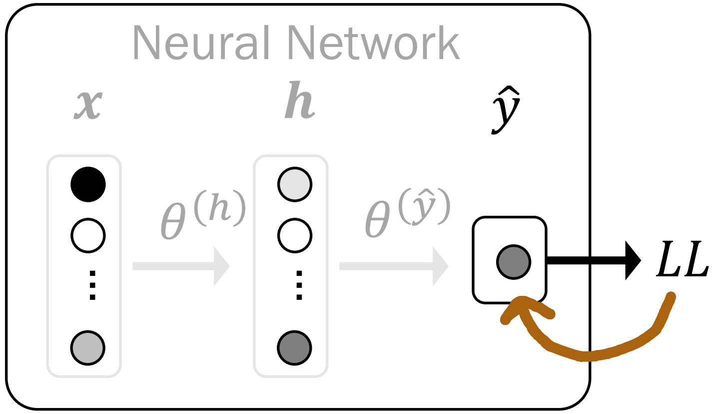
Next step: calculate these four gradients

Gradients to calculate:

$$\frac{\partial LL}{\partial \hat{y}} \qquad \frac{\partial \hat{y}}{\partial \theta_j^{(\hat{y})}}$$

$$\frac{\partial \hat{y}}{\partial h_j} \qquad \frac{\partial h_j}{\partial \theta_{i,j}^{(h)}}$$

Gradient #1



$$LL(\theta) = y \log \hat{y} + (1-y)\log(1-\hat{y})$$

$$\frac{\partial LL(\theta)}{\partial \hat{y}} = \frac{\partial}{\partial \hat{y}}[y \log \hat{y}] + \frac{\partial}{\partial \hat{y}}[(1-y)\log(1-\hat{y})]$$

$$= y \cdot \frac{1}{\hat{y}} - (1-y)\cdot\frac{1}{1-\hat{y}}$$

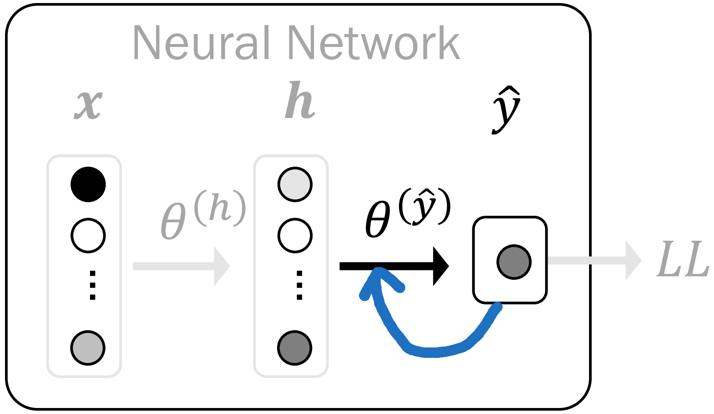
$$= \boxed{\frac{y-\hat{y}}{\hat{y}(1-\hat{y})}}$$

Gradients to calculate:

$$\boxed{\frac{\partial LL}{\partial \hat{y}}} \qquad \frac{\partial \hat{y}}{\partial \theta_j^{(\hat{y})}}$$

$$\frac{\partial \hat{y}}{\partial h_j} \qquad \frac{\partial h_j}{\partial \theta_{i,j}^{(h)}}$$

Gradient #2



$$\hat{y} = \sigma\left(\sum_{j=0}^{m_h} h_j \theta_j^{(\hat{y})}\right)$$

$$\hat{y} = \sigma\left(\sum_{j=0}^{m_h} h_j \theta_j^{(\hat{y})}\right) = \sigma(z) \quad \text{where } z = \sum_{j=0}^{m_h} h_j \theta_j^{(\hat{y})}$$

$$\frac{\partial \hat{y}}{\partial \theta_j^{(\hat{y})}} = \frac{\partial \hat{y}}{\partial z} \cdot \frac{\partial z}{\partial \theta_j^{(\hat{y})}} \quad = \sigma(z)[1 - \sigma(z)] \cdot \frac{\partial z}{\partial \theta_j^{(\hat{y})}}$$

$$= \boxed{\hat{y}[1 - \hat{y}] \cdot h_j}$$

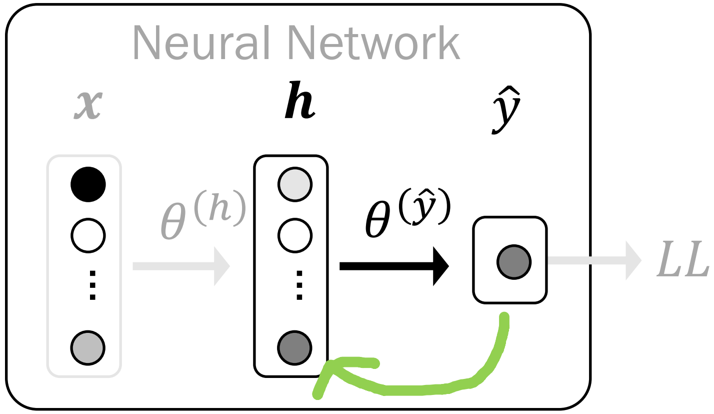
What! That's not scary!

Gradients to calculate:

$\frac{\partial LL}{\partial \hat{y}}$ $\quad \boxed{\frac{\partial \hat{y}}{\partial \theta_j^{(\hat{y})}}}$

$\frac{\partial \hat{y}}{\partial h_j}$ $\quad \frac{\partial h_j}{\partial \theta_{i,j}^{(h)}}$

Gradient #3



$$\hat{y} = \sigma\left(\sum_{j=0}^{m_h} h_j \theta_j^{(\hat{y})}\right)$$

$$\hat{y} = \sigma\left(\sum_{j=0}^{m_h} h_j \theta_j^{(\hat{y})}\right) = \sigma(z) \qquad \text{where } z = \sum_{j=0}^{m_h} h_j \theta_j^{(\hat{y})}$$

$$\frac{\partial \hat{y}}{\partial h_j} = \boxed{\hat{y}[1-\hat{y}] \cdot \theta_j^{(\hat{y})}}$$

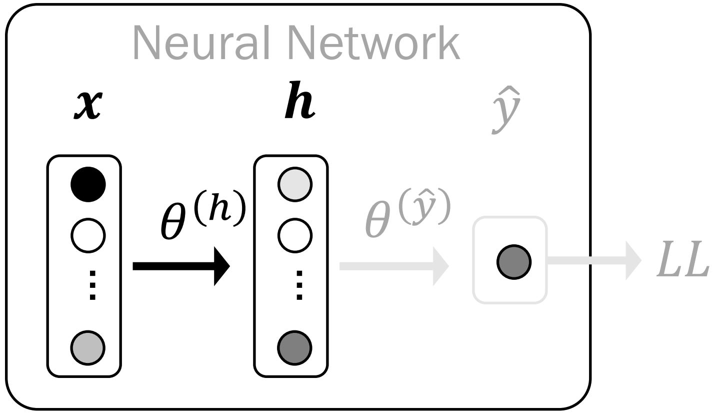
Wait, is it over?

Gradients to calculate:

$$\frac{\partial LL}{\partial \hat{y}} \qquad \frac{\partial \hat{y}}{\partial \theta_j^{(\hat{y})}}$$

$$\boxed{\frac{\partial \hat{y}}{\partial h_j}} ✅ \qquad \frac{\partial h_j}{\partial \theta_{i,j}^{(h)}}$$

Gradient #4



$$h_j = \sigma\left(\sum_{i=0}^{m_x} x_i \theta_{i,j}^{(h)}\right)$$

$$h_j = \sigma\left(\sum_{i=0}^{m_x} x_i \theta_{i,j}^{(h)}\right)$$

$$\frac{\partial h_j}{\partial \theta_{i,j}^{(h)}} = h_j\left[1 - h_j\right] x_i$$

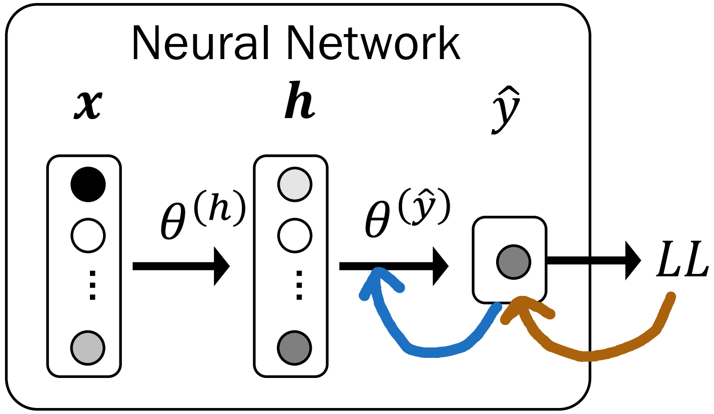
Can...we celebrate?

Gradients to calculate:

$$\frac{\partial LL}{\partial \hat{y}} \qquad \frac{\partial \hat{y}}{\partial \theta_j^{(\hat{y})}}$$

$$\frac{\partial \hat{y}}{\partial h_j} \qquad \frac{\partial h_j}{\partial \theta_{i,j}^{(h)}}$$

Put it all together: Gradient of output layer params



Goals:

$$\boxed{\frac{\partial LL(\theta)}{\partial \theta_j^{(\hat{y})}}}, \frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}}$$

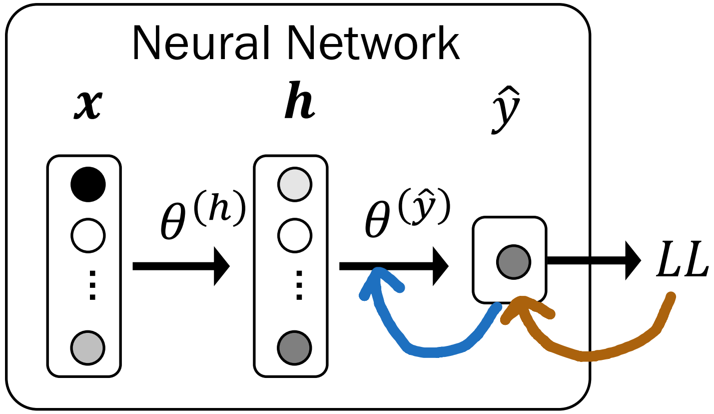
1. By Chain Rule, multiply partial derivatives

$$\frac{\partial LL(\theta)}{\partial \theta_j^{(\hat{y})}} = \underbrace{\frac{\partial LL}{\partial \hat{y}}}_{\text{grad1}} \cdot \underbrace{\frac{\partial \hat{y}}{\partial \theta_j^{(\hat{y})}}}_{\text{grad2}}$$

$$= \frac{y - \hat{y}}{\hat{y}(1 - \hat{y})} \cdot \hat{y}[1 - \hat{y}] \cdot h_j$$

$$= (y - \hat{y}) \cdot h_j$$

Put it all together: Gradient of output layer params



Goals:

$$\boxed{\frac{\partial LL(\theta)}{\partial \theta_j^{(\hat{y})}}}, \frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}}$$

1. By Chain Rule, multiply partial derivatives

$$\frac{\partial LL(\theta)^{(i)}}{\partial \theta_j^{(\hat{y})}} = \left(y^{(i)} - \hat{y}^{(i)}\right) \cdot h_j^{(i)}$$

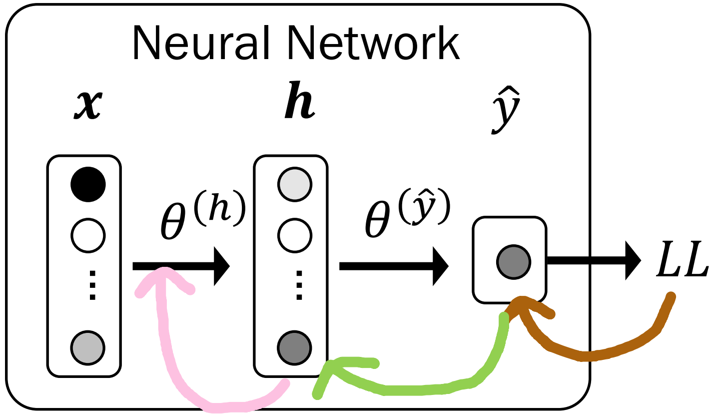
(gradient of LL w.r.t. i-th datapoint)

2. Sum up gradients w.r.t. each example

$$LL(\theta) = \sum_{i=1}^{n} LL(\theta)^{(i)}$$

$$\frac{\partial LL(\theta)}{\partial \theta_j^{(\hat{y})}} = \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)}\right) \cdot h_j^{(i)}$$

Put it all together: Gradient of hidden layer params



Goals:

$$\frac{\partial LL(\theta)}{\partial \theta_j^{(\hat{y})}}, \boxed{\frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}}}$$

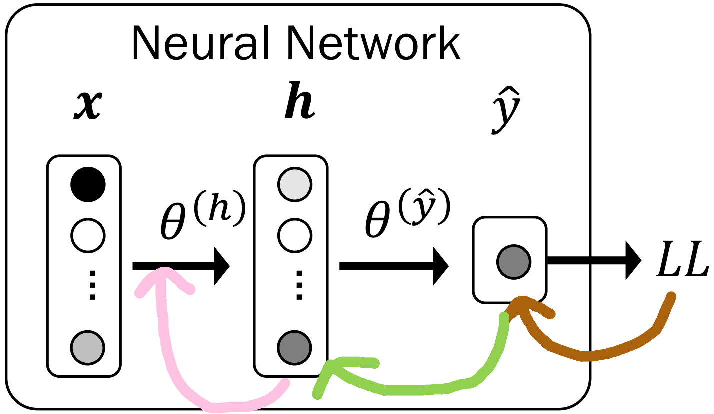
1. By Chain Rule, multiply partial derivatives

$$\frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}} = \underbrace{\frac{\partial LL}{\partial \hat{y}}}_{\text{grad1}} \cdot \underbrace{\frac{\partial \hat{y}}{\partial h_j}}_{\text{grad3}} \cdot \underbrace{\frac{\partial h_j}{\partial \theta_{i,j}^{(h)}}}_{\text{grad4}}$$

$$= \frac{y - \hat{y}}{\hat{y}(1-\hat{y})} \cdot \hat{y}[1-\hat{y}] \cdot \theta_j^{(\hat{y})} h_j [1 - h_j] x_i$$

$$= (y - \hat{y}) \cdot \theta_j^{(\hat{y})} h_j [1 - h_j] x_i$$

Put it all together: Gradient of output layer params



Goals:

$$\frac{\partial LL(\theta)}{\partial \theta_j^{(\hat{y})}}, \boxed{\frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}}} ✅$$

1. By Chain Rule, multiply partial derivatives
2. Sum up gradients w.r.t. each example

$$\frac{\partial LL(\theta)^{(k)}}{\partial \theta_{i,j}^{(h)}} = \left(y^{(k)} - \hat{y}^{(k)}\right) \cdot \theta_j^{(\hat{y})} h_j^{(k)} \left[1 - h_j^{(k)}\right] x_i^{(k)}$$

(gradient of LL w.r.t. k-th datapoint)

$$LL(\theta) = \sum_{k=1}^{n} LL(\theta)^{(k)}$$

$$\frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}} = \sum_{k=1}^{n} \left(y^{(k)} - \hat{y}^{(k)}\right) \cdot \theta_j^{(\hat{y})} h_j^{(k)} \left[1 - h_j^{(k)}\right] x_i^{(k)}$$

🙌🙌

Moment of silence

Congratulations 🙌 🙌 🙌

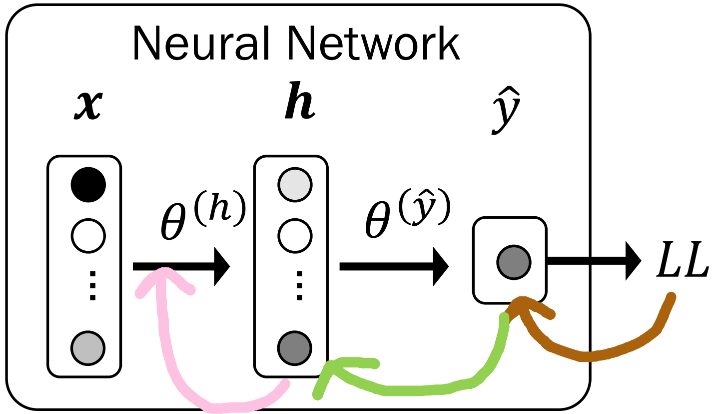
You now know **Backpropagation**.

<u>def</u> Using chain rule, **backpropagate** gradients from later layers into earlier layers by multiplying.

Backpropagation

Key idea of backpropagation:
Compute gradients by using chain rule.

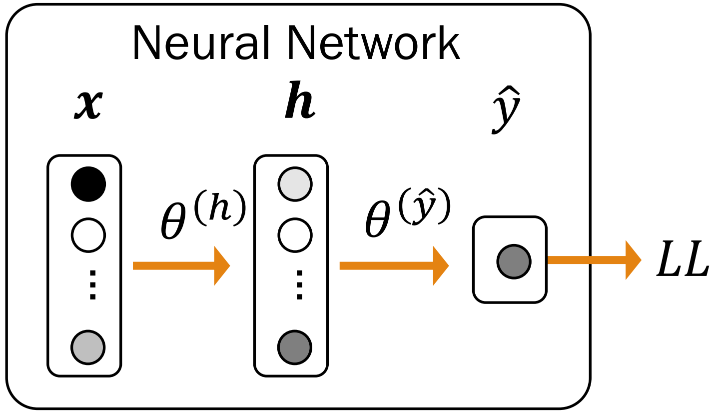
- Compute gradients of later layers…
- To help with gradients of earlier layers.



Train (backward pass)

Notice parallel to forward pass (i.e., propagation):

- Compute neurons of earlier layers…
- To help with neurons of later layers.



Predict (forward pass)

Training: Summary

1. Neural net model:

$$\hat{y} = \sigma\left(\sum_{j=0}^{m_h} h_j \theta_j^{(\hat{y})}\right) \qquad h_j = \sigma\left(\sum_{i=0}^{m_x} x_i \theta_{i,j}^{(h)}\right)$$

$$P(Y = 1|\boldsymbol{X} = \boldsymbol{x}) = \hat{y}$$
$$P(Y = 0|\boldsymbol{X} = \boldsymbol{x}) = 1 - \hat{y}$$

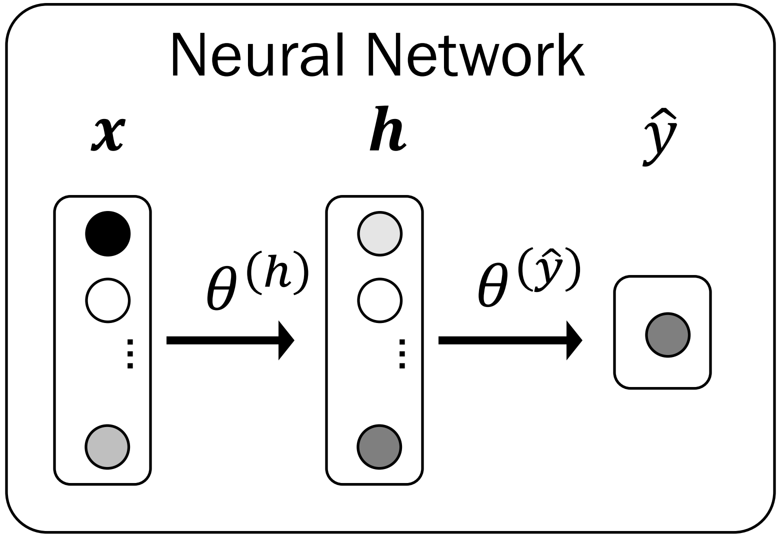
2. Compute log-likelihood of training data:

$$LL(\theta) = \sum_{i=1}^{n} y^{(i)} \log \hat{y}^{(i)} + \left(1 - y^{(i)}\right) \log\left(1 - \hat{y}^{(i)}\right)$$

3. Compute derivative of log-likelihood with respect to each $\theta_j, j = 0, 1, \dots, m$:

(we just did this)

Summary: Backpropagation



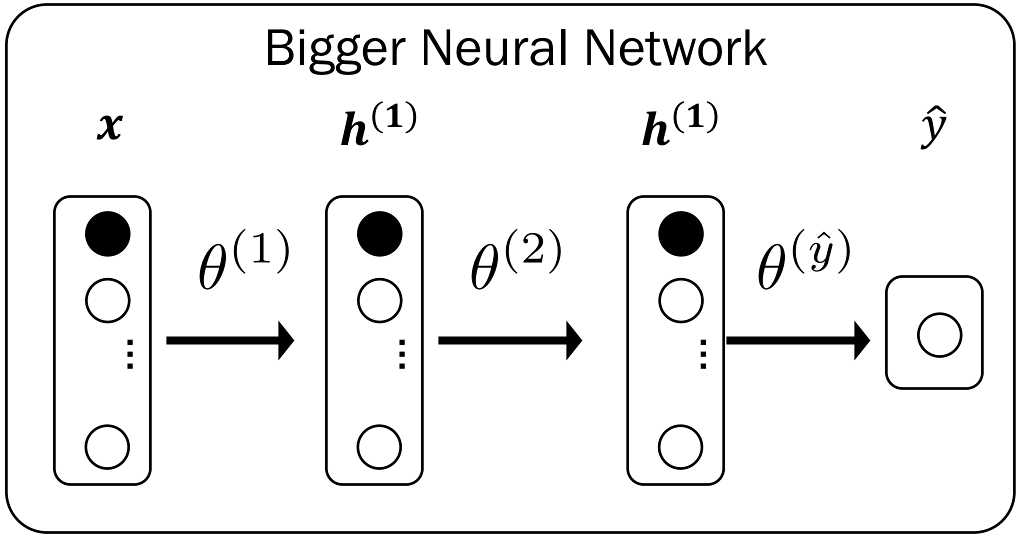
Gradient w.r.t. output layer parameters

$$\frac{\partial LL(\theta)}{\partial \theta_j^{(\hat{y})}} = \frac{\partial LL}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \theta_j^{(\hat{y})}} = \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)}\right) \cdot h_j^{(i)}$$

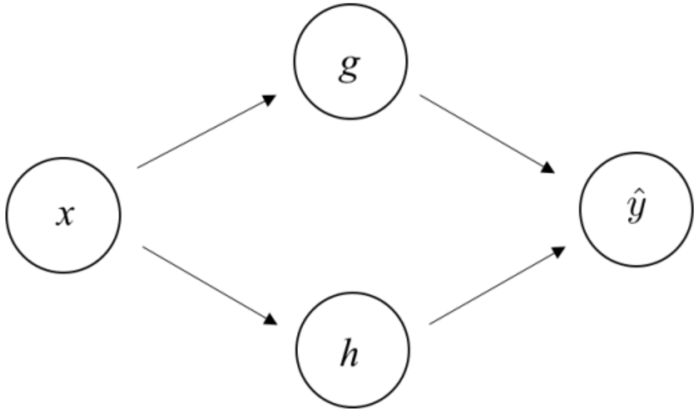
Gradient w.r.t. hidden layer parameters

$$\frac{\partial LL(\theta)}{\partial \theta_{i,j}^{(h)}} = \frac{\partial LL}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h_j} \cdot \frac{\partial h_j}{\partial \theta_{i,j}^{(h)}} = \sum_{k=1}^{n} \left(y^{(k)} - \hat{y}^{(k)}\right) \cdot \theta_j^{(\hat{y})} h_j^{(k)} \left[1 - h_j^{(k)}\right] x_i^{(k)}$$

What would you do here?



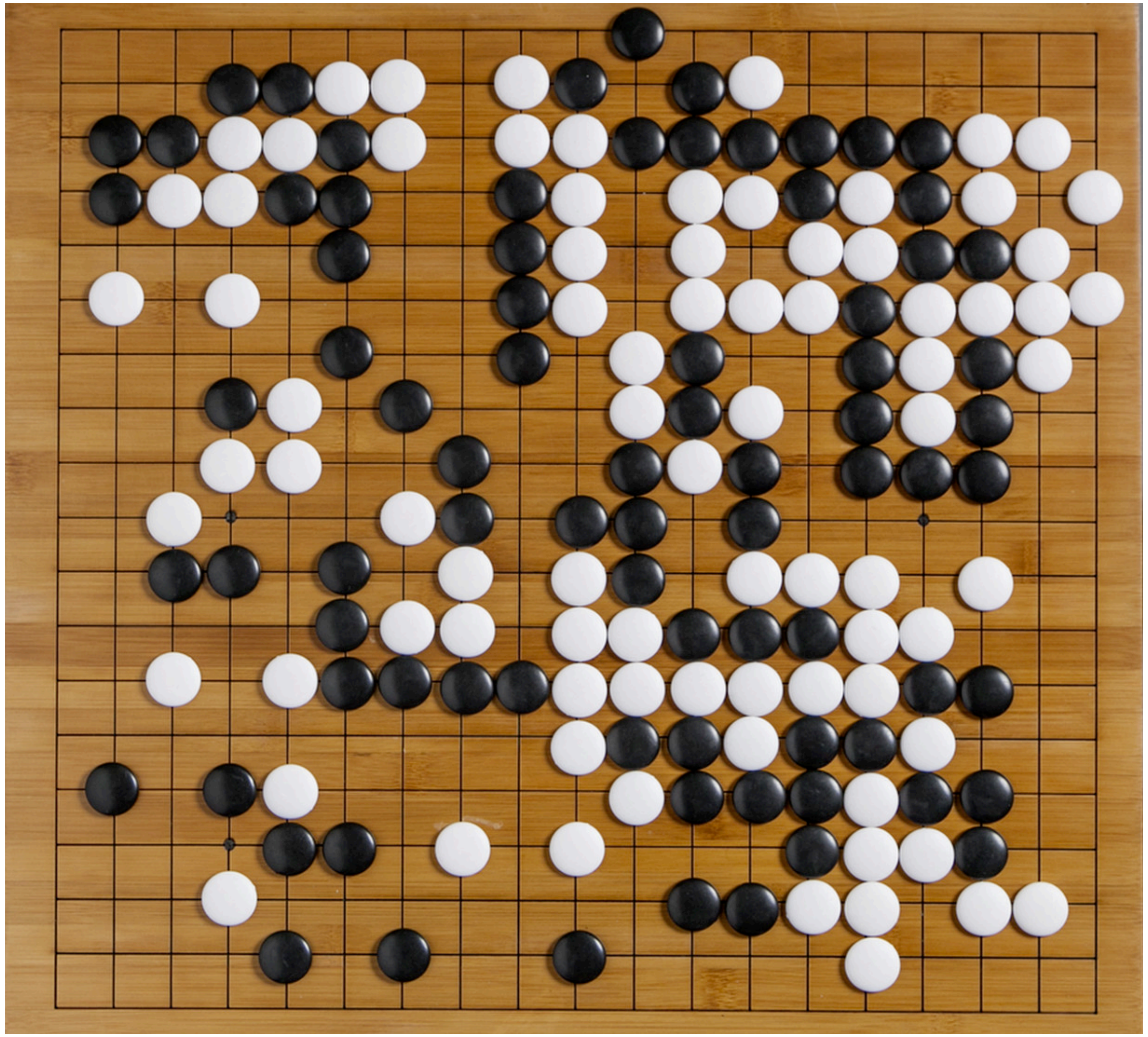
What would you do here?



$$g = \text{sigmoid}(\theta_1 \cdot x)$$
$$h = \text{sigmoid}(\theta_2 \cdot x)$$
$$\hat{y} = \text{sigmoid}(\theta_3 \cdot g + \theta_4 \cdot h)$$
$$LL(\theta) = \sum_{i=1}^{n} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

1. Calculate partial derivatives for one data instance
2. Use chain rule
3. Sigmoid derivatives come out simple with the right decomposition
4. You don’t need to give the most reduced answewr

Innovations in deep learning because of Chain Rule

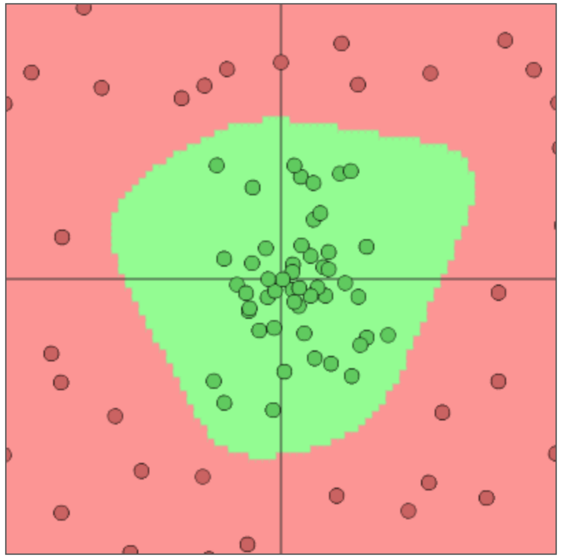


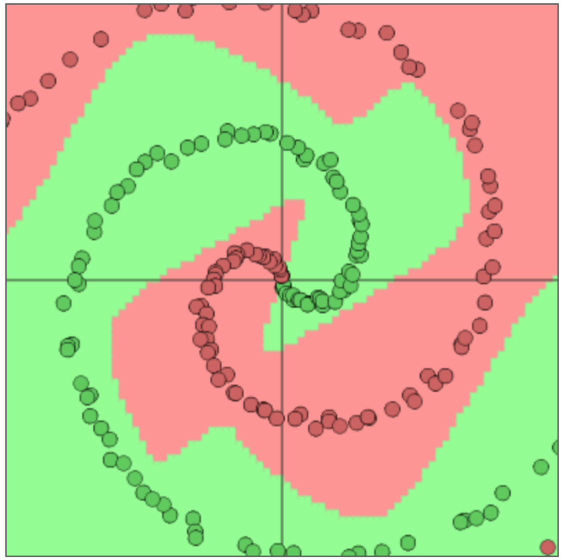
AlphaGO (2016)

Deep learning (neural networks) is the core idea behind the current revolution in AI.

Chain Rule enables us to train parameters in neural networks.

Neural networks can learn complex functions





The classifiers shown are learned by neural networks, which can model *nonlinearly* separable data.

http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

Today's plan

Intro to Deep Learning

- Parameters of a neural network
- Training neural networks
- Extra ideas

Multiple outputs?

Softmax is a generalization of the sigmoid function.

sigmoid(z): value in range [0, 1]

$z \in \mathbb{R}$:

$$P(Y = 1 | \boldsymbol{X} = \boldsymbol{x}) = \sigma(z)$$

(equivalent: Bernoulli p)

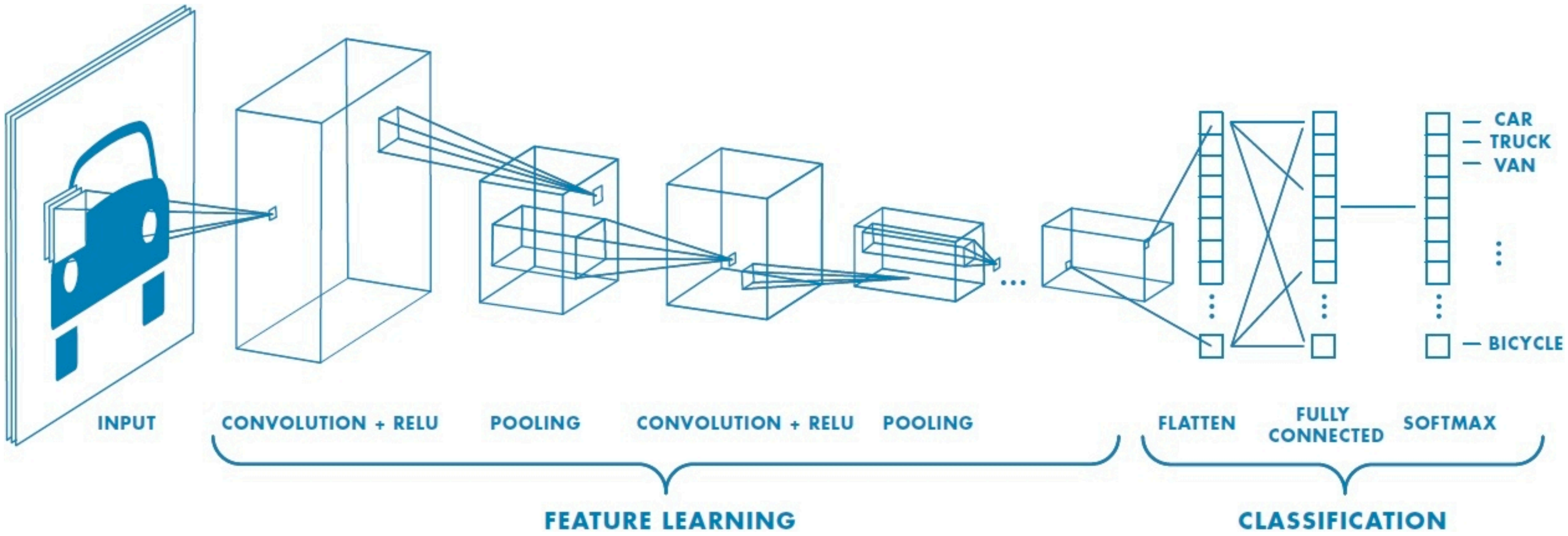
softmax(z): k-dimensional values in range[0,1] that add up to 1

$\boldsymbol{z} \in \mathbb{R}^k$:

$$P(Y = j | \boldsymbol{X} = \boldsymbol{x}) = \text{softmax}(\boldsymbol{z})_j$$

(equivalent: Multinomial $p_1, \dots, p_k$)

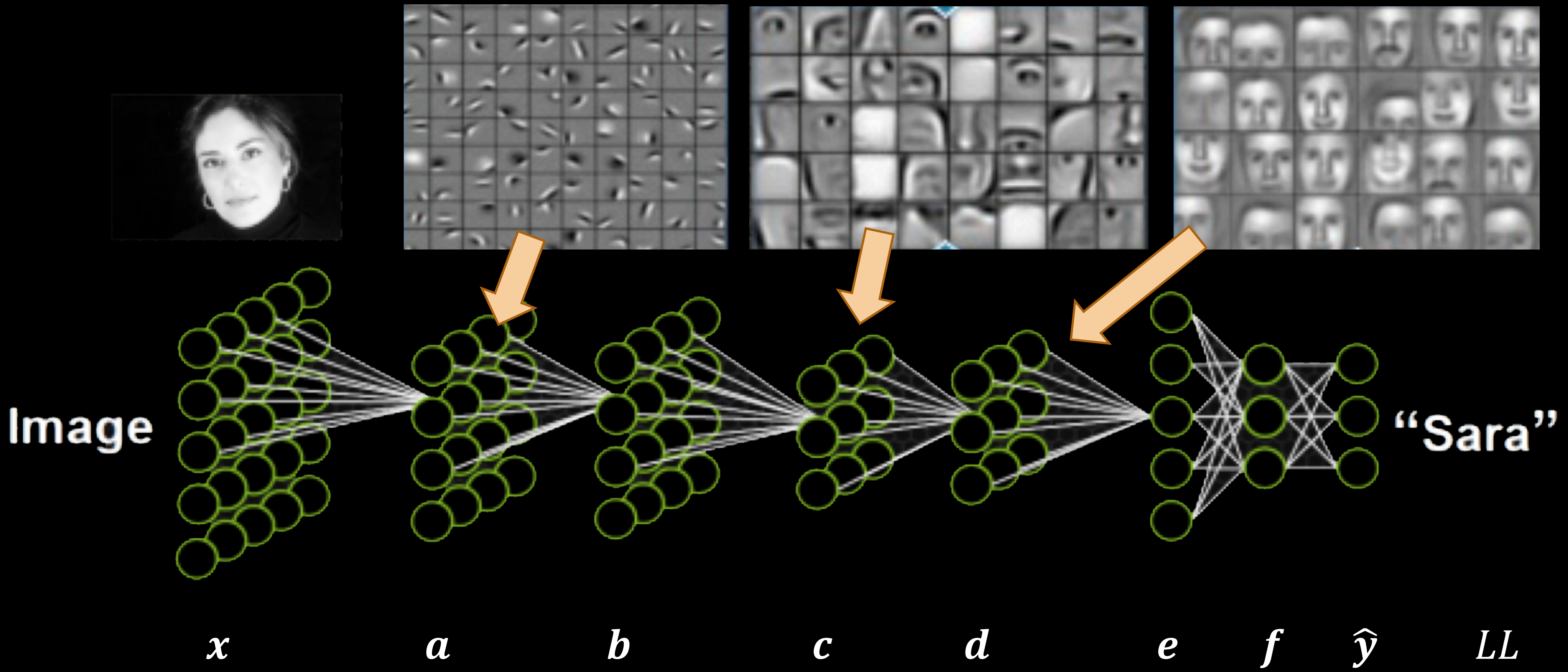
Shared weights?



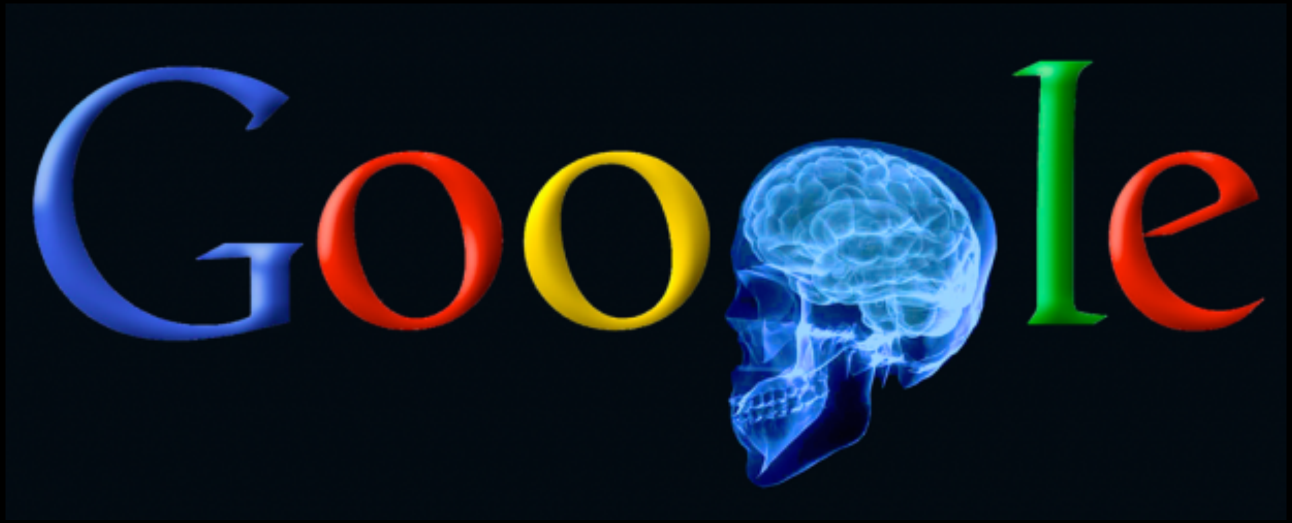
It turns out if you want to force some of your weights to be shared over different neurons, the math isn't much harder.

Convolution is an example of such weight-sharing and is used a lot for vision (Convolutional Neural Networks, CNN).

Neural networks with multiple layers

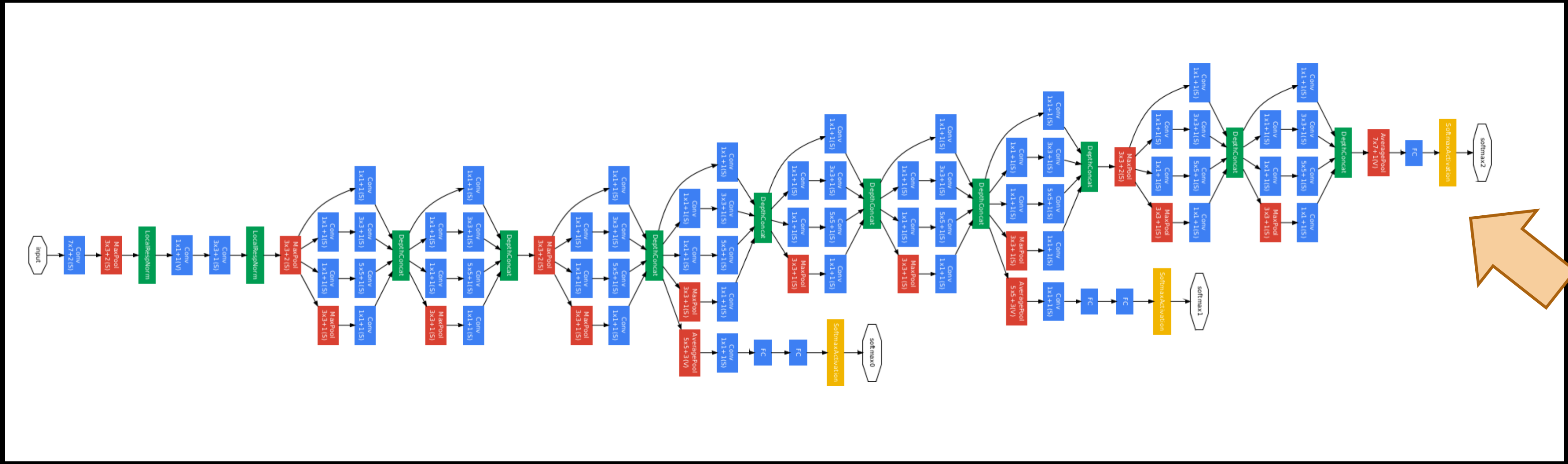


GoogLeNet (2015)



1 Trillion Artificial Neurons

(btw human brains have 1 billion neurons)



Multiple,
Multi class output

22 layers deep!

Szegedy et al., Going Deeper With Convolutions. CVPR 2015

Neurons learn features of the dataset

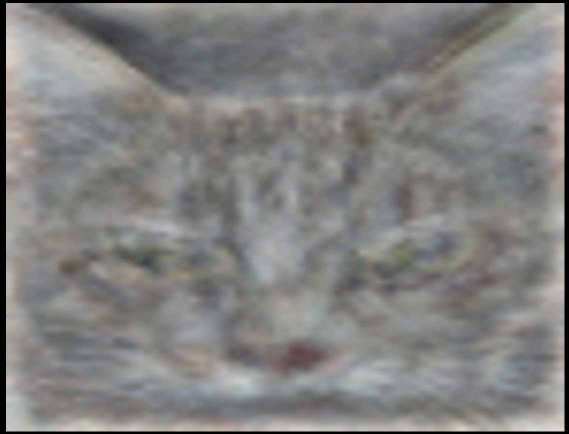
Neurons in later layers will respond strongly to high-level features of your **training data**.

If your training data is faces, you will get lots of face neurons.

If your training data is all of YouTube...

...you get a cat neuron.

Top stimuli in test set

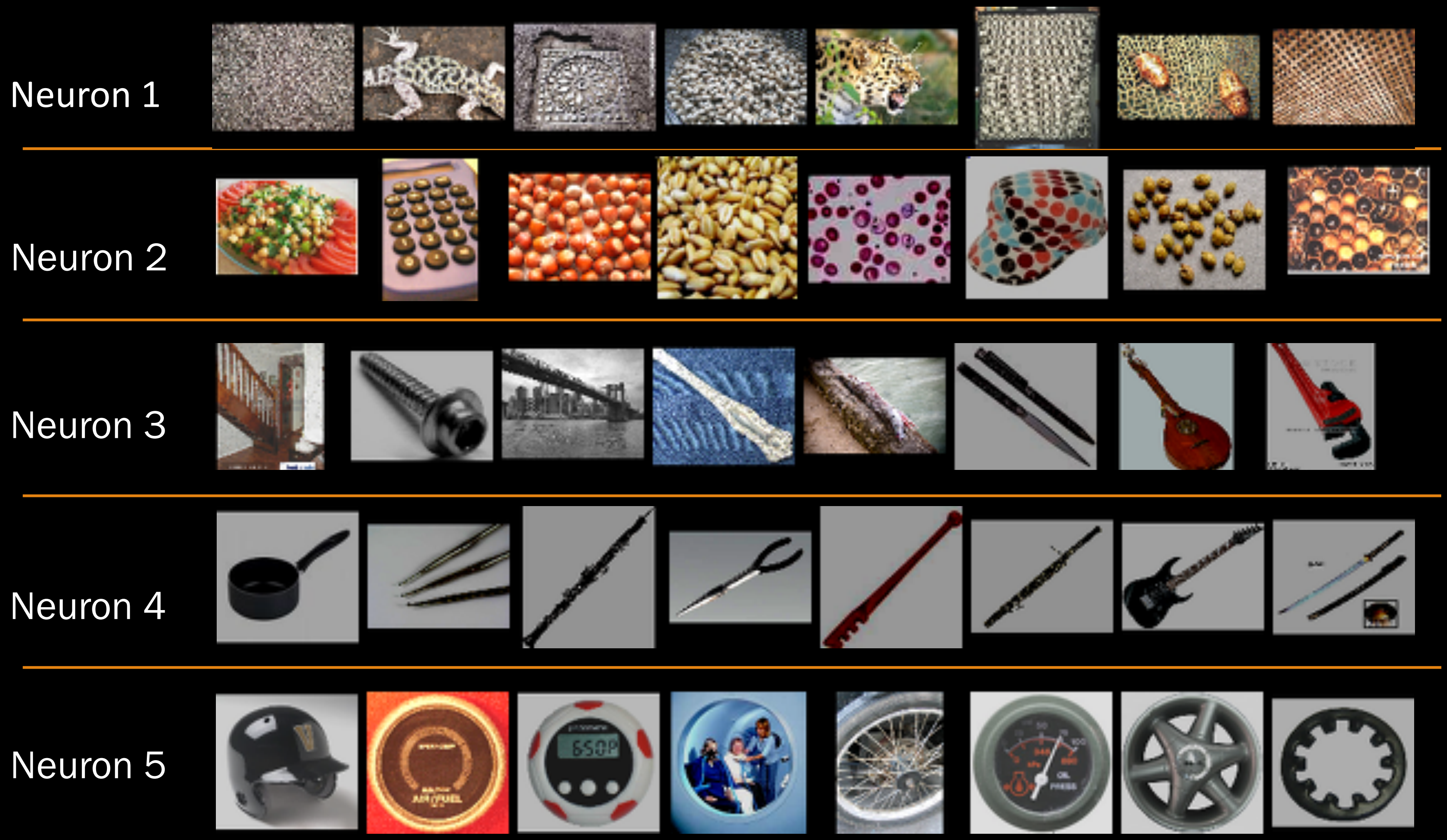


Optimal stimulus found by numerical optimization

Le, et al., Building high-level features using large-scale unsupervised learning. ICML 2012


Hire the smartest people in the world
Invent cat detector

Best neuron stimuli



Le, et al., Building high-level features using large-scale unsupervised learning. ICML 2012

Best neuron stimuli

Neuron 6

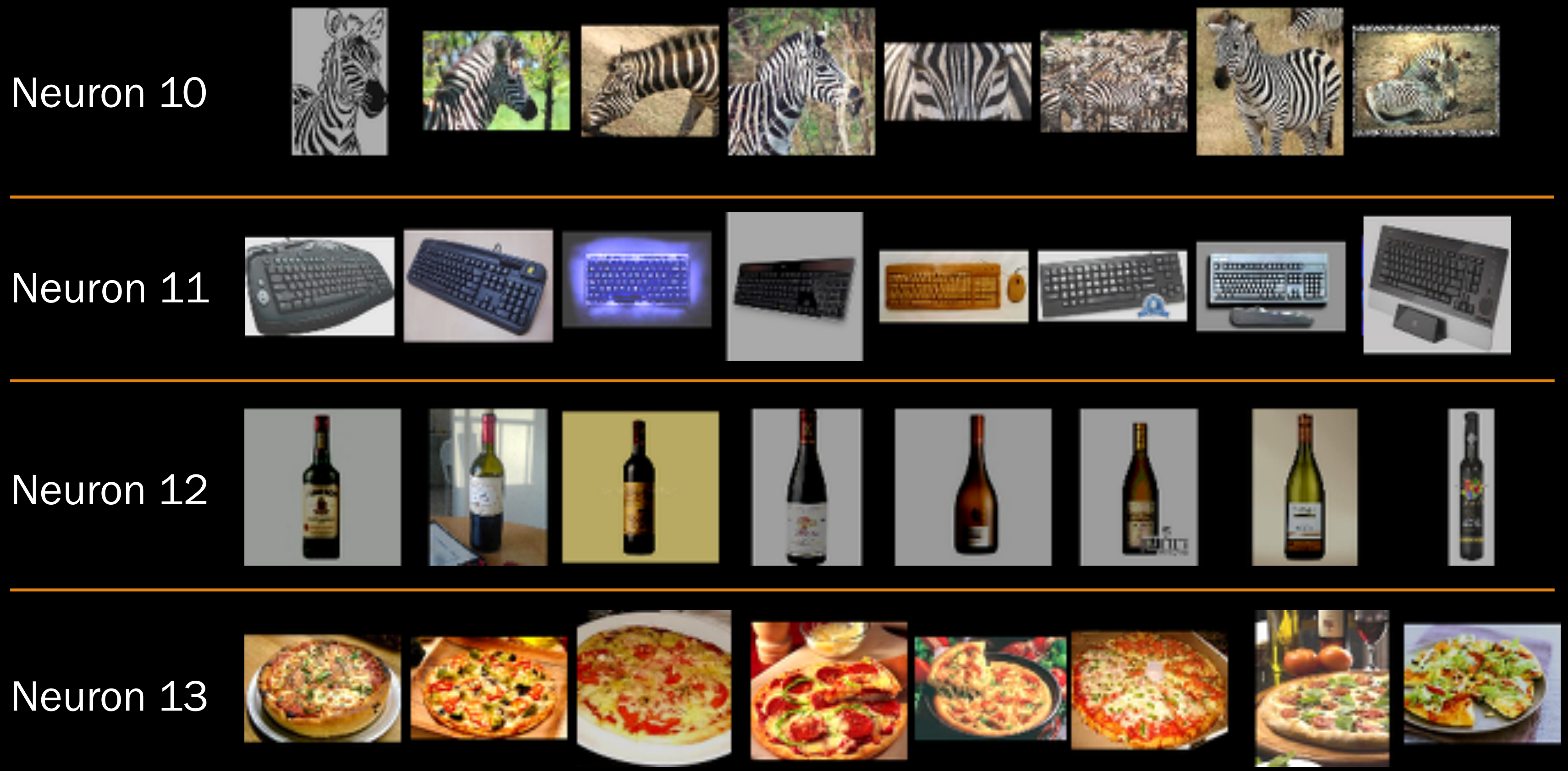
Neuron 7

Neuron 8

Neuron 9

Le, et al., Building high-level features using large-scale unsupervised learning. ICML 2012

Best neuron stimuli



Le, et al., Building high-level features using large-scale unsupervised learning. ICML 2012

ImageNet classification

22,000 categories

14,000,000 images

Hand-engineered features (SIFT, HOG, LBP),
Spatial pyramid, SparseCoding/Compression

Le, et al., Building high-level features using large-scale unsupervised learning. ICML 2012

22,000 is a lot of categories…

…
smoothhound, smoothhound shark, Mustelus mustelus
American smooth dogfish, Mustelus canis
Florida smoothhound, Mustelus norrisi
whitetip shark, reef whitetip shark, Triaenodon obseus
Atlantic spiny dogfish, Squalus acanthias
Pacific spiny dogfish, Squalus suckleyi
hammerhead, hammerhead shark
smooth hammerhead, Sphyrna zygaena
smalleye hammerhead, Sphyrna tudes
shovelhead, bonnethead, bonnet shark, Sphyrna tiburo
angel shark, angelfish, Squatina squatina, monkfish
electric ray, crampfish, numbfish, torpedo
smalltooth sawfish, Pristis pectinatus
guitarfish
roughtail stingray, Dasyatis centroura
butterfly ray
eagle ray
spotted eagle ray, spotted ray, Aetobatus narinari
cownose ray, cow-nosed ray, Rhinoptera bonasus
manta, manta ray, devilfish
Atlantic manta, Manta birostris
devil ray, Mobula hypostoma
grey skate, gray skate, Raja batis
little skate, Raja erinacea
…

Stingray

Mantaray

Le, et al., Building high-level features using large-scale unsupervised learning. ICML 2012

Best neuron stimuli

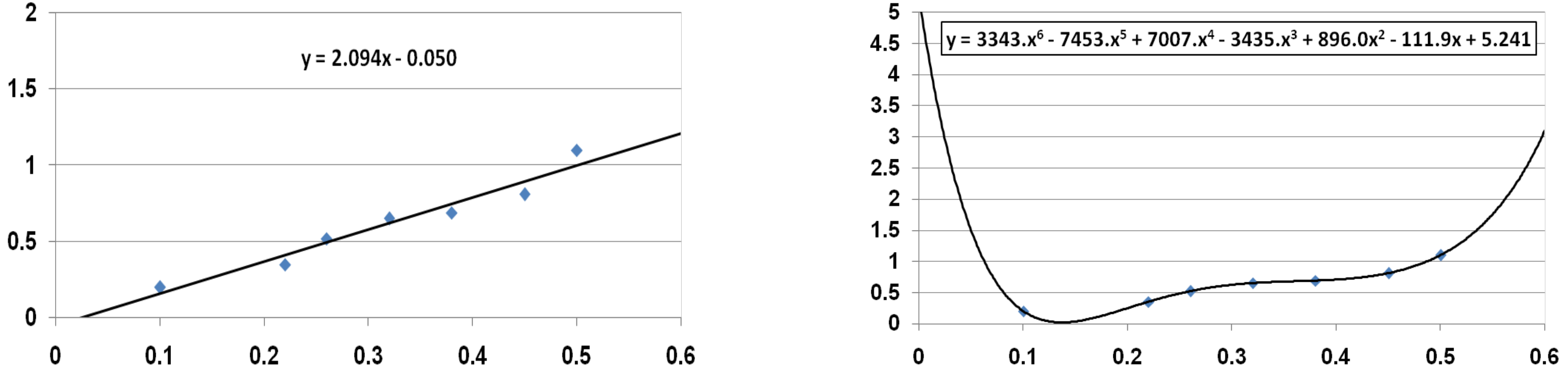
0.005%	1.5%	43.9%
Random guess	Pre Neural Networks	GoogLe Net (2015)

Le, et al., Building high-level features using large-scale unsupervised learning. ICML 2012
Szegedy et al., Going Deeper With Convolutions. CVPR 2015

Good ML = Generalization

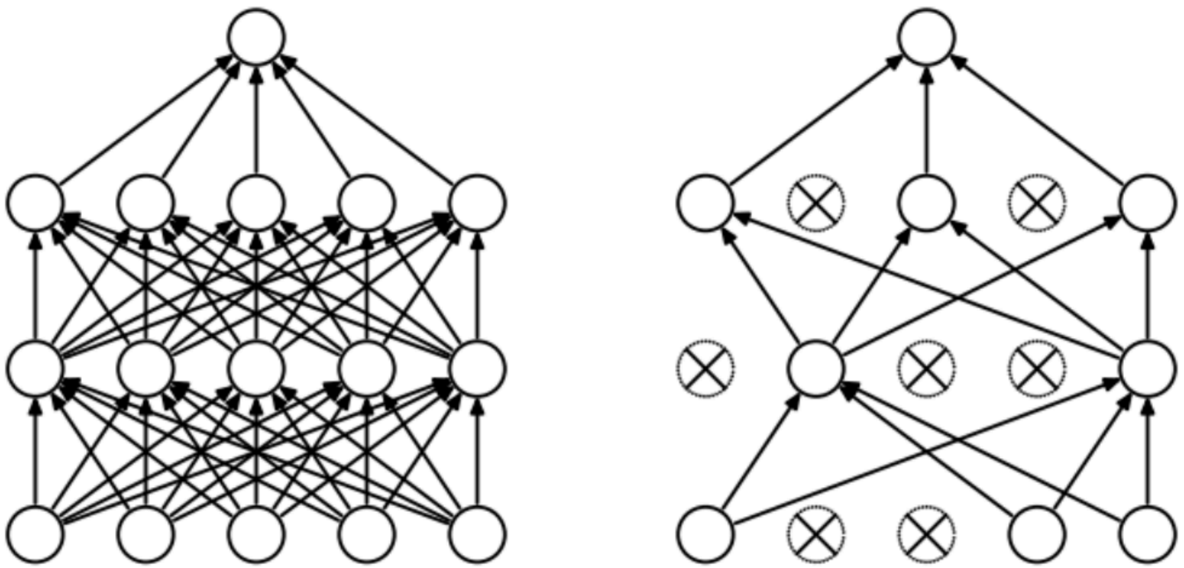
Goal of machine learning: build models that *generalize* well to predicting new data

Overfitting: fitting the training data too well, so we lose generality of model



- Polynomial on the right fits training data perfectly!
- Which would you rather use to predict a new data point?

Prevent overfitting?



(a) Standard Neural Net (b) After applying dropout.

Dropout (training technique)

When your model is training, randomly turn off your neurons with probability 0.5.

It will make your network more robust.

Making decisions?

Not everything is classification.

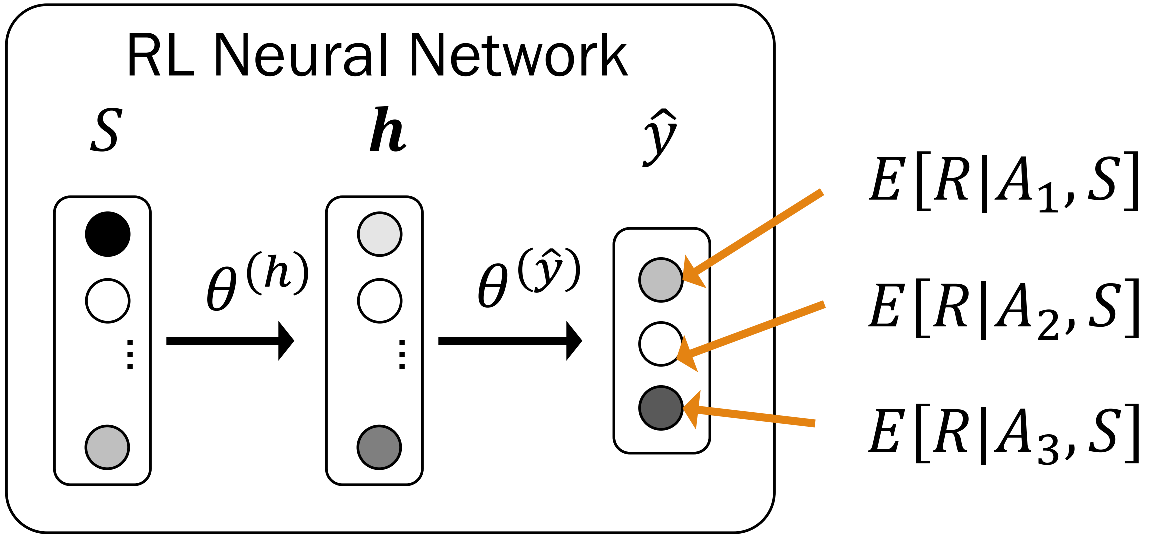
Deep Reinforcement Learning

Instead of having the output of a model be a probability, you make output an expectation.

http://cs.stanford.edu/people/karpathy/convnetjs/demo/rldemo.html

Deep Reinforcement Learning

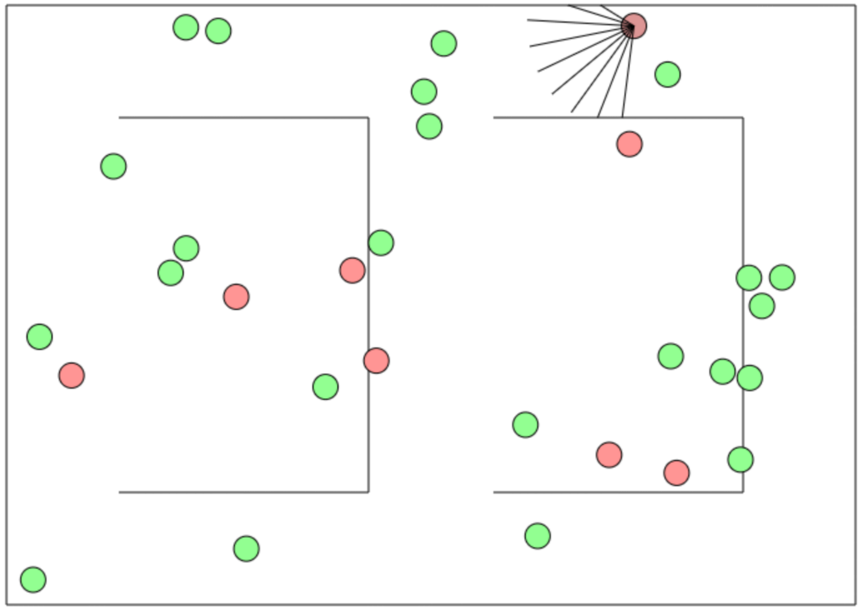
S: current state
R: reward
A_i: legal action



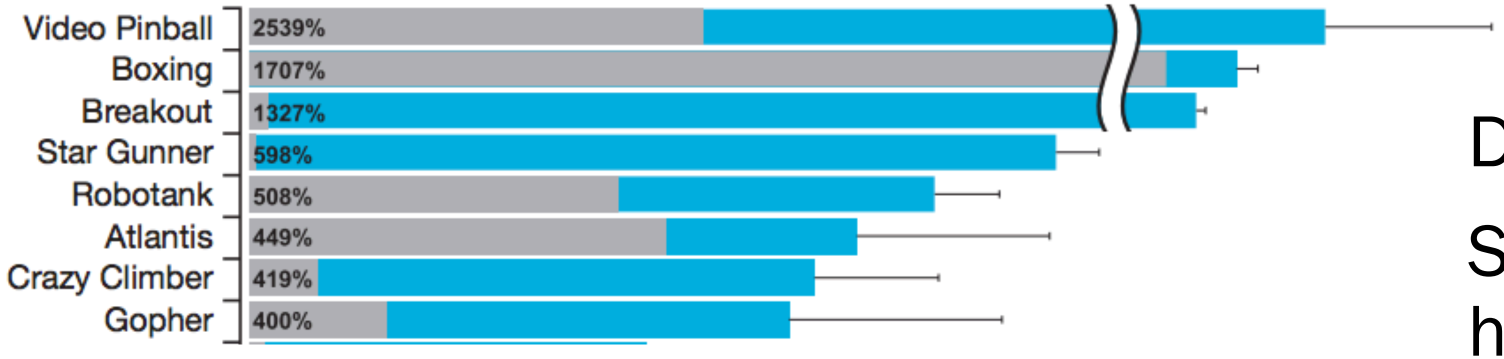
Input is a representation of current state, S

Output is expected reward for a given action

Deep Reinforcement Learning



http://cs.stanford.edu/people/karpathy/convnetjs/demo/rldemo.html



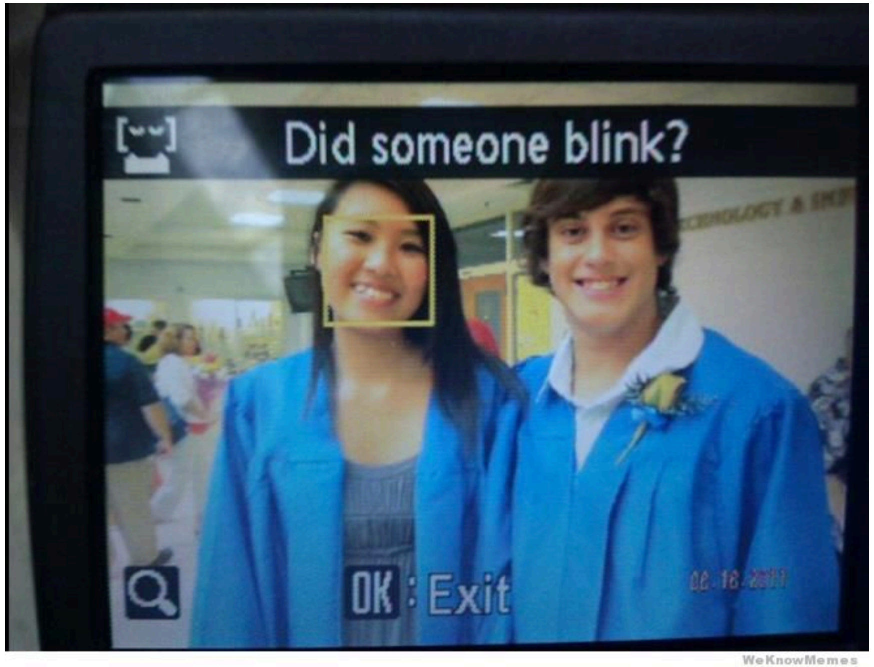
Deep Mind Atari Games

Score compared to best human

What’s missing?

Where is your data coming from?

Ethics and datasets



Sometimes machine learning feels universally unbiased.

We can even prove our estimators are “unbiased” (mathematically).

Google/Nikon/HP had biased datasets.

Should your data be unbiased?

Dataset: Google News

$$\overrightarrow{\text{man}} - \overrightarrow{\text{woman}} \approx \overrightarrow{\text{king}} - \overrightarrow{\text{queen}}$$

$$\overrightarrow{\text{man}} - \overrightarrow{\text{woman}} \approx \overrightarrow{\text{computer programmer}} - \overrightarrow{\text{homemaker}}.$$

Should our unbiased data collection reflect society's systemic bias?

Bolukbasi et al., Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings. NIPS 2016

How can we explain decisions?


If your task is image classification, reasoning about high-level features is relatively easy.

Everything can be visualized.

What if you are trying to classify social outcomes?

- Criminal recidivism
- Job performance
- Policing
- Terrorist risk
- At-risk kids

Ethics in Machine Learning is a whole new field. 🙂

Extra topic (optional)

Computer-generated random numbers

How does Python's `random()` work?

```python
import random
for i in range(5):
  # return next random floating point
  # in the range [0.0, 1.0)
  print(random.random())
```

```
0.9825275632982425
0.562507693641213‍9
0.1662498000287692
0.4845764780962842‍4
0.7937438138936983
```

Computers are deterministic, so we settle for **pseudo-randomness**: sequence of numbers that *looks* random but is deterministically generated

Linear congruential generator (LCG) is one of the simplest:

- Start with a seed number, X_0 (usually UNIX time)
- Next "random" number is:

$$X_n = (aX_n + c) \bmod m$$

- Effectiveness *very* sensitive to params $a, c,$ and m
- Sequence will eventually repeat

Python (today) uses the **Marsenne Twister:**

- Uses XOR/bitshifts
- 12 parameters
- 53-bit floating point numbers
- Sequence repeats every 2**19937-1 numbers

Generating a number according to a random distribution

`random.random()` generates a number $U \sim \text{Uni}(0,1)$.

Used on Problem Set 3 to simulate RVs:

- Bernoulli, Binomial, Geometric, and Negative Binomial
- *Approximated* Poisson/Exponential by using 60,000-step time interval

```
def simulateBernoulli(p):
    if random.random() < p:
        return 1
    return 0
```

One option to **precisely** generate a number from any distribution:

Inverse Transform Sampling

Inverse Transform sampling

Suppose we want to simulate a continuous random variable with cumulative distribution function F:

1. Let $U \sim \text{Uni}(0,1)$
2. Define $X = F^{-1}(U)$
3. Then X has CDF F.

F^{-1} is inverse of F, i.e., $F^{-1}(a) = b \Leftrightarrow a = F(b)$

Proof:

$$P(X \le x) = P(F^{-1}(U) \le x)$$

$$= P\big(U \le F(x)\big) \quad \text{(inverse)}$$

$$= F(x) \quad \text{(CDF of Normal: } P(U \le u) = u \text{ for } 0 < u < 1\text{)}$$

Simulating the Exponential distribution

1. Let $U \sim \text{Uni}(0,1)$
2. Define $X = F^{-1}(U)$
3. Then X has CDF F.

Suppose we want to generate the exponential distribution:

- $X \sim \text{Exp}(\lambda)$, with CDF $F(x) = 1 - e^{-\lambda x}$, where $x \geq 0$

Compute inverse: Let $F(X) = 1 - e^{-\lambda X} = U$ and solve for X.

$$e^{-\lambda X} = 1 - U$$
$$-\lambda X = \log(1 - U)$$
$$X = -\log(1 - U)/\lambda \longrightarrow X = F^{-1}(U) = -\log(1 - U)/\lambda$$

(Note: if $U \sim \text{Uni}(0,1)$, then $(1 - U) \sim \text{Uni}(0,1)$)

Simplify: $X = F^{-1}(U) = -\log(U)/\lambda$

If you can compute the inverse of the CDF, you can use Inverse Transform Sampling.
(Note: Normal RV doesn't have closed-form inverse; use Box-Muller)

Discrete inverse transform sampling

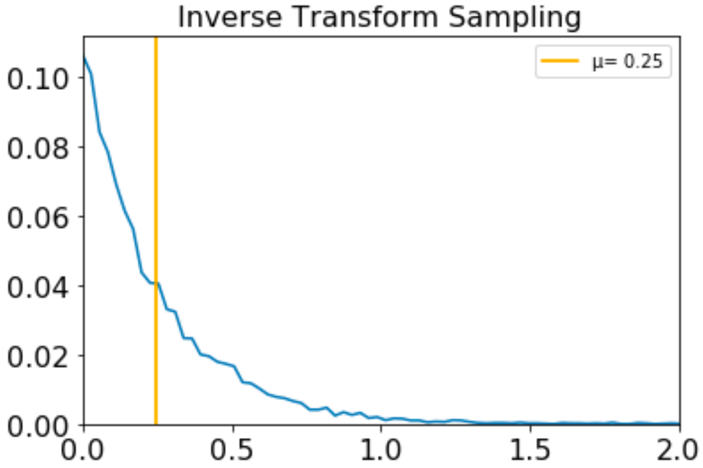
1. Let $U \sim \text{Uni}(0,1)$
2. Define $X = F^{-1}(U)$
3. Then X has CDF F.

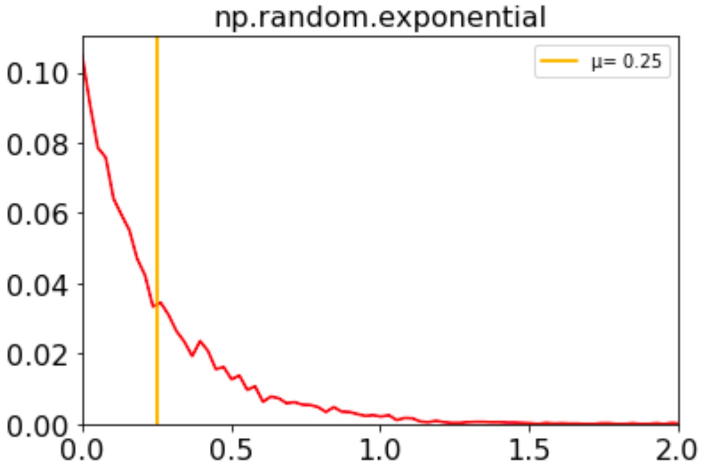
```
def inverseExpCDF(u, lamb):
    return -math.log(u)/lamb

def simulateExponential(lamb):
    u = random.random()
    return inverseExpCDF(u, lamb)
```

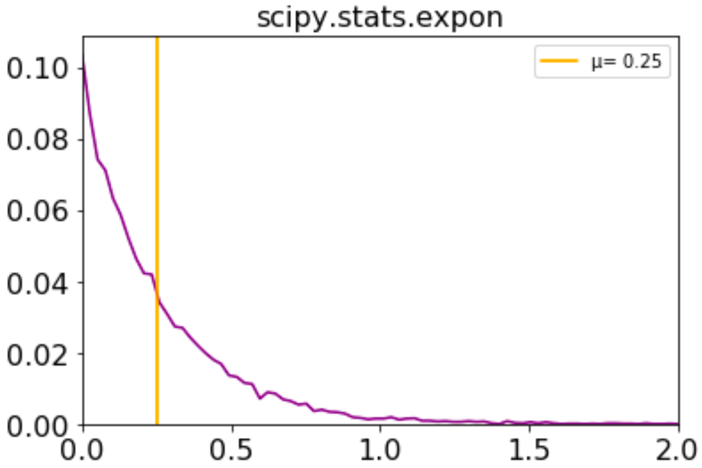
Pick a probability u

Return $x = F^{-1}(u)$

Simulate 10,000 trials and plot distribution:







Simulating the Poisson distribution

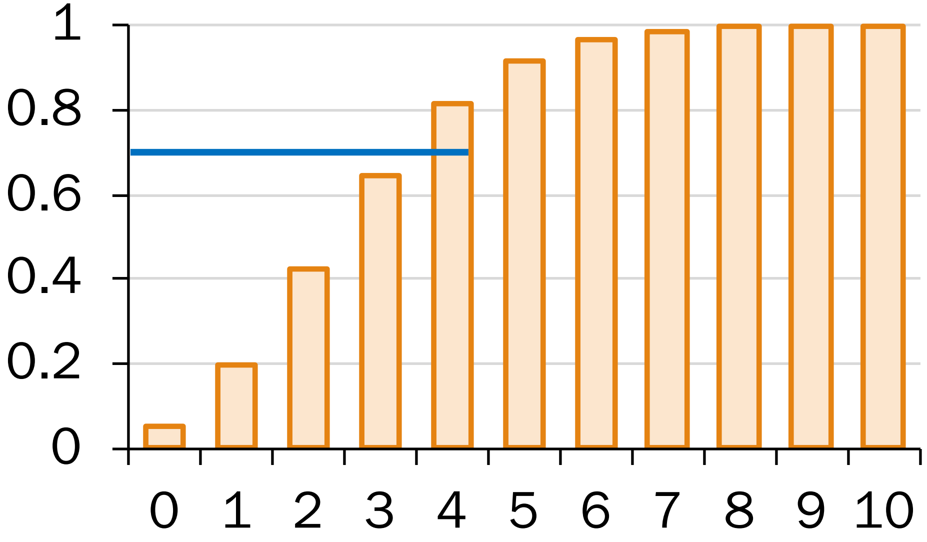
1. Let $U \sim \text{Uni}(0,1)$
2. Define $X = F^{-1}(U)$
3. Then X has CDF F.

Suppose we want to generate the Poisson distribution:

- $X \sim \text{Poi}(\lambda)$, with CDF $F(x) = \sum_{k=-\infty}^{x} P(X = k) = \sum_{k=0}^{x} \frac{\lambda^k}{k!} e^{-\lambda}$, where $x \geq 0$

Instead of computing the inverse:

1. Generate $U \sim \text{Uni}(0,1)$ $\quad u = 0.7$
2. Increase x until $F(x) \geq U$
3. Return that value of x $\quad x = 4$



Discrete inverse transform sampling

```
def simulatePoisson(lamb):
    u = random.random()
    x = 0
    CDF_so_far = pmfPoisson(lamb, x)
    while CDF_so_far < u:
        x += 1
        CDF_so_far += pmfPoisson(lamb, x)
    return x
```

while $\sum_{k=0}^{x} P(X = k) < u$

add $P(X = x + 1)$

Simulate 10,000 trials and plot distribution: