Machine Learning and Deep Learning
Lecture-08

By: Somnath Mazumdar
Assistant Professor
sma.digi@cbs.dk

Spring 2025

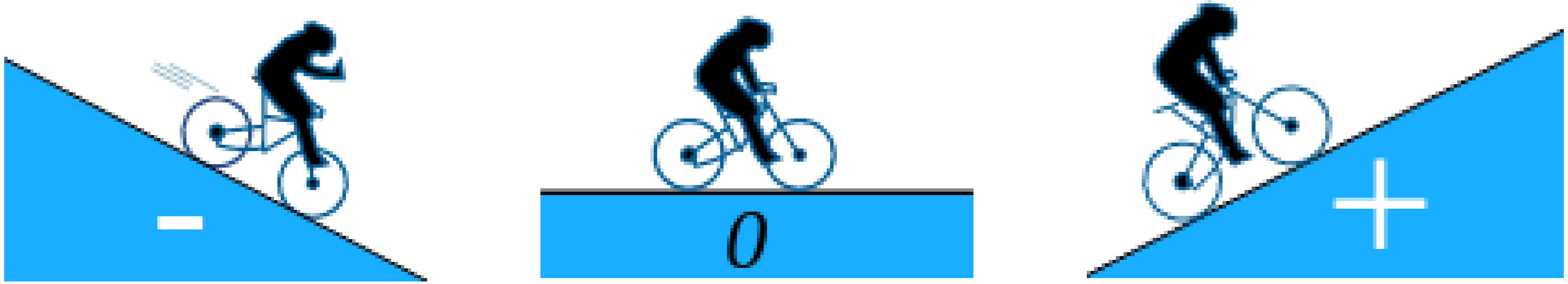
CDSCO2004U/7.5 ECTS

Overview

- Gradient Descent
- Stochastic Gradient Decent
- Regularization: L0/L1/L2
- Early Stopping
- Dropout
- BatchNorm

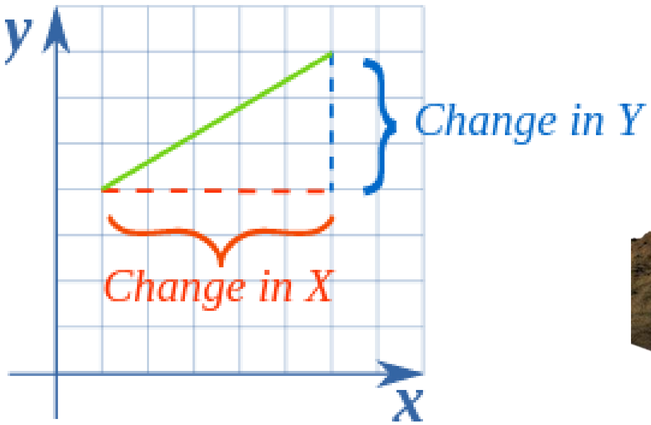
Gradient Descent

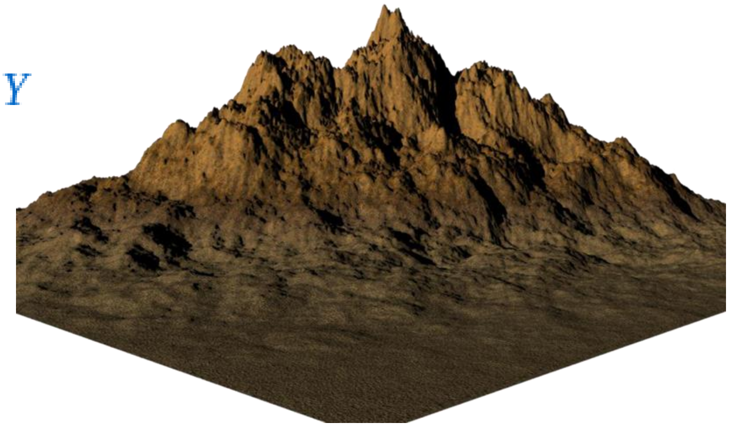
Gradient + Descent



Gradient presents slope
Descent: Moving down

Gradient = Change in Y/ Change in X



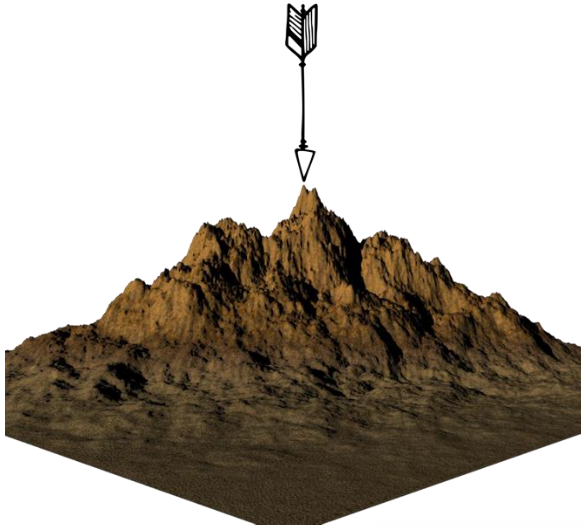


Gradient Descent (GD)

- An optimization algorithm.
- Capable of finding optimal solutions to a wide range of problems.

- Normal equations only solve linear least squares problems.
  - Normal equations cost $O(nd^2 + d^3)$.

- GD solves many other problems.
  - GD costs O(ndt) to run for ‘t’ iterations.
    - GD can be faster when ‘d’ is very large.



Gradient presents slope

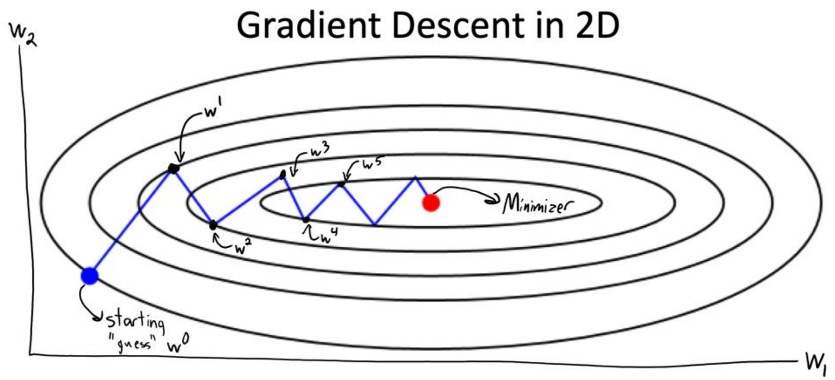
Gradient Descent (GD)

https://www.popsci.com/

Gradient Descent (GD)

- Idea: To tweak parameters iteratively to minimize a cost function.
  - It starts with a “guess” w^0.
  - Use gradient $\nabla f(w^0)$ to generate a better guess w^1.
  - Uses gradient $\nabla f(w^1)$ to generate a better guess w^2.
  - Uses gradient $\nabla f(w^2)$ to generate a better guess w^3.

- Limit of w^t as ‘t’ goes to ∞ has $\nabla f(w^t) = 0$.
- It converges to the global optimum if ‘f’ is convex.



Gradient in vector analysis, is a vector operator denoted ∇ (called del/nabla)

Learning Rate

- Amount that weights are updated is called step size or “learning rate.”
  - Learning rate is a hyperparameter range between 0 and 1.

- Learning rate controls how quickly the model is adapted to the problem
  - Smaller learning rates require more training epochs
  - larger learning rates require fewer training epochs.

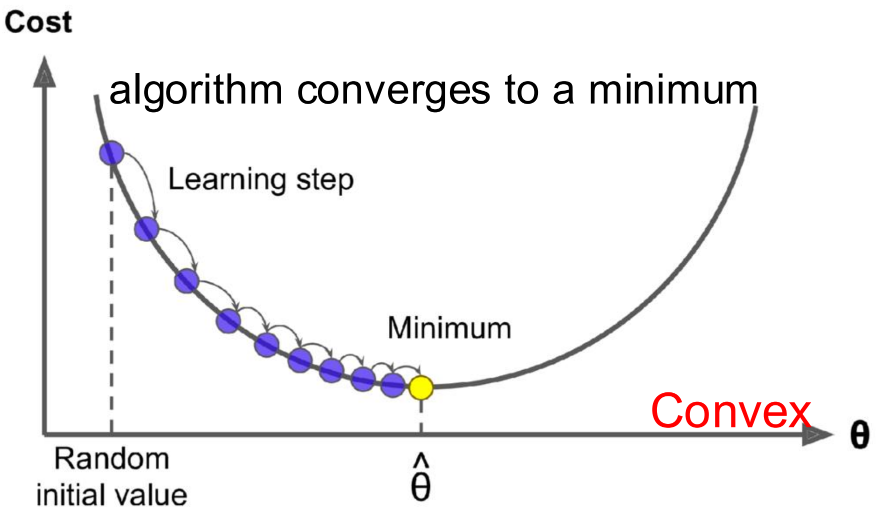
- Large learning rate can converge too quickly to a suboptimal solution.
  - Small learning rate would stuck the model.

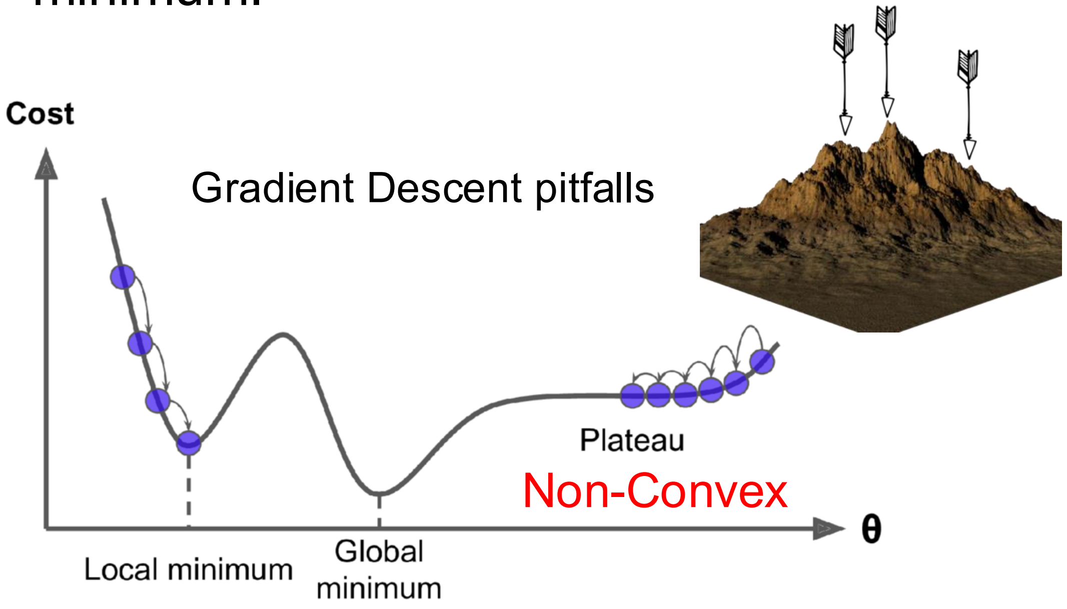
https://machinelearningmastery.com/

Gradient Descent (GD)

- Model parameters are initialized randomly.
- Tweak repeatedly to minimize cost function/MSE.
- Learning step size is proportional to slope of cost function
  - Steps gradually get smaller as parameters approach minimum.

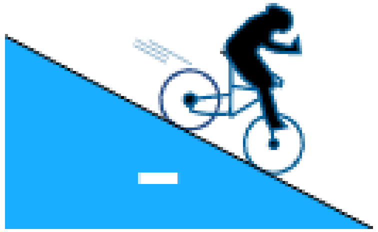
- Random initialization starts algorithm on left.
- Converges to local minimum (as good as global) minimum.
- Starts on right, will take a very long time to cross plateau.
  - If stops early, never reach global minimum.


algorithm converges to a minimum


Gradient Descent pitfalls

Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow by Aurélien Géron, 2019.

GD: Local Minimum



- Start with w^0 (initial guess)
- Generate new guess by moving in negative gradient direction: $w^1 = w^0 - \alpha^0 \nabla f(w^0)$
  - It decreases 'f' if "step size" α^0 is small enough.
  - We decrease α^0 if it increases 'f'.

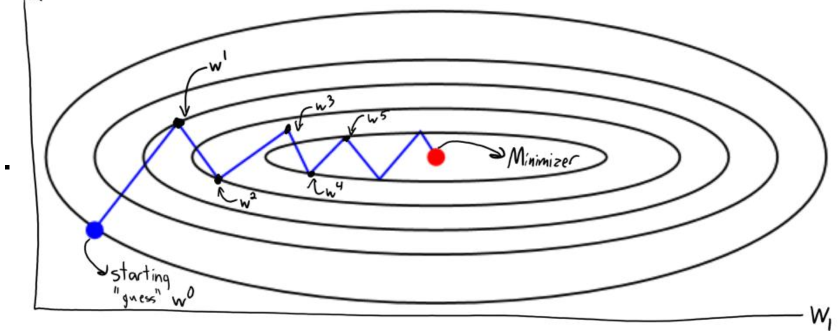
α = learning rate

- Repeat (to successively refine guess): $w^{t+1} = w^t - \alpha^t \nabla f(w^t) \quad \text{for } t = 1, 2, 3, \dots$

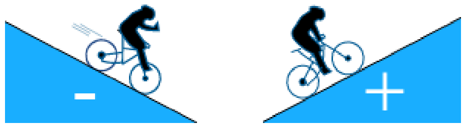
- Stop if not making progress or $\underbrace{\|\nabla f(w^t)\| \leq}_{\text{Approximate local minimum}} \varepsilon$ ↳ Some small scalar.

- Under weak conditions, algorithm converges to a 'w' with ∇f(w) = 0.
- 'f' is bounded, ∇f doesn't change arbitrarily fast, small and constant α^t .

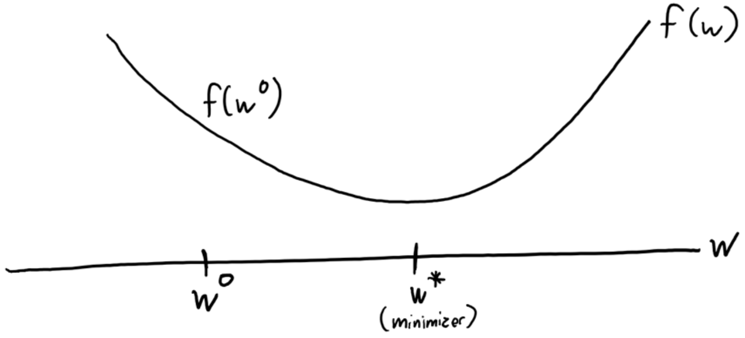
Gradient Descent in 2D

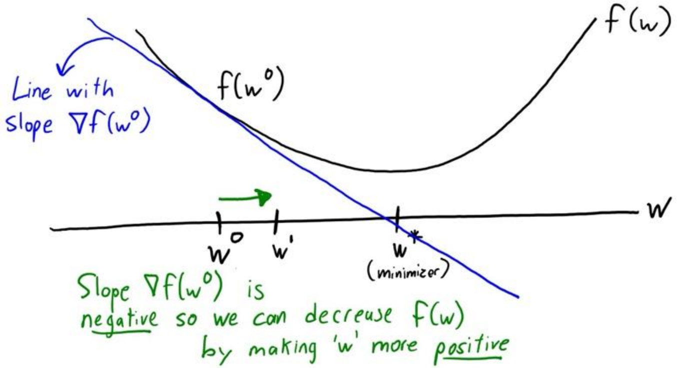


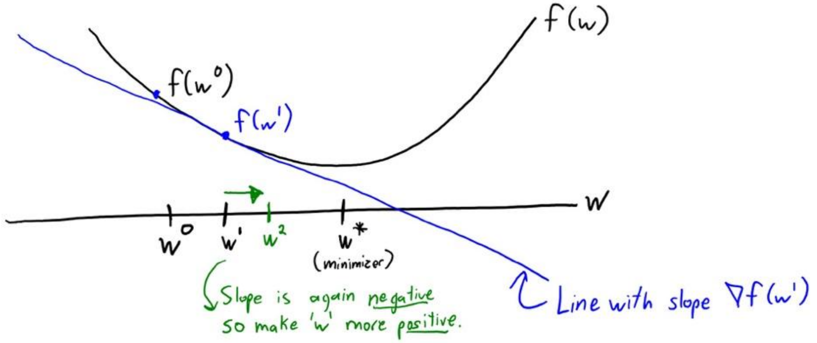
GD: Local Minimum

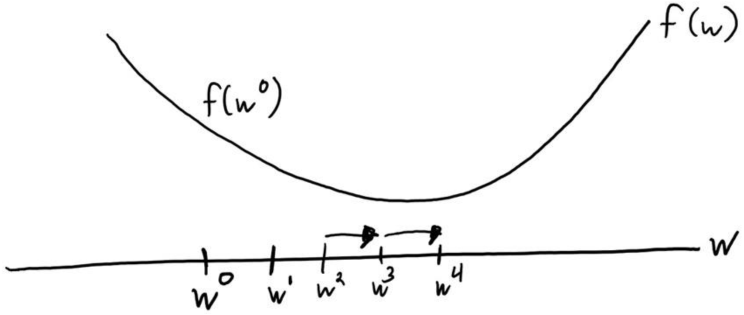


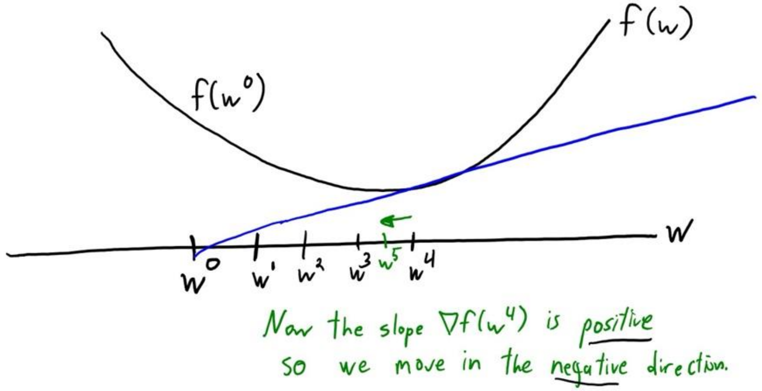
- GD based on: Given parameters ‘w’, direction of largest decrease is − ∇ f(w).




f(w)
Line with slope ∇f(w⁰)
f(w⁰)
w
w⁰
w¹
w* (minimizer)
Slope ∇f(w⁰) is negative so we can decrease f(w) by making 'w' more positive






f(w)
f(w⁰)
w
w⁰
w¹
w²
w³
w⁵
w⁴
Now the slope ∇f(w⁴) is positive so we move in the negative direction.

GD: Ensure all features have a similar scale

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD)

- DNN are trained using stochastic gradient descent.
  - Estimates error gradient for current state from training data.
  - Updates weights of model using backpropagation.
- We use SGD not really GD.
- Approach to discriminative learning of linear classifiers under convex loss functions (such as (linear) SVM and Logistic Regression).
- Application: text classification, natural language processing (sparse machine learning problems).
- Pros: Efficiency, Ease of implementation
- Cons: Need hyperparameters (regularization parameter, no of iterations)
  - sensitive to feature scaling.

Stochastic Gradient (SG)

- Iterative optimization algorithm to minimize averages

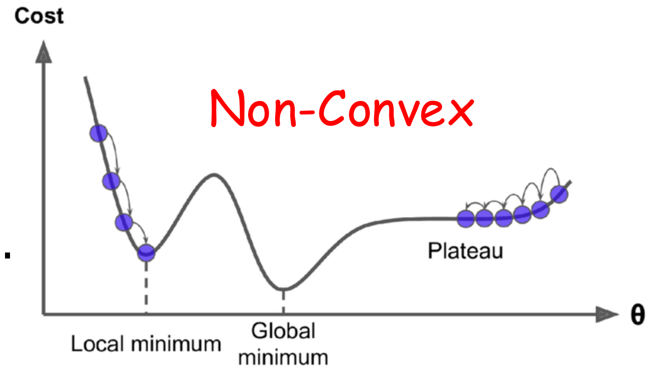
- Algorithm:
- Start with some initial guess, w^0.
  - Generate new guess by moving in negative gradient direction: $w^1 = w^0 - \alpha^0 \nabla f_i(w^0)$
    - For a random training example ‘i’.
  - Repeat successively refine
    - For a random training example ‘i’.

      $w^{t+1} = w^t - \alpha^t \nabla f_i(w^t)$ for $t = 1, 2, 3, \dots$

- Gradient Cost: Gradient is independent of ‘n’.
  - Iterations are ‘n’ times faster than GD iterations.

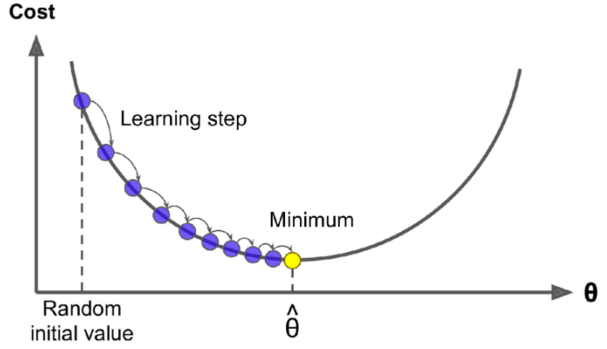
Stochastic Gradient Training

- We randomly samples gradients instead of averaging (all).
  - An average estimate --> faster to compute
- Why optimization is important?
  - NN objectives are highly non-convex (and worse with depth).
  - Optimization has huge influence on quality of model.

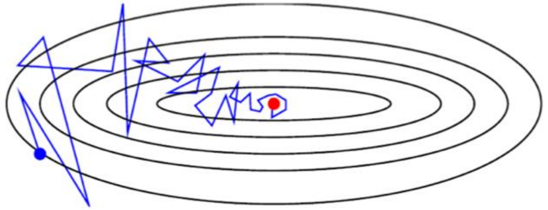

Non-Convex

- Standard training method is stochastic gradient (SG):
  - Choose a random example ‘i’.
  - Use backpropagation to get gradient with respect to all parameters.
  - Take a small step in negative gradient direction.
    - Hard to get SG to even find a local minimum.

Convex



Stochastic Gradient (SG)



$$f(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w)$$

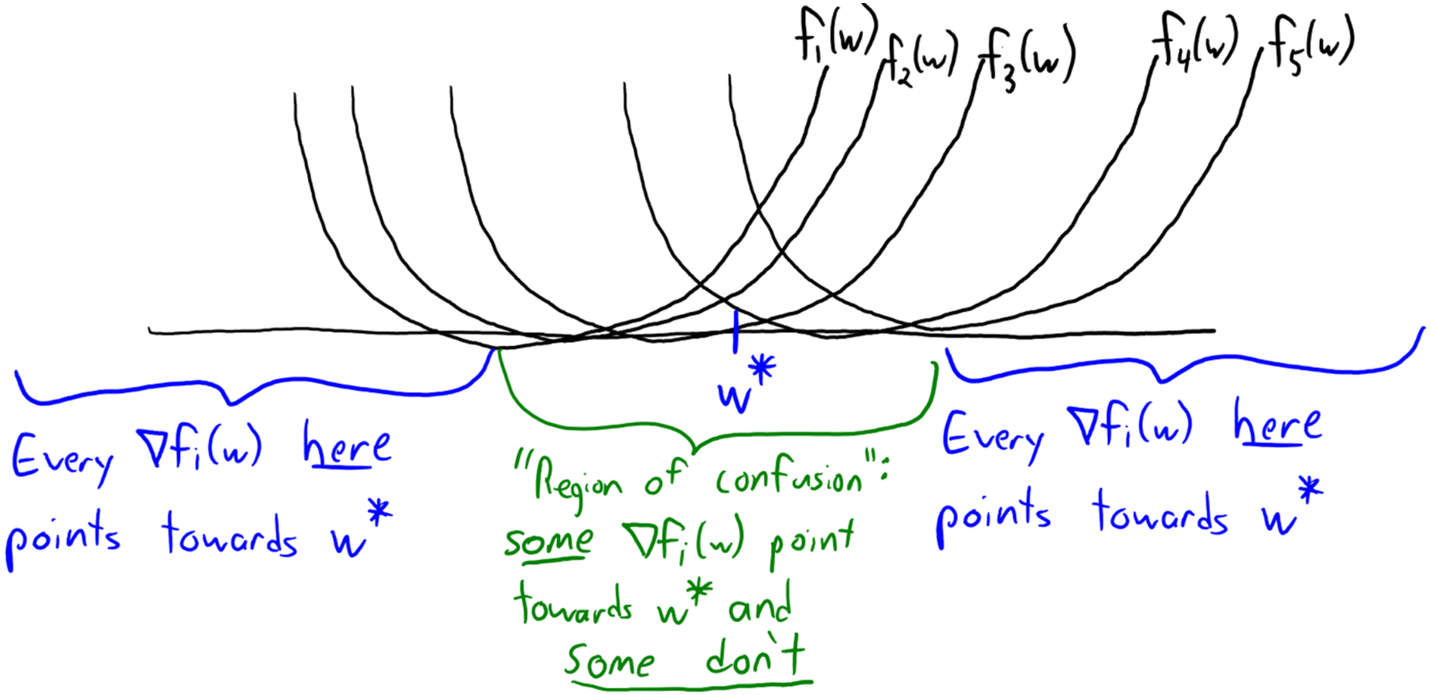
- SG minimizes average of smooth functions:
  - Function $f_i(w)$ is error for example 'i'.
- Iterations perform gradient descent on one random example 'i': $w^{t+1} = w^t - \alpha^t \nabla f_i(w^t)$
  - Cheap iterations even when 'n' is large, but doesn't always decrease 'f'.
  - Solves problem if α^t goes to 0 at an appropriate rate.

α = learning rate

- SG converges very slowly:
  - Large dataset --> No other option.
- Improve performance by reducing variance use:
  - "mini-batches" or random samples but no one random sample.
  - "Variance-reduced" methods for finite training sets.

Variance

- "Confusion" is captured by a kind of variance of gradients: $\frac{1}{n}\sum_{i=1}^{n}\|\nabla f_i(w^t) - \nabla f(w^t)\|^2$
- Variance = 0: Every step goes in right direction. (outside of confusion region)
- Variance = Small: Just inside region of confusion.
- Variance = Large: Many steps will point in wrong direction.



$f_1(w)$
$f_2(w)$
$f_3(w)$
$f_4(w)$
$f_5(w)$
w^*
Every $\nabla f_i(w)$ here points towards w^*
"Region of confusion": some $\nabla f_i(w)$ point towards w^* and some don't
Every $\nabla f_i(w)$ here points towards w^*

Learning Rate/Step size

- Learning rate controls how quickly the model is adapted to the problem
  - Smaller learning rates require more training epochs
  - larger learning rates require fewer training epochs.

- Large learning rate can converge too quickly to a suboptimal solution.
  - Small learning rate would stuck the model.

- For convergence, need decreasing step size.
  - Shrinks size to zero --> converge to w*.
- Achieve by using a step-size sequence E.g., $\alpha^t = .001/t$.
  - Works badly in practice: Initial steps are very large, Later steps get very tiny.
  - Constant or slowly-decreasing step-sizes and/or average w^t.

Batch and Mini-Batch GD

- At each GD step it calculates whole batch of training data.
  - Slow on very large training sets
  - Scales well with number of features.

- Mini-batch GD computes gradients on small random sets of instances called mini-batches.
  - Faster than Stochastic GD.

- Batch GD's stops at minimum (slow).
  - SGD and Mini-batch GD continue to walk around (faster).
  - All algorithms outputs similar models and predicts in exactly same way.

Regression/Penalized Approach: Regularization

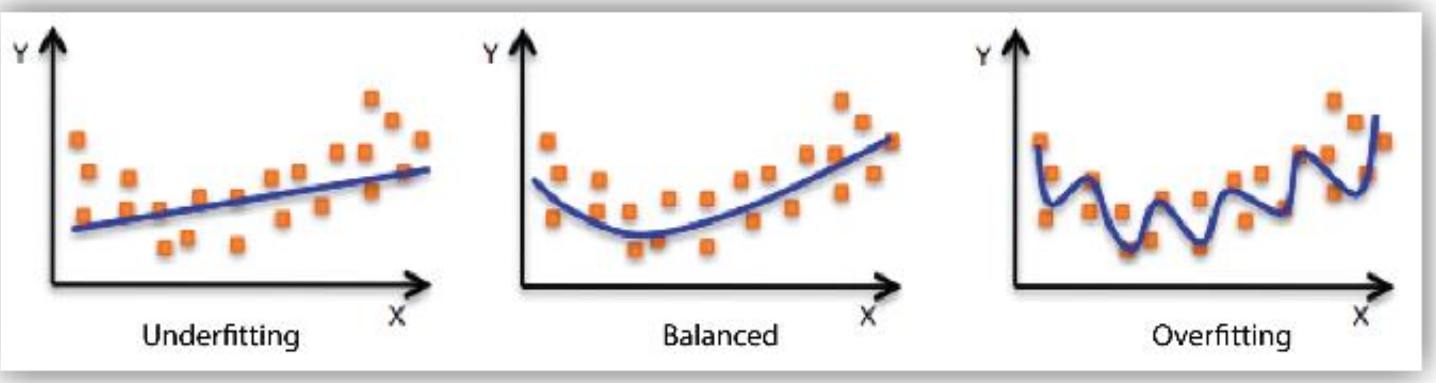
Context

High-dimensional problem is where the number of variables d is much larger than the number of observation n

1. High-dimensional problems: genetics, neauroscience
2. Usual least square estimator does not work!!

Regularization

- “True” mapping from x_i to y_i is complex.
  - Might need high-degree polynomial.
  - Might need to combine many features (don’t know “relevant” ones).
  - Complex models can overfit.
    - What do should do?
    - Reduce overfitting by regularizing model (constrain it)
      - Overfitting means you fit a too complex model to training data (curve is close to most observations) but actual prediction is very bad.
- Regularization: add a penalty on the complexity of model.
  - L2-regularization [Popular]



Regression:
Use overfitting and underfitting

Img: https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html

L_0-Penalty: Optimization

In regression analysis we want to avoid overfitting.

- It minimizes selection risk of variable and model selection:
  - L_0-penalized likelihood method is equivalent to the best subset selection.
  - Negative log-likelihood loss is equivalent to L_0-penalized likelihood.
  - Squared error (quadratic) loss is equivalent to L_0-penalized least squares.

- Ideally, balances between training error and number of features we use.
  - With λ=0, we get least squares with all features.
  - With λ=∞, we must set w=0 and not use any features.
  - With other λ, balances between training error and number of non-zeroes.
    - Larger λ puts more emphasis on having zeroes in 'w' (more selective important features).

We do not use it: computation is infeasible in high-dimensional statistical scenarios due to its nature of combinatorial optimization.

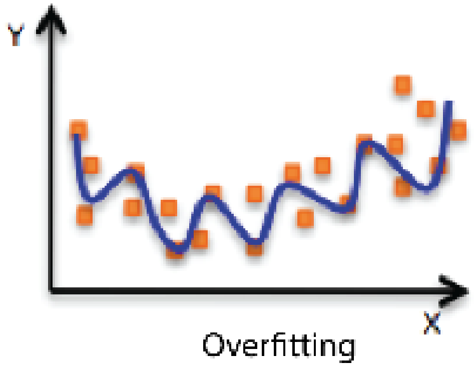
L2-Regularization

- Intuition: large slopes w_j tend to lead to overfitting.
  - Minimizes squared error plus penalty on L2-norm of ‘w’.
  - Balances getting low error vs. having small slopes ‘w_j’.
    - “Increase training error if it makes ‘w’ much smaller.”
    - Nearly-always reduces overfitting.

- Gradient measure how steep a slope or a line is.
- Gradients is calculated by dividing the vertical height by the horizontal distance.

- Regularization parameter λ > 0 controls “strength” of regularization.
  - Large λ puts large penalty on slopes.
  - L2 parameter norm penalty commonly known as weight decay / Ridge regression.

Lambda [scalar value] is called regularization rate



L2-Regularization

- Trade-off:
  - Increases training error.
  - Decreases approximation error.

- Forces algorithm to not only fit data but also keep model small weights.
- Regularization term should only be added to cost function during training.
- Hyperparameter α controls regularization of model.
  - If α = 0, just Linear Regression. [α range from 0.01 to 10]
  - Scale data before performing L2/Ridge Regression (StandardScaler)

- Drives weights closer to origin by adding a regularization term to objective function.
  - Tends to eliminate weights of least important features.

L2-Regularization

- How to choose λ?
  - Theory: as ‘n’ grows λ should be in the range O(1) to $(n^{1/2})$.
  - Practice: optimize validation set or cross-validation error.
  - This almost always decreases test error.

- Why use L2-Regularization?
  - “Almost always decreases test error”.
  - Solution ‘w’ is unique.
  - No collinearity issues.
  - Solution ‘w’ less sensitive to changes in X or y.
  - GD converges faster (bigger λ means fewer iterations).
  - Worst case: just set λ small and get the same performance.

L1/LASSO Regularization

- L1-regularization simultaneously regularizes and selects features.
  - Automatically performs feature selection and outputs a sparse model
  - α range from 0.001 to 0.1
  - Both ridge regression and LASSO work for high-dimensional problem but LASSO is GOOD --> it leads to a sparse estimator (many coefficients are 0).

L1-Regularization:

- Insensitive to data change.
- Decreased variance.
- Requires iterative solver.
- Solution is not unique.
- Many ‘w’ tend to be zero.

L2-Regularization:

- Insensitive to data change.
- Decreased variance.
- Closed-form solution.
- Solution is unique.
- All ‘w’ tend to be non-zero.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a},$$

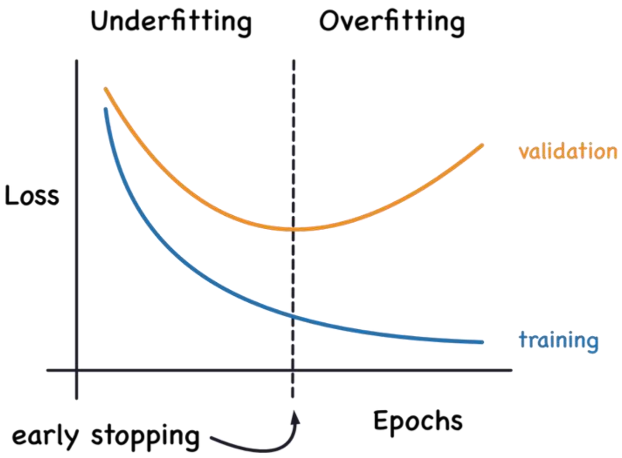
is a *closed form* of general quadratic equation $ax^2+bx+c = 0$

Early Stopping and Dropout

Early Stopping

- Regularization is crucial to neural network performance:
  - L2-regularization, early stopping, dropout.

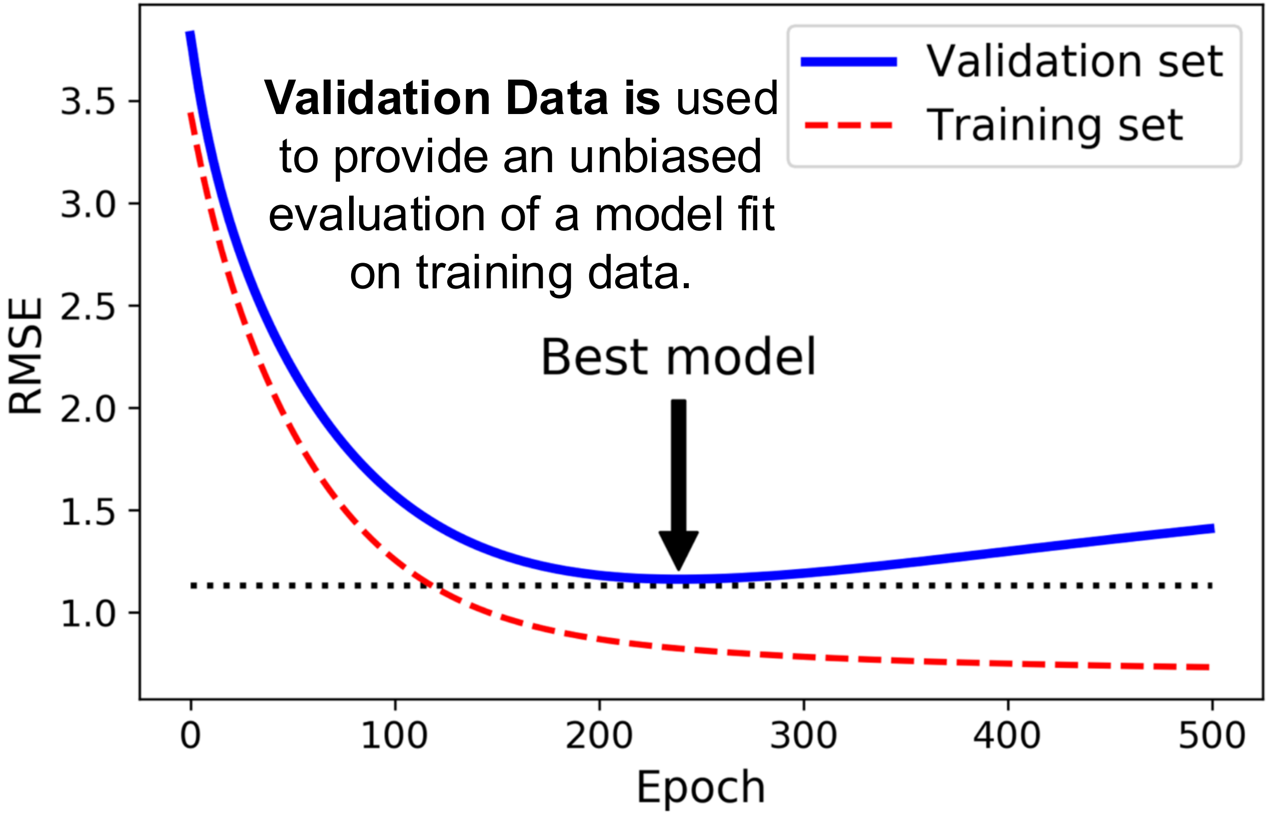
- Second common type of regularization is “early stopping”:
  - Monitor the validation error as we run Stochastic Gradient.
  - Stop algorithm if validation error starts increasing (overfitting data*).
  - Stop training as soon as validation error reaches minimum.



- Overfitting --> Model performs well on training data, but it does not generalize well.
- Underfitting --> Model is too simple to learn underlying structure of data.

Early Stopping Example

- A high-degree Polynomial Regression model being trained with GD.



- As epochs go by algorithm learns
- Its prediction error (RMSE) on training set goes down
- Prediction error on validation set is also going down.

Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow by Aurélien Géron, 2019.

Dropout*

- Popular regularization techniques for deep neural networks.
- NNs get a 1–2% accuracy boost simply by adding dropout.

- Working: At every training step, every input neurons has a probability (p) of being temporarily “dropped out.”
  - Means it will be entirely ignored during this training step, but it may be active during next step.
  - After training, neurons don’t get dropped anymore.

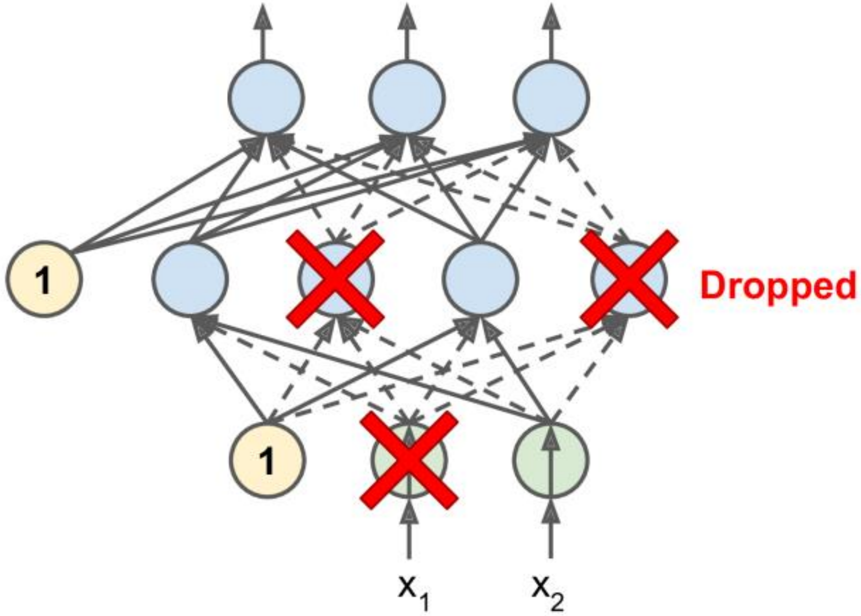
- Hyperparameter p is called dropout rate (set between 10% and 50%):
- E.g, RNN (closer to 20-30%), CNN(closer to 40-50%).

*Hinton, Geoffrey E., et al. "Improving neural networks by preventing co-adaptation of feature detectors." *arXiv preprint arXiv:1207.0580* (2012).

Dropout

- In this example: Apply dropout only to neurons in top one-to-three layers (excluding output layer).
  - Suppose p = 50%: During testing a neuron would be connected to twice as many input neurons as it would be (on average) during training.

- To compensate for this fact, rule of thumb:
  - Multiply each input connection weight by keep probability (1 – p) after training.
  - Alternatively, Divide each neuron’s output by keep probability during training.



Dropout

- Make sure to evaluate training loss after training.
  - If overfits training set: increase dropout rate.
  - If underfits training set: decreasing dropout rate.

- Dropout can be applied to hidden neurons
  - Between two hidden layers
  - Between last hidden layer and output layer.
- Dropout rate works as weight constraint on selected layers.

- Time-Effort trade-off: Well-tuned dropout significantly slows down convergence but results in a much better model.

BatchNorm

BatchNorm

- Vanishing gradients problem can be alleviated with better weight initialization, better optimizers or Batch Normalization[1].
- BatchNorm most successful architectural innovations in deep learning[2].
- BatchNorm aims to stabilize distribution (over a minibatch) of inputs to a given network layer during training.

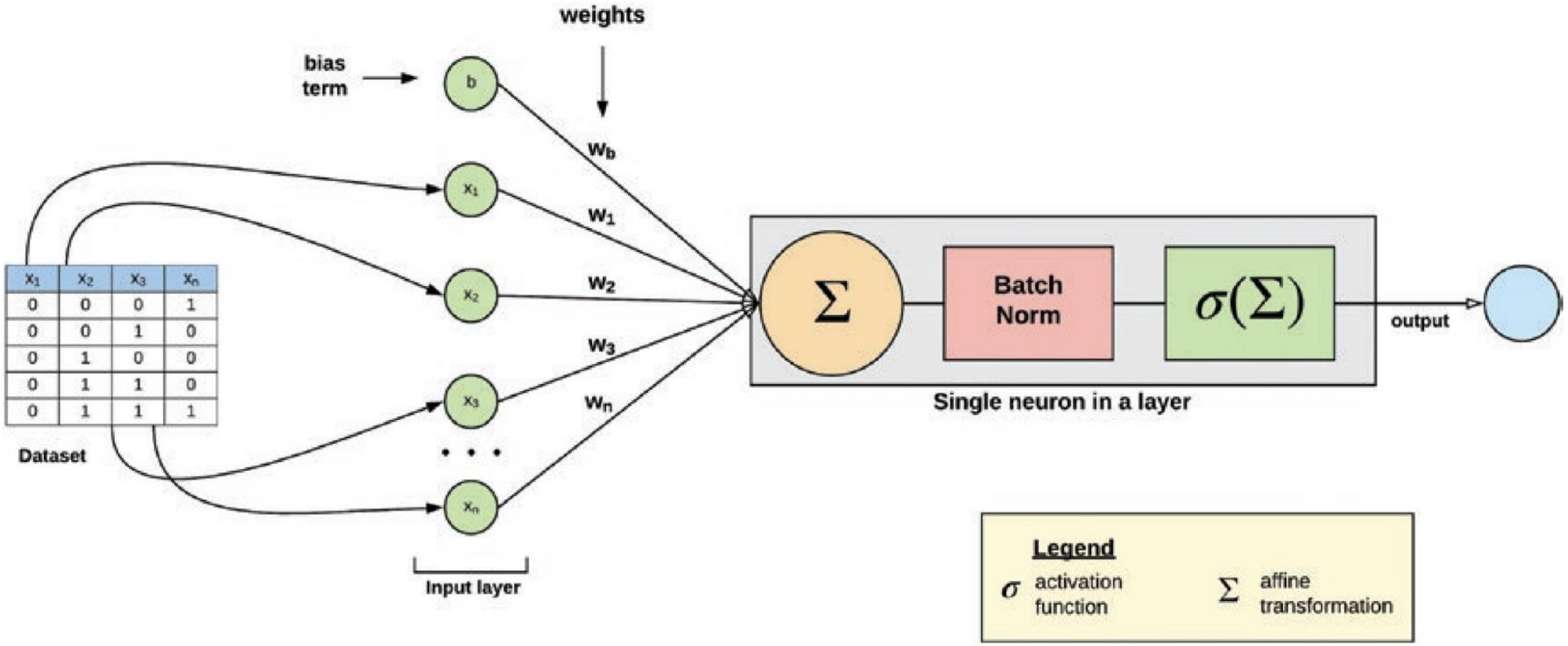
- BatchNorm Working: Operation lets model learn optimal scale and mean of each of layer's inputs.

1. First add an operation in model just before or after activation function of each hidden layer. This operation simply zero-center and normalizes each input.
2. Next, scales and shifts result using scaling and shifting vectors per layer.

[1] Sergey Ioffe and Christian Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," Proceedings of the 32nd International Conference on Machine Learning (2015): 448–456.

[2] Santurkar, Shibani, et al. "How does batch normalization help optimization?." *Advances in Neural Information Processing Systems*. 2018.

BatchNorm



x_1	x_2	x_3	x_n
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
0	1	1	1

- Note: If you add a BN layer as the very first layer of your NN, you do not need to standardize your training set; the BN layer will do it for you.
- Vanishing gradients problem can be reduced to a point that activation functions can be used for further solution.
- BN can improve many deep neural networks.

Bisong E. (2019) More on Optimization Techniques. In: Building Machine Learning and Deep Learning Models on Google Cloud Platform. Apress, Berkeley, CA

BatchNorm

- Batch Normalization is tricky to use in RNNs but sufficient for other nets.
  - Gradient Clipping* is often used RNNs to mitigate exploding gradients problem.
  - Gradient clipping does not help with vanishing gradients.
  - Gradient Clipping clips the gradients during backpropagation so that they never exceed some threshold.
- BN acts like a regularizer reducing need for other regularization techniques (such as dropout).
- BN adds runtime penalty to neural network.
  - Training is rather slow because each epoch takes much more time when BN in use.

*Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks." *International conference on machine learning*. 2013.