# Machine Learning: Deep Learning

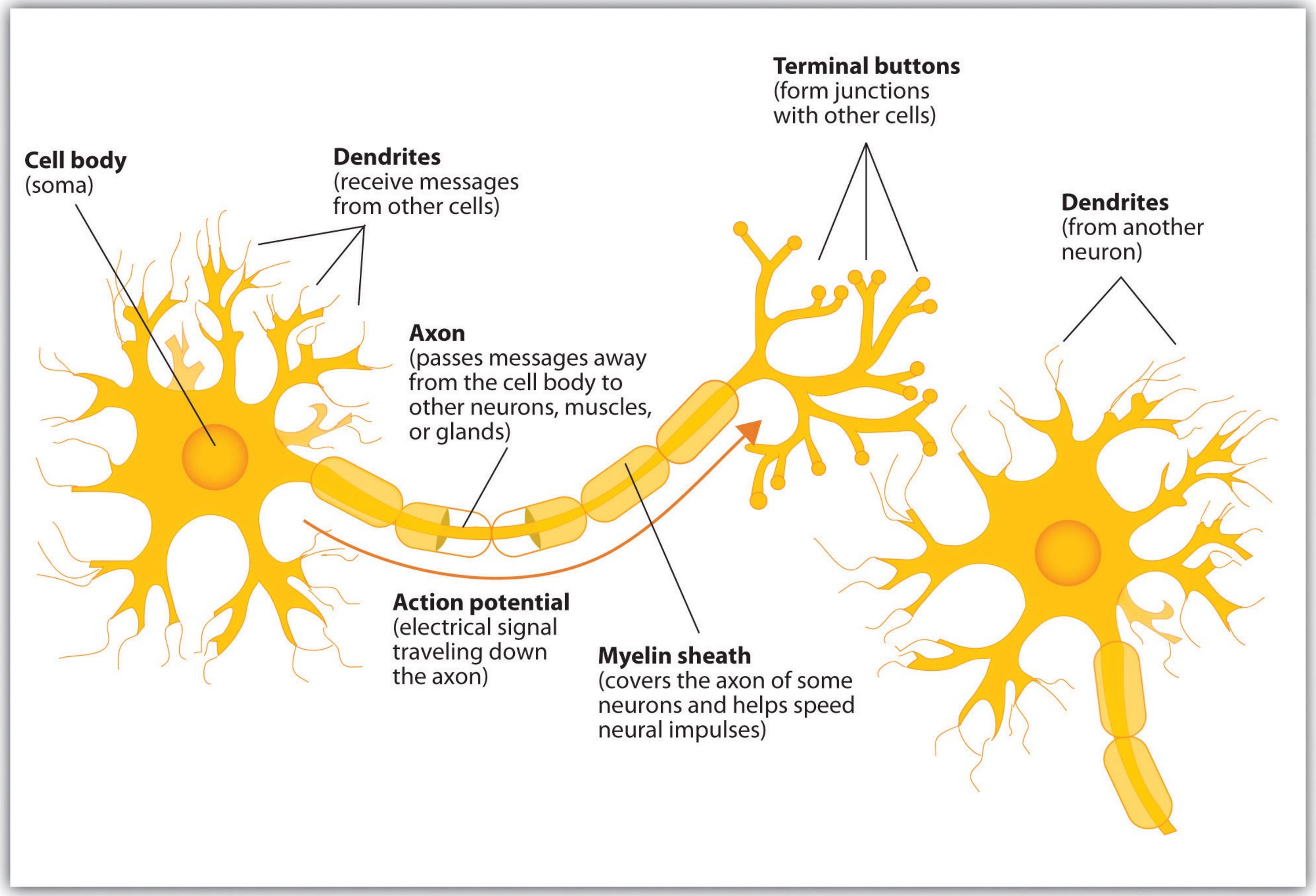
Forest Agostinelli
University of South Carolina

# Topics Covered in This Class

- **Part 1: Search**
  - Pathfinding
    - Uninformed search
    - Informed search
  - Adversarial search
  - Optimization
    - Local search
    - Constraint satisfaction
- **Part 2: Knowledge Representation and Reasoning**
  - Propositional logic
  - First-order logic
  - Prolog
- **Part 3: Knowledge Representation and Reasoning Under Uncertainty**
  - Probability
  - Bayesian networks
- **Part 4: Machine Learning**
  - Supervised learning
    - Inductive logic programming
    - Linear models
    - Deep neural networks
    - PyTorch
  - Reinforcement learning
    - Markov decision processes
    - Dynamic programming
    - Model-free RL
  - Unsupervised learning
    - Clustering
    - Autoencoders

# Artificial Neural Networks

- Inspired from biological neural networks
- Far from an exact model
- The main parallels are
    - Dendrites (inputs)
    - Action potential (activation function)
    - Axon terminals (outputs)

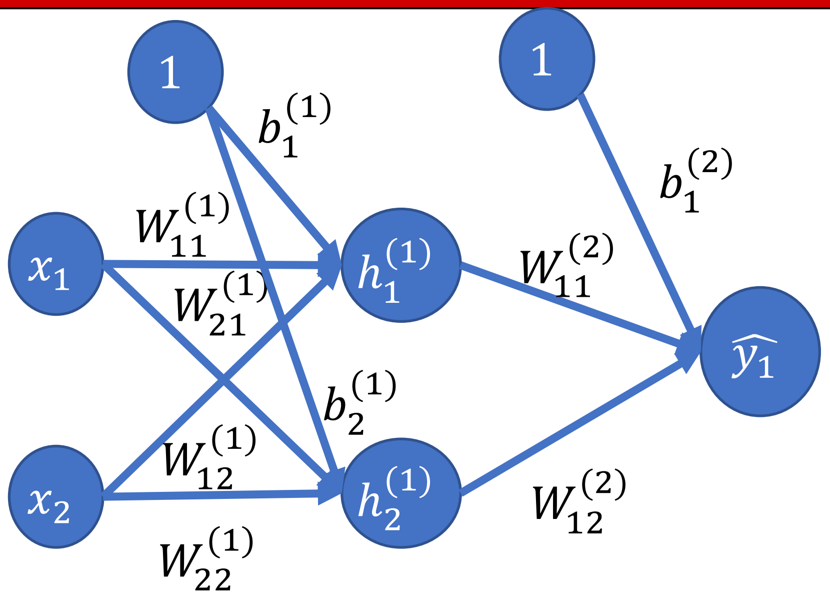


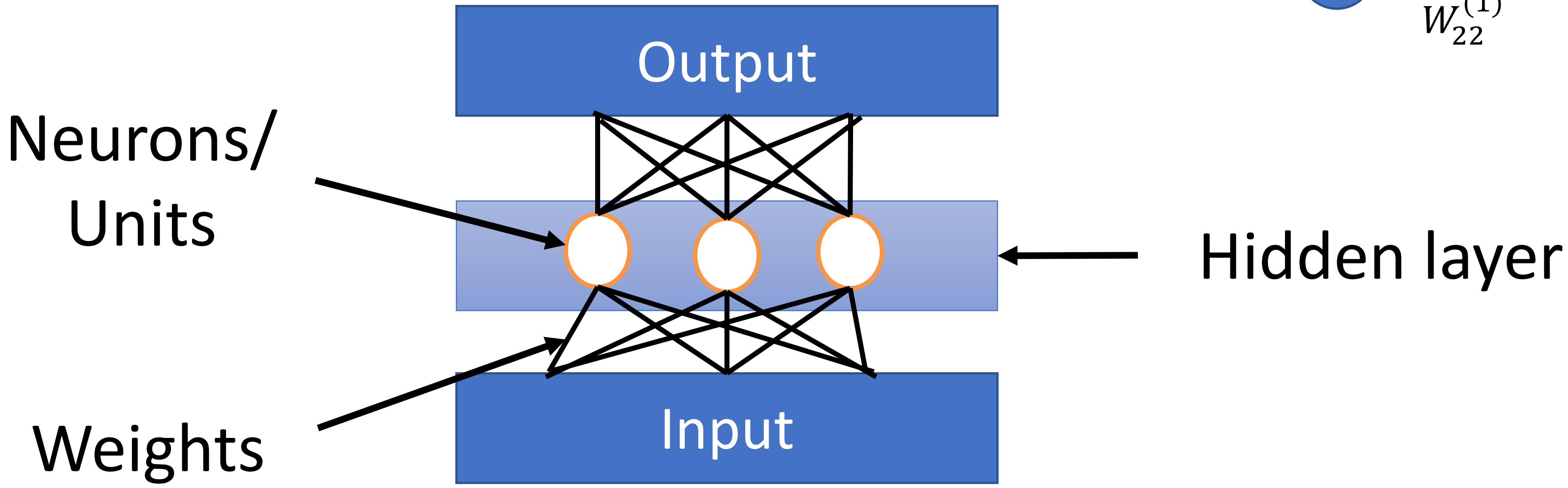

A biological neuron

Jennifer Walinga - https://opentextbc.ca/introductiontopsychology/chapter/3-1-the-neuron-is-the-building-block-of-the-nervous-system/

# Neural Networks

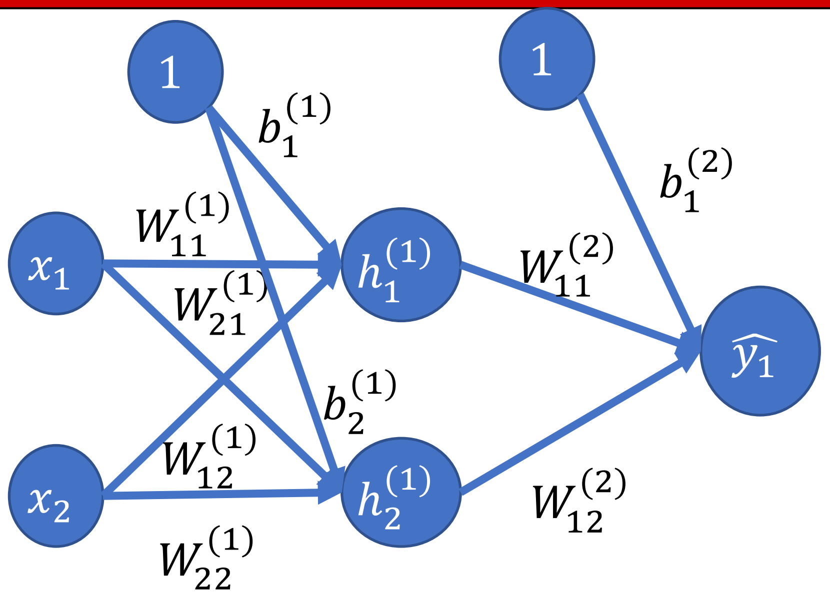
- $f(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{W}^{(2)}\sigma(\boldsymbol{W}^{(1)}\boldsymbol{x})$
  - $\boldsymbol{h}^{(1)} = \sigma(\boldsymbol{W}^{(1)}\boldsymbol{x})$
  - $f(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{W}^{(2)}\boldsymbol{h}^{1}$
- Where $\sigma$ is some non-linear function



# Quick Quiz: Linear Activation Functions

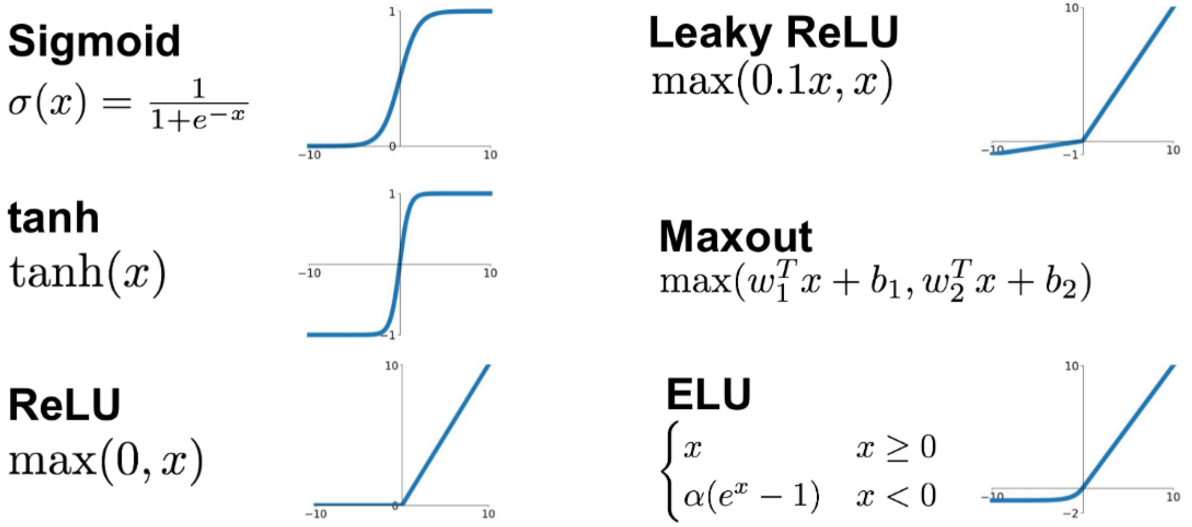
- $f(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{W}^{(2)}\sigma(\boldsymbol{W}^{(1)}\boldsymbol{x})$
  - $\boldsymbol{h}^{(1)} = \sigma(\boldsymbol{W}^{(1)}\boldsymbol{x})$
  - $f(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{W}^{(2)}\boldsymbol{h}^{1}$
- What if $\sigma$ is a linear function?

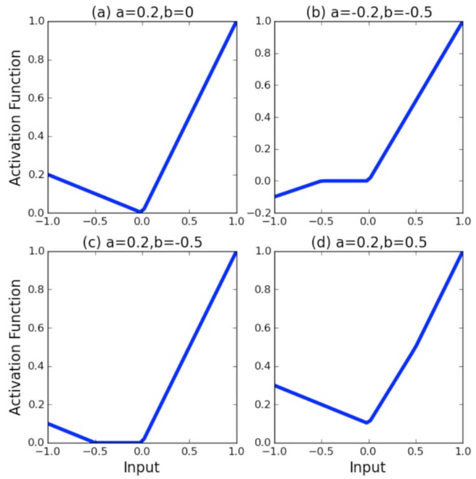



- The resulting function would also be linear. This is true no matter how deep the network is.

# Backpropagation: Activation Functions

- Allow neural network to learn non-linear functions
- Logistic (Sigmoid)
  - $\sigma(x) = \frac{1}{1+e^{-x}}$
  - $\sigma'(x) = \sigma(x)(1 - \sigma(x))$
- Rectified Linear Unit (ReLU)
  - $\sigma(x) = \max(0, x)$
  - $\sigma'(x) = 0$ if $x \leq 0$
  - $\sigma'(x) = 1$ if $x > 0$
  - Derivative undefined at zero but does not matter in practice
- Activation functions can also be parameterized and learned through gradient descent




**Adaptive piecewise linear units**

https://medium.com/@shrutijadon10104776/survey-on-activation-functions-for-deep-learning-9689331ba092

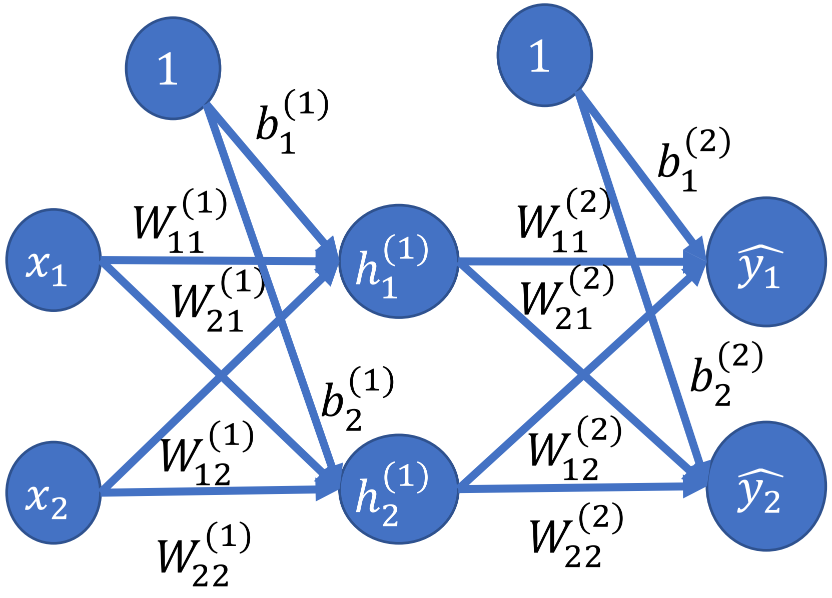

Agostinelli, F., Hoffman, M., Sadowski, P., & Baldi, P. (2014). Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*.

# Neural Networks: Universal Function Approximation

- Given enough hidden units, neural networks can approximate any function with arbitrary precision
- Cannot guarantee convergence

# Backpropagation

- We do gradient descent via backpropagation
  - Just the application of the chain rule
- $\boldsymbol{h}^{(1)} = \sigma(\boldsymbol{W}^{(1)}\boldsymbol{x})$
  - $h_j^{(1)} = \sigma(\sum W_{jk}^{(1)} x_k)$
- $f(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{W}^{(2)}\boldsymbol{h}^{(1)} = \hat{\boldsymbol{y}}$
  - $\hat{y}_i = \sum_j W_{ij}^{(2)} h_j^{(1)}$
- $E(\boldsymbol{w}) = \frac{1}{2}\|(\boldsymbol{y} - \hat{\boldsymbol{y}})\|_2^2 = \frac{1}{2}\|\boldsymbol{e}\|_2^2$
  - $= \frac{1}{2}\sum_i (y_i - \hat{y}_i)^2 = \frac{1}{2}\sum_i e_i^2$
- $\frac{\partial E(\mathbf{w})}{\partial W_{ij}^{(2)}} = \frac{\partial E(\mathbf{w})}{\partial \boldsymbol{e}} \frac{\partial \boldsymbol{e}}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial W_{ij}^{(2)}} = -(y_i - \hat{y}_i) h_j$
- $\frac{\partial E(\mathbf{w})}{\partial W_{jk}^{(1)}} = \frac{\partial E(\mathbf{w})}{\partial \boldsymbol{e}} \frac{\partial \boldsymbol{e}}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{h}^{(1)}} \frac{\partial \boldsymbol{h}^{(1)}}{\partial W_{jk}^{(1)}} = \sum_i -(y_i - \hat{y}_i) W_{ij}^{(2)} \sigma'(\sum W_{jk}^{(1)} x_k)\, x_k$

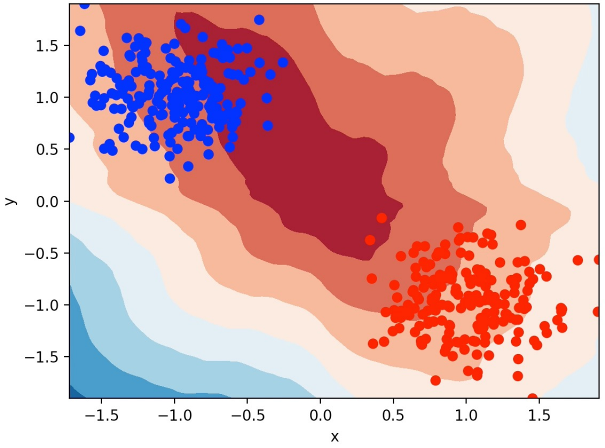


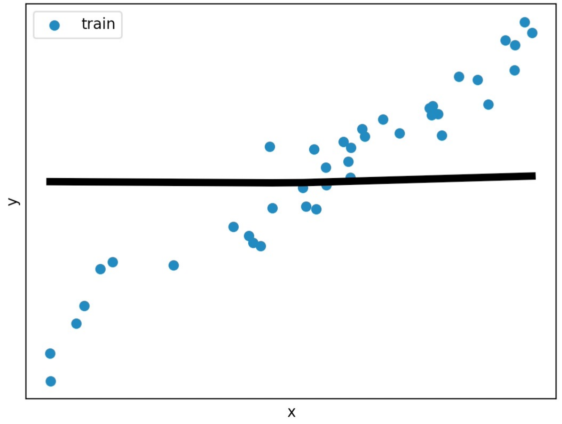
# Neural Networks: Regression and Classification

Classification

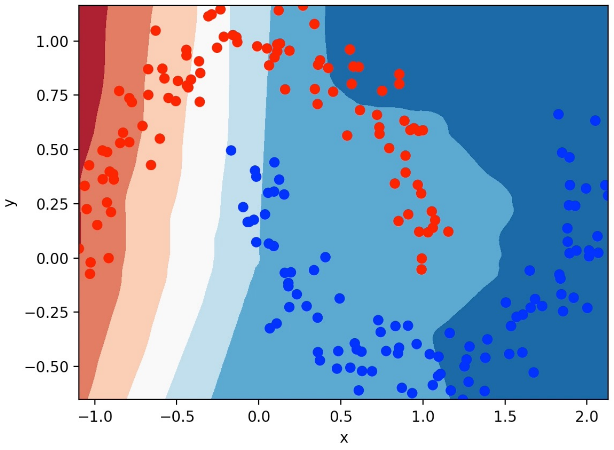
Regression

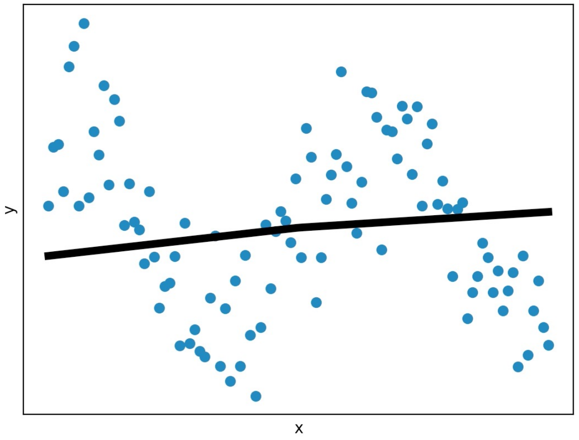
Linear




Non-Linear

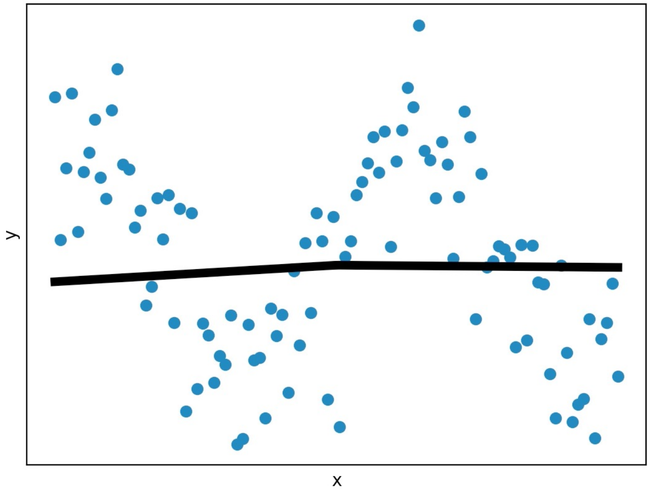


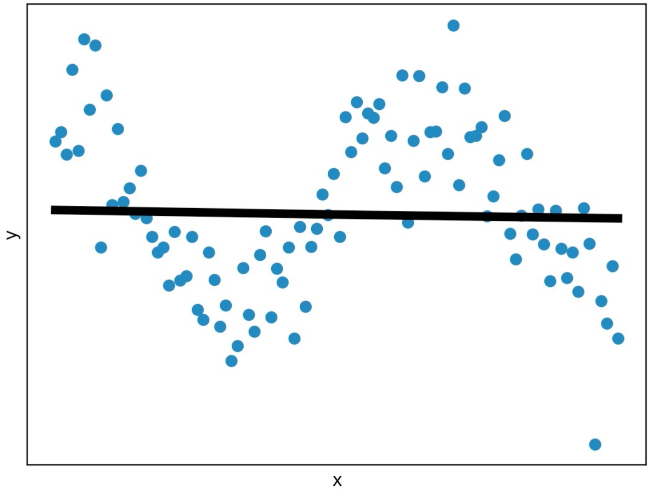
# Hyperparameters

- Parameters are learned from the data
- Hyperparameters are set before training
- Learning rate
  - Defines how large the steps will be during gradient descent
  - Usually denoted by $\alpha$
- Number of neurons
  - How “wide” the neural network is
- Many more!

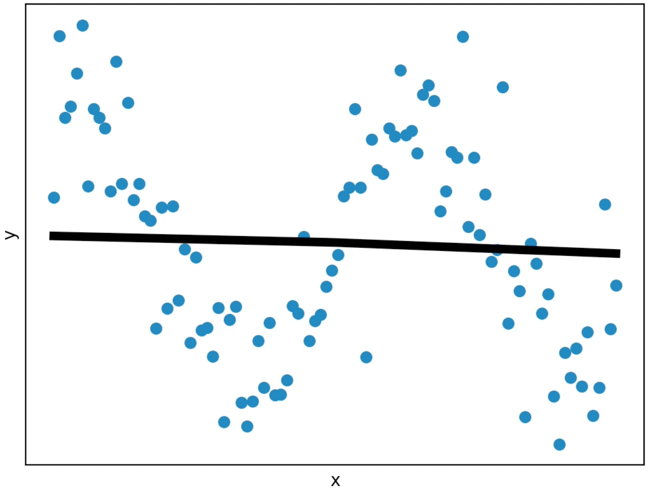
# Quick Quiz: What are the Best Hyperparameters?



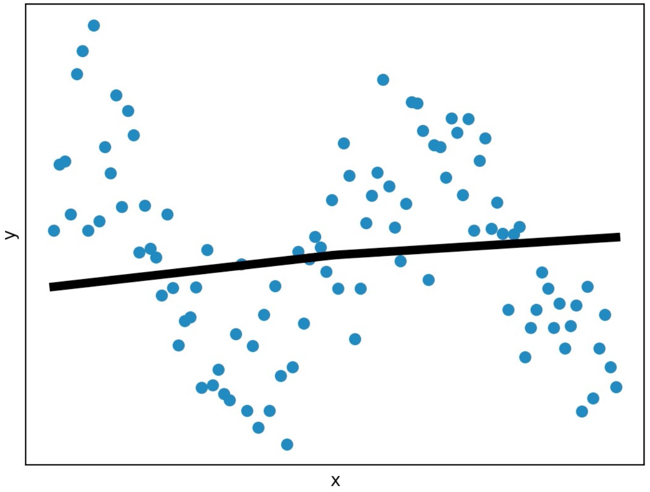

$\alpha = 0.1$, neurons = 100



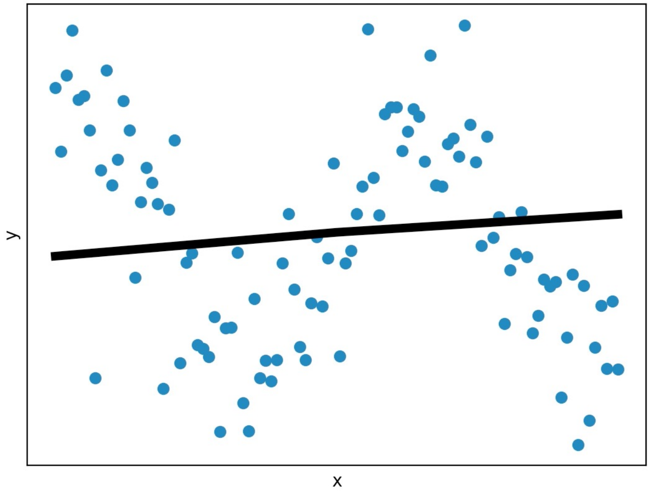

$\alpha = 0.25$, neurons = 100



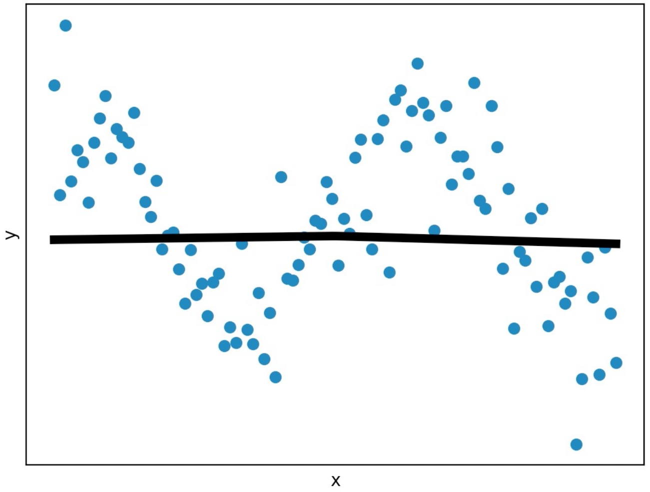

$\alpha = 0.75$, neurons = 100




$\alpha = 0.1$, neurons = 1000




$\alpha = 0.25$, neurons = 1000



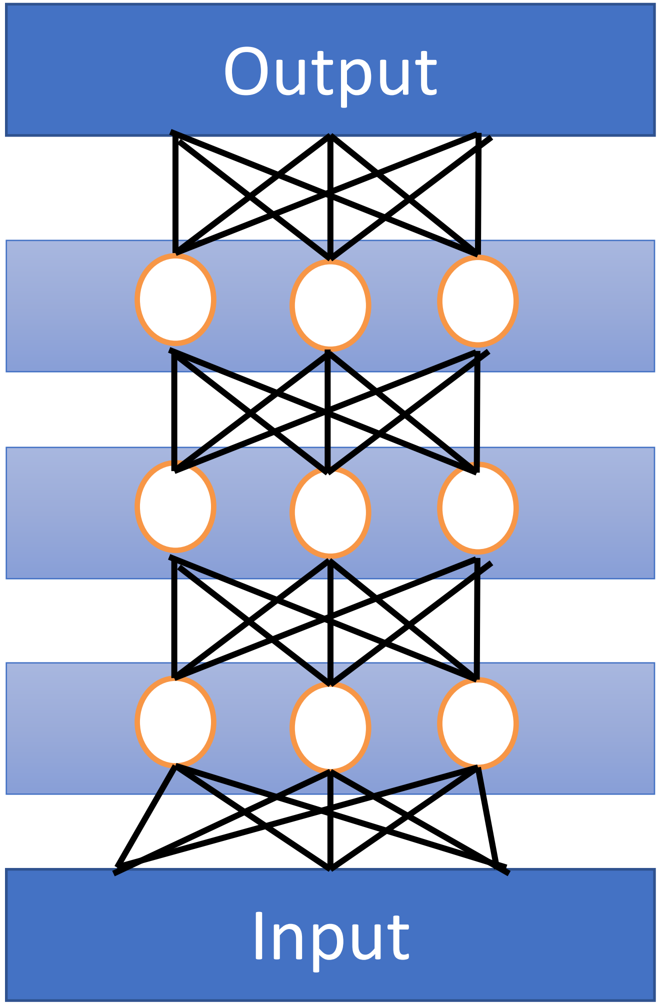

$\alpha = 0.75$, neurons = 1000

# Machine Learning

- There are many different machine learning methods
  - Linear models
  - Deep neural networks
  - Support vector machines
  - Decision trees
  - K-nearest neighbors
- The type of model to use depends on your data
- Deep learning is often the best out of these methods when the data
  - High-dimensional
  - Low-level
  - Plentiful
  - Has a non-linear relationship between the input and the output

# Deep Neural Networks

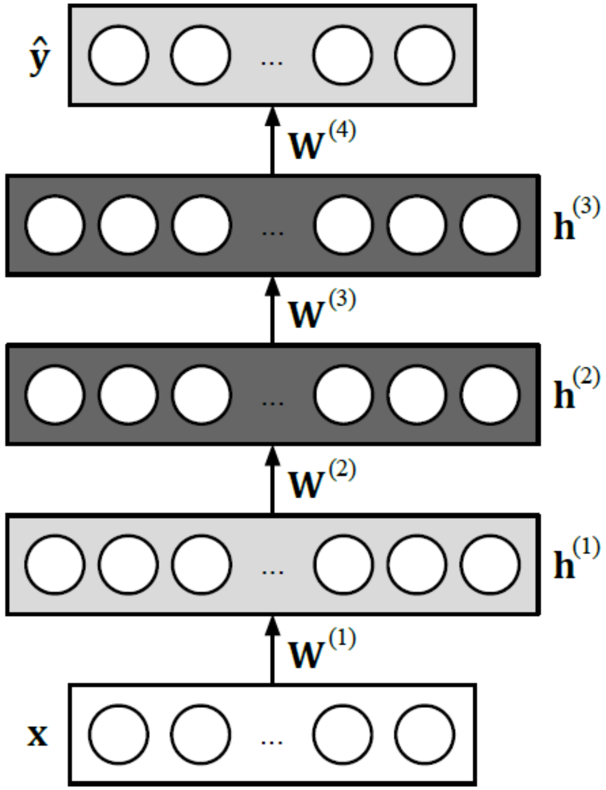
- Stack hidden layers to obtain a deep neural network
- "Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction."

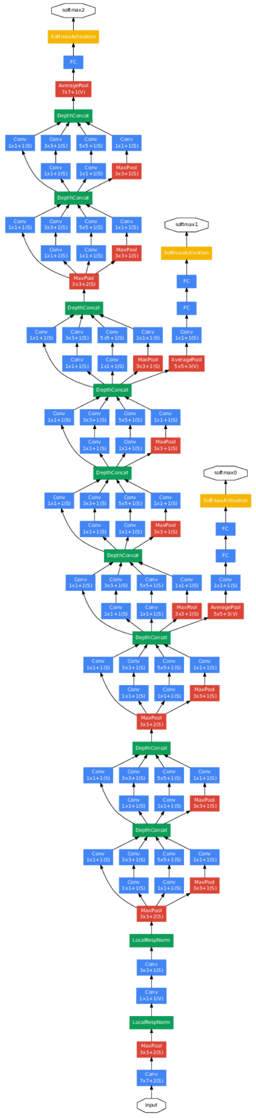


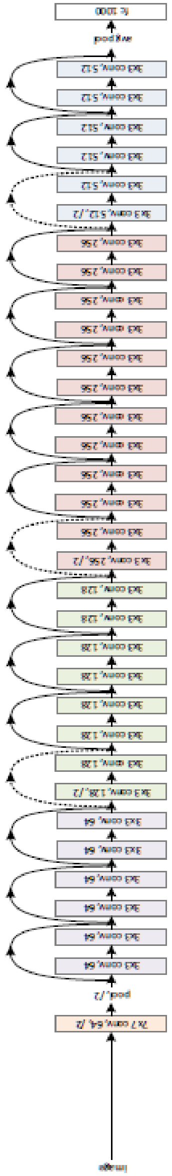

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. nature, 521(7553), 436.

# Deep Neural Networks

- There are a wide variety of ways one can create a deep neural network

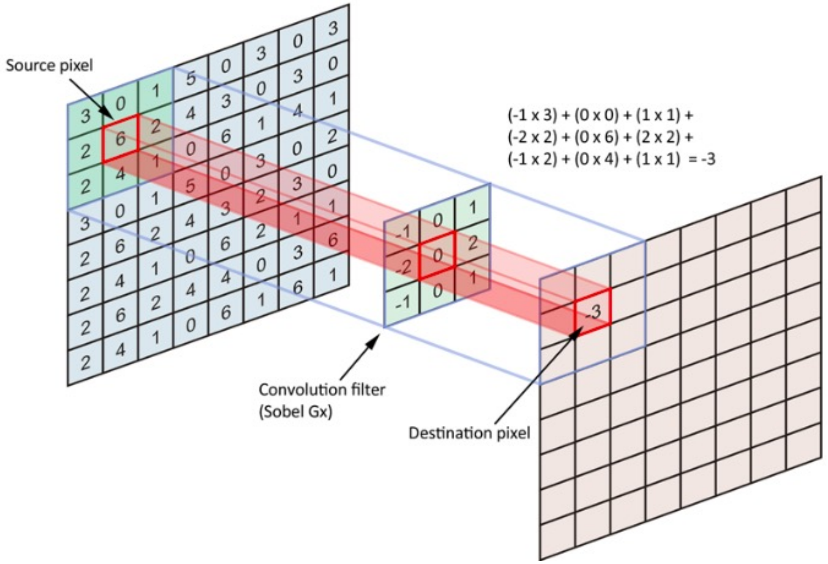


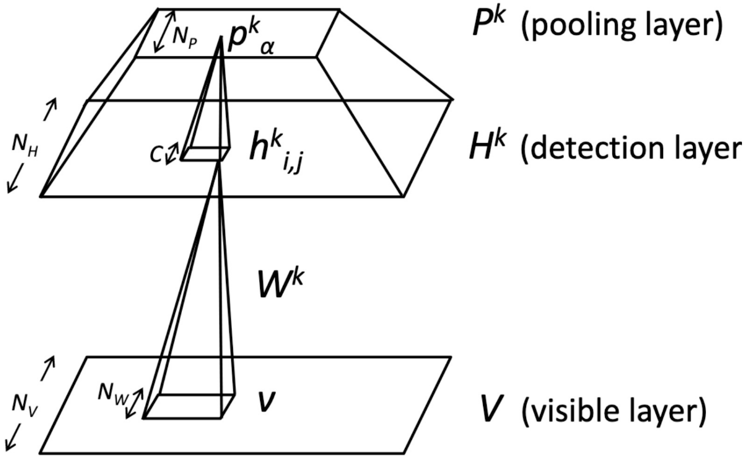





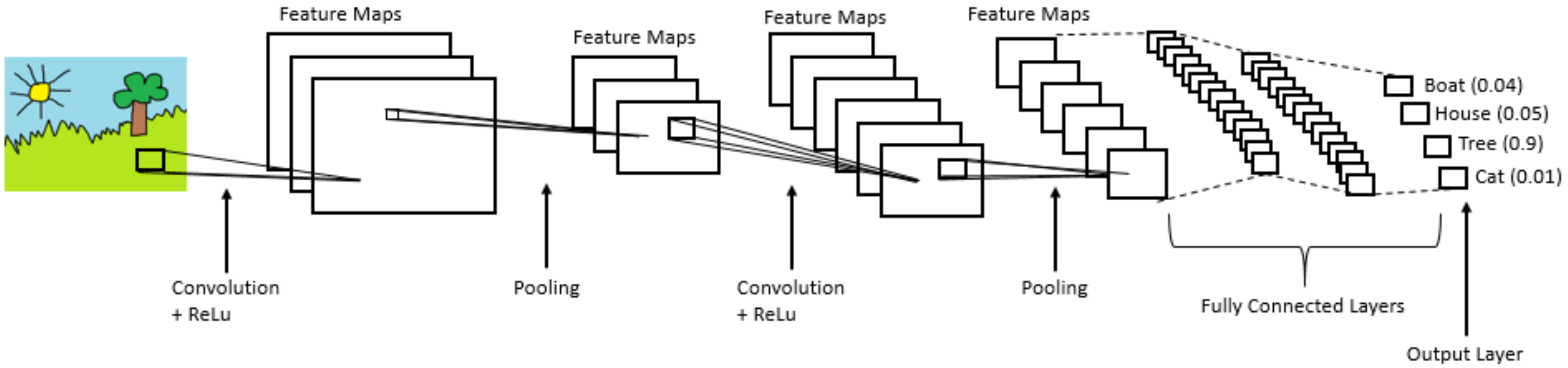
# Convolution and Pooling

- We can design neural networks to take advantage of structured data, such as images
- Convolution helps to add translation invariance
  - Strong inductive bias
- Pooling shrinks the representation, allowing subsequent layers to focus on higher-level information and have a larger receptive field



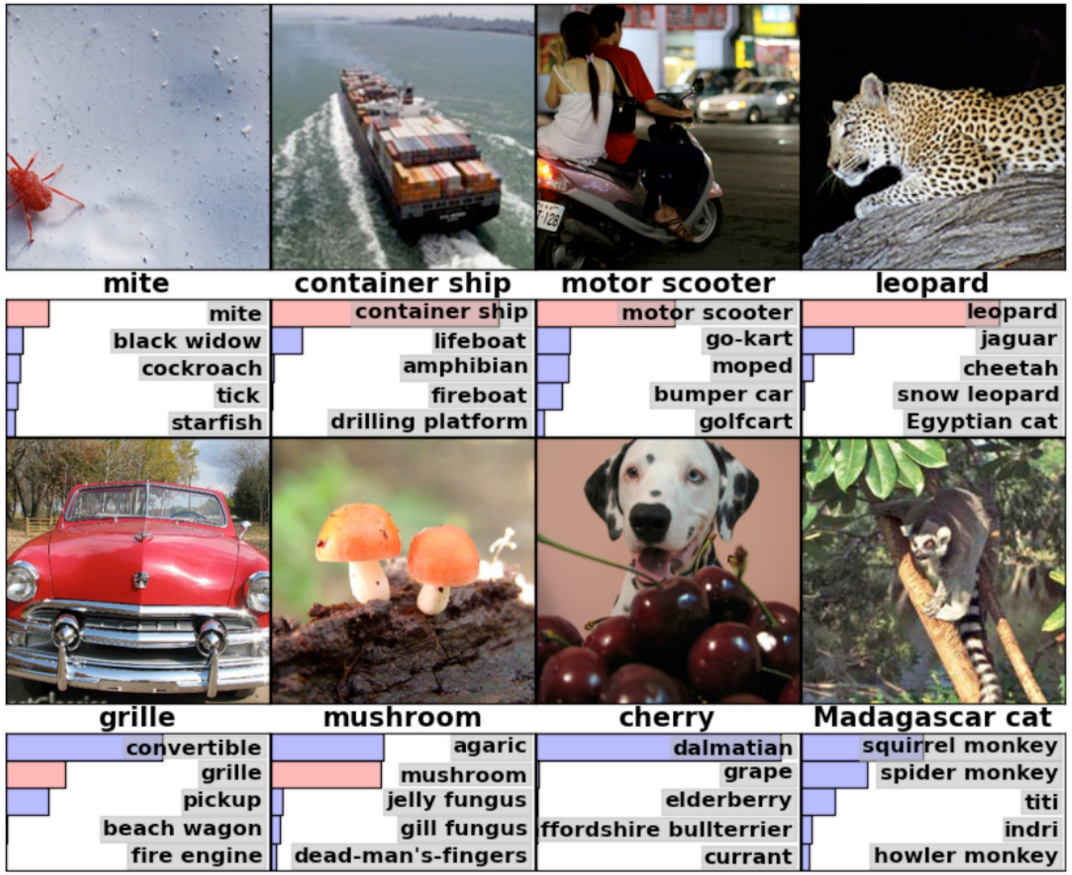



# Convolution and Pooling

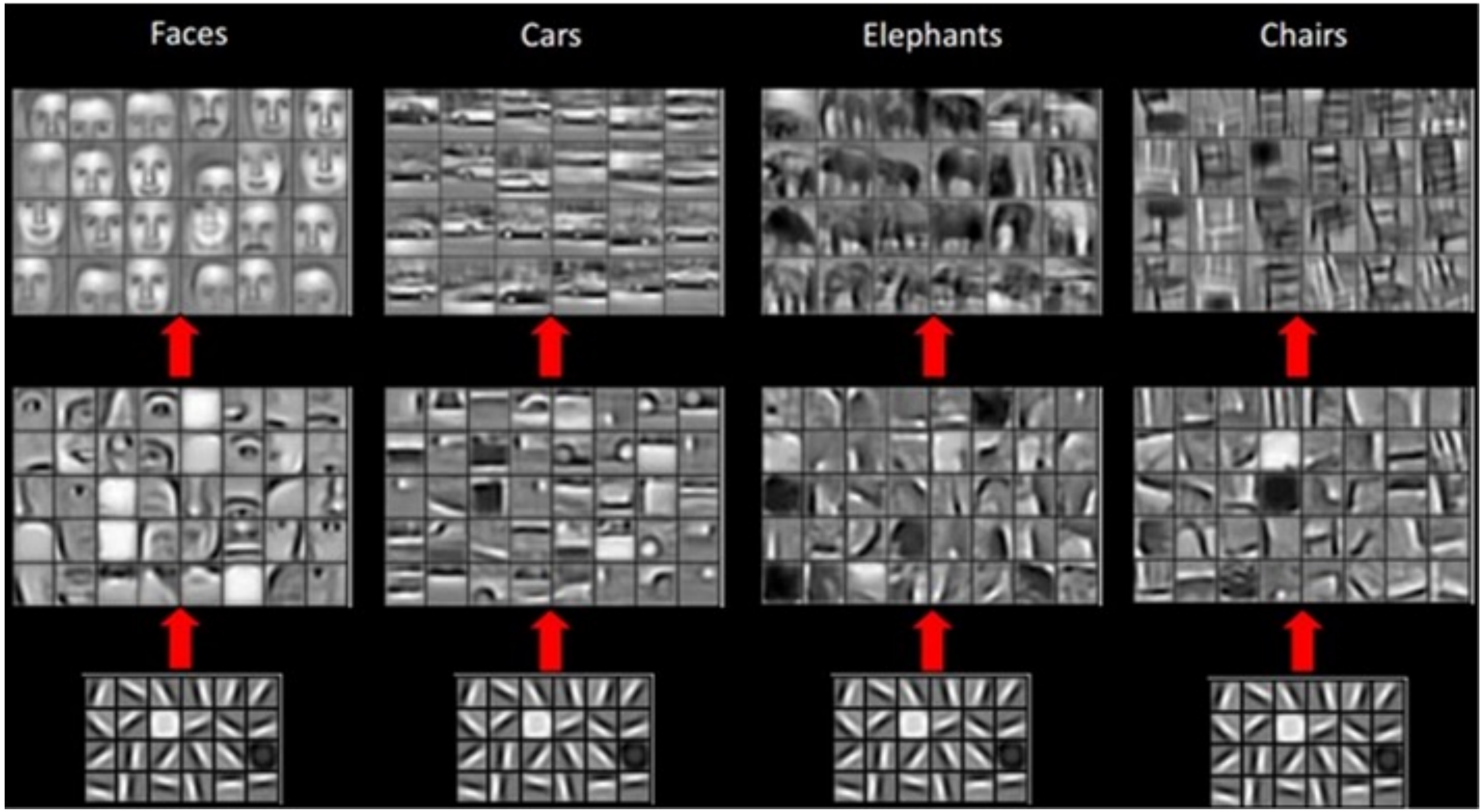


# Convolution and Pooling

- Was central to the breakthrough on the Imagenet dataset



# Deep Neural Networks




Layer 3

Layer 2

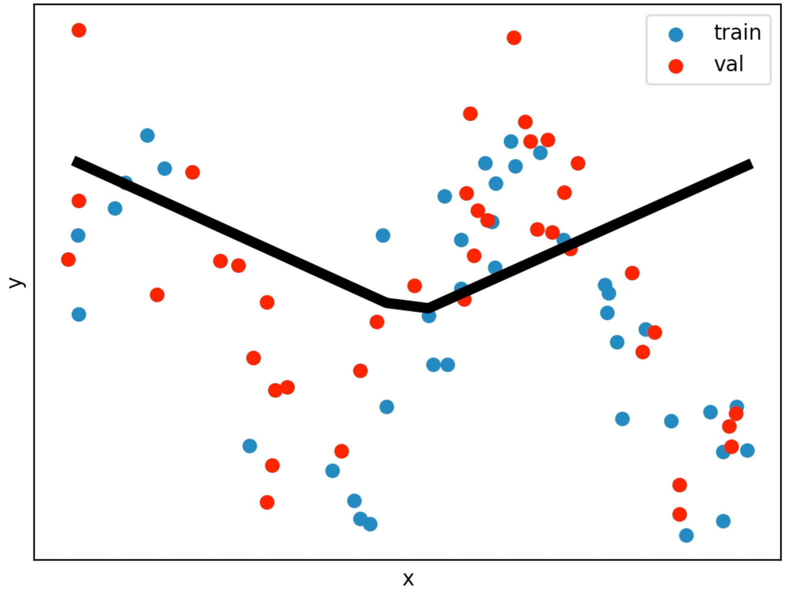
Layer 1

Lee, H., Grosse, R., Ranganath, R., & Ng, A. Y. (2009, June). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual ICML* (pp. 609-616). ACM.

# Overfitting/Regularization

- **Training Dataset**: Used to train neural network
- **Validation Dataset**: Not used to train neural network. Used to determine how well neural network generalizes
- **Test Dataset:** Only for seeing the final performance of neural network. Not used for training or validation.

Loss – Train: 0.042, Val: 0.35

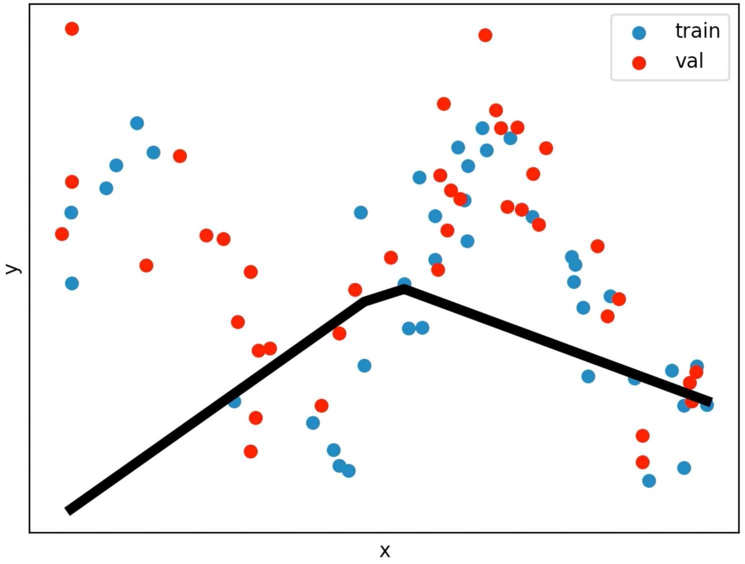



DNN 5 hidden layers and 500 neurons per layer

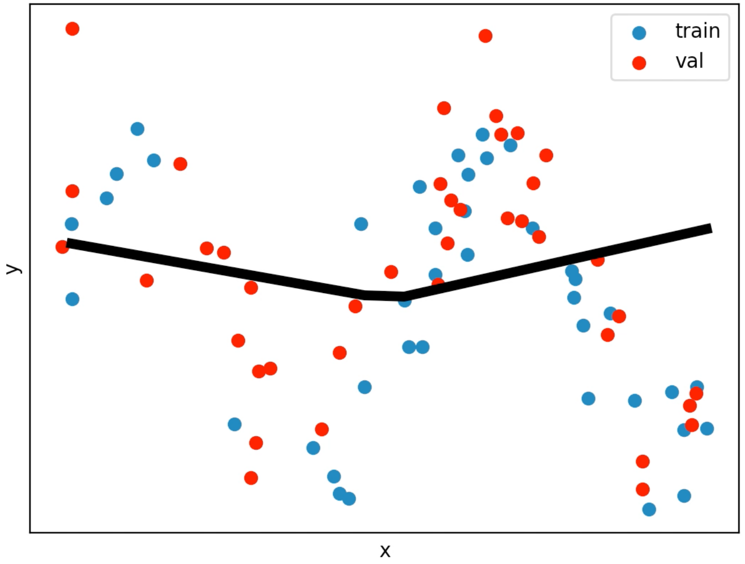
# Regularization: Weight Regularization

- Weight regularization
  - $E(\boldsymbol{w}) = \frac{1}{2n} \sum_n || \boldsymbol{y}_n - \widehat{\boldsymbol{y}}_n ||_2^2 + \lambda \sum_l \sum_i \sum_j W_{ij}^2$
  - Make the weights less "sensitive" to the input

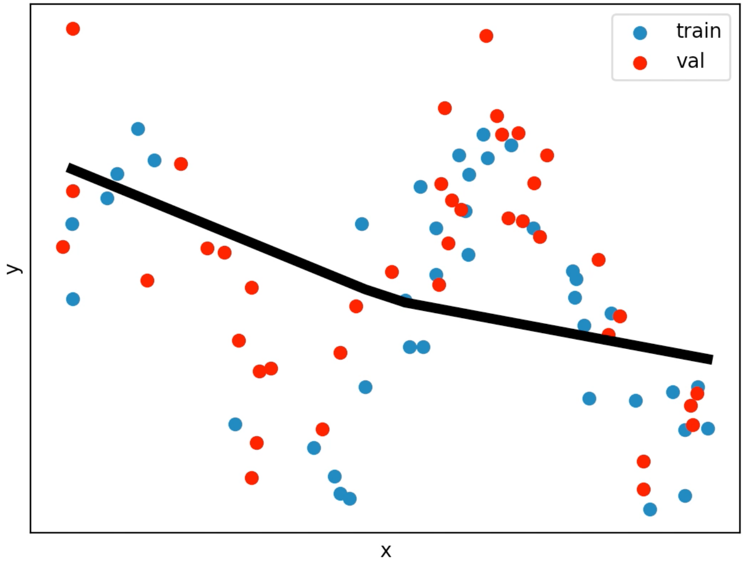
Loss – Train: 0.12, Val: 0.29, $\lambda = 0.01$




Loss – Train: 0.18, Val: 0.26, $\lambda = 0.05$

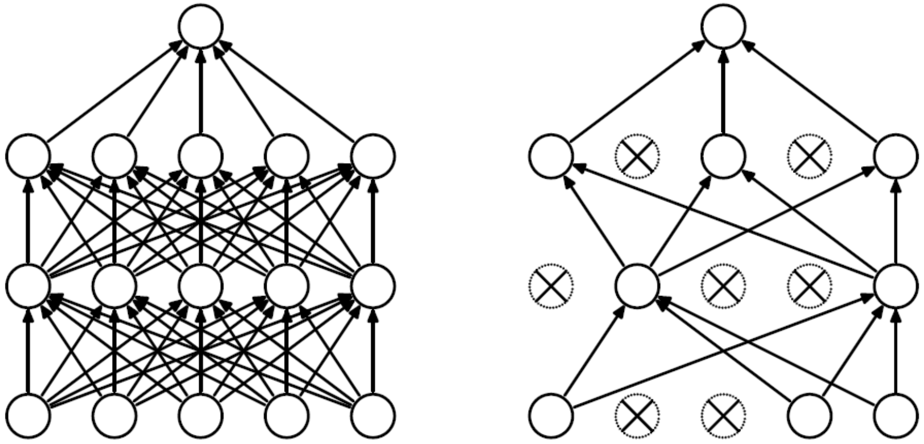



Loss – Train: 0.70, Val: 0.77, $\lambda = 0.2$



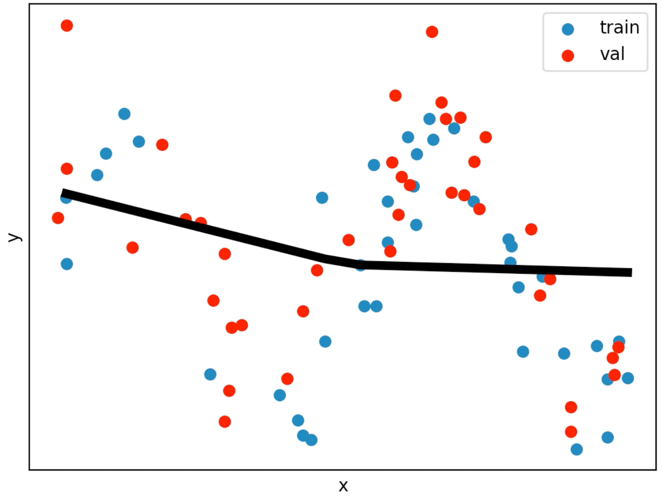
# Overfitting/Regularization: Dropout

- Dropout
  - Randomly drop connections between neurons during training

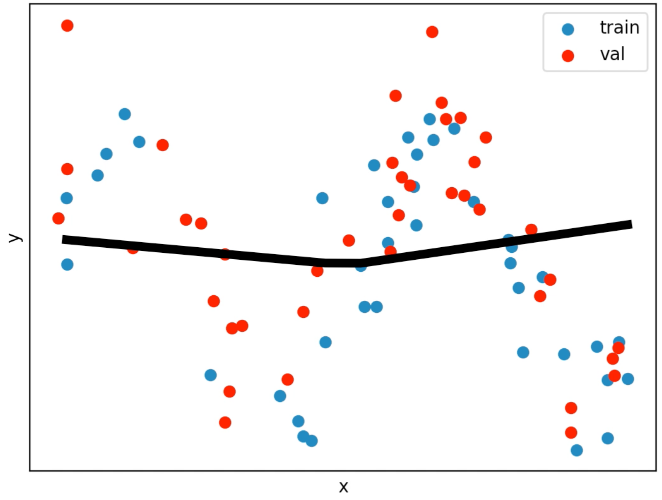


(a) Standard Neural Net (b) After applying dropout.

Loss – Train: 0.23, Val: 0.25, $p = 0.5$



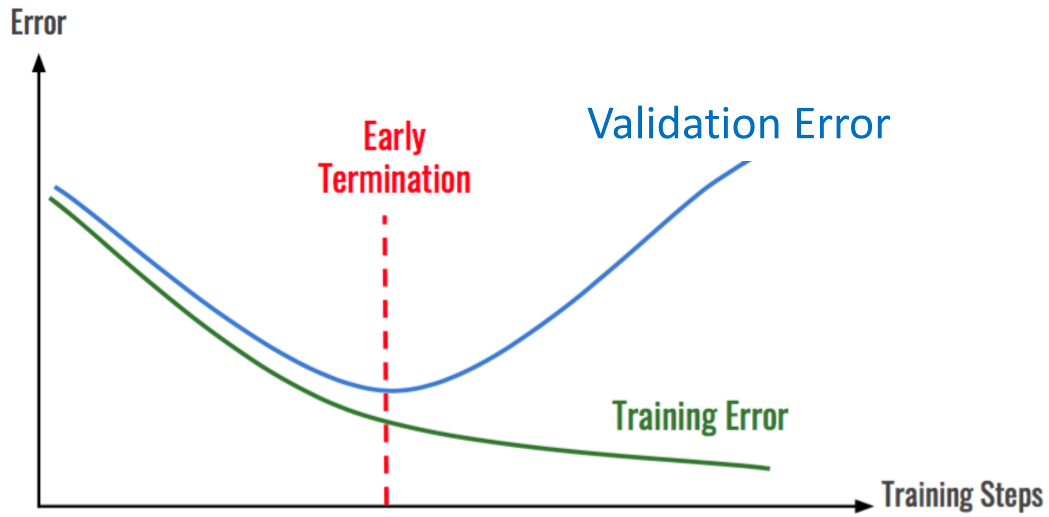

Loss – Train: 0.70, Val: 0.75, $p = 0.9$



# Regularization: Add More Data

- Harder to overfit if there is more data
- Collect more data
- Augment current data
    - Rotations, flipping, translations
    - Adding noise

# Overfitting/Regularization

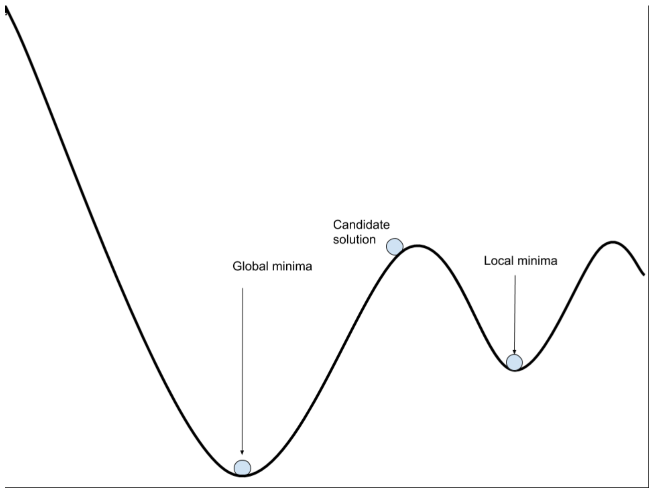



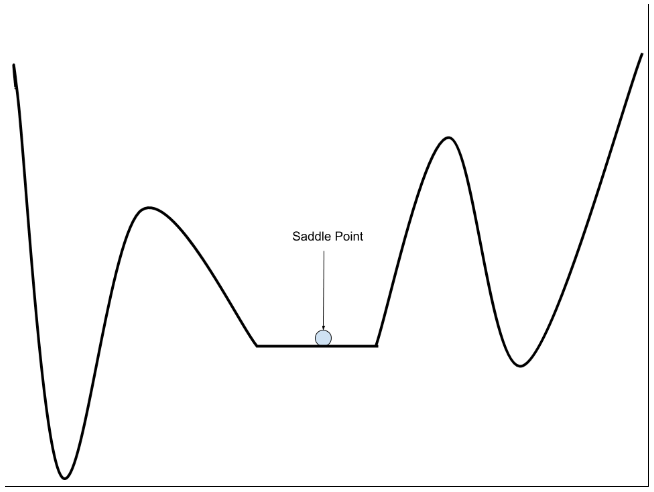
# Unbalanced Datasets

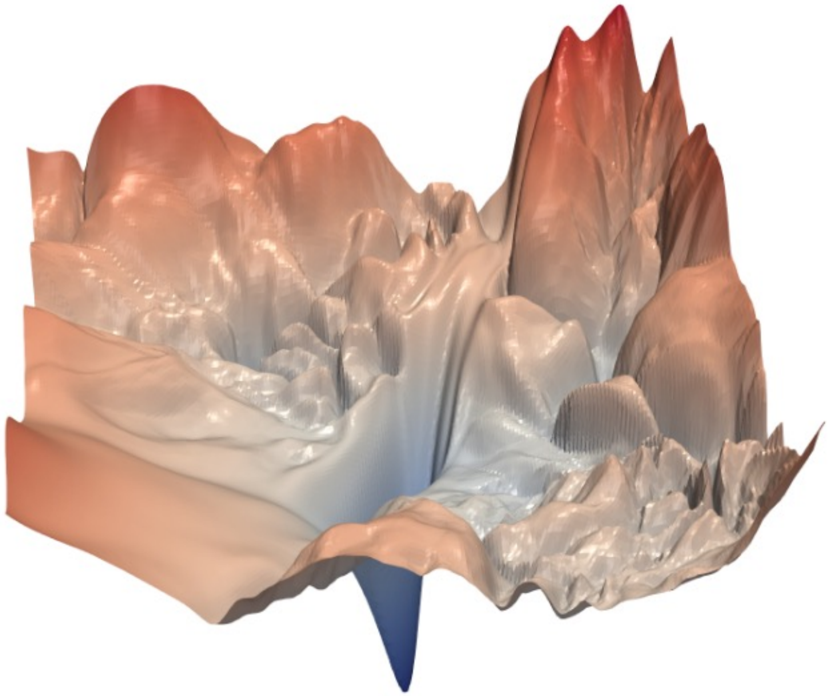
- Some datasets have classes that are significantly overrepresented
  - For example, I have many weather readings that were not followed by a hurricane and only a few that were followed by a hurricane
- If the training data reflects this imbalance, the model can get good prediction simply by being biased towards the overrepresented class
- Therefore, one must sample their training data so that it is balanced

# Optimization: Loss Surface

- No longer convex
- Local minima
- Saddle points



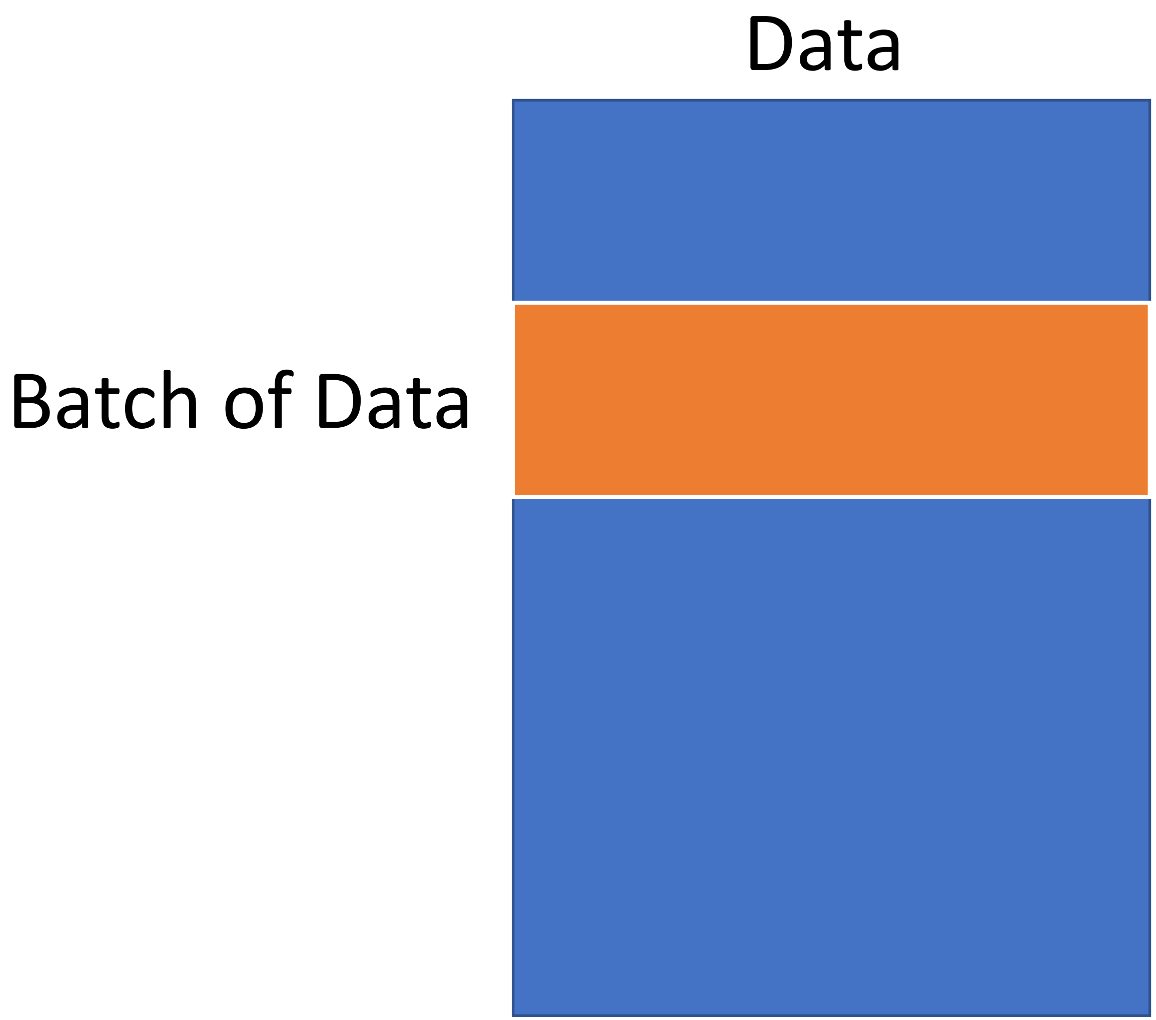



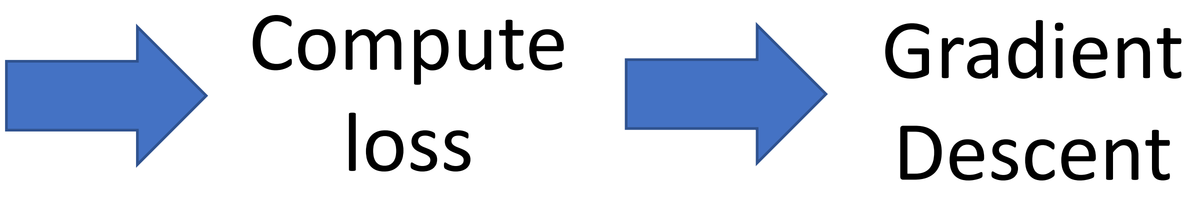



# Optimization: Gradient Based Methods

- Vanilla Gradient Descent
  - $\boldsymbol{w} = \boldsymbol{w} - \alpha \nabla_{\mathbf{w}} \mathrm{E}(\mathbf{w})$
- Gradient Descent with Momentum
  - $\boldsymbol{v} = \mu \boldsymbol{v} + \alpha \nabla_{\mathbf{w}} \mathrm{E}(\mathbf{w})$
  - $\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{v}$
- ADAM
  - $\boldsymbol{m} = \beta_1 \boldsymbol{m} + (1 - \beta_1)(\nabla_{\mathbf{w}} \mathrm{E}(\mathbf{w}))^2$ //estimate of the mean of the gradients
  - $\boldsymbol{v} = \beta_2 \boldsymbol{v} + (1 - \beta_2)(\nabla_{\mathbf{w}} \mathrm{E}(\mathbf{w}))^2$ //estimate of the variance of the gradients
  - $\widehat{\boldsymbol{m}}$ and $\widehat{\boldsymbol{v}}$ are bias corrected estimates of the mean and variance
  - $\boldsymbol{w} = \boldsymbol{w} - \frac{\alpha}{\sqrt{\widehat{\boldsymbol{v}}} + \epsilon} \widehat{\boldsymbol{m}}$
- Many others: https://ruder.io/optimizing-gradient-descent/

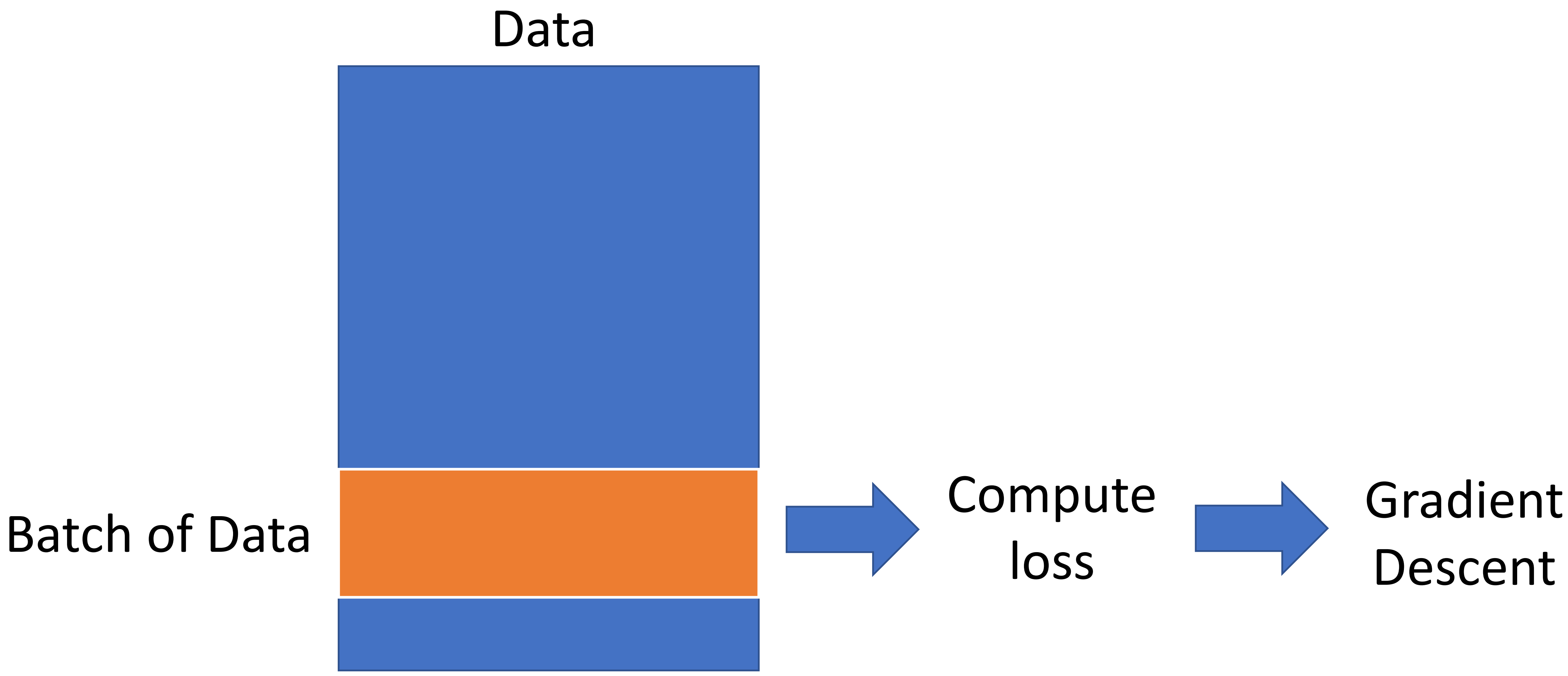
# Optimization: Stochastic Gradient Descent



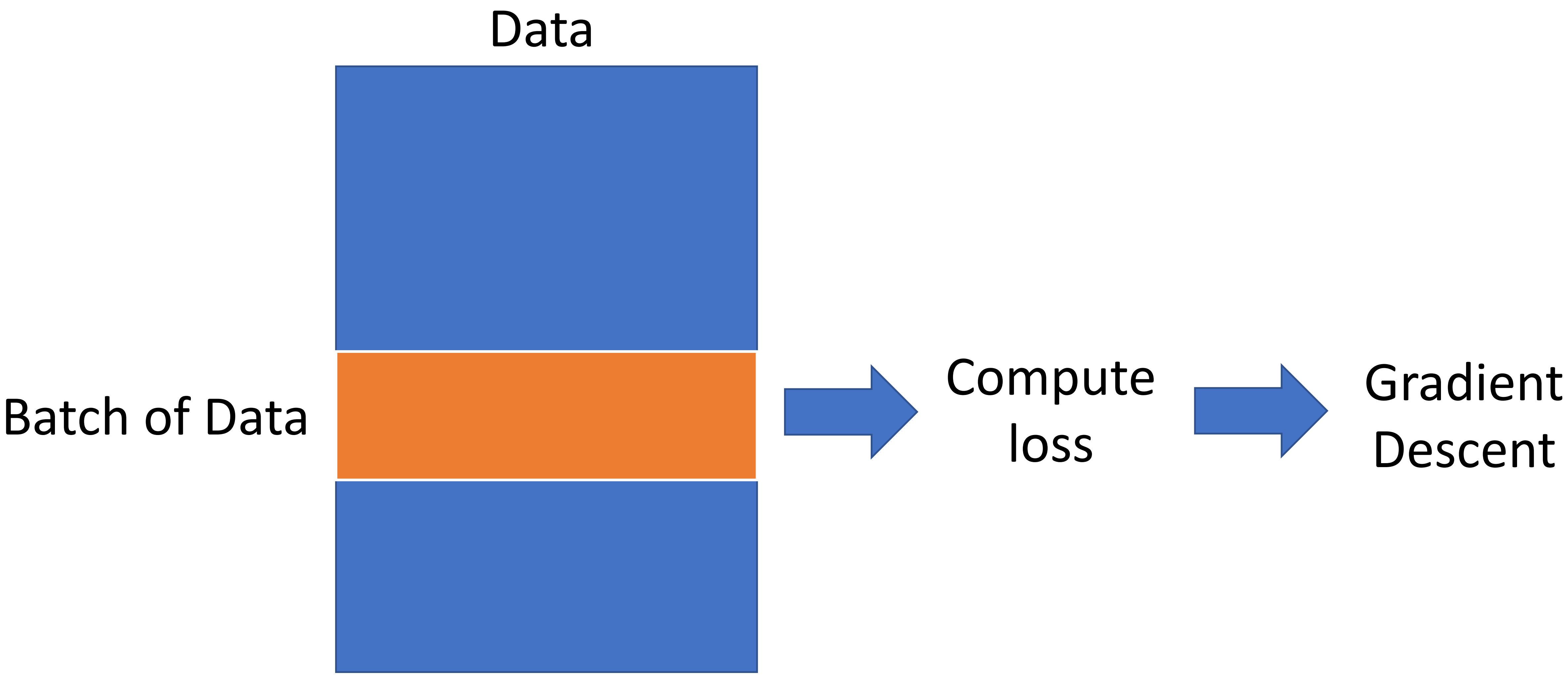




- There can be millions of datapoints
  - Gradient descent will take too long
- Do gradient descent on random subsets of the data, instead

# Optimization: Stochastic Gradient Descent

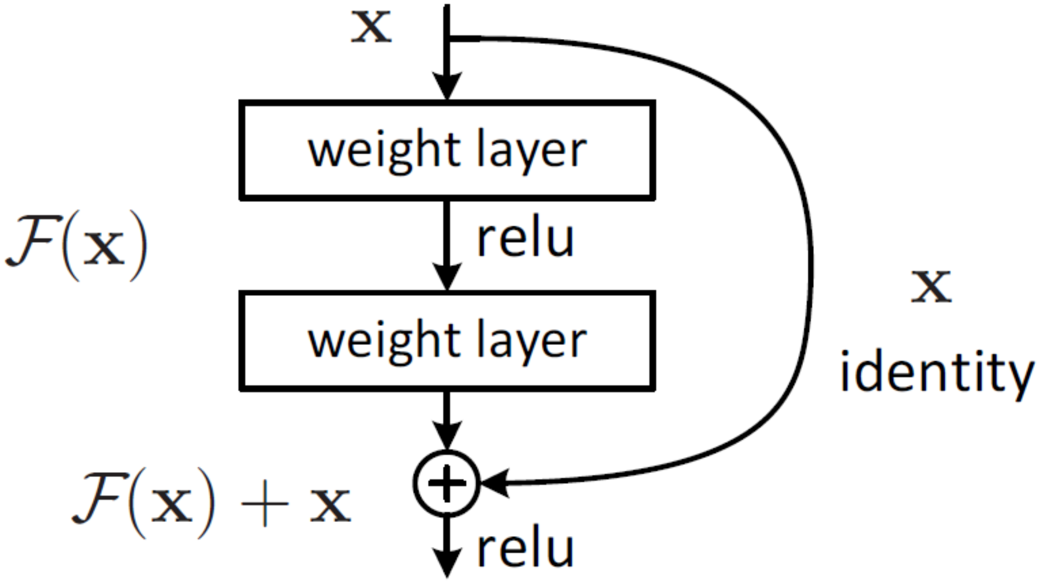


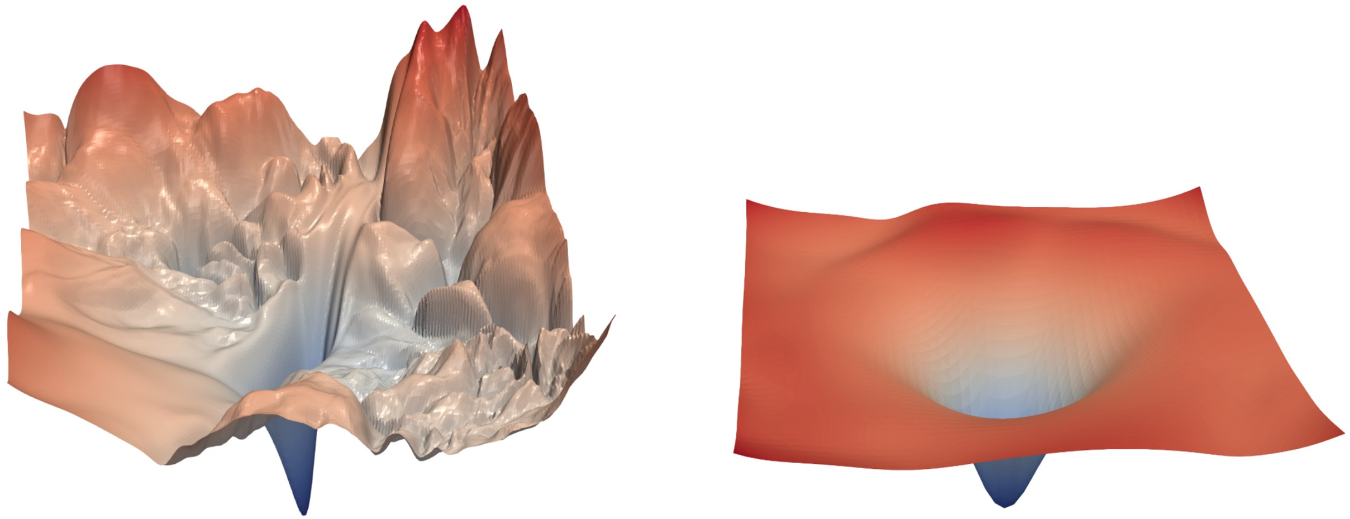
# Optimization: Stochastic Gradient Descent



# Optimization: Residual Neural Networks

- Training can become more difficult as the number of layers increases
- Adding skip connections allows us to train networks with hundreds of layers



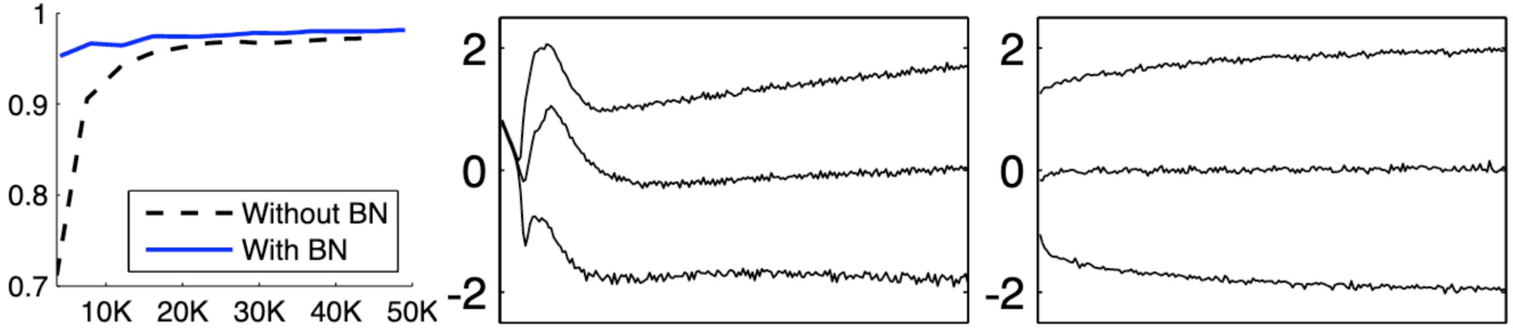
(a) without skip connections

(b) with skip connections

Loss surface

# Optimization: Batch Normalization

- Normalizes the input to the activation function to have a of mean 0 and standard deviation of 1
- Stabilizes training
  - Allows larger learning rates
  - Reduces importance of initialization
- $\boldsymbol{H} = \sigma(BN(\boldsymbol{WX}))$
- Adds some regularization




(a) (b) Without BN (c) With BN

# Optimization: Initialization

- The weights of the DNN are randomly initialized
- Initialization can play a large role in optimization
- Xavier/Glorot initialization is fairly common
- Initialization matters less when doing
    - Batch normalization
    - Weight normalization

# What to Try?

- Activation Function
  - Rectified Linear Units
- Gradient-Based Optimization
  - SGD with momentum
  - ADAM
- Convolution (for structured input like images or sound)
- Batch Normalization
- Residual Networks
- If overfitting?
  - Weight regularization
  - Dropout
    - In higher layers first

# Deep Learning/Machine Learning Demos

- https://p.migdal.pl/interactive-machine-learning-list/
- https://cs.stanford.edu/people/karpathy/convnetjs/demo/regression.html

# Relevant Papers


- **Xavier/Glorot Init:** Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010.
- **SGD w/ Momentum:** Sutskever, Ilya, et al. "On the importance of initialization and momentum in deep learning." *International conference on machine learning*. 2013.
- **Imagenet:** Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*. 2012.
- **ADAM:** Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014).
- **Batch Normalization:** Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." *arXiv preprint arXiv:1502.03167* (2015).
- **Residual Networks**: He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.