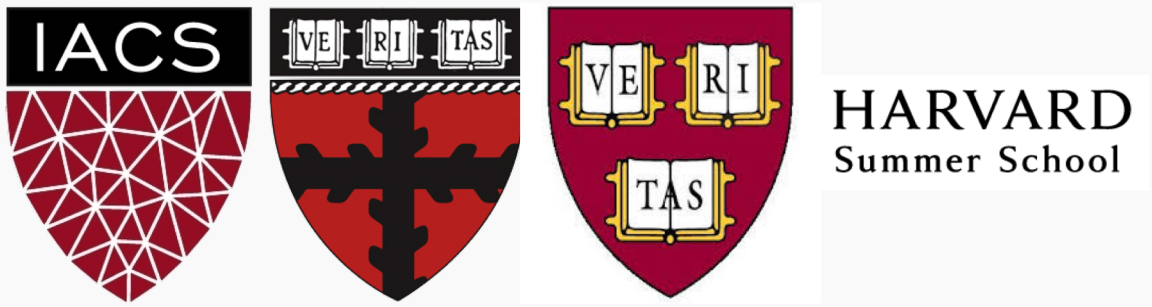
Lecture #11: Neural Network, Back Propagation, and Visualizing the Black Box

CS-S109A: Introduction to Data Science

Kevin Rader



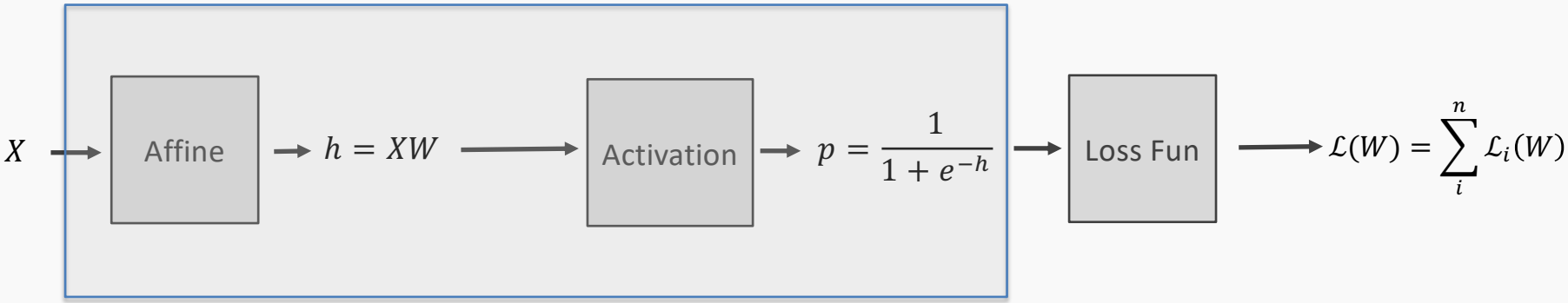
ANNOUNCEMENTS

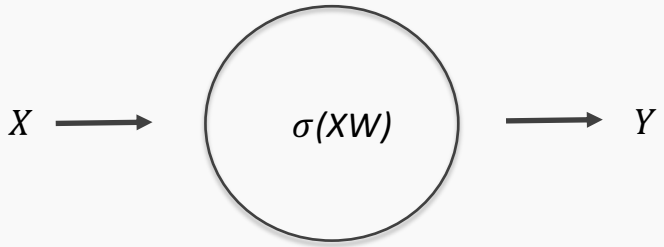
- **HW5** is due tomorrow (Tues) night at 11:59pm
- The **Final Exam** (individual work) will be posted tonight and is due Monday, Aug. 3. It is cumulative and will be more open-ended.
- We will go through a case study in **Wednesday's lecture**.
- **Friday's lab** will go through a little bit more on neural networks, but will mostly be a general review: both conceptual and technical.

Outline

- Brief Review of Neural Networks
- NN Design choices
- Visualizing the Black Box
- Back Propagation
- *Optimization* (not covered)
- *Regularization in NN* (not covered)

Brief Review of Neural Networks (NNs)

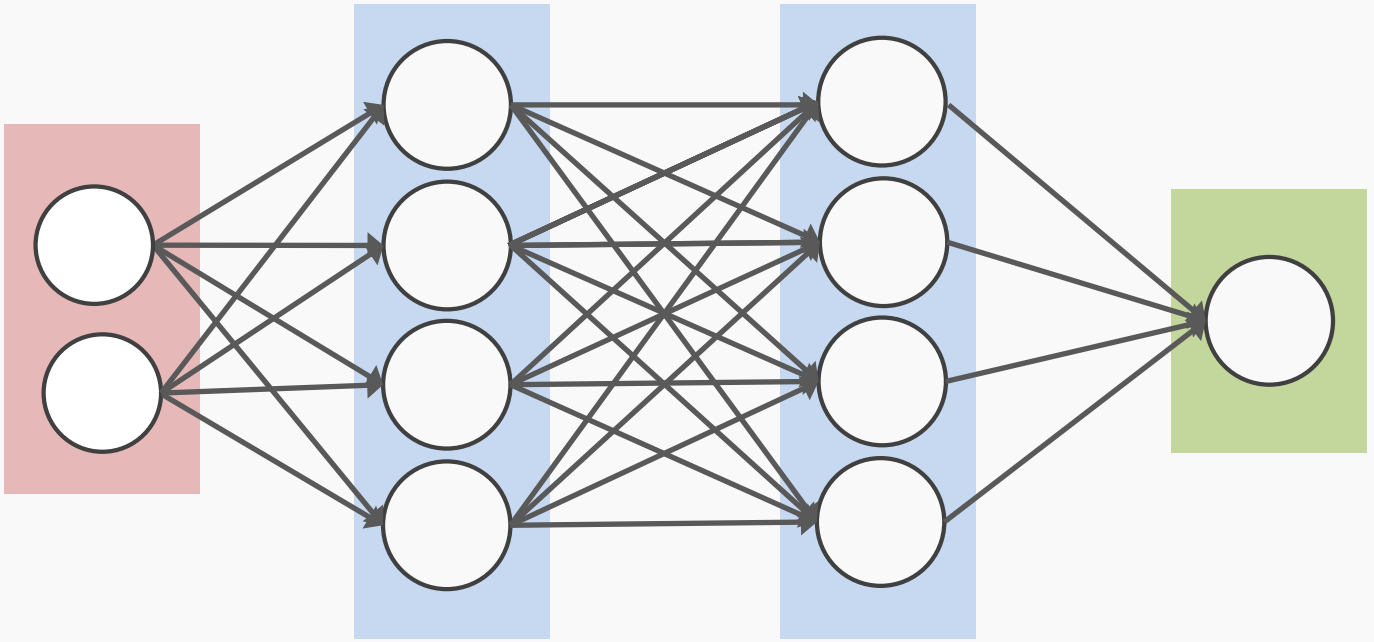




Neural Networks : Concept Check

- What are the weights in a NN? What are the nodes?
- What determines what loss function should be used when designing a NN? What loss function are reasonable to use?
- What determines the input and output shapes of a neural network?
- What activation function(s) should be considered when creating a NN?
- How could a NN be considered as a generalization of stacking?

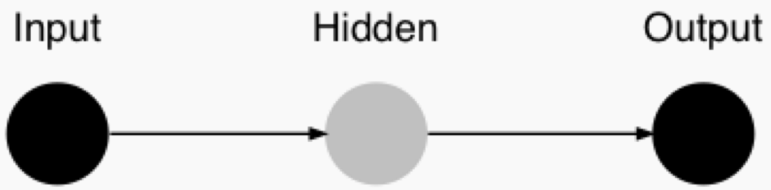
Anatomy of artificial neural network (ANN)

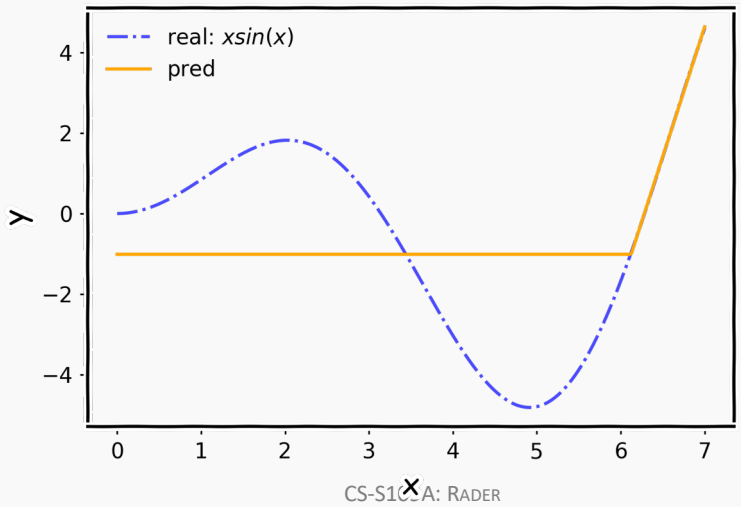


How many predictors are in this NN? How many hidden layers?
What would the Keras architecture look like?

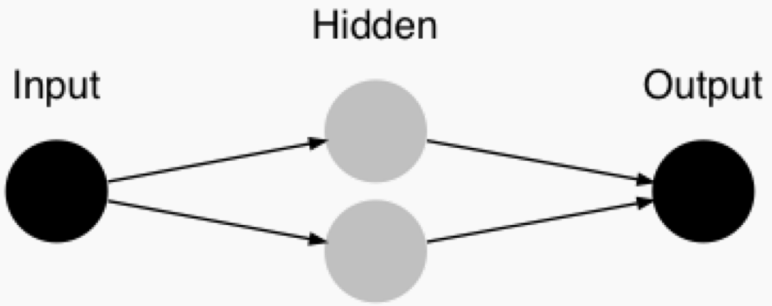
Design Choices: Architecture

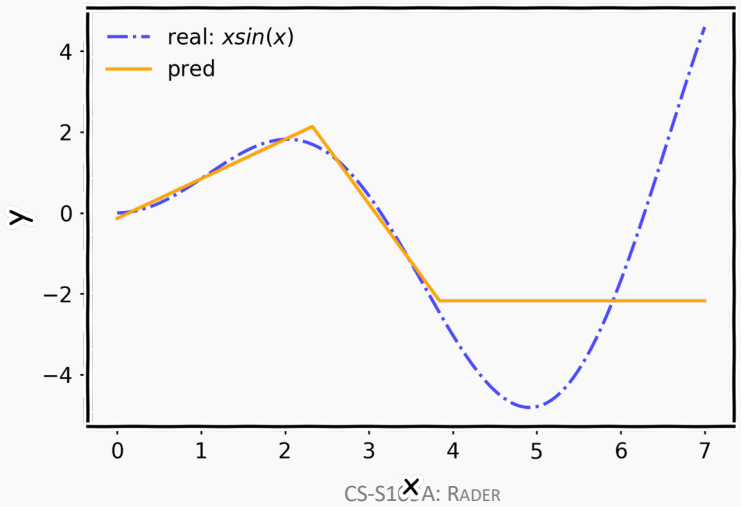
NN in action



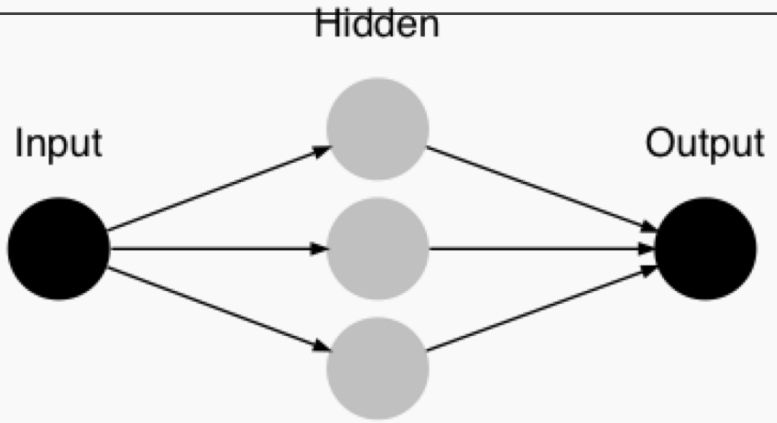


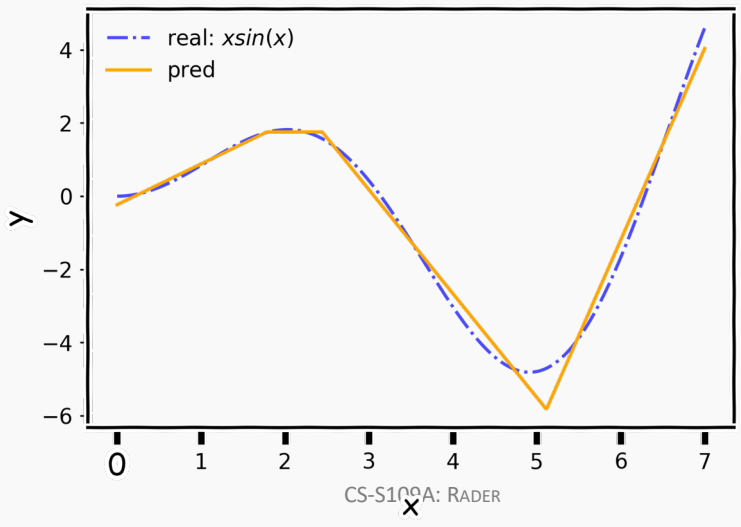
NN in action



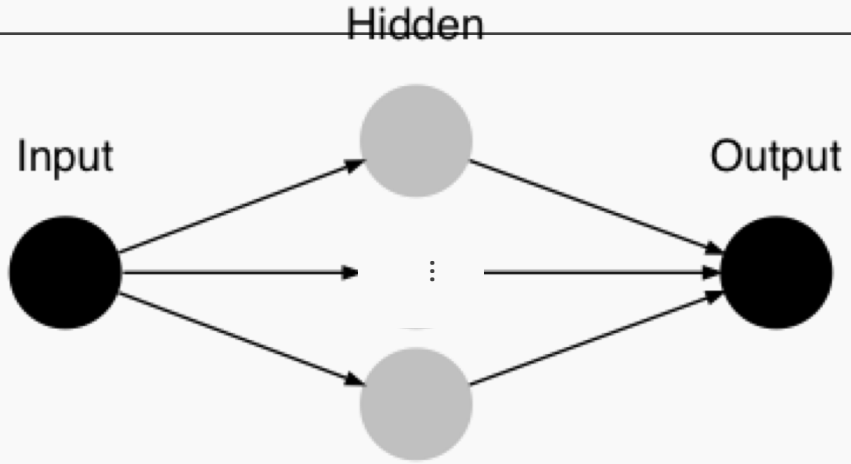


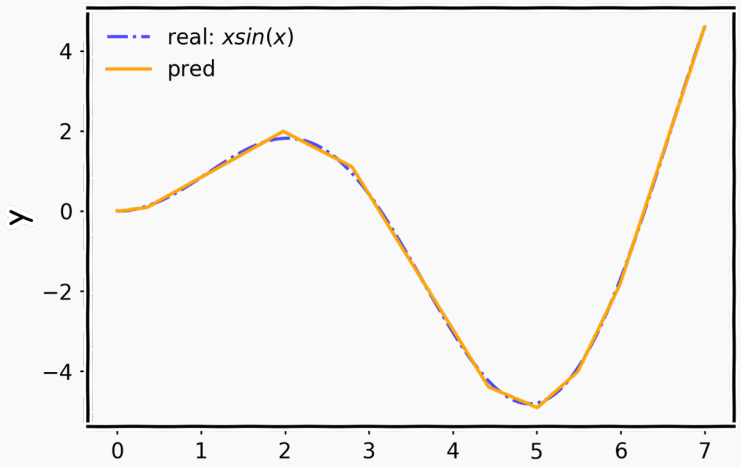
NN in action



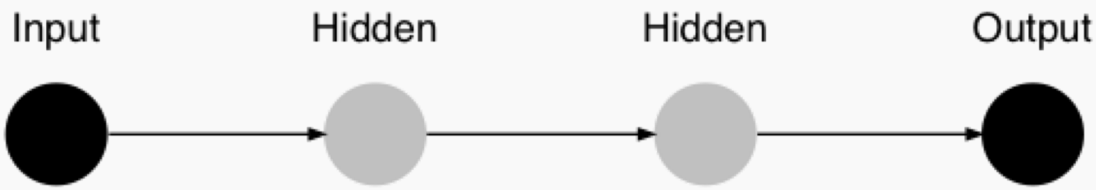


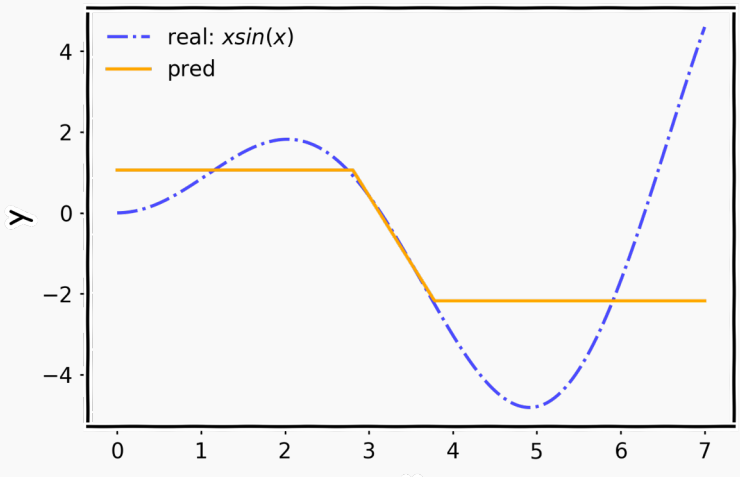
NN in action



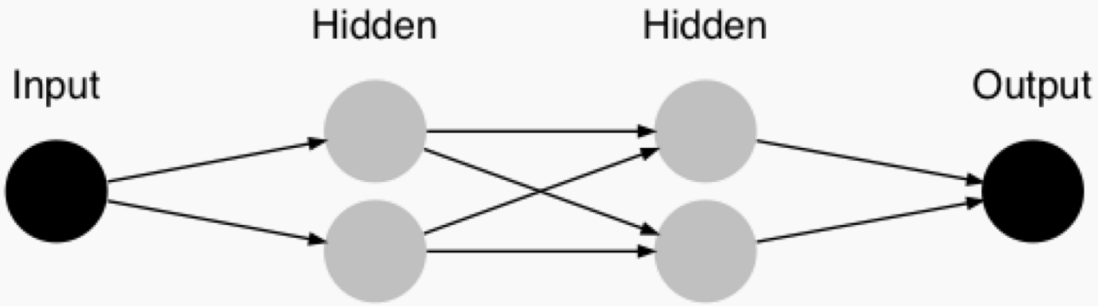


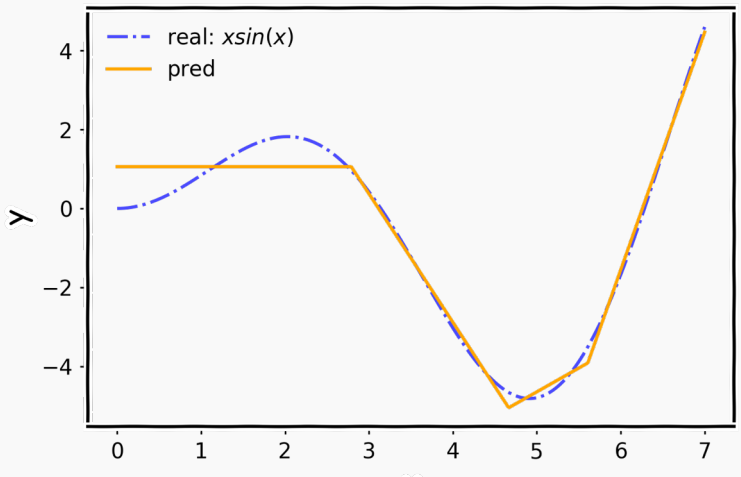
NN in action


Input
Hidden
Hidden
Output

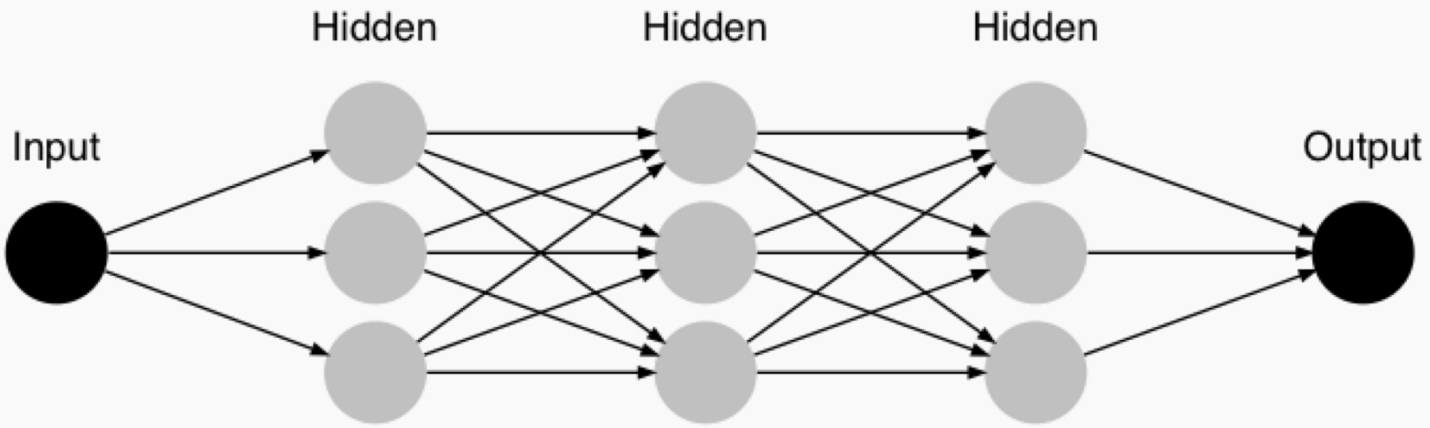

real: xsin(x)
pred
4
2
0
−2
−4
y
0
1
2
3
4
5
6
7
x

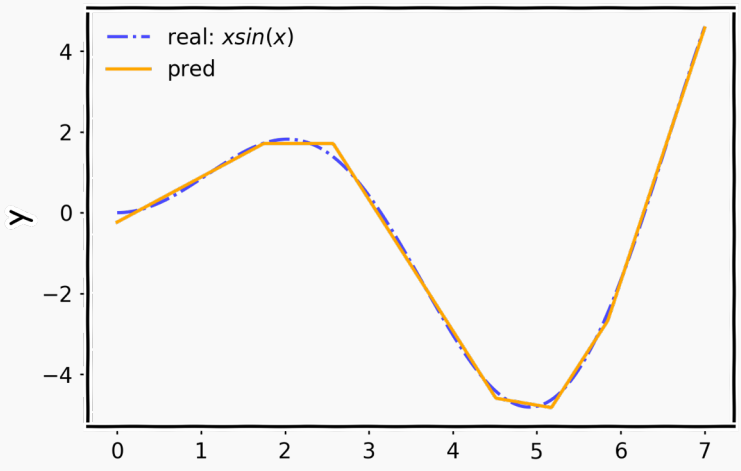
NN in action


Input
Hidden
Hidden
Output


real: xsin(x)
pred
y
x
4
2
0
−2
−4
0
1
2
3
4
5
6
7

NN in action


Hidden
Hidden
Hidden
Input
Output


real: xsin(x)
pred
y
x
4
2
0
−2
−4
0
1
2
3
4
5
6
7

Universal Approximation Theorem

Think of a Neural Network as function approximation.

$$Y = f(x) + \epsilon$$

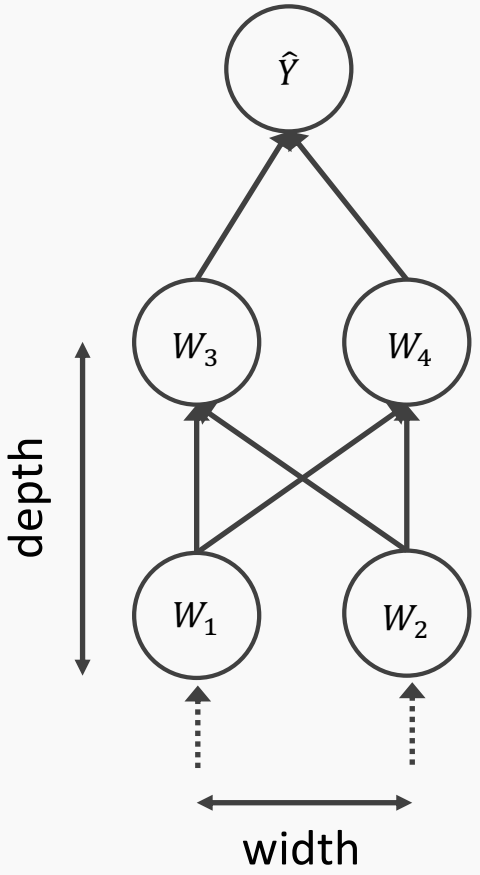
$$Y = \hat{f}(x) + \epsilon$$

$$\text{NN:} \Longrightarrow \hat{f}(x)$$

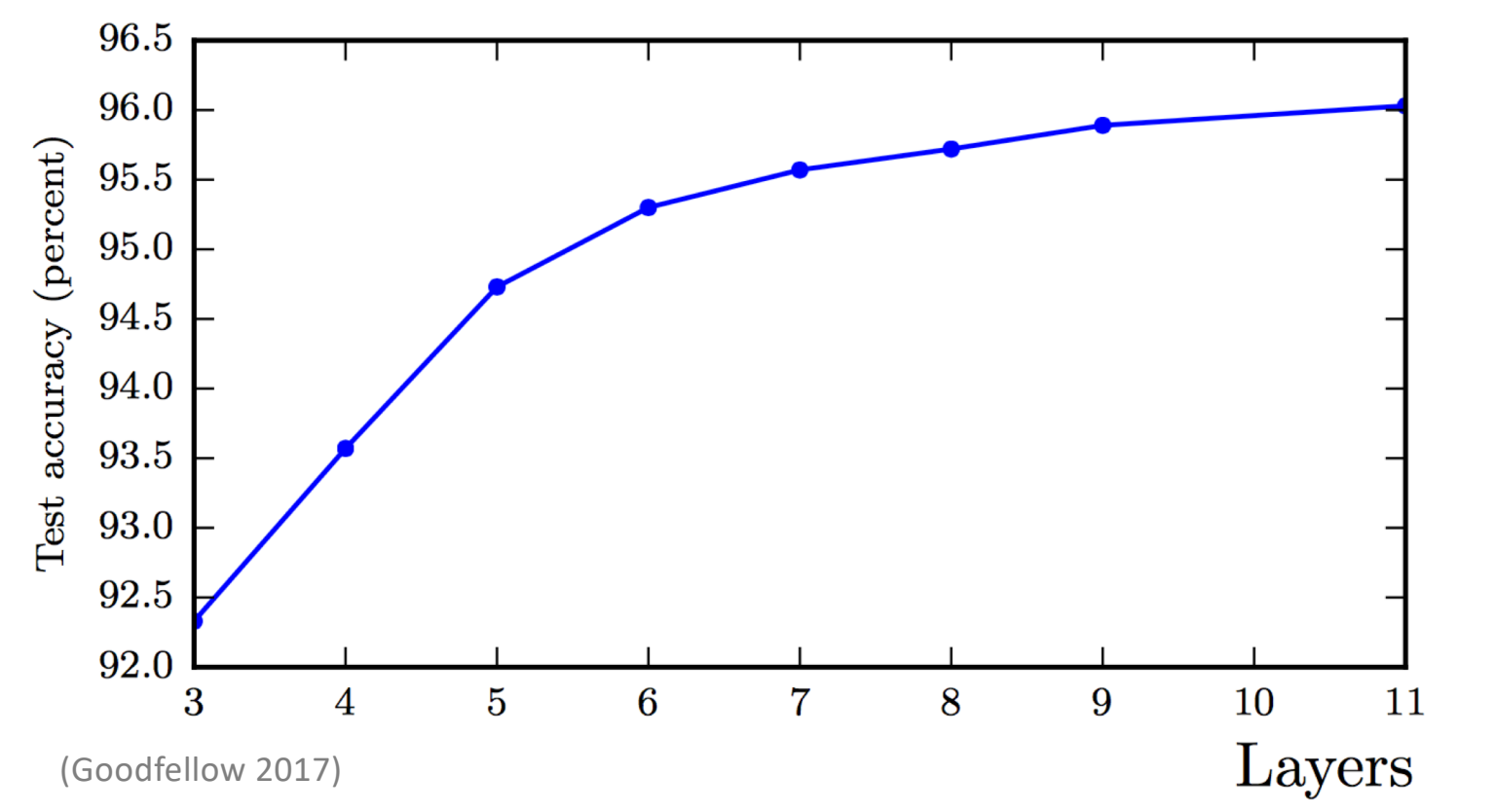
One hidden layer is enough to *represent* an approximation of any function to an arbitrary degree of accuracy

So why deeper?

- Shallow net may need (exponentially) more width
- Shallow net may overfit more



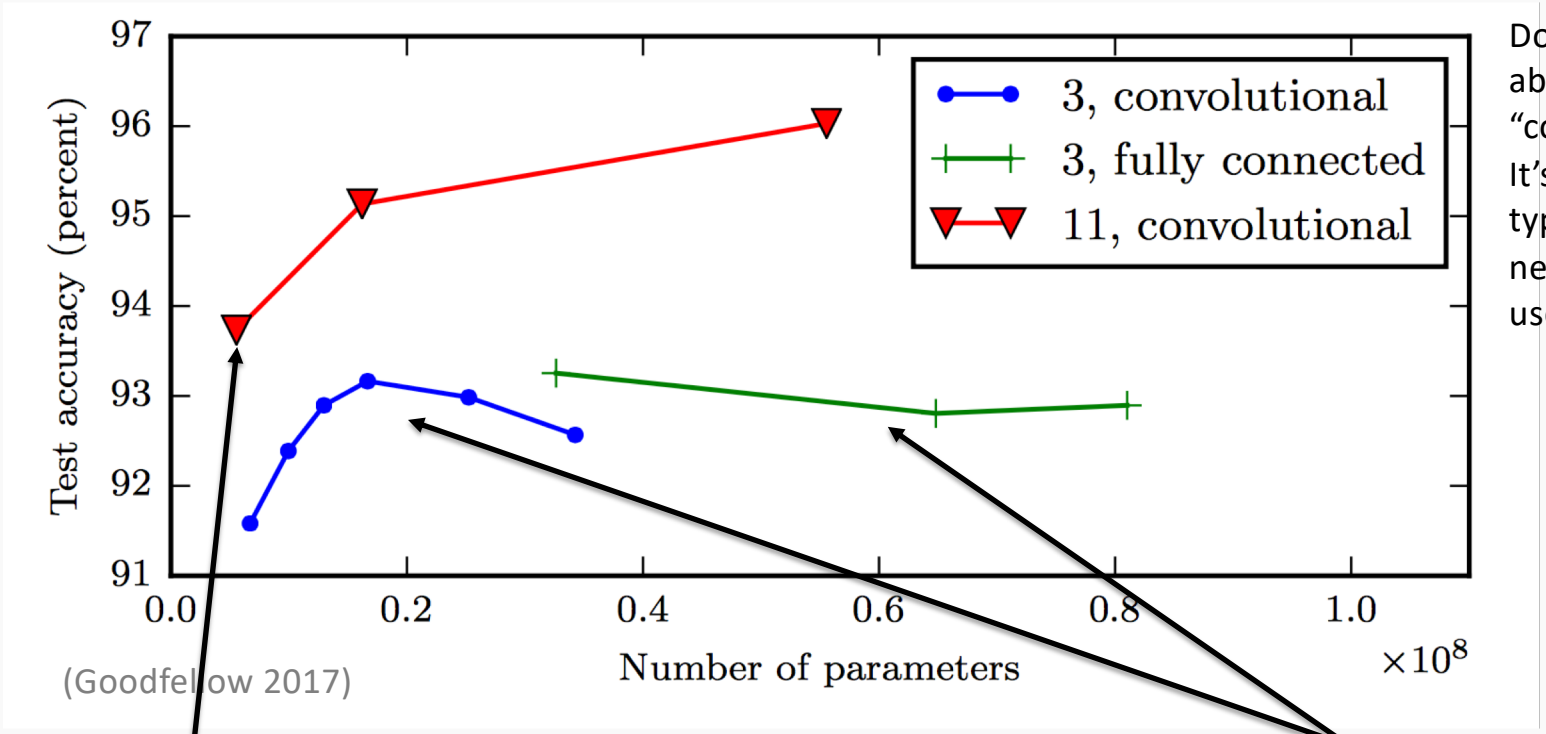
Better Generalization with Depth



(Goodfellow 2017)

Shallow Nets Overfit More

Depth helps, and it's not just because of more parameters



(Goodfellow 2017)

Don't worry about this word "convolutional". It's just a special type of neural network, often used for images.

The **11-layer net** generalizes better on the test set when controlling for number of parameters.

The 3-layer nets perform worse on the test set, even with similar number of total parameters.

Variable Importance

Variable Importance for Tree-Based Models

How does sklearn determine **variable importance** (feature_importance) from a tree-based model?

- It determines the improvement in the loss function every time a predictor is involved in a split.

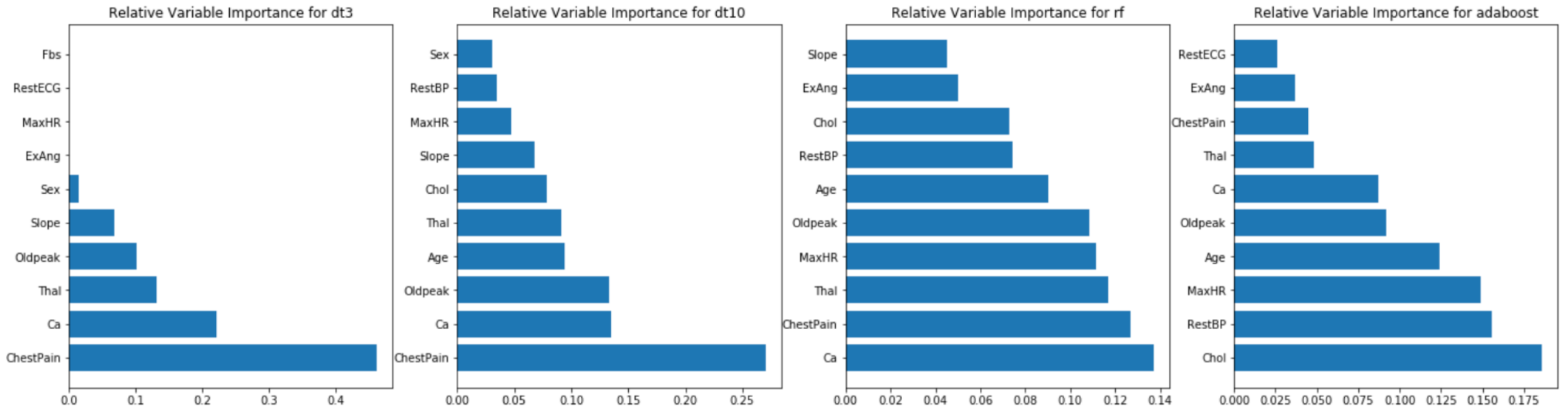
- More specifically, it calculate the total amount that the SSE (for regression) or Gini index (for classification) is improved (decreased) due to splits over a given predictor (averaged over all B trees if a bagged/random forest method).

How should variable importance compare across the various different tree models we've considered (trees, random forests/bagging, and boosting)?

A picture is worth a thousand words…

Variable Importance for trees, bags, and boosts

Below are the variable importance plots for the top 10 predictors for each of a (i) decision tree with maxdepth=3, (ii) decision tree with maxdepth=10, (iii) a random forest, and (iv) an adaboost classifier.



Compare them? Are the differences surprising?

Feature Importance in a Neural Network

How can one measure feature importance in a Neural Network?

Its not so easy ☹. What could potentially be done?

We can calculate the predictions from a neural net, and then fit a decision tree model to the predicted values.

What can go wrong with this approach?

Other Variable Importance Measures

What other approaches can be taken to measure variable importance?

Alternative:

- Record the prediction accuracy on the *oob* samples for each tree.
- Randomly permute the data for column j in the *oob* samples then record the accuracy again.
- The decrease in accuracy as a result of this permuting is averaged over all trees, and is used as a measure of the importance of variable j in the random forest.

Permutation Variable Importance

This idea of re-permuting a variable and refitting a model to see how much more poorly it performs is called **permutation feature importance.**

It is sometimes preferred to the standard feature importance, why?

When two features are correlated and one of the features is permuted, the model will still have access to the feature through its correlated feature.

What is the one glaring disadvantage to this?

Computational time, and things may not 'add up'.

Permutation Variable Importance in sklearn

To perform permutation variable importance in sklearn (or keras), one can use the ELI5 library:

https://eli5.readthedocs.io/en/latest/index.html

This is easy to do with sklearn models, but not easy to do with tensorflow (but it can be done).

An example is worth a thousand words:

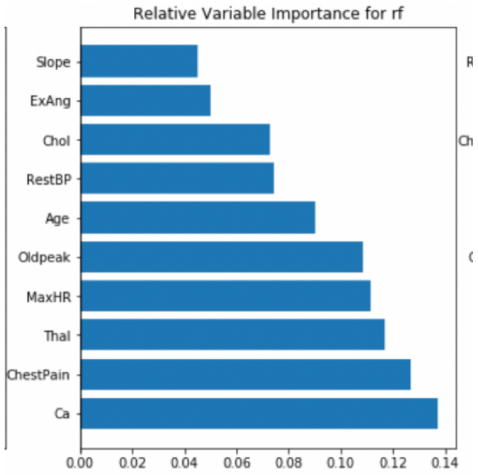
Permutation Variable Importance Example

```
#permutation importance
from eli5.sklearn import PermutationImportance
from eli5.permutation_importance import get_score_importances


perm = PermutationImportance(randomforest).fit(X_test, y_test)
#eli5.show_weights(perm,feature_names=X.columns)
print(X.columns)
eli5.explain_weights(perm)
```

```
Index(['Age', 'Sex', 'ChestPain', 'RestBP', 'Chol', 'Fbs', 'RestECG', 'MaxHR',
       'ExAng', 'Oldpeak', 'Slope', 'Ca', 'Thal'],
      dtype='object')
```

Weight	Feature
0.1082 ± 0.0161	x12
0.0689 ± 0.0435	x7
0.0393 ± 0.0675	x11
0.0361 ± 0.0636	x2
0.0328 ± 0.0359	x8
0.0295 ± 0.0245	x9
0.0295 ± 0.0131	x1
0.0197 ± 0.0245	x3
0.0164 ± 0.0688	x10
0.0131 ± 0.0131	x6
0.0098 ± 0.0161	x4
0 ± 0.0000	x5
-0.0098 ± 0.0334	x0



The problem with Variable Importance

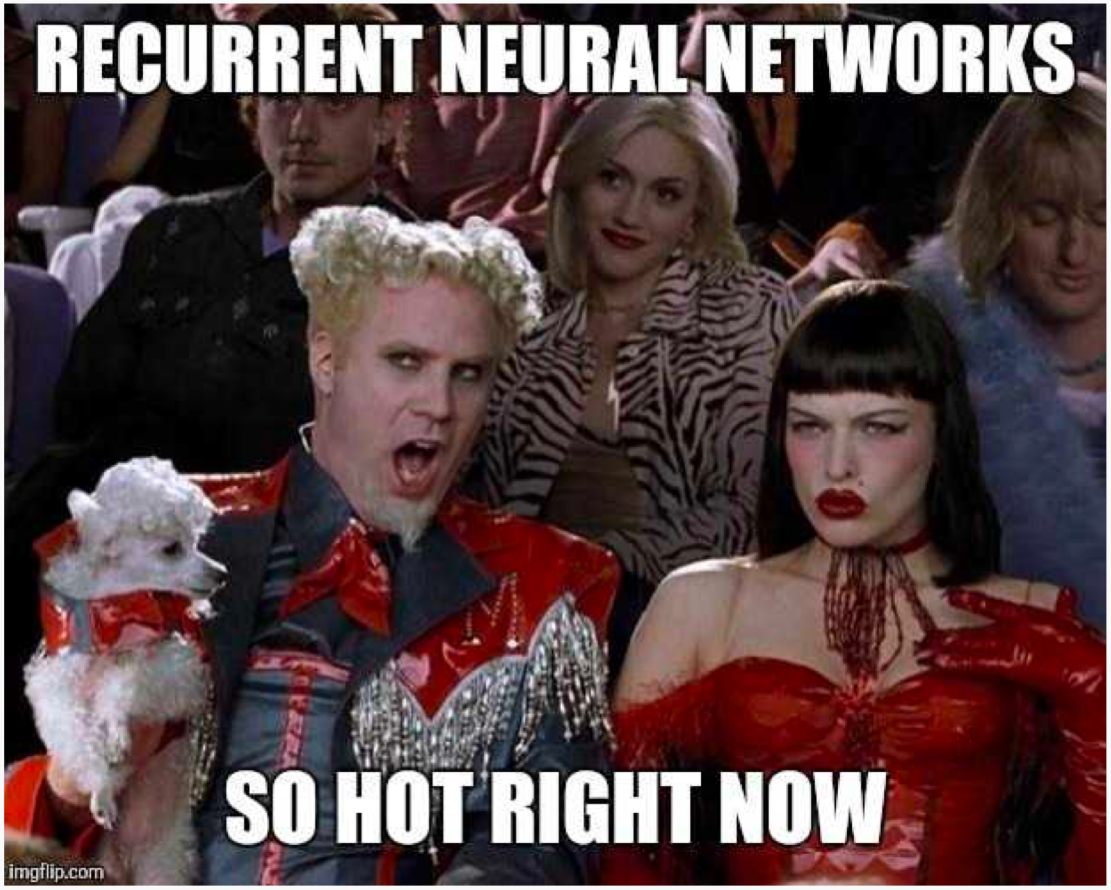
Variable Importance is great! It tells you what features are important in shaping the model.

But what is missing?

- It does not give any measure for how the predictors are related to the response (positive, negative, quasi-linear, curved, interactions, etc.).
- This is where the parametric model wins out! Inference and interpretations are much easier and the whole point in these models.

What can we do to measure these relationships in a machine learning or non-parametric model? Think: what did we do with *k*-NN?

What needs to be done algorithmically to put this in practice?


RECURRENT NEURAL NETWORKS
SO HOT RIGHT NOW
imgflip.com

Interpretation through Predictions

Parametric vs. Nonparametric models

In a machine learning model (like ensemble methods and neural networks), the association between predictors and the response are not measured directly as these models are 'black box' models:

Inputs (*X*, predictors). → black box (NN, sklearn, etc.) → Outputs (*Y*, response)

What if we care about how the predictors relate to the response? This is where we need to figure out what the black box is doing to transform the inputs into the outputs.

Simplest Approach: observed $\hat{Y}$ vs. X_j

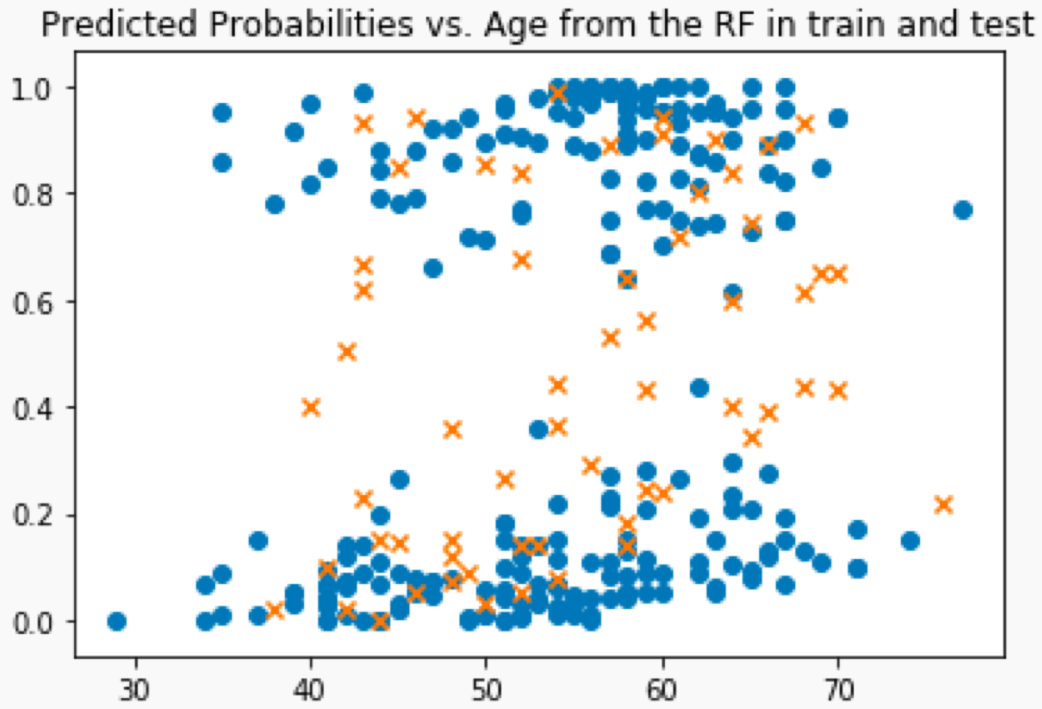
Use predict (or better yet, predict_proba) to plot the observed predicted values vs. the observed values for X_j.

What is a problem with this approach how can we fix it?

The fix is not so easy. We cannot just fit a logistic regression model so easily to the predicted probabilities. Why not?

An example is worth a thousand words…

Example: observed $\hat{Y}$ vs. X_j



Unboxing the black box

Inputs (*X*, predictors). → black box (NN, sklearn, etc.) → Outputs (*Y*, response)

This gives us our approach: vary the inputs (the predictors) and see what happens to the response.

If we care about the 'marginal' or 'conditional' effect of how a specific *X* relates to *Y*, then we should vary only one predictor at a time.

How should we handle the other predictors? That is to say, what value should we keep them at?

Unboxing the black box (cont.)

There are two general approaches to interpreting the machine learning model through predictions:

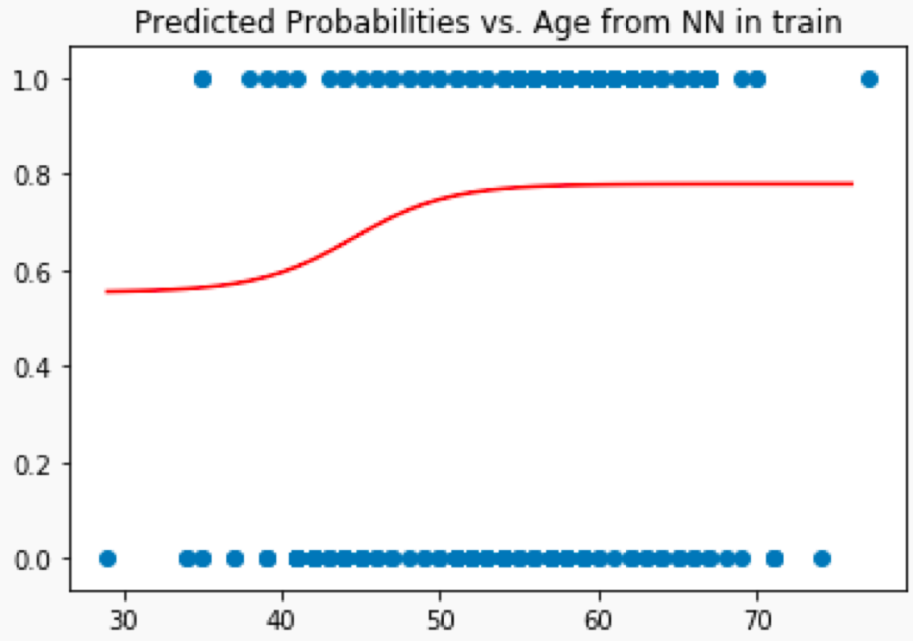
The approach is just like in multiple regression: what is the marginal effect of a unit change in X_j holding all the other predictors constant.

So at what values should we hold the other predictors?

There are two general approaches **holding the other predictors constant**:

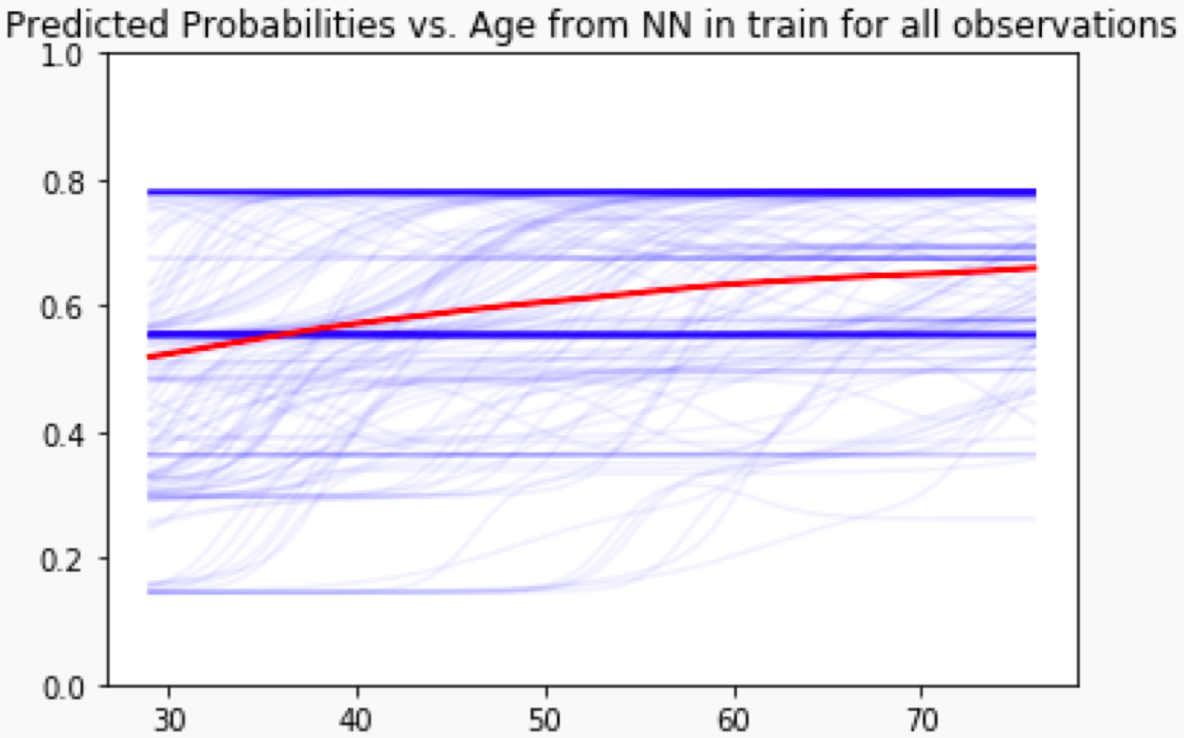
1. Predict $\hat{Y}$ at the **mean/median (or most common) value** for each of the other predictors, vary only the predictor you care about, X_j, and plot the predictions $\hat{Y}$ vs. X_j.
2. Predict $\hat{Y}$ at the **observed values of** for all the other predictors, vary only the predictor you care about, X_j, and plot the predictions $\hat{Y}$ vs. X_j. Essentially this means creating a new data frame for each observation, and imputing all reasonable values of X_j in.

Predicted probabilities vs. Age at the means



Interpret this plot. What does this say for how Cardiac Arrest is associated with Age?

Predicted probabilities vs. Age for all observations



Interpret this plot. What does this say for how Cardiac Arrest is associated with Age?

What if we want the joint relationship with two predictors?

How can we look at how the response relates to two predictors at once?

This is a bit trickier to do. Why?

We can go back to our friend: the classification boundary!

Should I use PCA components?

What if we want an overall picture of how the response relates to the predictors?

Well, this is where PCA components sometimes come in to play.

But this is no Bueno for interpretability. Why?

Adding Uncertainty

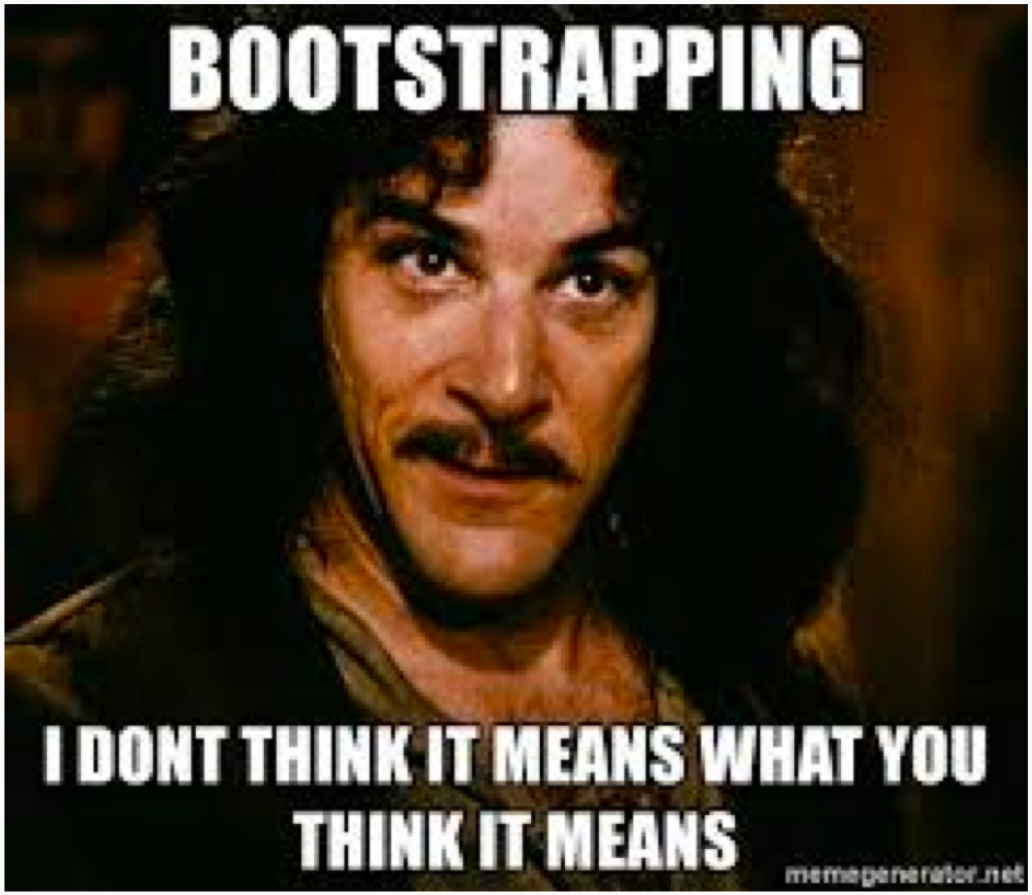
Adding Uncertainty

Not only should we plot predictions, but we can (should) also include the uncertainty of these predictions.

For example, there may be a big difference between the uncertainty of estimated probability curve if $n = 100$ vs. $n = 10{,}000$.

How can we add uncertainty measures to these predictions? How do we do it for linear/logistic regression?

Bootstrapping


BOOTSTRAPPING
I DONT THINK IT MEANS WHAT YOU THINK IT MEANS
memegenerator.net

Bootstrapping Basics, review

How do we perform a bootstrap? What are the key steps?

The bootstrap algorithm is an approach to mimic the uncertainty of taking the observed sample from a population. So we:

1. Sample the same number of observations as was observed (in train).
2. Sample with replacement (so we get some randomness).

Why does this work?

Bootstrapping Basics, review

When is a bootstrapping approach used?

1. When a probabilistic closed form solution is not known or not easy to get at.
2. When the assumptions for a parametric model break down.
3. When there is no 'formula' at all to add the uncertainty into a calculation.
4. Or we are just being lazy.

Bootstrapping Basics, review

How do we use the results of a bootstrap technique?

It provides us a re-created **estimate** every time we bootstrap.

This estimate could be a mean, a beta, a plot of average predictions, etc.

But they will almost always be for a summary, not a single observation, and thus the sample size is already taken care of (the *n* in the formula). Aka, do not divide by sqrt(n) again!

Bootstrapping for Predictions

So we can refit a model on a bootstrap sample of data, and look at the predicted probabilities each time.

Then we can take the top 97.5^{th} percentile and bottom 2.5^{th} percentile to build a 95% confidence interval for the predictions!

But how does this work for random forests and bagging models? Yikes!

Back Propagation

Learning the coefficients

Start with Regression or Logistic Regression

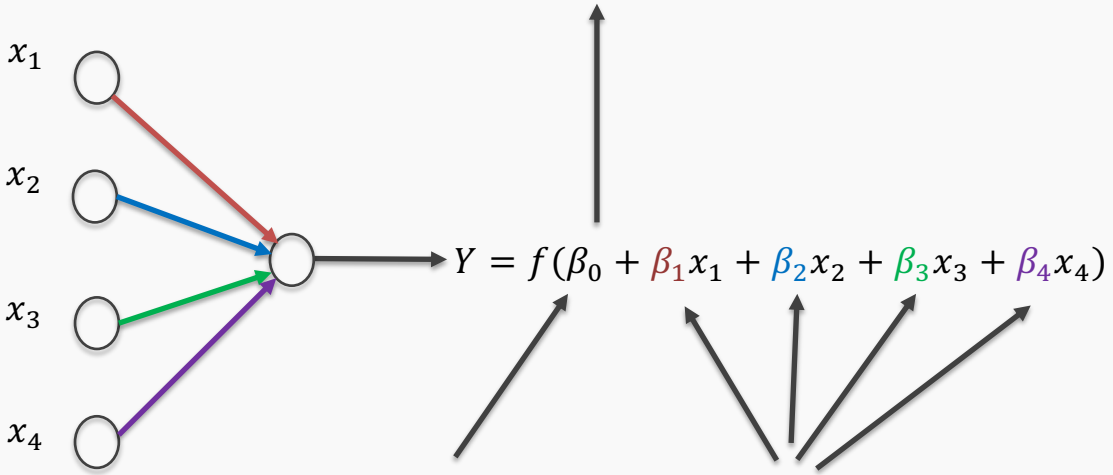
Classification

$f(X) = \frac{1}{1+e^{-W^T X}}$

Regression

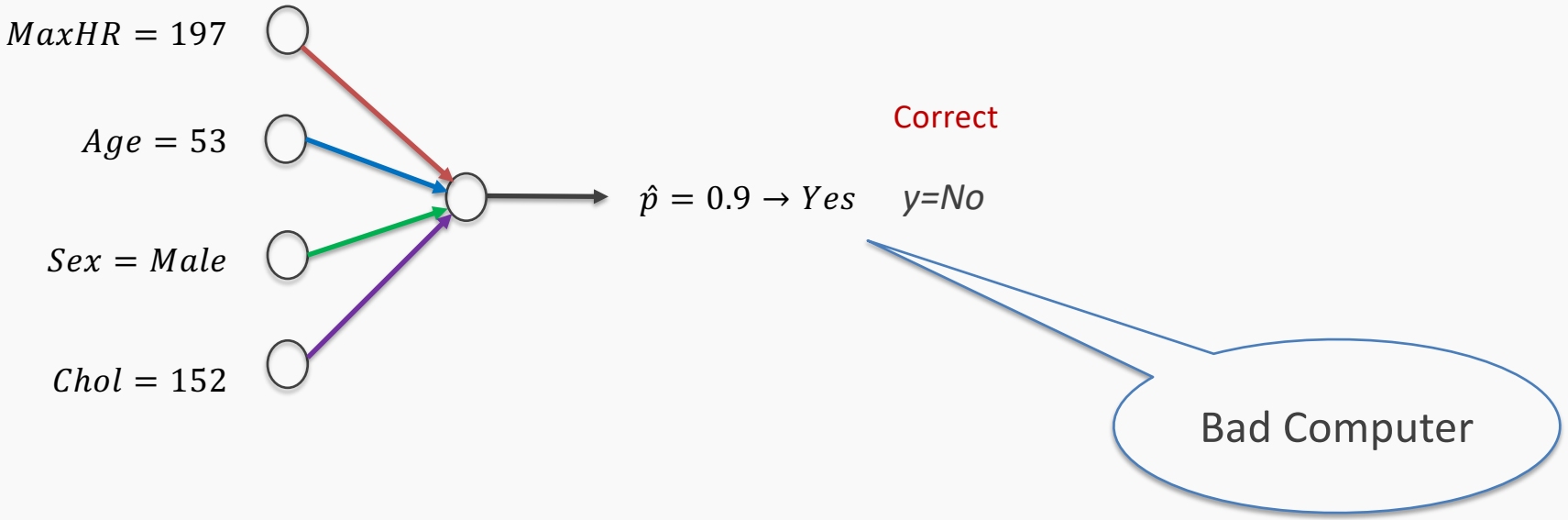
$f(X) = W^T X$

$W^T = [W_0, W_1, \ldots, W_4]$
$= [\beta_0, \beta_1, \ldots, \beta_4]$



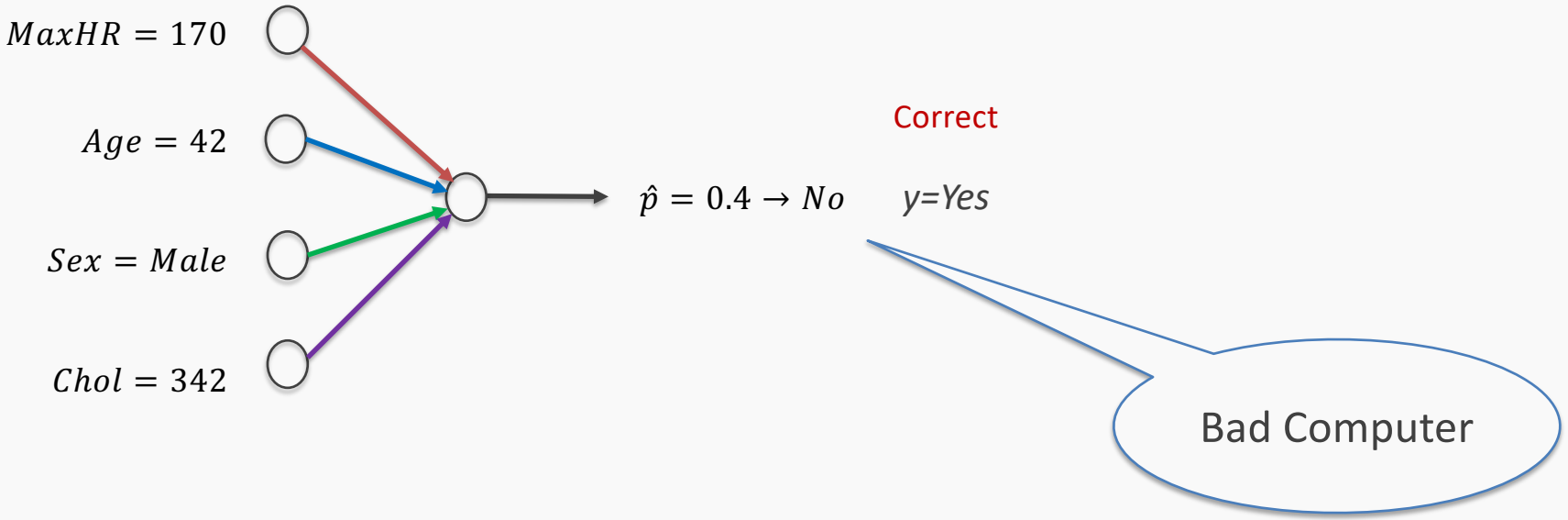
But what is the idea?

Start with all randomly selected weights. Most likely it will perform horribly.
For example, in our heart data, the model will be giving us the wrong answer.



But what is the idea?

Start with all randomly selected weights. Most likely it will perform horribly.
For example, in our heart data, the model will be giving us the wrong answer.



But what is the idea?

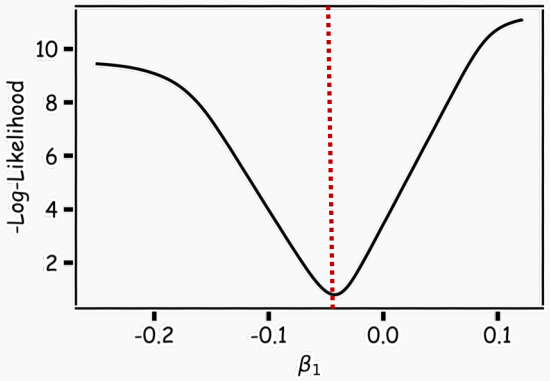
- **Loss Function:** Takes all of these results and averages them and tells us how bad or good the computer or those weights are.
- Telling the computer how **bad** or **good** is, does not really help.
- You want to tell it how to change those weights so it gets better.

Loss function: $\mathcal{L}(w_0, w_1, w_2, w_3, w_4)$

For now let's only consider one weight, $\mathcal{L}(w_1)$

Minimizing the Loss function

Ideally we want to know the value of w_1 that gives the minimal $\mathcal{L}(W)$

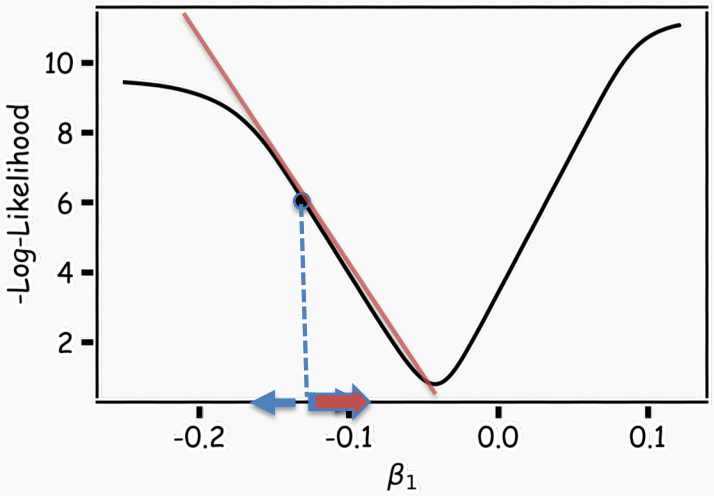


To find the optimal point of a function $\mathcal{L}(W)$

$$\frac{d\mathcal{L}(W)}{dW} = 0$$

And find the W that satisfies that equation. Sometimes there is no explicit solution for that.

Estimate of the regression coefficients: gradient descent



A more flexible method is

- Start from a random point
  1. Determine which direction to go to reduce the loss (left or right)
  2. Compute the slope of the function at this point and step to the right if slope is negative or step to the left if slope is positive
  3. Goto to #1

Minimization of the Loss Function

Question: What is the mathematical function that describes the slope?

Derivative

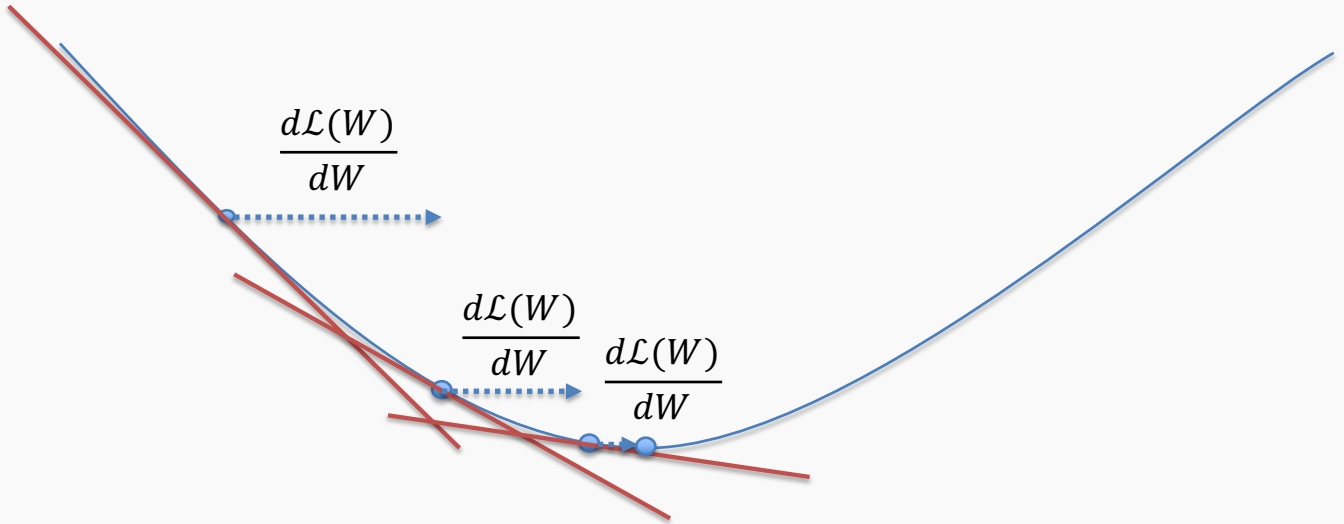
Question: How do we generalize this to more than one predictor?

Take the derivative with respect to each coefficient and do the same sequentially

Question: What do you think is a good approach for telling the model how to change (what is the step size) to become better?

Gradient Descent (cont.)

If the step is proportional to the slope then you avoid overshooting the minimum. How?



Let's play the game

We know that we want to go in the opposite direction of the derivative and we know we want to be making a step proportional to the derivative.

Making a step means:

$$w^{new} = w^{old} + step$$

Step size is proportional to derivative

Opposite direction of the derivative means:

$$w^{new} = w^{old} - \lambda \frac{d\mathcal{L}}{dw}$$

Learning Rate

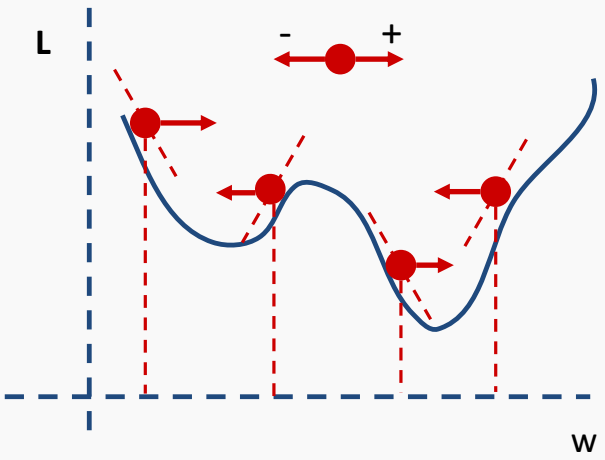
Change to more conventional notation:

$$w^{(i+1)} = w^{(i)} - \lambda \frac{d\mathcal{L}}{dw}$$

Gradient Descent

- Algorithm for optimization of first order to finding a minimum of a function.
- It is an iterative method.
- *L* is decreasing much faster in the direction of the negative derivative.
- The learning rate is controlled by the magnitude of λ.

$$w^{(i+1)} = w^{(i)} - \lambda \frac{d\mathcal{L}}{dw}$$



Considerations

- We still need to calculate the derivatives.
- We need to know what is the learning rate or how to set it.
- Local vs global minima.
- The full likelihood function includes summing up all individual '*errors'.* Unless you are a statistician, sometimes this includes hundreds of thousands of examples.

Considerations

- **We still need to calculate the derivatives.**
- We need to know what is the learning rate or how to set it.
- Local vs global minima.
- The full likelihood function includes summing up all individual '*errors'.* Unless you are a statistician, sometimes this includes hundreds of thousands of examples.

Calculate the Derivatives

Can we do it?

Wolfram Alpha can do it for us!

We need a formalism to deal with these derivatives.

Example: Logistic Regression derivatives

Chain Rule

Chain rule for computing gradients:

- $y = g(x) \qquad z = f(y) = f\big(g(x)\big) \qquad\qquad \boldsymbol{y} = g(\boldsymbol{x}) \qquad z = f(\boldsymbol{y}) = f\big(g(\boldsymbol{x})\big)$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}$$

- For longer chains:

$$\frac{\partial z}{\partial x_i} = \sum_{j_1} \cdots \sum_{j_m} \frac{\partial z}{\partial y_{j_1}} \cdots \frac{\partial y_{j_m}}{\partial x_i}$$

Logistic Regression derivatives

For logistic regression, the negative log of the likelihood is:

$$\mathcal{L} = \sum_i \mathcal{L}_i = -\sum_i \log L_i = -\sum_i [y_i \log p_i + (1 - y_i) \log(1 - p_i)]$$

$$\mathcal{L}_i = -y_i \log \frac{1}{1 + e^{-W^T X}} - (1 - y_i) \log(1 - \frac{1}{1 + e^{-W^T X}})$$

To simplify the analysis let us split it into two parts,

$$\mathcal{L}_i = \mathcal{L}_i^A + \mathcal{L}_i^B$$

So the derivative with respect to *W* is:

$$\frac{\partial \mathcal{L}}{\partial W} = \sum_i \frac{\partial \mathcal{L}_i}{\partial W} = \sum_i (\frac{\partial \mathcal{L}_i^A}{\partial W} + \frac{\partial \mathcal{L}_i^B}{\partial W})$$

$$\mathcal{L}_i^A = -y_i \log \frac{1}{1 + e^{-W^T X}}$$

Variables	Partial derivatives	Partial derivatives
$\xi_1 = -W^T X$	$\frac{\partial \xi_1}{\partial W} = -X$	$\frac{\partial \xi_1}{\partial W} = -X$
$\xi_2 = e^{\xi_1} = e^{-W^T X}$	$\frac{\partial \xi_2}{\partial \xi_1} = e^{\xi_1}$	$\frac{\partial \xi_2}{\partial \xi_1} = e^{-W^T X}$
$\xi_3 = 1 + \xi_2 = 1 + e^{-W^T X}$	$\frac{\partial \xi_3}{\partial \xi_2} = 1$	$\frac{\partial \xi_3}{\partial \xi_2} = 1$
$\xi_4 = \frac{1}{\xi_3} = \frac{1}{1 + e^{-W^T X}} = p$	$\frac{\partial \xi_4}{\partial \xi_3} = -\frac{1}{\xi_3^2}$	$\frac{\partial \xi_4}{\partial \xi_3} = -\frac{1}{\left(1 + e^{-W^T X}\right)^2}$
$\xi_5 = \log \xi_4 = \log p = \log \frac{1}{1 + e^{-W^T X}}$	$\frac{\partial \xi_5}{\partial \xi_4} = \frac{1}{\xi_4}$	$\frac{\partial \xi_5}{\partial \xi_4} = 1 + e^{-W^T X}$
$\mathcal{L}_i^A = -y\xi_5$	$\frac{\partial \mathcal{L}}{\partial \xi_5} = -y$	$\frac{\partial \mathcal{L}}{\partial \xi_5} = -y$
$\frac{\partial \mathcal{L}_i^A}{\partial W} = \frac{\partial \mathcal{L}_i}{\partial \xi_5} \frac{\partial \xi_5}{\partial \xi_4} \frac{\partial \xi_4}{\partial \xi_3} \frac{\partial \xi_3}{\partial \xi_2} \frac{\partial \xi_2}{\partial \xi_1} \frac{\partial \xi_1}{\partial W}$		$\frac{\partial \mathcal{L}_i^A}{\partial W} = -yXe^{-W^T X} \frac{1}{\left(1 + e^{-W^T X}\right)}$

$$\mathcal{L}_i^B = -(1-y_i)\log[1-\frac{1}{1+e^{-W^TX}}]$$

Variables	derivatives	Partial derivatives wrt to X,W
$\xi_1 = -W^TX$	$\frac{\partial \xi_1}{\partial W} = -X$	$\frac{\partial \xi_1}{\partial W} = -X$
$\xi_2 = e^{\xi_1} = e^{-W^TX}$	$\frac{\partial \xi_2}{\partial \xi_1} = e^{\xi_1}$	$\frac{\partial \xi_2}{\partial \xi_1} = e^{-W^TX}$
$\xi_3 = 1 + \xi_2 = 1 + e^{-W^TX}$	$\frac{\partial \xi_3}{\partial \xi_2} = 1$	$\frac{\partial \xi_3}{\partial 2} = 1$
$\xi_4 = \frac{1}{\xi_3} = \frac{1}{1+e^{-W^TX}} = p$	$\frac{\partial \xi_4}{\partial \xi_3} = -\frac{1}{\xi_3^2}$	$\frac{\partial \xi_4}{\partial \xi_3} = -\frac{1}{\left(1+e^{-W^TX}\right)^2}$
$\xi_5 = 1 - \xi_4 = 1 - \frac{1}{1+e^{-W^TX}}$	$\frac{\partial \xi_5}{\partial \xi_4} = -1$	$\frac{\partial \xi_5}{\partial \xi_4} = -1$
$\xi_6 = \log \xi_5 = \log(1-p) = \log \frac{1}{1+e^{-W^TX}}$	$\frac{\partial \xi_6}{\partial \xi_5} = \frac{1}{\xi_5}$	$\frac{\partial \xi_6}{\partial \xi_5} = \frac{1+e^{-W^TX}}{e^{-W^TX}}$
$\mathcal{L}_i^B = (1-y)\xi_6$	$\frac{\partial \mathcal{L}}{\partial \xi_6} = 1 - y$	$\frac{\partial \mathcal{L}}{\partial \xi_6} = 1 - y$
$\frac{\partial \mathcal{L}_i^B}{\partial W} = \frac{\partial \mathcal{L}_i^B}{\partial \xi_6}\frac{\partial \xi_6}{\partial \xi_5}\frac{\partial \xi_5}{\partial \xi_4}\frac{\partial \xi_4}{\partial \xi_3}\frac{\partial \xi_3}{\partial \xi_2}\frac{\partial \xi_2}{\partial \xi_1}\frac{\partial \xi_1}{\partial W}$		$\frac{\partial \mathcal{L}_i^B}{\partial W} = (1-y)X\frac{1}{\left(1+e^{-W^TX}\right)}$

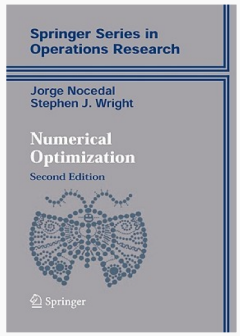
Considerations

- We still need to calculate the derivatives.
- **We need to know what is the learning rate or how to set it.**
- Local vs global minima.
- The full likelihood function includes summing up all individual '*errors'.* Unless you are a statistician, sometimes this includes hundreds of thousands of examples.

Learning Rate

Trial and Error.

There are many alternative methods which address how to set or adjust the learning rate, using the derivative or second derivatives and or the momentum. Presented in optional notes at the end of these slides.

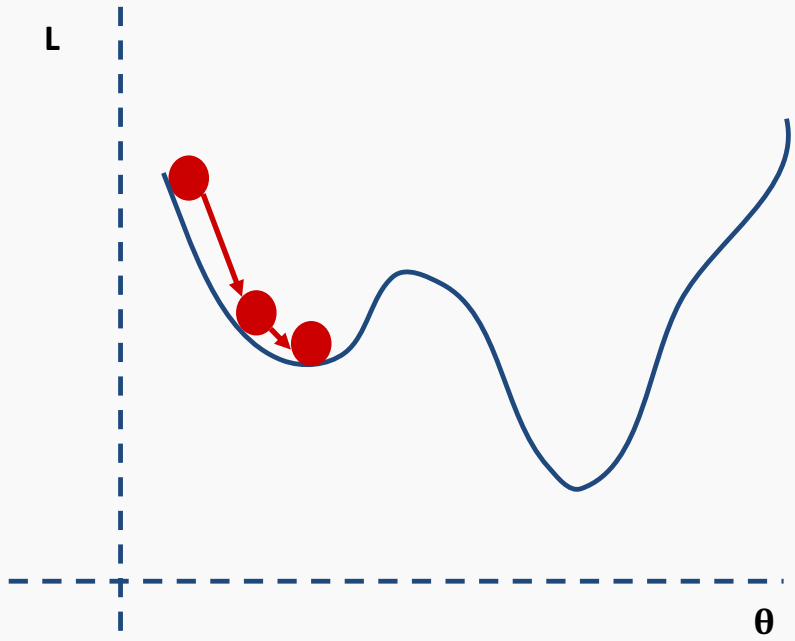


* J. Nocedal y S. Wright, “Numerical optimization”, Springer, 1999 🔗
* *TLDR*: J. Bullinaria, “Learning with Momentum, Conjugate Gradient Learning”, 2015 🔗

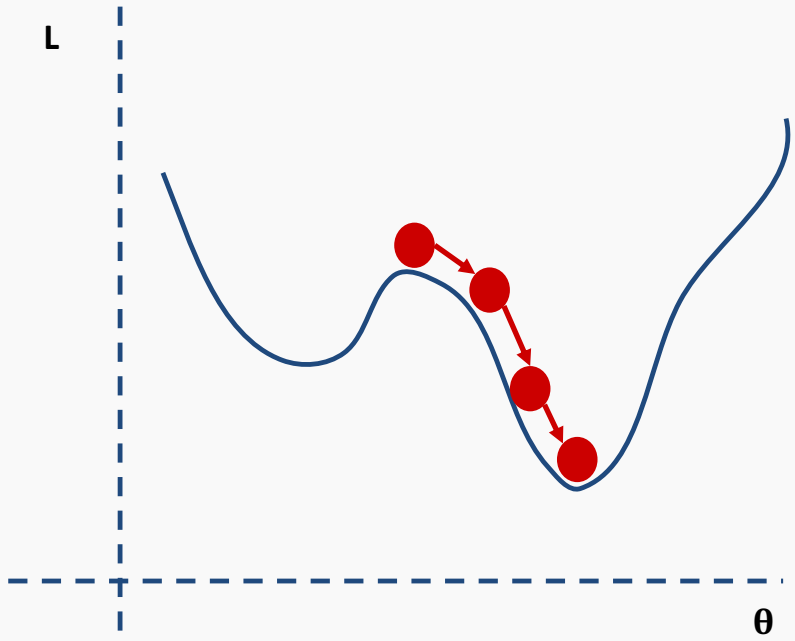
Considerations

- We still need to calculate the derivatives.
- We need to know what is the learning rate or how to set it.
- **Local vs global minima.**
- The full likelihood function includes summing up all individual '*errors'.* Unless you are a statistician, sometimes this includes hundreds of thousands of examples.

Local vs Global Minima



Local vs Global Minima



Local vs Global Minima

No guarantee that we get the global minimum.

Question: What would be a good strategy?

Considerations

- We still need to calculate the derivatives.
- We need to know what is the learning rate or how to set it.
- Local vs global minima.
- **The full likelihood function includes summing up all individual '*errors*'. Unless you are a statistician, sometimes this includes hundreds of thousands of examples.**

Batch and Stochastic Gradient Descent

$$\mathcal{L} = -\sum_i [y_i \log p_i + (1 - y_i) \log(1 - p_i)]$$

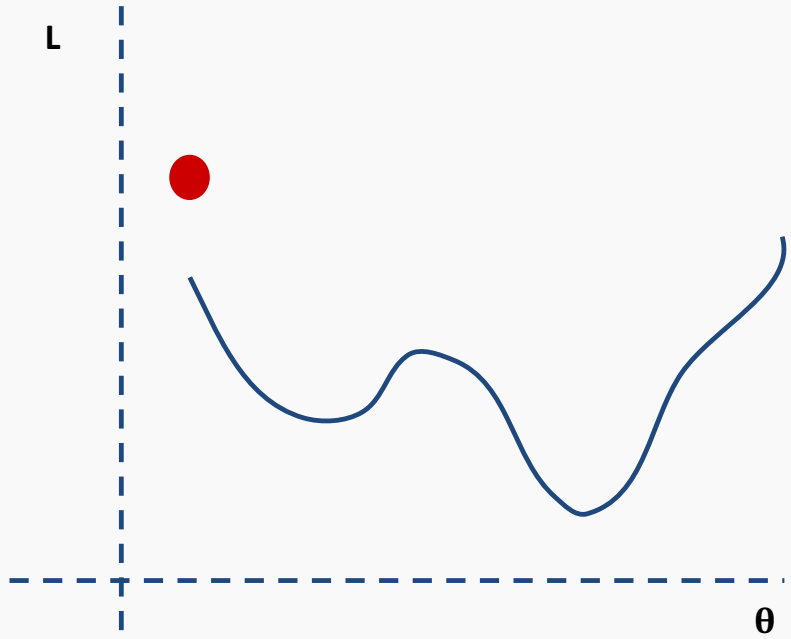
Instead of using all the observations for every step, use a subset of them (batch).

For each iteration *k*, use the following loss function to derive the derivatives:

$$\mathcal{L}^k = -\sum_{i \in b^k} [y_i \log p_i + (1 - y_i) \log(1 - p_i)]$$

which is an **approximation** to the full loss function.

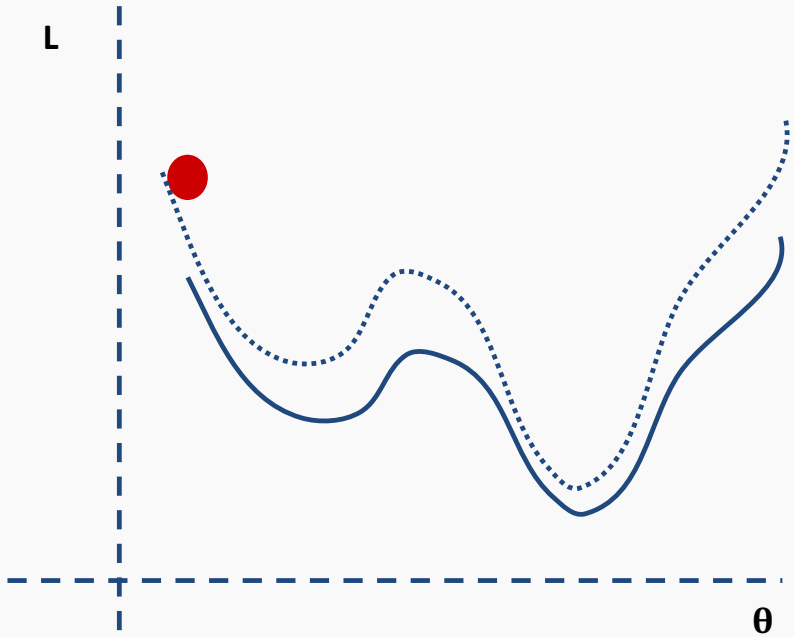
Batch and Stochastic Gradient Descent



Full Likelihood:

Batch Likelihood:

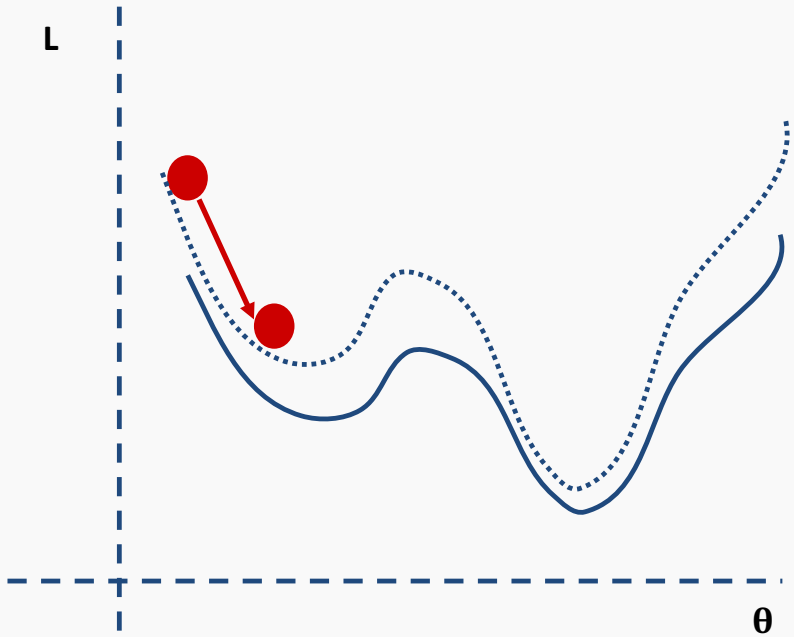
Batch and Stochastic Gradient Descent



Full Likelihood:

Batch Likelihood:

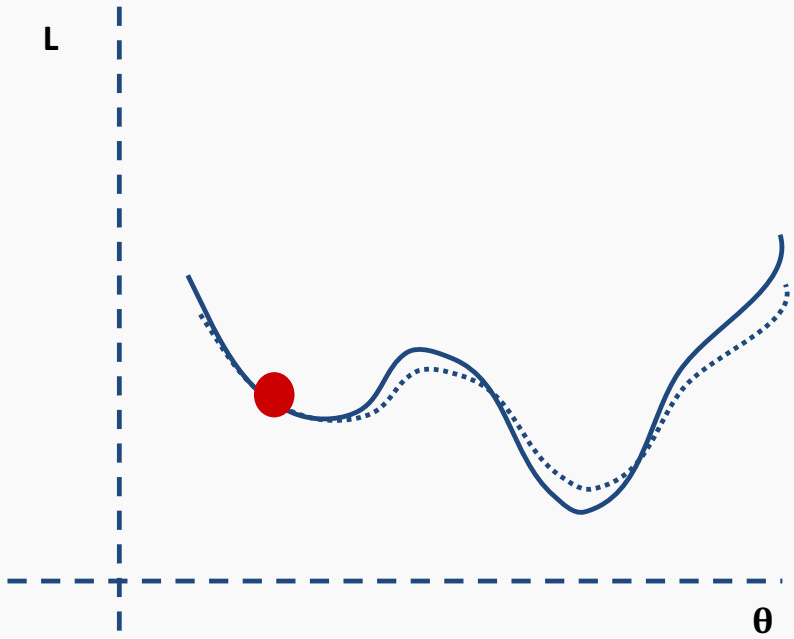
Batch and Stochastic Gradient Descent



Full Likelihood:

Batch Likelihood:

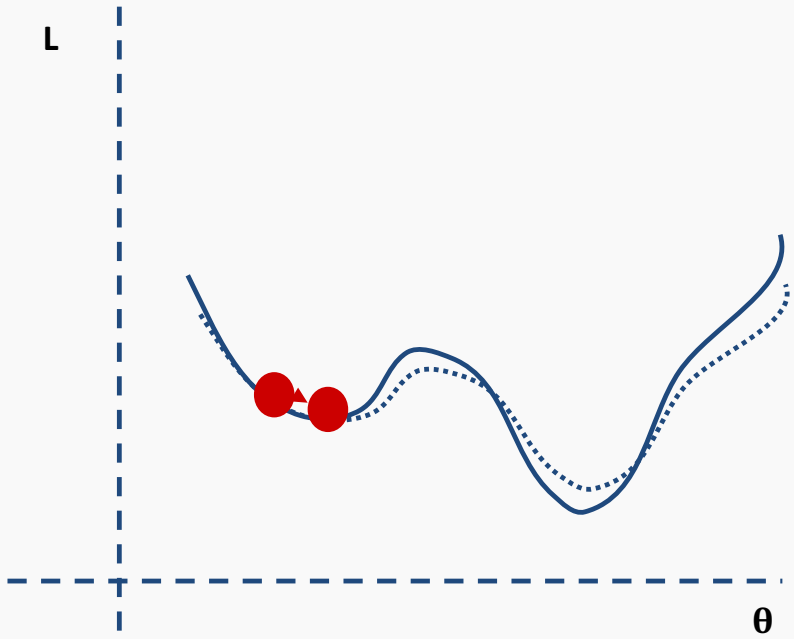
Batch and Stochastic Gradient Descent



Full Likelihood:

Batch Likelihood:

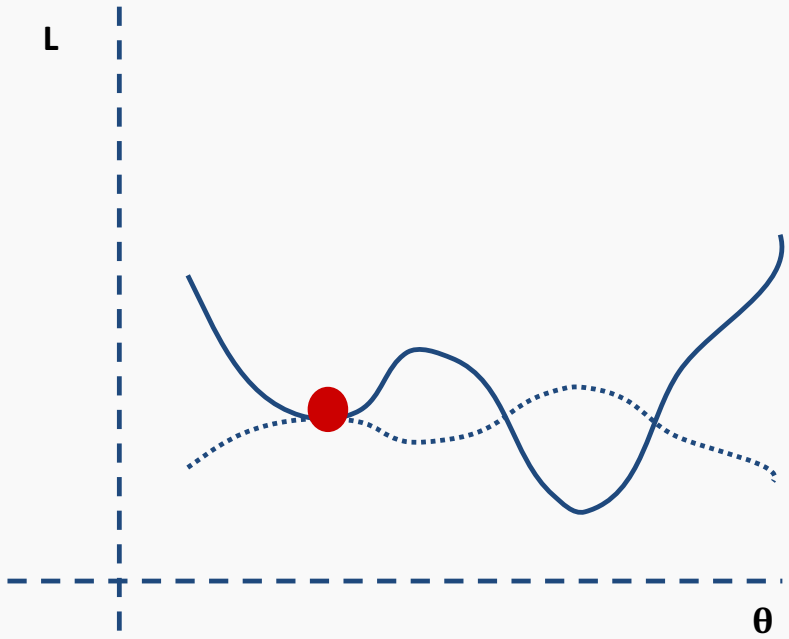
Batch and Stochastic Gradient Descent



Full Likelihood:

Batch Likelihood:

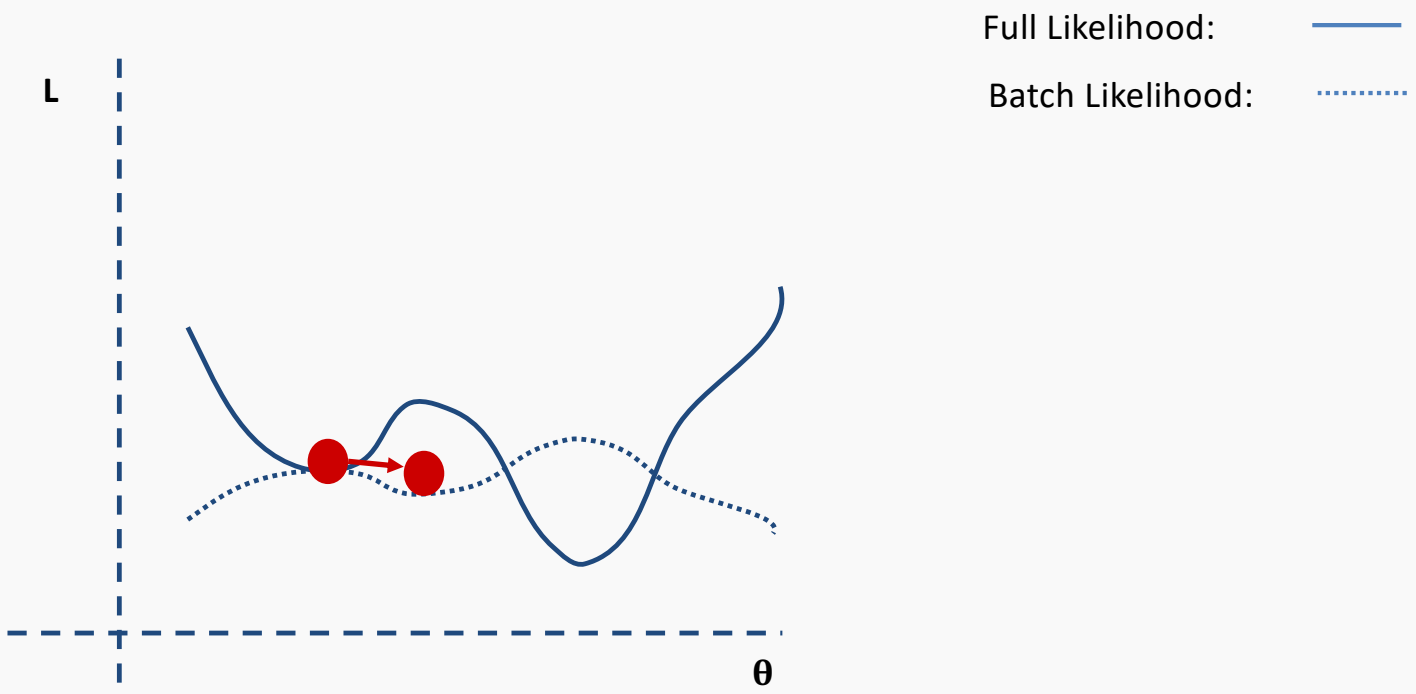
Batch and Stochastic Gradient Descent



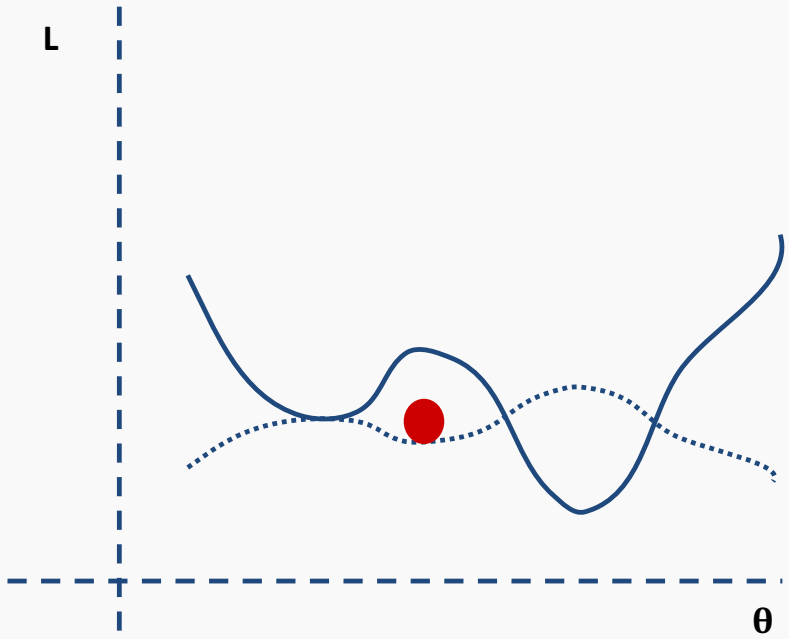
Full Likelihood:

Batch Likelihood:

Batch and Stochastic Gradient Descent



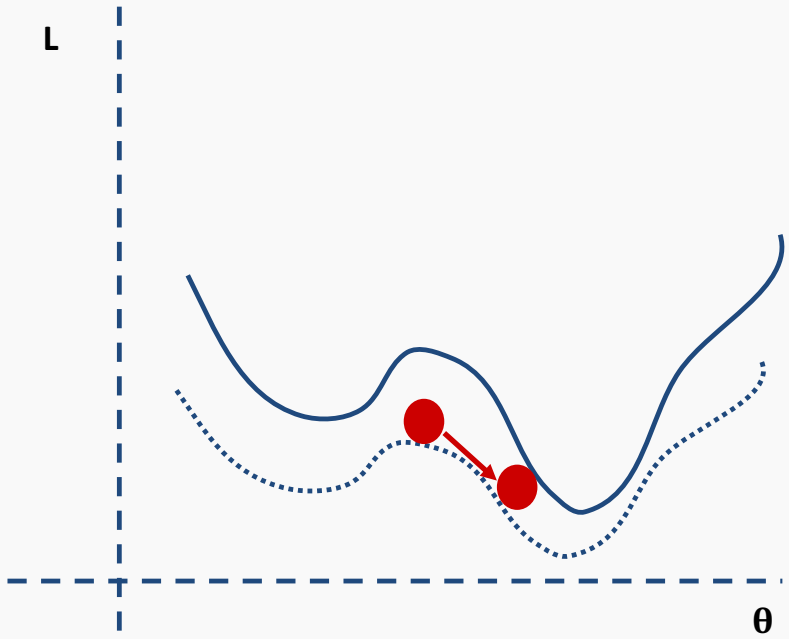
Batch and Stochastic Gradient Descent



Full Likelihood:

Batch Likelihood:

Batch and Stochastic Gradient Descent

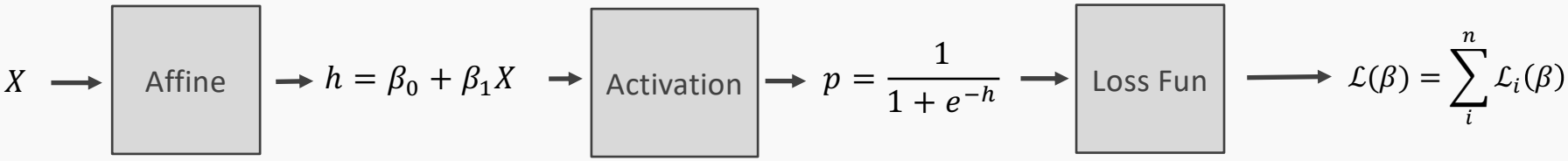


Full Likelihood:

Batch Likelihood:

Backpropagation

Backpropagation: Logistic Regression Revisited



$$\frac{\partial \mathcal{L}}{\partial p}\frac{\partial p}{\partial h}\frac{\partial h}{\partial \beta} \leftarrow \frac{\partial \mathcal{L}}{\partial p}\frac{\partial p}{\partial h} \leftarrow \frac{\partial \mathcal{L}}{\partial p}$$

$$\frac{\partial h}{\partial \beta_1} = X, \frac{d\mathcal{L}}{d\beta_0} = 1 \qquad \frac{\partial p}{\partial h} = \sigma(h)(1-\sigma(h)) \qquad \frac{\partial \mathcal{L}}{\partial p} = -y\frac{1}{p} - (1-y)\frac{1}{1-p}$$

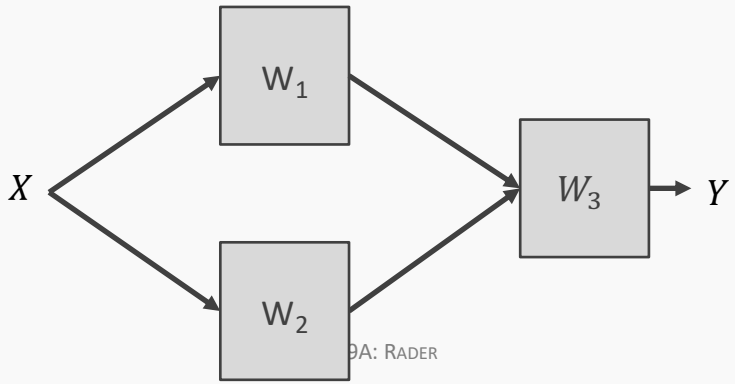
$$\frac{\partial \mathcal{L}}{\partial \beta_1} = -X\sigma(h)\big(1-\sigma(h)\big)\big[y\frac{1}{p} + (1-y)\frac{1}{1-p}\big]$$

$$\frac{\partial \mathcal{L}}{\partial \beta_0} = -\sigma(h)\big(1-\sigma(h)\big)\big[y\frac{1}{p} + (1-y)\frac{1}{1-p}\big]$$

Backpropagation

1. Derivatives need to be evaluated at some values of *X, y* and *W*.
2. But since we have an expression, we can build a function that takes as input *X, y, W* and returns the derivatives and then we can use gradient descent to update.
3. This approach works well but it does not generalize. For example if the network is changed, we need to write a new function to evaluate the derivatives.

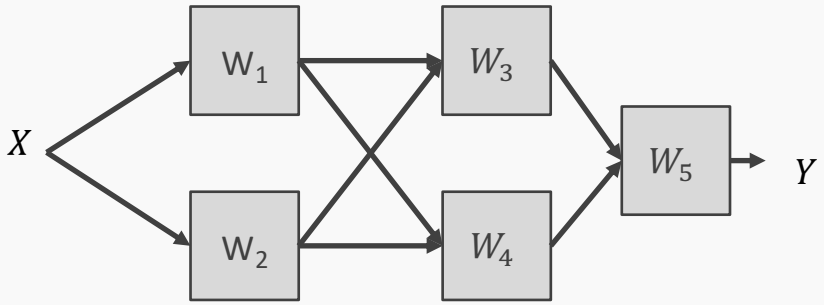
For example this network will need a different function for the derivatives



Backpropagation

1. Derivatives need to be evaluated at some values of *X, y* and *W*.
2. But since we have an expression, we can build a function that takes as input *X, y, W* and returns the derivatives and then we can use gradient descent to update.
3. This approach works well but it does not generalize. For example if the network is changed, we need to write a new function to evaluate the derivatives.

For example this network will need a different function for the derivatives



Backpropagation: a new game

Need to find a formalism to calculate the derivatives of the loss w.r.t. weights that is:

1. Flexible enough that adding a node or a layer or changing something in the network won't require to re-derive the functional form from scratch.
2. It is exact.
3. It is computationally efficient.

Hints:

1. Remember we only need to evaluate the derivatives at X_i, y_i and $W^{(k)}$.
2. We should take advantage of the chain rule we learned before

Idea 1: Evaluate the derivative at: $X=\{3\}, y=1, W=3$

Variables	derivatives	Value of the variable	Value of the partial derivative	$\frac{d\xi_n}{dW}$
$\xi_1 = -W^T X$	$\frac{\partial \xi_1}{\partial W} = -X$	-9	-3	-3
$\xi_2 = e^{\xi_1} = e^{-W^T X}$	$\frac{\partial \xi_2}{\partial \xi_1} = e^{\xi_1}$	e^{-9}	e^{-9}	$-3e^{-9}$
$\xi_3 = 1 + \xi_2 = 1 + e^{-W^T X}$	$\frac{\partial \xi_3}{\partial \xi_2} = 1$	$1+e^{-9}$	1	$-3e^{-9}$
$\xi_4 = \frac{1}{\xi_3} = \frac{1}{1 + e^{-W^T X}} = p$	$\frac{\partial \xi_4}{\partial \xi_3} = -\frac{1}{\xi_3^2}$	$\frac{1}{1 + e^{-9}}$	$\left(\frac{1}{1 + e^{-9}}\right)^2$	$-3e^{-9}\left(\frac{1}{1+e^{-9}}\right)^2$
$\xi_5 = \log \xi_4 = \log p = \log \frac{1}{1 + e^{-W^T}}$	$\frac{\partial \xi_5}{\partial \xi_4} = \frac{1}{\xi_4}$	$\log \frac{1}{1 + e^{-9}}$	$1 + e^{-9}$	$-3e^{-9}\left(\frac{1}{1+e^{-9}}\right)$
$\mathcal{L}_i^A = -y\xi_5$	$\frac{\partial \mathcal{L}}{\partial \xi_5} = -y$	$-\log \frac{1}{1 + e^{-9}}$	-1	$3e^{-9}\left(\frac{1}{1+e^{-9}}\right)$
$\frac{\partial \mathcal{L}_i^A}{\partial W} = \frac{\partial \mathcal{L}_i}{\partial \xi_5}\frac{\partial \xi_5}{\partial \xi_4}\frac{\partial \xi_4}{\partial \xi_3}\frac{\partial \xi_3}{\partial \xi_2}\frac{\partial \xi_2}{\partial \xi_1}\frac{\partial \xi_1}{\partial W}$			-3	0.00037018372

Basic functions

We still need to derive **derivatives** ☹

Variables	derivatives	Value of the variable	Value of the partial derivative	$\frac{d\xi_n}{dW}$
$\xi_1 = -W^T X$	$\frac{\partial \xi_1}{\partial W} = -X$	-9	-3	-3
$\xi_2 = e^{\xi_1} = e^{-W^T X}$	$\frac{\partial \xi_2}{d\partial \xi_1} = e^{\xi_1}$	e^{-9}	e^{-9}	$-3e^{-9}$
$\xi_3 = 1 + \xi_2 = 1 + e^{-W^T X}$	$\frac{\partial \xi_3}{\partial \xi_2} = 1$	$1+e^{-9}$	1	$-3e^{-9}$
$\xi_4 = \frac{1}{\xi_3} = \frac{1}{1+e^{-W^T X}} = p$	$\frac{\partial \xi_4}{\partial \xi_3} = -\frac{1}{\xi_3^2}$	$\frac{1}{1+e^{-9}}$	$\left(\frac{1}{1+e^{-9}}\right)^2$	$-3e^{-9}\left(\frac{1}{1+e^{-9}}\right)^2$
$\xi_5 = \log \xi_4 = \log p = \log \frac{1}{1+e^{-W^T X}}$	$\frac{\partial \xi_5}{\partial \xi_4} = \frac{1}{\xi_4}$	$\log \frac{1}{1+e^{-9}}$	$1+e^{-9}$	$-3e^{-9}\left(\frac{1}{1+e^{-9}}\right)$
$\mathcal{L}_i^A = -y\xi_5$	$\frac{\partial \mathcal{L}}{\partial \xi_5} = -y$	$-\log \frac{1}{1+e^{-9}}$	-1	$3e^{-9}\left(\frac{1}{1+e^{-9}}\right)$
$\frac{\partial \mathcal{L}_i^A}{\partial W} = \frac{\partial \mathcal{L}_i}{\partial \xi_5}\frac{\partial \xi_5}{\partial \xi_4}\frac{\partial \xi_4}{\partial \xi_3}\frac{\partial \xi_3}{\partial \xi_2}\frac{\partial \xi_2}{\partial \xi_1}\frac{\partial \xi_1}{\partial W}$			-3	0.00037018372

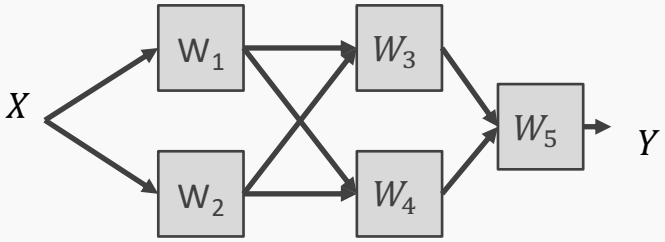
Basic functions

Notice though those are basic functions that my grandparent can do

$\xi_0 = X$	$\frac{\partial \xi_0}{\partial X} = 1$	`def x0(x):` `  return X`	`def derx0():` `  return 1`
$\xi_1 = -W^T \xi_0$	$\frac{\partial \xi_1}{\partial W} = -X$	`def x1(a,x):` `  return -a*X`	`def derx1(a,x):` `  return -a`
$\xi_2 = \mathrm{e}^{\xi_1}$	$\frac{\partial \xi_2}{\partial \xi_1} = e^{\xi_1}$	`def x2(x):` `  return np.exp(x)`	`def derx2(x):` `  return np.exp(x)`
$\xi_3 = 1 + \xi_2$	$\frac{\partial \xi_3}{\partial \xi_2} = 1$	`def x3(x):` `  return 1+x`	`def derx3(x):` `  return 1`
$\xi_4 = \frac{1}{\xi_3}$	$\frac{\partial \xi_4}{\partial \xi_3} = -\frac{1}{\xi_3^2}$	`def der1(x):` `  return 1/(x)`	`def derx4(x):` `  return  -(1/x)**(2)`
$\xi_5 = \log \xi_4$	$\frac{\partial \xi_5}{\partial \xi_4} = \frac{1}{\xi_4}$	`def der1(x):` `  return np.log(x)`	`def derx5(x)` `  return 1/x`
$\mathcal{L}_i^A = -y\xi_5$	$\frac{\partial \mathcal{L}}{\partial \xi_5} = -y$	`def der1(y,x):` `  return -y*x`	`def derL(y):` `  return -y`

Putting it altogether

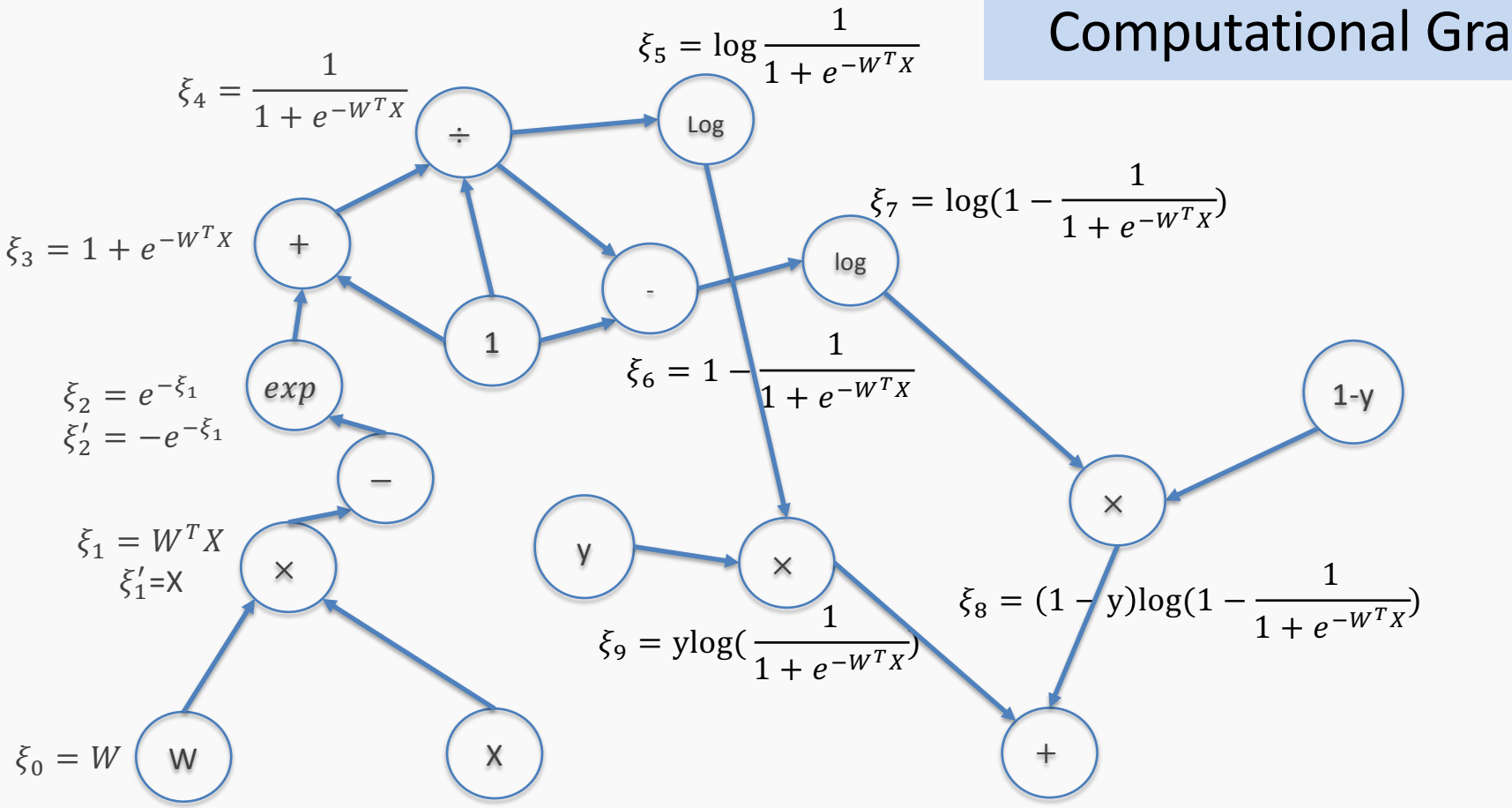
1. We specify the network structure



2. Following the computational graph …

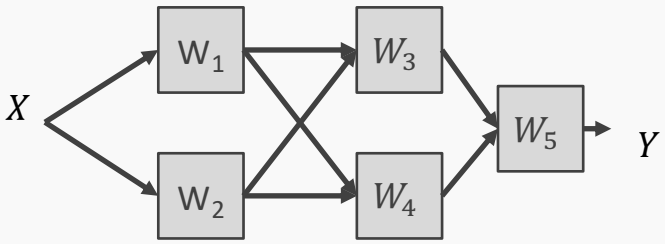
What is computational graph?

Computational Graph



Putting it altogether

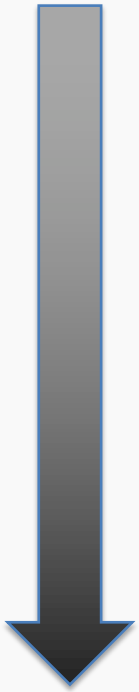
1. We specify the network structure



- Envision the computational graph (only needed for the reverse mode).
- At each node of the graph we build two functions: the evaluation of the variable and its partial derivative with respect to the previous variable (as shown in the table 3 slides back)
- Now we can either go forward or backward depending on the situation. In general, forward is easier to implement and to understand. The difference is clearer when there are multiple nodes per layer.

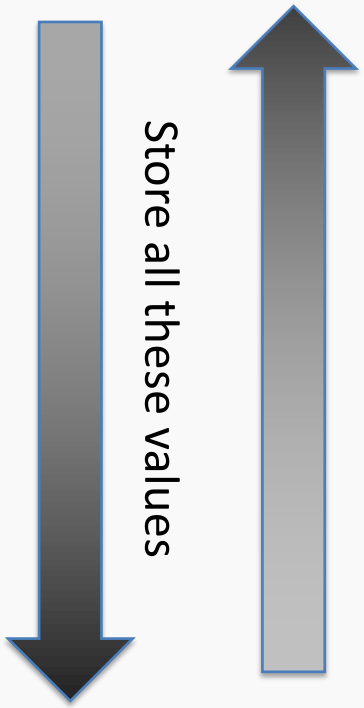
Forward mode: Evaluate the derivative at: $X=\{3\}, y=1, W=3$

Variables	derivatives	Value of the variable	Value of the partial derivative	$\frac{d\mathcal{L}}{d\xi_n}$
$\xi_1 = -W^T X$	$\frac{\partial \xi_1}{\partial W} = -X$	-9	-3	-3
$\xi_2 = e^{\xi_1} = e^{-W^T X}$	$\frac{\partial \xi_2}{\partial \xi_1} = e^{\xi_1}$	e^{-9}	e^{-9}	$-3e^{-9}$
$\xi_3 = 1 + \xi_2 = 1 + e^{-W^T X}$	$\frac{\partial \xi_3}{\partial \xi_2} = 1$	$1+e^{-9}$	1	$-3e^{-9}$
$\xi_4 = \frac{1}{\xi_3} = \frac{1}{1+e^{-W^T X}} = p$	$\frac{\partial \xi_4}{\partial \xi_3} = -\frac{1}{\xi_3^2}$	$\frac{1}{1+e^{-9}}$	$\left(\frac{1}{1+e^{-9}}\right)^2$	$-3e^{-9}\left(\frac{1}{1+e^{-9}}\right)^2$
$\xi_5 = \log \xi_4 = \log p = \log \frac{1}{1+e^{-W^T X}}$	$\frac{\partial \xi_5}{\partial \xi_4} = \frac{1}{\xi_4}$	$\log \frac{1}{1+e^{-9}}$	$1+e^{-9}$	$-3e^{-9}\left(\frac{1}{1+e^{-9}}\right)$
$\mathcal{L}_i^A = -y\xi_5$	$\frac{\partial \mathcal{L}}{\partial \xi_5} = -y$	$-\log \frac{1}{1+e^{-9}}$	-1	$3e^{-9}\left(\frac{1}{1+e^{-9}}\right)$
$\frac{\partial \mathcal{L}_i^A}{\partial W} = \frac{\partial \mathcal{L}_i}{\partial \xi_5}\frac{\partial \xi_5}{\partial \xi_4}\frac{\partial \xi_4}{\partial \xi_3}\frac{\partial \xi_3}{\partial \xi_2}\frac{\partial \xi_2}{\partial \xi_1}\frac{\partial \xi_1}{\partial W}$			-3	0.00037018372



Backward mode: Evaluate the derivative at: *X={3}, y=1, W=3*

Variables	derivatives	Value of the variable	Value of the partial derivative
$\xi_1 = -W^T X$	$\frac{\partial \xi_1}{\partial W} = -X$	-9	-3
$\xi_2 = e^{\xi_1} = e^{-W^T X}$	$\frac{\partial \xi_2}{\partial \xi_1} = e^{\xi_1}$	e^{-9}	e^{-9}
$\xi_3 = 1 + \xi_2 = 1 + e^{-W^T X}$	$\frac{\partial \xi_3}{\partial \xi_2} = 1$	$1+e^{-9}$	1
$\xi_4 = \frac{1}{\xi_3} = \frac{1}{1 + e^{-W^T X}} = p$	$\frac{\partial \xi_4}{\partial \xi_3} = -\frac{1}{\xi_3^2}$	$\frac{1}{1 + e^{-9}}$	$\left(\frac{1}{1 + e^{-9}}\right)^2$
$\xi_5 = \log \xi_4 = \log p = \log \frac{1}{1 + e^{-W^T X}}$	$\frac{\partial \xi_5}{\partial \xi_4} = \frac{1}{\xi_4}$	$\log \frac{1}{1 + e^{-9}}$	$1 + e^{-9}$
$\mathcal{L}_i^A = -y\xi_5$	$\frac{\partial \mathcal{L}}{\partial \xi_5} = -y$	$-\log \frac{1}{1 + e^{-9}}$	-1
$\frac{\partial \mathcal{L}_i^A}{\partial W} = \frac{\partial \mathcal{L}_i}{\partial \xi_5}\frac{\partial \xi_5}{\partial \xi_4}\frac{\partial \xi_4}{\partial \xi_3}\frac{\partial \xi_3}{\partial \xi_2}\frac{\partial \xi_2}{\partial \xi_1}\frac{\partial \xi_1}{\partial W}$			Type equation here.



Additional Material Outline

Optimization

- **Challenges in Optimization**
- Momentum
- Adaptive Learning Rate
- Parameter Initialization
- Batch Normalization

Regularization of NNs

- Norm Penalties
- Early Stopping
- Data Augmentation
- Sparse Representation
- Dropout

Learning vs. Optimization

Goal of learning: minimize generalization error, or the loss function

$$\mathcal{L}(\boldsymbol{W}) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim \boldsymbol{p}_{\boldsymbol{data}}}\left[L(f\left(x, W\right), y\right]$$

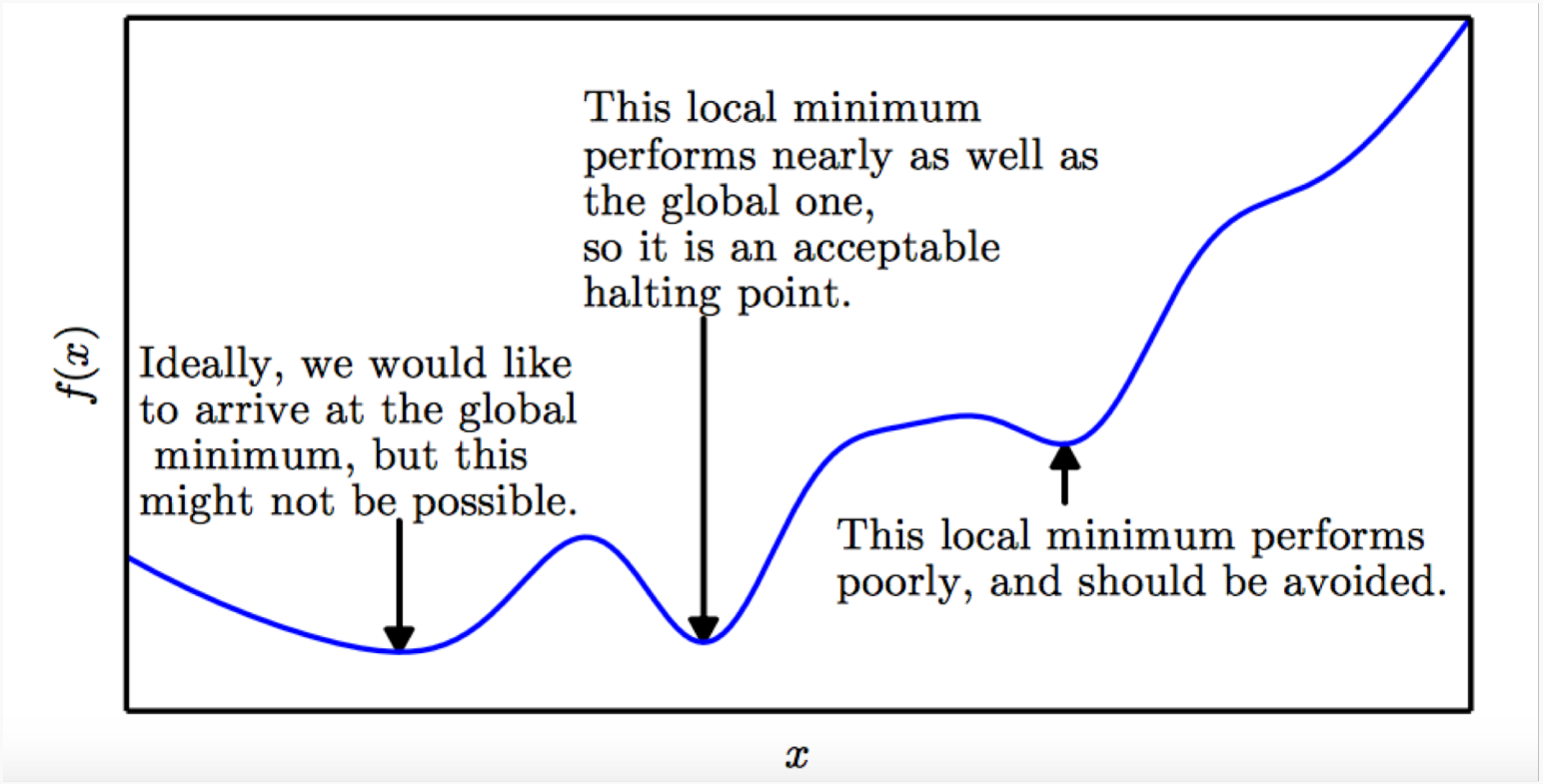
f is the neural network

In practice, empirical risk minimization:

$$\mathcal{L}(W) = \sum_i \left[L(f\left(x_i; W\right), y_i\right]$$

Quantity optimized different from the quantity we care about

Local Minima

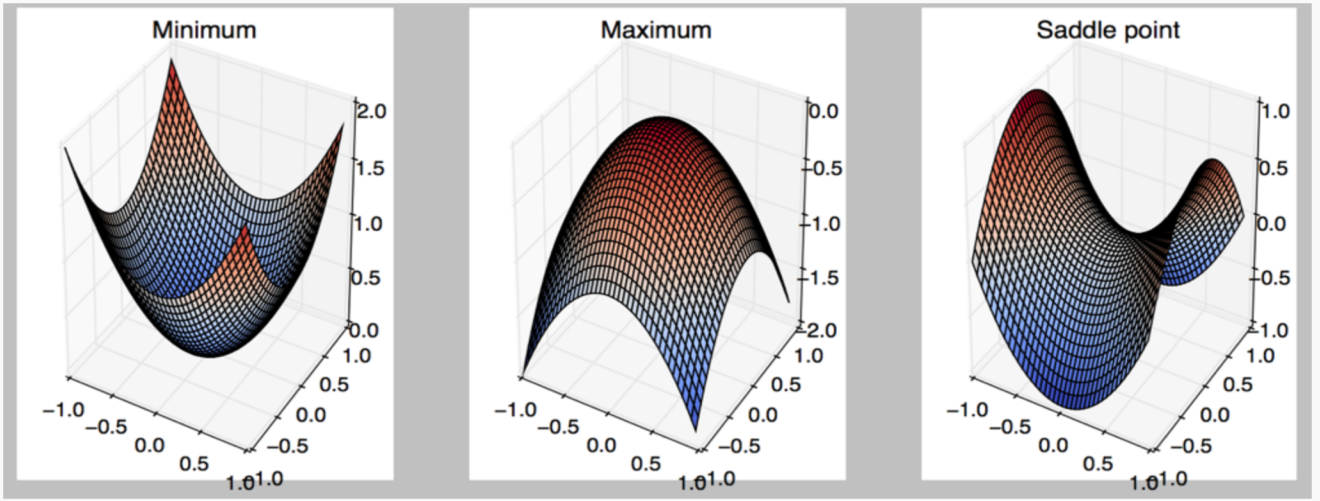


Goodfellow et al. (2016)

Critical Points

Points with zero gradient

2nd-derivate (Hessian) determines curvature



Goodfellow et al. (2016)

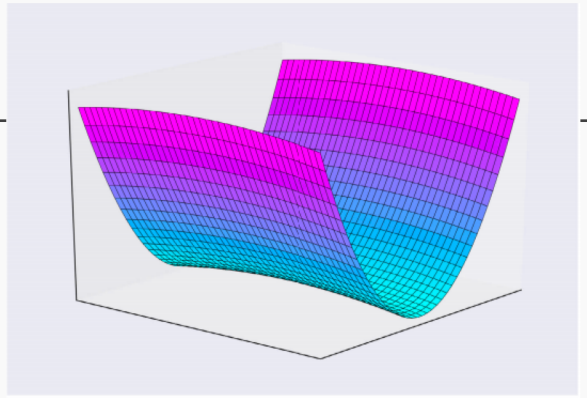
Local Minima

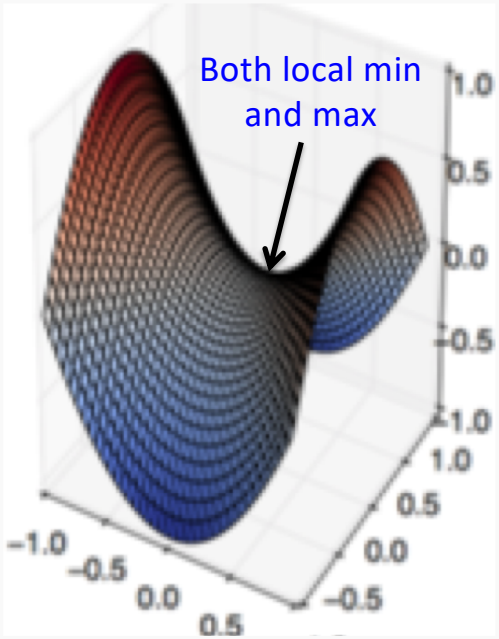
Old view: local minima is major problem in neural network training

Recent view:

- For sufficiently large neural networks, most local minima incur low cost
- Not important to find true global minimum

Saddle Points





Recent studies indicate that in high dim, saddle points are more likely than local min

Gradient can be very small near saddle points

Goodfellow et al. (2016)

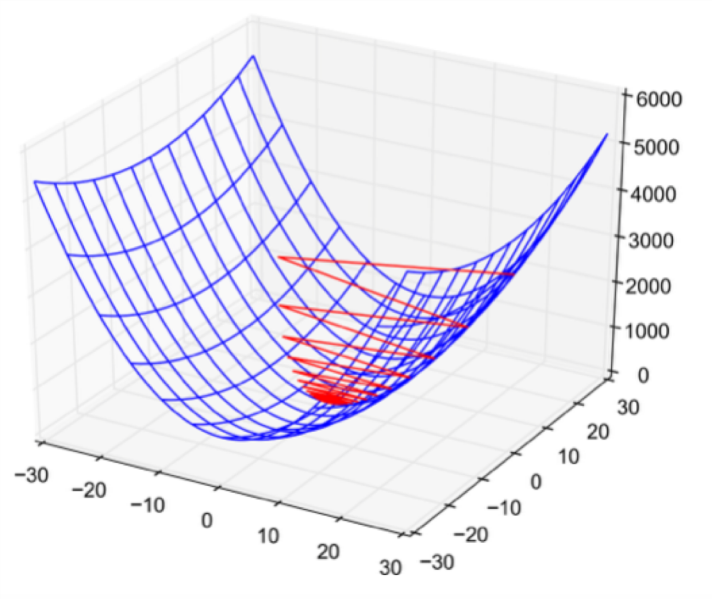
Poor Conditioning

Poorly conditioned Hessian matrix

- High curvature: small steps leads to huge increase

Learning is slow despite strong gradients

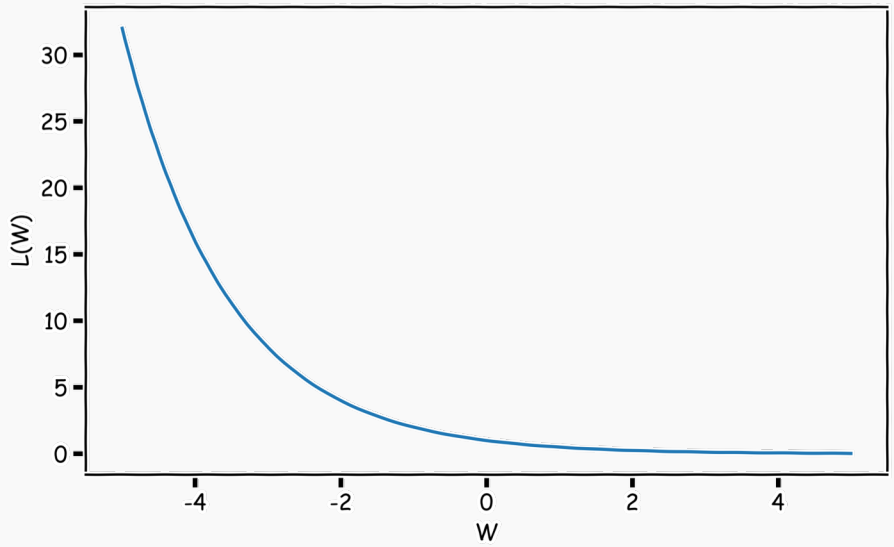
Oscillations slow down progress



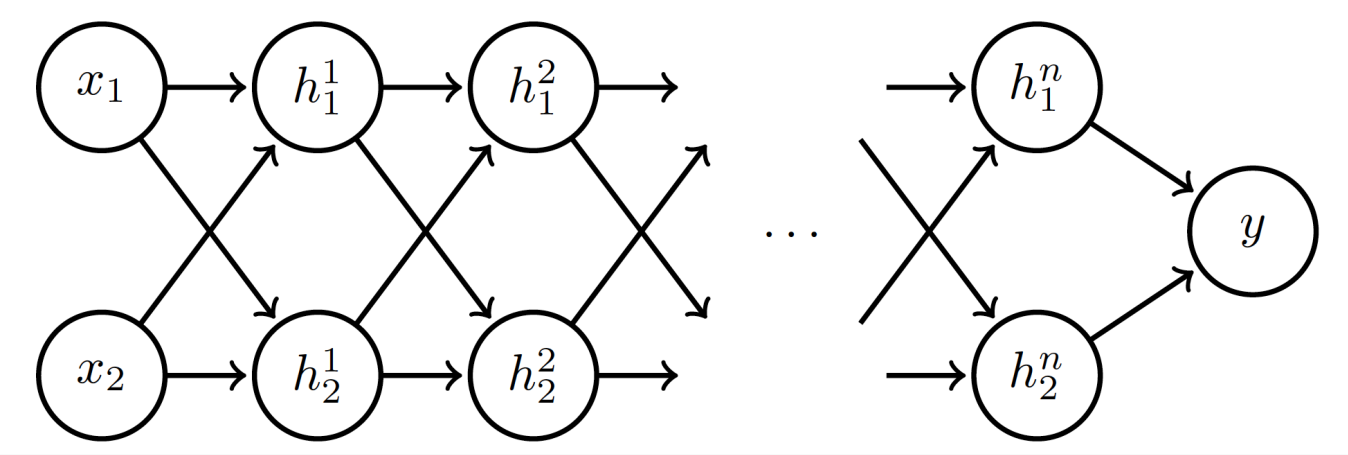
No Critical Points

Some cost functions do not have critical points. In particular classification.

WHY?



Exploding and Vanishing Gradients



Linear activation $h_i = Wx$

$h_i = Wh_{i-1}, \qquad i = 2, \ldots, n$

Exploding and Vanishing Gradients

$$\text{Suppose } \mathbf{W} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}:$$

$$\begin{bmatrix} h_1^1 \\ h_2^1 \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad \cdots \quad \begin{bmatrix} h_1^n \\ h_2^n \end{bmatrix} = \begin{bmatrix} a^n & 0 \\ 0 & b^n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Exploding and Vanishing Gradients

Suppose $x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

Case 1: $a = 1,\ b = 2$:

$$y \to 1, \quad \nabla y \to \begin{bmatrix} n \\ n2^{n-1} \end{bmatrix}$$

Explodes!

Case 2: $a = 0.5,\ b = 0.9$:

$$y \to 0, \quad \nabla y \to \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Vanishes!

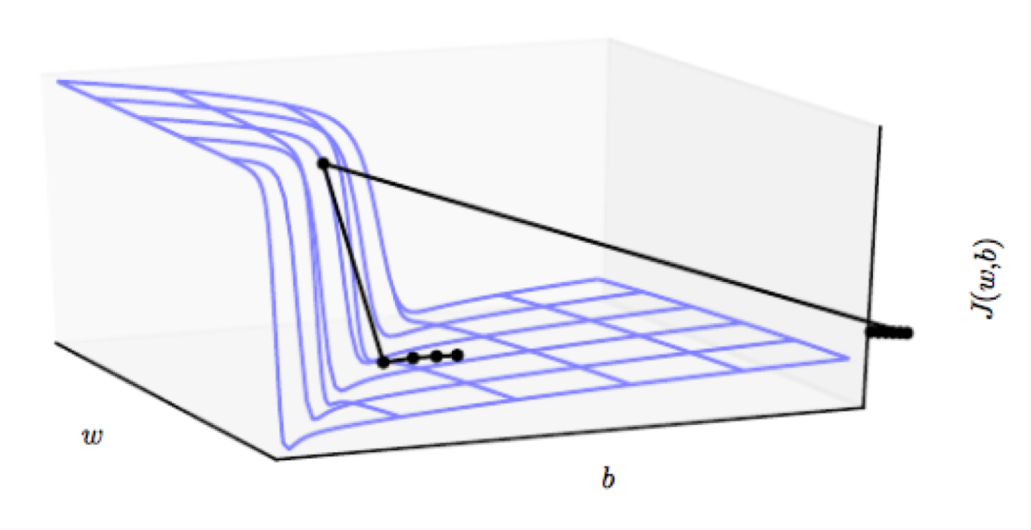
Exploding and Vanishing Gradients

Exploding gradients lead to cliffs

Can be mitigated using gradient clipping

$$\text{if } \|g\| > u$$

$$g \leftarrow \frac{gu}{\|g\|}$$



Goodfellow et al. (2016)

Outline

Optimization

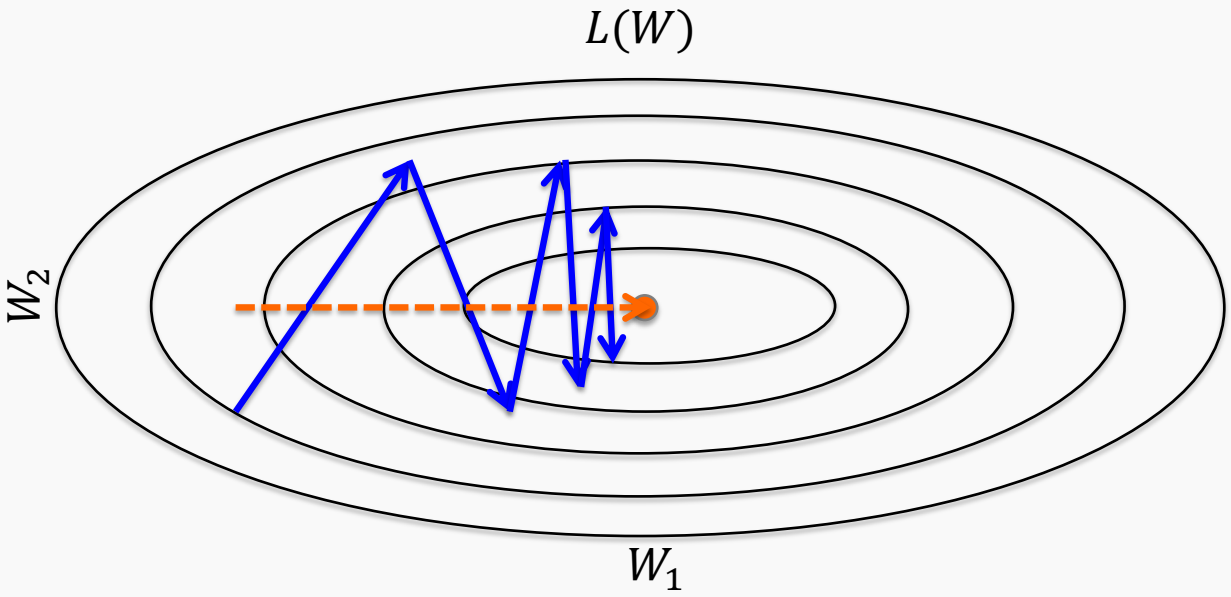
- Challenges in Optimization
- **Momentum**
- Adaptive Learning Rate
- Parameter Initialization
- Batch Normalization

Regularization of NN

- Norm Penalties
- Early Stopping
- Data Augmentation
- Sparse Representation
- Dropout

Momentum

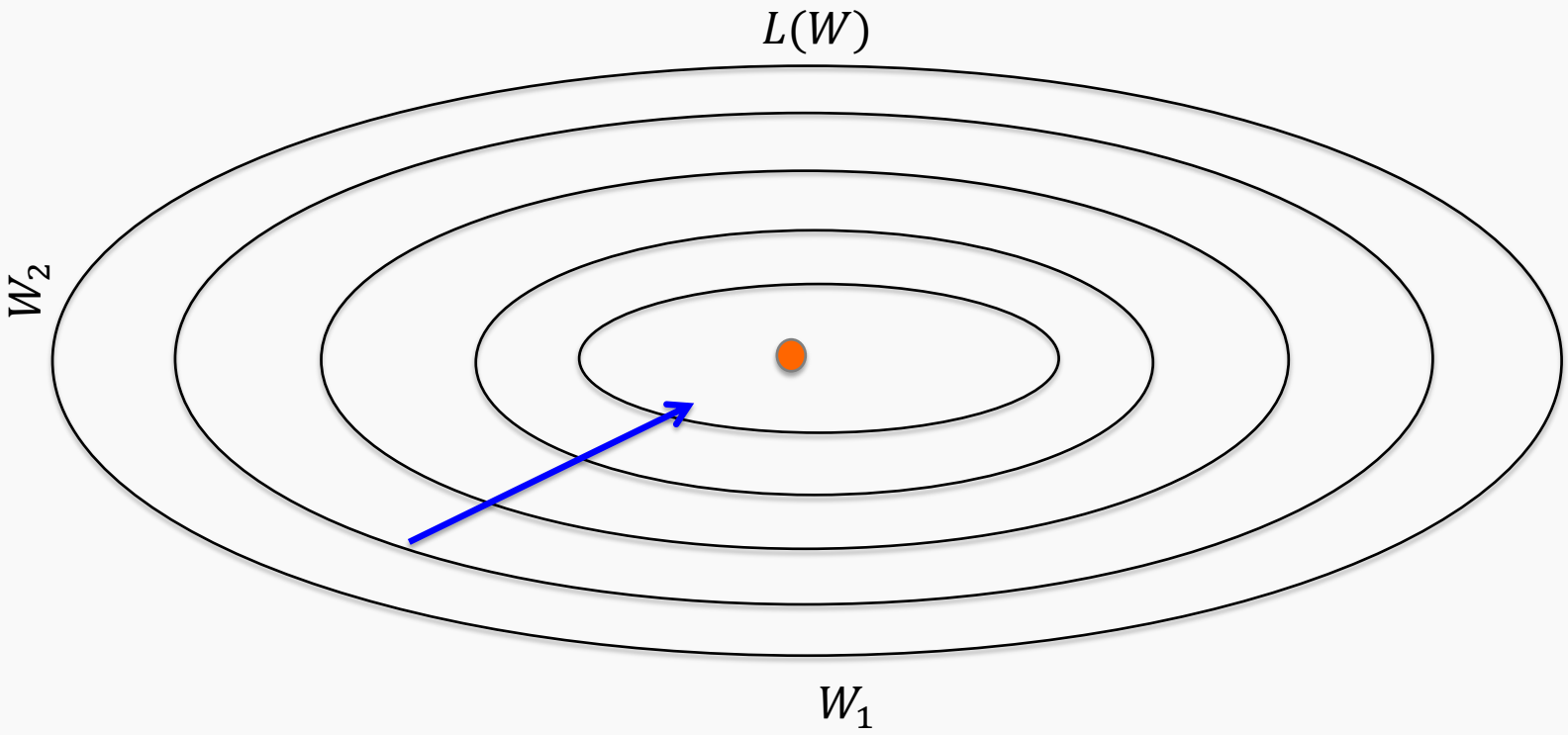
Oscillations because updates do not exploit curvature information



Average gradient presents faster path to optimal: vertical components cancel out

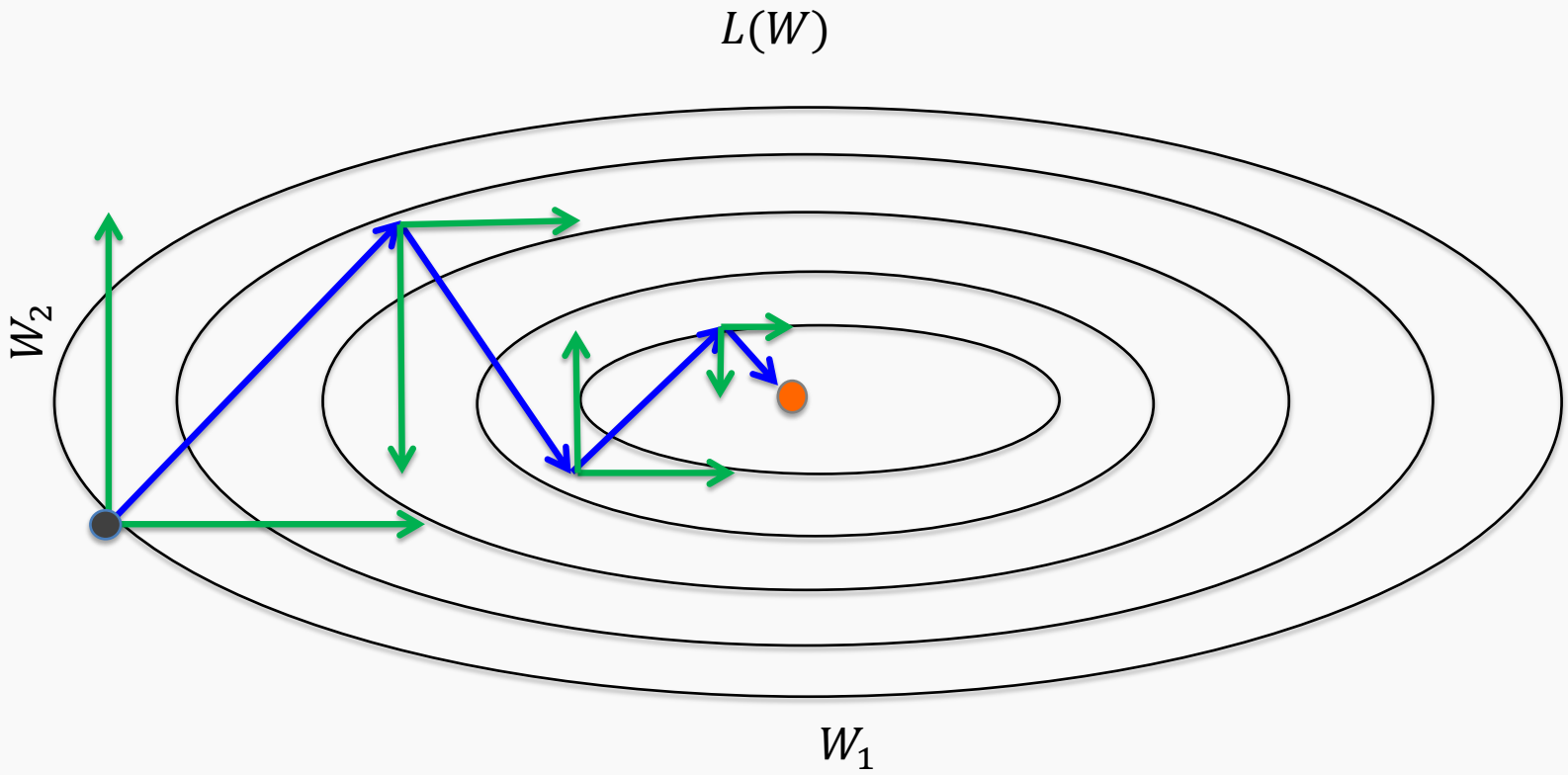
Momentum

Question: Why not this?



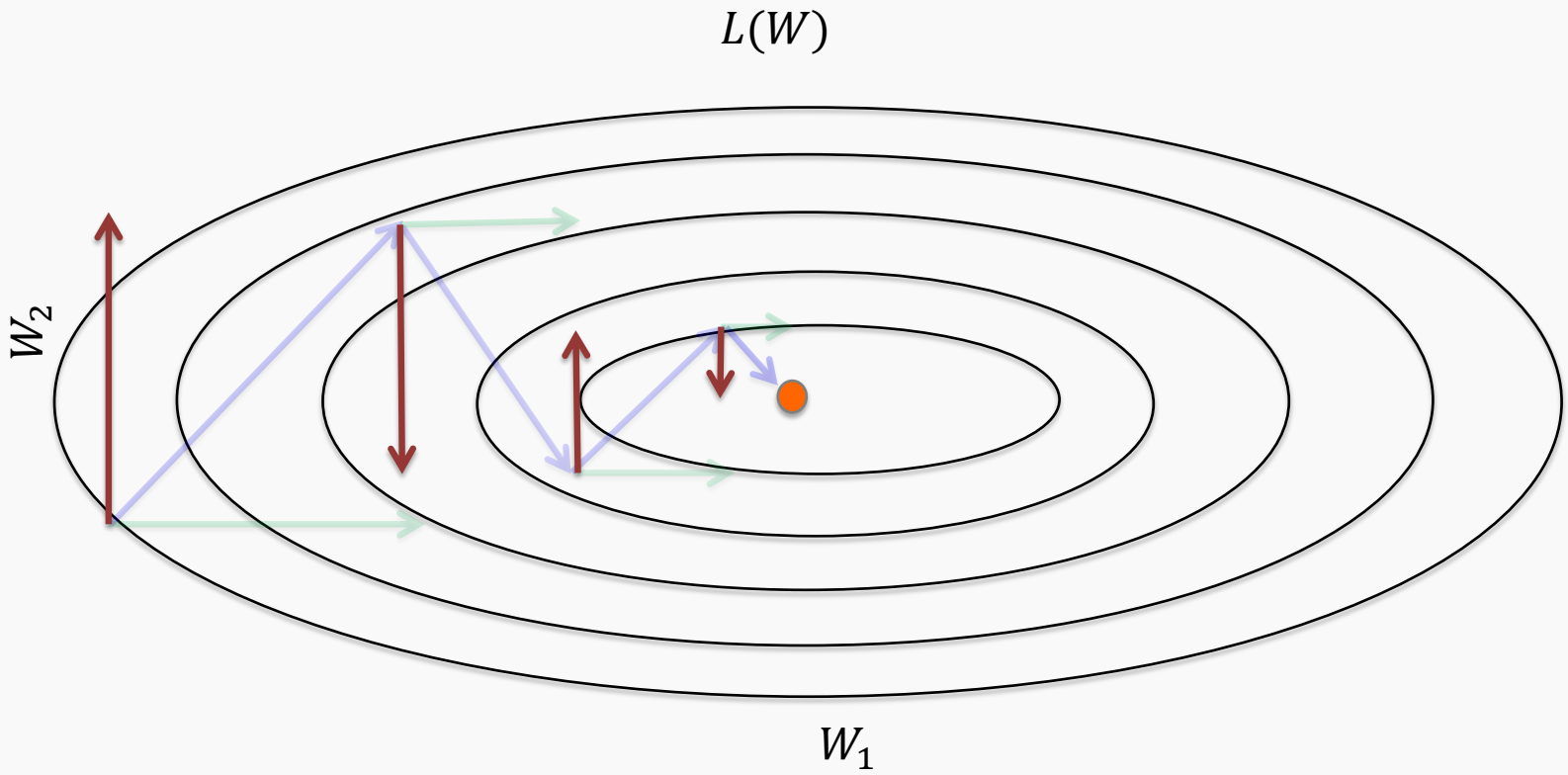
Momentum

Let us figure out an algorithm which will lead us to the minimum faster.



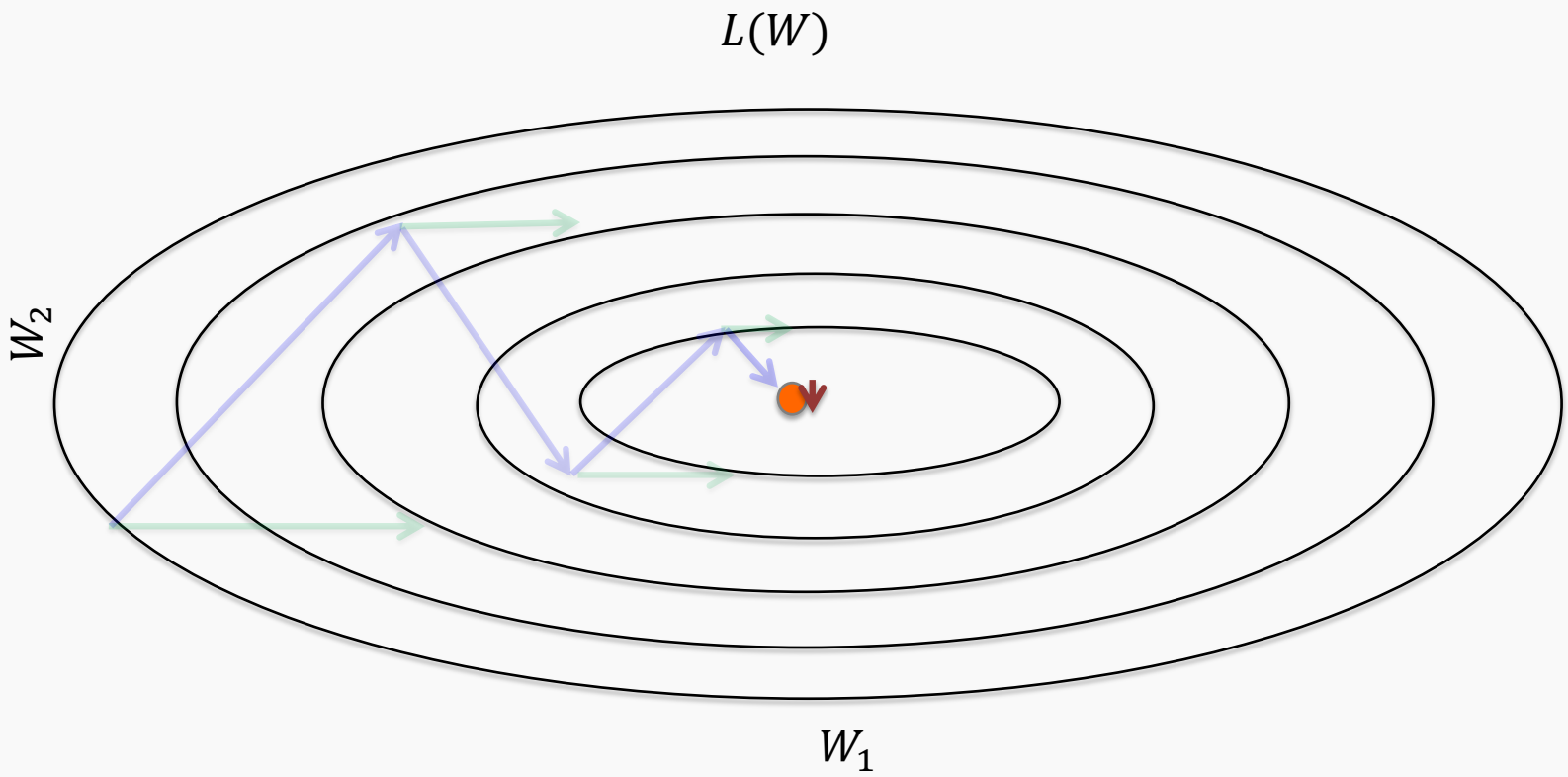
Momentum

Look each component at a time



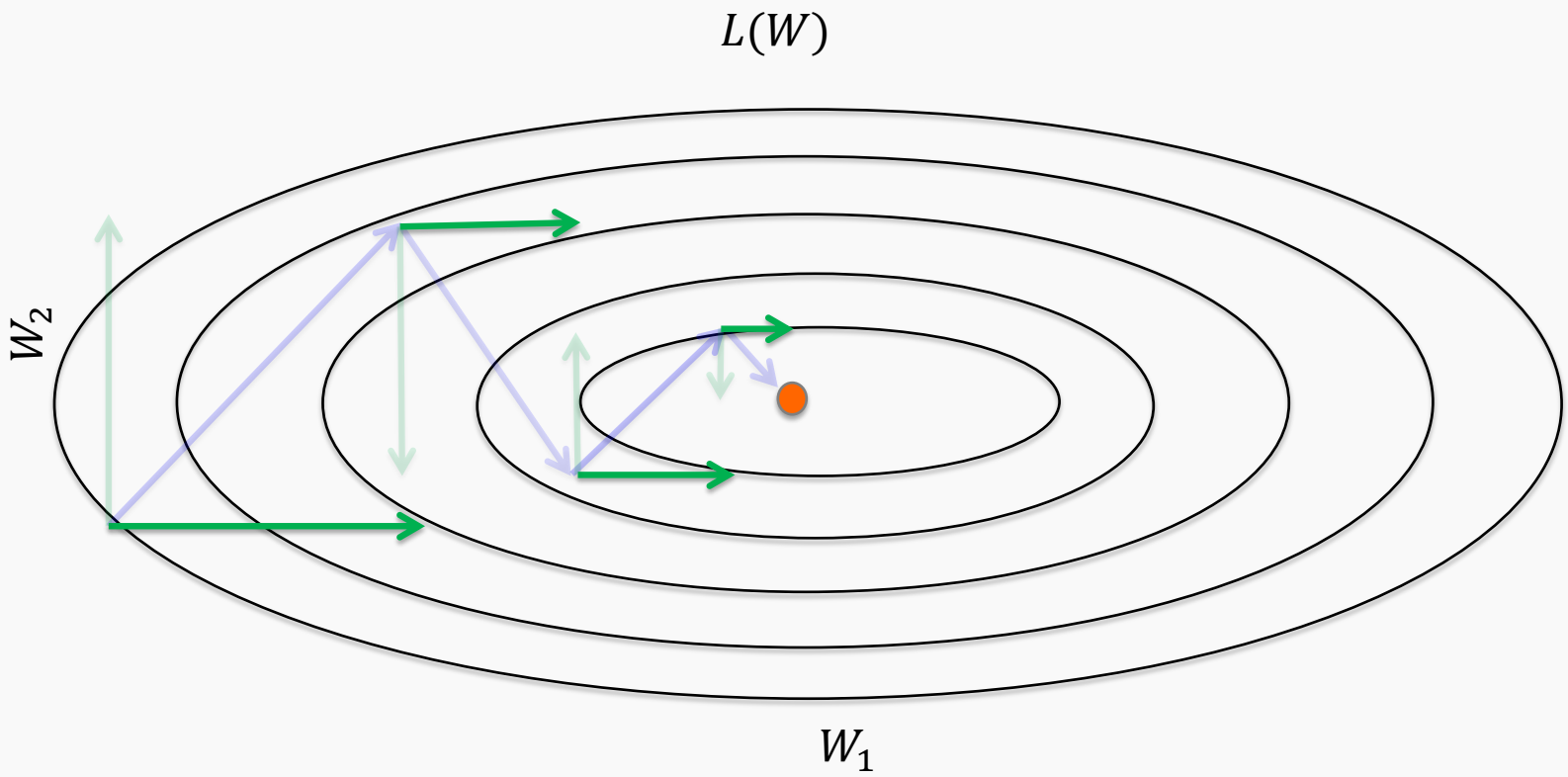
Momentum

Let us figure out an algorithm



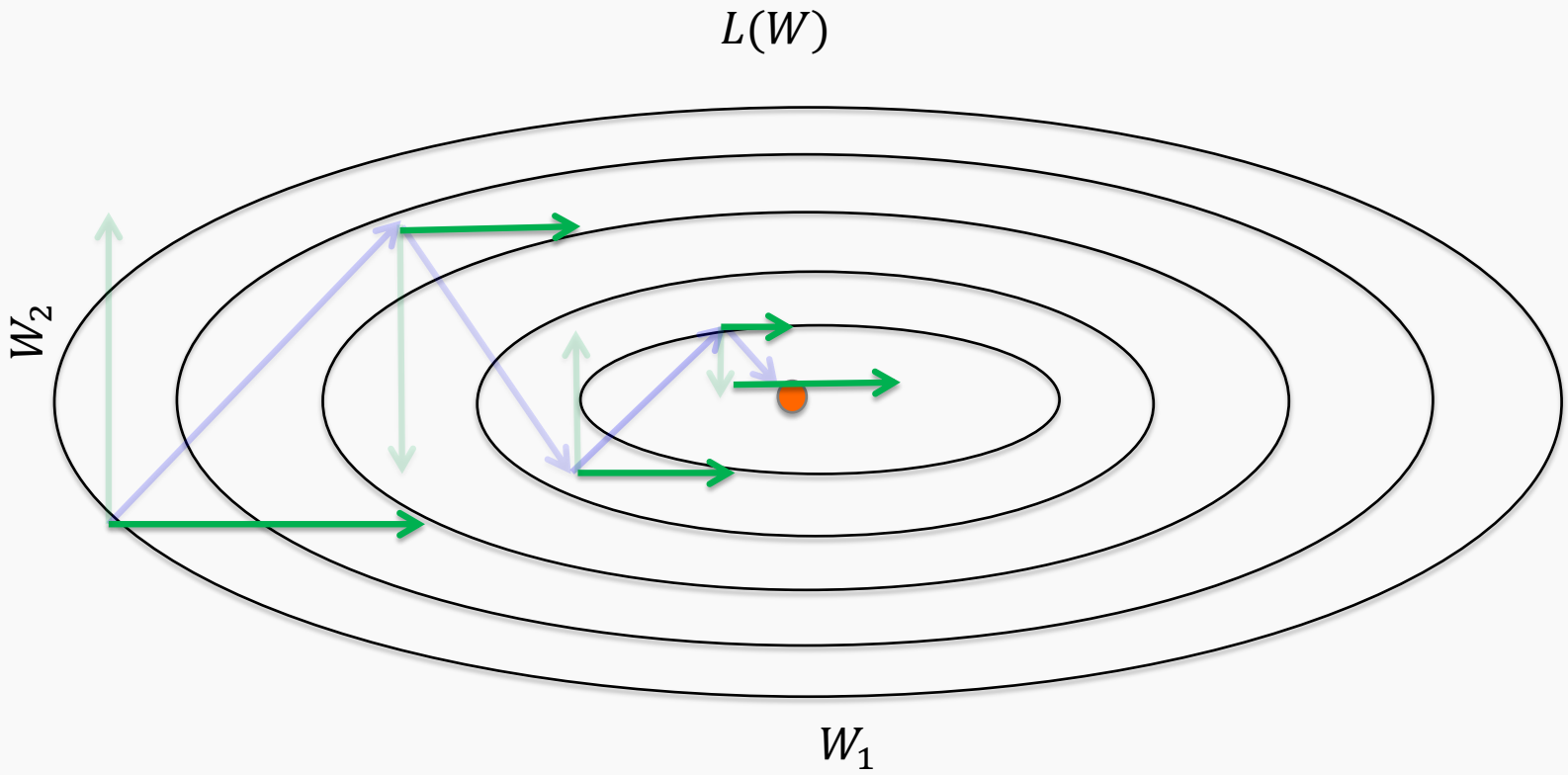
Momentum

Let us figure out an algorithm



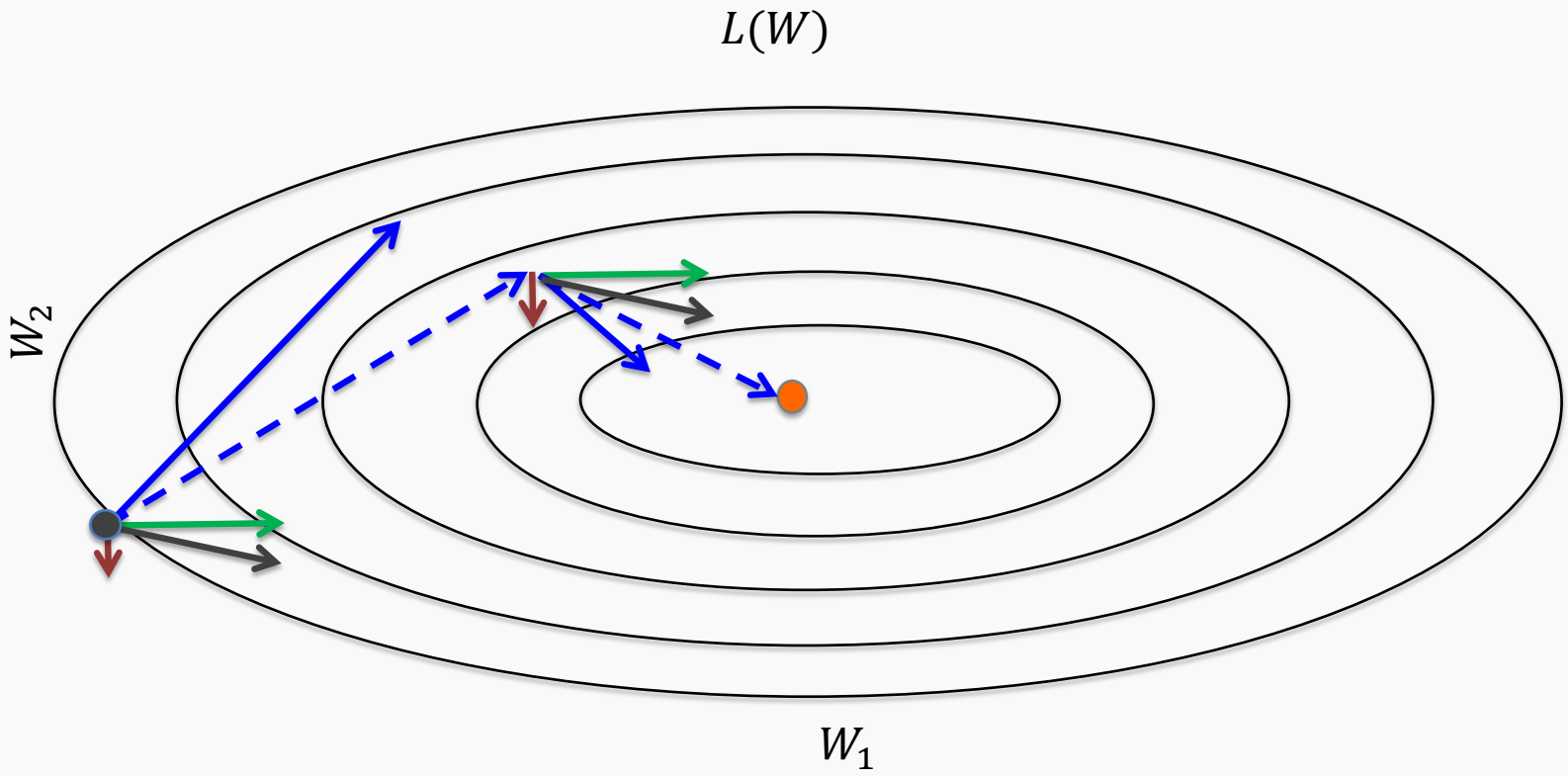
Momentum

Let us figure out an algorithm



Momentum

Let us figure out an algorithm



Momentum

f is the Neural Network

Old gradient descent:

$$g = \frac{1}{m}\sum_i \nabla_W L(f(x_i; W), y_i) \qquad W^* = W - \lambda g$$

$$g = g + average\ g\ from\ before$$

New gradient descent with momentum:

$$v = \alpha v + (1 - \alpha)\, g \qquad W^* = W - \lambda v$$

$\alpha \in [0,1)$ controls how quickly effect of past gradients decay

Nesterov Momentum

Apply an interim update:

$$\widetilde{W} = W + v$$

Perform a correction based on gradient at the interim point:

$$g = \frac{1}{m}\sum_i \nabla_W L(f(x_i; \widetilde{W}), y_i)$$

$$v = \alpha v - \varepsilon g$$

$$W = W + v$$

Momentum based on look-ahead slope

LiveSlides web content

To view

Download the add-in.

liveslides.com/download

Start the presentation.

Outline

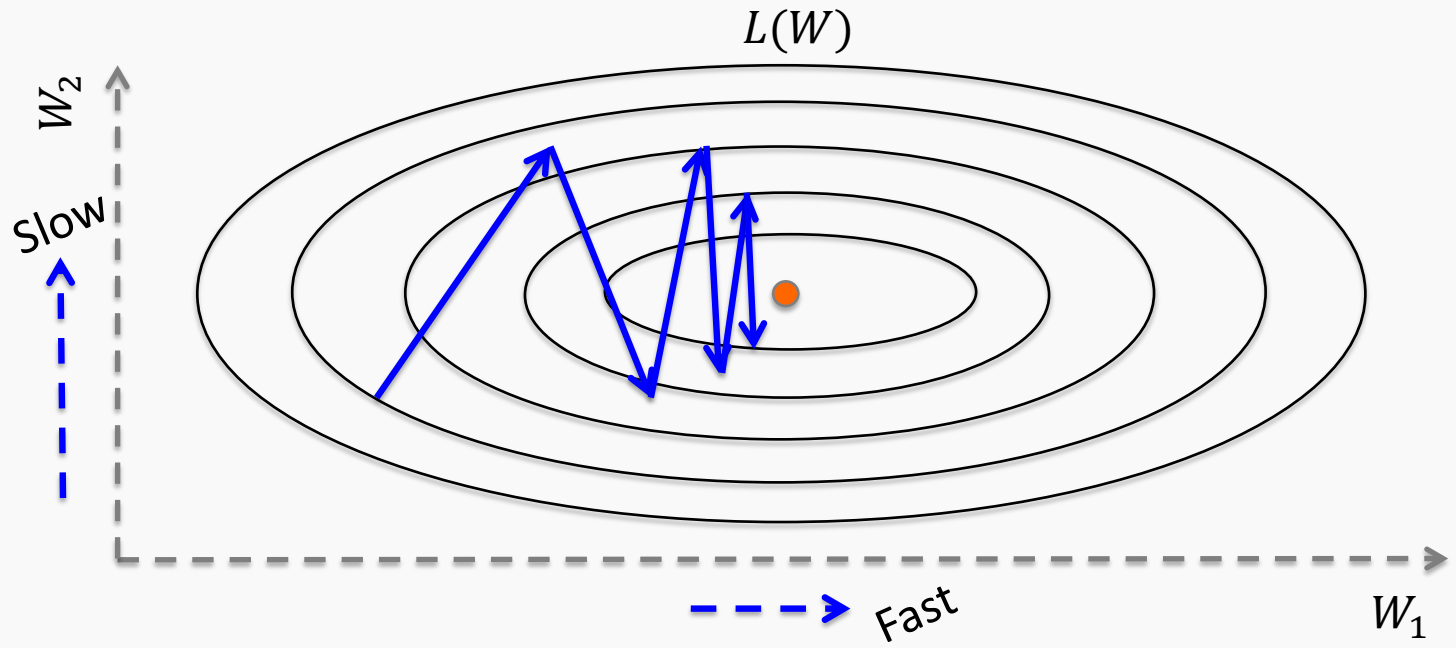
Optimization

- Challenges in Optimization
- Momentum
- **Adaptive Learning Rate**
- Parameter Initialization
- Batch Normalization

Regularization of NN

- Norm Penalties
- Early Stopping
- Data Augmentation
- Sparse Representation
- Dropout

Adaptive Learning Rates

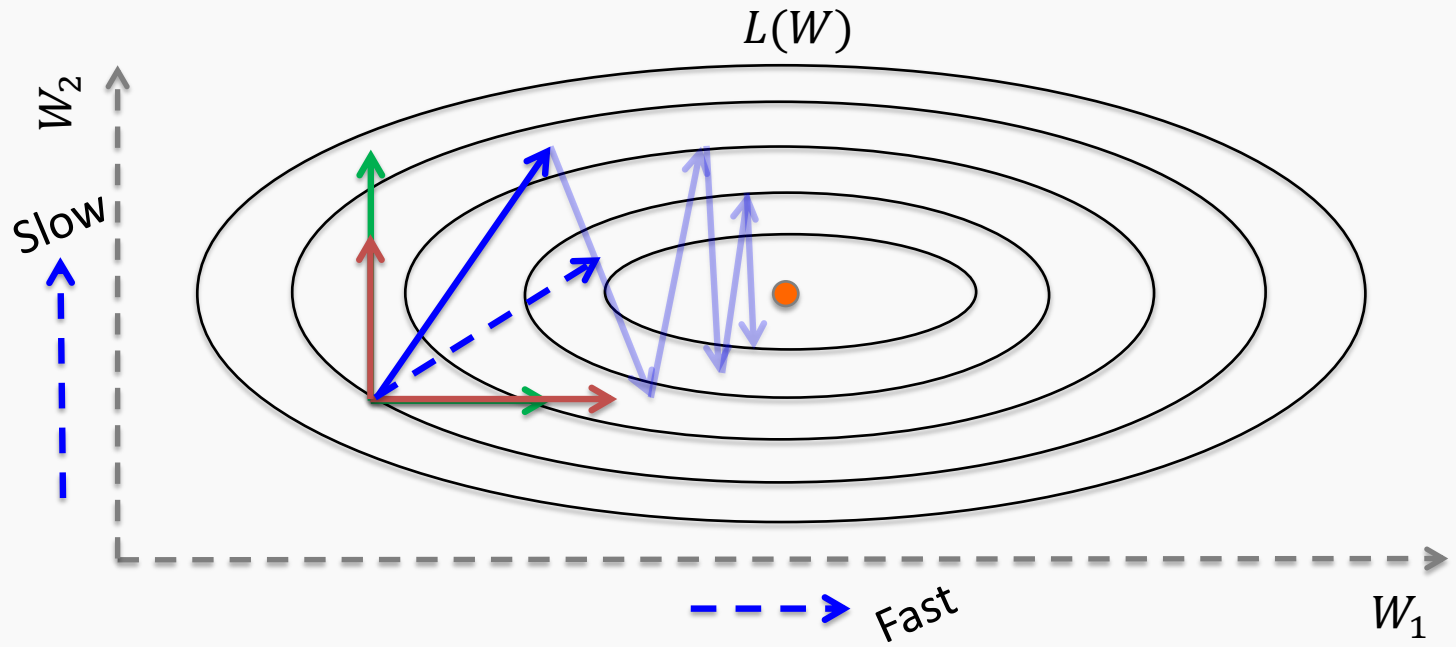


Oscillations along vertical direction

- Learning must be slower along parameter 2

Use a different learning rate for each parameter?

Adaptive Learning Rates

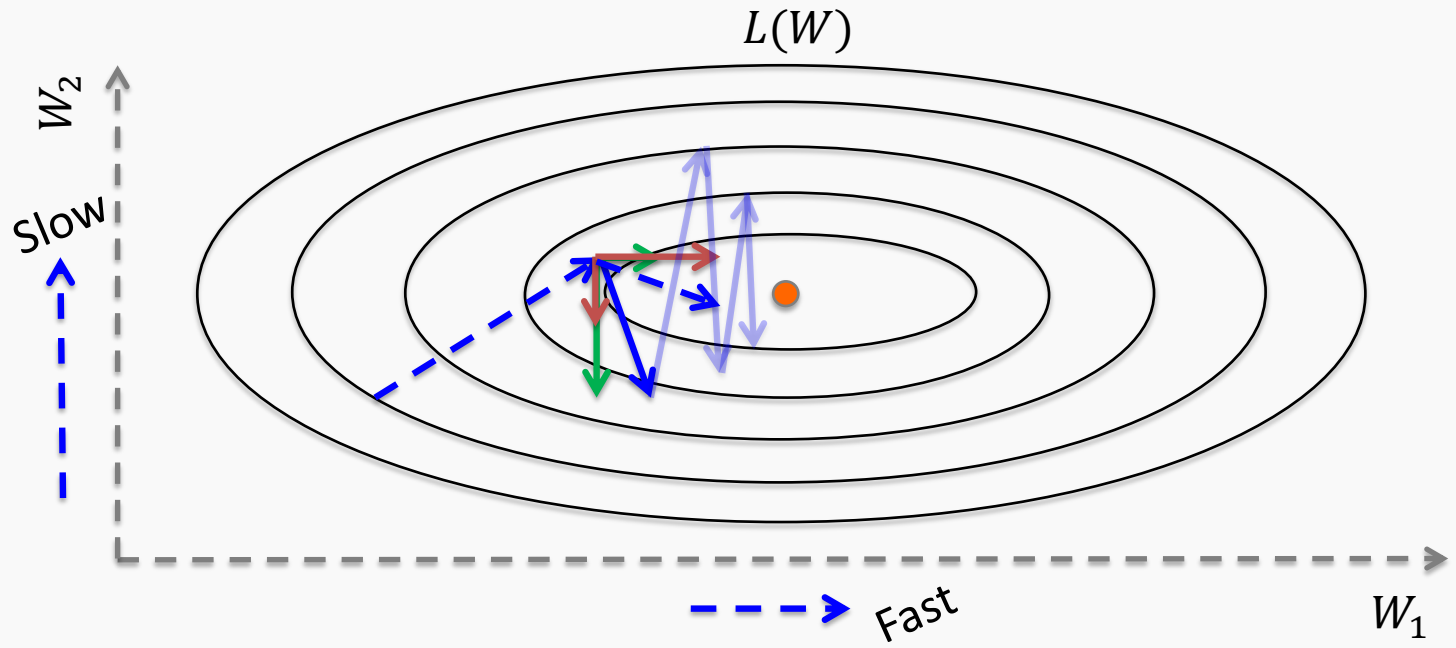


Oscillations along vertical direction

- Learning must be slower along parameter 2

Use a different learning rate for each parameter?

Adaptive Learning Rates

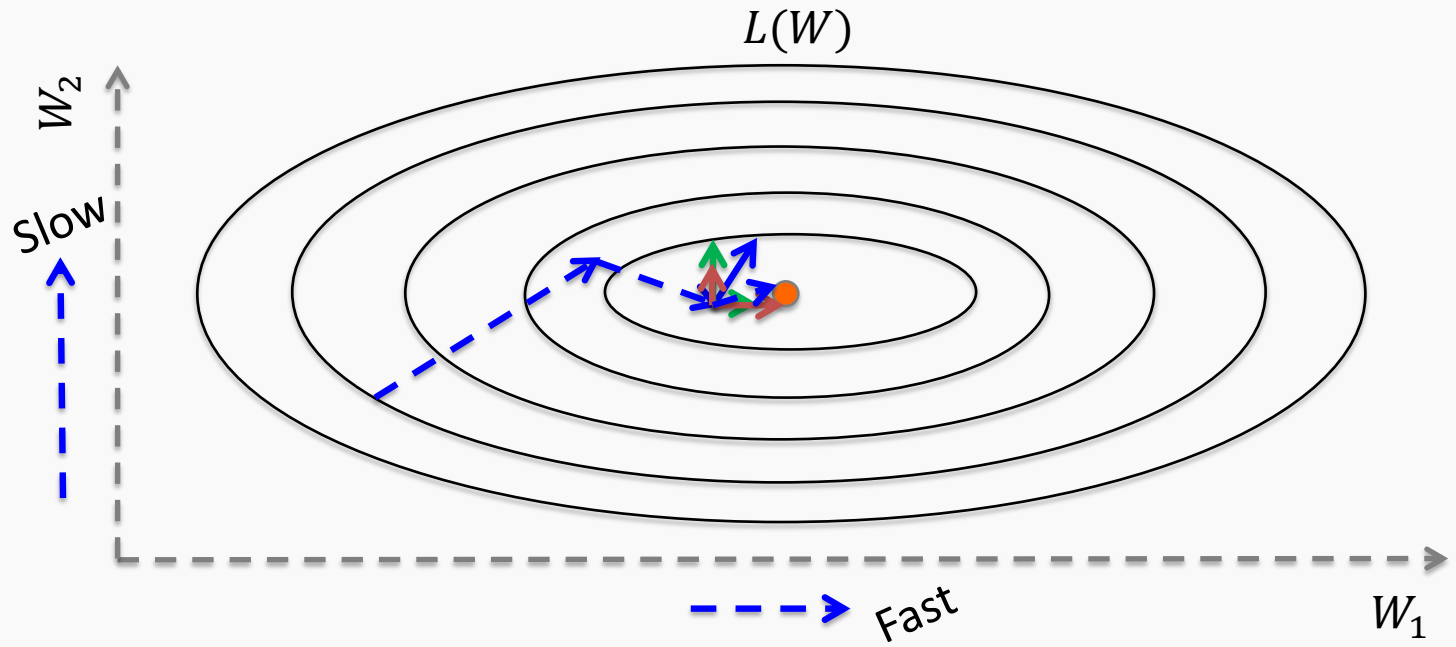


Oscillations along vertical direction

- Learning must be slower along parameter 2

Use a different learning rate for each parameter?

Adaptive Learning Rates

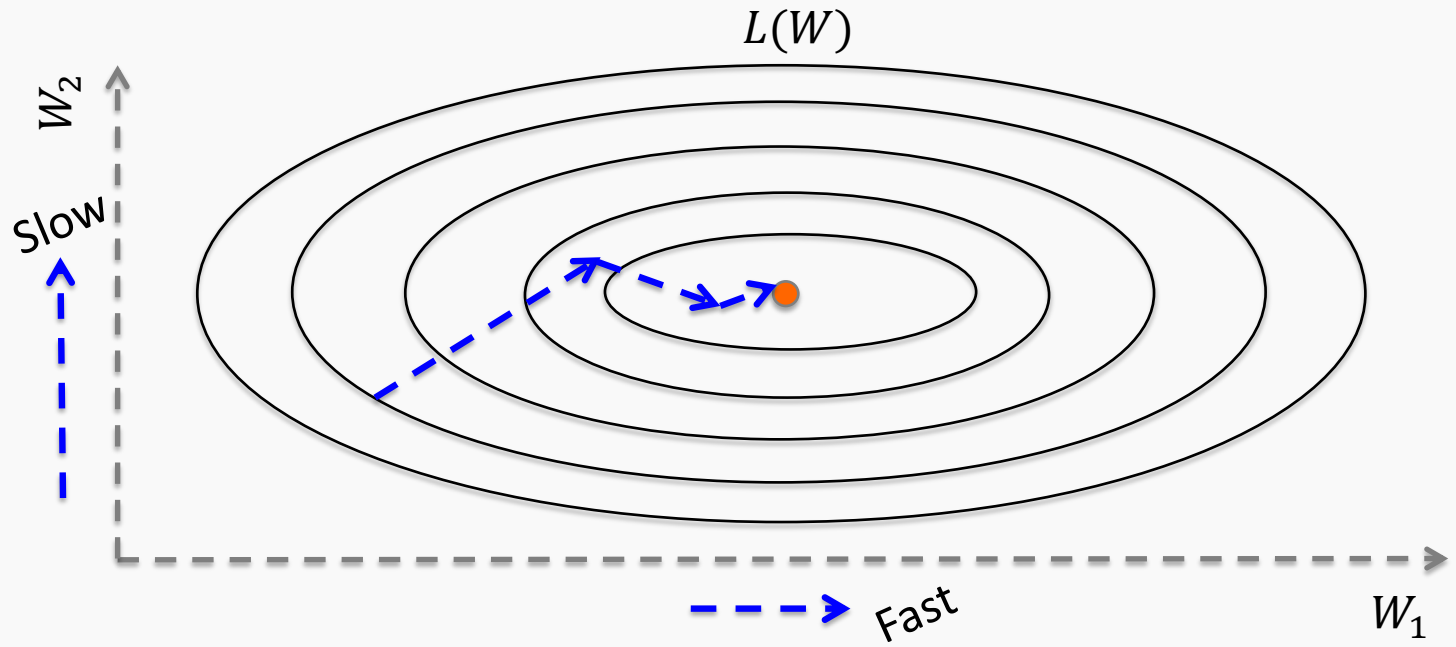


Oscillations along vertical direction

- Learning must be slower along parameter 2

Use a different learning rate for each parameter?

Adaptive Learning Rates



Oscillations along vertical direction

- Learning must be slower along parameter 2

Use a different learning rate for each parameter?

AdaGrad

- Accumulate squared gradients:

$$r_i = r_i + g_i^2$$

g is the gradient

- Update each parameter:

$$W_i = W_i - \frac{\epsilon}{\delta + \sqrt{r_i}} g_i$$

Inversely proportional to cumulative squared gradient

- Greater progress along gently sloped directions

AdaGrad

δ is a small number, making sure this does not become too large

Old gradient descent:

$$g = \frac{1}{m}\sum_i \nabla_W L(f(x_i; W), y_i) \qquad W^* \quad W - \lambda g$$

We would like $\lambda's$ not to be the same and inversely proportiona o the $|g_i|$

$$W_i^* = W_i - \lambda_i g_i \qquad \lambda_i \propto \frac{1}{|g_i|} = \frac{1}{\delta + |g_i|}$$

New gradient descent with adaptive learning rate:

$$r_i^* = r_i + g_i^2 \qquad W_i^* = W_i - \frac{\epsilon}{\delta + \sqrt{r_i}} \mathrm{g_i}$$

RMSProp

- For non-convex problems, AdaGrad can prematurely decrease learning rate
- Use exponentially weighted average for gradient accumulation

$$r_i = \rho r_i + (1-\rho) g_i^2$$

$$W_i = W_i - \frac{\epsilon}{\delta + \sqrt{r_i}} g_i$$

Adam

- RMSProp + Momentum
- Estimate first moment:

$$v_i = \rho_1 v_i + (1 - \rho_1) g_i$$

Also applies bias correction to v and r

- Estimate second moment:

$$r_i = \rho_2 r_i + (1 - \rho_2) g_i^2$$

- Update parameters:

$$W_i = W_i - \frac{\epsilon}{\delta + \sqrt{r_i}} \nu_i$$

Works well in practice, is fairly robust to hyper-parameters

Outline

Optimization

- Challenges in Optimization
- Momentum
- Adaptive Learning Rate
- **Parameter Initialization**
- Batch Normalization

Regularization of NN

- Norm Penalties
- Early Stopping
- Data Augmentation
- Sparse Representation
- Dropout

Parameter Initialization

- Goal: break symmetry between units
  - so that each unit computes a different function
- Initialize all weights (not biases) randomly
  - Gaussian or uniform distribution
- Scale of initialization?
  - *Large* -> grad explosion, *Small* -> grad vanishing

Xavier Initialization

- Heuristic for all outputs to have unit variance
- For a fully-connected layer with m inputs:

$$W_{ij} \sim N\left(0, \frac{1}{m}\right)$$

- For ReLU units, it is recommended:

$$W_{ij} \sim N\left(0, \frac{2}{m}\right)$$

Normalized Initialization

- Fully-connected layer with m inputs, n outputs:

$$W_{ij} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right)$$

- Heuristic trades off between initialize all layers have same activation and gradient variance
- Sparse variant when m is large
  - Initialize *k* nonzero weights in each unit

LiveSlides web content

To view

Download the add-in.

liveslides.com/download

Start the presentation.

Outline

Optimization

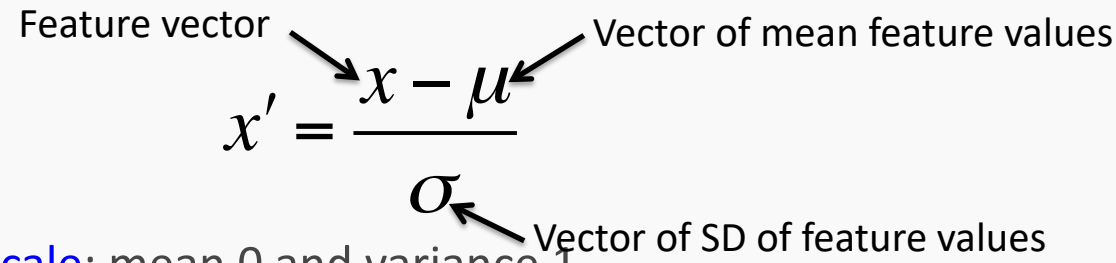
- **Challenges in Optimization**
- Momentum
- Adaptive Learning Rate
- Parameter Initialization
- **Batch Normalization**

Regularization of NN

- Norm Penalties
- Early Stopping
- Data Augmentation
- Sparse Representation
- Dropout

Feature Normalization

Good practice to normalize features before applying learning algorithm:

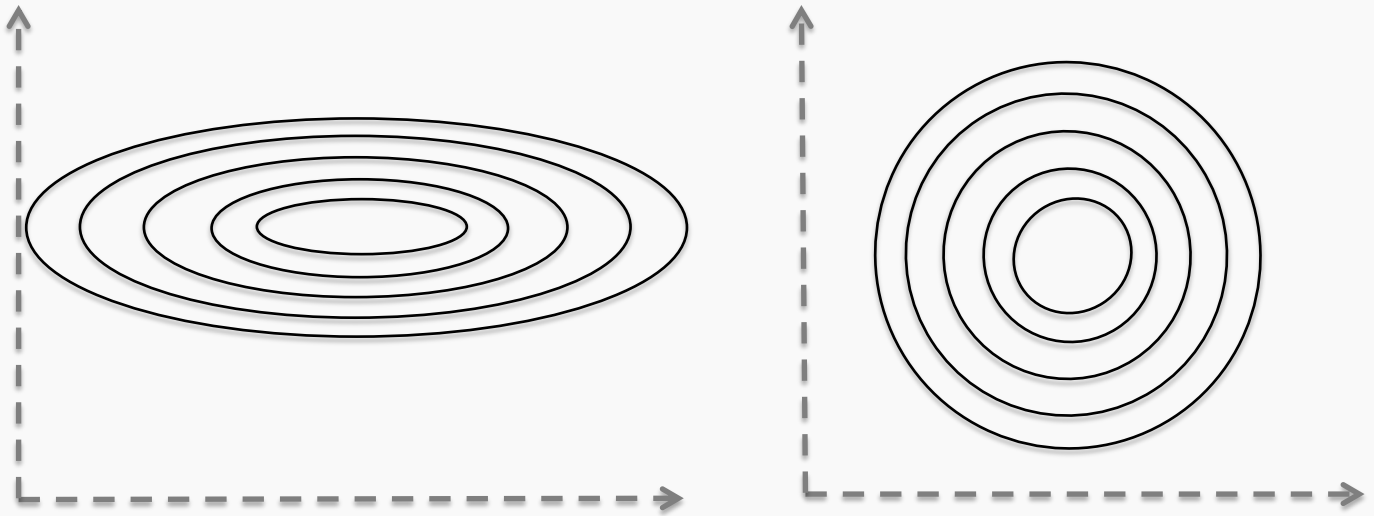


Features in same scale: mean 0 and variance 1

- Speeds up learning

Feature Normalization

$L(W)$

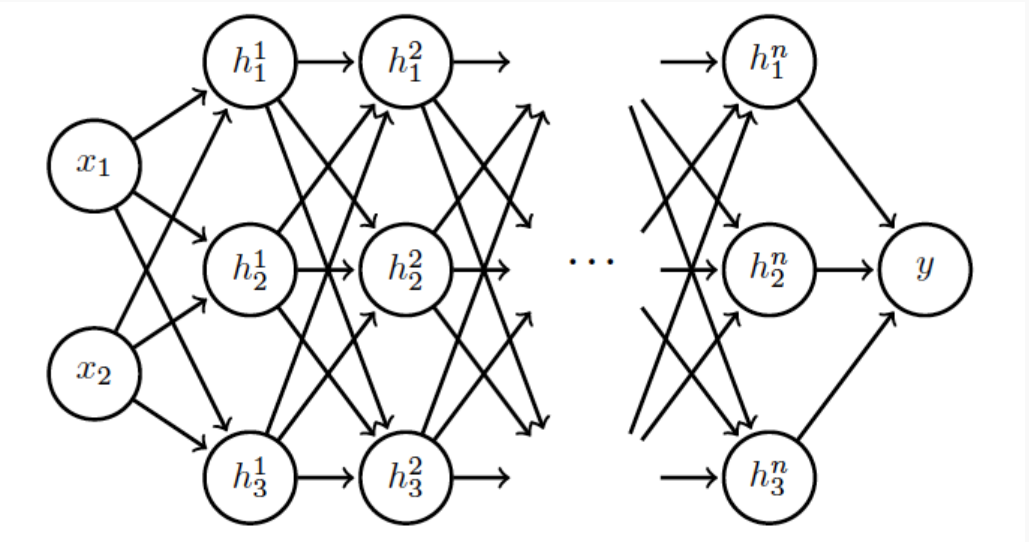


Before normalization

After normalization

Internal Covariance Shift

Each hidden layer changes distribution of inputs to next layer: *slows down learning*



Batch Normalization

Training time:

- Mini-batch of activations for layer to normalize

$$H = \begin{bmatrix} H_{11} & \cdots & H_{1K} \\ \vdots & \ddots & \vdots \\ H_{N1} & \cdots & H_{NK} \end{bmatrix}$$

K hidden layer activations

N data points in mini-batch

Batch Normalization

Training time:

- Mini-batch of activations for layer to normalize

where

$$H' = \frac{H - \mu}{\sigma}$$

$$\mu = \frac{1}{m}\sum_i H_{i,:} \qquad \sigma = \sqrt{\frac{1}{m}\sum_i (H - \mu)_i^2 + \delta}$$

Vector of mean activations across mini-batch

Vector of SD of each unit across mini-batch

Batch Normalization

Training time:

- Normalization can reduce expressive power
- Instead use:

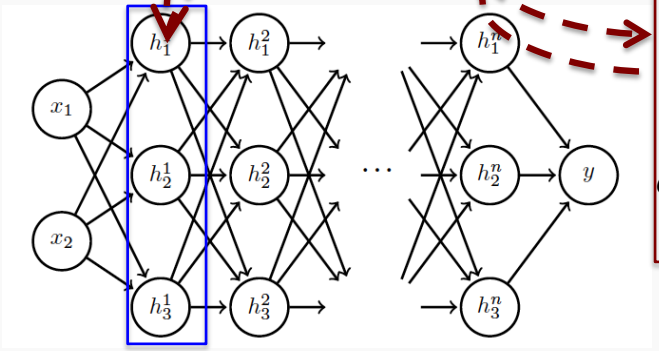
$$\gamma H' + \beta$$

Learnable parameters

- Allows network to control range of normalization

Batch Normalization

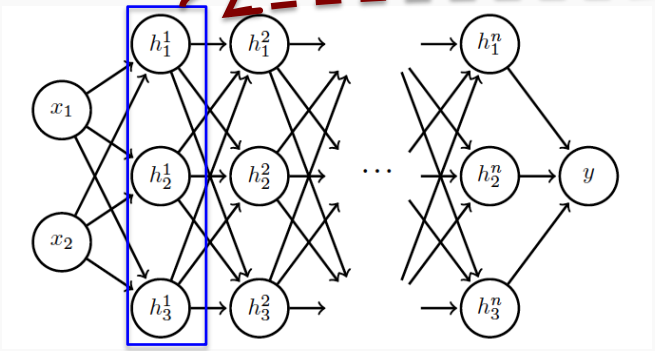
Batch 1



$$\mu^1 = \frac{1}{m}\sum_i H_{i,:}$$

$$\sigma^1 = \sqrt{\frac{1}{m}\sum_i (H-\mu)_i^2 + \delta}$$

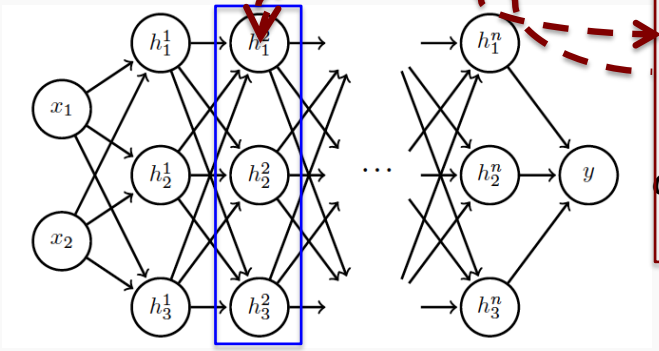
Batch *N*



Add normalization operations for layer 1

Batch Normalization

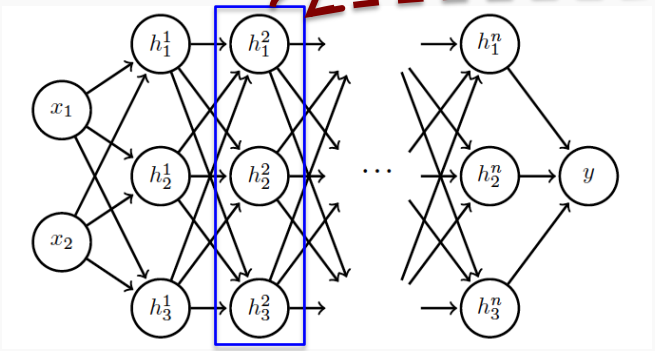
Batch 1



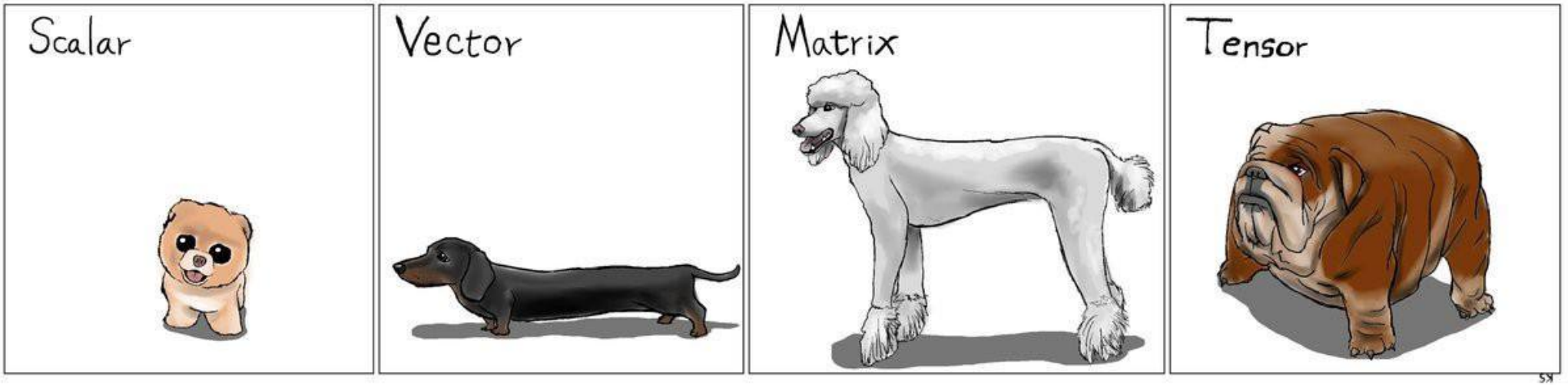
$$\mu^2 = \frac{1}{m}\sum_i H_{i,:}$$

$$\sigma^2 = \sqrt{\frac{1}{m}\sum_i (H-\mu)_i^2 + \delta}$$

Batch *N*



Add normalization operations for layer 2 and so on …


Scalar
Vector
Matrix
Tensor

Outline

Optimization

- Challenges in Optimization
- Momentum
- Adaptive Learning Rate
- Parameter Initialization
- Batch Normalization

Regularization of NN

- Norm Penalties
- Early Stopping
- Data Augmentation
- Sparse Representation
- Dropout

Regularization

Regularization is any modification we make to a learning algorithm that is intended to **reduce its generalization error** but not its training error.

Outline

Optimization

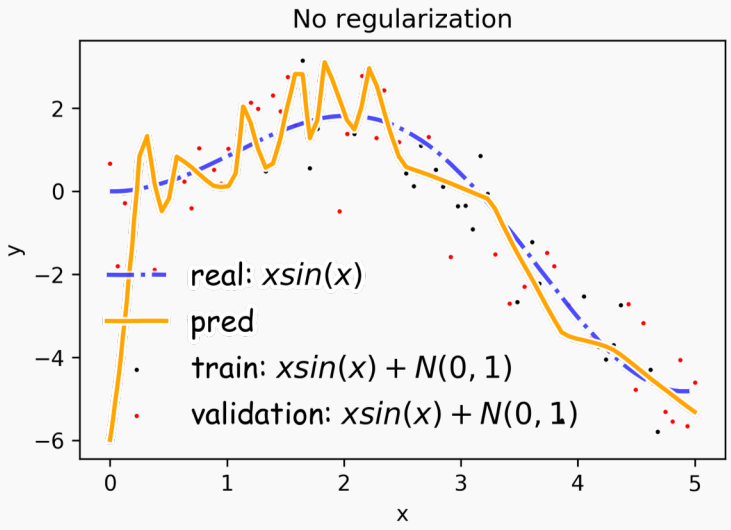
- Challenges in Optimization
- Momentum
- Adaptive Learning Rate
- Parameter Initialization
- Batch Normalization

Regularization of NN

- **Norm Penalties**
- Early Stopping
- Data Augmentation
- Sparse Representation
- Dropout

Overfitting

Fitting a deep neural network with 5 layers and 100 neurons per layer can lead to a very good prediction on the training set but poor prediction on validations set.



Norm Penalties

We used to optimize:

$$L(W; X, y)$$

Change to …

$$L_R(W; X, y) = L(W; X, y) + \alpha\Omega(W)$$

L_2 regularization:

- Weights decay
- MAP estimation with Gaussian prior

$$\Omega(W) = \frac{1}{2} \| W \|_2^2$$

L_1 regularization:

- encourages sparsity
- MAP estimation with Laplacian prior

$$\Omega(W) = \frac{1}{2} \| W \|_1$$

Norm Penalties

We used to optimize:

$$W^{(i+1)} = W^{(i)} - \lambda \frac{\partial L}{\partial W}$$

Change to …

$$L_R(W; X, y) = L(W; X, y) + \frac{1}{2}\alpha W^2$$

$$W^{(i+1)} = W^{(i)} - \lambda \frac{\partial L}{\partial W} - \lambda\alpha\, W$$

Weights decay in proportion to size

Biases not penalized

L_2 regularization:

- **Decay of weights**
- MAP estimation with Gaussian prior

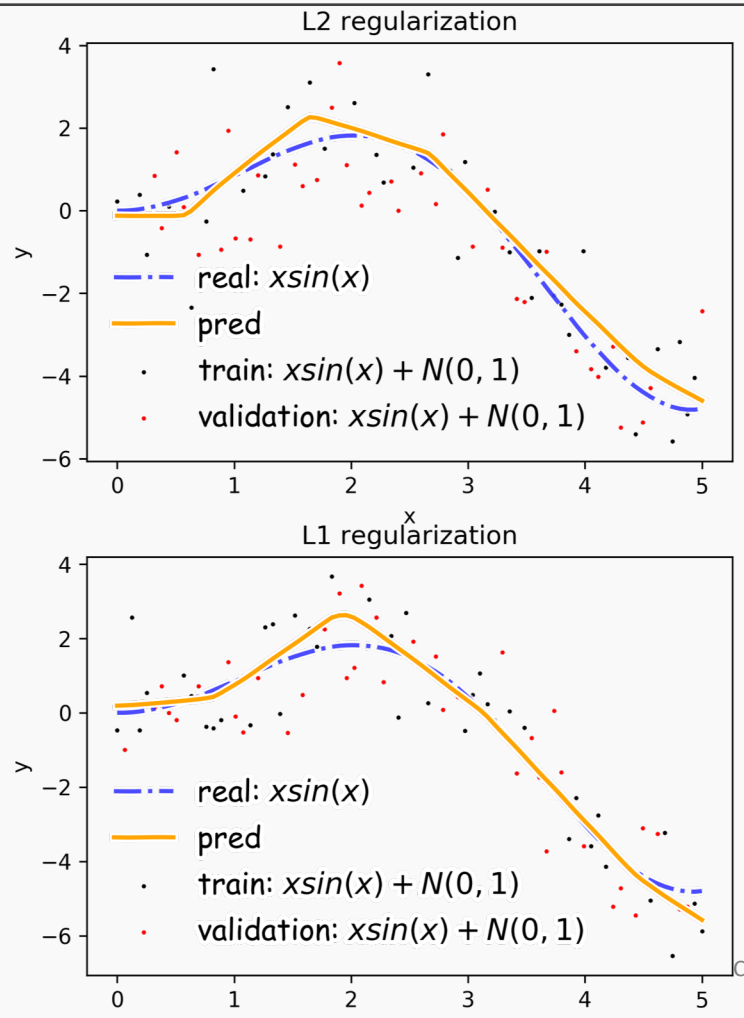
$$\Omega(W) = \frac{1}{2} \| W \|_2^2$$

L_1 regularization:

- encourages sparsity
- MAP estimation with Laplacian prior

$$\Omega(W) = \frac{1}{2} \| W \|_1$$

Norm Penalties



$$\Omega(W) = \frac{1}{2} \| W \|_2^2$$

$$\Omega(W) = \frac{1}{2} \| W \|_1$$

Outline

Optimization

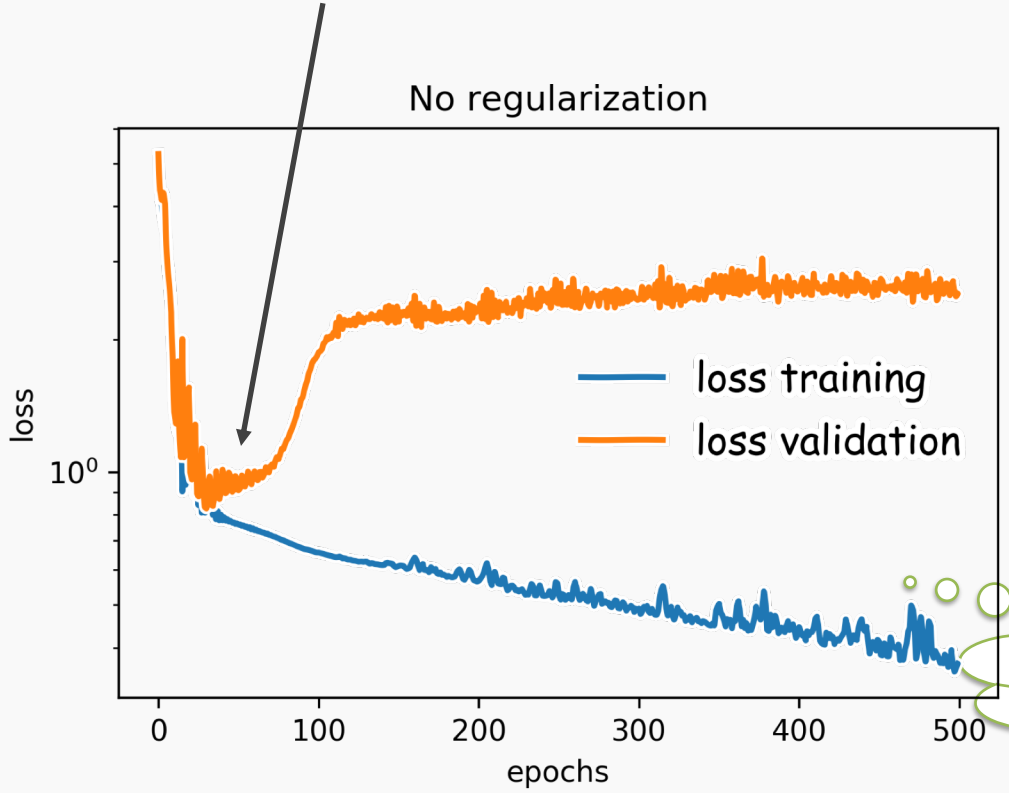
- Challenges in Optimization
- Momentum
- Adaptive Learning Rate
- Parameter Initialization
- Batch Normalization

Regularization of NN

- Norm Penalties
- **Early Stopping**
- Data Augmentation
- Sparse Representation
- Dropout

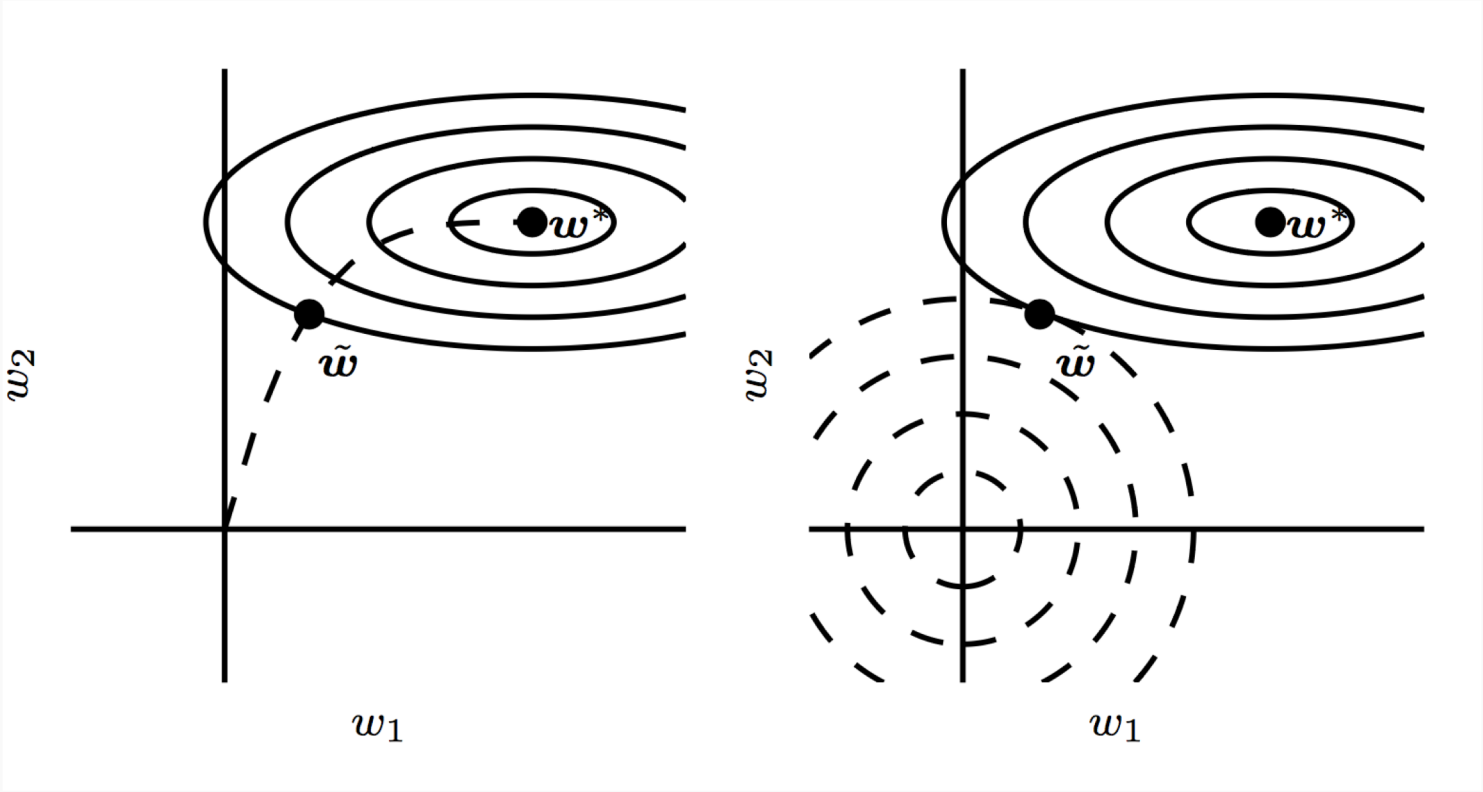
Early Stopping

Early stopping: terminate while validation set performance is better



Training time can be treated as a hyperparameter

Early Stopping



Outline

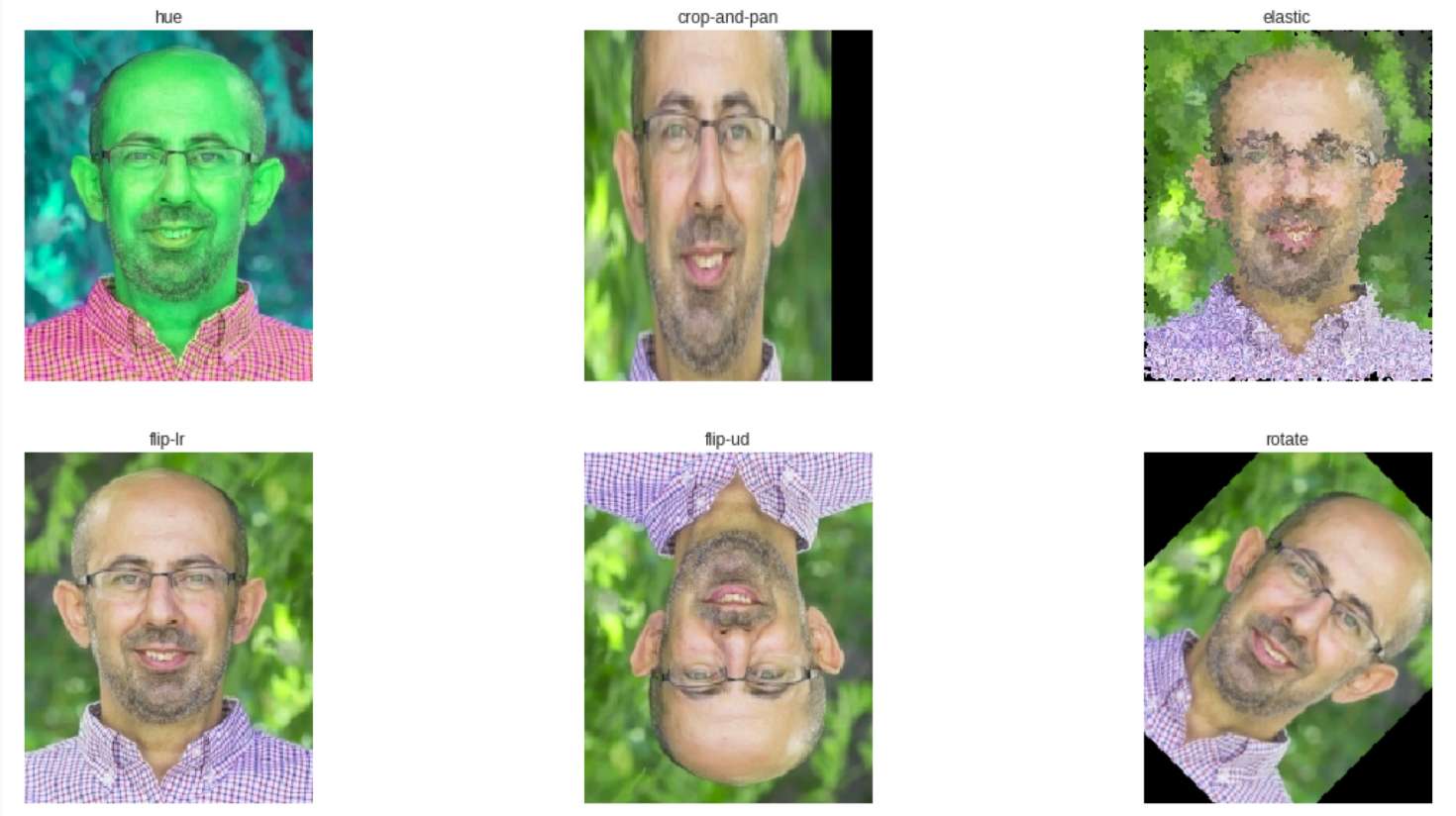
Optimization

- Challenges in Optimization
- Momentum
- Adaptive Learning Rate
- Parameter Initialization
- Batch Normalization

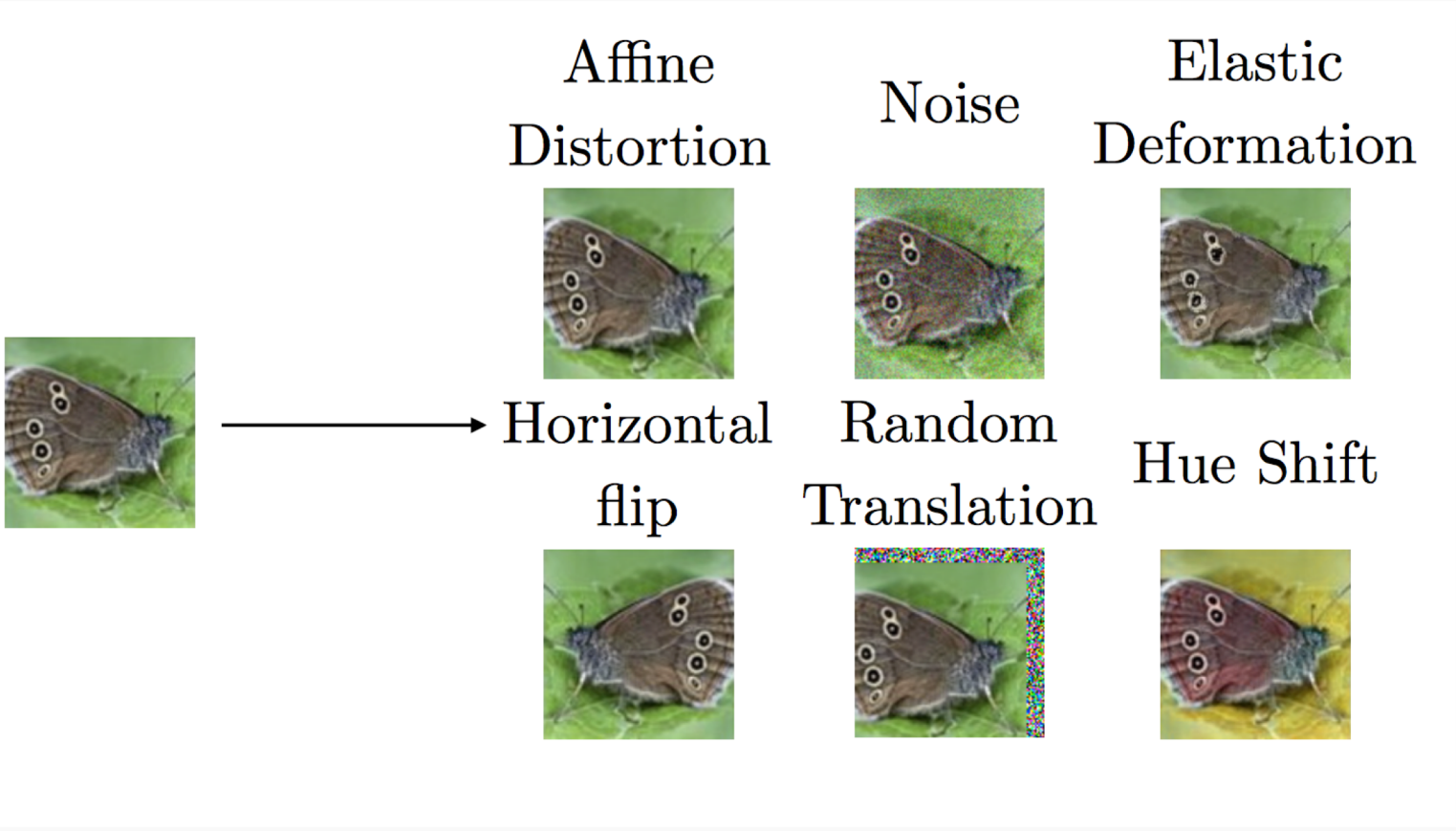
Regularization of NN

- Norm Penalties
- Early Stopping
- **Data Augmentation**
- Sparse Representation
- Dropout

Data Augmentation



Data Augmentation



Outline

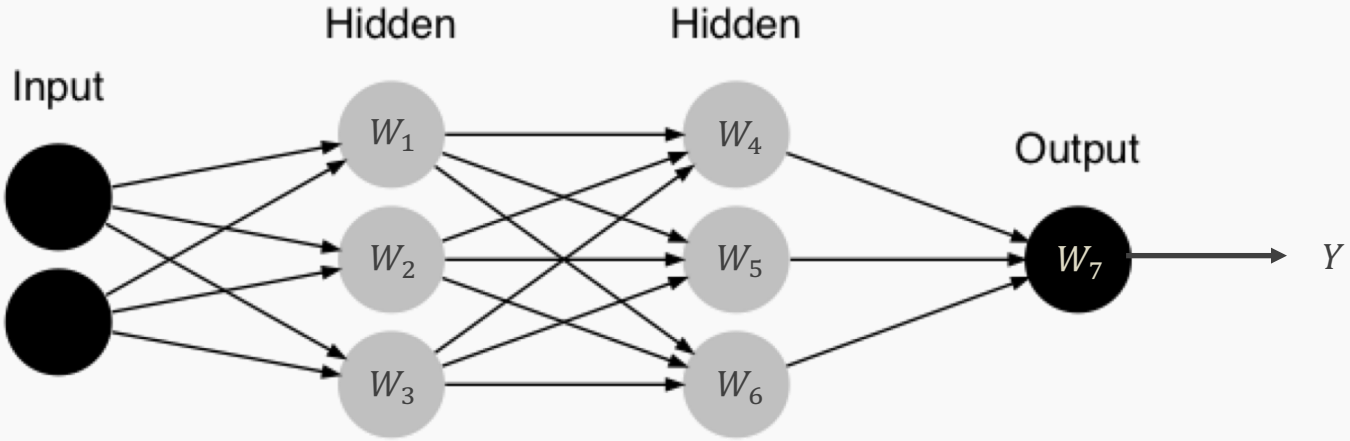
Optimization

- Challenges in Optimization
- Momentum
- Adaptive Learning Rate
- Parameter Initialization
- Batch Normalization

Regularization of NN

- Norm Penalties
- Early Stopping
- Data Augmentation
- **Sparse Representation**
- Dropout

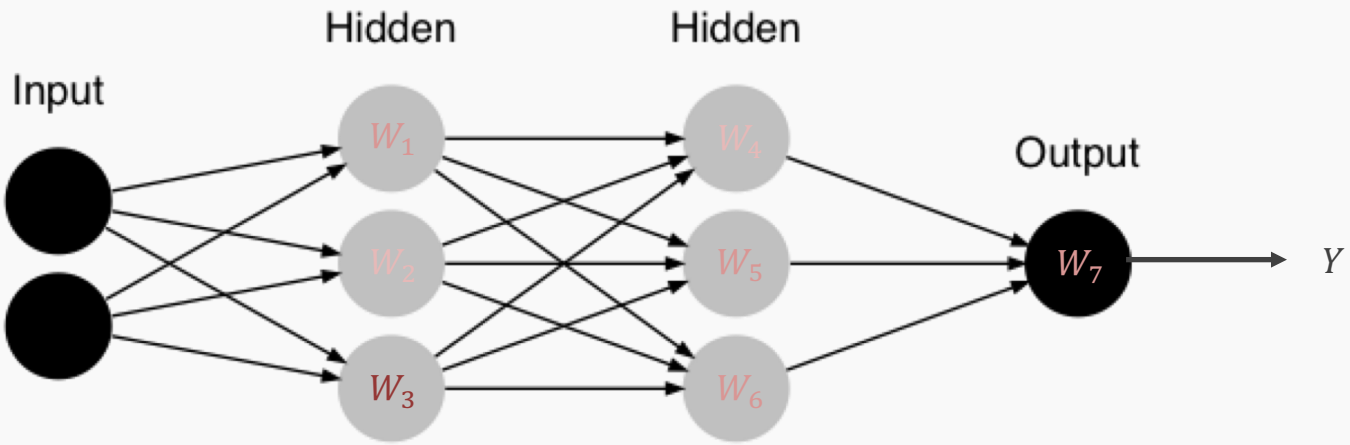
Sparse Representation



$$L(\theta; X, y)$$

$$[4.34] = \underbrace{[3.2 \quad 2.0 \quad 1.8]}_{W_7} \begin{bmatrix} 2 \\ -2.2 \\ 1.3 \end{bmatrix}$$

Sparse Representation

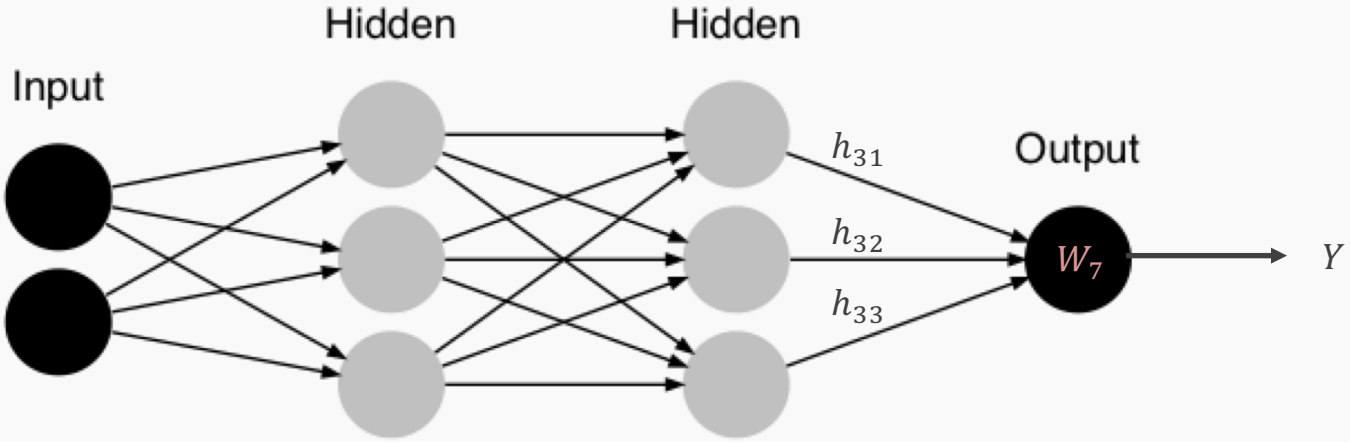


$$L_R(W; X, y) = L(\theta; X, y) + \alpha\Omega(W)$$

$$[0.69] = \underbrace{[0.5 \quad .2 \quad 0.1]}_{W_7} \begin{bmatrix} 2 \\ -2.2 \\ 1.3 \end{bmatrix}$$

Weights in output layer

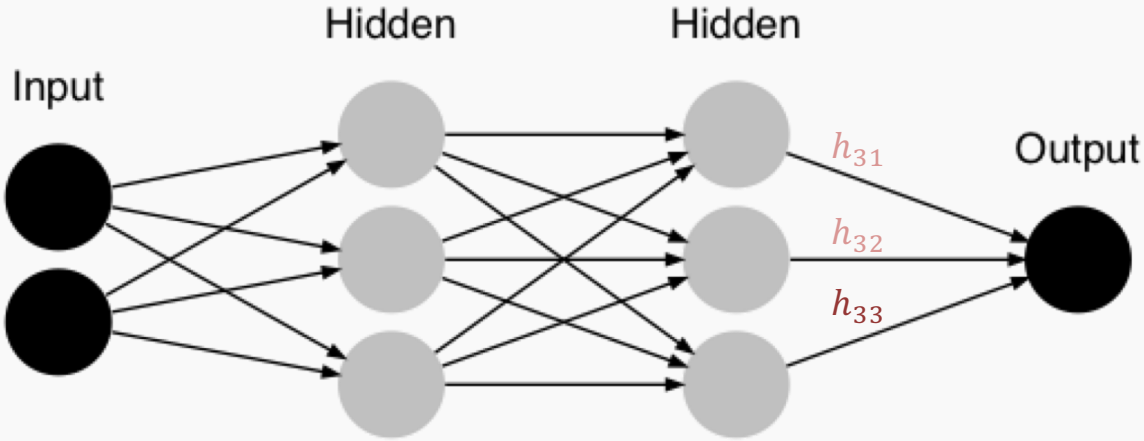
Sparse Representation



$$L(\theta; X, y)$$

$$[4.34] = [3.2 \quad 2 \quad 1] \begin{bmatrix} 2 \\ -2.2 \\ 1.3 \end{bmatrix} \Big\} \; h_{31}, h_{32}, h_{33}$$

Sparse Representation



$$L_R(W; X, y) = L(\theta; X, y) + \alpha\Omega(h)$$

$$[1.3] = [3.2 \quad 2 \quad 1] \begin{bmatrix} 0 \\ -0.2 \\ .9 \end{bmatrix} \Big\} \; h_{31}, h_{32}, h_{33}$$

Output of hidden layer

Outline

Optimization

- Challenges in Optimization
- Momentum
- Adaptive Learning Rate
- Parameter Initialization
- Batch Normalization

Regularization of NN

- Norm Penalties
- Early Stopping
- Data Augmentation
- Sparse Representation
- **Dropout**

Noise Robustness

Random perturbation of network weights

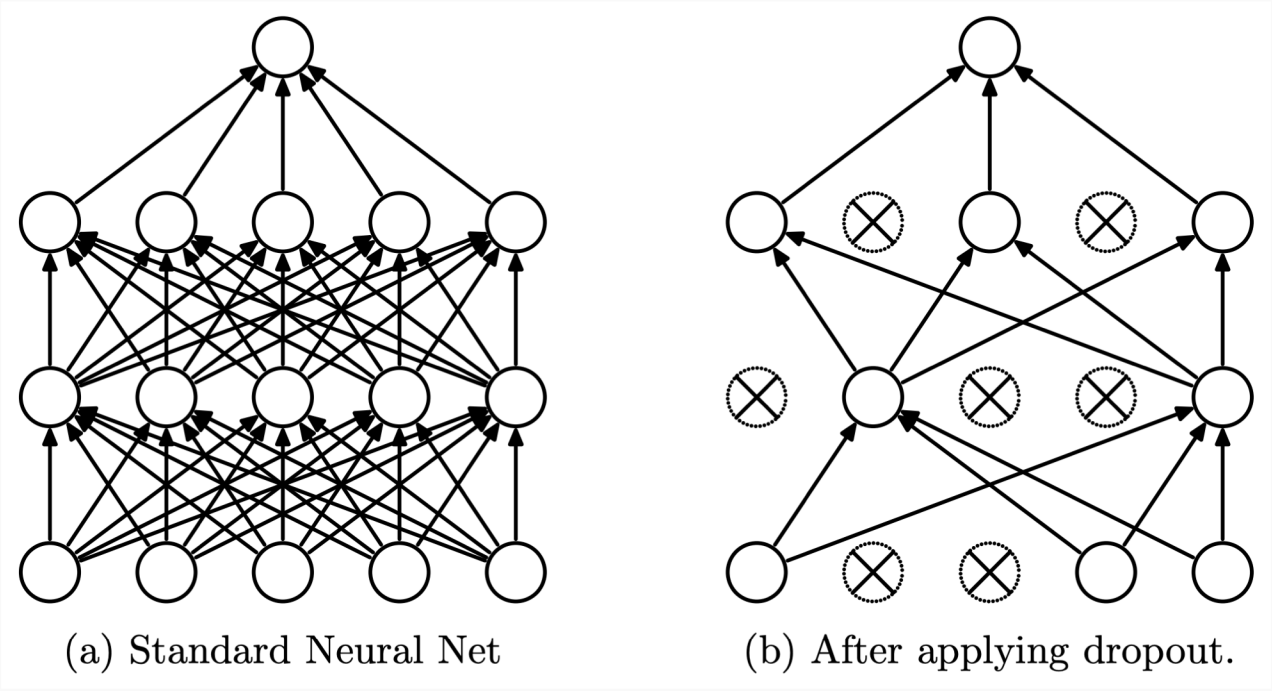
- Gaussian noise: Equivalent to minimizing loss with regularization term
- Encourages smooth function: small perturbation in weights leads to small changes in output

Injecting noise in output labels

- Better convergence: prevents pursuit of hard probabilities

Dropout

- Randomly set some neurons and their connections to zero (i.e. “dropped”)
- Prevent overfitting by reducing co-adaptation of neurons
- Like training many random sub-networks

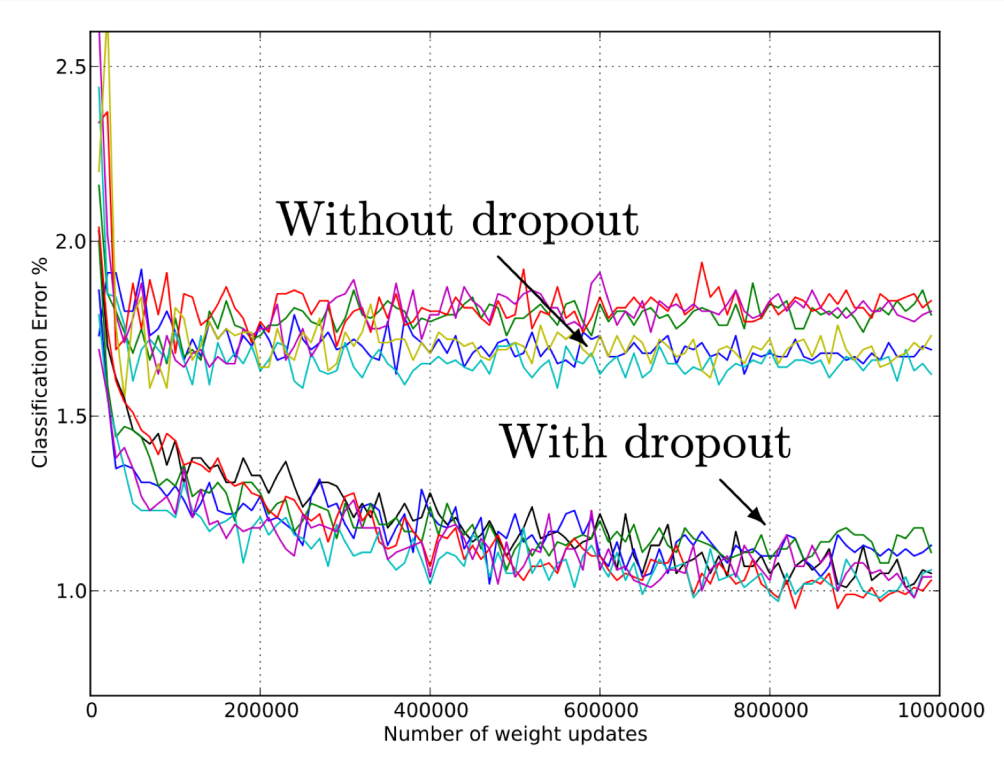


(a) Standard Neural Net

(b) After applying dropout.

Dropout

- Widely used and highly effective
- Proposed as an alternative to ensembling, which is too expensive for neural nets



Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.

http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf

Dropout: Stochastic GD

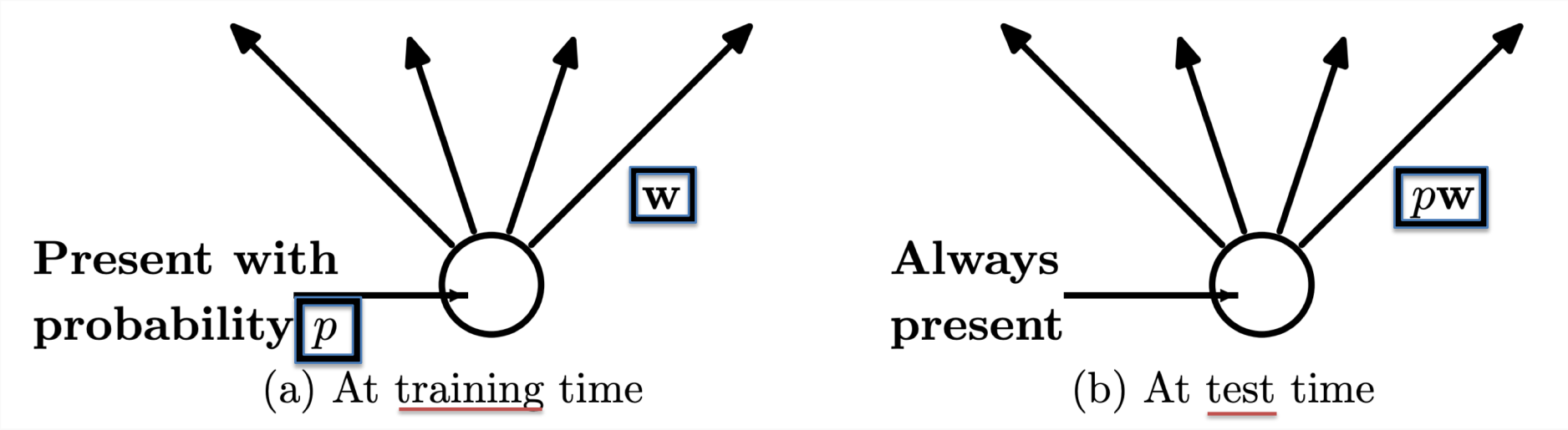
For each new example/mini-batch:

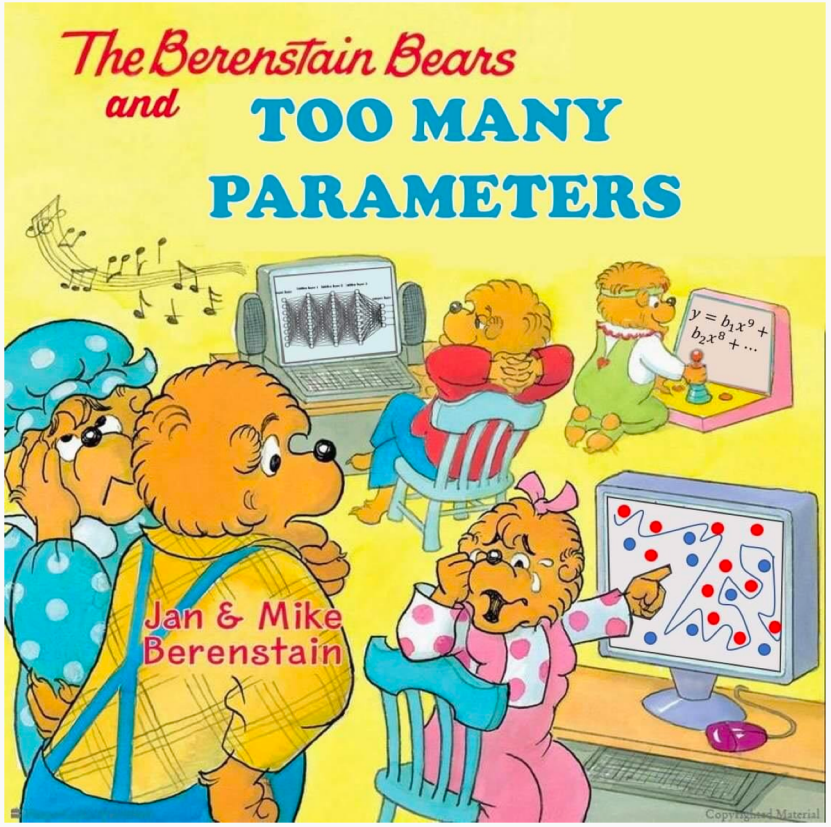
- Randomly sample a binary mask μ independently, where μ_i indicates if input/hidden node i is included
- Multiply output of node i with μ_i, and perform gradient update

Typically, an input node is **included** with **prob=0.8**, hidden node with **prob=0.5.**

Dropout: Weight Scaling

- We can think of dropout as training many of sub-networks
- At **test time**, we can "aggregate" over these sub-networks by **reducing connection weights in proportion to dropout probability, *p***




The Berenstain Bears
and
TOO MANY
PARAMETERS
$y = b_1x^9 + b_2x^8 + \ldots$
Jan & Mike
Berenstain
Copyrighted Material