Welcome and Deep Feedforward Networks

Deep Learning Decal

Hosted by Machine Learning at Berkeley

Welcome!

ML@B

Agenda

Course Logistics

Background on Machine Learning

Deep Feedforward Networks

Optimization

Regularization

Questions

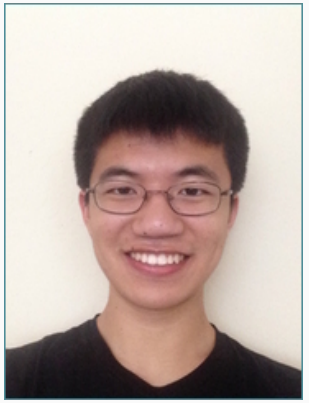
Course Logistics

Instructors

James Bartlett

Jordan Prosky

Quinn Tran



Michael Zhang

Neel Kant

Course Details

ML@B

- Wednesday 5:00 - 7:00 pm
  - Reading group from 6:00 - 7:00 pm
- Book: *Deep Learning*, by Ian Goodfellow, Yoshua Bengio, and Aaron Courville, available at deeplearningbook.org
  - Chapters 6-20

Grading

ML@B

- 40% attendance
- Project Option:
  - 5% Proposal
  - 5% Milestone 1
  - 10 % Milestone 2
  - 40 % Final Project Submission
- Reading Group Option:
  - 20 % Reading Group
  - 40 % Weekly Question Submission
- Choose which option best suites your goals for this class

Background on Machine Learning

Kinds of Problems in Machine Learning

ML@B

- Supervised Learning
- Unsupervised Learning
- Semi-supervised Learning / Active Learning

Kinds of Problems in Machine Learning

ML@B

- Supervised Learning
- Unsupervised Learning
- Semi-supervised Learning / Active Learning

Kinds of Problems in Machine Learning

ML@B

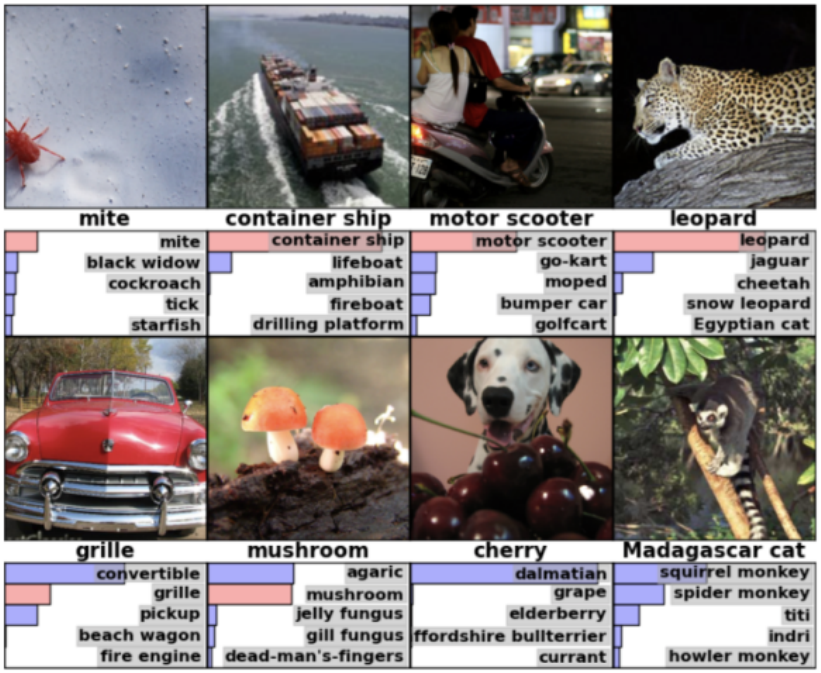
- Supervised Learning
- Unsupervised Learning
- Semi-supervised Learning / Active Learning

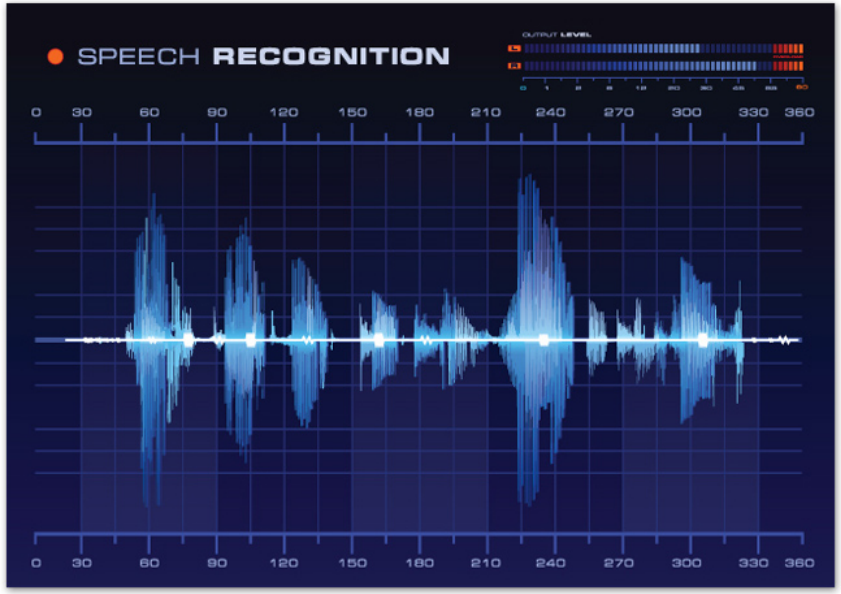
Kinds of Problems in Machine Learning

ML@B

- Supervised Learning
- Unsupervised Learning
- Semi-supervised Learning / Active Learning

Examples of Supervised Learning

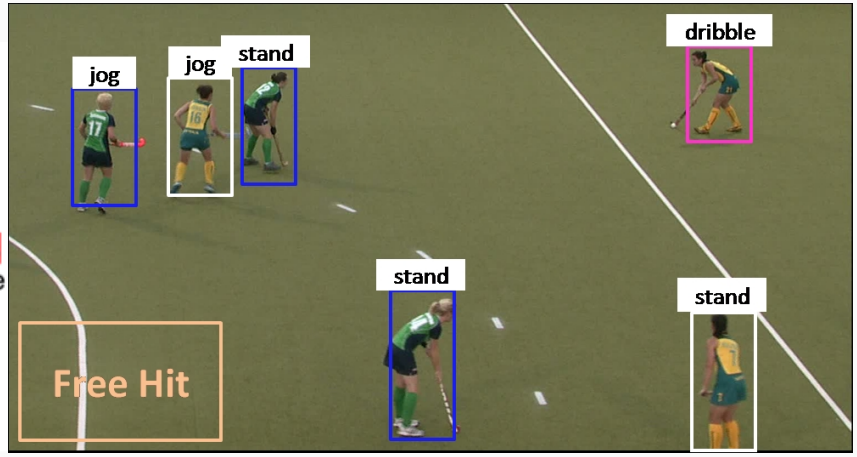




Prod:
The hotel is really beautiful. Moviestar feeling and decadence from yesterday. The pool is designed by Johnny Weissmuller. So it was a trendy pool. The food at the restaurant was really good. Very nice and helpful service at the frontfesk.

Cons: this is what made my grade a 3 instead of 4. We had problems to get the wi-fi working. If you're not depend this is not interesting. We talked several times with the front desk.

When we're there they had party event in the pool area between noon and 5 PM. The pool area was occupied with young party animals. So the area wasn't fun for UD.



ML@B

Goal

Given some data, X, and labels for that data y, we wish to find a map $f : X \mapsto y$. Such that given new unlabelled data, our function can predict its labels.

Deep Feedforward Networks

Biological Motivations

- Human brain is composed of neurons.
- Each neuron receives dopaminergic signals.
- Dopaminergic signals act as reward signal, much like gradient update.

Biological Motivations

- Human brain is composed of neurons.
- Each neuron receives dopaminergic signals.
- Dopaminergic signals act as reward signal, much like gradient update.

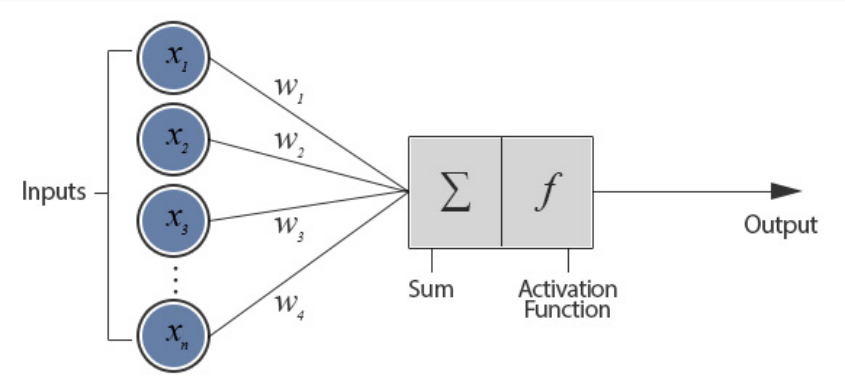
Biological Motivations

ML@B

- Human brain is composed of neurons.
- Each neuron receives dopaminergic signals.
- Dopaminergic signals act as reward signal, much like gradient update.

What does an artificial neuron look like?

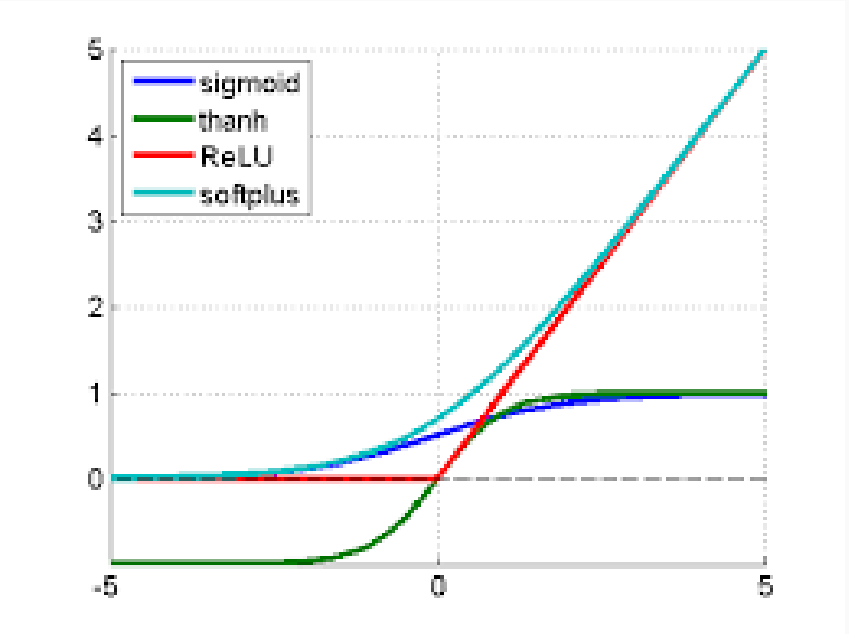
- Takes in input from each neuron in the previous layer.
- Outputs to each neuron in the next layer.
- Maintains a set of weights for each input.



$$y = g(\sum_i w_i y_i)$$

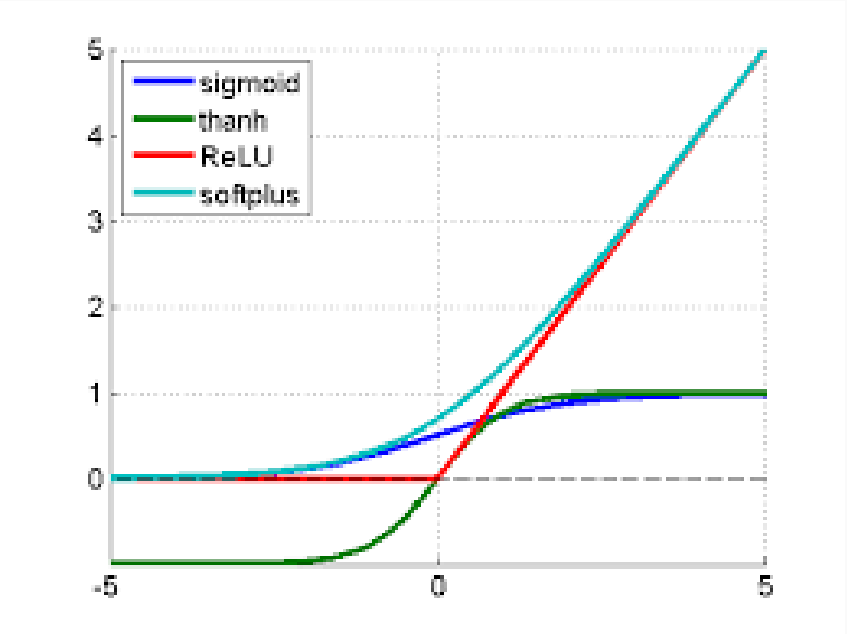
Activation Function

- Adds nonlinearity to networks.
- Various kinds of activation functions are shown on the right.



Activation Function

- Adds nonlinearity to networks.
- Various kinds of activation functions are shown on the right.



Universal Approximation Theorem

Theorem

Any function can be arbitrarily approximated by a feedforward network with only one hidden layer.

Universal Approximation Theorem

Theorem

Any function can be arbitrarily approximated by a feedforward network with only one hidden layer.

So why do we use Deep Neural Networks?

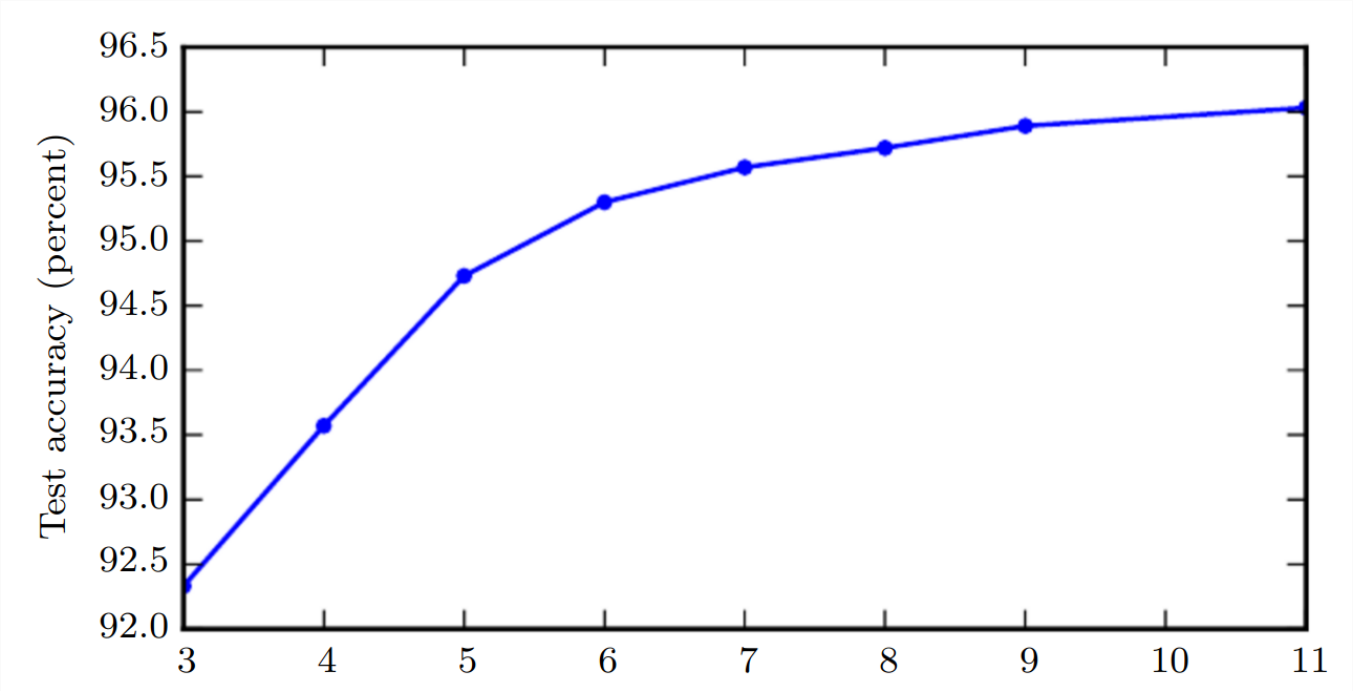
Universal Approximation Theorem

ML@B

Theorem

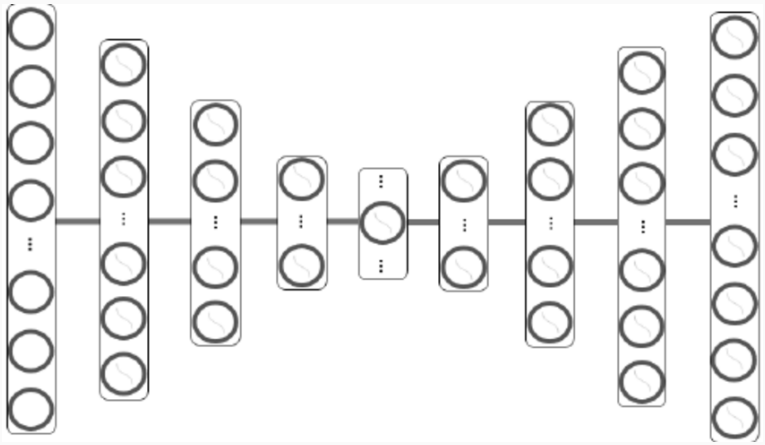
Any function can be arbitrarily approximated by a feedforward network with only one hidden layer.

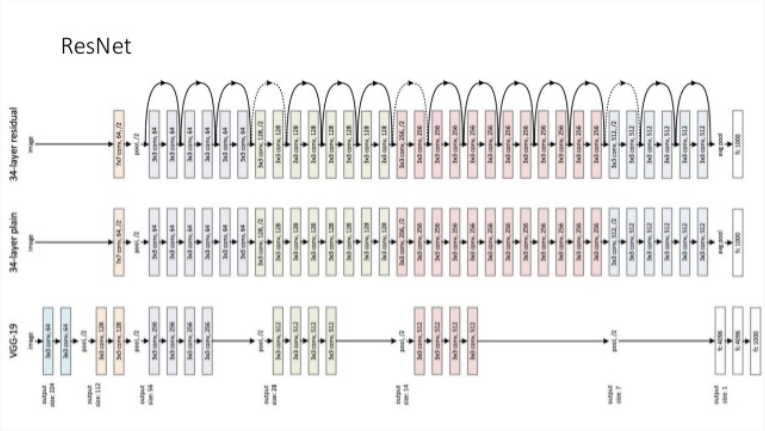
So why do we use Deep Neural Networks?



Deep Net Architectures

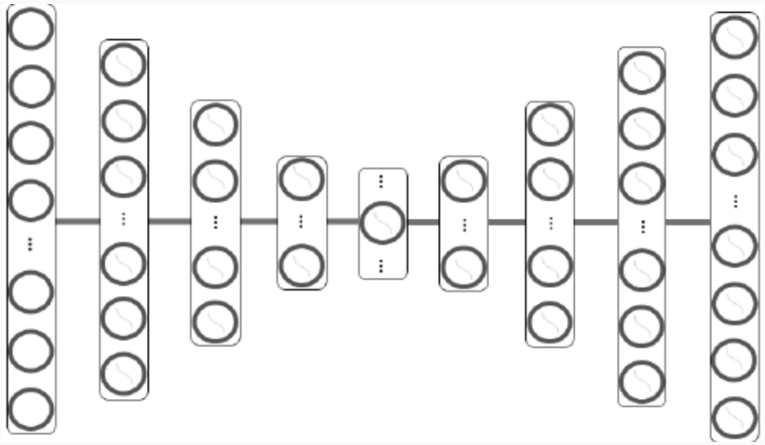
- When do we want to use a deep network?
- A wide network?
- A narrow network?

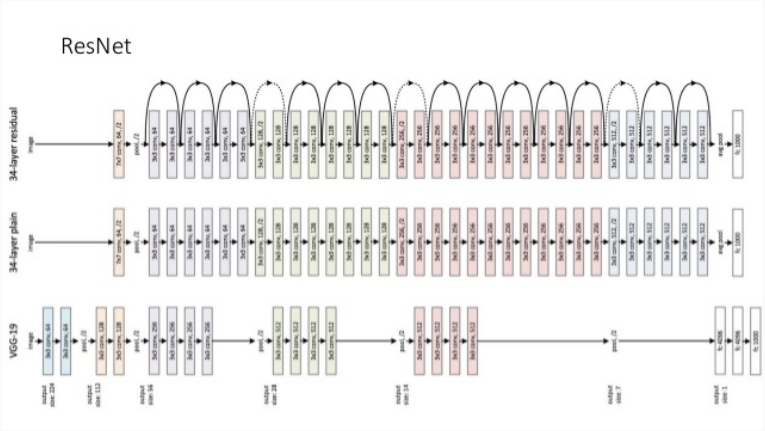




Deep Net Architectures

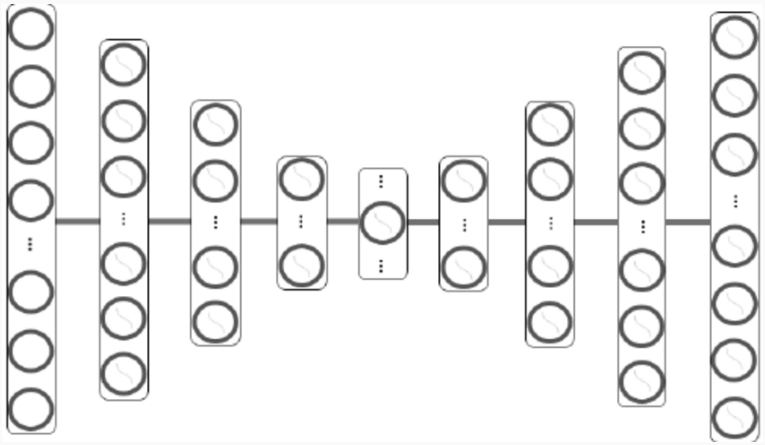
- When do we want to use a deep network?
- A wide network?
- A narrow network?

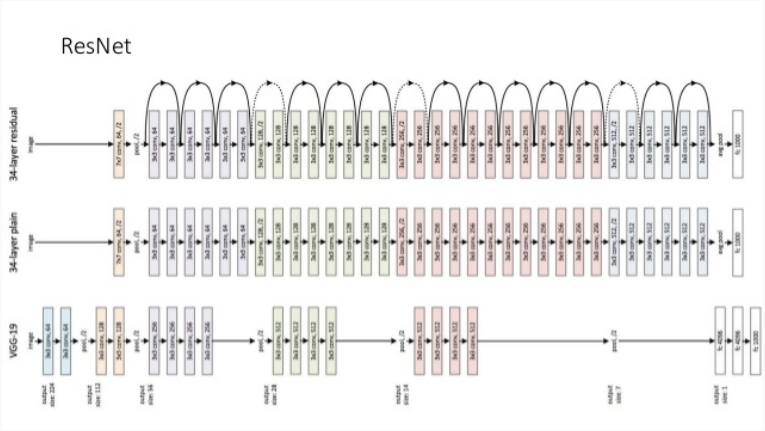




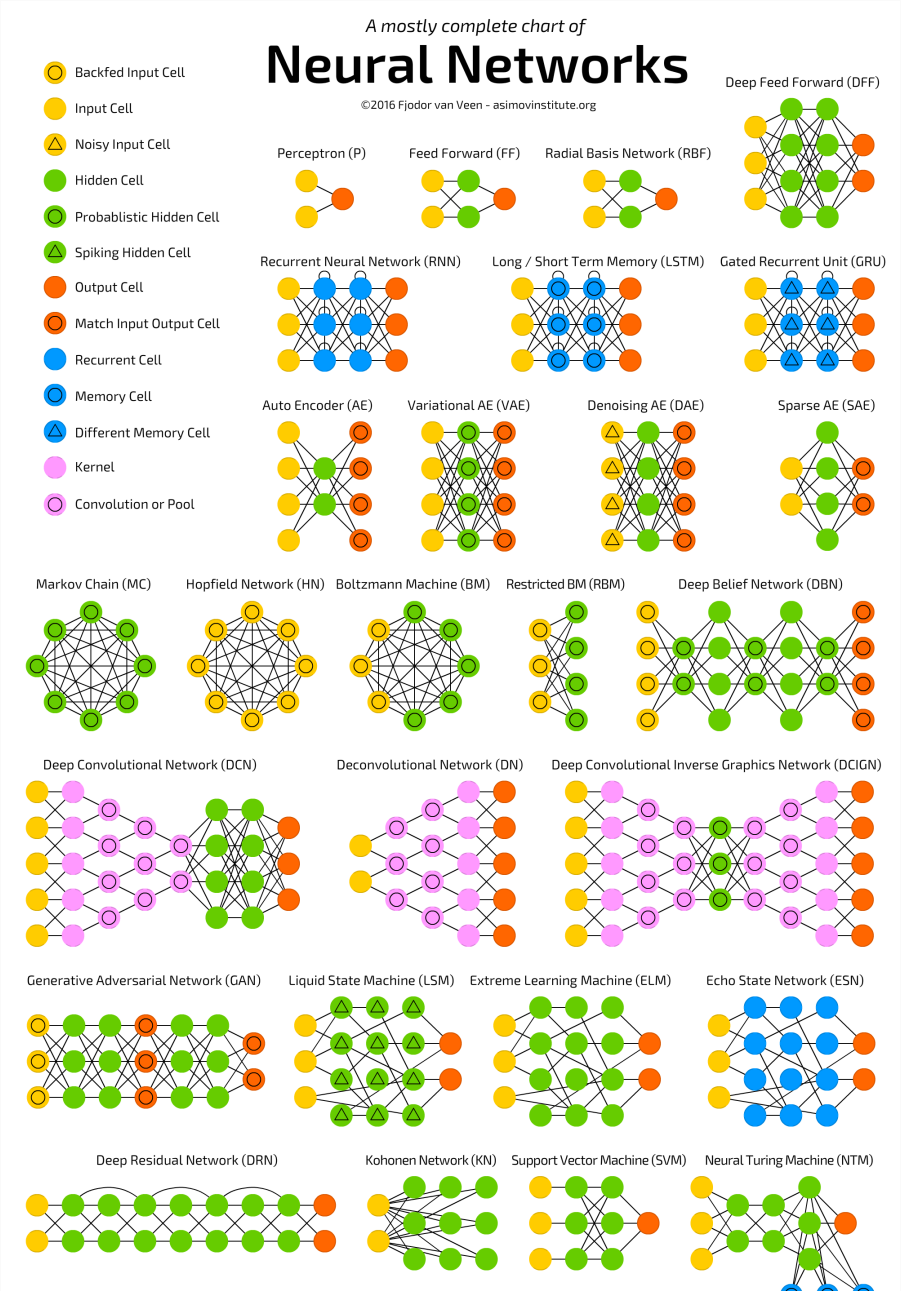
Deep Net Architectures

- When do we want to use a deep network?
- A wide network?
- A narrow network?





Deep Net Architectures



Optimization

How do we teach a deep net?

1. Define an objective.

How do we teach a deep net?

1. Define an objective.
2. Quantify the objective.

How do we teach a deep net?

1. Define an objective.
2. Quantify the objective.
3. Optimize the network weights.

How do we teach a deep net?

ML@B

1. Define an objective. ✓
2. Quantify the objective.
3. Optimize the network weights.

How do we teach a deep net?

ML@B

1. Define an objective. ✓
2. Quantify the objective. ✗
3. Optimize the network weights. ✗

Loss Functions

- Loss function J should encapsulate qualitative objective.
- Has to be differentiable.
- Has to be computationally tractable.

Examples:

- **Mean squared error**
- **Cross entropy**
- Zero-one loss
- Hinge loss
- Logistic loss

Loss Functions

- Loss function J should encapsulate qualitative objective.
- Has to be differentiable.
- Has to be computationally tractable.

Examples:

- **Mean squared error**
- **Cross entropy**
- Zero-one loss
- Hinge loss
- Logistic loss

Loss Functions

- Loss function J should encapsulate qualitative objective.
- Has to be differentiable.
- Has to be computationally tractable.

Examples:

- **Mean squared error**
- **Cross entropy**
- Zero-one loss
- Hinge loss
- Logistic loss

Loss Functions

ML@B

- Loss function J should encapsulate qualitative objective.
- Has to be differentiable.
- Has to be computationally tractable.

Examples:

- **Mean squared error**
- **Cross entropy**
- Zero-one loss
- Hinge loss
- Logistic loss

Loss Functions

- Loss function J should encapsulate qualitative objective.
- Has to be differentiable.
- Has to be computationally tractable.

Examples:

- **Mean squared error**
- **Cross entropy**
- Zero-one loss
- Hinge loss
- Logistic loss

Loss Functions

- Loss function J should encapsulate qualitative objective.
- Has to be differentiable.
- Has to be computationally tractable.

Examples:

- **Mean squared error**
- **Cross entropy**
- Zero-one loss
- Hinge loss
- Logistic loss

Loss Functions

- Loss function J should encapsulate qualitative objective.
- Has to be differentiable.
- Has to be computationally tractable.

Examples:

- **Mean squared error**
- **Cross entropy**
- Zero-one loss
- Hinge loss
- Logistic loss

Loss Functions

ML@B

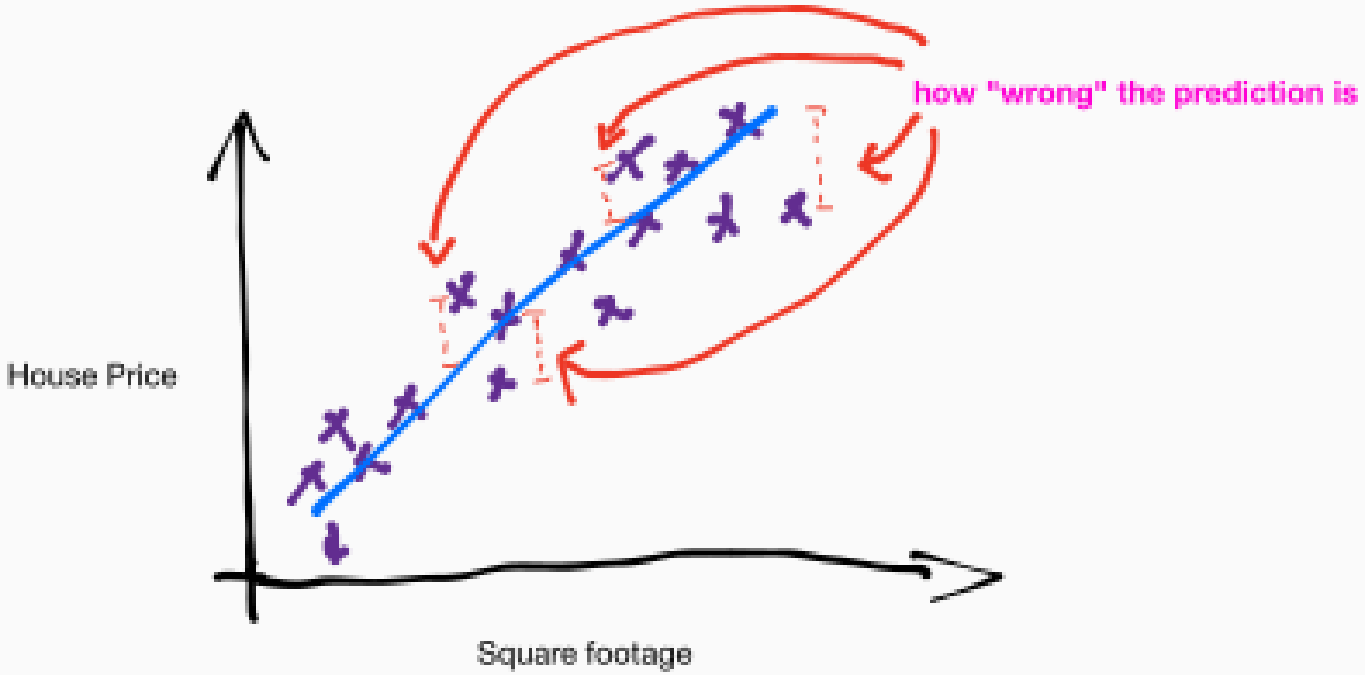
- Loss function J should encapsulate qualitative objective.
- Has to be differentiable.
- Has to be computationally tractable.

Examples:

- **Mean squared error**
- **Cross entropy**
- Zero-one loss
- Hinge loss
- Logistic loss

Mean Squared Error

$$\frac{1}{N}\sum_{i}^{N}(\hat{y}-y)^2$$



Cross Entropy Loss

$$-\frac{1}{N}\sum_{i}^{N}\sum_{j}^{m}(y_{ij}\log(\hat{y}_{ij}))$$

- Used for categorical classification tasks
- Negative log-likelihood of the predicted probability distribution under the real probability distribution
- Equivalent to minimizing KL-divergence between the predicted probability distribution and the real probability distribution.

Cross Entropy Loss

$$-\frac{1}{N}\sum_{i}^{N}\sum_{j}^{m}(y_{ij}\log(\hat{y}_{ij}))$$

- Used for categorical classification tasks
- Negative log-likelihood of the predicted probability distribution under the real probability distribution
- Equivalent to minimizing KL-divergence between the predicted probability distribution and the real probability distribution.

Cross Entropy Loss

$$-\frac{1}{N}\sum_{i}^{N}\sum_{j}^{m}(y_{ij}\log(\hat{y}_{ij}))$$

- Used for categorical classification tasks
- Negative log-likelihood of the predicted probability distribution under the real probability distribution
- Equivalent to minimizing KL-divergence between the predicted probability distribution and the real probability distribution.

How do we teach a deep net?

ML@B

1. Define an objective. ✓
2. Quantify the objective.
3. Optimize the network weights.

How do we teach a deep net?

ML@B

1. Define an objective. ✓
2. Quantify the objective. ✓
3. Optimize the network weights.

How do we teach a deep net?

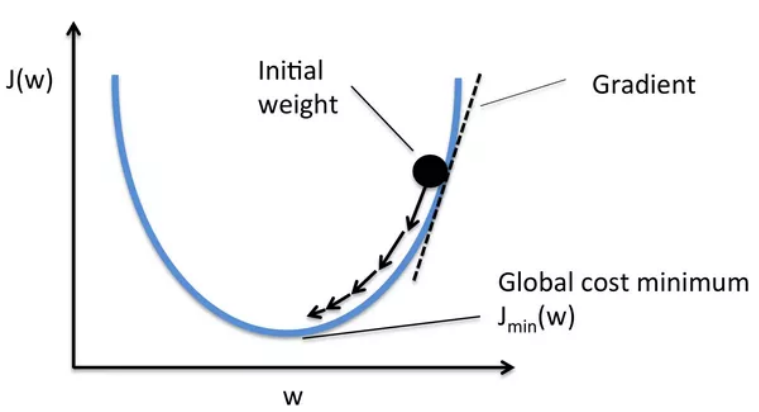
ML@B

1. Define an objective. ✓
2. Quantify the objective. ✓
3. Optimize the network weights. ✗

Gradient Descent

- We have a loss function, J.
- Wish to minimize J w.r.t. θ.

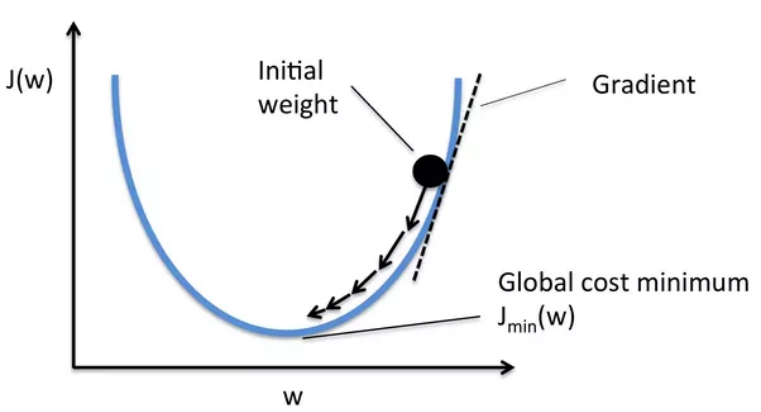
$$\theta := \theta - \eta \nabla_\theta J(\theta)$$



Gradient Descent

- We have a loss function, J.
- Wish to minimize J w.r.t. θ.

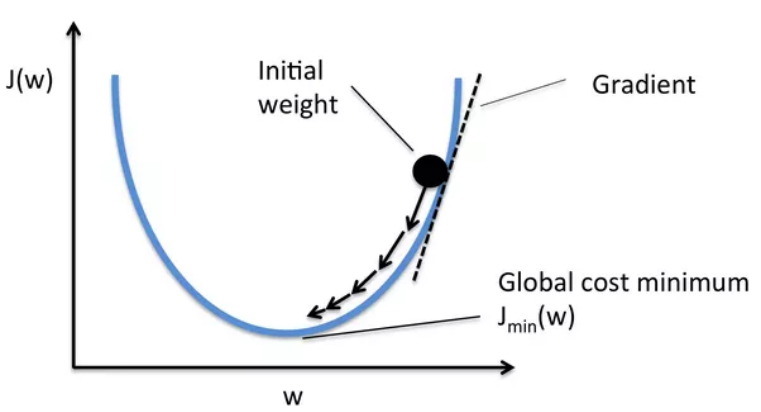
$$\theta := \theta - \eta \nabla_\theta J(\theta)$$



Gradient Descent

- We have a loss function, J.
- Wish to minimize J w.r.t. θ.

$$\theta := \theta - \eta \nabla_\theta J(\theta)$$

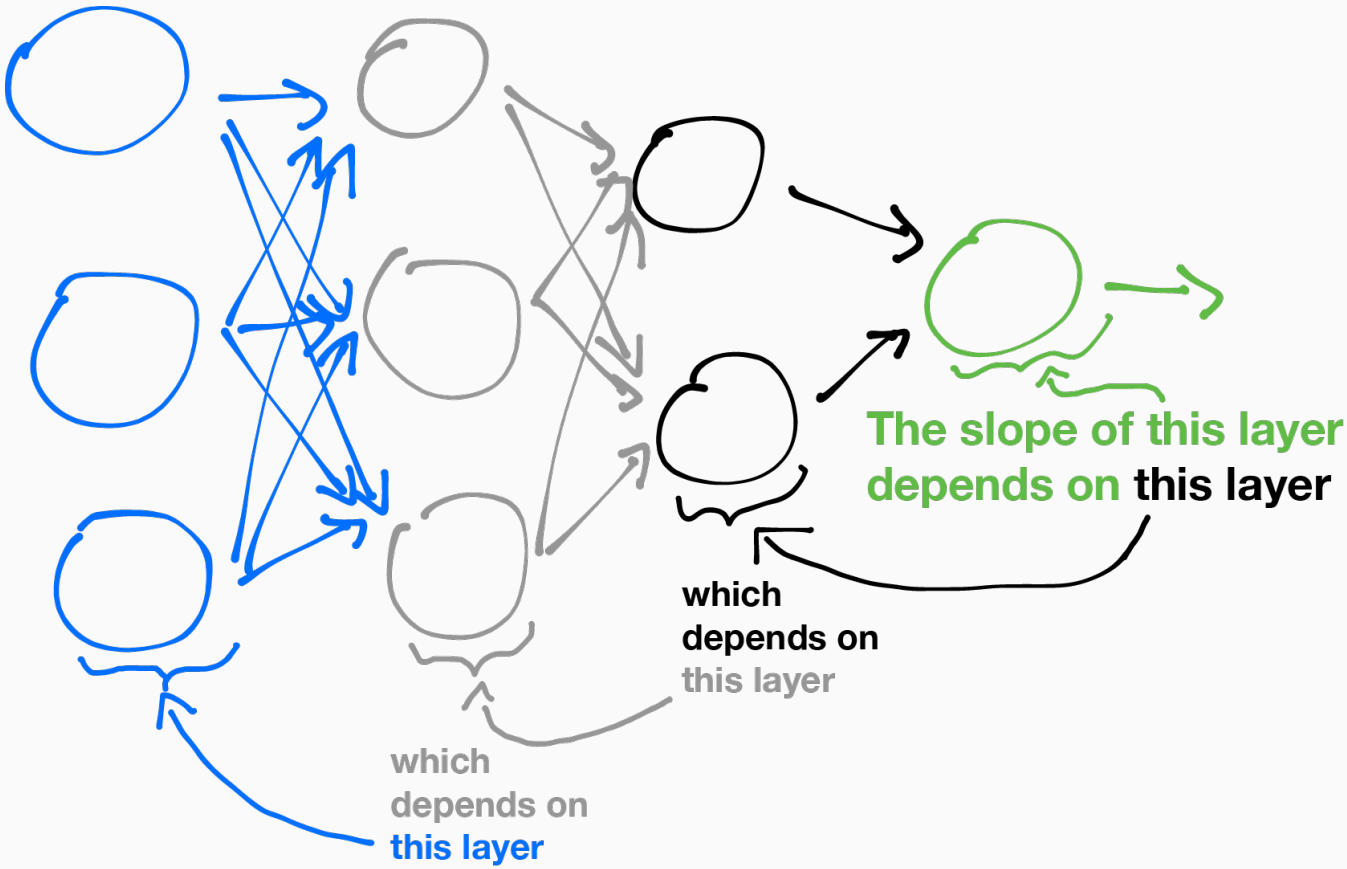


Stochastic Gradient Descent

ML@B

- Key difference: update based on random sample of training set instead of whole training set.
- Will go more in depth in future lecture.

Backpropagation



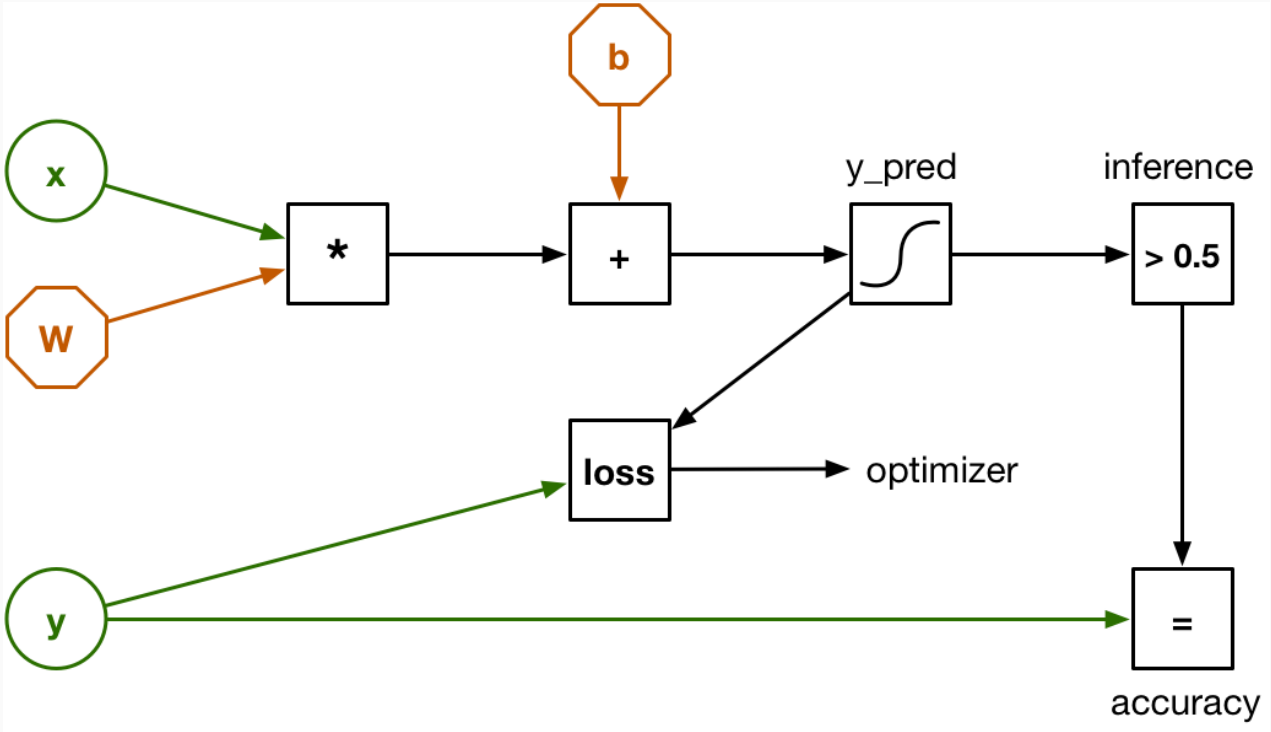
https://ml.berkeley.edu/blog/2017/02/04/tutorial-3/

Generalized Backprop: Differentiation of Computational Graphs

Goal: We wish to calculate the derivative of an output (or outputs) of a computational graph, with respect to another part of the graph.

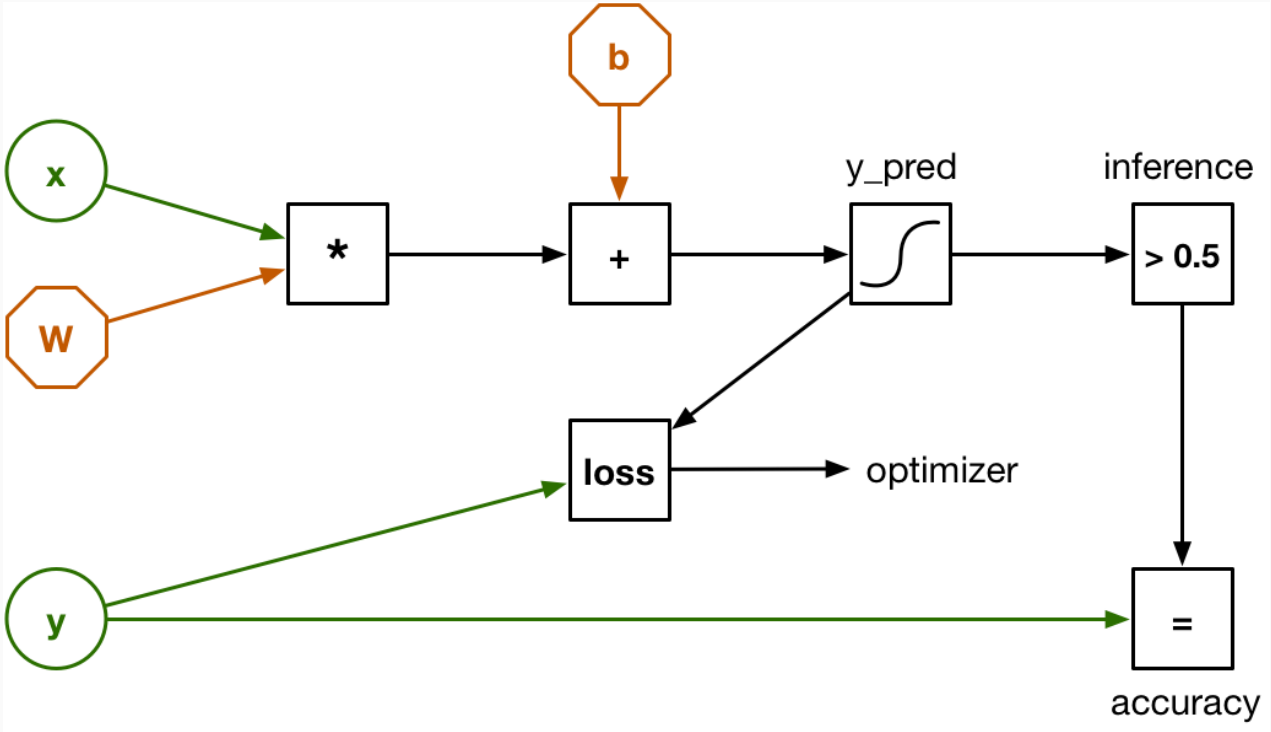
What is a computational graph?

- Each node is an operation
- Each edge indicates flow of data through the graph.



What is a computational graph?

- Each node is an operation
- Each edge indicates flow of data through the graph.



Methods for computing derivatives

ML@B

- Numerical methods
- Symbolic Differentiation
- Autodifferentiation

Methods for computing derivatives

ML@B

- Numerical methods
- Symbolic Differentiation
- Autodifferentiation

Methods for computing derivatives

ML@B

- Numerical methods
- Symbolic Differentiation
- Autodifferentiation

Autodifferentiation

- Exact method for computing derivatives
- Instead of using symbols and rules like symbolic differentiation, we define a new arithmetic: Dual Numbers.

Autodifferentiation

ML@B

- Exact method for computing derivatives
- Instead of using symbols and rules like symbolic differentiation, we define a new arithmetic: Dual Numbers.

Dual Numbers

Extend real numbers by adding a component, much like how Complex numbers have an imaginary component, you can think of dual numbers as having an additional "derivative" component.

$$\mathbf{x} = (x, \dot{x})$$

Now we have to define a new arithmetic on these new kinds of numbers.

Dual Number arithmetic

Addition:

$$(x, \dot{x}) + (y, \dot{y}) = (x + y, \dot{x} + \dot{y})$$

Multiplication:

$$(x, \dot{x}) \times (y, \dot{y}) = (xy, x\dot{y} + \dot{x}y)$$

Division:

$$\frac{1}{(x, \dot{x})} = (\frac{1}{x}, -\frac{\dot{x}}{x^2})$$

Autodiff

How do we use this arithmetic to compute arbitrary derivatives?

Example: Suppose we wish to calculate the derivative of $f(x, y) = x^2 y$ w.r.t x.

$$
\begin{aligned}
\mathbf{x} &= (x, 1) \\
\mathbf{y} &= (y, 0) \\
\mathbf{x}^2 &= (x, 1) \times (x, 1) = (x^2, x * 1 + x * 1) \\
\mathbf{x}^2 \times \mathbf{y} &= (x^2, 2x) \times (y, 0) = (x^2 y, x^2 * 0 + 2x * y) \\
&= (x^2 y, 2xy)
\end{aligned}
$$

How do we teach a deep net?

ML@B

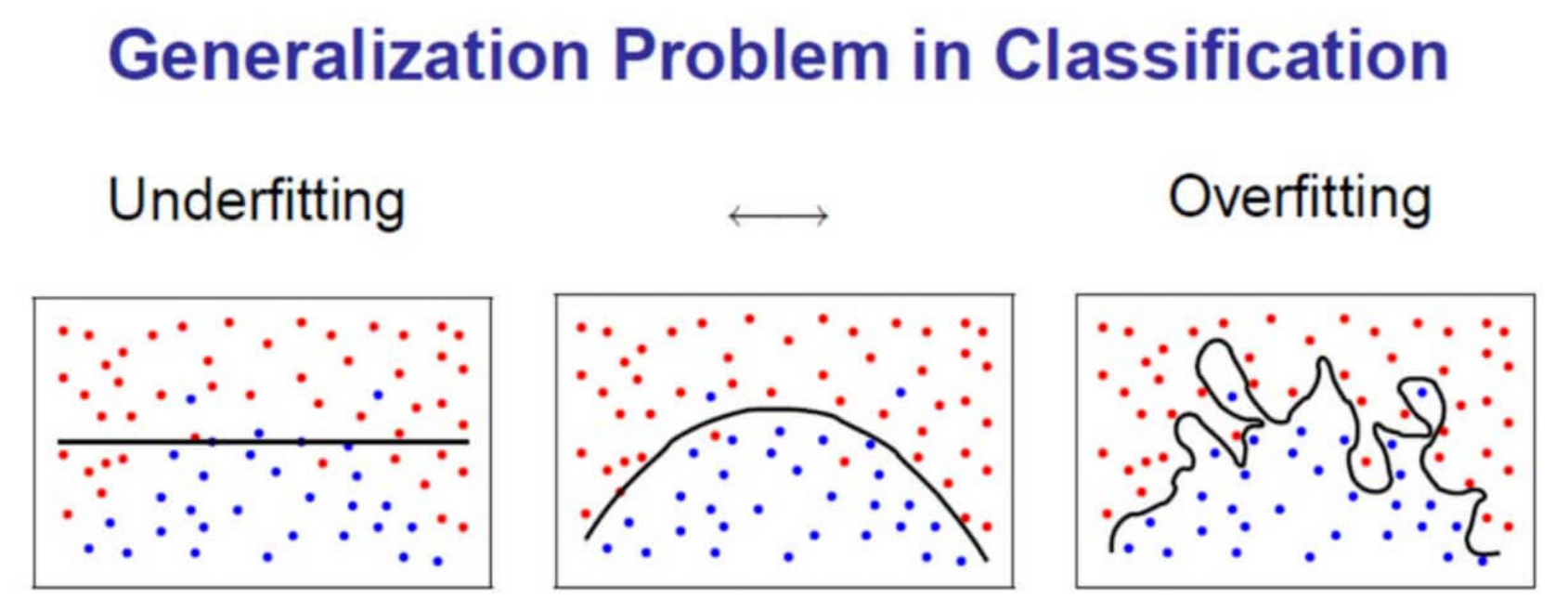
1. Define an objective. ✓
2. Quantify the objective. ✓
3. Optimize the network weights.

How do we teach a deep net?

ML@B

1. Define an objective. ✓
2. Quantify the objective. ✓
3. Optimize the network weights. ✓

Problem: Overfitting



Regularization

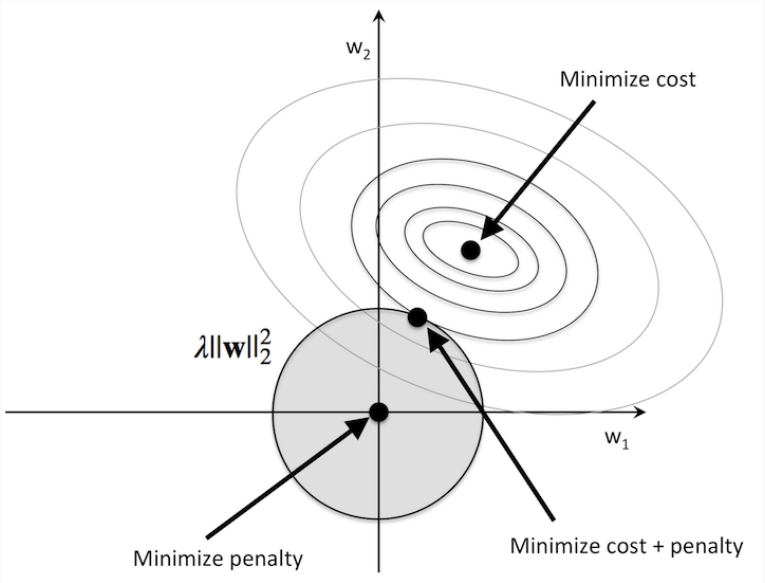
How to we regularize Neural Nets?

ML@B

- Parameter penalties (L_1 and L_2 penalties)
- Dataset augmentation
- Ensemble methods
- Dropout
- Adversarial Training

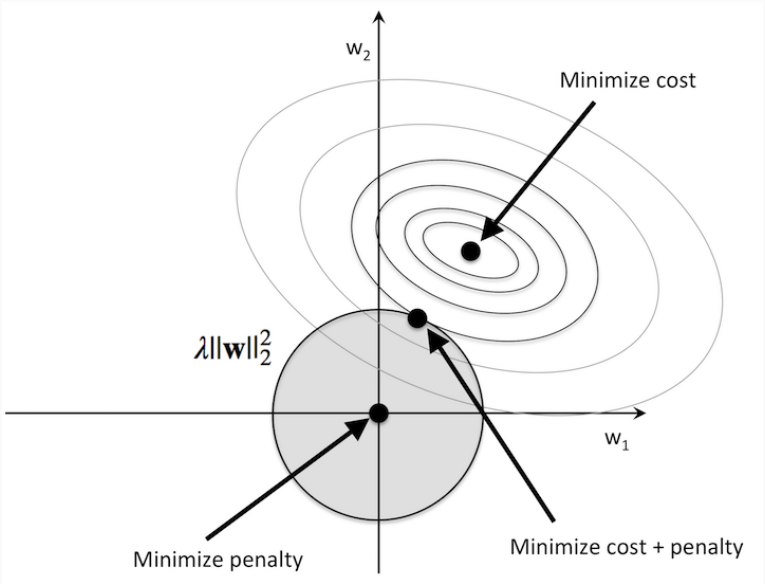
L_2 weight penalty

- Add L_2 penalty term
- $\lambda||\boldsymbol{w}||^2$
- Adds constraint to optimization problem.



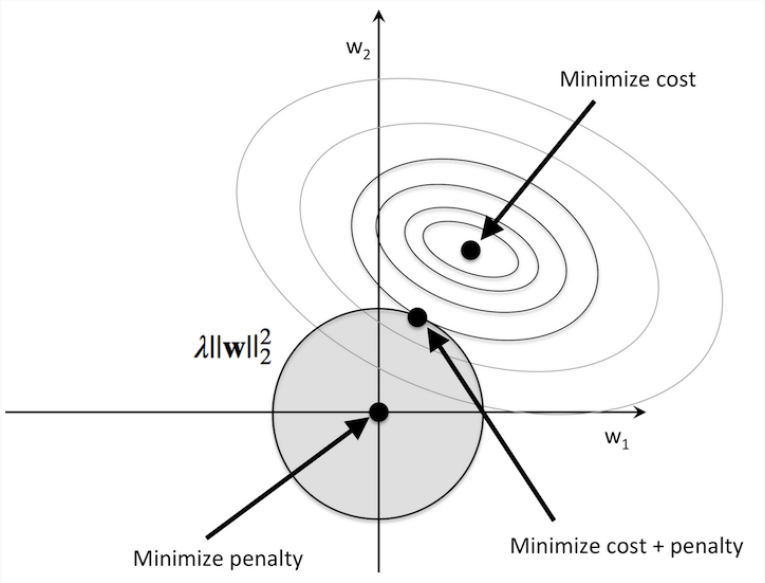
L_2 weight penalty

- Add L_2 penalty term
- $\lambda||\boldsymbol{w}||^2$
- Adds constraint to optimization problem.



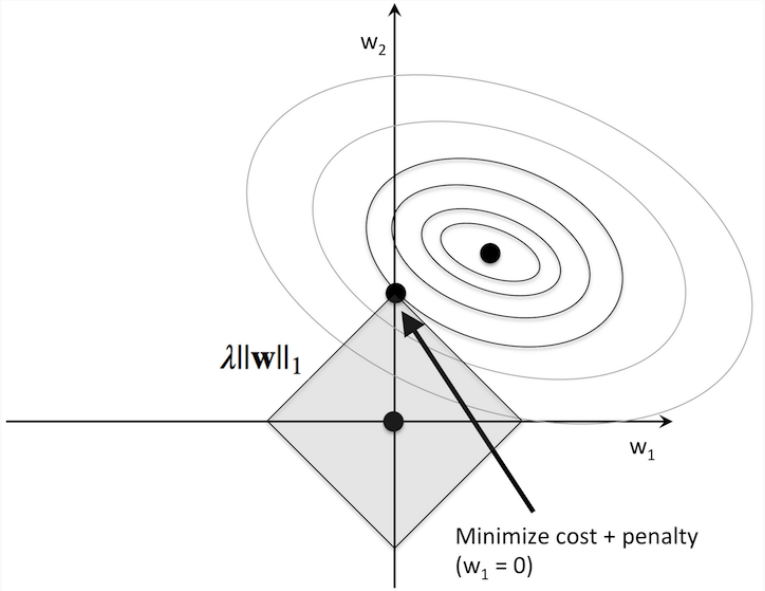
L_2 weight penalty

- Add L_2 penalty term
- $\lambda||\boldsymbol{w}||^2$
- Adds constraint to optimization problem.



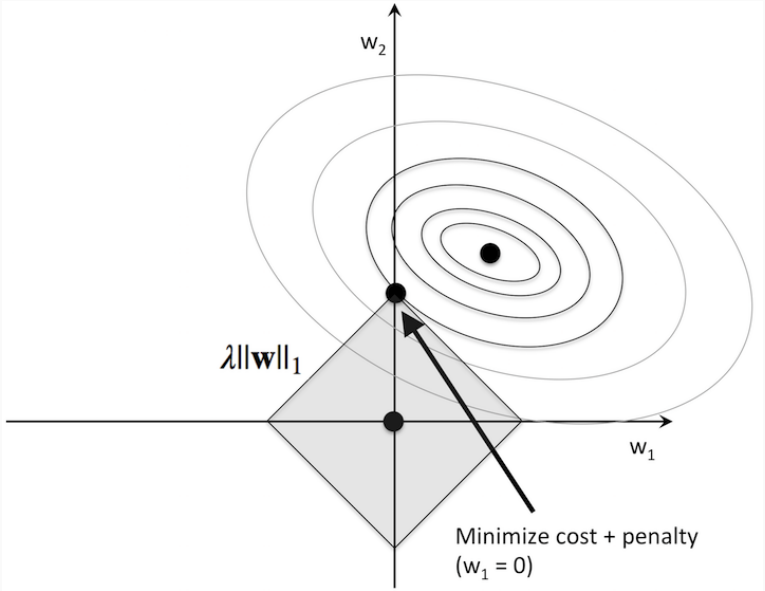
L_1 weight penalty

- Add L_1 penalty term
- $\lambda||\boldsymbol{w}||$
- Creates sparsity in representation
- Acts as feature selector



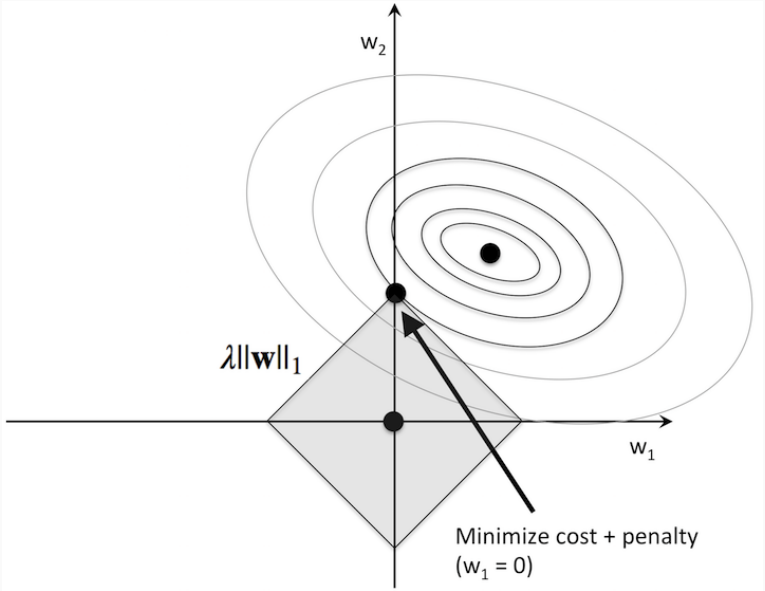
L_1 weight penalty

- Add L_1 penalty term
- $\lambda||\boldsymbol{w}||$
- Creates sparsity in representation
- Acts as feature selector



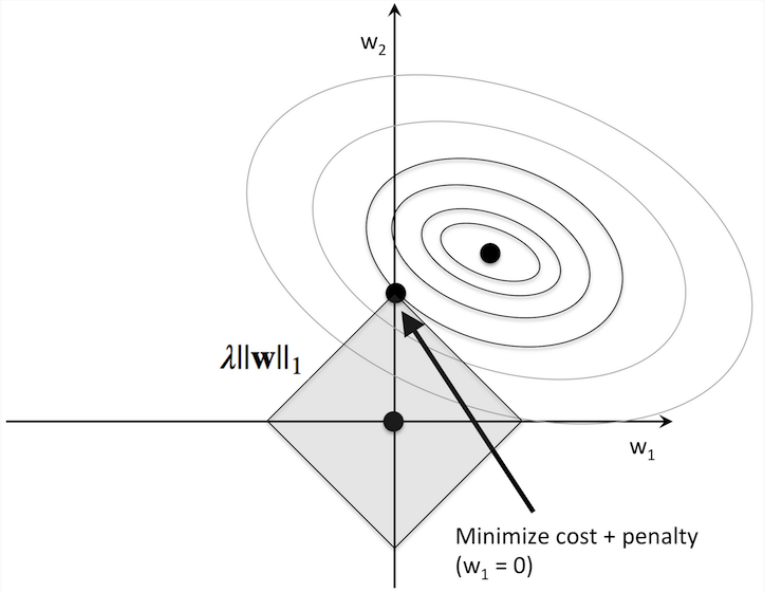
L_1 weight penalty

- Add L_1 penalty term
- $\lambda||\boldsymbol{w}||$
- Creates sparsity in representation
- Acts as feature selector



L_1 weight penalty

- Add L_1 penalty term
- $\lambda||\boldsymbol{w}||$
- Creates sparsity in representation
- Acts as feature selector



Dataset Augmentation

ML@B

- Training on more data leads to better generalization
- Can create new fake data to make our dataset larger
- Examples
  - Translation/Rotation of images for image classification
  - Cropping images
  - Adding random noise to data

Dataset Augmentation

ML@B

- Training on more data leads to better generalization
- Can create new fake data to make our dataset larger
- Examples
  - Translation/Rotation of images for image classification
  - Cropping images
  - Adding random noise to data

Dataset Augmentation

ML@B

- Training on more data leads to better generalization
- Can create new fake data to make our dataset larger
- Examples
  - Translation/Rotation of images for image classification
  - Cropping images
  - Adding random noise to data

Dataset Augmentation

ML@B

- Training on more data leads to better generalization
- Can create new fake data to make our dataset larger
- Examples
  - Translation/Rotation of images for image classification
  - Cropping images
  - Adding random noise to data

Dataset Augmentation

ML@B

- Training on more data leads to better generalization
- Can create new fake data to make our dataset larger
- Examples
  - Translation/Rotation of images for image classification
  - Cropping images
  - Adding random noise to data

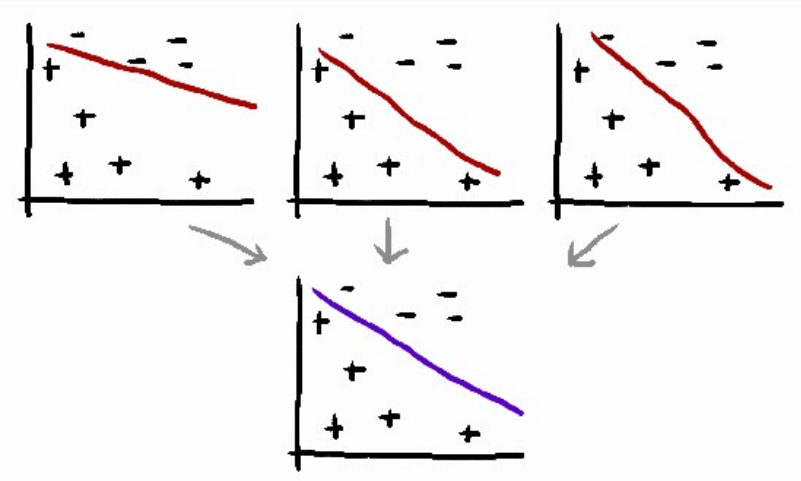
Dataset Augmentation

ML@B

- Training on more data leads to better generalization
- Can create new fake data to make our dataset larger
- Examples
  - Translation/Rotation of images for image classification
  - Cropping images
  - Adding random noise to data

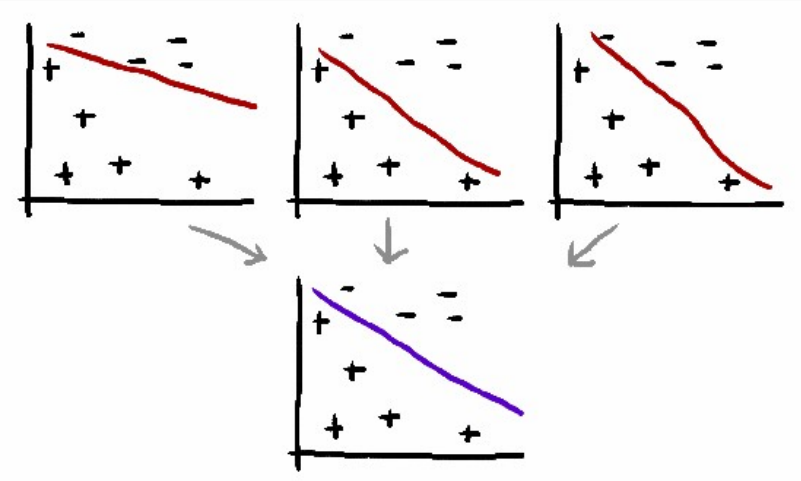
Ensemble Methods

- Combine different models together to improve generalization.
- Example ensemble methods
  - Bagging
  - Model Averaging
  - Dropout
  - Boosting



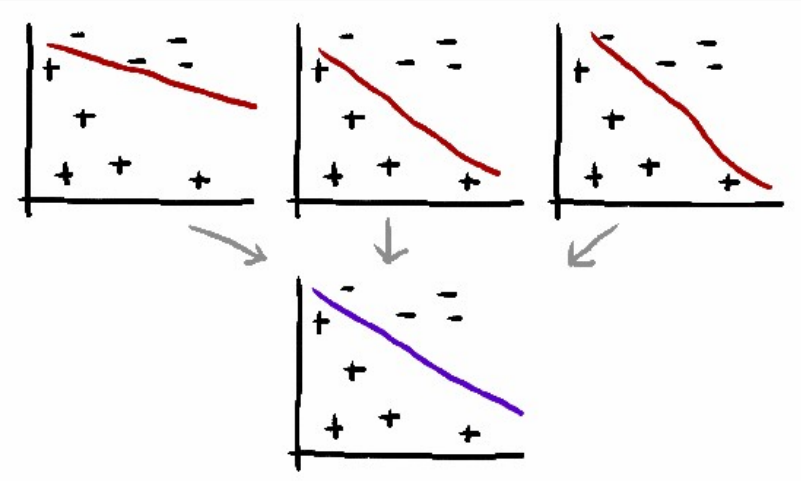
Ensemble Methods

- Combine different models together to improve generalization.
- Example ensemble methods
  - Bagging
  - Model Averaging
  - Dropout
  - Boosting



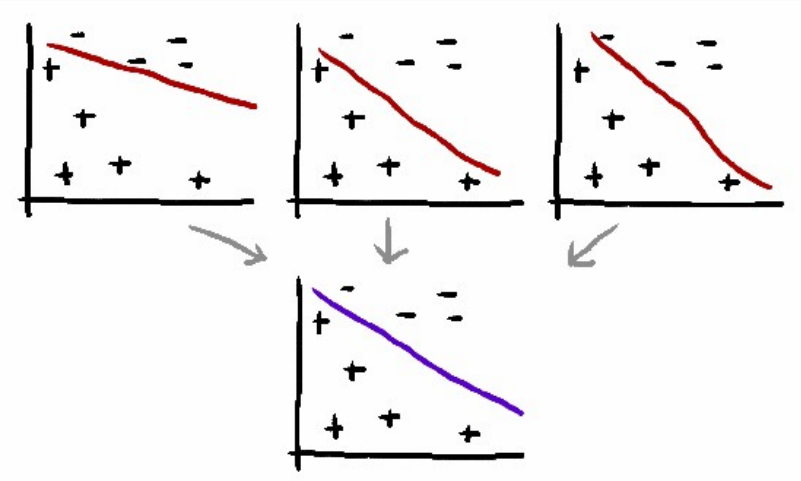
Ensemble Methods

- Combine different models together to improve generalization.
- Example ensemble methods
  - Bagging
  - Model Averaging
  - Dropout
  - Boosting



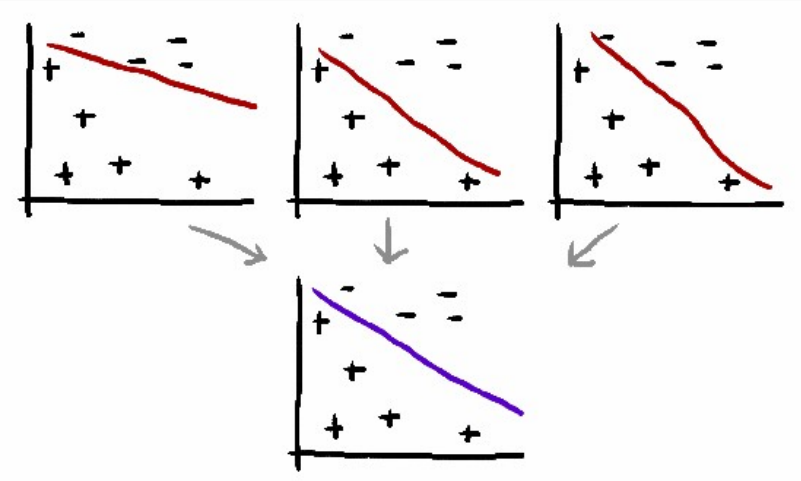
Ensemble Methods

- Combine different models together to improve generalization.
- Example ensemble methods
  - Bagging
  - Model Averaging
  - Dropout
  - Boosting



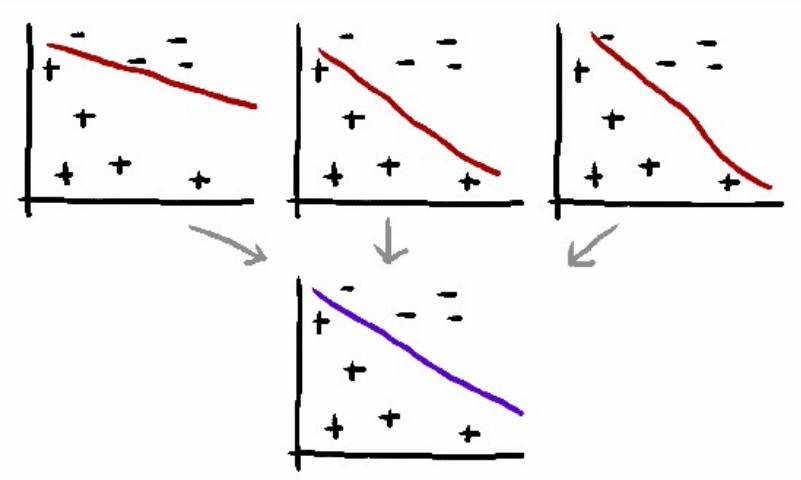
Ensemble Methods

- Combine different models together to improve generalization.
- Example ensemble methods
  - Bagging
  - Model Averaging
  - Dropout
  - Boosting



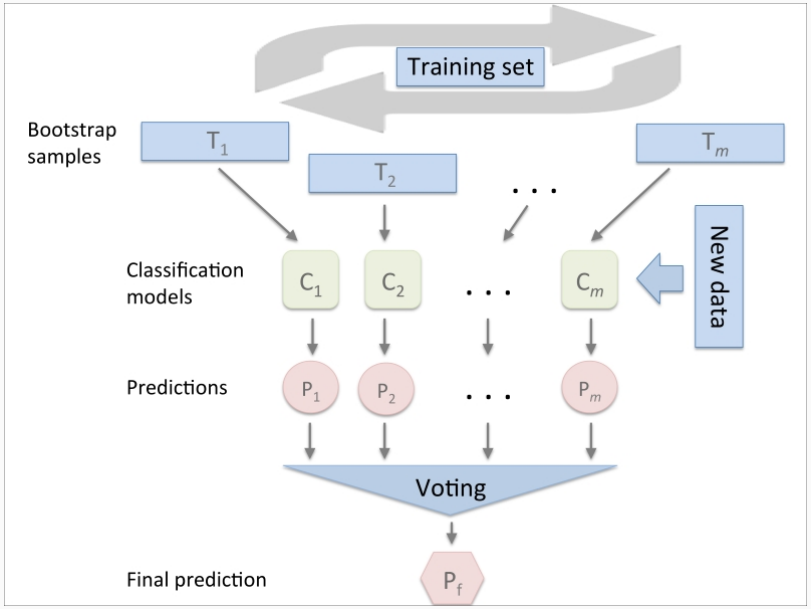
Ensemble Methods

- Combine different models together to improve generalization.
- Example ensemble methods
  - Bagging
  - Model Averaging
  - Dropout
  - Boosting



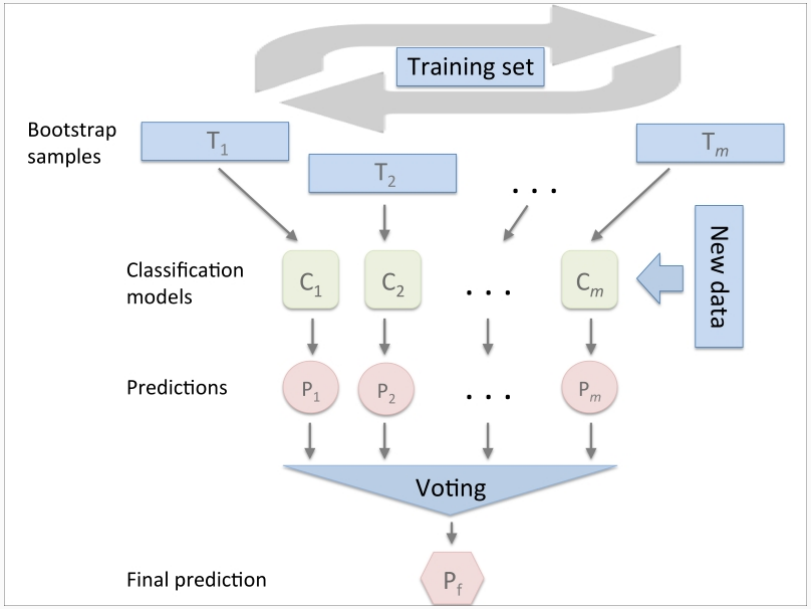
Bagging

- Create k different datasets from your original dataset by sampling with replacement.
- Train a model on each of the k datasets
- Take majority vote or average over the k different models to come up with decision.



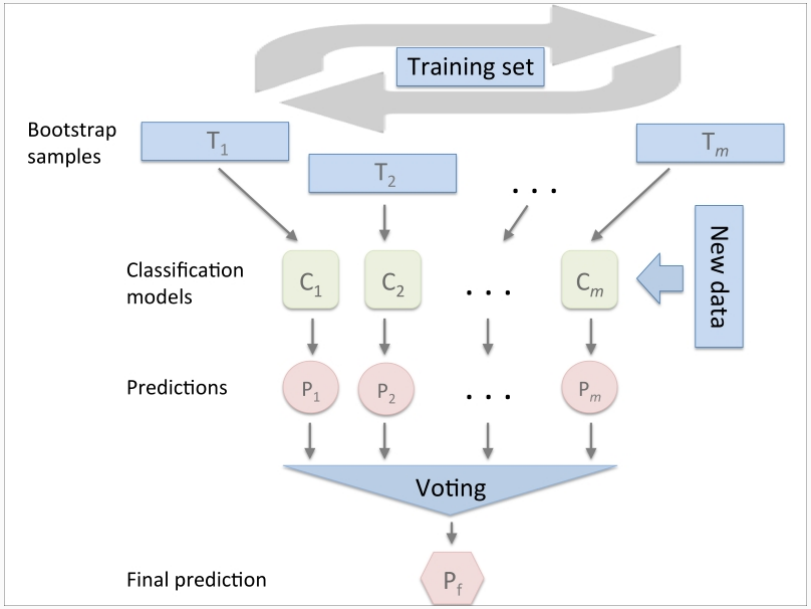
Bagging

- Create k different datasets from your original dataset by sampling with replacement.
- Train a model on each of the k datasets
- Take majority vote or average over the k different models to come up with decision.



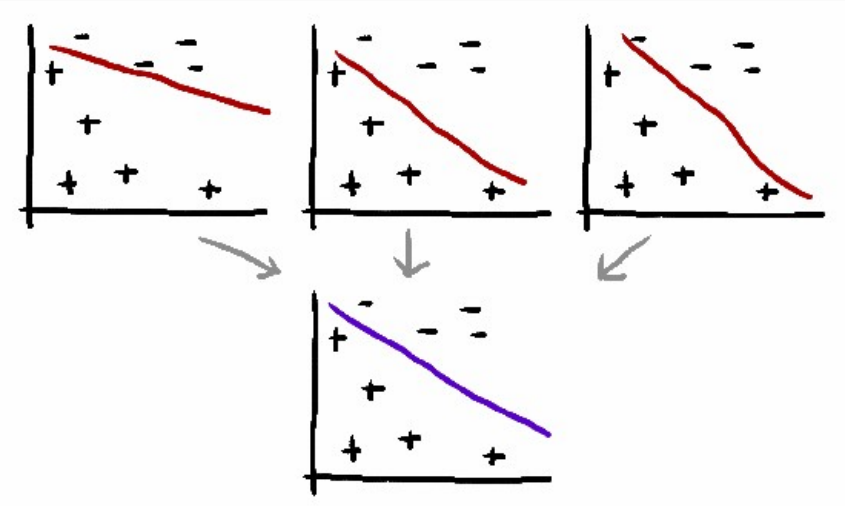
Bagging

- Create k different datasets from your original dataset by sampling with replacement.
- Train a model on each of the k datasets
- Take majority vote or average over the k different models to come up with decision.



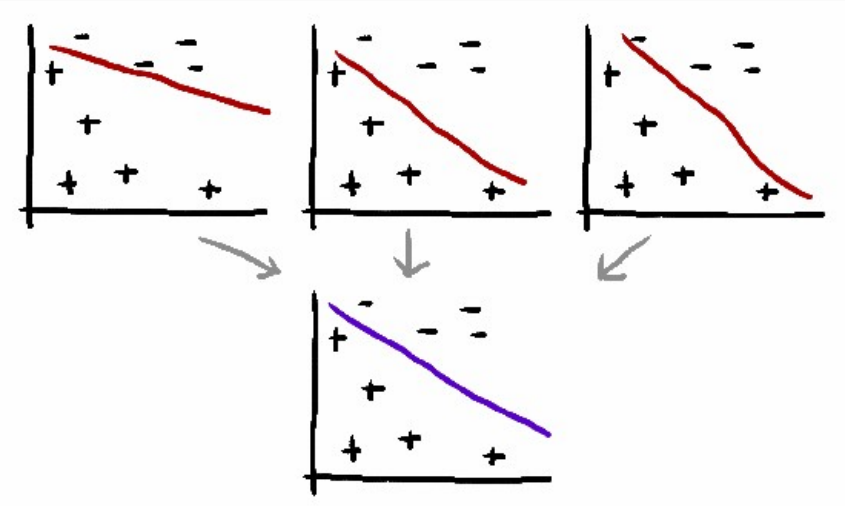
Model Averaging

- Take any combination of k models
  - Trained on different datasets OR
  - Have different architectures OR
  - Have different loss functions OR
  - ...
- Average across model outputs



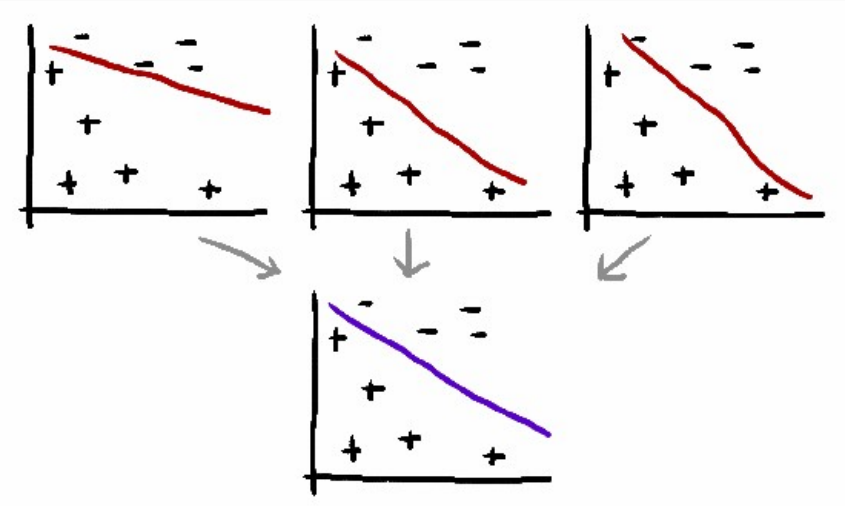
Model Averaging

- Take any combination of k models
  - Trained on different datasets OR
  - Have different architectures OR
  - Have different loss functions OR
  - ...
- Average across model outputs



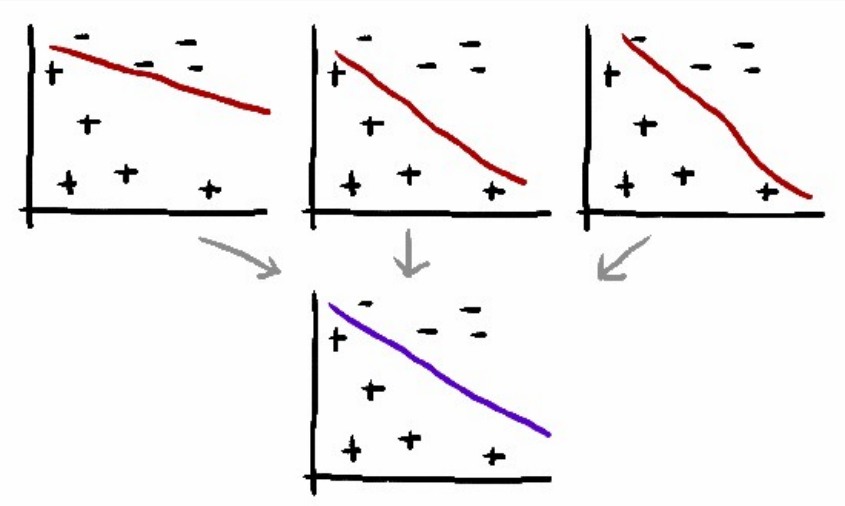
Model Averaging

- Take any combination of k models
  - Trained on different datasets OR
  - Have different architectures OR
  - Have different loss functions OR
  - ...
- Average across model outputs



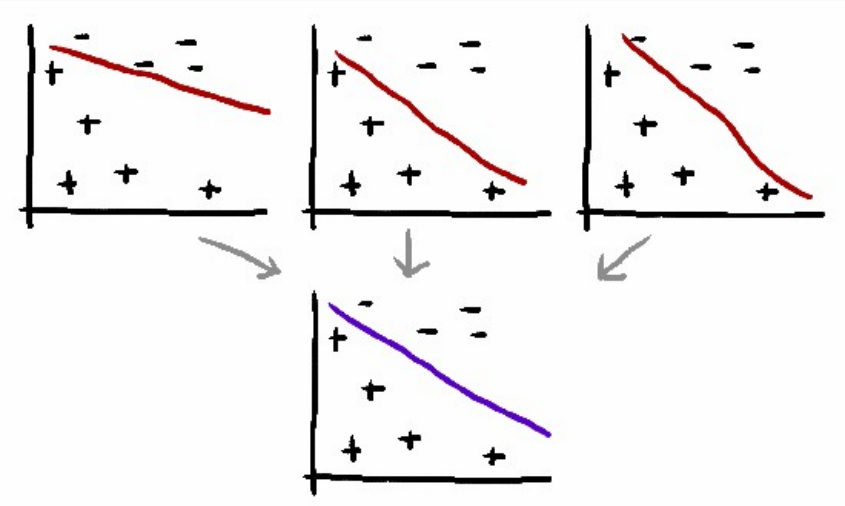
Model Averaging

- Take any combination of k models
  - Trained on different datasets OR
  - Have different architectures OR
  - Have different loss functions OR
  - ...
- Average across model outputs



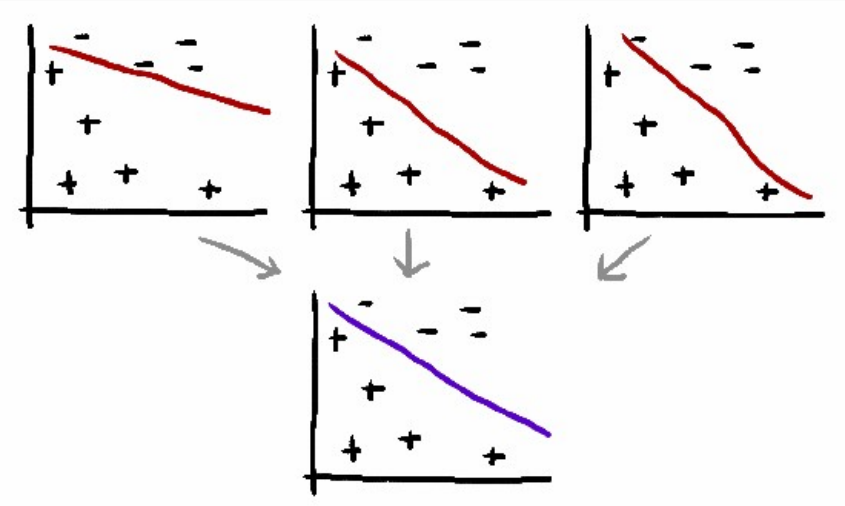
Model Averaging

- Take any combination of k models
  - Trained on different datasets OR
  - Have different architectures OR
  - Have different loss functions OR
  - ...
- Average across model outputs



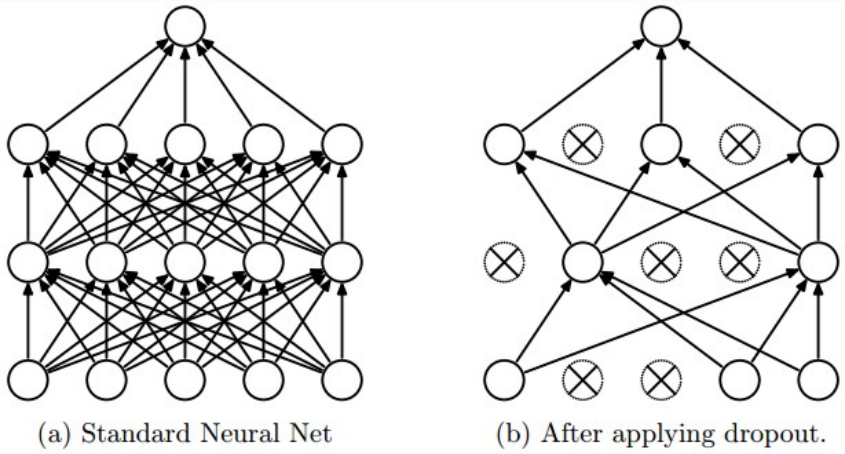
Model Averaging

- Take any combination of k models
  - Trained on different datasets OR
  - Have different architectures OR
  - Have different loss functions OR
  - ...
- Average across model outputs



Dropout

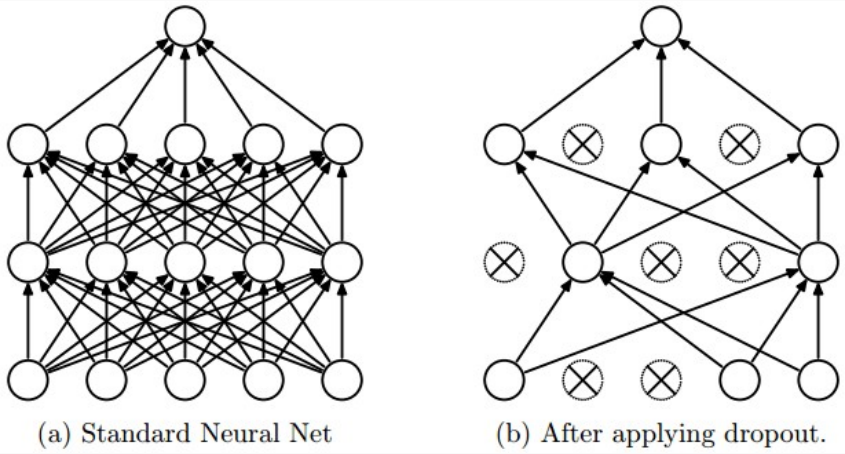
- Randomly "drop" neurons during training with some probability p.
- Keep all neurons during testing.
- Efficiently creates an ensemble of different network architectures.



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

Dropout

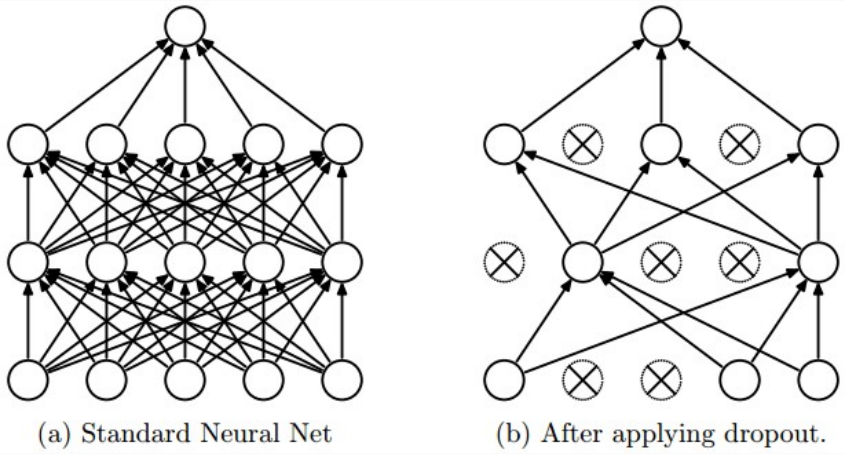
- Randomly "drop" neurons during training with some probability p.
- Keep all neurons during testing.
- Efficiently creates an ensemble of different network architectures.



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

Dropout

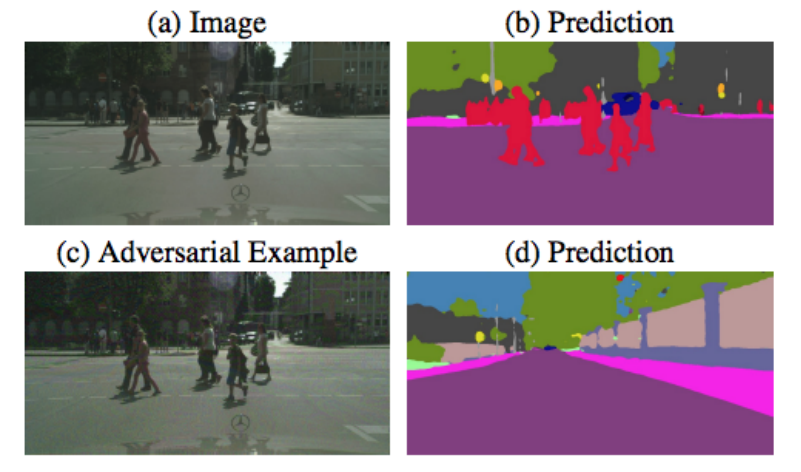
- Randomly "drop" neurons during training with some probability p.
- Keep all neurons during testing.
- Efficiently creates an ensemble of different network architectures.



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

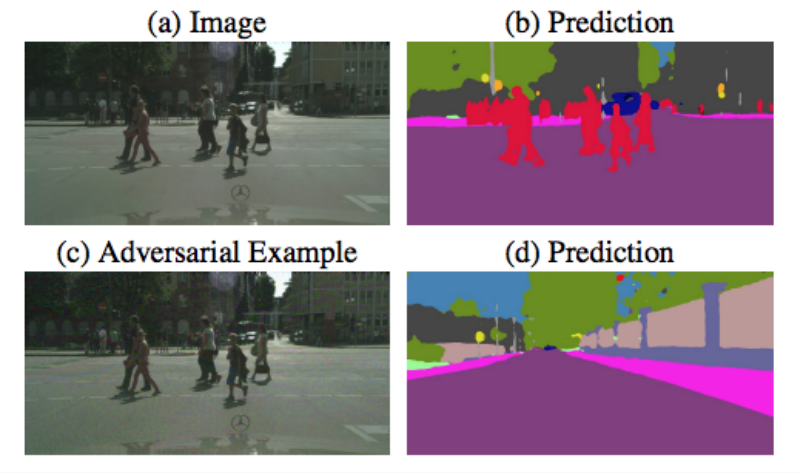
Adversarial Training

- Neural Networks are highly susceptible to small changes in their input
- An adversary can choose noise to add to input that "fools" the neural network.
- There are many methods to do this:
  - Fast Gradient Sign Method (FGSM)
  - Jacobian-based Saliency Map Approach (JSMA)
  - Carlini Methods



Adversarial Training

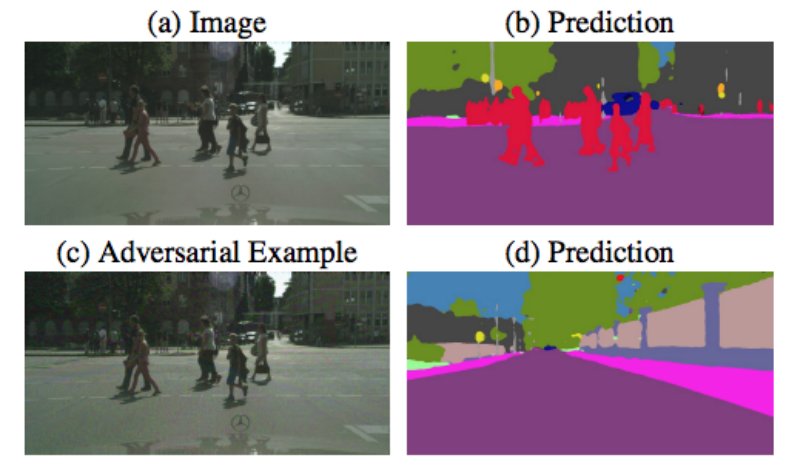
- Neural Networks are highly susceptible to small changes in their input
- An adversary can choose noise to add to input that "fools" the neural network.
- There are many methods to do this:
  - Fast Gradient Sign Method (FGSM)
  - Jacobian-based Saliency Map Approach (JSMA)
  - Carlini Methods



Adversarial Training

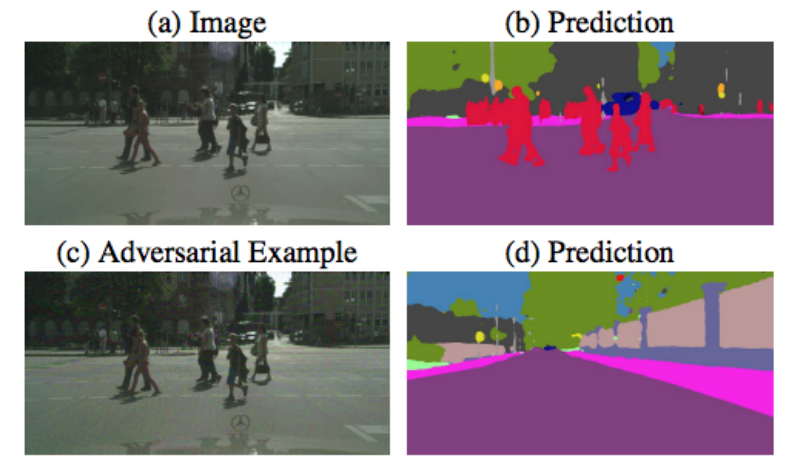
ML@B

- Neural Networks are highly susceptible to small changes in their input
- An adversary can choose noise to add to input that "fools" the neural network.
- There are many methods to do this:
  - Fast Gradient Sign Method (FGSM)
  - Jacobian-based Saliency Map Approach (JSMA)
  - Carlini Methods



Adversarial Training

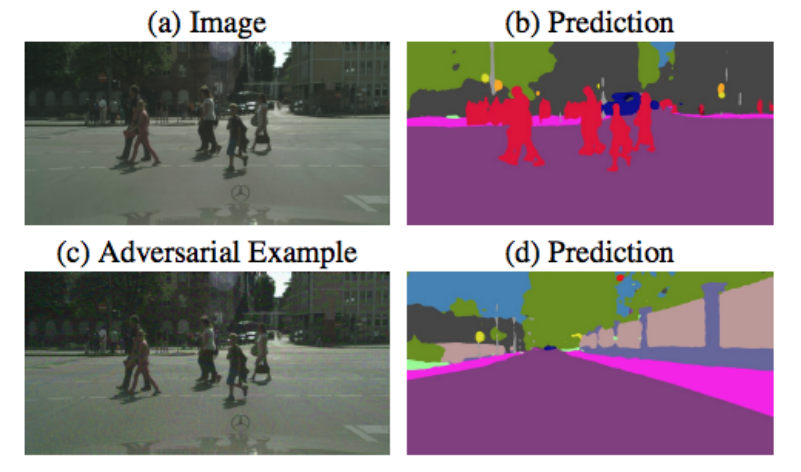
- Neural Networks are highly susceptible to small changes in their input
- An adversary can choose noise to add to input that "fools" the neural network.
- There are many methods to do this:
  - Fast Gradient Sign Method (FGSM)
  - Jacobian-based Saliency Map Approach (JSMA)
  - Carlini Methods



Adversarial Training

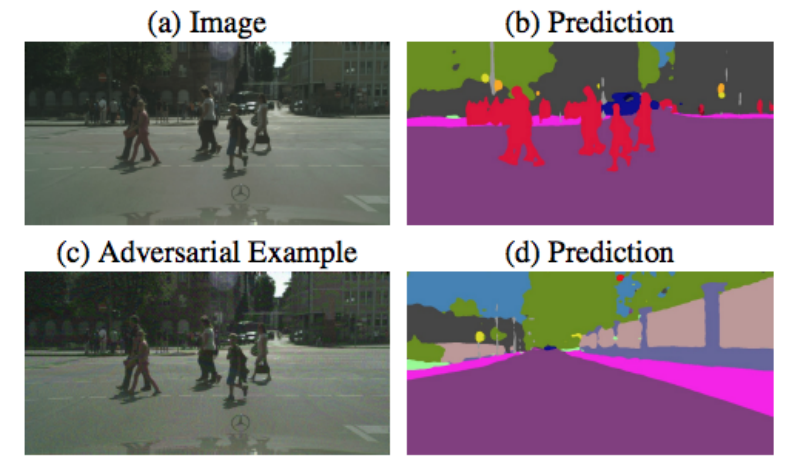
ML@B

- Neural Networks are highly susceptible to small changes in their input
- An adversary can choose noise to add to input that "fools" the neural network.
- There are many methods to do this:
  - Fast Gradient Sign Method (FGSM)
  - Jacobian-based Saliency Map Approach (JSMA)
  - Carlini Methods



Adversarial Training

- Neural Networks are highly susceptible to small changes in their input
- An adversary can choose noise to add to input that "fools" the neural network.
- There are many methods to do this:
  - Fast Gradient Sign Method (FGSM)
  - Jacobian-based Saliency Map Approach (JSMA)
  - Carlini Methods



Questions

Questions?