Training
Deep Learning models

Dr. Eran Raviv

July 2022

Agenda for today

- Recap from yesterday: what have we learned?

Agenda for today

- Recap from yesterday: what have we learned?
- Training neural network

Agenda for today

- Recap from yesterday: what have we learned?
- Training neural network
- Numerical Optimization

Agenda for today

- Recap from yesterday: what have we learned?
- Training neural network
- Numerical Optimization
- Tackle overfitting

Agenda for today

- Recap from yesterday: what have we learned?
- Training neural network
- Numerical Optimization
- Tackle overfitting
- First deep learning model

Quick recap

- Deep learning are variable-transformation systems

Quick recap

- Deep learning are variable-transformation systems
- They are highly nonlinear

Quick recap

- Deep learning are variable-transformation systems
- They are highly nonlinear
- They are highly flexible

Quick recap

- Deep learning are variable-transformation systems
- They are highly nonlinear
- They are highly flexible
- We went over the concepts of weights, layers and activation functions

Quick recap

- Deep learning are variable-transformation systems
- They are highly nonlinear
- They are highly flexible
- We went over the concepts of weights, layers and activation functions
- We created a tight link with the field of statistics

Training neural networks

Numerical computation in general

- Solve mathematical problems by updating the estimates of the solution via an iterative process - a recipe.

Numerical computation in general

- Solve mathematical problems by updating the estimates of the solution via an iterative process - a recipe.
- It all boils down to what computers can do best: compute, fast.

Numerical computation in general

- Solve mathematical problems by updating the estimates of the solution via an iterative process - a recipe.
- It all boils down to what computers can do best: compute, fast.
- Deep learning algorithms require high amount of numerical computation.

Numerical computation in general

- Solve mathematical problems by updating the estimates of the solution via an iterative process - a recipe.
- It all boils down to what computers can do best: compute, fast.
- Deep learning algorithms require high amount of numerical computation.
- Mainly because we can't close-form solve complicated, realistic problems.

A stroll down memory lane: derivatives

- If $z = f(y)$, the derivative $f'(y) := \frac{dz}{dy}$ helps us figure out how much the output of the function should change with a small (infinitesimal) ε nudge to the parameter y, i.e $f(y+\varepsilon) \approx f(y) + \varepsilon f'(y)$.

A stroll down memory lane: derivatives

- If $z = f(y)$, the derivative $f'(y) := \frac{dz}{dy}$ helps us figure out how much the output of the function should change with a small (infinitesimal) ε nudge to the parameter y, i.e $f(y+\varepsilon) \approx f(y) + \varepsilon f'(y)$.
- A gradient is a general notion of derivative when there is a vector of parameters (rather than a scaler).

A stroll down memory lane: derivatives

- If $z = f(y)$, the derivative $f'(y) := \frac{dz}{dy}$ helps us figure out how much the output of the function should change with a small (infinitesimal) ε nudge to the parameter y, i.e $f(y+\varepsilon) \approx f(y) + \varepsilon f'(y)$.
- A gradient is a general notion of derivative when there is a vector of parameters (rather than a scaler).
- For no special reason other than mathematical convention we use the notation of partial derivative with respect to each parameter like so

$$f'(parameter_i) := \frac{\partial function}{\partial parameter_i}.$$

A stroll down memory lane: chain rule

- We use the chain rule of calculus. If $y = g(x)$ and $z = f(g(x)) = f(y)$ then the derivative of z with respect to x is given by
$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

- In our context: the impact off parameter i on the loss function is found by first finding the impact of the activation function on the loss function and then the impact of the of the parameter on the activation function.

Reminder

- We need the Numerical optimization because we can't solve for the parameters using analytical formulae.

Reminder

- We need the Numerical optimization because we can't solve for the parameters using analytical formulae.
- Gradients make it easy for us to find a good solution.

Reminder

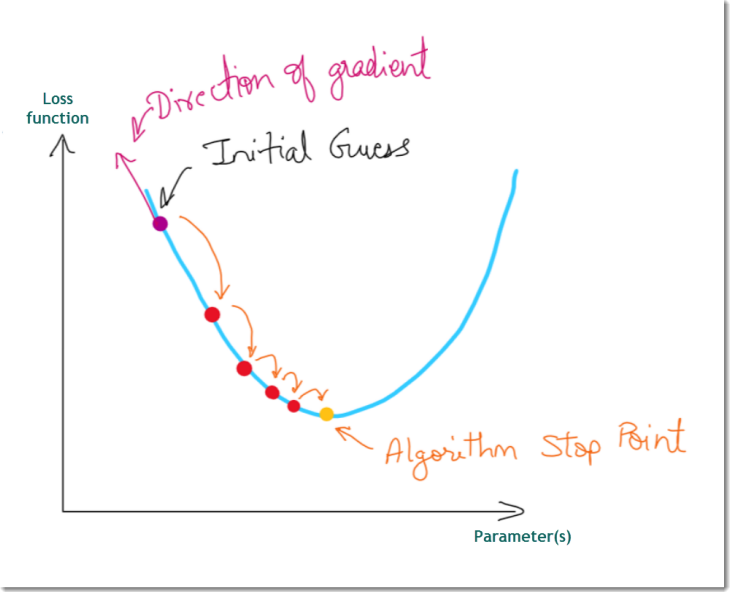
- We need the Numerical optimization because we can't solve for the parameters using analytical formulae.
- Gradients make it easy for us to find a good solution.
- The good solutions are just that. Good, but not optimal in any sense.

Reminder

- We need the Numerical optimization because we can't solve for the parameters using analytical formulae.
- Gradients make it easy for us to find a good solution.
- The good solutions are just that. Good, but not optimal in any sense.
- Over the years researchers found new tricks to progress faster towards a solution.

Enter: gradient based optimization

Intuition


Direction of gradient
Loss function
Initial Guess
Algorithm Stop Point
Parameter(s)

(batch) Gradient descent

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta \nabla_{\mathbf{w}} \mathcal{R}(\mathbf{w})$$

(batch) Gradient descent

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta \nabla_{\mathbf{w}} \mathcal{R}(\mathbf{w})$$

- k denotes the number of iteration. η denotes the learning rate.
  Reminder: $\mathcal{R}(\mathbf{w}) = \mathrm{E}\left(\mathcal{J}(\mathbf{w})\right) = \mathrm{E}(\frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\widehat{y}_i, y_i))$. Finally, $\nabla_{\mathbf{w}}$ denotes the gradient (the vector of partial derivatives. Each partial derivative with respect to a particular weight/parameter).
  What is the problem with having $\mathcal{R}$ there?

(batch) Gradient descent

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta \nabla_{\mathbf{w}} \mathcal{R}(\mathbf{w})$$

- k denotes the number of iteration. η denotes the learning rate.
  Reminder: $\mathcal{R}(\mathbf{w}) = \mathrm{E}\left(\mathcal{J}(\mathbf{w})\right) = \mathrm{E}(\frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\widehat{y}_i, y_i))$. Finally, $\nabla_{\mathbf{w}}$ denotes the gradient (the vector of partial derivatives. Each partial derivative with respect to a particular weight/parameter).
  What is the problem with having $\mathcal{R}$ there?
- We do what we can. The expectation is estimated using all available data (for now..).

(batch) Gradient descent

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta \nabla_{\mathbf{w}} \mathcal{R}(\mathbf{w})$$

- k denotes the number of iteration. η denotes the learning rate.
  Reminder: $\mathcal{R}(\mathbf{w}) = \mathrm{E}\left(\mathcal{J}(\mathbf{w})\right) = \mathrm{E}(\frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\widehat{y}_i, y_i))$. Finally, $\nabla_{\mathbf{w}}$ denotes the gradient (the vector of partial derivatives. Each partial derivative with respect to a particular weight/parameter).
  What is the problem with having $\mathcal{R}$ there?
- We do what we can. The expectation is estimated using all available data (for now..).
- Why the minus in the formula?

(batch) Gradient descent (continued)

So

$$\mathbf{w}_{(k+1)} = \mathbf{w}_{(k)} - \eta \nabla_{\mathbf{w}} \widehat{\mathrm{E}}$$

(batch) Gradient descent (continued)

So

$$\mathbf{w}_{(k+1)} = \mathbf{w}_{(k)} - \eta \nabla_{\mathbf{w}} \widehat{\mathrm{E}}$$

- Each "nudge" requires computation of the entire data for evaluating $\nabla_{\mathbf{w}} \mathcal{L}(\widehat{y}_i, y_i)$. This is known as *batch* gradient descent.

(batch) Gradient descent (continued)

So

$$\mathbf{w}_{(k+1)} = \mathbf{w}_{(k)} - \eta \nabla_{\mathbf{w}} \widehat{\mathrm{E}}$$

- Each "nudge" requires computation of the entire data for evaluating $\nabla_{\mathbf{w}} \mathcal{L}(\widehat{y}_i, y_i)$. This is known as *batch* gradient descent.
- Note that $\widehat{y}_i$ is a function of the input $\mathbf{x}$ and of course the weights $\mathbf{w}$. In our text book they use (adhering to their notation) they use the more exhaustive

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L\left(\boldsymbol{x}^{(i)}, y^{(i)}, \boldsymbol{\theta}\right)$$

(batch) Gradient descent (continued)

- Using all the data is ideal in terms of accuracy, but it is also computationally expensive. It is rarely used in practical model-training.

(batch) Gradient descent (continued)

- Using all the data is ideal in terms of accuracy, but it is also computationally expensive. It is rarely used in practical model-training.
- We can trade accuracy for computational cost. 

(batch) Gradient descent (continued)

- Using all the data is ideal in terms of accuracy, but it is also computationally expensive. It is rarely used in practical model-training.
- We can trade accuracy for computational cost. 
- Enter *stochastic gradient descent*.

Stochastic gradient descent

- AKA On-line gradient descent, or sequential gradient descent.

Stochastic gradient descent

- AKA On-line gradient descent, or sequential gradient descent.
- Easy. Pick a data point at random and update the weight vector based on the gradient with all calculation based on that data point only.

Stochastic gradient descent

- Quite intuitive.

Stochastic gradient descent

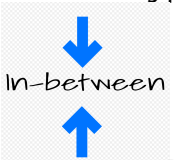
- Quite intuitive.
- We are saving enormous amount of computation. The loss function is computed only based on one data point (indeed: for all parameters).

Stochastic gradient descent

- Quite intuitive.
- We are saving enormous amount of computation. The loss function is computed only based on one data point (indeed: for all parameters).
- Direction of update not very stable though. Teaser: what influence the stability of this choice? for which data this choice is ok, and for which it is unwise?

Stochastic gradient descent

- Quite intuitive.
- We are saving enormous amount of computation. The loss function is computed only based on one data point (indeed: for all parameters).
- Direction of update not very stable though. Teaser: what influence the stability of this choice? for which data this choice is ok, and for which it is unwise?
- We have seen *batch* and *stochastic*. Typically we choose something



Enter mini-batch gradient descent

- Between batch gradient descent and stochastic gradient descent.

Enter mini-batch gradient descent

- Between batch gradient descent and stochastic gradient descent.
- Stochastic gradient descent is a special case of mini-batch gradient descent with batch size $= 1$.

Enter mini-batch gradient descent

- Between batch gradient descent and stochastic gradient descent.
- Stochastic gradient descent is a special case of mini-batch gradient descent with batch size $= 1$.
- More computationally efficient than the batch gradient descent, and not as noisy as the fully stochastic version.

Enter mini-batch gradient descent

- Between batch gradient descent and stochastic gradient descent.
- Stochastic gradient descent is a special case of mini-batch gradient descent with batch size $= 1$.
- More computationally efficient than the batch gradient descent, and not as noisy as the fully stochastic version.
- The size of the mini-batch is another hyper-parameter to consider.

Enter mini-batch gradient descent

- Between batch gradient descent and stochastic gradient descent.
- Stochastic gradient descent is a special case of mini-batch gradient descent with batch size $= 1$.
- More computationally efficient than the batch gradient descent, and not as noisy as the fully stochastic version.
- The size of the mini-batch is another hyper-parameter to consider.
- In both stochastic and mini-batch version of the algorithms we pass through all the data. However, the mini-batch version benefits for algebraic practical implementation (vectorization) which allows gradient computing in parallel.

Minibatch size, and few comments

Two extremes:

- $M \Rightarrow$ Batch gradient descent. High computational cost (ok if you have small data).

Minibatch size, and few comments

Two extremes:

- $M \Rightarrow$ Batch gradient descent. High computational cost (ok if you have small data).
- $1 \Rightarrow$ Stochastic gradient descent. Noisy.

Minibatch size, and few comments

Two extremes:

- $M \Rightarrow$ Batch gradient descent. High computational cost (ok if you have small data).
- $1 \Rightarrow$ Stochastic gradient descent. Noisy.
- In practice it would be somewhere in between. Values of $2^{\{6,7,8,9\}}$ are common.

Minibatch size, and few comments

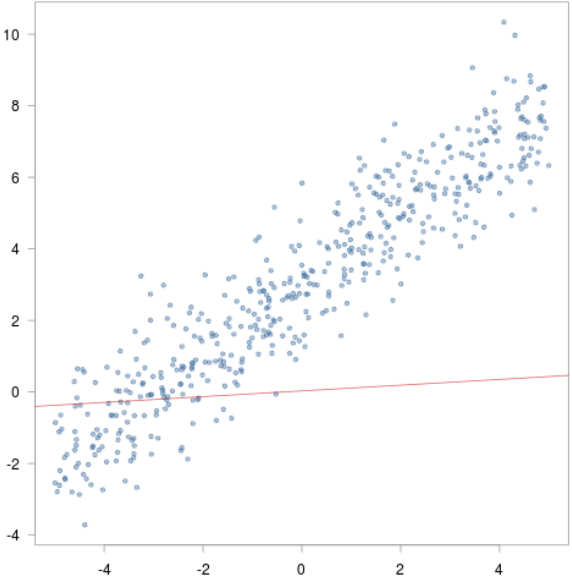
Two extremes:

- $M \Rightarrow$ Batch gradient descent. High computational cost (ok if you have small data).
- $1 \Rightarrow$ Stochastic gradient descent. Noisy.
- In practice it would be somewhere in between. Values of $2^{\{6,7,8,9\}}$ are common.
- The specialized language is a bit confusing. I would not use the word "batch" to indicate that the whole sample is used +, minibatch is also a stochastic algorithm. However, we (you) should adhere to the convention.

Numerical computation - putting it all together

1. Initialize the weights.
2. Forward propagate: the inputs from a training set are passed through the neural network and an output is computed.
3. You define an error function. It captures the difference between the actual output and your model's output, given current model weights.
4. Backpropagation - change the weights for the neurons, in order to reduce the error.
5. Update - weights are changed according to the results of the previous step.
6. **Iterate** - because the weights are only nudged at each step, many iterations are required in order for the network to learn. The amount of iterations needed to converge depends on the learning rate, the network architecture, and the specific optimization method used.

Illustration:



Accelerated gradient methods

Accelerated gradient methods

- Training is the process by which we reach a satisfactory solution.

Accelerated gradient methods

- Training is the process by which we reach a satisfactory solution.
- The landscapes of optimization regions are often described as valleys, trenches, canals and ravines - it is NOT a smooth terrain to traverse.

Accelerated gradient methods

- Training is the process by which we reach a satisfactory solution.
- The landscapes of optimization regions are often described as valleys, trenches, canals and ravines - it is NOT a smooth terrain to traverse.
- With many parameters, training is slow.

Accelerated gradient methods

- Training is the process by which we reach a satisfactory solution.
- The landscapes of optimization regions are often described as valleys, trenches, canals and ravines - it is NOT a smooth terrain to traverse.
- With many parameters, training is slow.
- Over time researchers found ways to speed up the convergence towards a solution.

Accelerated gradient methods

- Training is the process by which we reach a satisfactory solution.
- The landscapes of optimization regions are often described as valleys, trenches, canals and ravines - it is NOT a smooth terrain to traverse.
- With many parameters, training is slow.
- Over time researchers found ways to speed up the convergence towards a solution.
- We now review few seminal contributions.

Learning rate decay

- Initial learning rates should be high but reduce over time (You et al., 2019).

Learning rate decay

- Initial learning rates should be high but reduce over time (You et al., 2019).
- Start with large steps and decrease them over time.

Learning rate decay

- Initial learning rates should be high but reduce over time (You et al., 2019).
- Start with large steps and decrease them over time.
- The two most common decay functions are exponential decay and inverse decay (e.g. Duchi et al. 2011).

Learning rate decay

- Initial learning rates should be high but reduce over time (You et al., 2019).
- Start with large steps and decrease them over time.
- The two most common decay functions are exponential decay and inverse decay (e.g. Duchi et al. 2011).
- Same step size for all parameters (still).

Gradient descent with momentum

- Should all parameters enjoy the same "nudge" pace? **No.**

Gradient descent with momentum

- Should all parameters enjoy the same "nudge" pace? **No.**
- An early idea by Jacobs (1988) suggests to adapt the learning rate per parameter, based on the sign of the gradient.

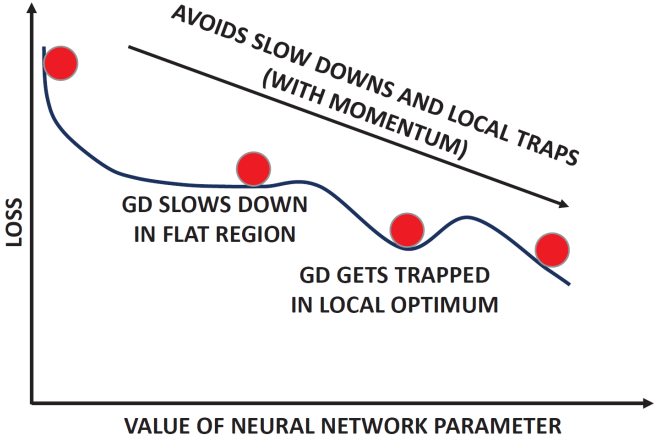
Gradient descent with momentum

- Should all parameters enjoy the same "nudge" pace? **No.**
- An early idea by Jacobs (1988) suggests to adapt the learning rate per parameter, based on the sign of the gradient.
- But we can do better.

Gradient descent with momentum

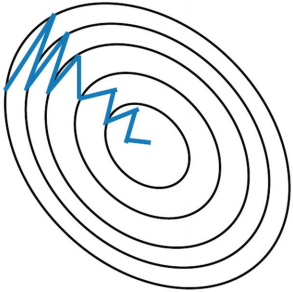
- Should all parameters enjoy the same "nudge" pace? **No.**
- An early idea by Jacobs (1988) suggests to adapt the learning rate per parameter, based on the sign of the gradient.
- But we can do better.
- We now explain the intuition behind adjusting the learning rate based on momentum.

Gradient descent with momentum (continued)



Gradient descent with momentum (continued)

Stochastic Gradient Descent **withhout** Momentum



Stochastic Gradient Descent **with** Momentum

Gradient descent with momentum (continued)

- We introduce another hyperparameter β, say $\beta = 0.9$. It controls the speed of learning rate adjustment.

Gradient descent with momentum (continued)

- We introduce another hyperparameter β, say $\beta = 0.9$. It controls the speed of learning rate adjustment.
- We give preference to direction where change is consistent.

Gradient descent with momentum (continued)

- We introduce another hyperparameter β, say $\beta = 0.9$. It controls the speed of learning rate adjustment.
- We give preference to direction where change is consistent.
- Opposite changes cancel each other in the moving average calculation.

Gradient descent with momentum (continued)

- We introduce another hyperparameter β, say $\beta = 0.9$. It controls the speed of learning rate adjustment.
- We give preference to direction where change is consistent.
- Opposite changes cancel each other in the moving average calculation.
- Implication: less zig-zagging.

Gradient descent with momentum (continued)

How does it work?

- Instead of simply updating:

$$\mathbf{w}_{k+1} = \mathbf{w}_t - \eta \nabla_{\mathbf{w}} \mathcal{L}$$

We now use

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \nu_{k+1},$$

where

$$\nu_{k+1} = \beta \nu_k + \eta \nabla_{\mathbf{w}} \mathcal{L}$$

Note I have remove dependency on data and parameters in the loss function, to ease readabiliy

Gradient descent with momentum (continued)

- Momentum increases speed when the cost surface is "ugly" because it damps the size of the steps along directions of high curvature. Thus yielding a larger *effective learning rate* along the directions of low curvature.

Gradient descent with momentum (continued)

- Momentum increases speed when the cost surface is "ugly" because it damps the size of the steps along directions of high curvature. Thus yielding a larger *effective learning rate* along the directions of low curvature.
- A nice reference here (but also nice in general) is Hinton 2012.

Adagrad

- Adding some friction: use the squared, element-wise, derivatives as a way to adjust the learning rate. Slow down if the speed is "too fast", and increase speed when it's "too slow".

$\tilde{\nu}$ to make a differentiate it from the ν notation we use for momentum.

Adagrad

- Adding some friction: use the squared, element-wise, derivatives as a way to adjust the learning rate. Slow down if the speed is "too fast", and increase speed when it's "too slow".
- Instead of updating using the moving average of the gradient, we "nudge" the parameters like so:

$$w_{k+1,i} = w_{k,i} - \eta \frac{\nabla_{\mathbf{w}} \mathcal{L}_i}{\sqrt{\tilde{\nu}_{k,i} + \varepsilon}},$$

where $\tilde{\nu}_i$ is a sum of squared partial derivatives for parameter i up to iteration k.

$\tilde{\nu}$ to make a differentiate it from the ν notation we use for momentum.

Adagrad

- Adding some friction: use the squared, element-wise, derivatives as a way to adjust the learning rate. Slow down if the speed is "too fast", and increase speed when it's "too slow".
- Instead of updating using the moving average of the gradient, we "nudge" the parameters like so:

$$w_{k+1,i} = w_{k,i} - \eta \frac{\nabla_{\mathbf{w}} \mathcal{L}_i}{\sqrt{\tilde{v}_{k,i} + \varepsilon}},$$

  where $\tilde{v}_i$ is a sum of squared partial derivatives for parameter i up to iteration k.
- So if that sum is large, we don't high learning rate, while when the sum is small, we need to increase the learning rate.

$\tilde{v}$ to make a differentiate it from the v notation we use for momentum.

Adagrad

- Adding some friction: use the squared, element-wise, derivatives as a way to adjust the learning rate. Slow down if the speed is "too fast", and increase speed when it's "too slow".
- Instead of updating using the moving average of the gradient, we "nudge" the parameters like so:

$$w_{k+1,i} = w_{k,i} - \eta \frac{\nabla_{\mathbf{w}} \mathcal{L}_i}{\sqrt{\tilde{v}_{k,i} + \varepsilon}},$$

  where $\tilde{v}_i$ is a sum of squared partial derivatives for parameter i up to iteration k.
- So if that sum is large, we don't high learning rate, while when the sum is small, we need to increase the learning rate.
- $\tilde{v}$ to make it differentiate it from this v notation we used for momentum.

What is the problem with this algorithm?

RMS (Root Mean Square) prop*

The most straight forward extension for Adagrad you can think of.

- Before (with Adagrad):

$$\tilde{v}_i = \sum_{j=1}^{k} \left(\frac{\partial \mathcal{L}}{\partial w_i} \right)_j^2$$

"slide 29 in lecture 6"

RMS (Root Mean Square) prop*

The most straight forward extension for Adagrad you can think of.

- Before (with Adagrad):

$$\tilde{v}_i = \sum_{j=1}^{k} \left(\frac{\partial \mathcal{L}}{\partial w_i} \right)_j^2$$

- Now, forget iteration which are too far back:

$$\tilde{v}_{k+1,i} = \tilde{\beta} \tilde{v}_{k,i} + (1 - \tilde{\beta}) \left(\frac{\partial \mathcal{L}}{\partial w_i} \right)_{k+1}^2$$

"slide 29 in lecture 6"

Adam - adaptive moment estimation

- Combines previous good ideas into one algorithm. MA of past gradients (as in momentum), *and* MA of the past squared gradient (as in RMSprop).

Adam - adaptive moment estimation

- Combines previous good ideas into one algorithm. MA of past gradients (as in momentum), *and* MA of the past squared gradient (as in RMSprop).
- So (back to vector notation):

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \frac{\nu_{k+1}}{\sqrt{\tilde{\nu}_{k+1}} + \varepsilon},$$

where we now have both ν and $\tilde{\nu}$ involved.

Adam - adaptive moment estimation

- Combines previous good ideas into one algorithm. MA of past gradients (as in momentum), *and* MA of the past squared gradient (as in RMSprop).
- So (back to vector notation):

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \frac{\nu_{k+1}}{\sqrt{\tilde{\nu}_{k+1}} + \varepsilon},$$

  where we now have both ν and $\tilde{\nu}$ involved.
- A most popular variant.

Numerical prompts

- Normalizing inputs to speed the training process.
  Batch normalization is a common practice where you standardize the output of one layer, before passing it to the next layer.

Numerical prompts

- Normalizing inputs to speed the training process.
  Batch normalization is a common practice where you standardize the output of one layer, before passing it to the next layer.
- Weight Initialization.

Numerical prompts

- Normalizing inputs to speed the training process.
  Batch normalization is a common practice where you standardize the output of one layer, before passing it to the next layer.
- Weight Initialization.
- Gradient Checking.

Next topic of today:

Regularization

Regularization as a concept

- The goal of regularization is to enable better generalization of the model.

Regularization as a concept

- The goal of regularization is to enable better generalization of the model.
- Instead of trying to be right on the first go, we overshot and scale back complexity.

Regularization as a concept

- The goal of regularization is to enable better generalization of the model.
- Instead of trying to be right on the first go, we overshot and scale back complexity.
- There are many strategies to do that.

Regularization as a concept

- The goal of regularization is to enable better generalization of the model.
- Instead of trying to be right on the first go, we overshot and scale back complexity.
- There are many strategies to do that.
- A good regularizer $\Rightarrow$ <u>reduces variance</u> without increasing bias, much.

Regularization as a concept

- When do we need to use regularization?

Regularization as a concept

- When do we need to use regularization?
- Why regularization helps us?

Regularization as a concept

- When do we need to use regularization?
- Why regularization helps us?
    - Prevents model's over-flexibility $\Rightarrow$ less expressive

Regularization as a concept

- When do we need to use regularization?
- Why regularization helps us?
  - Prevents model's over-flexibility $\Rightarrow$ less expressive
  - Equivalent to "contaminating" the training data (helping us to generalize on new data).

Bias variance tradeoff

The MSE can be decomposed like so:

$$\mathcal{MSE} = \mathcal{E}_x \left\{ \text{Bias}_{\mathbf{w}}[\hat{f}(x; \mathbf{w})]^2 + \text{Var}_{\mathbf{w}}[\hat{f}(x; \mathbf{w})] \right\} + \sigma^2$$

Bias variance tradeoff

The MSE can be decomposed like so:

$$\mathcal{MSE} = \mathcal{E}_x \left\{ \text{Bias}_{\mathbf{w}}[\hat{f}(x; \mathbf{w})]^2 + \text{Var}_{\mathbf{w}}[\hat{f}(x; \mathbf{w})] \right\} + \sigma^2$$

- We can't do anything about σ^2

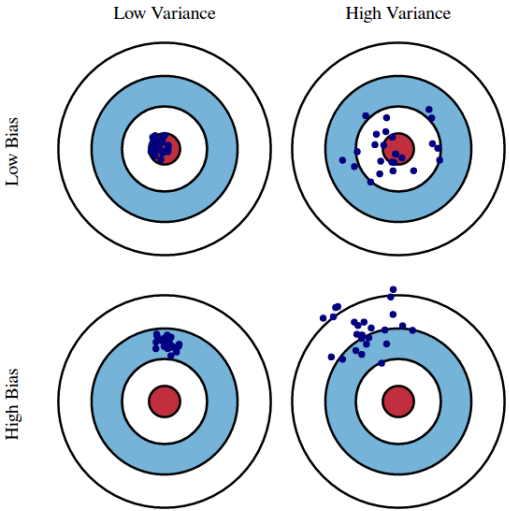
Bias variance tradeoff

The MSE can be decomposed like so:

$$\mathcal{MSE} = \mathcal{E}_x \left\{ \text{Bias}_{\mathbf{w}}[\hat{f}(x; \mathbf{w})]^2 + \text{Var}_{\mathbf{w}}[\hat{f}(x; \mathbf{w})] \right\} + \sigma^2$$

- We can't do anything about σ^2
- We can trade bias for variance. And more relevant: the reverse.

Bias variance tradeoff



We now review few ways to regularize deep learning models

Regularization - Data augmentation

- A paradigm shift from old-school econometrics.

Regularization - Data augmentation

- A paradigm shift from old-school econometrics.
- Rotate the data.

Regularization - Data augmentation

- A paradigm shift from old-school econometrics.
- Rotate the data.
- Add data which was generated synthetically.

Regularization - norm restrictions

$$\tilde{\mathcal{L}}(\mathbf{w}; \boldsymbol{X}, \boldsymbol{y}) = \mathcal{L}(\mathbf{w}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\mathbf{w}),$$

$\Omega(\cdot)$ is usually a quadratic norm penalty. E.g $\Omega(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2^2$. ($L^2$ regularization, or Tikhonov regularization).

Regularization - norm restrictions

$$\tilde{\mathcal{L}}(\mathbf{w}; \boldsymbol{X}, \boldsymbol{y}) = \mathcal{L}(\mathbf{w}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\mathbf{w}),$$

$\Omega(\cdot)$ is usually a quadratic norm penalty. E.g $\Omega(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2^2$. ($L^2$ regularization, or Tikhonov regularization).

- Another common penalty is the L^1 regularization: $\Omega(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_1 = \sum_i |w_i|$

Regularization - norm restrictions

$$\tilde{\mathcal{L}}(\mathbf{w}; \boldsymbol{X}, \boldsymbol{y}) = \mathcal{L}(\mathbf{w}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\mathbf{w}),$$

$\Omega(\cdot)$ is usually a quadratic norm penalty. E.g $\Omega(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_2^2$. ($L^2$ regularization, or Tikhonov regularization).

- Another common penalty is the L^1 regularization: $\Omega(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|_1 = \sum_i |w_i|$
- What is the difference?

Regularization - norm restrictions (continued)

- α is the hyperparameter controlling the amount of shrinkage.

Regularization - norm restrictions (continued)

- α is the hyperparameter controlling the amount of shrinkage.
- Sometimes it is referred to as weight decay.

Regularization - norm restrictions (continued)

- α is the hyperparameter controlling the amount of shrinkage.
- Sometimes it is referred to as weight decay.
- $\alpha = 0 \quad \Rightarrow$ no regularization.

Regularization - norm restrictions (continued)

- α is the hyperparameter controlling the amount of shrinkage.
- Sometimes it is referred to as weight decay.
- $\alpha = 0 \quad \Rightarrow$ no regularization.
- We can keep $\alpha_l = \alpha$ (l for layer), but we don't have to.

Regularization - early stopping

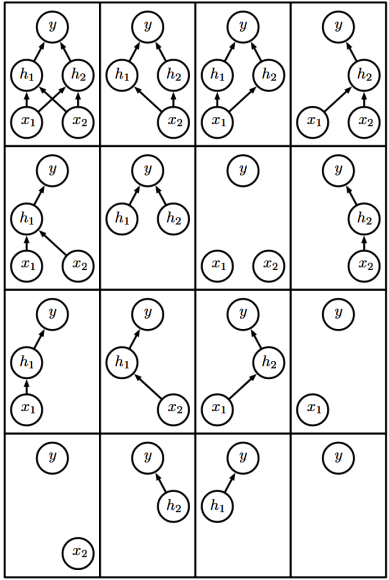
- The number of training steps is just another hyperparameter.

Regularization - early stopping

- The number of training steps is just another hyperparameter.
- If η is the learning rate and we have say τ training iteration then the product $\eta \times \tau$ is the effective capacity of the model.

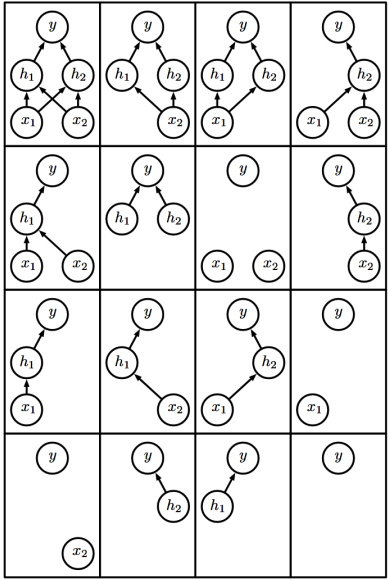
Dropout

- Powerful regularization technique.



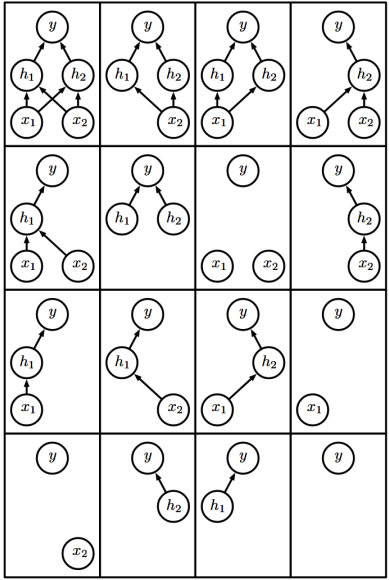
Dropout

- Powerful regularization technique.
- Can be thought of as model averaging.



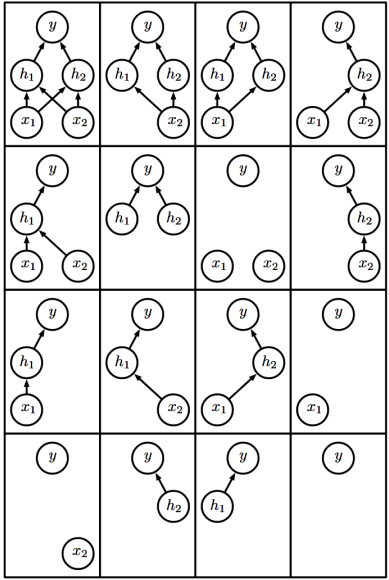
Dropout

- Powerful regularization technique.
- Can be thought of as model averaging.
- The weights can't "rely" on inputs to be there.



Dropout

- Powerful regularization technique.
- Can be thought of as model averaging.
- The weights can't "rely" on inputs to be there.
- The probability of masking a neuron doesn't each layer.



Now Let's code!

References

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research, 12(Jul):2121–2159.

Hinton, G. E. (2012). A practical guide to training restricted boltzmann machines. In Neural networks: Tricks of the trade, pages 599–619. Springer.

Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. Neural networks, 1(4):295–307.

You, K., Long, M., Wang, J., and Jordan, M. I. (2019). How does learning rate decay help modern neural networks? arXiv preprint arXiv:1908.01878.