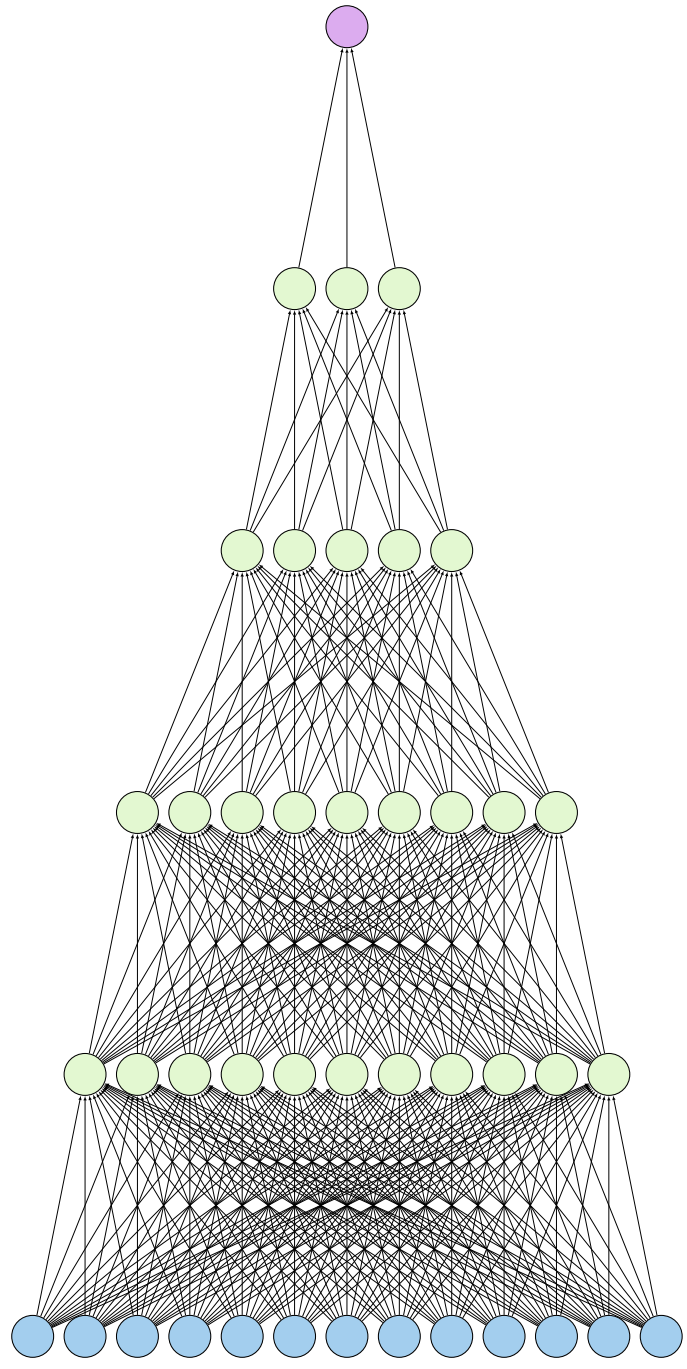
Foundations of Deep Learning for the Social Sciences

Christopher Urban

University of North Carolina at Chapel Hill



Introducing the Instructor

A short bio:

- I'm a final year Ph.D. candidate in Quantitative Psychology at the University of North Carolina at Chapel Hill
- I want to understand and predict human behavior
- Currently fascinated by neural networks and deep learning

Contact info:

- Website: `cjurban.github.io`
- Email: `cjurban@live.unc.edu`

Introducing the Attendees

Let's go around and share:

- Your name and institution
- Your research interests
- Preferred statistical software package(s) and/or programming language(s)
- Your reason(s) for enrolling in this workshop
- A fun fact about you(?)

Introducing the Workshop

Overarching Goal:

Learn how to use deep learning for predictive and explanatory modeling of human behavior

Workshop content split into three chunks:

Day 1, morning: Intro and foundational concepts

Day 1, afternoon: Foundational models

Day 2: Connections to psychometrics

First Day Learning Outcomes

In the morning, you will learn:

- Some achievements of modern deep learning
- The definition of a deep learning model and some connections to traditional statistical models
- The basics of how deep learning models are fitted including the backpropagation algorithm, automatic differentiation, and stochastic gradient-based optimization

First Day Learning Outcomes, Continued

In the afternoon, you will learn:

- The definition of the multilayer perceptron and applications to cross-sectional data
- The defintion of the recurrent neural network (RNN) and applications to longitudinal data
- Why RNNs struggle with long-term dependencies and solutions based on gated architectures
- The definition of the convolutional neural network and applications to image data
- Universal approximation properties of each model type
- The basics of tuning neural network hyperparameters
- How to evaluate and compare fitted deep learning models

Second Day Learning Outcomes

You will learn:

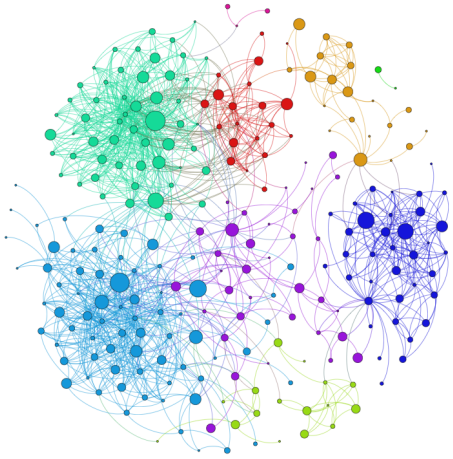
- How to fit structural equation models using backpropagation and stochastic gradient-based optimization
- The defintion of the autoencoder and applications
- The basics of non-amortized and amortized variational inference
- How to use variational methods to fit complex latent factor models

Learning Outcomes Checkpoint

- Let's briefly discuss the history of deep learning and some of its recent achievements.

The Rise of Social Data Science: The Data Boom

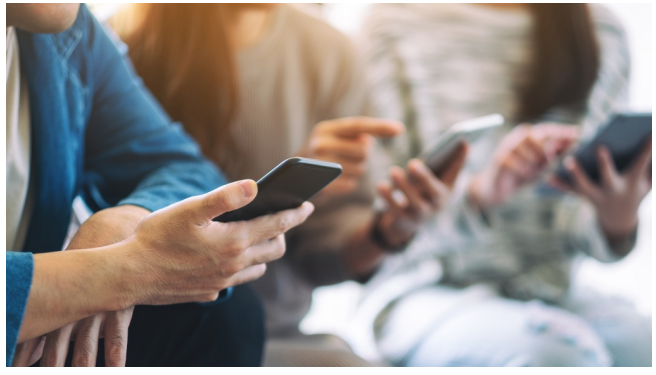
Over the past 15–20 years, progressively larger scale and more complex behavioral data has become available:



Social media data

Wearable sensor data

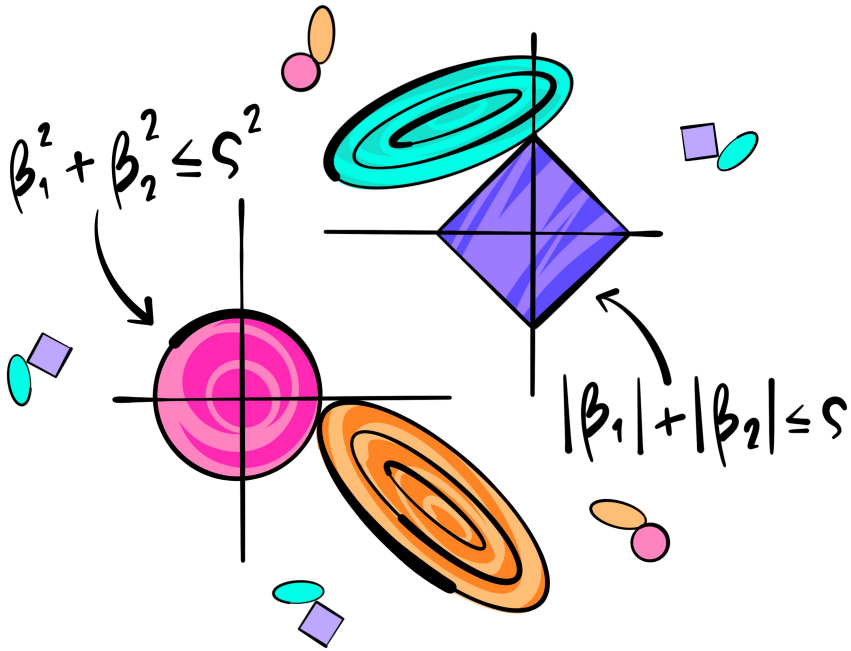
Electronic health records



Smartphone data

The Rise of Social Data Science: The Methods Boom

Simultaneously, interest has grown in using statistical and machine learning methods to model these data:

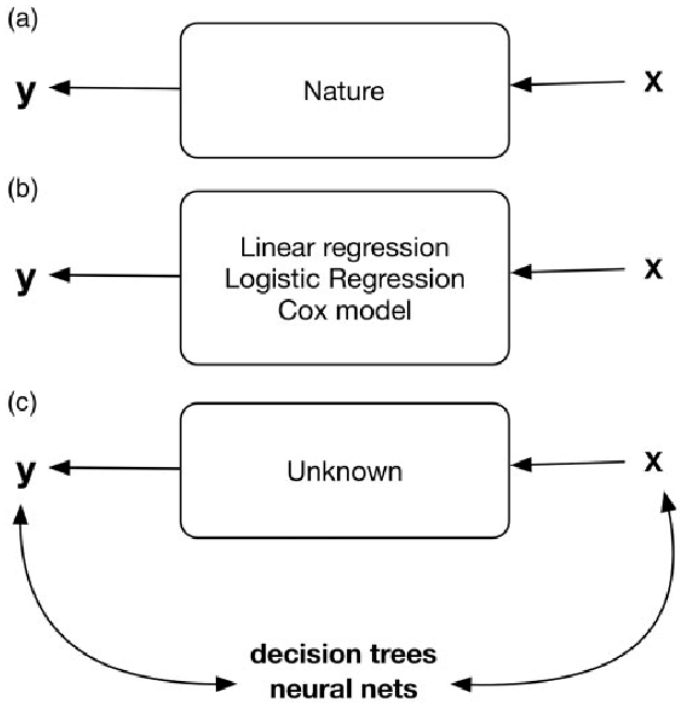


Regularization

- For (factor, group, and/or network) structure — Sun et al. (2016), Huang, Chen, & Weng (2017), Epskamp & Fried (2018), Scharf & Nestler (2019).
- For estimation stability in high dimensions — Jacobucci, Grimm, & McArdle (2016).
- For differential item functioning — Belzak & Bauer (2020), Belzak (2021).

The Rise of Social Data Science: The Methods Boom

Simultaneously, interest has grown in using statistical and machine learning methods to model these data:



Predictive modeling

- As a component of risk detection — e.g., for suicide, self harm, manic/depressive episodes — Yarkoni & Westfall (2017), Walsh, Ribeiro, & Franklin (2017).
- To combat the replication crisis — e.g., via cross-validation — Cudeck & Browne (1983), Koul, Becchio, & Cavallo (2018).

The Rise of Social Data Science: Thoughts

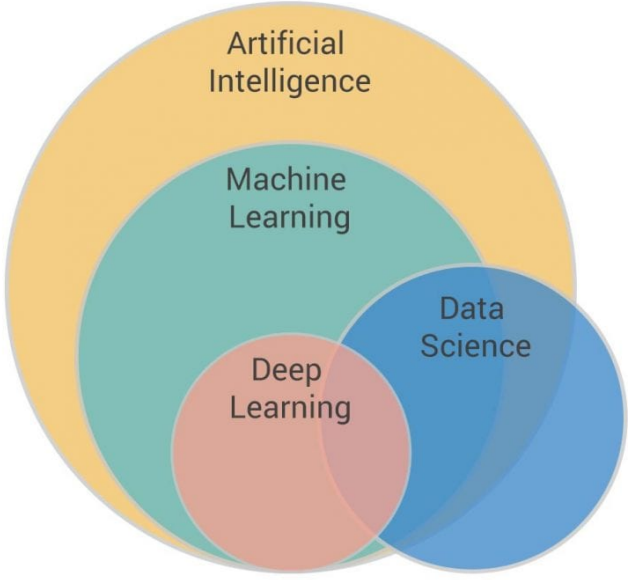
Important themes:

- Larger scale — more people and more variables
- More complex — e.g., not-at-random missingness, network structure, lagged relationships, audiovisual data
- Recognition that most relationships are nonlinear
- Recognition that models should generalize to new data

To summarize, data and modeling procedures appear to be catching up with the long-suspected notion that *human systems are complex*.

The Rise of Deep Learning

Deep learning (DL) researchers have long been interested in modeling complex phenomena.



DL overlaps with statistics, but the fields typically have different goals.

Statistics aims to help people make optimal decisions given assumptions and data.

- Statistical models almost always have interpretable parameters.

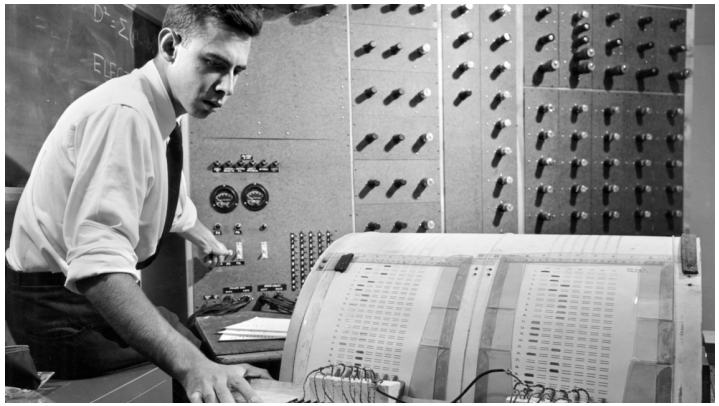
DL is a subfield of *artificial intelligence* — i.e., the study of autonomous agents.

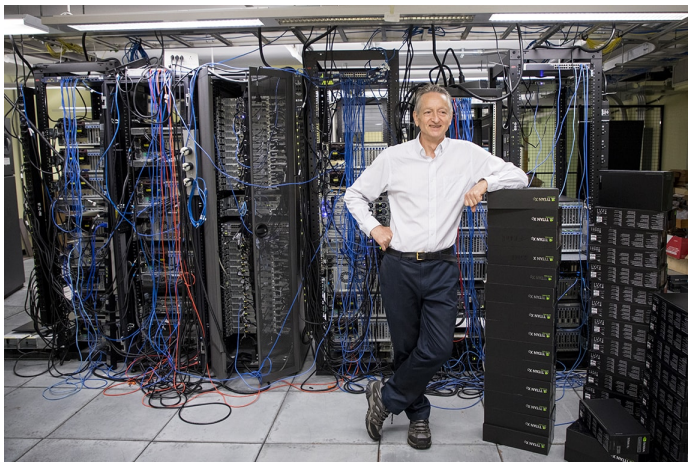
- Models don't need interpretable parameters — just need to interact with the environment effectively.

A Brief History of Deep Learning: The Early Years

1940s–1960s. Initial work inspired by biology — goal was to build models of neurons.

- McCulloch & Pitts (1943), Rosenblatt (1958)





1980s–1990s. Popularization of backpropagation-based model fitting.

- Rumelhart, Hinton, & Williams (1986).

Models developed for images and sequences.

- LeCun et al. (1989), Hochreiter & Schmidhuber (1997).

Universal approximation theorems proved.

- Cybenko (1989), Siegelmann & Sontag (1992).

A Brief History of Deep Learning: Later Years

2006–2011. Some of the first really "deep" models for computer vision.

- Hinton (2007), Ranzato et al. (2007), Bengio et al. (2007).

2012. The highly influential AlexNet model for image recognition was published.

- Krizhevsky, Sutskever, & Hinton (2007).

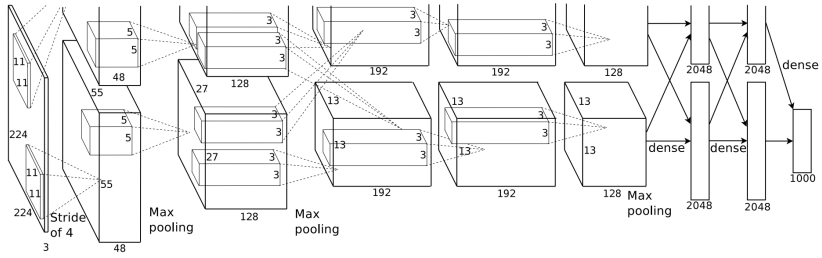


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

2014–Present The generative modeling boom. Development of generative adversarial networks, variational autoencoders, and more.

- Goodfellow et al. (2014), Kingma & Welling (2014), Rezende, Mohamed, & Wierstra (2014).

What Can Deep Learning Do?

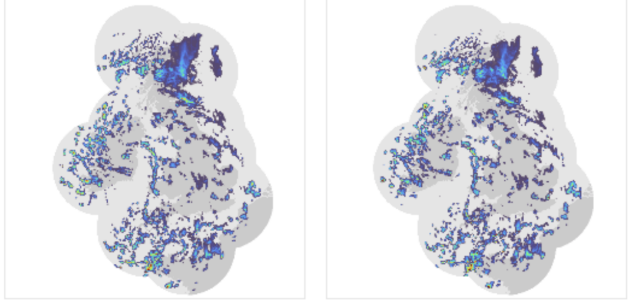
DL excels at modeling complex processes:

Game playing. Typically uses a method called *reinforcement learning* wherein model learns by playing millions of times.

- Examples: AlphaGo for Go, deep-Q-nets for Atari 2600.

Weather forecasting. Based on generating fake radar data.

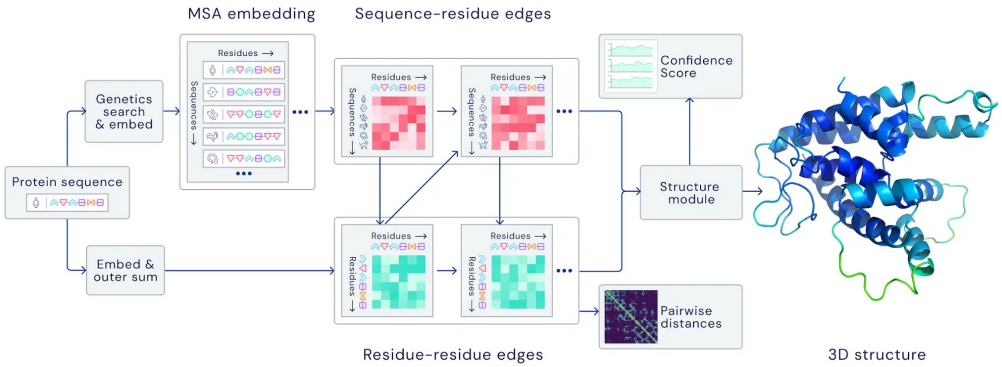
- Example: Nowcasting precipitation.



What Else Can Deep Learning Do?

Predict protein structure. Based on an *attention mechanism* that pays attention to most important parts of data.

- Example: AlphaFold.



Natural language processing. Typically based on models for sequences.

- Examples: Google Translate, GPT-3 for generating realistic text.

```
Q: What is your favorite animal?
A: My favorite animal is a dog.

Q: Why?
A: Because dogs are loyal and friendly.

Q: What are two reasons that a dog might be in a bad mood?
A: Two reasons that a dog might be in a bad mood are if it is hungry or if it is hot.

Q: How many eyes does a giraffe have?
A: A giraffe has two eyes.

Q: How many legs does a frog have?
A: A frog has four legs.

Q: Are there any animals with three legs?
A: No, there are no animals with three legs.

Q: Why don't animals have three legs?
A: Animals don't have three legs because they would fall over.
```

And Last But Not Least, Computer Vision

- Vehicle safety and self-driving — e.g., Tesla, Waymo
- Facial recognition — e.g., cell phones, mass surveillance
- Reverse image search — e.g., Google Lens
- Generating realistic fake images and movies — e.g., Deepfakes, DALL-E, Stable Diffusion

Some Natural Questions

Why hasn't DL been more widely adopted in the social sciences?

- Not enough data?
- Available data doesn't have the right structure?
- Interpretation difficulties?
- Limited methodological training available?

Can we augment traditional statistical models used in the social sciences with DL and retain the benefits of both?

- More on this tomorrow!

Learning Outcomes Checkpoint

- We briefly discussed the history of deep learning and some of its recent achievements.
- Now, let's define what it means for a model to be a *deep learning model*.

Defining "Deep Learning Model"

Two different (but related) definitions typically used.

1. **Computational definition.** The model passes the predictors through a sequence of processing steps to predict the outcome.
   - E.g., linear regression models pass the predictors through one processing step — multiply by weights and add intercepts
   - Deep learning models pass the predictors through multiple such processing steps

2. **Probabilistic definition.** The model assumes that the data were generated by building complicated concepts from simple concepts.
   - E.g., in images, objects made from parts, parts made from edges, edges made from pixels
   - E.g., in social sciences, districts made from schools, schools made from classrooms, classrooms made from students

The Computational Definition: Machine Learning Models

Machine learning models look like this:

$$\hat{y} = f_{\theta}(\mathbf{x})$$

where $\boldsymbol{\theta}$ are parameters.

Example 1: Linear regression model

$$\hat{y} = \mathbf{w}^{\top}\mathbf{x} + b$$

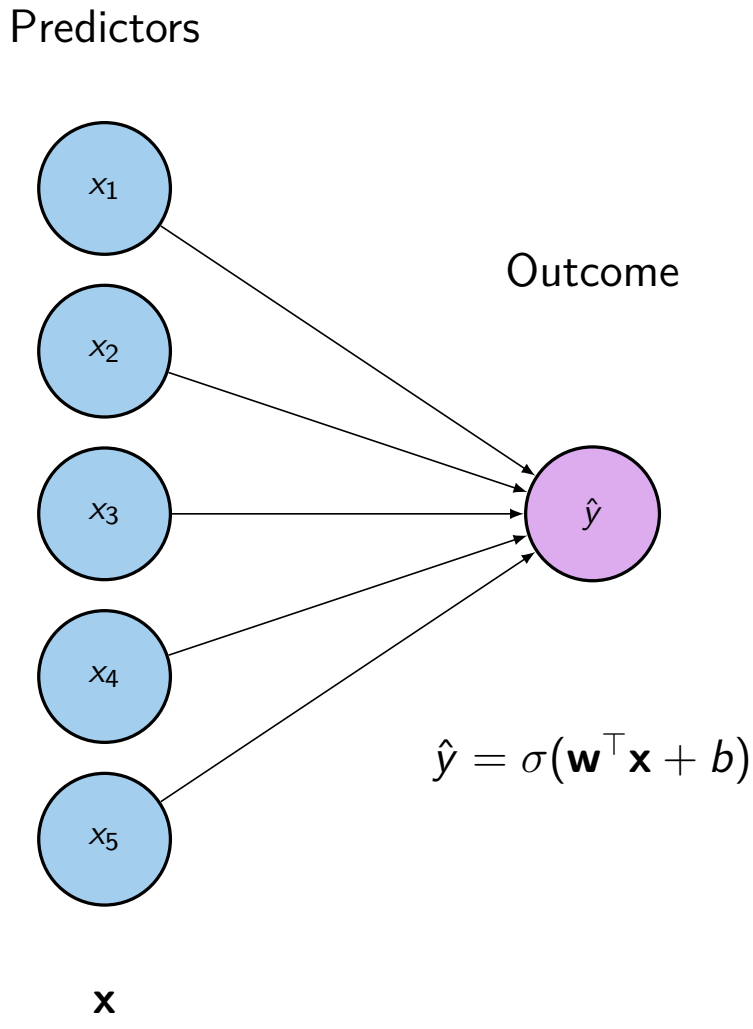
where $\mathbf{w}$ are weights and b is an intercept.

Example 2: Logistic regression model

$$\hat{y} = \sigma(\mathbf{w}^{\top}\mathbf{x} + b)$$

where $\sigma(z) = 1/(1 + \exp[-z])$ is the inverse logistic link function.

Logistic Regression Schematic Diagram



The Computational Definition: Deep Learning Models

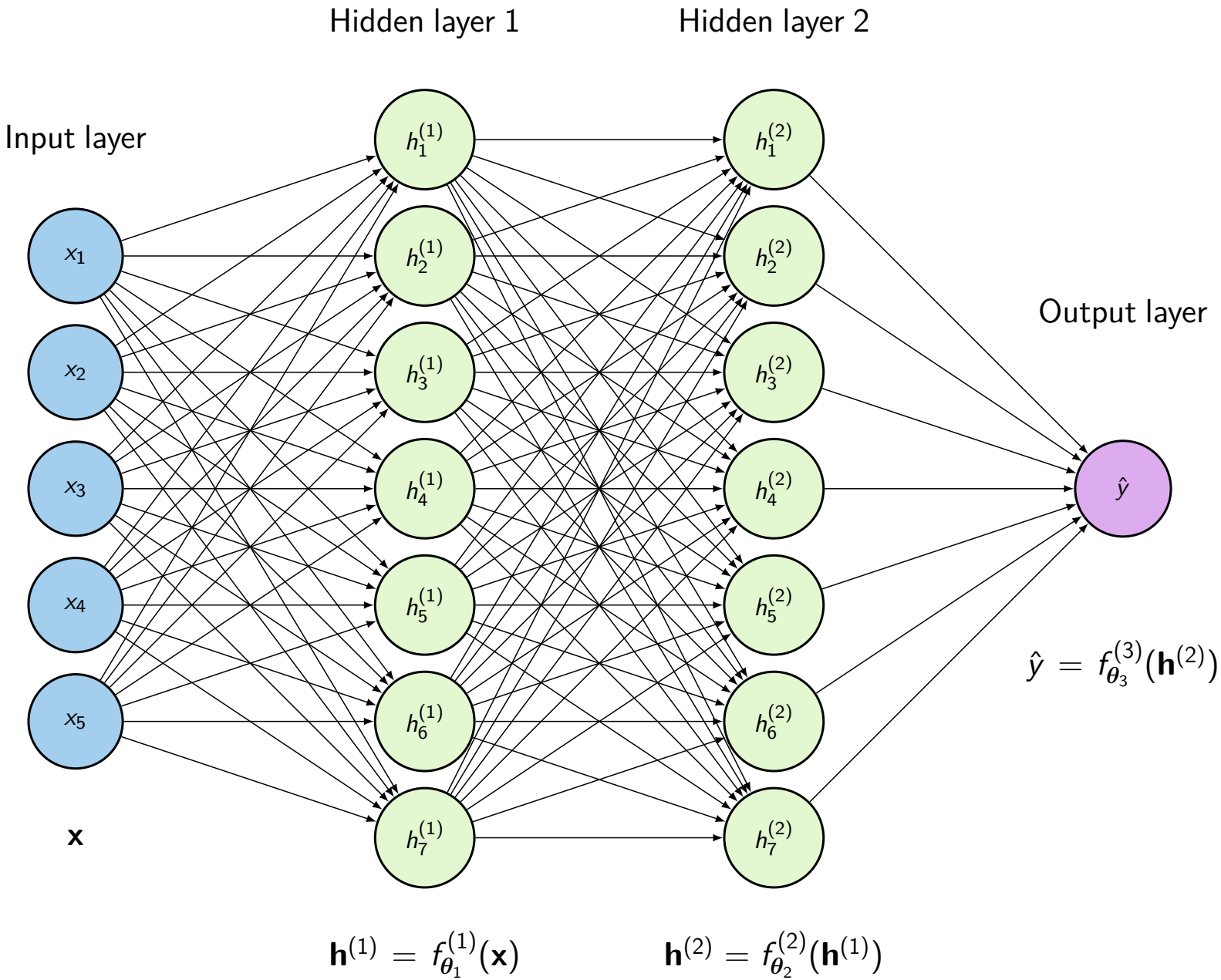
Deep learning models look like this:

$$\hat{y} = f_{\boldsymbol{\theta}_L}^{(L)}\left(f_{\boldsymbol{\theta}_{L-1}}^{(L-1)}\left(\ldots f_{\boldsymbol{\theta}_3}^{(3)}\left(f_{\boldsymbol{\theta}_2}^{(2)}\left(f_{\boldsymbol{\theta}_1}^{(1)}(\mathbf{x})\right)\right)\ldots\right)\right)$$

where $\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3, \ldots, \boldsymbol{\theta}_{L-1}, \boldsymbol{\theta}_L$ are parameters.

Each $f_{\boldsymbol{\theta}_l}^{(l)}(\cdot)$ is called a *layer*.

Deep Learning Model Schematic Diagram



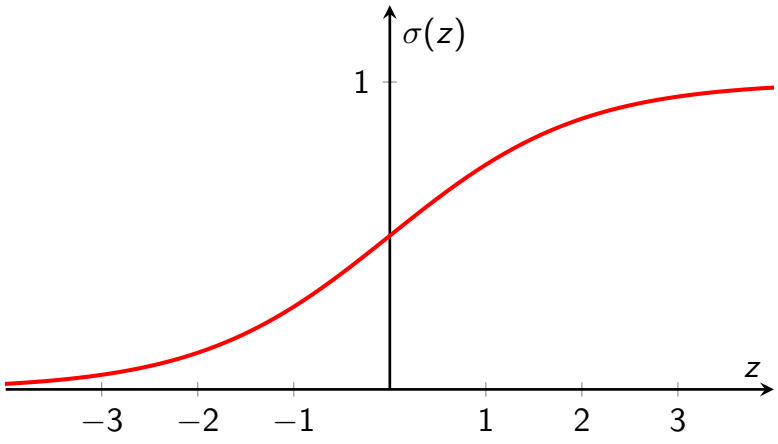
Deep Learning Model Example: Multilayer Perceptron

One of the most basic DL models, the *multilayer perceptron* (MLP), is just a "stack" of generalized linear models.

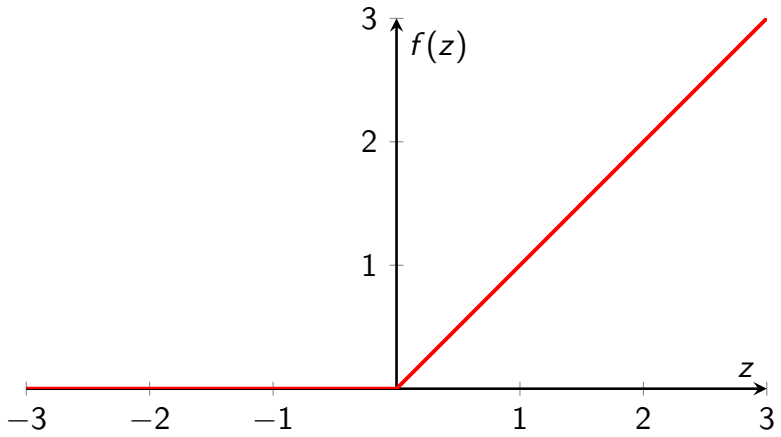
Example 3: Multilayer perceptron

$$\mathbf{h}^{(l)} = f^{(l)}(\mathbf{W}_l \mathbf{h}^{(l-1)} + \mathbf{b}_l), \qquad l = 1, \ldots, L,$$

where $\mathbf{h}^{(0)} = \mathbf{x}$, $\mathbf{h}^{(L)} = \hat{y}$, $\mathbf{W}_l$ is a weight matrix, $\mathbf{b}_l$ are intercepts, and $f^{(l)}(\cdot)$ is an activation (i.e., inverse link) function.



Sigmoid activation function:
$\sigma(z) = 1/(1 + \exp[-z])$
This is just the inverse logistic link!

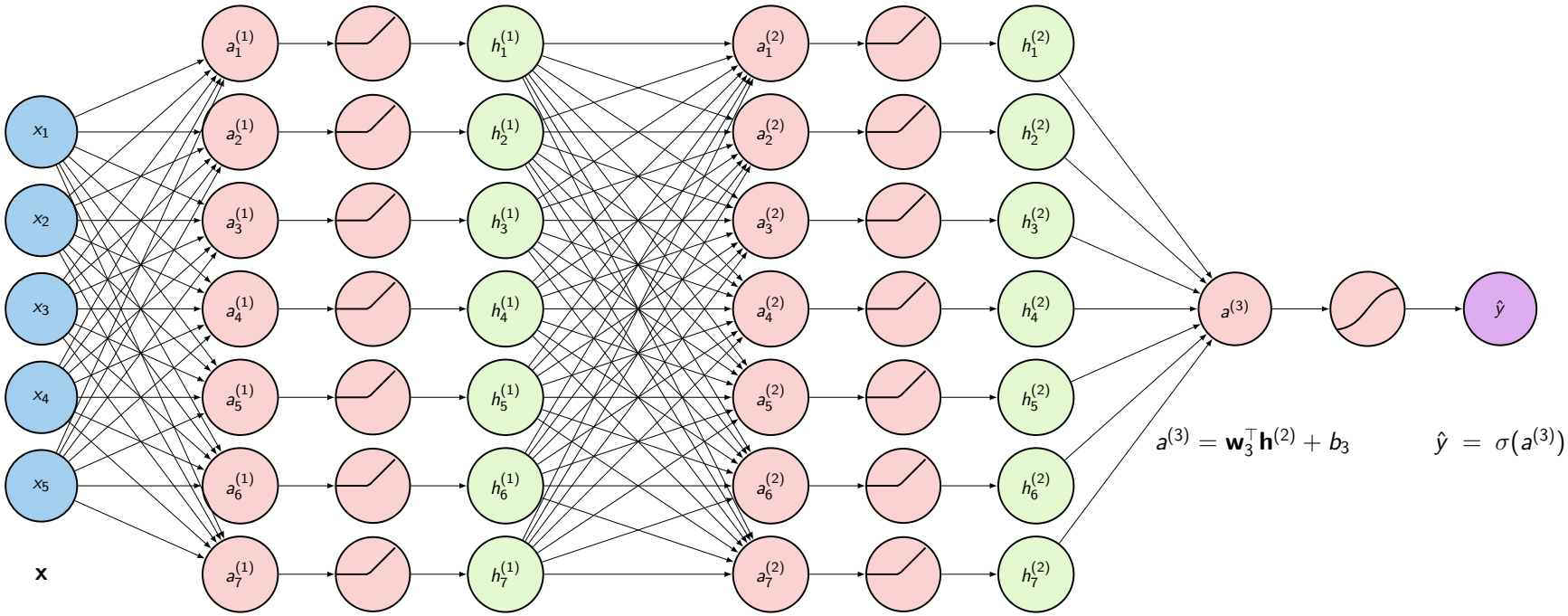


ReLU activation function:
$f(z) = \max(z, 0)$

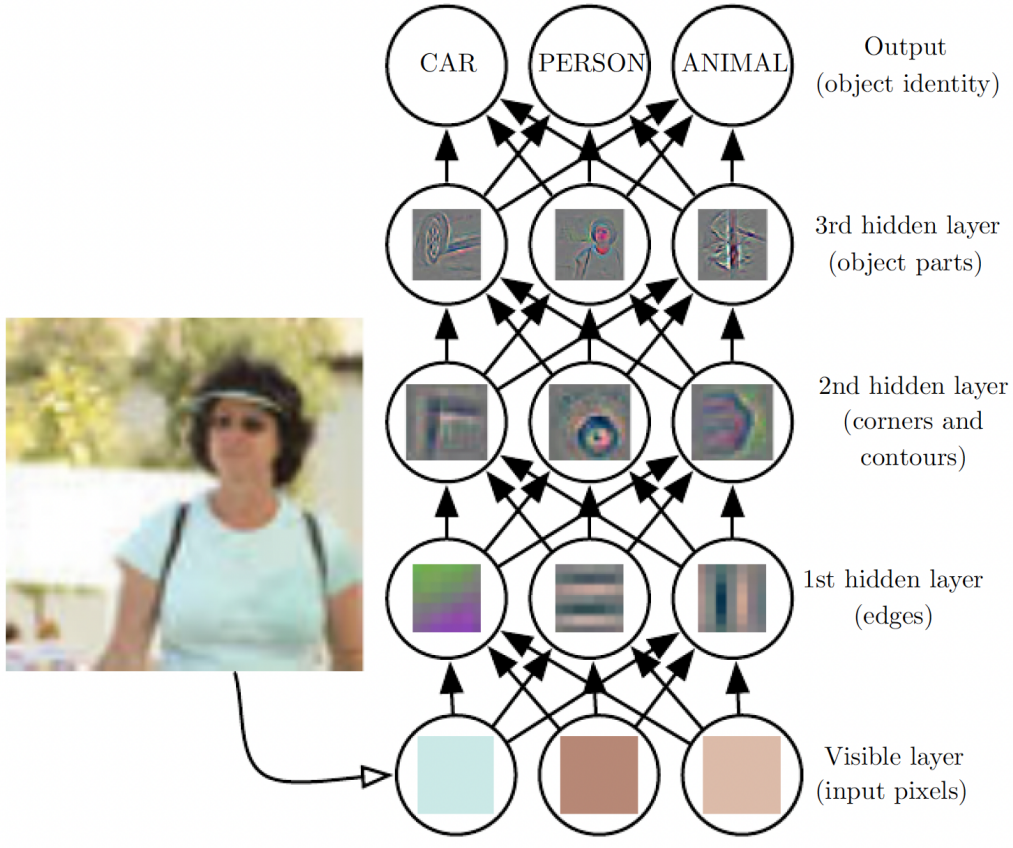
Multilayer Perceptron Schematic Diagram

We might use the model below to predict a binary outcome.

I've expanded the schematic to show additional internal computations.



The Probabilistic Definition



Rather than processing the data in a sequence of steps, the model includes *multiple layers of latent variables*.

This approach aims to capture progressively more and more complicated concepts as the layers progress.

This is a less commonly used definition — and often, models are deep in both senses!

Learning Outcomes Checkpoint

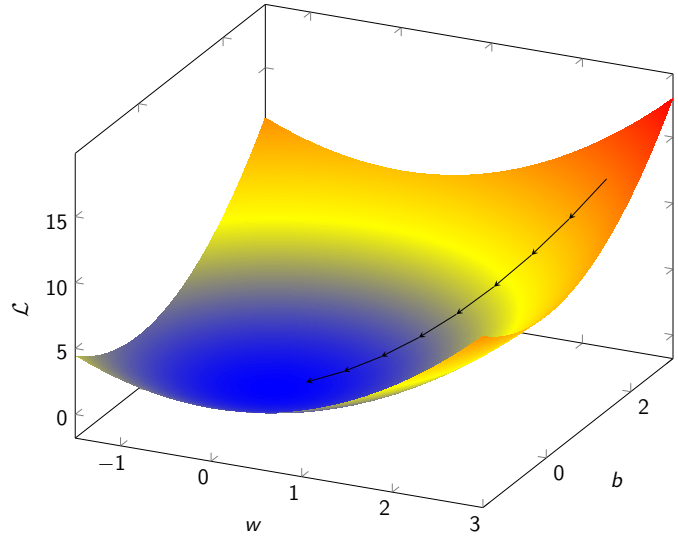
- We've defined what it means for a model to be a deep learning model.
- Now let's discuss how to fit deep learning models. You will get a basic understanding of:[1]
  - how the *backpropagation algorithm* works; and
  - how *stochastic gradient methods* work.

[1] Some of these slides will look fairly math-y. To reassure you — my goal is to describe these topics to you intuitively and to give you the equations in case you want to go back through these slides yourself.

How Do We Estimate DL Model Parameters?

1. Pick a loss function $\mathcal{L}$ — intuitively, this measures how accurate our model is
   - Squared error loss $\mathcal{L} = \frac{1}{2}(y - \hat{y})^2$
   - Logistic loss $\mathcal{L} = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$
2. Find the derivative $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$ — intuitively, this tells us how to change the parameters to make the model more accurate
3. Update the parameters using stochastic gradient method

DL models can be huge and complex, so researchers use automatic procedures to find derivatives and stochastic gradient methods that work well in a variety of settings.

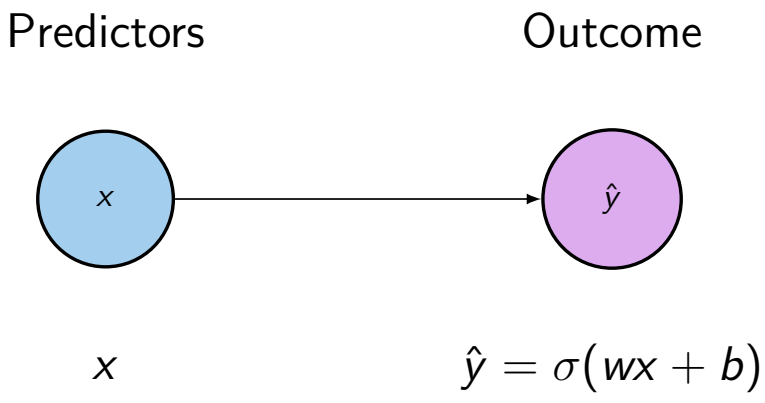


Understanding DL Concepts Through Logistic Regression

We will work through fitting a logistic regression model using a deep learning approach.

We will begin with the univariate case, then move on to the multivariate case.

The univariate logistic regression model looks like this:



For example, $x \geq 0$ might be the number of hours slept last night, and $y \in \{0, 1\}$ might be whether one experiences a depressive episode the following day.

Logistic Regression Loss Function

The logistic regression loss function is the logistic loss:

$$\begin{aligned}\mathcal{L} &= -[y \log(\hat{y}) + (1-y)\log(1-\hat{y})] \\ &= \begin{cases} -\log(\hat{y}) & \text{if } y = 1 \\ -\log(1-\hat{y}) & \text{if } y = 0. \end{cases}\end{aligned}$$

The intuition here is that we want to predict the probability of y being 1 or 0:

$$\begin{aligned}\Pr(y = 1 \mid x) = \hat{y} &= \sigma(wx + b), \\ \Pr(y = 0 \mid x) = 1 - \hat{y} &= 1 - \sigma(wx + b).\end{aligned}$$

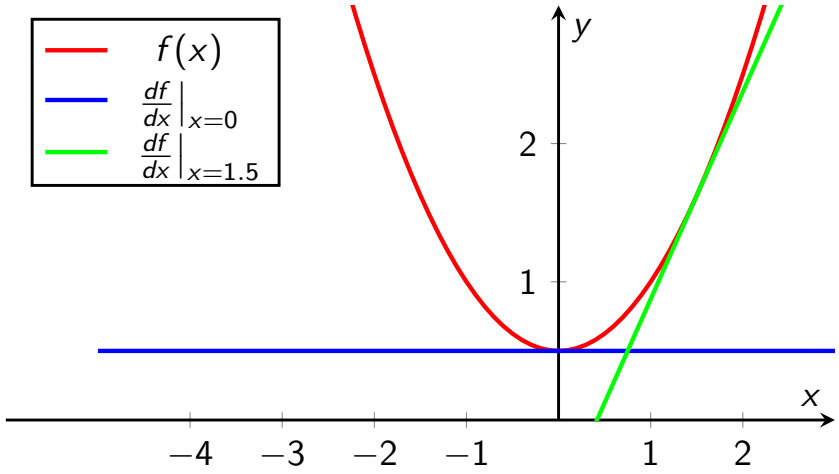
When $y = 1$, we want to increase $\Pr(y = 1 \mid x)$; when $y = 0$, we want to increase $\Pr(y = 0 \mid x)$.

Minimizing the Loss by Taking Derivatives

To fit the model, we want the value of $\boldsymbol{\theta} = (w, b)$ that minimizes the loss $\mathcal{L}$ — intuitively, this produces the most accurate model.

To do this, we will need the derivative $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}} = (\frac{\partial \mathcal{L}}{\partial w}, \frac{\partial \mathcal{L}}{\partial b})$.

Recall (maybe): The derivative evaluated at a point $x = x_0$ is the best linear approximation to the function at x_0; it tells us how fast the function is changing at x_0.



We'll first compute the derivatives analytically, then show how to compute them using an efficient algorithm called *backpropagation*.

A Couple of Calculus Facts

To get analytic derivatives for logistic regression, we need the following facts from calculus class.

Chain rule:

$$\frac{df(g(x))}{dx} = \frac{df}{dg}\frac{dg}{dx}.$$

Sigmoid derivative:

$$\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x)).$$

Analytic Derivatives for Logistic Regression

The univariate logistic regression loss written in full is:

$$\mathcal{L} = -[y \log(\sigma(wx + b)) + (1 - y) \log(1 - \sigma(wx + b))].$$

This is how you would compute $\frac{\partial \mathcal{L}}{\partial w}$ in calculus class:

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial w} &= -\frac{\partial}{\partial w}[y \log(\sigma(wx + b)) + (1 - y) \log(1 - \sigma(wx + b))] \\
&= -y \frac{1}{\sigma(wx + b)} \sigma(wx + b)(1 - \sigma(wx + b))x \\
&\quad -(1 - y) \frac{1}{1 - \sigma(wx + b)} \sigma(wx + b)(1 - \sigma(wx + b))x \\
&= -yx(1 - \sigma(wx + b)) - (1 - y)x\sigma(wx + b).
\end{aligned}$$

We can compute $\frac{\partial \mathcal{L}}{\partial b}$ similarly:

$$\frac{\partial \mathcal{L}}{\partial b} = -y(1 - \sigma(wx + b)) - (1 - y)\sigma(wx + b).$$

Analytic derivatives can be computationally inefficient due to the presence of repeated expressions (here, $\sigma(wx + b)$).

Analytic Derivatives for Logistic Regression

Instead of writing out the full analytic derivatives, we can write things more elegantly using the chain rule.

Computing the loss:

$$z = wx + b$$
$$\hat{y} = \sigma(z)$$
$$\mathcal{L} = -[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

Computing the derivatives:

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = -\frac{y}{\hat{y}} - \frac{1 - y}{1 - \hat{y}}$$
$$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \sigma(z)(1 - \sigma(z))$$
$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial \mathcal{L}}{\partial z} x$$
$$\frac{\partial \mathcal{L}}{\partial b} = \frac{\partial \mathcal{L}}{\partial z}$$

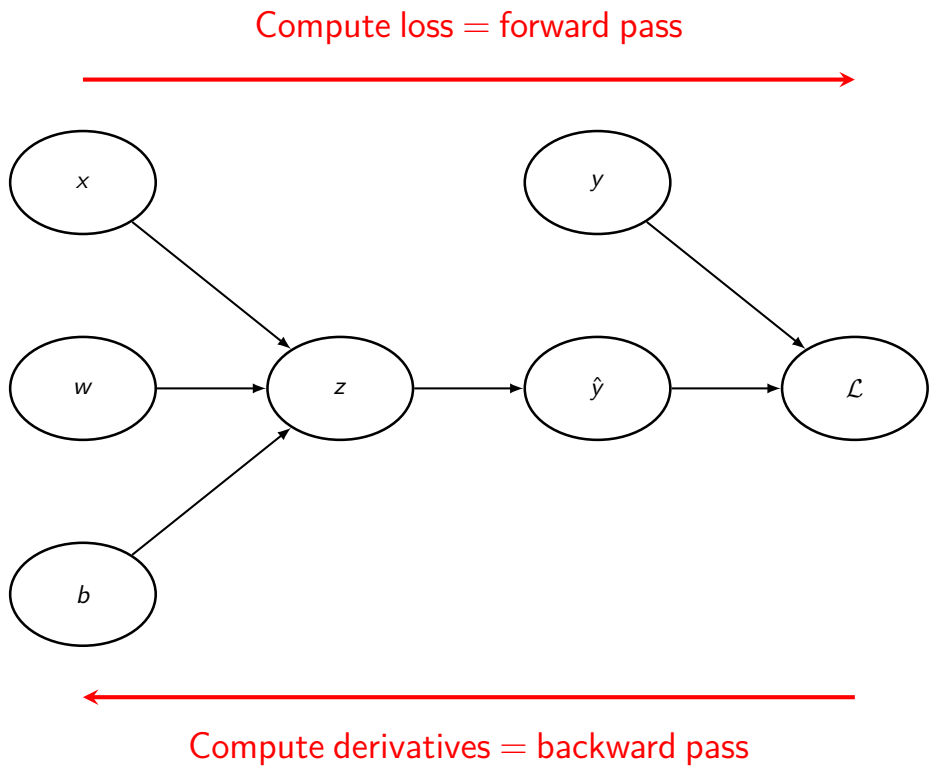
If we compute the value of $\frac{\partial \mathcal{L}}{\partial z}$ once, we can store it and reuse it to calculate both $\frac{\partial \mathcal{L}}{\partial w}$ and $\frac{\partial \mathcal{L}}{\partial b}$, eliminating repeated expressions.
This is the basic idea of backpropagation!

Backpropagation: It's About Computation Graphs

Backpropagation relies on the idea of a *computation graph*.

Basic idea: Computation graphs are directed acyclic graphs (DAGs) whose nodes represent values and whose edges represent operations.

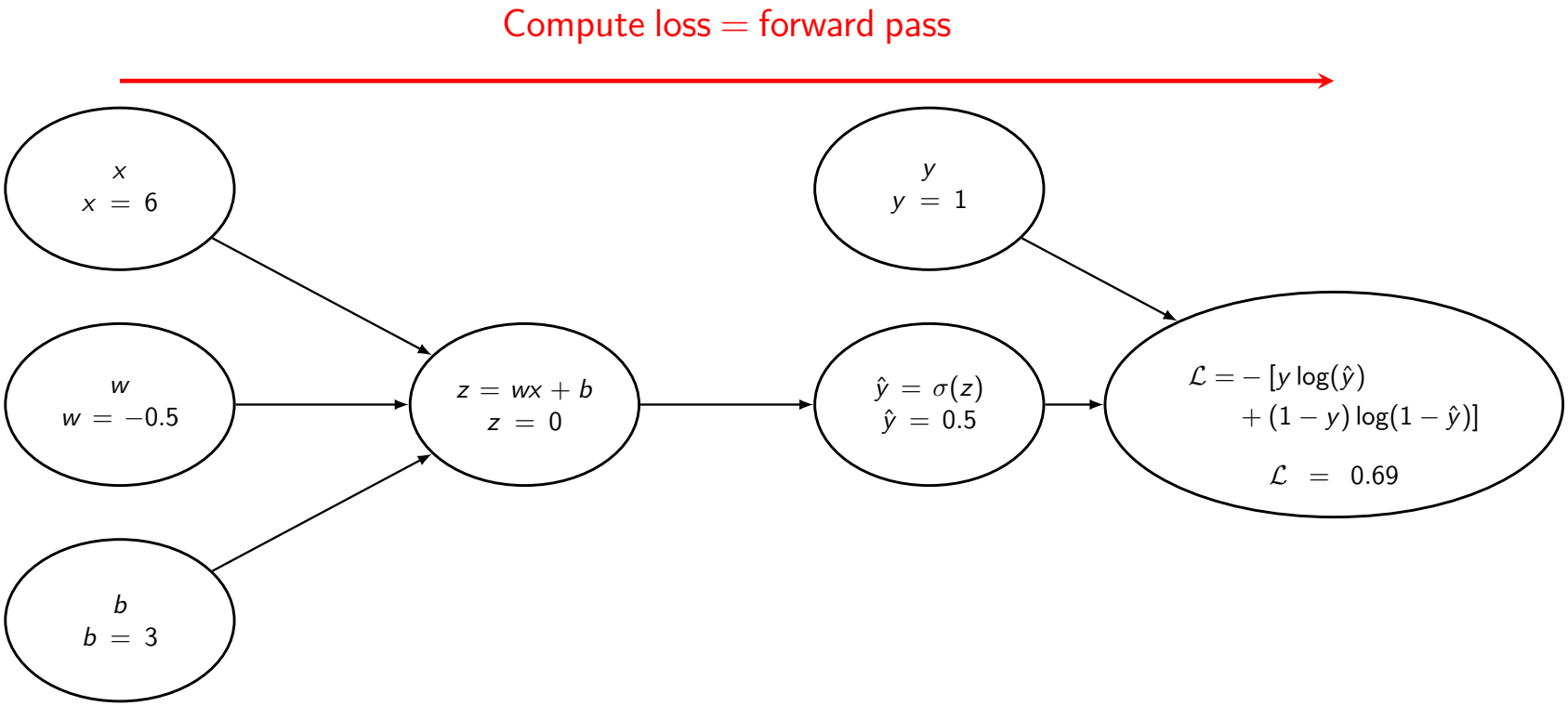
For logistic regression, the computation graph looks like this:



The Forward Pass: Logistic Regression

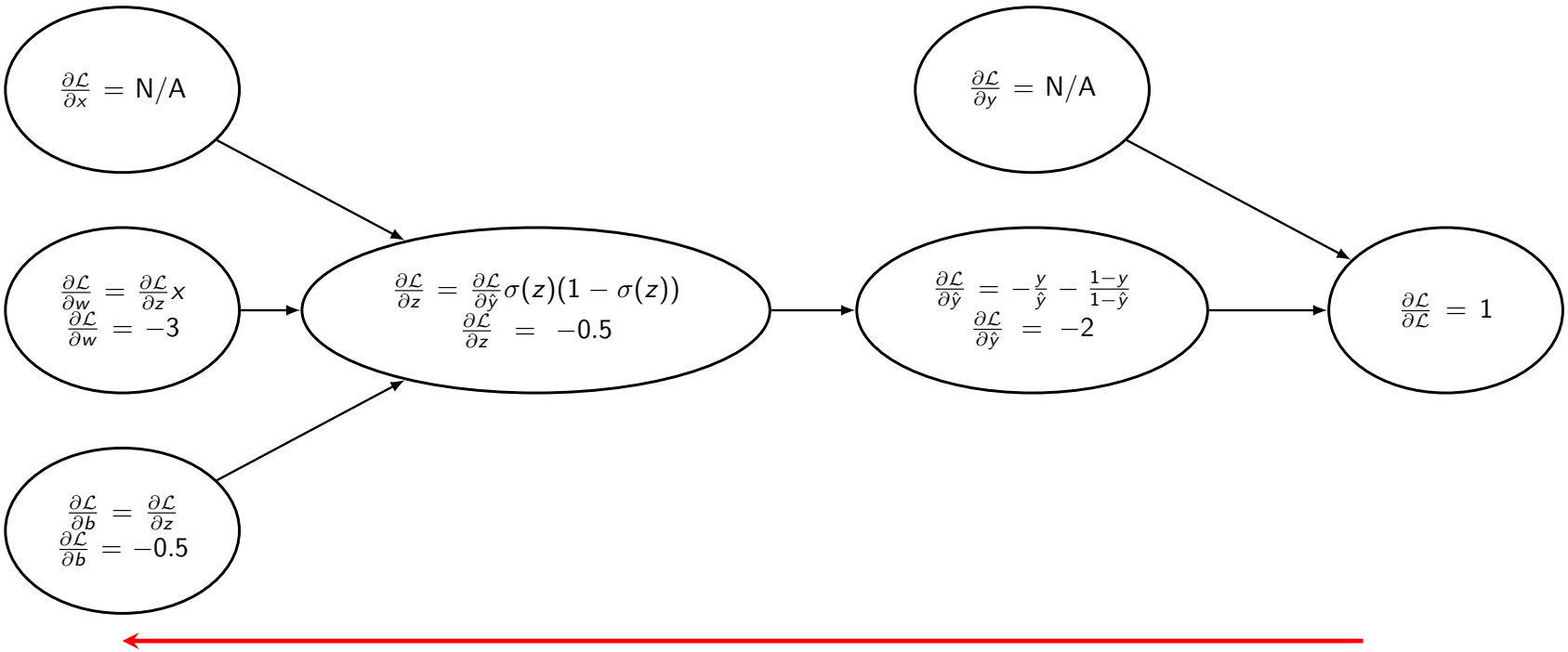
Recall that x is number of hours slept last night and y is whether one experiences a depressive episode the following day.

Given specific values of x, w, and b, backpropagation starts by making a forward pass through the computation graph to compute the loss value.



The Backward Pass: Logistic Regression

Backpropagation then makes a backward pass — i.e., it *backpropagates* — through the computation graph to compute $\frac{\partial \mathcal{L}}{\partial w}$ and $\frac{\partial \mathcal{L}}{\partial b}$.



You can get a feel for why this is computationally efficient — even though both $\frac{\partial \mathcal{L}}{\partial w}$ and $\frac{\partial \mathcal{L}}{\partial b}$ depend on $\frac{\partial \mathcal{L}}{\partial z}$, we only needed to compute it once.

Another Calculus Fact

So far, we've only had one arrow coming out of each node.

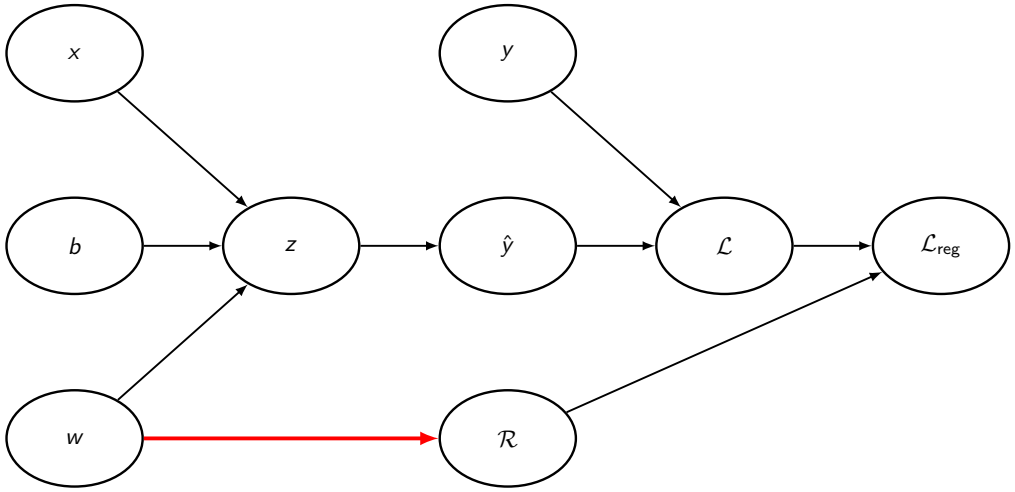
In many cases, though, we have multiple arrows coming out of some nodes. This happens all the time in DL. It also happens, for example, when we add regularization (which we'll see in a moment).

To handle this case, we need another calculus fact.

Two-variable chain rule:

$$\frac{d}{dx} f(g(x), h(x)) = \frac{df}{dg}\frac{dg}{dx} + \frac{df}{dh}\frac{dh}{dx}.$$

Regularized Logistic Regression



Computing the loss:

$$z = wx + b$$
$$\hat{y} = \sigma(z)$$
$$\mathcal{L} = -[y \log(\hat{y}) + (1-y)\log(1-\hat{y})]$$
$$\mathcal{R} = \frac{1}{2}w^2$$
$$\mathcal{L}_{\text{reg}} = \mathcal{L} + \lambda\mathcal{R}$$

Computing the derivatives:

$$\frac{\partial \mathcal{L}_{\text{reg}}}{\partial \mathcal{L}} = 1$$
$$\frac{\partial \mathcal{L}_{\text{reg}}}{\partial \mathcal{R}} = \lambda$$
$$\frac{\partial \mathcal{L}_{\text{reg}}}{\partial \hat{y}} = \frac{\partial \mathcal{L}_{\text{reg}}}{\partial \mathcal{L}}\left[-\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}\right]$$
$$\frac{\partial \mathcal{L}_{\text{reg}}}{\partial z} = \frac{\partial \mathcal{L}_{\text{reg}}}{\partial \hat{y}}\sigma(z)(1-\sigma(z))$$
$$\frac{\partial \mathcal{L}_{\text{reg}}}{\partial w} = \frac{\partial \mathcal{L}_{\text{reg}}}{\partial z}x + \frac{\partial \mathcal{L}_{\text{reg}}}{\partial \mathcal{R}}w$$
$$\frac{\partial \mathcal{L}_{\text{reg}}}{\partial b} = \frac{\partial \mathcal{L}_{\text{reg}}}{\partial z}$$

Multivariate Setting

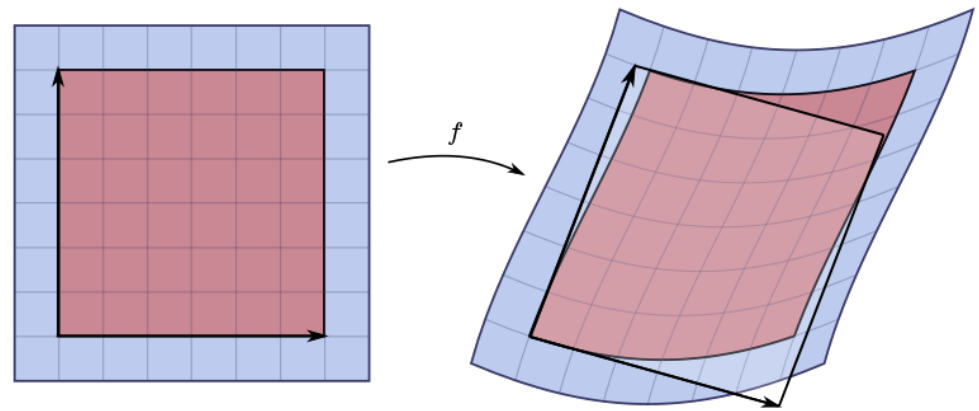
The concepts are quite similar in the multivariate setting.

The big difference is that we now use multivariable derivatives called *Jacobians*.

Similar to the 1D derivative, it's the best linear approximation to a multivariable function at a point.

For a multivariable function $f : \mathbb{R}^n \to \mathbb{R}^m$, the Jacobian is a matrix:

$$\frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}.$$



A Calculus Fact for the Multivariate Setting

The chain rule extends to multivariable functions!

Multivariable chain rule. For multivariable functions $f : \mathbb{R}^n \to \mathbb{R}^m$ and $g : \mathbb{R}^p \to \mathbb{R}^n$,

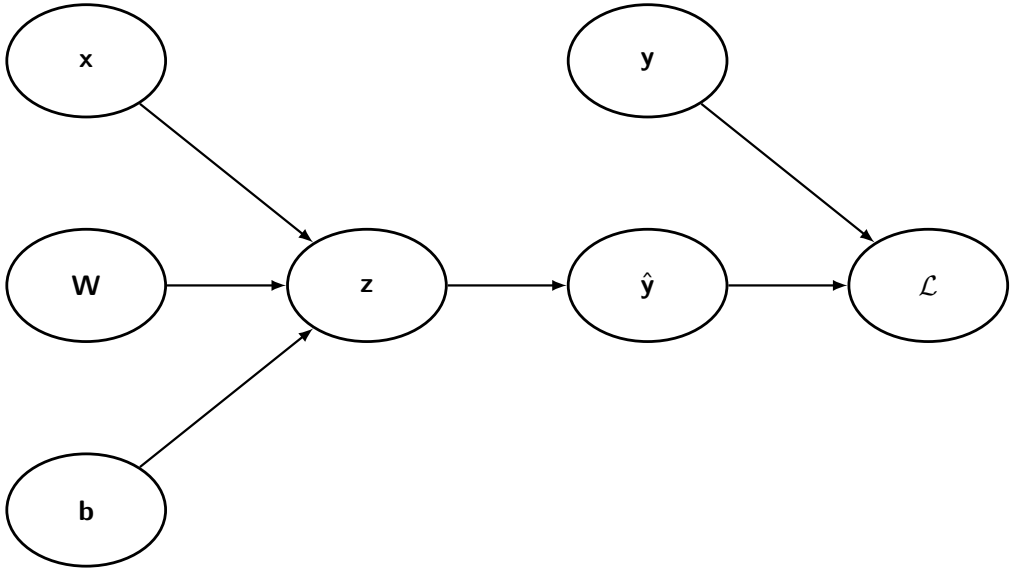
$$\frac{\partial f}{\partial \mathbf{x}} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial \mathbf{x}}$$

$$= \begin{bmatrix} \frac{\partial f_1}{\partial g_1} & \cdots & \frac{\partial f_1}{\partial g_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial g_1} & \cdots & \frac{\partial f_m}{\partial g_n} \end{bmatrix} \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \cdots & \frac{\partial g_1}{\partial x_p} \\ \vdots & \ddots & \vdots \\ \frac{\partial g_m}{\partial x_1} & \cdots & \frac{\partial g_m}{\partial x_p} \end{bmatrix}.$$

This is super similar to the single variable chain rule:

$$\frac{df(g(x))}{dx} = \frac{df}{dg}\frac{dg}{dx}.$$

Multivariate Logistic Regression



Computing the loss:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$
$$\hat{\mathbf{y}} = \sigma(\mathbf{z})$$
$$\mathcal{L} = -[\mathbf{y}^\top \log(\hat{\mathbf{y}}) + (\mathbf{1} - \mathbf{y})^\top \log(\mathbf{1} - \hat{\mathbf{y}})]$$

Computing the derivatives:[ab]

$$\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} = -\mathbf{y}^\top \operatorname{diag}(\hat{\mathbf{y}})^{-1} - (\mathbf{1} - \mathbf{y})^\top \operatorname{diag}(\mathbf{1} - \hat{\mathbf{y}})^{-1}$$
$$\frac{\partial \mathcal{L}}{\partial \mathbf{z}} = \frac{\partial \mathcal{L}}{\partial \hat{\mathbf{y}}} \operatorname{diag}(\sigma(\mathbf{z}) \odot (\mathbf{1} - \sigma(\mathbf{z})))$$
$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \left(\frac{\partial \mathcal{L}}{\partial \mathbf{z}}\right)^\top \mathbf{x}^\top$$
$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \frac{\partial \mathcal{L}}{\partial \mathbf{z}} \mathbf{I}$$

[a] $\odot$ is elementwise multiplication.

[b] The shape of $\frac{\partial \mathcal{L}}{\partial \mathbf{W}}$ is the same as $\mathbf{W}$ — this is called the *shape convention*.

Automatic Differentiation

Automatic differentiation (AD) is a way of taking a program that computes a value and automatically computing derivatives of that value.

Backpropagation is just AD applied to neural networks.

AD software works by breaking the original program down into individual operations called *primitives* that each have routines for computing derivatives.

Original program:

$$\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$$
$$\hat{\mathbf{y}} = \sigma(\mathbf{z})$$
$$\mathcal{L} = -[\mathbf{y}^\top \log(\hat{\mathbf{y}}) + (\mathbf{1} - \mathbf{y})^\top \log(\mathbf{1} - \hat{\mathbf{y}})]$$

Primitive Operations:

$$\mathbf{v}_1 = \mathbf{W}\mathbf{x}$$
$$\mathbf{z} = \mathbf{v}_1 + \mathbf{b}$$
$$\mathbf{v}_2 = -\mathbf{z}$$
$$\mathbf{v}_3 = \exp(\mathbf{v}_2)$$
$$\mathbf{v}_4 = 1 + \mathbf{v}_3$$
$$\hat{\mathbf{y}} = 1/\mathbf{v}_4$$
$$\mathbf{v}_5 = \log(\hat{\mathbf{y}})$$
$$\mathbf{v}_6 = \mathbf{1} - \hat{\mathbf{y}}$$
$$\mathbf{v}_7 = \log(\mathbf{v}_6)$$
$$\mathbf{v}_8 = \mathbf{1} - \mathbf{y}$$
$$\mathbf{v}_9 = \mathbf{v}_8^\top \mathbf{v}_7$$
$$\mathbf{v}_{10} = \mathbf{y}^\top \mathbf{v}_5$$
$$\mathbf{v}_{11} = \mathbf{v}_9 + \mathbf{v}_{10}$$
$$\mathcal{L} = -\mathbf{v}_{11}$$

More on this in the programming segment!

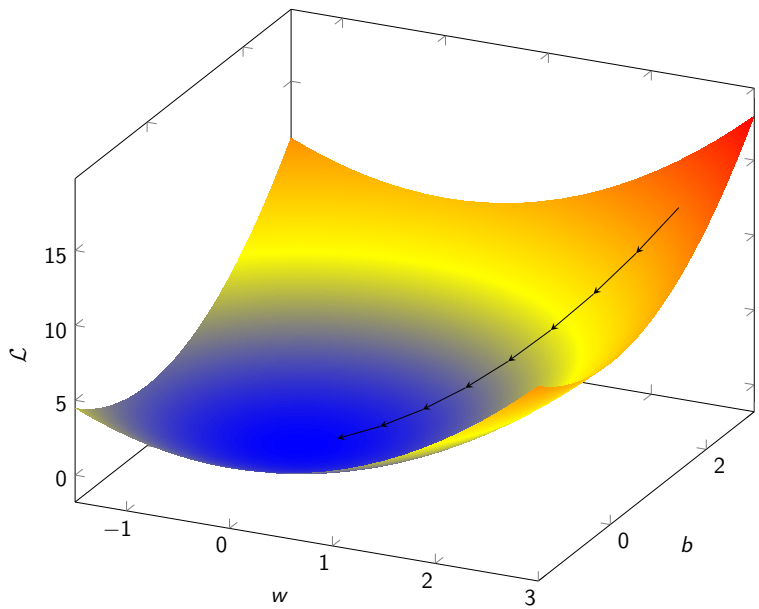
Using Our Derivatives

Ok, so now we have derivatives. What do we do with them?

They point in the steepest uphill direction on the loss surface. So, we follow them backward to go downhill — called *gradient descent*:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \epsilon \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_t},$$

where $\varepsilon > 0$ is a step size called the *learning rate*.



The Issue of Big Data

The loss function applies to a single observation. Really, we want to minimize the *cost function*, which is the mean loss for the observed data:

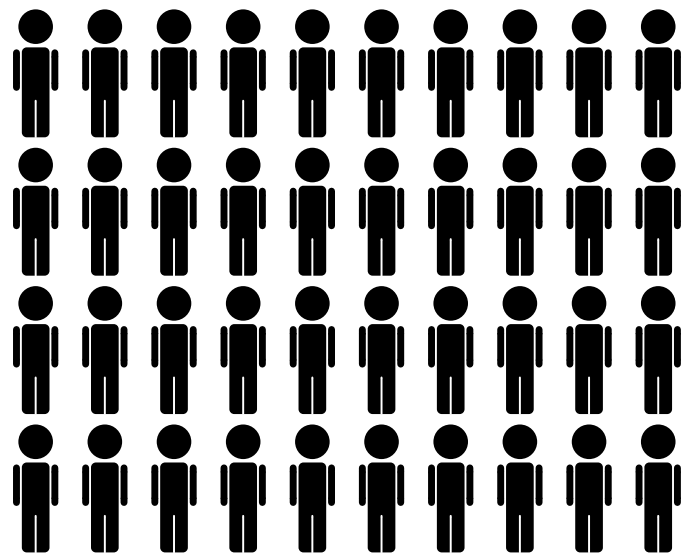
$$\mathcal{C} = \mathbb{E}_{p_{\text{data}}}[\mathcal{L}] \approx \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_i.$$

So, we really need the derivative of the *cost* with respect to the parameters:

$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{\theta}} \approx \frac{1}{N} \sum_{i=1}^{N} \frac{\partial \mathcal{L}_i}{\partial \boldsymbol{\theta}}.$$

However, data sets in many modern applications can be very large, with many thousands — or even millions — of people.

This makes the cost derivative very slow to compute.



Solution: Stochastic Gradient Descent

Randomly sample a few folks at each iteration to make a *mini-batch*, then use the mini-batch to compute the derivative!

This is called *stochastic gradient descent* (SGD):

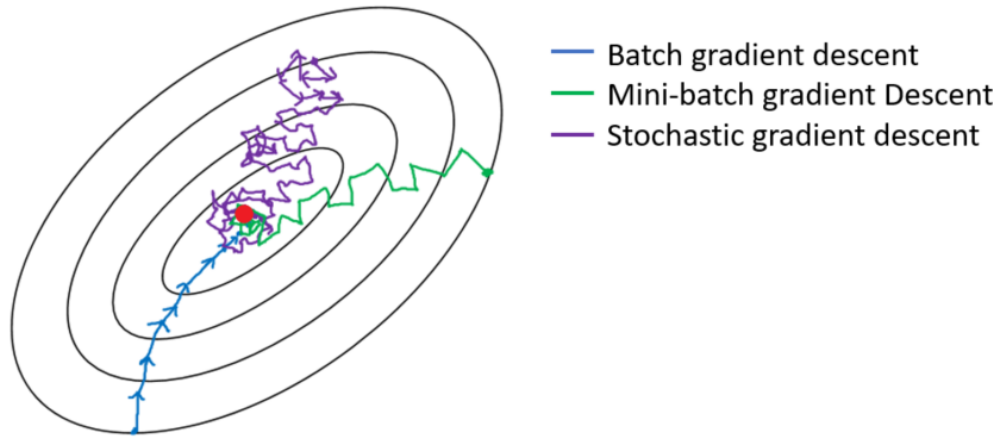
$$\frac{\partial \mathcal{C}}{\partial \boldsymbol{\theta}} \approx \frac{1}{B} \sum_{i=1}^{B} \frac{\partial \mathcal{L}_i}{\partial \boldsymbol{\theta}} = \mathbf{g}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \epsilon_t \mathbf{g},$$

where $B < N$ is the mini-batch size.

For SGD, the learning rate ε_t has to decay over time to make sure we get convergence.

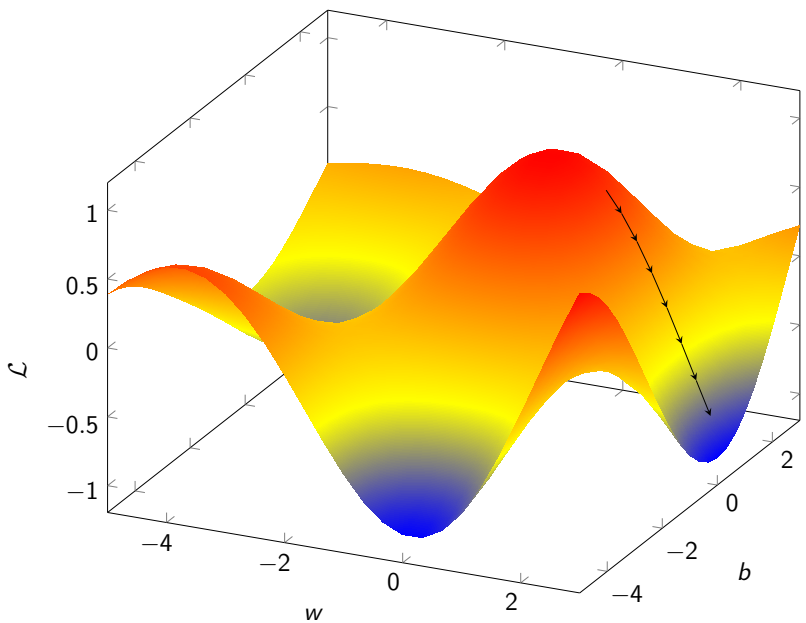
SGD can be faster than gradient descent with big data sets, but it can be unstable and tricky to tune.

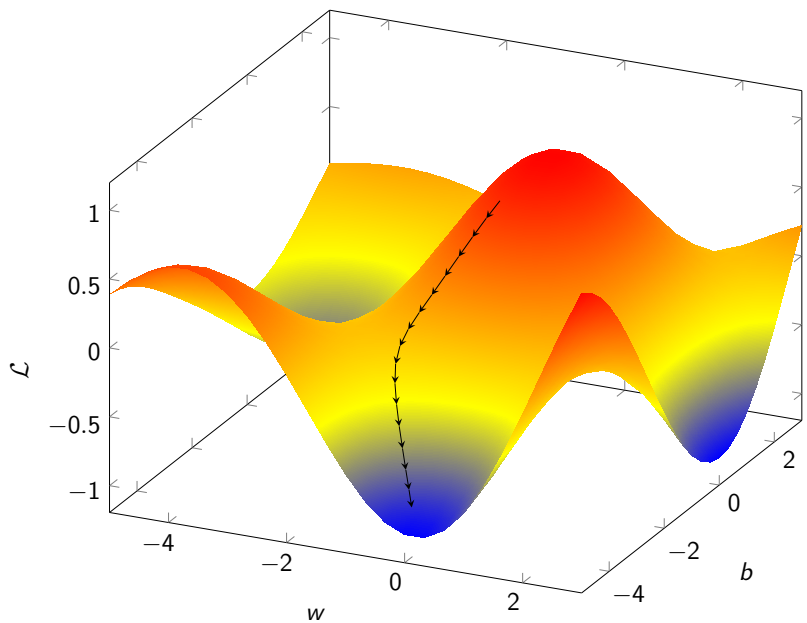


The Issue of Non-Convexity

Minimizing the loss is tricky with neural nets. Their losses are *non-convex* — i.e., there are multiple minima, saddle points, plateaus, ravines, and many other potentially tricky issues.

Same issue can happen in IRT and SEM — may obtain different parameter estimates with different starting values.





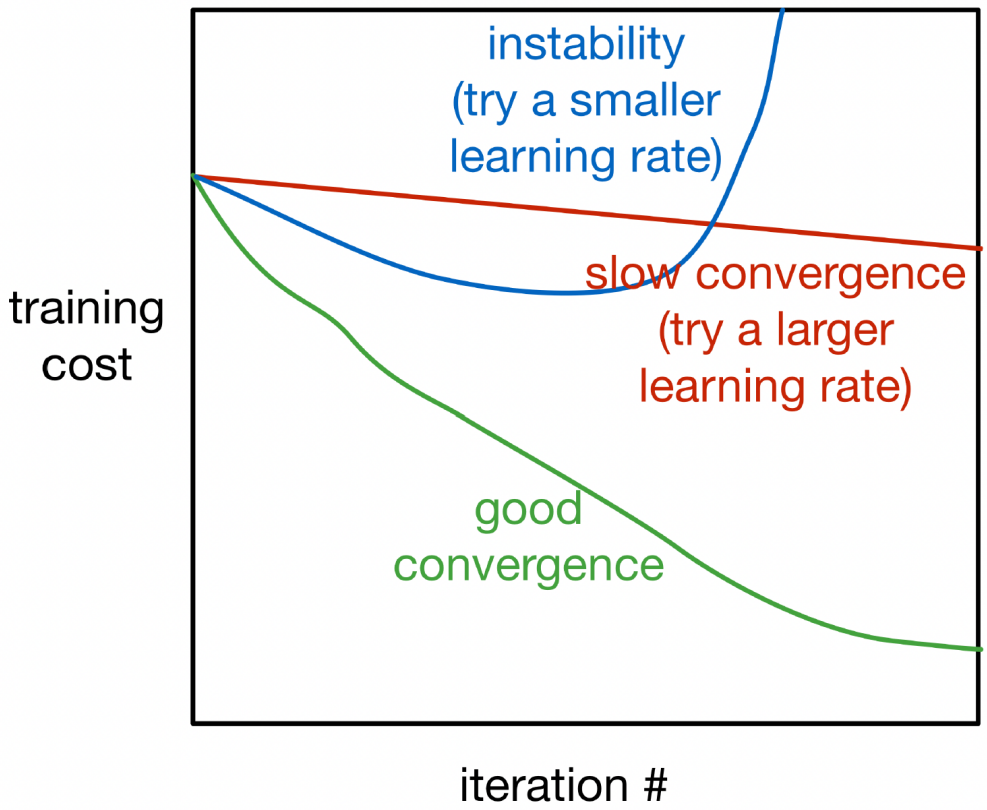
Practical Tips

- Trying multiple random starts helps find the best solution and assess stability.
- Using an *adaptive stochastic gradient* method like Adam[2] (which uses a little bit of curvature information) often requires less tuning and can successfully navigate difficult loss regions like saddle points and ravines. Unlike SGD, adaptive optimizers typically only require a single starting learning rate value ε.
  - We'll be using Adam in our coding examples later.

[2] Interestingly, while Adam is an extremely popular default optimizer in the deep learning community, it is not guaranteed to converge in general. A related optimizer that has convergence guarantees and is implemented in many deep learning frameworks is called Amsgrad.

Practical Tips, Continued

- It is *very* important to tune the learning rate. It is common to do this on a logarithmic scale, e.g., for values in $\{1 \times 10^{-2}, 1 \times 10^{-3}, \ldots, 1 \times 10^{-6}\}$.
- When manually tuning ε, plotting the cost as a function of fitting iteration helps diagnose what kind of adjustment we should make.



Learning Outcomes Checkpoint

- We've discussed how to fit deep learning models, including
  - how the *backpropagation algorithm* works;
  - how *stochastic gradient methods* work; and
  - some practical fitting tips.
- Now, we will move on to learning about basic deep learning models. We'll starting with a recap and deeper dive into multilayer perceptrons.

Recap: Multilayer Perceptrons

Earlier, we discussed the *multilayer perceptron* (MLP), a basic deep learning model that can predict outcomes in the cross-sectional setting.

It's basically a stack of generalized linear models that, when composed together, can model very complex nonlinear relationships between the predictors and the outcomes.

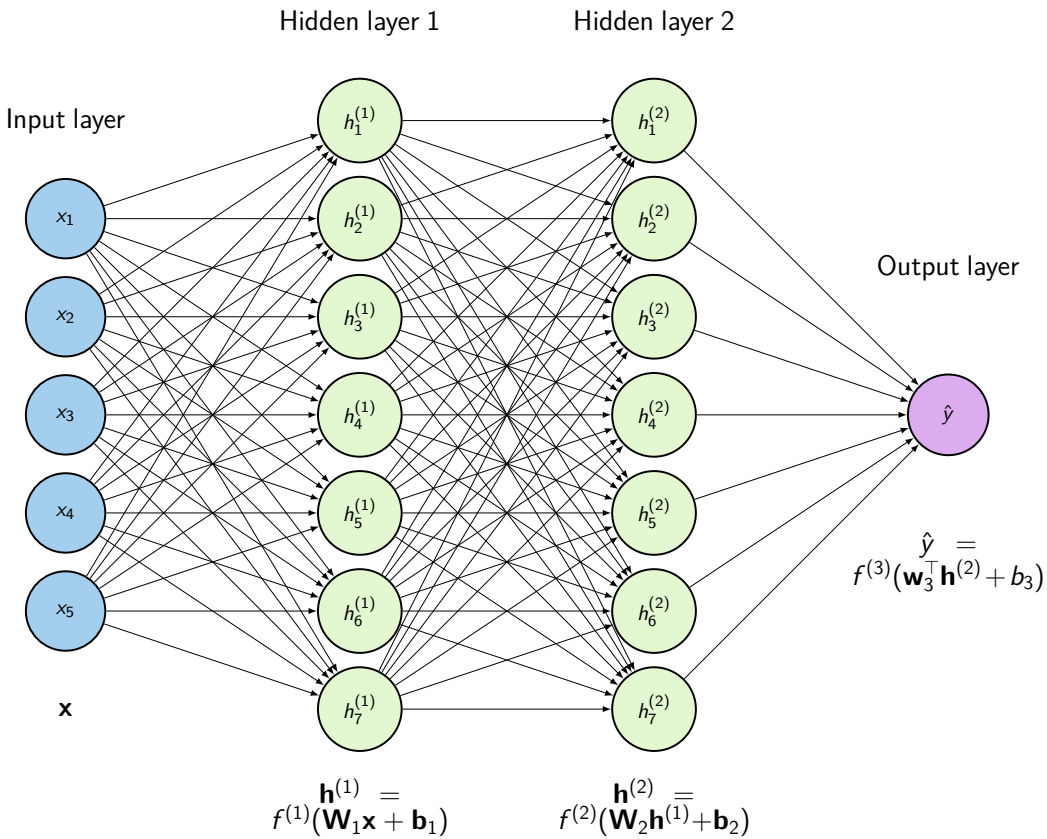
We'll discuss MLPs conceptually here, then check out some code examples in a bit.

Multilayer Perceptron Definition

Definition: Multilayer perceptron

$$\mathbf{h}^{(l)} = f^{(l)}(\mathbf{W}_l \mathbf{h}^{(l-1)} + \mathbf{b}_l), \qquad l = 1, \ldots, L,$$

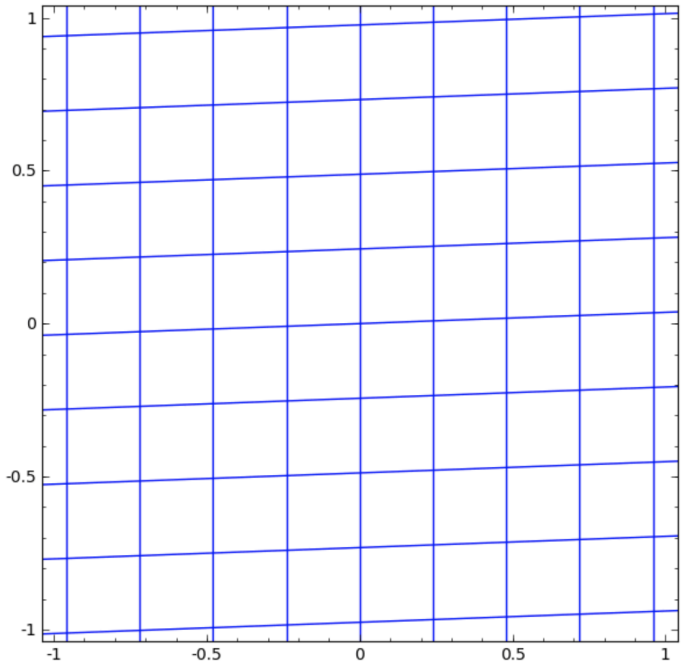
where $\mathbf{h}^{(0)} = \mathbf{x}$, $\mathbf{h}^{(L)} = \hat{y}$, $\mathbf{W}_l$ is a weight matrix, $\mathbf{b}_l$ are intercepts, and $f^{(l)}(\cdot)$ is an activation (i.e., inverse link) function.

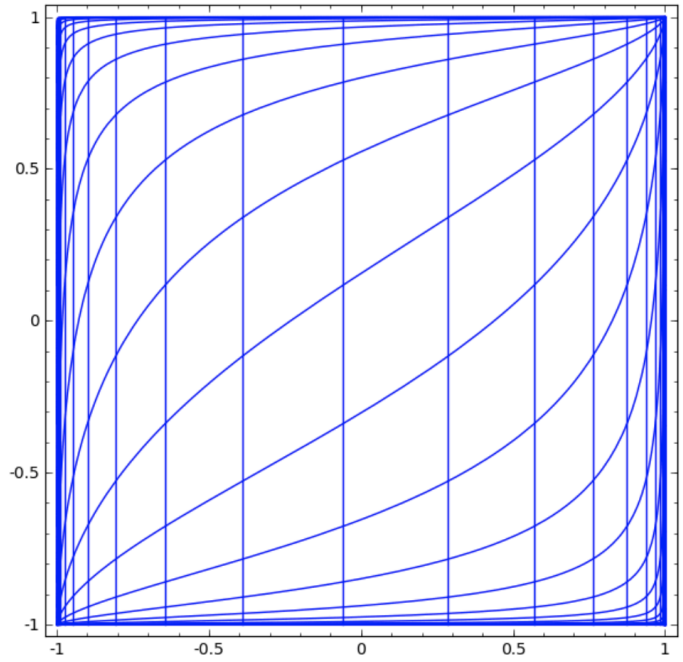


Transforming the Predictor Space

This link has nice visuals demonstrating how MLPs progressively transform the predictor space to model complex relationships:

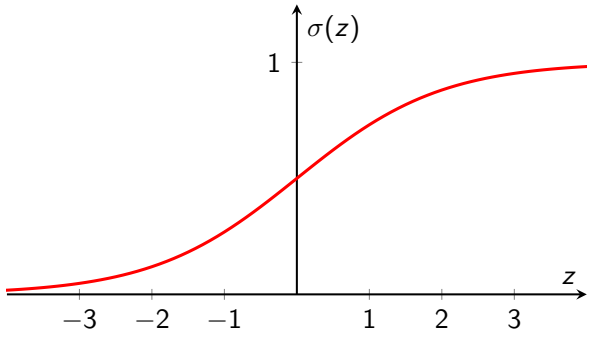
`https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/`



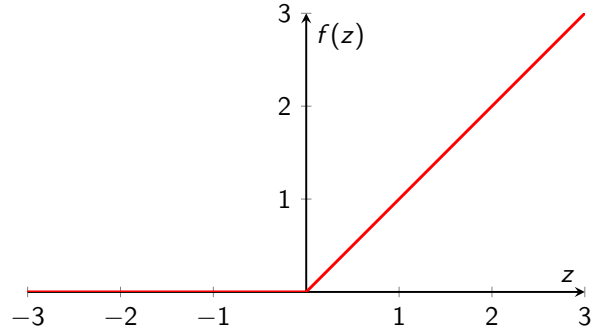


Picking the Activation Functions

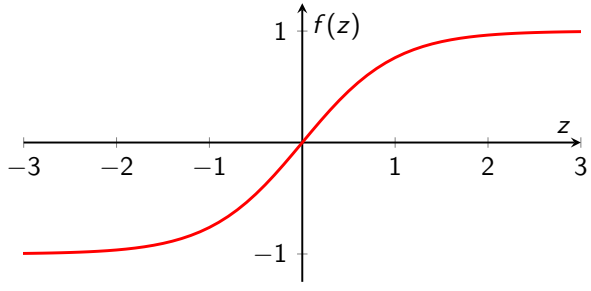
The intermediate activation functions have a ton of possibilities:



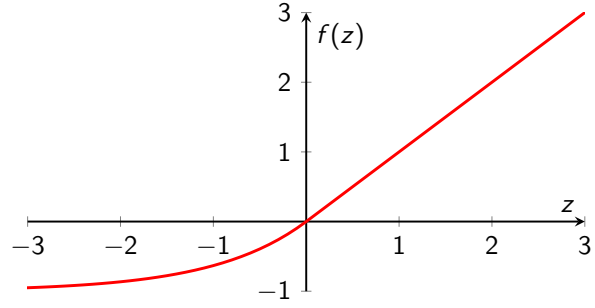
Sigmoid activation function:
$\sigma(z) = 1/(1 + \exp[-z])$



ReLU activation function:
$f(z) = \max(z, 0)$



Tanh activation function:
$f(z) = (\exp[z] - \exp[-z])/(\exp[z] + \exp[-z])$



ELU activation function:
$f(z) = \max(z, 0) + \exp[\min(z, 0)] - 1$

But, they have optimization difficulties in the flat regions, especially in models with many layers.

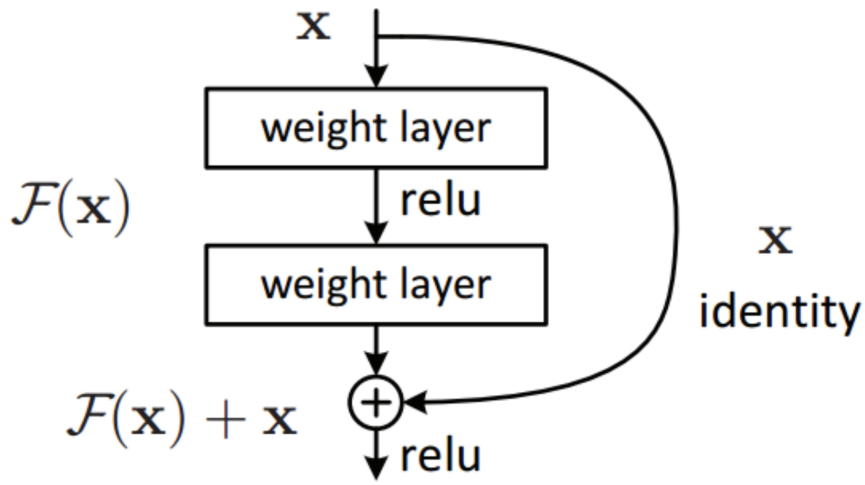
Activation Optimization Issues

When parameters are in flat regions of activation functions, often causes "gradient vanishing" in deep models.

- Because we're multiplying lots of derivatives, small derivatives get smaller going back, causing fitting signal to vanish.

Solutions:

- Initialize the weights appropriately — this done by default in many deep learning frameworks.
- In very deep models, people often add highways called *residual connections* to help information flow between layers.



The Final Activation Function

The choice for the final activation function depends on the kinds of outcomes we have.

- If the outcome is continuous, use an *identity activation* function to predict a continuous value:

$$f(z) = z.$$

- If the outcome is binary (e.g., $y \in \{\text{cried today}, \text{did not cry today}\}$), use a *sigmoid activation* function to predict the probability that $y = 1$:

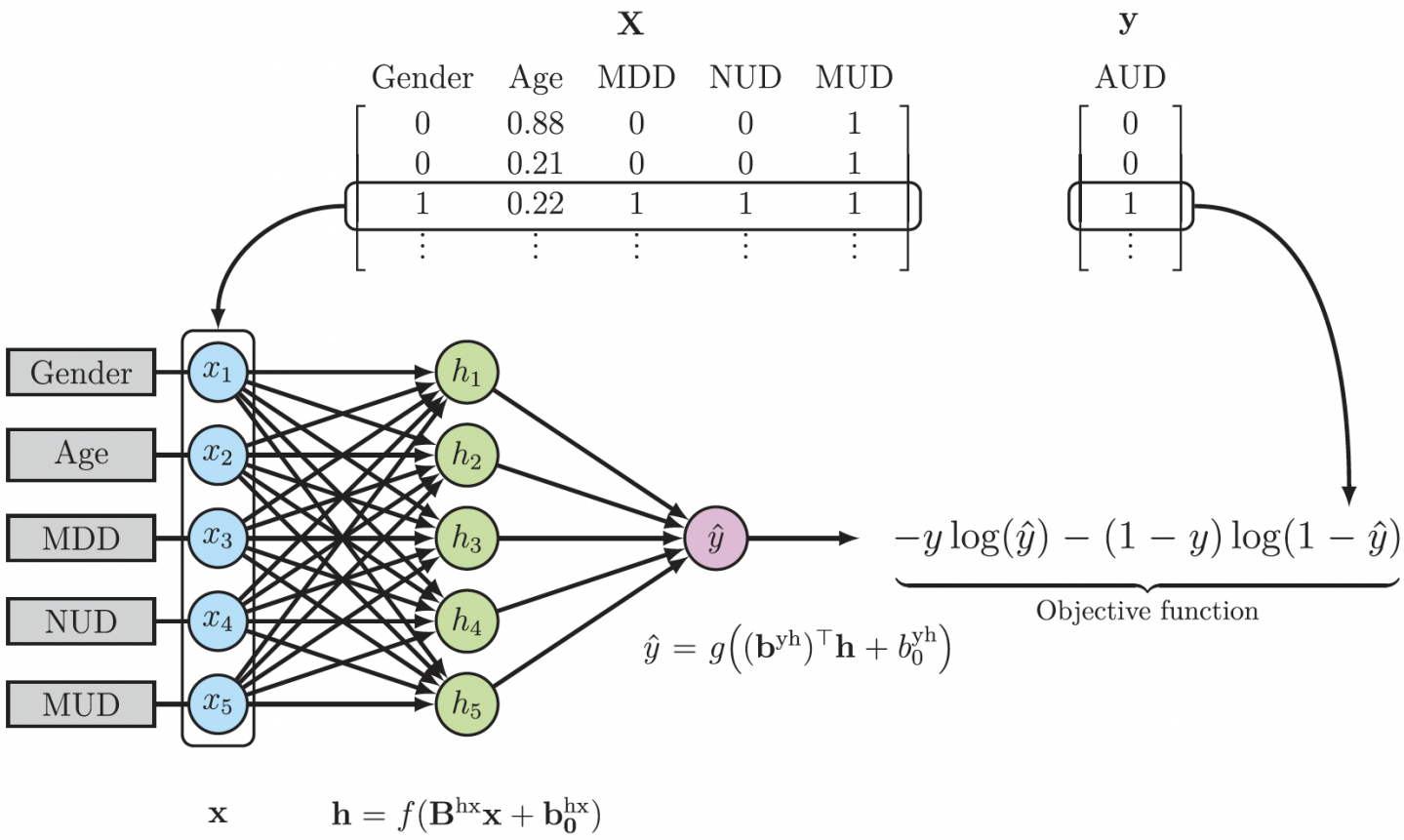
$$\sigma(z) = 1/(1 + \exp[-z]).$$

- If the outcome is nominal with K categories (e.g., $y \in \{\text{angry}, \text{sad}, \text{happy}\}$), use a *softmax activation* to predict the probability that $y = k,\ k = 1, \ldots, K$:

$$f_k(\mathbf{z}) = \exp(z_k) / \sum_{i=1}^{K} \exp[z_i].$$

MLP Example

To fix ideas, here's an MLP example from Katie Gates' and my deep learning primer for psychologists.

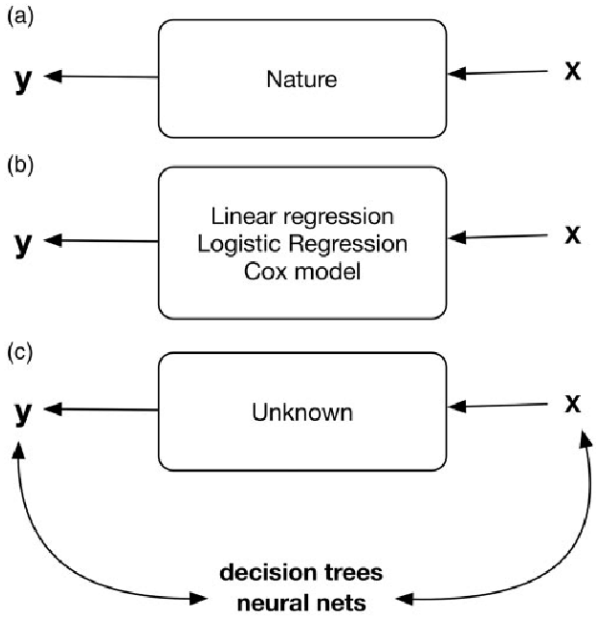


Note. Age is normalized such that its values range from zero to one. AUD = alcohol use disorder; MDD = major depressive disorder; MUD = marijuana use disorder; NUD = nicotine use disorder. See the online article for the color version of this figure.

Overparameterization and Interpretability

MLPs are *overparameterized* — they have more parameters than the number of equations defining the model.

Overparameterization → there are infinitely many sets of parameters that model the relationship between predictors and outcomes, so noparameter sets have intrinsic meaning — i.e., they're not *interpretable* (see Katie Gates' and my deep learning primer for more details).



This is an issue in scientific settings where interpretable parameters are desired, but it's less of an issue for predictive modeling.

But, see this book for methods to explain black-box models: christophm.github.io/interpretable-ml-book/

Universal Approximation Theorems

Universal approximation theorems (UATs) are mathematical results that show that MLPs (and sometimes other kinds of neural nets) can approximate any nonlinear association between predictors and outcomes:

- en.wikipedia.org/wiki/Universal_approximation_theorem

They are often cited as theoretical justification for modeling unknown relationships with neural nets.

Unfortunately, UATs often make unrealistic assumptions and don't guarantee that the true relationship will be captured in practice.

Indeed, in practice with continuous cross-sectional data, MLPs perform similarly to alternative models (e.g., support vector machines, random forests).

In my opinion, main benefit of MLPs is that they can be used as *flexible modeling components* in a variety of settings.

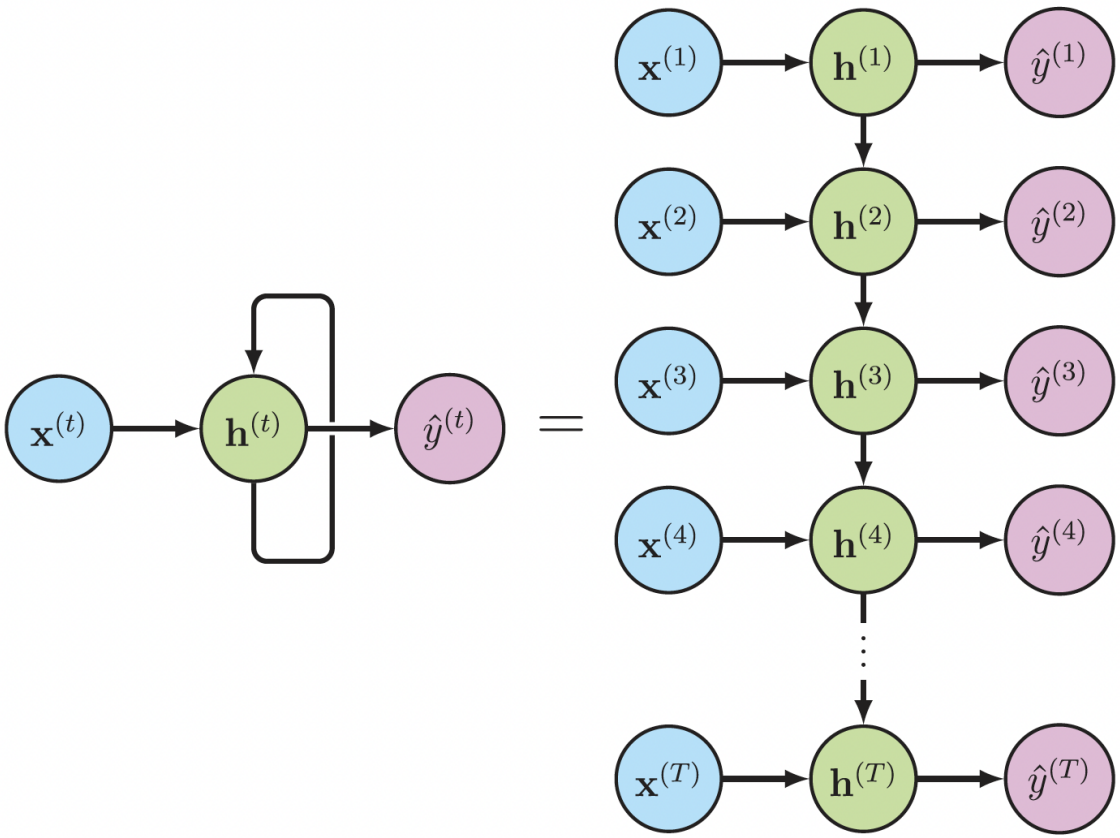
Learning Outcomes Checkpoint

- We've just discussed the multilayer perceptron in some detail.
- Now we'll move on to recurrent neural networks, which are used to model longitudinal data.

Recurrent Neural Networks

Recurrent neural networks (RNNs) are very useful for predicting outcomes using longitudinal data.

Intuitively, they're just MLPs with loops carrying information forward through time — and we *reuse the same MLP* at each time point.



RNNs: The Equations

Definition: Recurrent neural network

$$\mathbf{h}^{(t)} = f(\mathbf{W}_{\text{hx}}\mathbf{x}^{(t)} + \mathbf{W}_{\text{hh}}\mathbf{h}^{(t-1)} + \mathbf{b}_{\text{hx}}), \qquad t = 1, \ldots, T,$$
$$\hat{y}^{(t)} = g(\mathbf{W}_{\text{yh}}\mathbf{h}^{(t)} + b_{\text{yh}}) \qquad t = 1, \ldots, T,$$

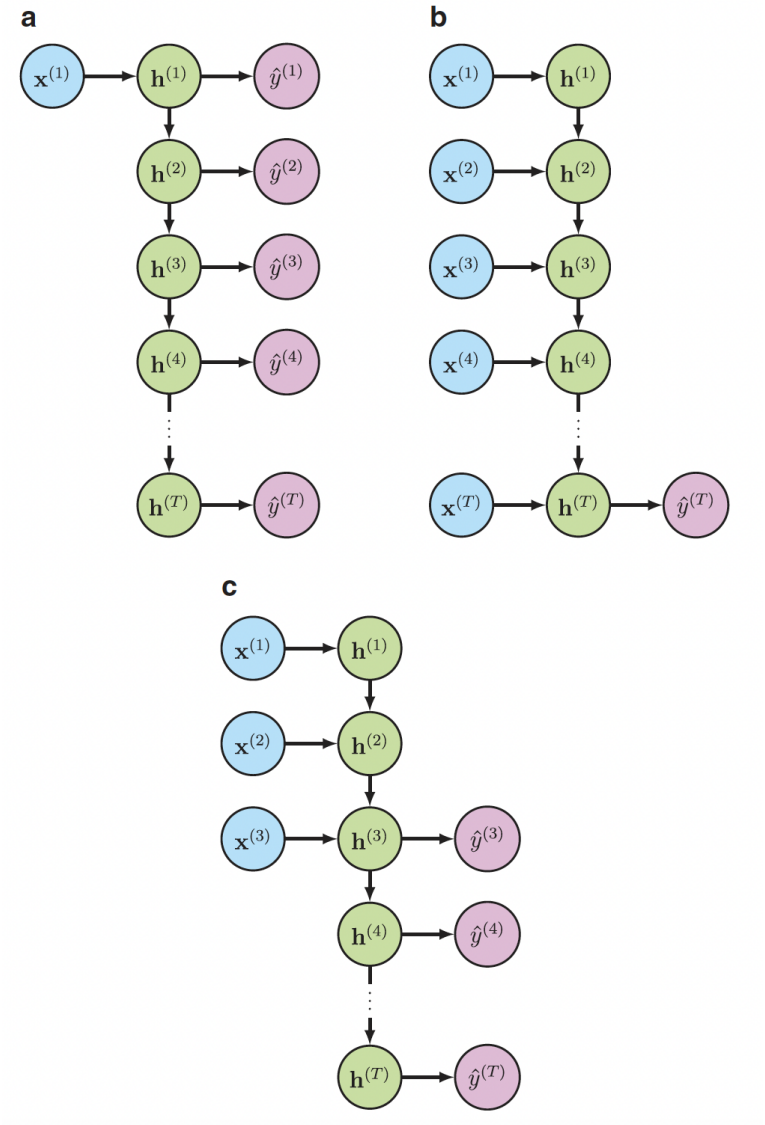
where $\mathbf{h}^{(0)} = \mathbf{0}$, $\mathbf{W}_{\text{hx}}$ are weights from the predictors to the hidden layer, $\mathbf{W}_{\text{hh}}$ are weights from the previous to the current hidden layer, $\mathbf{b}_{\text{hx}}$ are intercepts applied to the hidden layer, $\mathbf{W}_{\text{yh}}$ are weights from the hidden layer to the outcome, and b_{yh} are intercepts applied to the outcome.

This clearly shows that the current hidden layer values depend on the previous hidden layer values.

Also, we can see that the same parameters get applied at each time point. This is called *weight sharing* and makes RNNs quite computationally efficient.

Different Kinds of RNNs

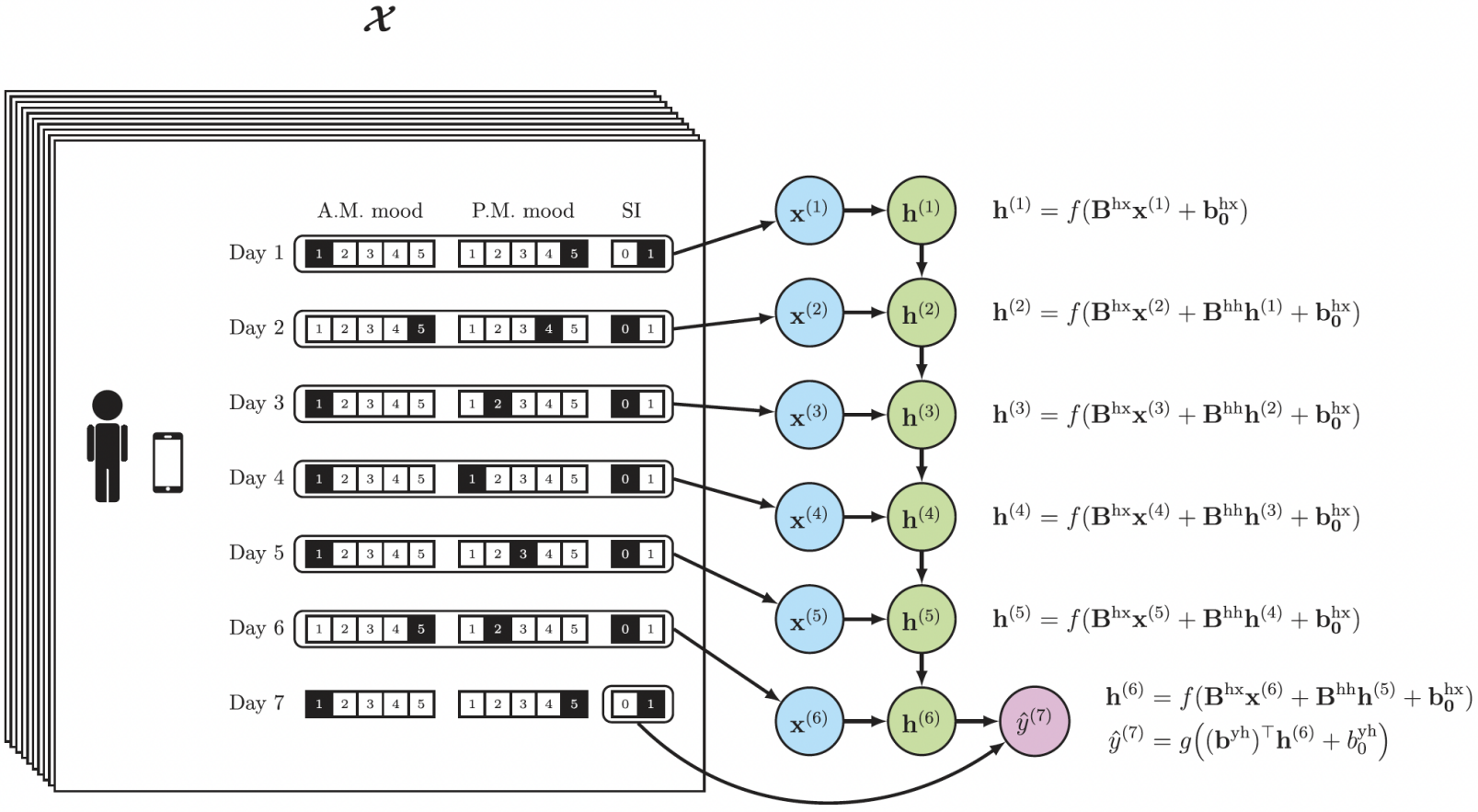
RNNs can be *one-to-many*, *many-to-one*, and *many-to-many*.



RNN Example

An RNN example from Katie Gates' and my deep learning primer for psychologists.

Schematic Visualizing How Recurrent Neural Networks May Be Applied to Daily Diary Data to Predict Suicidal Ideation



Note. Data for a single simulated participant is shown; the participant's response to a given question at a given time point is shaded in black. SI = suicidal ideation. See the online article for the color version of this figure.

Exploding and Vanishing Derivatives

The simple RNNs we've discussed often have optimization issues due to the hidden layer connections.

To get intuition about why, notice what happens in the recurrence relation

$$h^{(t)} = wh^{(t-1)}, \qquad t = 1, \ldots, T.$$

We have

$$h^{(t)} = w^t h^{(0)},$$

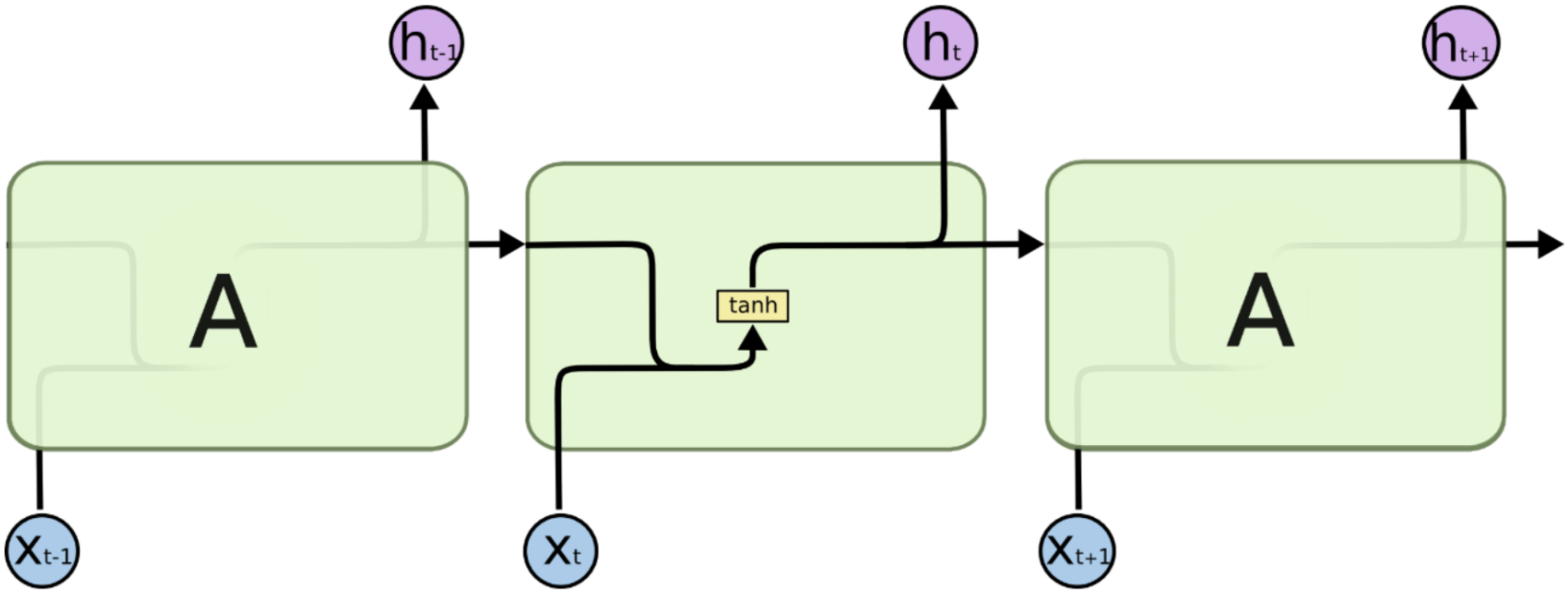
which explodes when $|w| > 1$ and vanishes when $|w| < 1$.

A similar process happens in simple RNNs, causing derivatives to also explode or vanish during backprop. See Lipton, Berkowitz, and Elkan (2015) in the supplemental materials for a detailed discussion.

Solution: Gated Architectures

The key idea of gated RNNs is that *the model learns to control the flow of information over time*.

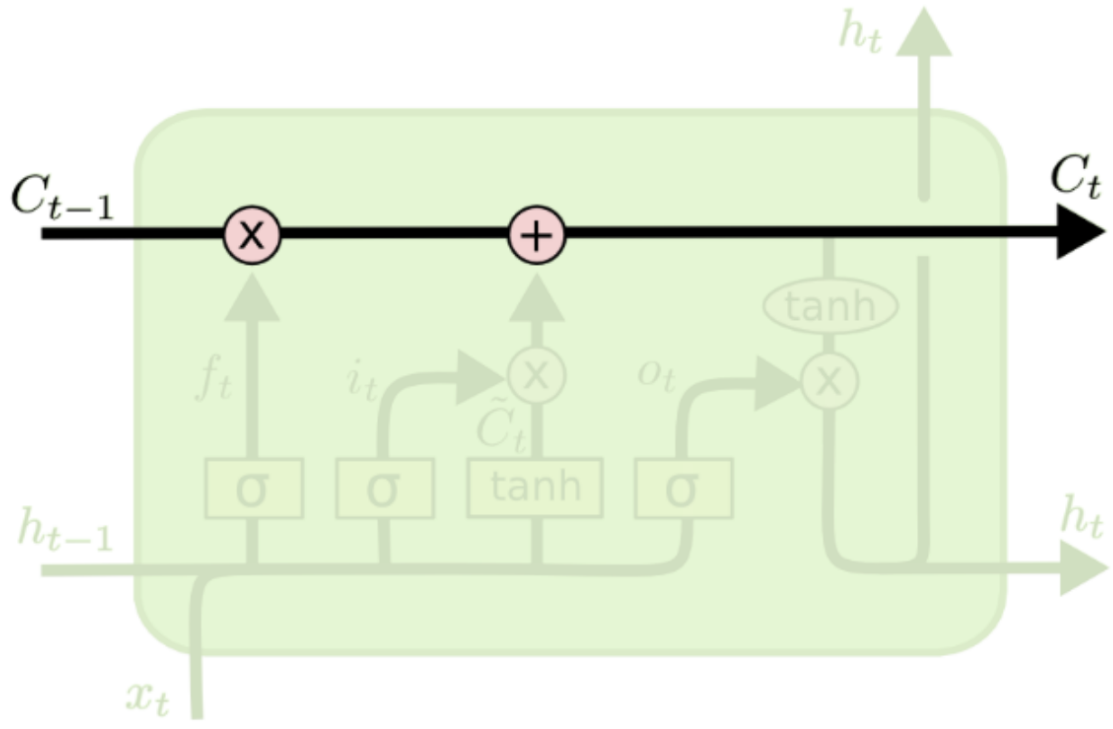
In the standard RNN, information just flows freely:[3]



[3] Graphics here and on subsequent LSTM slides taken from Olah (2015) in supplemental materials. His blog is an incredibly clear and intuitive resource!

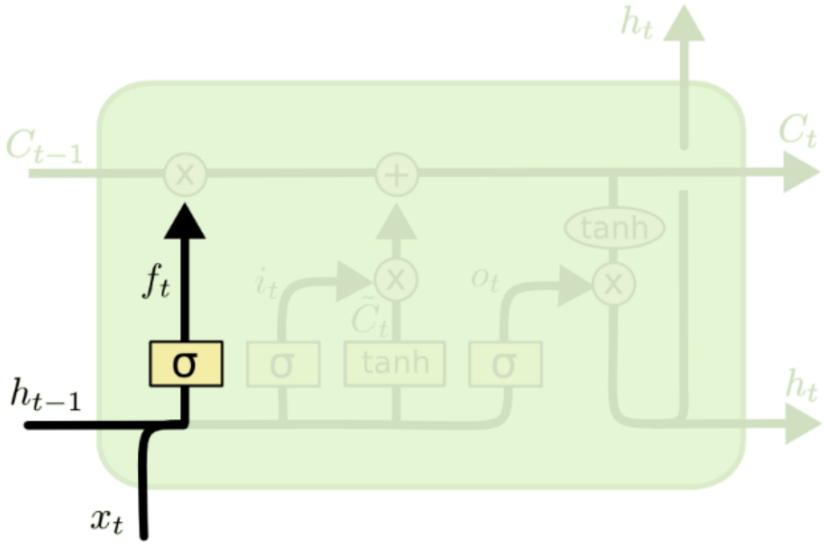
Long Short-Term Memory: The Cell State

It's basically a conveyor belt for information that the model adds to or subtracts from via gating mechanisms.



Long Short-Term Memory: The Forget Gate

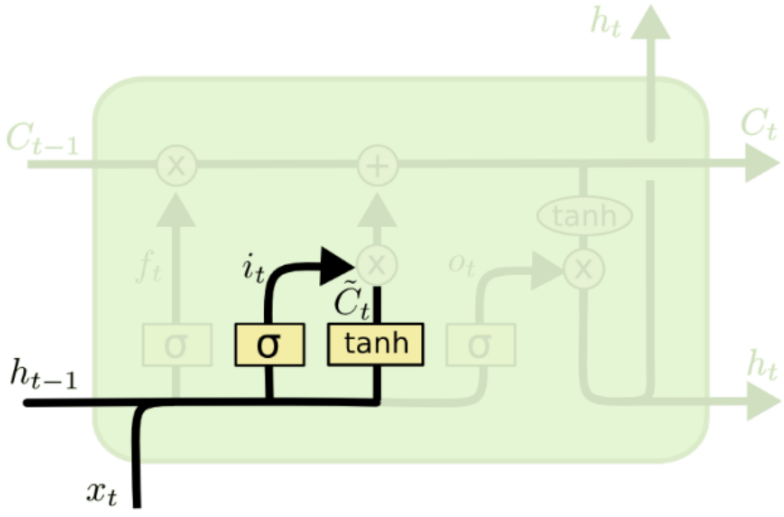
f_t is between 0 and 1, so this gate decides what proportion of the total information in the conveyor belt remains.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Long Short-Term Memory: The Input Gate

It decides whether we should add or subtract information from the conveyor belt.

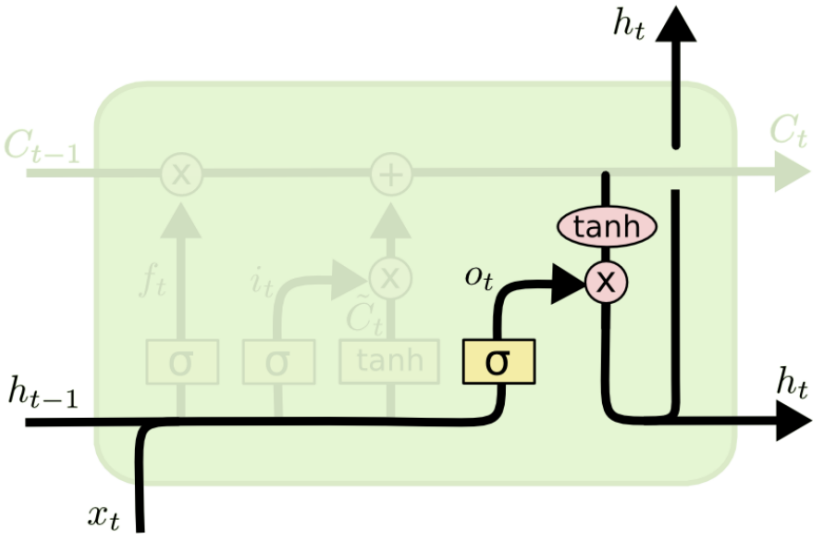


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Long Short-Term Memory: The Output Gate

It decides what information from the conveyor belt we should output to make predictions and pass on to the next time step.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

Learning Outcomes Checkpoint

- We've just discussed recurrent neural networks, which are used to model longitudinal data.
- Our next goal is to understand convolutional neural networks, which are used to model images (and sometimes sequences, too).

Convolutional Neural Networks

Convolutional neural networks (CNNs) are best known as models for identifying the content of images.

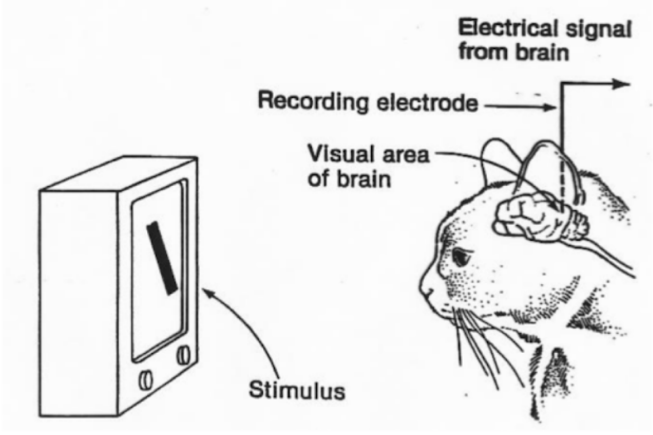
Lots of real-world uses:

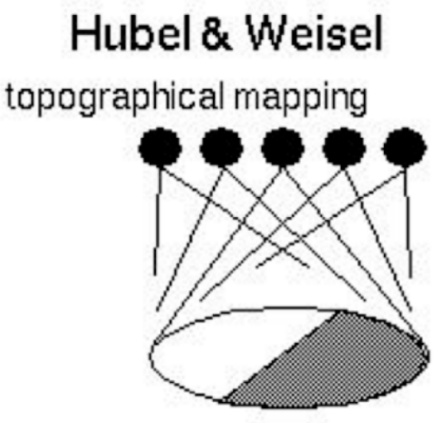
- Object detection (driving, flying, satellite imagery, etc.)
- Facial recognition (phone ID, funny app backgrounds)
- Medical image analysis
- Reading and classifying handwritten documents

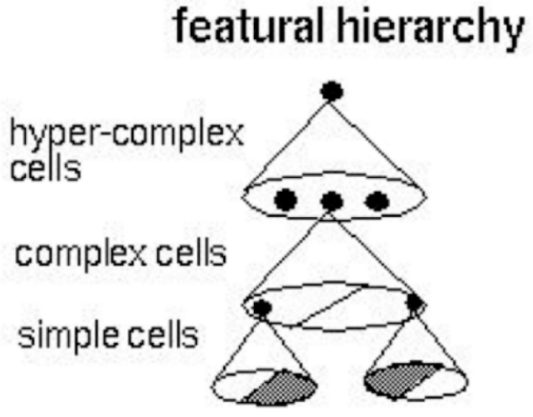
A CNN could be used to answer, "Which emotion is being shown here?"

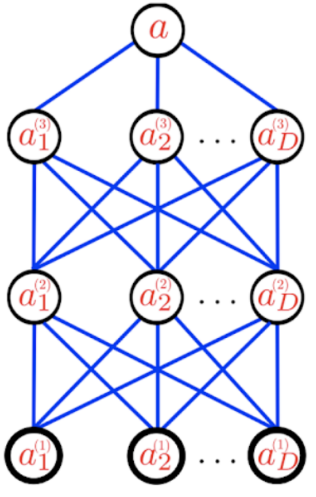
CNNs Inspired by the Visual Cortex

D. Hubel and T. Wiesel, 1959









http://cbl.eng.cam.ac.uk/pub/Intranet/MLG/ReadingGroup/cnn_basics.pdf

What is a Convolution?

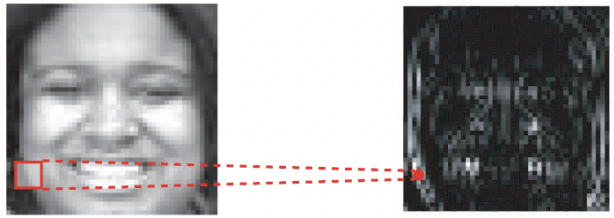
It's a little matrix that slides over an image, multiplies each value underneath, then adds all the values together to produce a single value.

Intuitively, it summarizes the information from little patches of the image.

They've been used in image processing for a long time. This one detects vertical edges.

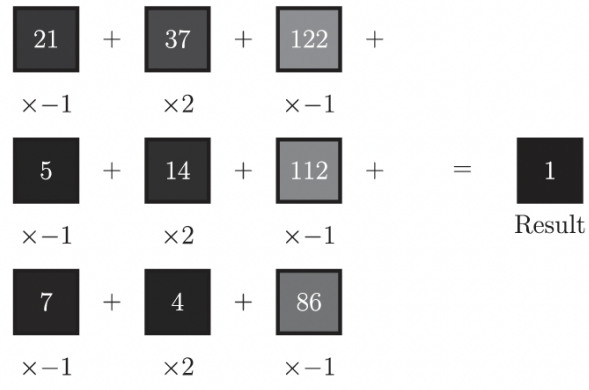
$$\mathbf{B}^{\text{hx}} = \begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}$$

Kernel for vertical edge detection

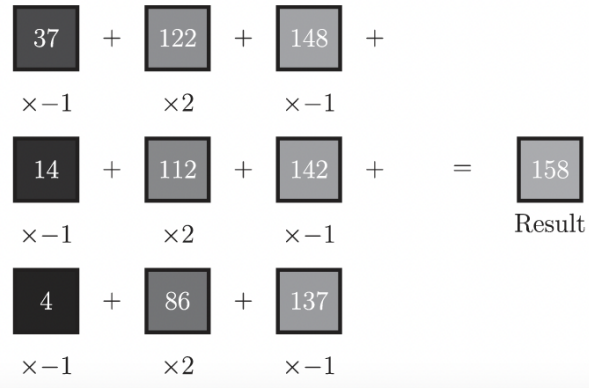


Original image

Convolution



A patch of the original image



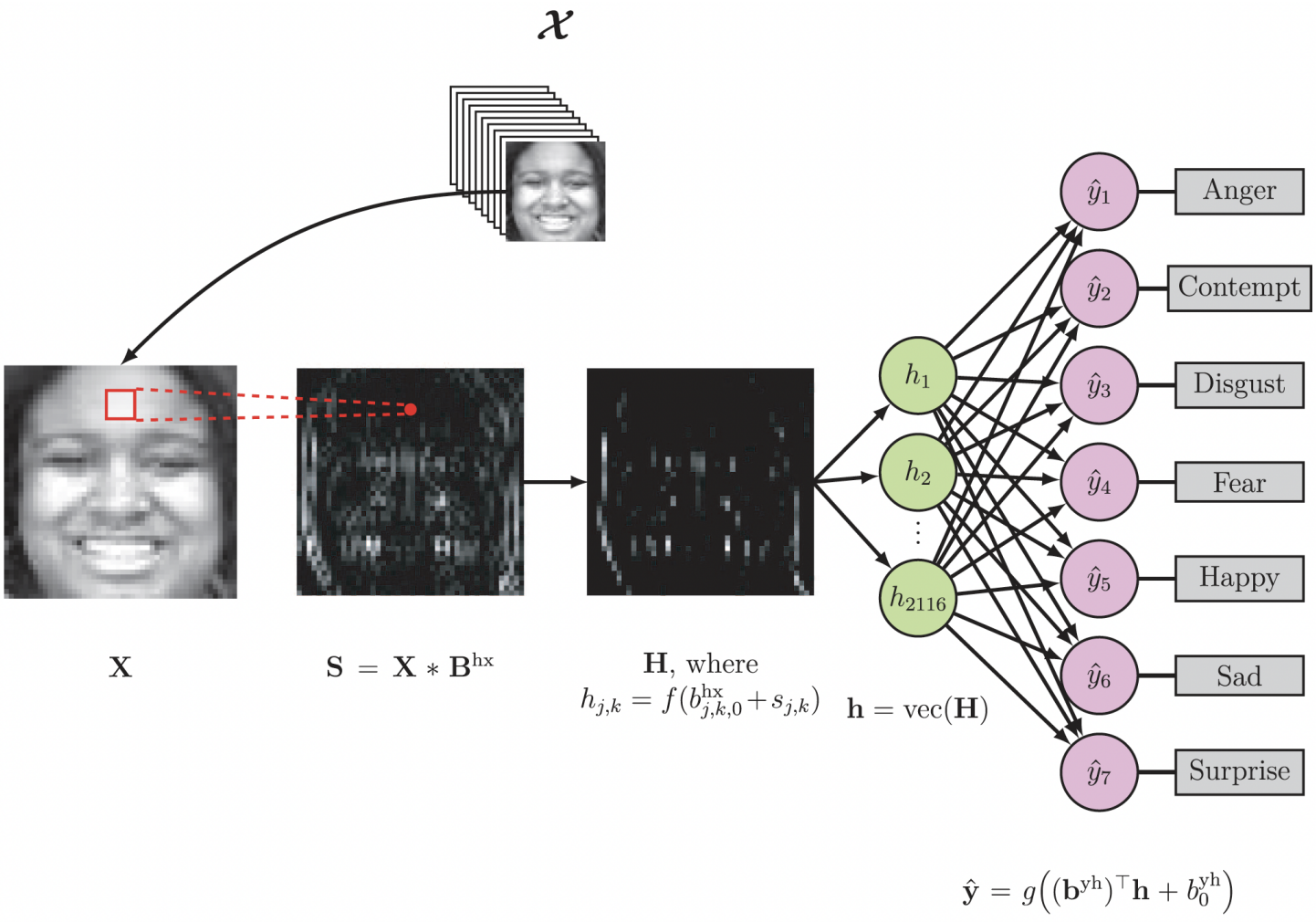
Convolution in Real Time

Lots of nice convolution GIFs out there — they really help with visualizing the process.

For example, we'll check out:
commons.wikimedia.org/wiki/File:2D_Convolution_Animation.gif

CNN Example

The CNN example from Katie Gates' and my deep learning primer for psychologists.

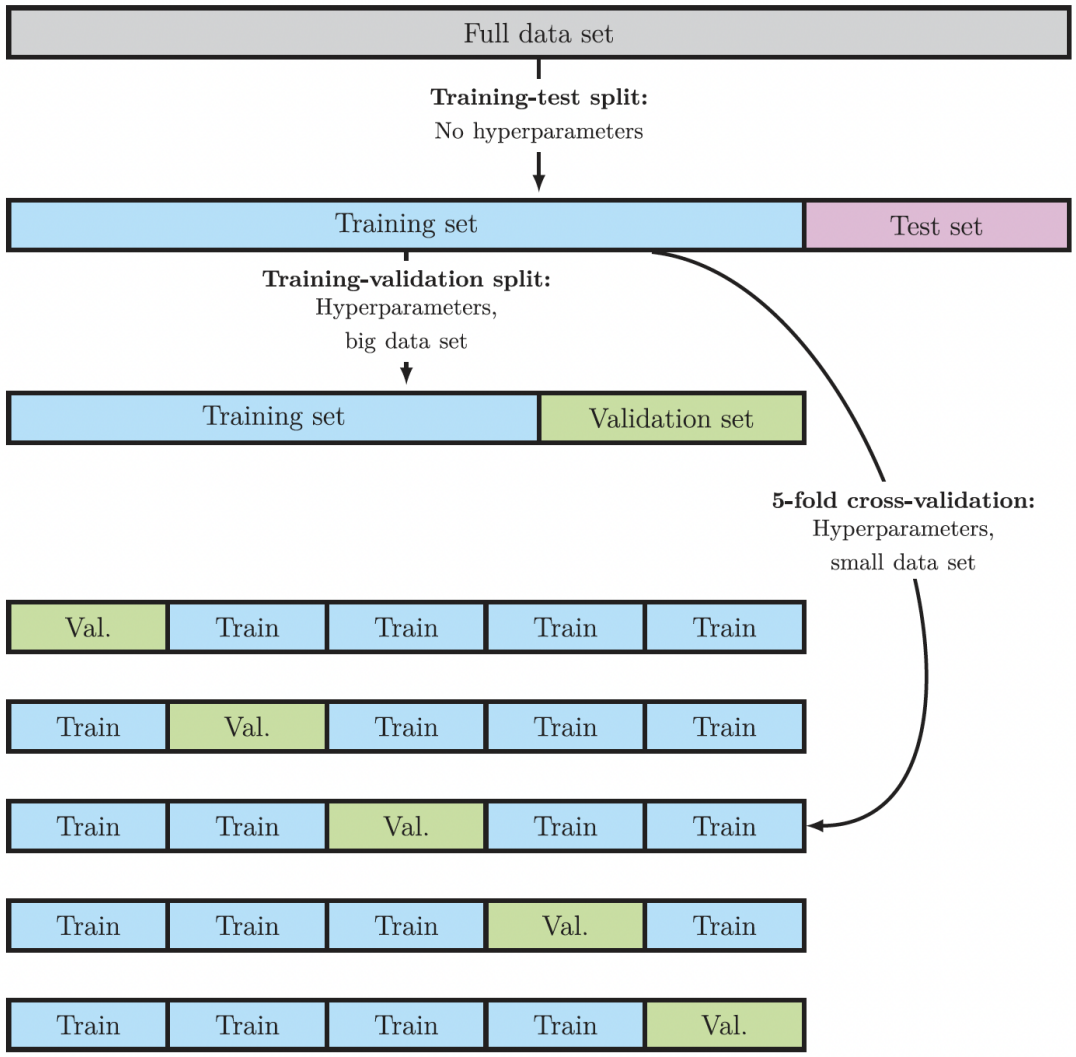


Learning Outcomes Checkpoint

- We've just discussed convolutional neural networks, which are used to model images.
- Our final learning outcome for this lecture is to briefly discuss tuning deep learning model hyperparameters and model comparison.

Comparing Fitted Models

In machine learning, key goal is ensuring models perform well on new data — i.e., models *generalize*. Usually assess using holdout sets or cross-validation.



Picking the Best

Often multiple models are fitted and compared.

The model with the best test set/cross-validation loss or performance metric is chosen.

In deep learning, it's a good idea to compare results across a few trials (e.g., 5–10) to ensure ranking is consistent across random starts.

Deep Learning Model Hyperparameters

Hyperparameters are parameters whose values are set by the user rather than estimated from the data.

Deep learning models have:

- the learning rate (most important);
- the mini-batch size;
- the number, sizes, and types of the hidden layers;
- the activation functions.

What else?

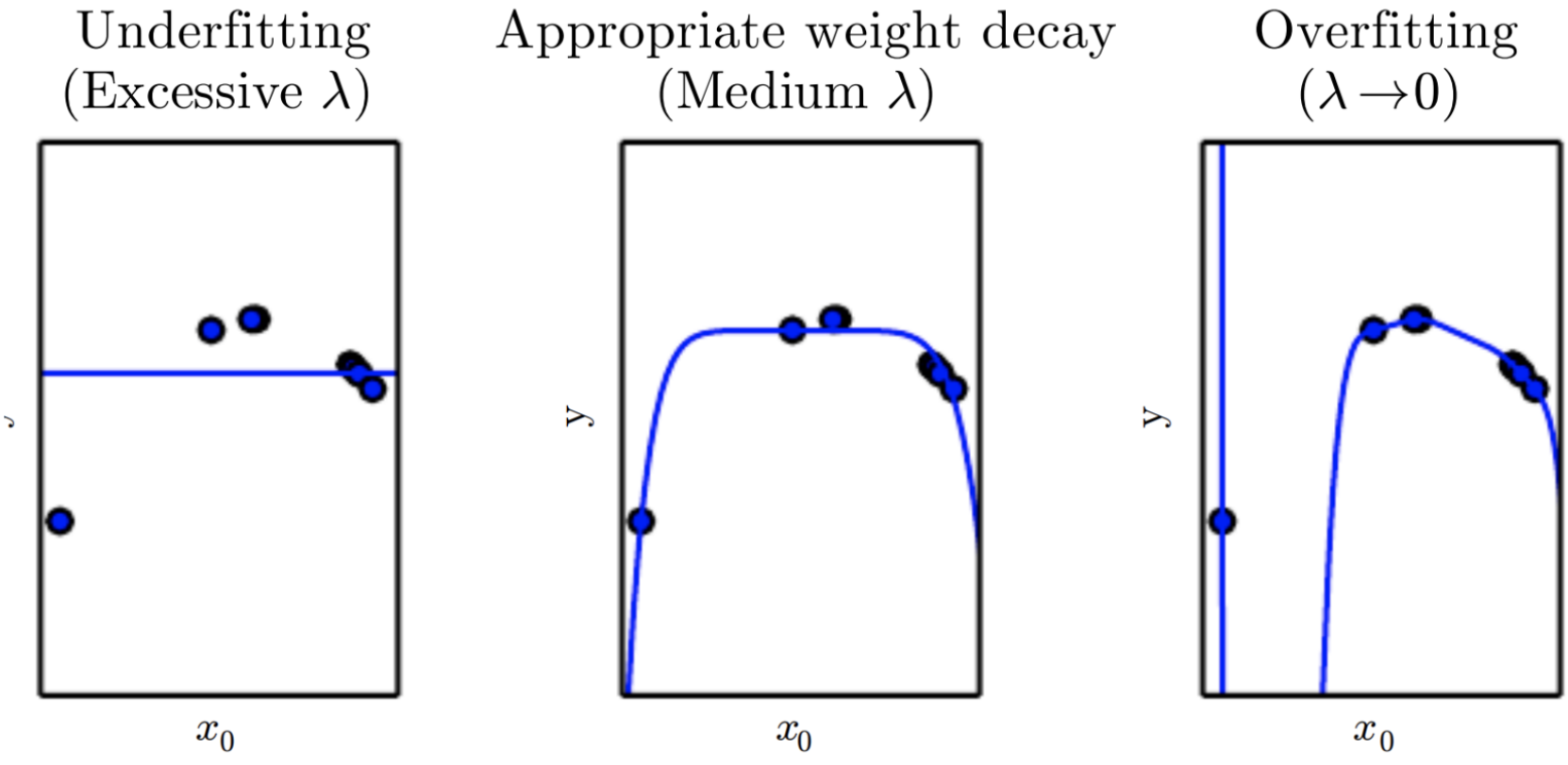
Regularization for Deep Learning

Neural networks are extremely flexible. So it's often important in practice to regularize to reduce overfitting.

There are a ton of neural net regularization methods. We'll just discuss a few here.

Weight Decay

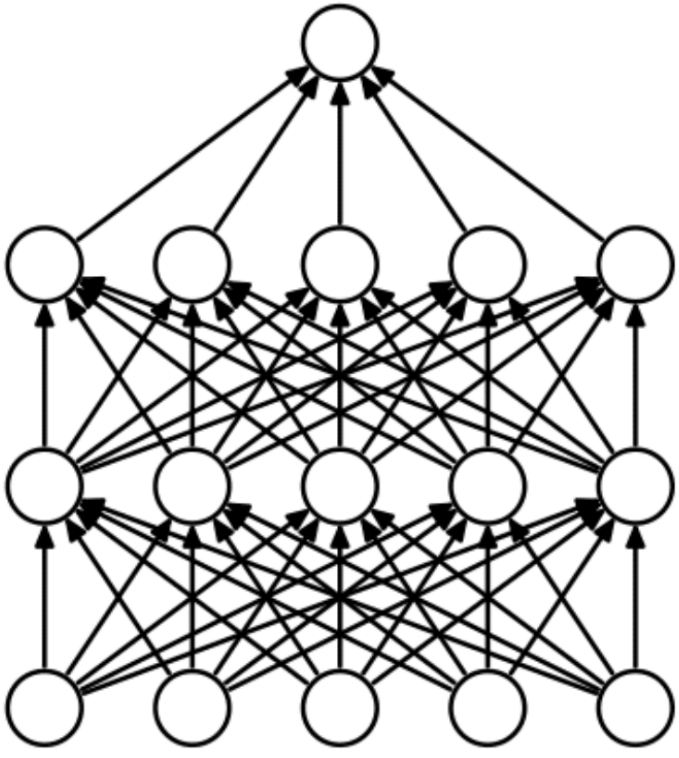
Adds a penalty term to the loss to shrink the neural net weights.



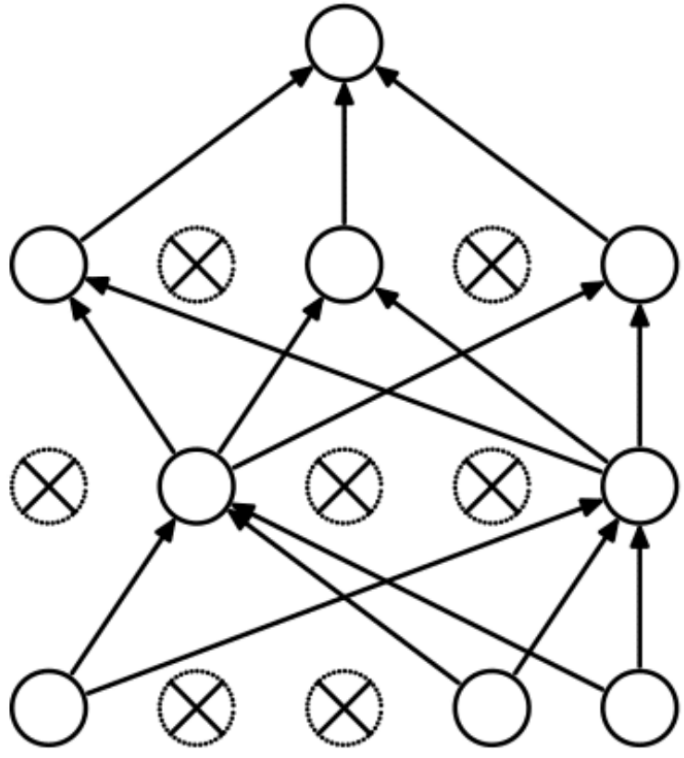
Need to tune penalty parameter λ.

Dropout

Randomly masks or "drops out" some hidden layer values.



(a) Standard Neural Net

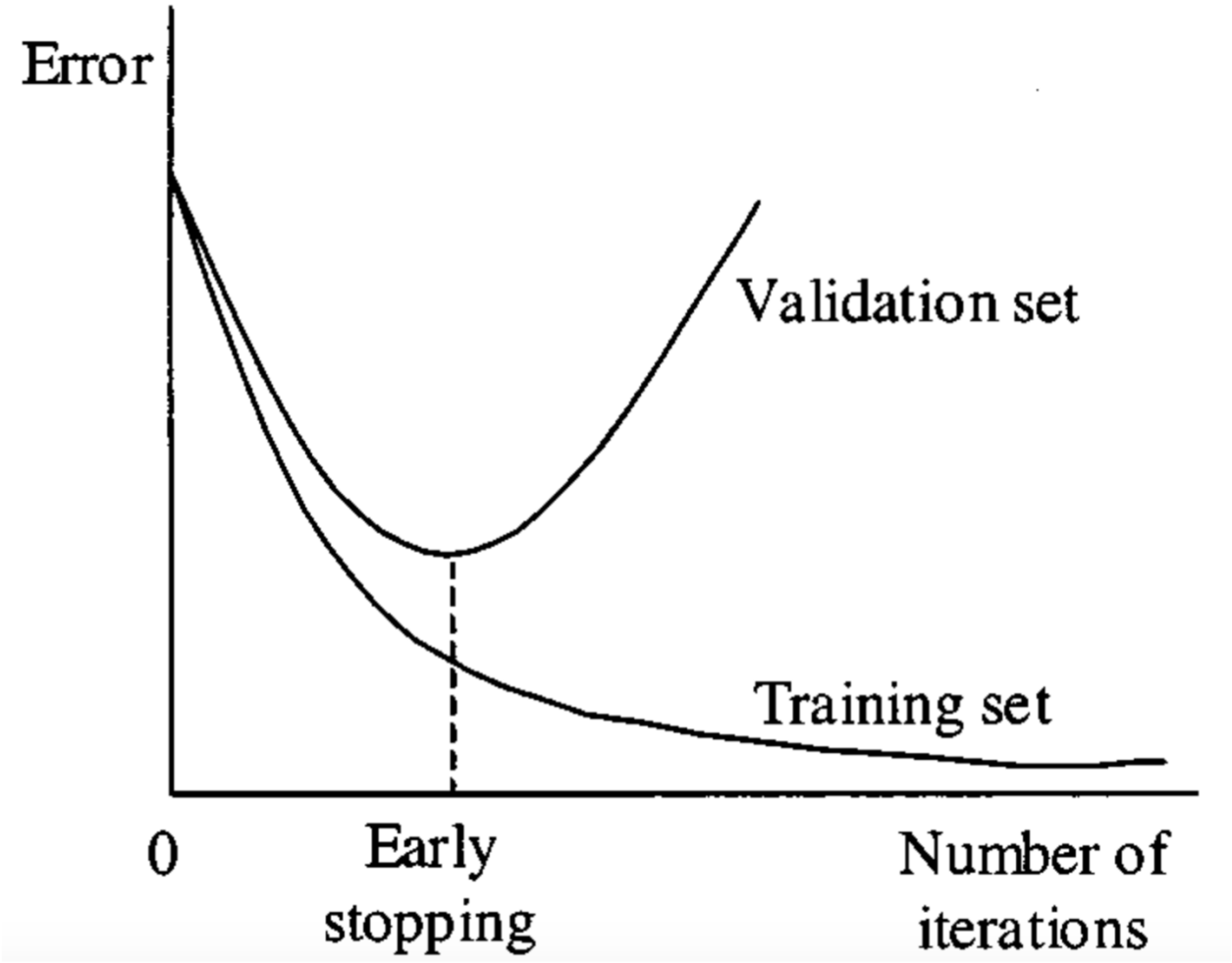


(b) After applying dropout.

Need to tune proportion of dropout in each layer.

Early Stopping

Fitting is terminated when the cost starts to increase on a holdout set.



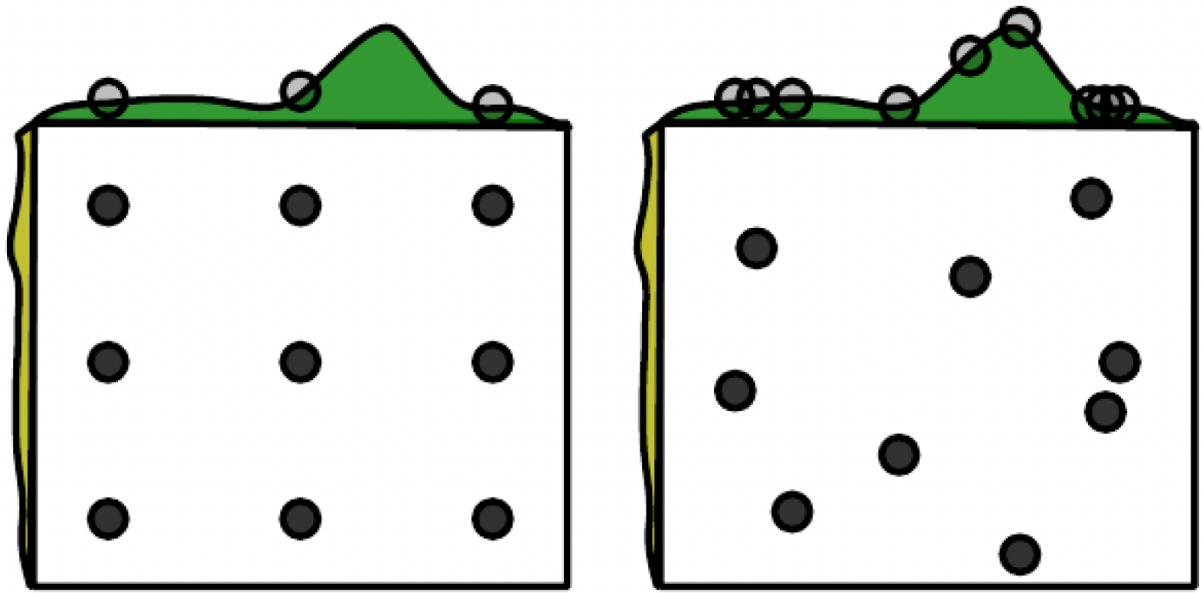
Actually, not much tuning here — but do need to decide what "increase" means.

How to Tune

Usually we pick a range of plausible values for each hyperparameter, then try each combination and compare the fitted models' performance — called *grid search*.

- E.g., if the learning rate $\varepsilon \in \{0.1, 0.01\}$ and the batch size $B \in \{100, 500\}$, we would fit four models for each possible combination

Alternately, can randomly sample combinations to make tuning faster — called *random search*.



Grid Random

How to Tune Better

There are more advanced hyperparameter tuning approaches.

An approach I've found to work well is *Bayesian optimization*, which predicts the best combination using a Gaussian process.

It's beyond the scope here, but if you're interested, check out the Supplemental Materials as well as the Ax Python package.

Learning Outcomes Checkpoint

Final checkpoint of the day!

- We've briefly discuss tuning deep learning model hyperparameters and model comparison.

Any questions before coding?