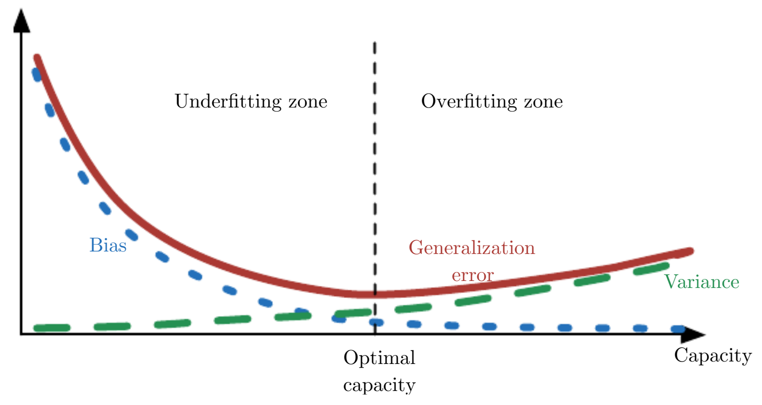
# Regularization



- Bias how much does expected value of estimator differ from true value

- Variance is the variation of the estimator

# Introduction

- any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error
  - put extra constraints on a machine learning model, such as adding restrictions on the parameter values.
  - add extra terms in the objective function that can be thought of as corresponding to a soft constraint on the parameter values.
  - Or ensemble methods combine multiple hypotheses that explain the training data.
- Constraints and penalties are designed to encode specific kinds of prior knowledge.
- Or these constraints and penalties are designed to express a generic preference for a simpler model class in order to promote generalization.
- Sometimes penalties and constraints are necessary to make an underdetermined problem determined.

# Introduction (2)

- Deep learning algorithms are typically applied to extremely complicated domains such as images, audio sequences and text, for which the true generation process essentially involves simulating the entire universe.
  - controlling the complexity of the model is not a simple matter of finding the model of the right size, with the right number of parameters.
  - the best fitting model (in the sense of minimizing generalization error) is a large model that has been regularized appropriately.
  - A generic observation re deep learning models is that among parameters we penalise weights not biases.

# Optimization

Hessian also can be used for second derivative tests: its eigenvalues all positives means minima

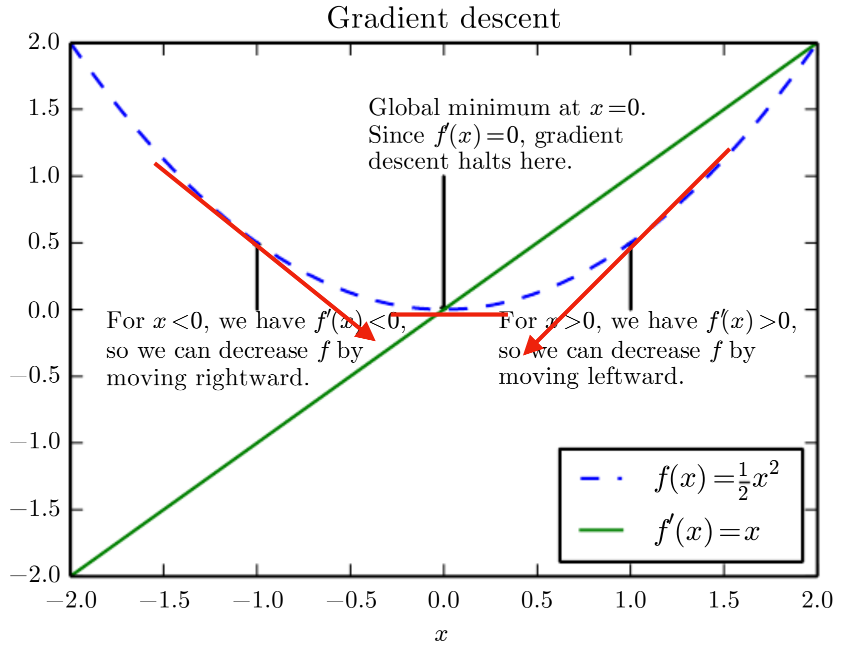
- First derivative finds the tangent to the curve
  - Zero implies that tangent has no slope, I.e. its a maxima, minima, saddle point
  - Multiple dimensions a matrix of partial derivative along each dimensions known as Jacobian
- Second derivative finds the tangent to the first derivative (measures curvature)
  - Positive second derivative implies local minima
  - Negative implies local maxima
  - Multiple dimensions a matrix known as Hessian
- Optimization algorithms such as gradient descent that use only the Jacobian are called first-order optimization algorithms.

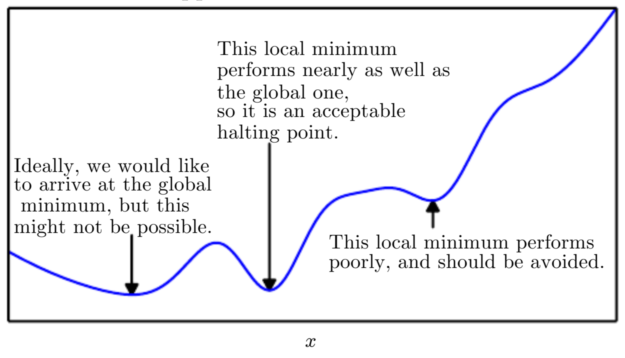
$$\boldsymbol{x}' = \boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$$

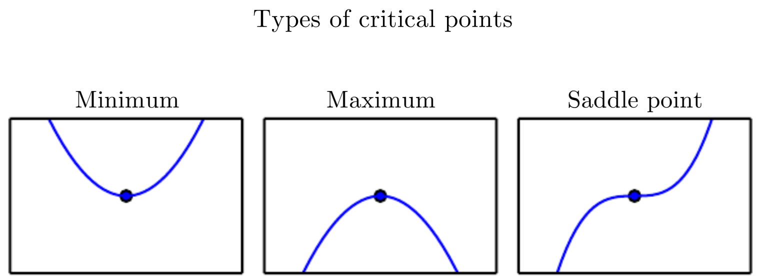
- Optimization algorithms such as Newton's method that also use the Hessian matrix are called second-order optimization algorithms (g is gradient, H is Hessian, Taylor series approximation optimization)

$$\epsilon^* = \frac{\boldsymbol{g}^\top \boldsymbol{g}}{\boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}}.$$

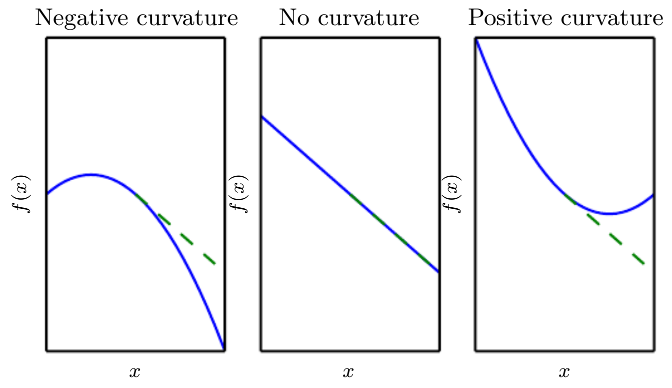
# Gradient Descent







## First derivative



## Second derivative

Quadratic functions with various curvature. The dashed line indicates the value of the cost function we would expect based on the gradient information alone as we make a gradient step downhill. In the case of negative curvature, the cost function actually decreases faster than the gradient predicts. In the case of no curvature, the gradient predicts the decrease correctly. In the case of positive curvature, the function decreases slower than expected and eventually begins to increase, so too large of step sizes can actually increase the function inadvertently.

## Parameter Norm Penalties

- Generic formulation

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta})$$

- L2

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^\top \boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}),$$

$$\boldsymbol{w} \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon\nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}).$$

- Multiplicatively shrinking the weights at each update
- Over the course of training the effect of weight decay is to rescale weight along axes defined by the Hessian.
  - Only directions along which the parameters contribute significantly to reducing the objective function directions that do not contribute to reducing the objective function, a small eigenvalue of the Hessian will not significantly increase the gradient so these weights are reduced towards zero.
- For linear regression we can find that over course of training

$$\boldsymbol{w} = (\boldsymbol{X}^\top \boldsymbol{X} + \alpha \boldsymbol{I})^{-1} \boldsymbol{X}^\top \boldsymbol{y}.$$

Adding variance for linear regression case which would make it shrink weights on features with low covariance with output

# Parameter Norm Penalties

- L1

$$\Omega(\theta) = \sum^{i} |w_i|$$

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha||\boldsymbol{w}||_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}),$$

$$w \leftarrow (1 - \epsilon\alpha) sign(w) - \epsilon \nabla_w J(w; X, y)$$

- Also scales with Hessian eigenvalues but in this case makes weight more sparse, i.e. explicitly makes weights zero
    - Some features can be discarded

# Constrained Optimization as norm penalties

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\Omega(\boldsymbol{\theta}).$$

- Constrained optimisation where we constrain each $\Omega(\theta)$ is less than *k* can be converted into unconstrained using Lagrange multiplier

$$\mathcal{L}(\boldsymbol{\theta}, \alpha; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha(\Omega(\boldsymbol{\theta}) - k).$$

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \alpha^*) = \arg\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + \alpha^*\Omega(\boldsymbol{\theta}).$$

- \alpha^* is optimum Lagrange multipler
- Problem now is same as previous minimisation
- Previous problem is a specific instance of constrained optimisation where for example L2 regulariser weight are constrained to lie in unit ball around origin
- While norm penalties may make weights zero, constrained optimisation may check for only where constraints are violated and project back into constrained region.

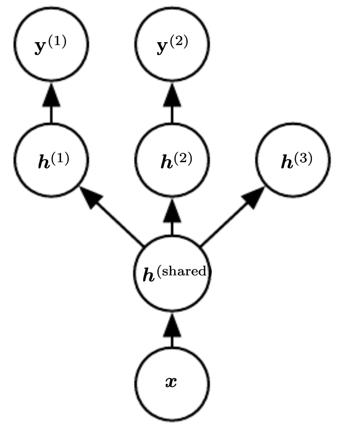
# Dataset Augmentation

- Best way to generalise is to train on more data
  - Create fake data
    - Translation, rotation, colour modifications
    - Add noise to input
      - For some models, the addition of noise with infinitesimal variance at the input of the model is equivalent to imposing a penalty on the norm of the weights

.

# Multi task learning

- Shared layers on multi-task learning
    - Cross Entropy Loss
    - Siamese/Ranking loss, measuring similarity of embeddings



## Early Stopping

- When training large models with sufficient representational capacity to overfit the task, we often observe that training error decreases steadily over time, but validation set error begins to rise again.
- Instead of running our optimization algorithm until we reach a (local) minimum of validation error, we run it until the error on the validation set has not improved for some amount of time.
- Every time the error on the validation set improves, we store a copy of the model parameters.
- **Extra credit question, derive eqn 7.45 starting from eqn 7.1**

# Ensemble methods

- idea is to train several different models separately, then have all of the models vote on the output for test examples. This is an example of a general strategy in machine learning called model averaging.

- The reason that model averaging works is that different models will usually not make all the same errors on the test set.

- To a first approximation, dropout can be thought of as a method of making bagging practical for ensembles of very many large neural networks. Bagging involves training multiple models, and evaluating multiple models on each test example.

  - training and evaluating such networks is costly in terms of runtime and memory.

- Dropout provides an inexpensive approximation to training and evaluating a bagged ensemble of exponentially many neural networks.

- to learn with bagging, we define k different models, construct *k* different datasets by sampling from the training set with replacement, and then train model *i* on dataset *i*.

# Dropout

- Dropout aims to approximate this process, but with an exponentially large number of neural networks. Specifically, to train with dropout, we use a minibatch-based learning algorithm that makes small steps, such as stochastic gradient descent.
- Each time we load an example into a minibatch, we randomly sample a different binary mask to apply to all of the input and hidden units in the network.
- The mask for each unit is sampled independently from all of the others.
- The probability of sampling a mask value of one (causing a unit to be included) is a hyperparameter fixed before training begins.
- We then run forward propagation, back-propagation, and the learning update as usual.

.

# Dropout

- Dropout training is not quite the same as bagging training.
- In the case of bagging, the models are all independent. In the case of dropout, the models share parameters, with each model inheriting a different subset of parameters from the parent neural network.
- This **parameter sharing** makes it possible to represent an exponential number of models with a tractable amount of memory.
- In the case of bagging, each model is trained to convergence on its respective training set. In the case of dropout, typically most models are not explicitly trained at all—usually, the model is large enough that it would be infeasible to sample all possible sub- networks within the lifetime of the universe.
- Instead, a tiny fraction of the possible sub-networks are each trained for a single step, and the parameter sharing causes the remaining sub-networks to arrive at good settings of the parameters. These are the only differences.

# Inference

- Usually ensemble inference is average over all models
- All models in dropout are intractable
- It turns out that ensemble inference in dropout can be approximated by a *weight scaling inference rule:* the model with all units, but with the weights going out of unit *i* multiplied by the probability of including unit *i*
    - capture the right expected value of the output from that unit.
- Because we usually use an inclusion probability of 0.5, the weight scaling rule usually amounts to dividing the weights by 2 at the end of training, and then using the model as usual.
- Another way to achieve the same result is to multiply the states of the units by 2 during training.
- Either way, the goal is to make sure that the expected total input to a unit at test time is roughly the same as the expected total input to that unit at train time, even though half the units at train time are missing on average.
-

# MAP and regularisation

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} p_{\mathrm{model}}(\mathbb{X};\boldsymbol{\theta}) \tag{5.56}$$

$$= \arg\max_{\boldsymbol{\theta}} \prod_{i=1}^{m} p_{\mathrm{model}}(\boldsymbol{x}^{(i)};\boldsymbol{\theta}) \tag{5.57}$$

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}\sim\hat{p}_{\mathrm{data}}} \log p_{\mathrm{model}}(\boldsymbol{x};\boldsymbol{\theta}).$$

$$D_{\mathrm{KL}}\left(\hat{p}_{\mathrm{data}} \| p_{\mathrm{model}}\right) = \mathbb{E}_{\mathbf{x}\sim\hat{p}_{\mathrm{data}}}\left[\log \hat{p}_{\mathrm{data}}(\boldsymbol{x}) - \log p_{\mathrm{model}}(\boldsymbol{x})\right].$$

- p_data true generating function
- p_model be a parametric family of pdf indexed by theta
- \hat{p}_data empirical data (training) pdf
- If goal is to predict *y* given *X* then this is similar to classification problem

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} P(\boldsymbol{Y} \mid \boldsymbol{X};\boldsymbol{\theta}).$$

- MSE training of linear regression and MLE can be related if we assume that p_model is Normal where mean is the predicted value
- MAP estimate further uses prior distribution on theta

$$\boldsymbol{\theta}_{\mathrm{MAP}} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta} \mid \boldsymbol{x}) = \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{x} \mid \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}).$$

- If p(\theta) is Normal with zero mean and identity covariance, MAP estimate can be shown to be linear regression with L2 normalisation